
Dive into
Deep Learning
MATHEMATICS FOR DEEP LEARNING
Alexander J Smola · Rachel Hu · Brent Werness

Table of contents

Mathematics for Deep Learning 1

33 Linear Algebra 3
Resources and Further Reading . . . 4
33.1 Vectors, Angles, and Projections . . . 5
33.1.1 Vectors and Their Geometry . . . 5
33.1.2 Subspaces and Least Squares . . . 10
33.1.3 Similarity in High Dimensions . . . 15
33.1.4 Hyperplanes and Decision Boundaries . . . 19
33.1.5 Summary . . . 23
33.1.6 Exercises . . . 23
33.2 Matrices as Linear Maps . . . 27
33.2.1 Linear Maps . . . 27
33.2.2 Rank and Invertibility . . . 34
33.2.3 The Determinant . . . 39
33.2.4 Summary . . . 46
33.2.5 Exercises . . . 46
33.3 Eigendecompositions . . . 49
33.3.1 Eigenvalues and Diagonalization . . . 50
33.3.2 Symmetric Matrices and Quadratic Forms . . . 60
33.3.3 Summary . . . 65
33.3.4 Exercises . . . 65
33.4 Spectra in Practice . . . 67
33.4.1 Locating and Computing Eigenvalues . . . 67
33.4.2 Iteration and Stability . . . 73
33.4.3 Summary . . . 78
33.4.4 Exercises . . . 78
33.5 Singular Value Decomposition and Low-Rank Approximation . . . 81
33.5.1 The Singular Value Decomposition . . . 82
33.5.2 Low-Rank Approximation . . . 88
33.5.3 Solving Linear Systems with the SVD . . . 95
33.5.4 The SVD in Deep Learning . . . 99
33.5.5 Summary . . . 104
33.5.6 Exercises . . . 104
33.6 Further Reading . . . 109
33.6.1 Geometry and Linear Maps . . . 109
33.6.2 Eigenvalues and Spectra . . . 110

33.6.3 Low Rank Beyond the Truncated SVD . . . 111

34 Calculus and Automatic Differentiation 113
Resources and Further Reading . . . 114
34.1 Single Variable Calculus . . . 115
34.1.1 The Derivative . . . 116
34.1.2 Computing Derivatives . . . 118
34.1.3 The Mean Value Theorem . . . 121
34.1.4 Gradient Descent in One Variable . . . 123
34.1.5 Curvature and Taylor Series . . . 126
34.1.6 Nondifferentiable Points . . . 135
34.1.7 Summary . . . 139
34.1.8 Exercises . . . 139
34.2 Multivariable Calculus . . . 143
34.2.1 Partial Derivatives and the Gradient . . . 144
34.2.2 The Geometry of Gradients . . . 146
34.2.3 The Hessian . . . 150
34.2.4 Constrained Optimization and Lagrange Multipliers . . . 157
34.2.5 Summary . . . 159
34.2.6 Exercises . . . 159
34.3 The Chain Rule on Computational Graphs . . . 161
34.3.1 The Chain Rule as a Sum over Paths . . . 162
34.3.2 Backpropagation . . . 164
34.3.3 Summary . . . 168
34.3.4 Exercises . . . 168
34.4 Matrix Calculus . . . 169
34.4.1 The Jacobian . . . 170
34.4.2 The Chain Rule for Jacobians . . . 173
34.4.3 Derivatives with Respect to Vectors and Matrices . . . 176
34.4.4 Summary . . . 182
34.4.5 Exercises . . . 182
34.5 Automatic Differentiation . . . 185
34.5.1 Forward Mode and Dual Numbers . . . 185
34.5.2 Reverse Mode and the Tape . . . 189
34.5.3 Composing Jacobian Products . . . 196
34.5.4 Summary . . . 200
34.5.5 Exercises . . . 200
34.6 Integral Calculus . . . 204
34.6.1 The Definite Integral . . . 205
34.6.2 The Fundamental Theorem . . . 207
34.6.3 Multiple Integrals . . . 214
34.6.4 Summary . . . 221
34.6.5 Exercises . . . 221
34.7 Expectations and Their Gradients . . . 223
34.7.1 Expectations as Integrals . . . 224
34.7.2 Differentiation under the Integral Sign . . . 225
34.7.3 Score-Function and Pathwise Gradients . . . 227
34.7.4 Summary . . . 229
34.7.5 Exercises . . . 230
34.8 Further Reading . . . 231

34.8.1 Derivatives at Kinks . . . 231
34.8.2 Derivatives of Vector and Matrix Maps . . . 232
34.8.3 Automatic Differentiation . . . 232
34.8.4 Integration . . . 233

35 Optimization **235**
Notation . . . 236
Resources and Further Reading . . . 237
35.1 Gradient Descent and Curvature . . . 238
35.1.1 Descent Directions and Smoothness . . . 238
35.1.2 The Quadratic Model and the Condition Number . . . 244
35.1.3 Momentum and Acceleration . . . 249
35.1.4 Summary . . . 253
35.1.5 Exercises . . . 253
35.2 Curvature Methods: Newton, Quasi-Newton, and Trust Regions . . . 256
35.2.1 Newton's Method . . . 257
35.2.2 Quasi-Newton Methods . . . 260
35.2.3 Trust Regions . . . 262
35.2.4 Summary . . . 265
35.2.5 Exercises . . . 265
35.3 Convexity . . . 266
35.3.1 Convex Sets . . . 267
35.3.2 Convex Functions . . . 268
35.3.3 Consequences of Convexity . . . 276
35.3.4 Recognizing Convexity . . . 282
35.3.5 Beyond Convexity . . . 284
35.3.6 Summary . . . 288
35.3.7 Exercises . . . 288
35.4 Stochastic Gradient Methods . . . 292
35.4.1 Minibatch Gradients . . . 293
35.4.2 Step-Size Decay . . . 297
35.4.3 Rates for a Fixed Budget . . . 302
35.4.4 Variance Reduction . . . 308
35.4.5 Summary . . . 311
35.4.6 Exercises . . . 312
35.5 Adaptive Methods . . . 315
35.5.1 Per-Coordinate Step Sizes . . . 316
35.5.2 Coordinate Scaling and Its Limits . . . 320
35.5.3 Decoupled Weight Decay . . . 326
35.5.4 Warmup . . . 328
35.5.5 Structured Preconditioners . . . 329
35.5.6 Summary . . . 331
35.5.7 Exercises . . . 331
35.6 Constrained Optimization . . . 335
35.6.1 Equality Constraints and Lagrange Multipliers . . . 335
35.6.2 Inequality Constraints and the KKT Conditions . . . 338
35.6.3 Projections and Projected Gradient Descent . . . 341
35.6.4 Proximal Operators and Coordinate Descent . . . 347
35.6.5 Summary . . . 352
35.6.6 Exercises . . . 353

35.7 Duality . . . 356
35.7.1 The Dual Problem . . . 357
35.7.2 Strong Duality . . . 360
35.7.3 Multipliers as Prices . . . 363
35.7.4 The Convex Conjugate . . . 365
35.7.5 Worked Examples . . . 369
35.7.6 Summary . . . 374
35.7.7 Exercises . . . 374
35.8 Numerical Stability and Conditioning . . . 378
35.8.1 Floating-Point Arithmetic . . . 379
35.8.2 Stable Softmax and Cross-Entropy . . . 383
35.8.3 Catastrophic Cancellation . . . 387
35.8.4 Conditioning . . . 393
35.8.5 Summary . . . 401
35.8.6 Exercises . . . 401
35.9 Further Reading . . . 405
35.9.1 Gradient Methods . . . 405
35.9.2 Stochastic and Adaptive Methods . . . 406
35.9.3 Constraints and Duality . . . 407
35.9.4 Floating Point . . . 408

36 Probability and Statistical Learning **409**
Resources and Further Reading . . . 410
36.1 Random Variables . . . 411
36.1.1 Densities . . . 411
36.1.2 Mean and Variance . . . 418
36.1.3 Tails . . . 420
36.1.4 Summary . . . 424
36.1.5 Exercises . . . 424
36.2 Joint Distributions and Transformations . . . 427
36.2.1 Joint and Conditional Densities . . . 427
36.2.2 Covariance and Correlation . . . 434
36.2.3 Transformations of Random Variables . . . 438
36.2.4 Summary . . . 442
36.2.5 Exercises . . . 442
36.3 Distributions . . . 444
36.3.1 Discrete Distributions . . . 445
36.3.2 Continuous Distributions . . . 452
36.3.3 The Multivariate Gaussian . . . 460
36.3.4 Summary . . . 464
36.3.5 Exercises . . . 465
36.4 The Exponential Family . . . 469
36.4.1 One Form for Many Distributions . . . 469
36.4.2 Maximum Entropy . . . 472
36.4.3 The Log-Partition Function . . . 475
36.4.4 Origins of the Softmax . . . 479
36.4.5 Conjugate Priors . . . 482
36.4.6 Summary . . . 486
36.4.7 Exercises . . . 486
36.5 Statistics . . . 489

36.5.1 Estimators and Their Sampling Distribution . . . 490
36.5.2 Hypothesis Testing . . . 500
36.5.3 Confidence Intervals . . . 509
36.5.4 The Bootstrap . . . 512
36.5.5 Summary . . . 515
36.5.6 Exercises . . . 515
36.6 Maximum Likelihood . . . 521
36.6.1 The Maximum Likelihood Principle . . . 521
36.6.2 Losses as Negative Log-Likelihoods . . . 524
36.6.3 Accuracy of the Maximum Likelihood Estimate . . . 528
36.6.4 MAP Estimation and Regularization . . . 537
36.6.5 Summary . . . 540
36.6.6 Exercises . . . 540
36.7 Latent Variables, EM, and the ELBO . . . 545
36.7.1 Mixture Models . . . 545
36.7.2 The Evidence Lower Bound . . . 546
36.7.3 Expectation–Maximization . . . 547
36.7.4 Summary . . . 551
36.7.5 Exercises . . . 551
36.8 Bayesian Computation . . . 552
36.8.1 The Posterior of Bayesian Logistic Regression . . . 553
36.8.2 Rejection and Importance Sampling . . . 560
36.8.3 Markov Chain Monte Carlo . . . 564
36.8.4 Variational Inference . . . 570
36.8.5 Comparison of the Methods . . . 574
36.8.6 Summary . . . 576
36.8.7 Exercises . . . 576
36.9 Concentration Inequalities . . . 581
36.9.1 Exponential Tail Bounds . . . 581
36.9.2 Probability in High Dimension . . . 591
36.9.3 Summary . . . 595
36.9.4 Exercises . . . 595
36.10 Uniform Convergence and Generalization . . . 598
36.10.1 Uniform Convergence . . . 599
36.10.2 Rademacher Complexity . . . 602
36.10.3 Interpolation and Double Descent . . . 608
36.10.4 Summary . . . 613
36.10.5 Exercises . . . 614
36.11 Naive Bayes . . . 618
36.11.1 Bayes' Rule for Classification . . . 618
36.11.2 Learning by Counting . . . 621
36.11.3 Fashion-MNIST by Counting . . . 623
36.11.4 Evaluating the Classifier . . . 628
36.11.5 Summary . . . 633
36.11.6 Exercises . . . 634
36.12 Further Reading . . . 638
36.12.1 Probability Models . . . 638
36.12.2 Estimation and Testing . . . 639
36.12.3 Computing with Posteriors . . . 640
36.12.4 Concentration and Learning Theory . . . 640

36.12.5 Classifiers from Counts . . . 641

37 Information Theory and Divergences 643
Resources and Further Reading . . . 644
37.1 Entropy, Cross-Entropy, and KL Divergence . . . 645
37.1.1 Information and Entropy . . . 647
37.1.2 KL Divergence and Cross-Entropy . . . 652
37.1.3 The Classification Loss . . . 656
37.1.4 Entropy and KL Divergence for Densities . . . 661
37.1.5 Summary . . . 665
37.1.6 Exercises . . . 666
37.2 Codes, Compression, and Perplexity . . . 669
37.2.1 Symbol Codes . . . 669
37.2.2 Stream Codes and Description Length . . . 674
37.2.3 Entropy Rate and Typical Sequences . . . 679
37.2.4 Perplexity . . . 682
37.2.5 Summary . . . 684
37.2.6 Exercises . . . 684
37.3 f-Divergences . . . 688
37.3.1 The f-Divergence Family . . . 689
37.3.2 Forward and Reverse KL . . . 696
37.3.3 Total Variation, Pinsker's Inequality, and Data Processing . . . 699
37.3.4 The Fenchel Conjugate and the Critic Bound . . . 704
37.3.5 Summary . . . 708
37.3.6 Exercises . . . 708
37.4 Test Functions and Transport . . . 711
37.4.1 Integral Probability Metrics and MMD . . . 712
37.4.2 Optimal Transport and the Wasserstein Distance . . . 717
37.4.3 Fixed Kernels and Trained Critics . . . 723
37.4.4 Summary . . . 723
37.4.5 Exercises . . . 724
37.5 Score-Based Comparisons . . . 727
37.5.1 The Score and the Fisher Divergence . . . 728
37.5.2 Stein's Identity . . . 731
37.5.3 The Kernel Stein Discrepancy . . . 733
37.5.4 Training Objectives and Their Divergences . . . 736
37.5.5 Summary . . . 739
37.5.6 Exercises . . . 739
37.6 Mutual Information . . . 742
37.6.1 Measures of Dependence . . . 743
37.6.2 Processing and Prediction . . . 749
37.6.3 Communication and Compression . . . 755
37.6.4 The Information Bottleneck . . . 759
37.6.5 Summary . . . 763
37.6.6 Exercises . . . 763
37.7 Estimating Mutual Information . . . 767
37.7.1 A Pair with Known Information . . . 767
37.7.2 Nonparametric Estimation . . . 768
37.7.3 Variational Lower Bounds . . . 775
37.7.4 InfoNCE . . . 778

37.7.5 Summary . . . 786
37.7.6 Exercises . . . 786
37.8 Further Reading . . . 790
37.8.1 Entropy and Losses . . . 791
37.8.2 Codes and Compression . . . 791
37.8.3 Comparing Distributions . . . 793
37.8.4 Mutual Information . . . 794
37.8.5 Estimating Mutual Information . . . 795

38 Dynamics: Differential Equations and Generative Flows 797
Resources and Further Reading . . . 798
38.1 Ordinary Differential Equations and Numerical Solvers . . . 799
38.1.1 Vector Fields and Flows . . . 801
38.1.2 Linear ODEs and Stability . . . 806
38.1.3 Numerical Solvers . . . 812
38.1.4 Summary . . . 821
38.1.5 Exercises . . . 821
38.2 Neural ODEs, Adjoints, and Continuous Flows . . . 825
38.2.1 Neural ODEs . . . 826
38.2.2 The Adjoint Method . . . 831
38.2.3 Continuous Normalizing Flows . . . 835
38.2.4 Summary . . . 840
38.2.5 Exercises . . . 841
38.3 Stochastic Differential Equations . . . 844
38.3.1 Brownian Motion . . . 845
38.3.2 Itô Calculus . . . 849
38.3.3 The Ornstein–Uhlenbeck Process . . . 859
38.3.4 Simulation . . . 864
38.3.5 Summary . . . 868
38.3.6 Exercises . . . 869
38.4 The Fokker–Planck Equation and Probability Flow . . . 873
38.4.1 From Paths to Densities . . . 874
38.4.2 The Fokker–Planck Equation . . . 876
38.4.3 The Score Function . . . 882
38.4.4 The Continuity Equation and the Probability-Flow ODE . . . 885
38.4.5 Time Reversal . . . 891
38.4.6 Summary . . . 897
38.4.7 Exercises . . . 897
38.5 Score Matching and Diffusion Models . . . 902
38.5.1 Learning the Score . . . 902
38.5.2 Sampling with a Score . . . 909
38.5.3 Scores Across Noise Levels . . . 912
38.5.4 Deterministic Sampling and Guidance . . . 916
38.5.5 Summary . . . 921
38.5.6 Exercises . . . 921
38.6 Flow Matching and Optimal Transport . . . 925
38.6.1 Flow Matching . . . 927
38.6.2 Rectified Flow . . . 932
38.6.3 Optimal Transport and Straightness . . . 940
38.6.4 Sampling with a Learned Field . . . 943

38.6.5 Summary 945
38.6.6 Exercises 946
38.7 Further Reading 949
38.7.1 Solvers 949
38.7.2 Continuous-Time Models 950
38.7.3 Noise 950
38.7.4 Learned Generators 951

References **953**

Mathematics for Deep Learning

This appendix gathers the mathematical results the main text relies on, developed far enough to be useful and no further. It is organized as six chapters, each intuition-first and paired with runnable examples. *Linear algebra* covers vectors, matrices, norms, and the decompositions behind every layer. *Calculus* develops derivatives, the chain rule, and the multivariable and matrix calculus that backpropagation automates. *Optimization* treats convexity, gradient methods, and the convergence and conditioning results the training loop depends on.

The remaining chapters supply the probabilistic and dynamical foundations: *probability and statistical learning* (distributions, estimation, and generalization), *information theory* (entropy and the KL divergence that most losses descend from, codes, other ways to compare two distributions, and mutual information), and *dynamics* (differential equations and the continuous-time view that connects to modern normalizing-flow and diffusion models). Read it start to finish, or dip in when a main-text chapter cites a result.

Chapter 33

Linear Algebra

Every network in this book is built from matrix products. A linear layer multiplies its input by a weight matrix, attention multiplies queries by keys, and a recurrent network multiplies its state by the same matrix at every step. Section 1.3 taught how to compute these products and rarely asked what a matrix does to the vectors it acts on. This chapter does, with proofs. It is additional reading, and the main text does not depend on it.

Take a Fashion-MNIST image, a list of 784 numbers, and a layer that maps it to 256 features. How similar are two images? How much information does the layer lose? What happens to a signal that passes through the same map fifty times? We cannot draw 784 dimensions, so the answers must come from algebra that keeps the geometry of the plane.

Around 1810 Gauss fitted the orbit of the asteroid Pallas to more observations than unknowns by least squares, a problem the first section solves and the last takes up again. The central idea of linear algebra is that a matrix becomes simple in the right directions. In orthonormal axes, lengths and angles are dot products. Along its eigenvectors, a matrix acts as multiplication by a number. With a pair of singular bases, every matrix becomes a list of independent stretches. These ideas explain why attention divides its scores by $\sqrt{d}$ (Section 10.2), when the gradients of a recurrent network explode (Section 8.6), and what a low-rank fine-tuning adapter can represent.

The first two sections apply a matrix once. Section 33.1 carries lengths, angles, and projections into any dimension, and Section 33.2 finds which inputs a matrix sends to zero and when it can be undone. The rest of the chapter chooses coordinates to suit the matrix: eigenvectors for a square matrix applied repeatedly (Section 33.3), then what eigenvalues leave unsaid (Section 33.4), and singular vectors for every matrix (Section 33.5). Three results matter most: projection onto a subspace gives the closest point, a symmetric matrix has an orthonormal basis of eigenvectors (the spectral theorem), and a truncated singular value decomposition is the best approximation of its rank. Two matrices recur: the running matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$, with complex eigenvalues, and the shear $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$, with too few eigenvectors.

A first reading can skim the determinant in n dimensions and, in Section 33.4, Gershgorin discs, PageRank, and pseudospectra. We work in real finite-dimensional spaces and use complex numbers only where eigenvalues require them. How libraries compute eigenvalues and solve large systems is left to Section 33.6.

Resources and Further Reading

The books, courses, and notes below teach this chapter's material again, at greater length or from another side; each names its section, and most are free online. What lies beyond the chapter is the subject of Section 33.6.

Books

- Mathematics for Machine Learning by Deisenroth, Faisal, and Ong: a free book whose chapters 2 (Linear Algebra), 3 (Analytic Geometry), and 4 (Matrix Decompositions) cover the whole chapter from a machine-learning perspective, from the projections of Section 33.1 to the SVD of Section 33.5.
- Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares by Boyd and Vandenberghe: a free book with slides and code, organized around least squares and data, for the fitting problems of Section 33.1 and the pseudoinverse of Section 33.5.
- Introduction to Linear Algebra, 6th ed. by Gilbert Strang: the standard first course (a commercial book with sample chapters and problems online), built on the column picture and the four fundamental subspaces of Section 33.2 and Section 33.5.
- Linear Algebra and Learning from Data by Gilbert Strang: the SVD-centered companion to MIT 18.065, close to the low-rank approximation, principal components, and least squares of Section 33.5.
- Linear Algebra Done Right, 4th ed. by Sheldon Axler: a free, rigorous, determinant-free route to eigenvalues and the spectral theorem, for readers who want every proof behind Section 33.3.
- Numerical Linear Algebra by Trefethen and Bau: conditioning, stability, QR, and the floating-point algorithms behind `solve`, `eig`, and `svd`, the numerical side of Section 33.2 and Section 33.5.3.2.

Courses and video lectures

- MIT 18.06 Linear Algebra by Gilbert Strang: a full lecture course with notes, exams, and solutions, covering Section 33.2 and Section 33.3.
- MIT 18.065 Matrix Methods in Data Analysis, Signal Processing, and Machine Learning by Gilbert Strang: the follow-on course that connects the SVD and low-rank approximation of Section 33.5 to deep learning and optimization.
- Essence of Linear Algebra by 3Blue1Brown: short animated lectures on vectors, linear maps, determinants, and eigenvectors, the pictures of Section 33.1 through Section 33.3.
- Singular Value Decomposition by Steve Brunton: lectures on the geometry of the SVD, low-rank approximation, principal components, and randomized SVD, the material of Section 33.5.

Tutorials and notes

- Linear Algebra Review and Reference by Kolter, updated by Do, for Stanford CS229: a compact refresher on the notation of Section 33.1 and Section 33.2 as machine-learning courses use it.
- Immersive Linear Algebra by Ström, Åström, and Akenine-Möller: a free online book with interactive figures of linear maps, determinants, and eigenvectors, for Section 33.2 and Section 33.3.

33.1 Vectors, Angles, and Projections

The Basics treated every input as a vector: an image became a list of pixel values, and a word became an embedding. Models do three geometric things with these vectors. They compare, as when we ask whether two embeddings are similar. They fit, as when we look for the combination of features that comes closest to the targets. And they separate, as when a classifier decides on which side of a boundary an input falls. In the plane each of these is a picture of lengths, angles, and shadows. Inputs live in spaces of hundreds or thousands of dimensions, which nobody can draw. Do the pictures still mean anything there?

Telling t-shirts from trousers is one instance of the question. Each 28×28 Fashion-MNIST image (Section 3.2) is a list of 784 numbers, so it is a point in $\mathbb{R}^{784}$. A simple rule assigns a new image to the class whose average image it lies closer to. In the plane "closer" is something we can see; in $\mathbb{R}^{784}$ it is a calculation, and it takes one dot product per image. On the $2{,}000$ test images of those two classes the rule is right about 92% of the time. That so simple a rule works says that the two clouds of points lie well apart, and that "closer" can be computed where it cannot be seen.

The dot product carries the geometry of the plane into any dimension. Lengths are square roots of dot products, the shadow of one vector on another is a ratio of dot products, and the angle between two vectors is defined from them. The section has three parts. The first treats a pair of vectors: projection onto a line, the angle, and the Cauchy–Schwarz inequality, which Section 1.3 proved algebraically and we prove again from a picture. The second supplies the vocabulary of subspaces that the rest of the chapter uses (span, independence, basis, dimension) and solves least squares by projection, with the QR factorization as its tool. The third takes these results to the images, where no picture is possible: it asks how similar unrelated vectors are in high dimensions, and it reads a linear classifier as a hyperplane.

Of these results, three matter most: the residual of a projection is orthogonal to the line or subspace we project onto, which makes the projection the closest point; Cauchy–Schwarz, which gives the angle a meaning in any dimension; and the size $1/\sqrt{d}$ of the cosine between random directions in $\mathbb{R}^d$. Apart from loading the images, all the code is plain NumPy, the same in every framework.

33.1.1 Vectors and Their Geometry

33.1.1.1 Points and Directions

A vector admits two geometric readings. Read as a *point*, $\mathbf{v} = (3, 2)$ is the location 3 units to the right of the *origin* and 2 units up. This reading turns a dataset into a cloud of points, and the task of telling t-shirts from trousers into geometry: separate two clouds. Read as a *direction*, the same vector says "take 3 steps to the right and 2 steps up." An arrow with that displacement can start anywhere, so all the parallel arrows in Figure 33.1 are the same vector.

The direction view makes vector addition visual: follow one arrow, then the other, tip to tail, as in Figure 33.2.

Subtraction has a similar reading. From the identity $\mathbf{u} = \mathbf{v} + (\mathbf{u} - \mathbf{v})$, the vector $\mathbf{u} - \mathbf{v}$ is the direction that takes us from the point $\mathbf{v}$ to the point $\mathbf{u}$.

Everything else in this section is built from one operation, the *dot product* of Section 1.3. For two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$ it multiplies them entry by entry and adds the products:

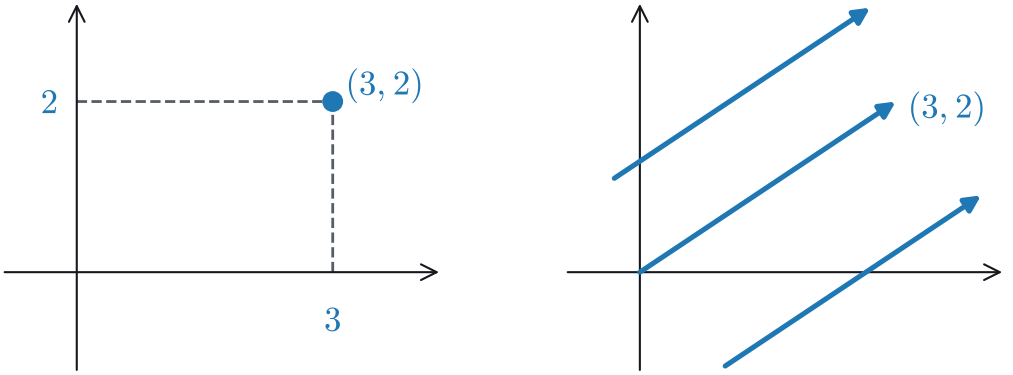


Figure 33.1: A vector can be read two ways: as a *point* whose first component is the x-coordinate and second is the y-coordinate (left), or as a *direction*, an arrow that can start anywhere, so every arrow shown is the same vector $(3, 2)$ (right).

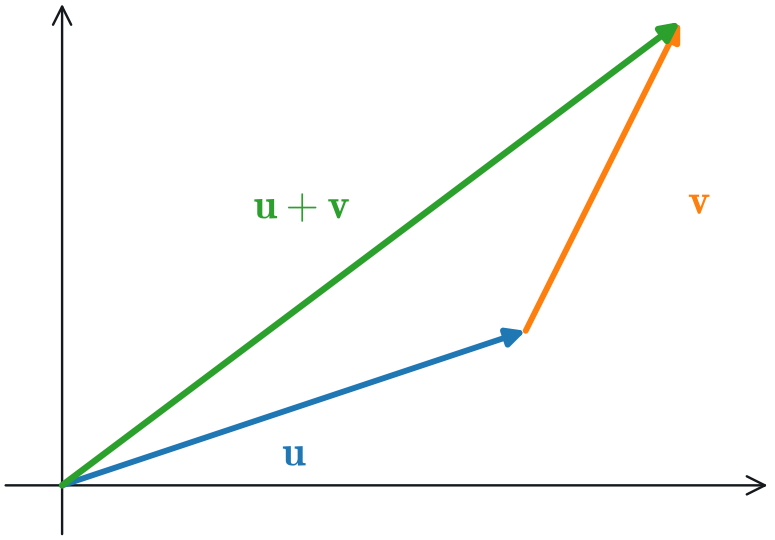


Figure 33.2: We can visualize vector addition by first following one vector, and then another, placing them tip to tail.

$$\mathbf{u}^\top \mathbf{v} = \sum_{i=1}^{d} u_i v_i. \tag{33.1}$$

It is symmetric, $\mathbf{u}^\top \mathbf{v} = \mathbf{v}^\top \mathbf{u}$, so we also write it with a dot, $\mathbf{u} \cdot \mathbf{v}$: the dot when we think of two arrows, the transpose when the product is part of a matrix expression. For $\mathbf{u} = (1, 3)$ and $\mathbf{v} = (2, 1)$ it is $2 + 3 = 5$, and the squared length of $\mathbf{u}$ is $\mathbf{u} \cdot \mathbf{u} = 1 + 9 = 10$.

33.1.1.2 Projection onto a Line

How much of a vector $\mathbf{v}$ points along another vector $\mathbf{w}$? Put differently: which multiple $t\mathbf{w}$ comes closest to $\mathbf{v}$? This is the smallest fitting problem there is. We approximate $\mathbf{v}$ by one direction and choose the best coefficient.

Take $\mathbf{v} = (1, 3)$ and $\mathbf{w} = (2, 1)$ and try a few values of t. With $t = 1$ the error $\mathbf{v} - t\mathbf{w} = (-1, 2)$ has squared length 5; with $t = 0.5$ the error is $(0, 2.5)$, of squared length 6.25; with $t = 1.5$ it is $(-2, 1.5)$, again of squared length 6.25. The best value appears to be $t = 1$, and the closest point is $(2, 1)$. Its error is called the *residual*. Here $\mathbf{r} = (-1, 2)$, and its dot product with $\mathbf{w}$ vanishes: $\mathbf{r} \cdot \mathbf{w} = -2 + 2 = 0$. The shortest error meets the line at a right angle. The lengths fit together as in a right triangle: $\|\mathbf{v}\|^2 = 10$, the closest point has squared length 5, the residual has squared length 5, and $10 = 5 + 5$.

The general statement keeps this shape.

Proposition (orthogonal projection). *Let* $\mathbf{w} \neq \mathbf{0}$. *The multiple of* $\mathbf{w}$ *closest to* $\mathbf{v}$ *is*

$$\operatorname{proj}_{\mathbf{w}} \mathbf{v} = \frac{\mathbf{v} \cdot \mathbf{w}}{\mathbf{w} \cdot \mathbf{w}} \mathbf{w}, \tag{33.2}$$

and the residual $\mathbf{r} = \mathbf{v} - \operatorname{proj}_{\mathbf{w}} \mathbf{v}$ *satisfies* $\mathbf{r} \cdot \mathbf{w} = 0$.

Proof. We minimize the squared error as a function of the coefficient t. Expanding the dot product of $\mathbf{v} - t\mathbf{w}$ with itself gives a quadratic in t,

$$q(t) = \|\mathbf{v} - t\mathbf{w}\|^2 = \|\mathbf{w}\|^2 t^2 - 2\,(\mathbf{v} \cdot \mathbf{w})\, t + \|\mathbf{v}\|^2,$$

whose leading coefficient $\|\mathbf{w}\|^2$ is positive, so it has a single minimum where $q'(t) = 2\|\mathbf{w}\|^2 t - 2\,\mathbf{v} \cdot \mathbf{w}$ vanishes, at $t^\star = \mathbf{v} \cdot \mathbf{w} / \mathbf{w} \cdot \mathbf{w}$. This is Equation 33.2. For the residual, $\mathbf{r} \cdot \mathbf{w} = \mathbf{v} \cdot \mathbf{w} - t^\star\, \mathbf{w} \cdot \mathbf{w} = 0$. ∎

On the example, $q(t) = 5t^2 - 10t + 10$, minimized at $t^\star = 5/5 = 1$ with value 5, which matches the trial values above. We call $\operatorname{proj}_{\mathbf{w}} \mathbf{v}$ the *projection*, or *shadow*, of $\mathbf{v}$ on $\mathbf{w}$. The shadow is the part of $\mathbf{v}$ along $\mathbf{w}$, and the residual is the part at right angles to $\mathbf{w}$. The zero vector has no direction, and for $\mathbf{w} = \mathbf{0}$ the formula would divide by zero.

The right angle is what makes lengths add. Two vectors $\mathbf{a}$, $\mathbf{b}$ are *orthogonal* when $\mathbf{a} \cdot \mathbf{b} = 0$. For orthogonal vectors, expanding the squared length of the sum gives *Pythagoras' theorem*,

$$\|\mathbf{a} + \mathbf{b}\|^2 = \|\mathbf{a}\|^2 + 2\,\mathbf{a} \cdot \mathbf{b} + \|\mathbf{b}\|^2 = \|\mathbf{a}\|^2 + \|\mathbf{b}\|^2. \tag{33.3}$$

Orthogonal pieces add in squared length, with no cross term, in every dimension. Applied to $\mathbf{v} = \operatorname{proj}_{\mathbf{w}} \mathbf{v} + \mathbf{r}$, it reads $\|\mathbf{v}\|^2 = \|\operatorname{proj}_{\mathbf{w}} \mathbf{v}\|^2 + \|\mathbf{r}\|^2$, which is $10 = 5 + 5$ on the example.

The length of the shadow, with a sign that says whether it points along $\mathbf{w}$ or against it, is the coefficient times $\|\mathbf{w}\|$:

$$\frac{\mathbf{v} \cdot \mathbf{w}}{\|\mathbf{w}\|}. \tag{33.4}$$

On the example it is $5/\sqrt{5} = \sqrt{5}$, the length of $(2, 1)$. This *signed length* returns at the end of the section as the signed distance of a point to a hyperplane. Figure 33.3 draws the example.

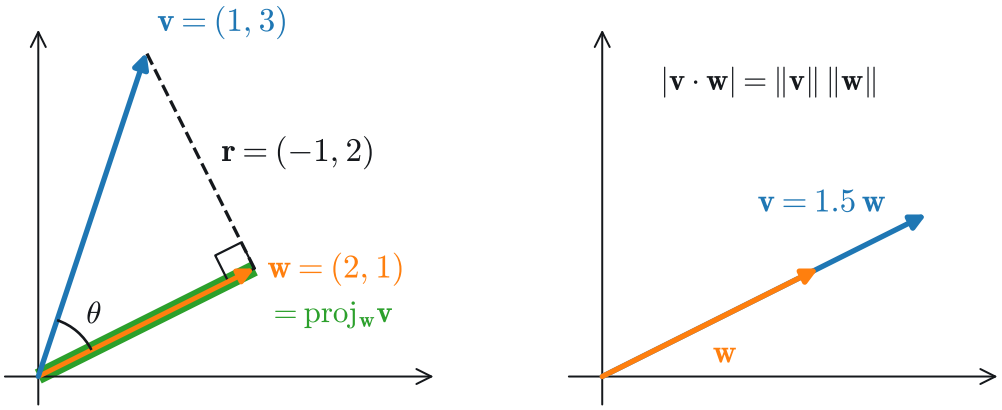


Figure 33.3: (a) The shadow of $\mathbf{v} = (1, 3)$ on $\mathbf{w} = (2, 1)$ is $(2, 1)$; the residual $(-1, 2)$ meets $\mathbf{w}$ at a right angle, so $\|\mathbf{v}\|^2 = 10 = 5 + 5$. (b) The Cauchy–Schwarz equality case: $\mathbf{v}$ is collinear with $\mathbf{w}$ and the residual is zero.

33.1.1.3 Cauchy–Schwarz and the Angle

In 784 dimensions there is no protractor. Can "the angle between two images" mean anything? The answer rests on a question that sounds simpler: is the shadow of a vector ever longer than the vector itself? Panel (a) of Figure 33.3 says no, since the shadow is a leg of a right triangle whose hypotenuse is $\mathbf{v}$. Pythagoras turns that picture into a proof valid in every dimension.

Proposition (Cauchy–Schwarz). *For all vectors* $\mathbf{v}, \mathbf{w}$*,*

$$|\mathbf{v} \cdot \mathbf{w}| \le \|\mathbf{v}\| \, \|\mathbf{w}\|, \tag{33.5}$$

with equality if and only if $\mathbf{v}$ *and* $\mathbf{w}$ *are collinear (one is a multiple of the other).*

Proof. If $\mathbf{w} = \mathbf{0}$, both sides are zero and the vectors are collinear. Otherwise split $\mathbf{v}$ into its shadow and residual. By Pythagoras Equation 33.3,

$$\|\mathbf{v}\|^2 = \|\operatorname{proj}_{\mathbf{w}} \mathbf{v}\|^2 + \|\mathbf{r}\|^2 \ge \|\operatorname{proj}_{\mathbf{w}} \mathbf{v}\|^2 = \frac{(\mathbf{v} \cdot \mathbf{w})^2}{\|\mathbf{w}\|^2}.$$

Multiplying by $\|\mathbf{w}\|^2$ and taking square roots gives the inequality. Equality holds exactly when $\mathbf{r} = \mathbf{0}$, that is, when $\mathbf{v} = t^\star\mathbf{w}$. Conversely, if $\mathbf{v} = t\mathbf{w}$, then $|\mathbf{v}\cdot\mathbf{w}| = |t|\,\|\mathbf{w}\|^2 = \|\mathbf{v}\|\,\|\mathbf{w}\|$. ■

The inequality says that the shadow is never longer than the vector, and that it is as long only when nothing is left over. The proof in Section 1.3 was algebraic. This one is panel (a) of the figure, and panel (b) is the equality case.

We turn to the angle. In the plane, the law of cosines gives $\mathbf{v}\cdot\mathbf{w} = \|\mathbf{v}\|\|\mathbf{w}\|\cos\theta$ (Section 1.3). In $\mathbb{R}^d$ there is no triangle to measure, so the identity becomes the definition. For nonzero $\mathbf{v}, \mathbf{w} \in \mathbb{R}^d$, the *angle* between them is

$$\theta = \arccos\left(\frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\|\mathbf{w}\|}\right) \in [0, \pi]. \tag{33.6}$$

The arccos accepts only arguments in $[-1, 1]$, and Cauchy–Schwarz states that the ratio lies there. So the definition is legitimate in every dimension, and the planar identity now holds everywhere, as a consequence of the definition:

$$\mathbf{v}\cdot\mathbf{w} = \|\mathbf{v}\|\|\mathbf{w}\|\cos\theta. \tag{33.7}$$

The equality cases of Cauchy–Schwarz are $\theta = 0$ (same direction) and $\theta = \pi$ (opposite directions).

An objection: is this defined angle the one we would measure? Two vectors always lie in a plane (a line, if they are collinear), and inside that plane school geometry applies. The projection gives coordinates for that plane. Put $\mathbf{q}_1 = \mathbf{w}/\|\mathbf{w}\|$ and, when $\mathbf{r} \neq \mathbf{0}$, $\mathbf{q}_2 = \mathbf{r}/\|\mathbf{r}\|$. These two vectors have unit length and are orthogonal, so dot products of their combinations are computed as in the plane. In these coordinates $\mathbf{w} = (\|\mathbf{w}\|, 0)$ and $\mathbf{v} = (\mathbf{v}\cdot\mathbf{w}/\|\mathbf{w}\|,\ \|\mathbf{r}\|)$. The angle a protractor would measure between these two planar arrows has cosine $(\mathbf{v}\cdot\mathbf{w}/\|\mathbf{w}\|)/\|\mathbf{v}\|$, which is the defined value. So the definition agrees with ordinary geometry in the plane spanned by the two vectors. On the example, $\cos\theta = 5/(\sqrt{10}\sqrt{5}) = 1/\sqrt{2}$ and $\theta = 45°$.

For nonzero vectors, orthogonality means $\theta = \pi/2$. The definition by the dot product also covers the zero vector, which is orthogonal to every vector, although no angle is defined for it.

In code the definition takes two lines. We try it on the worked pair and on a pair chosen to make trouble: $\mathbf{v} = (10^8 + 1, 3, 7)$ and $3\mathbf{v}$, which are collinear, so their angle is zero and the ratio should be 1. For this pair the cell also prints the computed ratio minus 1 and the `arccos` of the ratio without the clip.

```
def angle(v, w):
    cos = v @ w / (np.linalg.norm(v) * np.linalg.norm(w))
    return np.arccos(np.clip(cos, -1, 1))  # roundoff can push |cos| past 1

v = np.array([1e8 + 1, 3, 7])
cos = v @ (3 * v) / (np.linalg.norm(v) * np.linalg.norm(3 * v))
with np.errstate(invalid='ignore'):        # silence the warning for nan
    print(angle(np.array([1., 3]), np.array([2., 1])), cos - 1,
```

```
            np.arccos(cos), angle(v, 3 * v))
```

```
0.7853981633974484 2.220446049250313e-16 nan 0.0
```

The first value is $0.785 = \pi/4$, the angle of the worked pair. The other three are the reason for the clip in `angle`, which is the part of the cell worth remembering. In exact arithmetic Cauchy–Schwarz keeps the ratio in $[-1, 1]$. In floating point the ratio can overshoot by one unit in the last place: for the collinear pair it comes out as $1 + 2.2 \times 10^{-16}$, and `arccos` of that is `nan`. With the clip, `angle` returns 0.

Cauchy–Schwarz also bounds the length of a sum by the sum of the lengths. The *triangle inequality* $\|\mathbf{v}+\mathbf{w}\| \le \|\mathbf{v}\| + \|\mathbf{w}\|$ follows by expanding $\|\mathbf{v}+\mathbf{w}\|^2$ and bounding the cross term $2\,\mathbf{v}\cdot\mathbf{w}$ by $2\,\|\mathbf{v}\|\|\mathbf{w}\|$ with Equation 33.5, as in Section 1.3.

33.1.2 Subspaces and Least Squares

33.1.2.1 Span, Bases, and Subspaces

Suppose we measure a quantity three times, at times $t = 0, 1, 2$, and want the straight line $C + Dt$ that fits the measurements best. Collect the measurements in $\mathbf{y} \in \mathbb{R}^3$, the times in $\mathbf{t} = (0, 1, 2)$, and let $\mathbf{1} = (1, 1, 1)$. The line's values at the three times form the vector $C\mathbf{1} + D\mathbf{t}$, so choosing the line means choosing the point of the form $C\mathbf{1} + D\mathbf{t}$ that is closest to $\mathbf{y}$. As C and D vary, these candidates fill a plane through the origin in $\mathbb{R}^3$. Projection onto a line handled one direction; we now need words for planes and their higher-dimensional relatives. A reader who knows the standard definitions can go straight to the next subsection, which finishes the line fit.

A weighted sum $a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ is a *linear combination* of $\mathbf{v}_1, \dots, \mathbf{v}_k$. The set of all of them is their *span*,

$$\operatorname{span}(\mathbf{v}_1, \dots, \mathbf{v}_k) = \{a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k : a_1, \dots, a_k \in \mathbb{R}\}.$$

In $\mathbb{R}^2$ (Figure 33.4), the span of one nonzero vector is the line through the origin in its direction. The span of two vectors not on a common line through the origin is the whole plane. The candidates of the line fit are $\operatorname{span}(\mathbf{1}, \mathbf{t})$.

A span is closed under addition and scaling: adding two linear combinations or scaling one gives another linear combination. Any nonempty set of vectors with these two closure properties is called a *subspace*. Every subspace contains $\mathbf{0}$ (scale any element by zero), so a line that misses the origin is not a subspace. The subspaces of $\mathbb{R}^2$ are the origin alone, the lines through the origin, and $\mathbb{R}^2$ itself; in $\mathbb{R}^3$ the planes through the origin join the list.

A spanning list can be redundant. If one vector already lies in the span of the others, deleting it shrinks the list without shrinking the span. A list is *linearly independent* when this cannot happen, that is, when the only linear combination $a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ equal to $\mathbf{0}$ is the trivial one with every $a_i = 0$. The vectors $\mathbf{1}$ and $\mathbf{t}$ are independent: $a_1\mathbf{1} + a_2\mathbf{t} = (a_1,\ a_1 + a_2,\ a_1 + 2a_2)$ vanishes only if $a_1 = a_2 = 0$.

A *basis* of a subspace is a linearly independent list that spans it. The vectors $\mathbf{e}_1 = (1, 0)$ and $\mathbf{e}_2 = (0, 1)$ form the *standard basis* of $\mathbb{R}^2$, but the slanted pair in Figure 33.4 is an equally valid basis. Every basis of a given subspace has the same number of elements (a theorem we use without proof), and that number is the subspace's *dimension*. This gives a precise

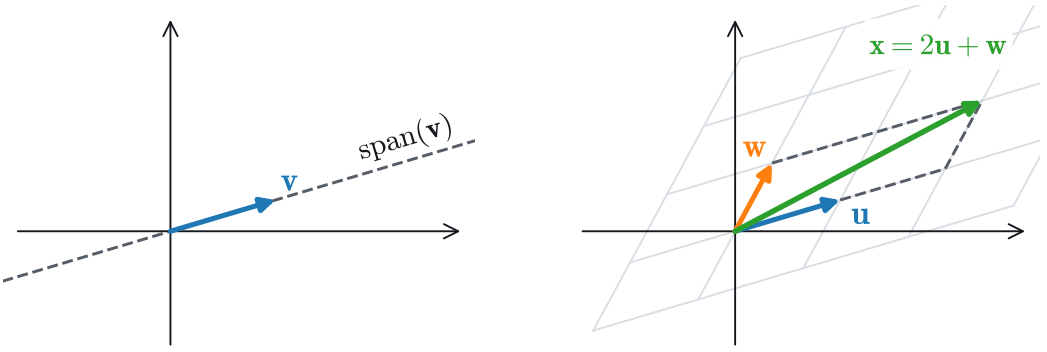


Figure 33.4: Left: the span of a single nonzero vector $\mathbf{v}$, the set of all of its scalar multiples, is the line through the origin in its direction, a one-dimensional subspace. Right: two vectors not on a common line through the origin span the whole plane; the dashed parallelogram construction resolves $\mathbf{x}$ as $2\mathbf{u}+\mathbf{w}$, coordinates that are unique because $\mathbf{u}$ and $\mathbf{w}$ are linearly independent.

meaning to the d in "d-dimensional space": $\mathbb{R}^d$ has dimension d because $\mathbf{e}_1, \dots, \mathbf{e}_d$ is a basis. The candidates of the line fit form a two-dimensional subspace of $\mathbb{R}^3$.

Independence is what makes coordinates well defined. If $\mathbf{v}_1, \dots, \mathbf{v}_k$ is a basis of a subspace S, every $\mathbf{x} \in S$ can be written as $\mathbf{x} = a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$ for exactly one choice of coefficients. One representation exists because the basis spans S. Two of them, with coefficients a_i and b_i, would differ by $\sum_i (a_i - b_i)\,\mathbf{v}_i = \mathbf{0}$, and independence forces $a_i = b_i$ for every i.

A basis therefore identifies a k-dimensional subspace with $\mathbb{R}^k$: once the basis is fixed, the coefficient list $(a_1, \dots, a_k)$ names the point. For the line fit, the coefficients of a candidate are the intercept C and the slope D.

33.1.2.2 Projection onto a Subspace

We can now finish the line fit, for measurements $\mathbf{y} = (6, 0, 0)$. The best line corresponds to the point $C\mathbf{1} + D\mathbf{t}$ of the plane $\operatorname{span}(\mathbf{1}, \mathbf{t})$ closest to $\mathbf{y}$. For a line, the closest point left a residual orthogonal to the line. For a plane we expect the residual to be orthogonal to the whole plane, hence to both $\mathbf{1}$ and $\mathbf{t}$ (proved below). That gives two equations for the two unknowns:

$$\begin{aligned} \mathbf{1} \cdot (\mathbf{y} - C\mathbf{1} - D\mathbf{t}) &= 6 - 3C - 3D = 0, \\ \mathbf{t} \cdot (\mathbf{y} - C\mathbf{1} - D\mathbf{t}) &= 0 - 3C - 5D = 0. \end{aligned}$$

Subtracting the first equation from the second gives $2D = -6$, so $D = -3$ and $C = 5$. The best line is $5 - 3t$. Its values at the three times are $(5, 2, -1)$, and the errors are $\mathbf{r} = \mathbf{y} - (5, 2, -1) = (1, -2, 1)$. Both checks pass, $\mathbf{r} \cdot \mathbf{1} = 1 - 2 + 1 = 0$ and $\mathbf{r} \cdot \mathbf{t} = 0 - 2 + 2 = 0$, and Pythagoras splits the squared length of the measurements: $36 = 30 + 6$. For contrast, projecting $\mathbf{y}$ onto $\mathbf{1}$ alone gives the coefficient $\mathbf{y} \cdot \mathbf{1} / \mathbf{1} \cdot \mathbf{1} = 2$, the mean of the measurements, with residual $(4, -2, -2)$ of squared length 24. Adding the direction $\mathbf{t}$ cuts the squared error from 24 to 6. Figure 33.5 shows the fit in $\mathbb{R}^3$.

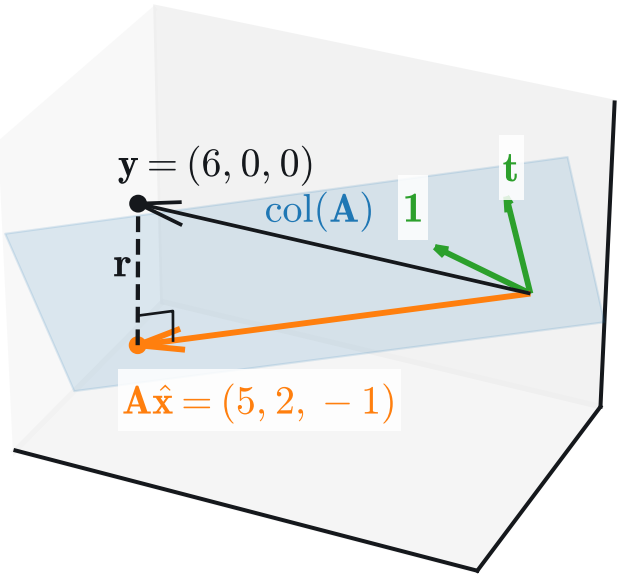


Figure 33.5: Least squares as projection: the measurements $\mathbf{y} = (6, 0, 0)$ do not lie in the plane of vectors $C\mathbf{1} + D\mathbf{t}$; the closest point of the plane is $(5, 2, -1)$, the fit $5 - 3t$, and the error $(1, -2, 1)$ is perpendicular to both $\mathbf{1}$ and $\mathbf{t}$.

For the general statement it helps to have a basis whose vectors are orthogonal to each other. A basis $\mathbf{q}_1, \dots, \mathbf{q}_k$ of a subspace $S \subseteq \mathbb{R}^n$ is *orthonormal* when each vector has unit length and distinct vectors are orthogonal. Such a basis always exists: Section 33.1.2.3 builds one from any finite list of vectors that spans S, such as the columns of a matrix, and it needs nothing beyond projection onto a line. Stacking the vectors as the columns of an $n \times k$ matrix $\mathbf{Q}$ writes orthonormality compactly as $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}_k$, because entry (i, j) of $\mathbf{Q}^\top \mathbf{Q}$ is $\mathbf{q}_i \cdot \mathbf{q}_j$. In this chapter the letter $\mathbf{Q}$ always denotes a matrix with orthonormal columns. The proposition that follows is the result of this section that the rest of the chapter uses most.

Proposition (projection onto a subspace). *Let* $\mathbf{Q}$ *have orthonormal columns spanning* S*, and set* $\mathbf{P} = \mathbf{Q}\mathbf{Q}^\top$*. Then for every* $\mathbf{x} \in \mathbb{R}^n$*:* (i) $\mathbf{P}\mathbf{x} \in S$*, and the residual* $\mathbf{x} - \mathbf{P}\mathbf{x}$ *is orthogonal to every vector of* S*;* (ii) $\mathbf{P}\mathbf{x}$ *is the unique closest point of* S *to* $\mathbf{x}$*;* (iii) $\mathbf{P}^2 = \mathbf{P}$ *and* $\mathbf{P}^\top = \mathbf{P}$*.*

Proof. For (i), $\mathbf{P}\mathbf{x} = \mathbf{Q}(\mathbf{Q}^\top \mathbf{x})$ is a linear combination of the columns of $\mathbf{Q}$, hence lies in S. Multiplying the residual by $\mathbf{Q}^\top$ gives

$$\mathbf{Q}^\top(\mathbf{x} - \mathbf{P}\mathbf{x}) = \mathbf{Q}^\top \mathbf{x} - (\mathbf{Q}^\top \mathbf{Q})\,\mathbf{Q}^\top \mathbf{x} = \mathbf{0},$$

so the residual is orthogonal to each $\mathbf{q}_i$, and therefore to every combination of them. For (ii), take any $\mathbf{s} \in S$ and split $\mathbf{x} - \mathbf{s} = (\mathbf{x} - \mathbf{P}\mathbf{x}) + (\mathbf{P}\mathbf{x} - \mathbf{s})$. The second piece lies in S and the first is orthogonal to it, so Pythagoras Equation 33.3 gives

$$\|\mathbf{x} - \mathbf{s}\|^2 = \|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2 + \|\mathbf{P}\mathbf{x} - \mathbf{s}\|^2 \geq \|\mathbf{x} - \mathbf{P}\mathbf{x}\|^2,$$

with equality only when $\mathbf{s} = \mathbf{P}\mathbf{x}$. For (iii), $\mathbf{P}^2 = \mathbf{Q}(\mathbf{Q}^\top \mathbf{Q})\mathbf{Q}^\top = \mathbf{Q}\mathbf{Q}^\top = \mathbf{P}$, and $\mathbf{P}^\top = (\mathbf{Q}^\top)^\top \mathbf{Q}^\top = \mathbf{P}$. ■

The orthogonality in (i) also characterizes the closest point. If $\mathbf{s} \in S$ and $\mathbf{x} - \mathbf{s}$ is orthogonal to S, then $\mathbf{P}\mathbf{x} - \mathbf{s} = (\mathbf{x} - \mathbf{s}) - (\mathbf{x} - \mathbf{P}\mathbf{x})$ lies in S and is orthogonal to S, hence to itself, so it is zero. This is why the two equations of the line fit found the closest point.

Property (iii) says two things. $\mathbf{P}^2 = \mathbf{P}$ means projecting twice changes nothing, because the first projection already lies in S. $\mathbf{P}^\top = \mathbf{P}$ is what makes the projection *orthogonal*. A matrix with $\mathbf{P}^2 = \mathbf{P}$ but $\mathbf{P}^\top \neq \mathbf{P}$ is an *oblique* projector: it moves vectors into its column space along a slanted direction and in general does not return the closest point (5). For $k = 1$, with the single unit column $\mathbf{q} = \mathbf{w}/\|\mathbf{w}\|$, the matrix $\mathbf{P} = \mathbf{q}\mathbf{q}^\top$ reproduces Equation 33.2.

The line fit is one instance of *least squares*, the fitting problem behind linear regression. Let $\mathbf{A}$ be a data matrix whose columns are features and whose rows are examples, and let $\mathbf{y}$ hold the targets. Least squares asks for the coefficient vector $\mathbf{x}$ that minimizes the squared error,

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|^2. \tag{33.8}$$

As $\mathbf{x}$ ranges over all coefficient vectors, $\mathbf{A}\mathbf{x}$ ranges over the span of the columns of $\mathbf{A}$, its *column space* $\mathrm{col}(\mathbf{A})$. So least squares asks for the point of $\mathrm{col}(\mathbf{A})$ closest to $\mathbf{y}$, and by the proposition this is the point whose residual is orthogonal to every column. Writing "orthogonal to every column" as $\mathbf{A}^\top(\mathbf{y} - \mathbf{A}\hat{\mathbf{x}}) = \mathbf{0}$ gives the *normal equations*

$$\mathbf{A}^\top \mathbf{A}\,\hat{\mathbf{x}} = \mathbf{A}^\top \mathbf{y}. \tag{33.9}$$

They are the general form of the 2×2 system we solved by hand: for the line fit, $\mathbf{A} = [\mathbf{1}\ \mathbf{t}]$ has $\mathbf{A}^\top\mathbf{A} = \left(\begin{smallmatrix} 3 & 3 \\ 3 & 5 \end{smallmatrix}\right)$ and $\mathbf{A}^\top\mathbf{y} = (6, 0)$. When the columns of $\mathbf{A}$ are linearly independent, $\mathbf{A}^\top\mathbf{A}$ is invertible (1) and the coefficients are unique. When they are dependent, the fitted point $\mathbf{A}\hat{\mathbf{x}}$ is still unique, being the closest point of the column space, but several coefficient vectors produce it.

In code there are thus two routes to the fit $5 - 3t$: solve the normal equations, or project with $\mathbf{P} = \mathbf{Q}\mathbf{Q}^\top$. For $\mathbf{Q}$ we call the library routine `np.linalg.qr`, which returns an orthonormal basis of the column space of a matrix whose columns are independent (we build one by hand below).

```
A = np.array([[1., 0], [1, 1], [1, 2]])     # columns 1 and t
y = np.array([6., 0, 0])
x_hat = np.linalg.solve(A.T @ A, A.T @ y)   # normal equations
Q, R = np.linalg.qr(A)                      # orthonormal basis of col(A)
P = Q @ Q.T
r = y - P @ y
print(x_hat.round(6), (P @ y).round(6), r.round(6))
print(np.linalg.norm(P @ P - P), np.linalg.norm(Q.T @ r))
```

```
[ 5. -3.] [ 5.  2. -1.] [ 1. -2.  1.]
1.962615573354719e-16 0.0
```

Both routes return what we found by hand: the coefficients $C = 5$ and $D = -3$, the fitted values $(5, 2, -1)$ as the projection $\mathbf{Py}$, and the residual $(1, -2, 1)$. The last two numbers are zero up to roundoff, as the proposition requires.

On this small problem the two routes agree. They need not when the columns are nearly dependent: forming $\mathbf{A}^\top\mathbf{A}$ squares the condition number of the problem, its sensitivity to rounding errors (Section 33.5.3.2), which is why library least-squares solvers work with an orthonormal basis of the column space instead.

33.1.2.3 Gram–Schmidt and QR

An orthonormal basis can be built by repeated projection. Take two independent vectors, keep the direction of the first, and remove from the second its shadow on the first. What remains is orthogonal to the first vector, by the projection proposition.

Start with $\mathbf{a}_1 = (3, 4)$ and $\mathbf{a}_2 = (1, 0)$. Normalizing the first gives $\mathbf{q}_1 = \mathbf{a}_1/5 = (0.6, 0.8)$. The shadow of $\mathbf{a}_2$ on $\mathbf{q}_1$ is $(\mathbf{q}_1 \cdot \mathbf{a}_2)\,\mathbf{q}_1 = 0.6\,(0.6, 0.8)$, and subtracting it leaves $(1, 0) - (0.36, 0.48) = (0.64, -0.48)$, a vector of length 0.8 that is orthogonal to $\mathbf{q}_1$ ($0.384 - 0.384 = 0$). Normalizing it gives $\mathbf{q}_2 = (0.8, -0.6)$. Reading the construction backward expresses each original vector in the new basis:

$$\mathbf{a}_1 = 5\,\mathbf{q}_1, \qquad \mathbf{a}_2 = 0.6\,\mathbf{q}_1 + 0.8\,\mathbf{q}_2.$$

In matrix form this is $\mathbf{A} = \mathbf{QR}$ with $\mathbf{A} = [\mathbf{a}_1\ \mathbf{a}_2]$, $\mathbf{Q} = [\mathbf{q}_1\ \mathbf{q}_2]$, and $\mathbf{R} = \left(\begin{smallmatrix} 5 & 0.6 \\ 0 & 0.8 \end{smallmatrix}\right)$, whose entries are the dot products $\mathbf{q}_i \cdot \mathbf{a}_j$. The matrix $\mathbf{R}$ is upper triangular because $\mathbf{a}_j$ involves only $\mathbf{q}_1, \ldots, \mathbf{q}_j$.

The general procedure is the *Gram–Schmidt process*. Process linearly independent vectors $\mathbf{a}_1, \ldots, \mathbf{a}_k$ in order. For each $\mathbf{a}_j$, subtract its projections Equation 33.2 onto the directions

$\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$ already produced, and normalize what remains. The output is an orthonormal basis of $\operatorname{span}(\mathbf{a}_1, \dots, \mathbf{a}_k)$ and the *QR factorization* $\mathbf{A} = \mathbf{QR}$, with $\mathbf{Q}$ having orthonormal columns and $\mathbf{R}$ upper triangular with positive diagonal. Independence is needed for the factorization: a vector in the span of its predecessors leaves a zero remainder, which cannot be normalized. For an orthonormal basis of the span alone we skip such a vector and continue, so any spanning list will do. This is the existence claim we used for the projection onto a subspace.

Library routines compute the same factorization by a different algorithm, based on reflections, which is more accurate in floating point. Their answer agrees with Gram–Schmidt only up to signs: flipping the sign of a column of $\mathbf{Q}$ together with the corresponding row of $\mathbf{R}$ leaves the product unchanged. When $\mathbf{A}$ has independent columns, these sign flips are the only freedom. So when we run the two steps above next to `np.linalg.qr`, we should expect the same columns, possibly negated.

```
a1, a2 = np.array([3., 4]), np.array([1., 0])
q1 = a1 / np.linalg.norm(a1)
v = a2 - (q1 @ a2) * q1                    # remove the shadow on q1
q2 = v / np.linalg.norm(v)
Q_qr, R_qr = np.linalg.qr(np.stack([a1, a2], axis=1))
print(np.stack([q1, q2], axis=1).round(6))
print(Q_qr.round(6))
print(R_qr.round(6))
```

```
[[ 0.6  0.8]
 [ 0.8 -0.6]]
[[-0.6 -0.8]
 [-0.8  0.6]]
[[-5.  -0.6]
 [ 0.  -0.8]]
```

Gram–Schmidt gives the columns $(0.6, 0.8)$ and $(0.8, -0.6)$. The library returns the negatives of both, with $\mathbf{R}$'s diagonal -5 and -0.8: the same basis and, up to sign, the same dot products 5, 0.6, 0.8 that we computed by hand. In floating point the order of the subtractions matters. On nearly dependent columns, the version above, called *classical* Gram–Schmidt, loses orthogonality (8).

33.1.3 Similarity in High Dimensions

So far every result came with a picture in the plane or in $\mathbb{R}^3$. We now ask what the results say where no picture is possible: how similar two images are, and what similarity to expect between vectors that are unrelated. We load the images of the opening through `d2l.FashionMNIST` (Section 3.2) as NumPy arrays, one flattened image of 784 pixels per row, with pixel values in $[0, 1]$.

```
data = d2l.FashionMNIST()
X_train = data.train.data.numpy().reshape(-1, 784) / 255
y_train = data.train.targets.numpy()
X_test = data.val.data.numpy().reshape(-1, 784) / 255
y_test = data.val.targets.numpy()
X_train.shape, X_test.shape
```

```
((60000, 784), (10000, 784))
```

Each of the $70{,}000$ images is a point in $\mathbb{R}^{784}$.

33.1.3.1 Cosine Similarity

The ratio in Equation 33.6 is the *cosine similarity* of Section 1.3,

$$\cos\theta = \frac{\mathbf{v}\cdot\mathbf{w}}{\|\mathbf{v}\|\,\|\mathbf{w}\|} \;\in\; [-1,1], \tag{33.10}$$

which compares directions and ignores lengths: an image and a copy of it dimmed to 10% brightness are far apart pixel by pixel, yet their cosine similarity is 1.

This invariance makes cosine similarity the standard score for nearest-neighbor retrieval over embeddings, and for contrastive objectives such as SimCLR (Chen et al. 2020), which pull the embeddings of two views of the same image together in direction. Attention uses a different score: the *unnormalized* dot product, scaled by $1/\sqrt{d}$ as explained below, so the lengths of queries and keys matter there.

33.1.3.2 Near-Orthogonality of Random Directions

Two unrelated images, or two unrelated embeddings: what cosine should we expect between them by chance? Without an answer, no observed cosine can be called large. The cleanest reference is a direction chosen uniformly at random, with no preference for any part of the space.

Proposition (near-orthogonality). *Fix a unit vector* $\mathbf{u}\in\mathbb{R}^d$ *and let* $\mathbf{v}$ *be drawn uniformly from the unit sphere in* $\mathbb{R}^d$. *Then* $\cos\theta = \mathbf{u}\cdot\mathbf{v}$ *satisfies*

$$\mathbb{E}[\cos\theta] = 0, \qquad \mathrm{Var}(\cos\theta) = \frac{1}{d}, \qquad \Pr(|\cos\theta| \ge \varepsilon) \le \frac{1}{d\,\varepsilon^2} \quad \text{for every } \varepsilon > 0.$$

Proof. The uniform distribution on the sphere does not change under rotations, so we may rotate coordinates until $\mathbf{u} = \mathbf{e}_1$; then $\cos\theta = v_1$. The distribution is also unchanged by $\mathbf{v}\mapsto -\mathbf{v}$, so $\mathbb{E}[v_1] = 0$. For the variance, every coordinate plays the same role, so $\mathbb{E}[v_i^2]$ is the same for all i. Since $\sum_i v_i^2 = \|\mathbf{v}\|^2 = 1$, we get $d\,\mathbb{E}[v_1^2] = \mathbb{E}\big[\sum_i v_i^2\big] = 1$, hence $\mathrm{Var}(\cos\theta) = \mathbb{E}[v_1^2] = 1/d$. The tail bound is Chebyshev's inequality $\Pr(|X-\mathbb{E}[X]|\ge\varepsilon)\le \mathrm{Var}(X)/\varepsilon^2$ of Section 1.6. ■

The typical cosine between random directions is of size $1/\sqrt{d}$. For $d = 784$ the standard deviation is $1/28 \approx 0.036$, so two random directions in the space of images are within a few degrees of orthogonal. Chebyshev bounds the chance of a cosine above 0.1 in absolute value by $1/(784\cdot 0.01) \approx 0.13$; the true probability is below 0.01. The bound is loose because it uses only the variance; exponential bounds of the same kind appear in Section 36.9.

Low dimensions behave differently (Figure 33.6). In the plane ($d = 2$) the angle between a fixed and a random direction is uniform on $[0,\pi]$, so "within 0.1 radians of orthogonal" and "within 0.1 radians of collinear" are equally likely, about 6% each. The *cosine*, however, piles up at ± 1, because $\cos\theta$ changes slowly near $\theta = 0$ and $\theta = \pi$. The probability of $|\cos\theta| > 0.9$ is about 0.29, against about 0.06 for $|\cos\theta| < 0.1$. In $d = 3$ the cosine is uniform on $[-1,1]$, a fact known to Archimedes. From $d = 4$ on, the density has its peak

at 0; by $d = 10$ it is a bell of width $1/\sqrt{10} \approx 0.32$, and by $d = 1000$ a spike of width about 0.03 (6 derives the densities).

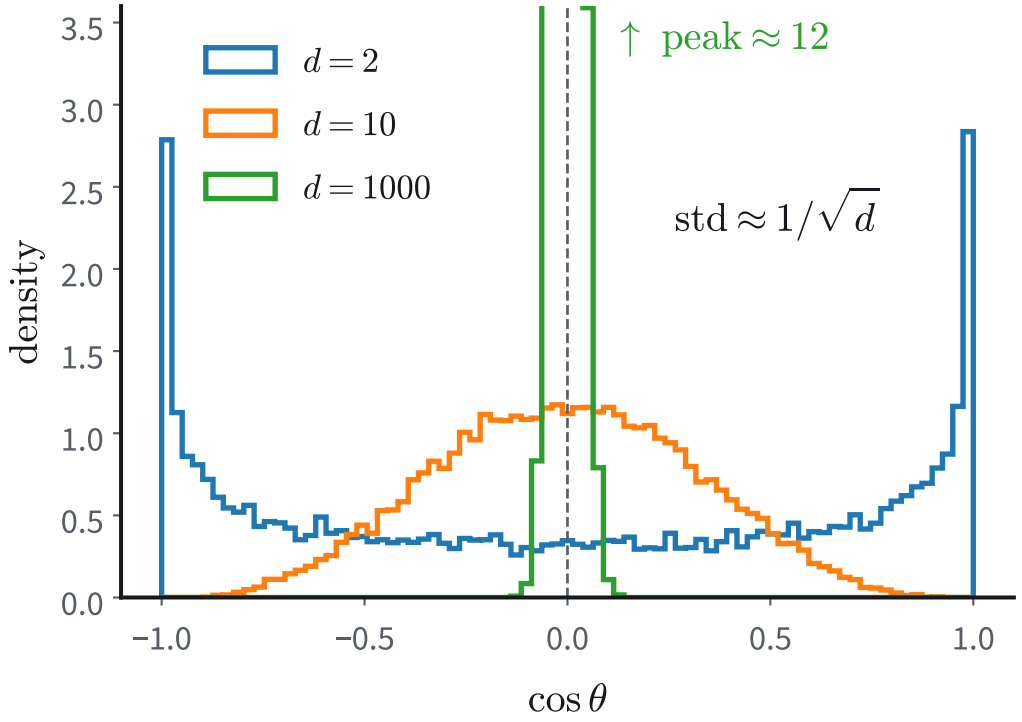


Figure 33.6: Histograms of the cosine between independent random unit vectors at dimensions $d = 2$, 10, and 1000. In $d = 2$ the cosine piles up near ± 1 although the angle is uniform. In moderate dimension the histogram is a bell around 0, and by $d = 1000$ it is a narrow spike of width $\approx 1/\sqrt{d}$, as the proposition predicts.

A simulation in $d = 784$ gives the reference that we will hold the images against. We draw random unit vectors by normalizing Gaussian vectors, whose distribution does not change under rotations. The same cell estimates a quantity derived for attention below: the variance of $\mathbf{q} \cdot \mathbf{k}$ for independent standard normal entries, in 64 dimensions.

```
rng = np.random.default_rng()
d = 784
U = rng.standard_normal((10000, d))
U /= np.linalg.norm(U, axis=1, keepdims=True)
V = rng.standard_normal((10000, d))
V /= np.linalg.norm(V, axis=1, keepdims=True)
q, k = rng.standard_normal((10000, 64)), rng.standard_normal((10000, 64))
np.std((U * V).sum(axis=1)), 1 / np.sqrt(d), np.var((q * k).sum(axis=1))
```

```
(np.float64(0.03554061363225),
 np.float64(0.03571428571428571),
 np.float64(65.38802562506136))
```

The standard deviation is about 0.036, as the proposition says, and the variance of the raw dot product is about 64, the dimension.

Real data need not follow the isotropic reference. Do two unrelated Fashion-MNIST images look orthogonal? We compare $5{,}000$ random pairs of test images, first as they are and then after subtracting the mean training image.

```
def cosines(A, B):
    """Cosine similarity between corresponding rows of A and B."""
    return (A * B).sum(axis=1) / (np.linalg.norm(A, axis=1)
                                  * np.linalg.norm(B, axis=1))

i, j = rng.integers(0, len(X_test), (2, 5000))
mean_image = X_train.mean(axis=0)
raw = cosines(X_test[i], X_test[j])
centered = cosines(X_test[i] - mean_image, X_test[j] - mean_image)
raw.mean(), raw.std(), centered.mean(), centered.std()
```

```
(np.float64(0.5921925878554625),
 np.float64(0.1767312159719317),
 np.float64(0.0009877647378530555),
 np.float64(0.3397352598108595))
```

They do not. Raw images have cosines around 0.6 with a standard deviation of about 0.18, nowhere near 0 ± 0.036. Pixels are non-negative, so every image lies in the same orthant of $\mathbb{R}^{784}$ and no pair has a negative dot product. Subtracting the mean image centers the cosines at 0, but their standard deviation, about 0.34, is nearly ten times the isotropic value. By the proposition, it is the spread of random directions in eight or nine dimensions ($1/0.34^2$ lies between 8 and 9). Measured by their cosines, the centered images behave like a cloud in fewer than ten dimensions, not 784, a first sign of the low-rank structure of Section 33.5. A cosine therefore counts as large only when it exceeds the spread of unrelated pairs from the same data. Retrieval over embeddings relies on training to push related pairs above that spread.

33.1.3.3 The Scale of Attention Scores

Dot-product attention (Section 10.2) compares a query $\mathbf{q} \in \mathbb{R}^d$ with keys $\mathbf{k}_1, \mathbf{k}_2, \ldots$ and turns the scores $\mathbf{q} \cdot \mathbf{k}_j / \sqrt{d}$ into weights with a softmax (Vaswani et al. 2017). A short variance computation explains the $\sqrt{d}$. Suppose all $2d$ entries of $\mathbf{q}$ and $\mathbf{k}$ are mutually independent, with mean zero and variance one. Each summand $q_i k_i$ of $\mathbf{q} \cdot \mathbf{k}$ then has mean zero and variance $\mathbb{E}[q_i^2]\,\mathbb{E}[k_i^2] = 1$, and distinct summands are uncorrelated, since $\mathbb{E}[(q_i k_i)(q_j k_j)] = \mathbb{E}[q_i]\,\mathbb{E}[k_i]\,\mathbb{E}[q_j]\,\mathbb{E}[k_j] = 0$ for $i \neq j$. The variances of the d summands therefore add:

$$\operatorname{Var}(\mathbf{q} \cdot \mathbf{k}) = \sum_{i=1}^{d} \operatorname{Var}(q_i k_i) = d.$$

The raw scores have standard deviation $\sqrt{d}$, which grows with the width of the head, and dividing by $\sqrt{d}$ standardizes them. Without the division the softmax saturates. With $d = 64$ the raw scores have standard deviation 8, and two scores 8 apart get softmax weights 0.9997 and 0.0003; divided by 8, they get 0.73 and 0.27. The geometric reading agrees

with the computation: each vector has length about $\sqrt{d}$, and the cosine between unrelated directions has size about $1/\sqrt{d}$, so their dot product has size about $\sqrt{d} \cdot \sqrt{d} \cdot 1/\sqrt{d} = \sqrt{d}$.

Near-orthogonality does not make unrelated keys harmless. Each unrelated cosine is of order $1/\sqrt{d}$ and centered at 0, but the largest of n of them grows, to leading order, like $\sqrt{2 \ln n/d}$, which for a large index is several times the typical size $1/\sqrt{d}$. A related key must beat that level, which is what training arranges (2 puts numbers on it).

33.1.4 Hyperplanes and Decision Boundaries

A linear classifier answers two questions about an input: on which side of a boundary does it lie, and how far from the boundary is it? Throughout, we write the boundary as $\mathbf{w} \cdot \mathbf{x} + b = 0$.

33.1.4.1 Hyperplanes and Signed Distance

In the plane the boundary is a line, and a drawing shows how $\mathbf{w}$ and b place it. Take $\mathbf{w} = (2, 1)$ and $b = -1$. The points with $2x + y - 1 = 0$ form the line $y = 1 - 2x$. Every point $\mathbf{x}$ on it has $\mathbf{w} \cdot \mathbf{x} = -b$, so its shadow on $\mathbf{w}$ has the same signed length Equation 33.4,

$$\frac{\mathbf{w} \cdot \mathbf{x}}{\|\mathbf{w}\|} = \frac{-b}{\|\mathbf{w}\|} = \frac{1}{\sqrt{5}}.$$

So the line is perpendicular to $\mathbf{w}$ and crosses it at signed position $-b/\|\mathbf{w}\|$ along $\mathbf{w}$, at distance $|b|/\|\mathbf{w}\| = 1/\sqrt{5} \approx 0.45$ from the origin.

The same description works in any dimension. A *hyperplane* in $\mathbb{R}^d$ is a set $\{\mathbf{x} : \mathbf{w} \cdot \mathbf{x} + b = 0\}$ with $\mathbf{w} \neq \mathbf{0}$; the vector $\mathbf{w}$ is its *normal* and b its *offset*. It is a shifted copy of the subspace $\mathbf{w} \cdot \mathbf{x} = 0$, which has dimension $d - 1$ (Section 33.2.2.2): a line in the plane, a plane in $\mathbb{R}^3$, and a 783-dimensional flat in the space of images. It divides the space into two *half-spaces*, $\mathbf{w} \cdot \mathbf{x} + b > 0$ and $\mathbf{w} \cdot \mathbf{x} + b < 0$. Changing b slides the hyperplane along $\mathbf{w}$ without rotating it.

How far is an arbitrary point $\mathbf{x}$ from the hyperplane? Pick any point $\mathbf{x}_0$ on it, so that $\mathbf{w} \cdot \mathbf{x}_0 = -b$. The distance is the signed length of the shadow of the displacement $\mathbf{x} - \mathbf{x}_0$ on the normal, so the *signed distance* is

$$\frac{\mathbf{w} \cdot (\mathbf{x} - \mathbf{x}_0)}{\|\mathbf{w}\|} = \frac{\mathbf{w} \cdot \mathbf{x} + b}{\|\mathbf{w}\|}. \tag{33.11}$$

It does not depend on which $\mathbf{x}_0$ we picked. It is positive on the side that $\mathbf{w}$ points to, negative on the other, and zero on the hyperplane. In the example, the origin is at signed distance $-1/\sqrt{5}$, and the point $(1, 1)$ has score $\mathbf{w} \cdot \mathbf{x} + b = 2$, so it lies at $+2/\sqrt{5} \approx 0.89$ on the side $\mathbf{w}$ points to; its closest point on the line is $(1, 1) - (2/5)(2, 1) = (0.2, 0.6)$. The point $(0, 1)$ scores 0 and lies on the line (Figure 33.7).

A classifier also needs to know whether a point lies on the correct side. For a labeled point with $y \in \{-1, +1\}$, the *margin* is $y\,(\mathbf{w} \cdot \mathbf{x} + b)/\|\mathbf{w}\|$, the signed distance multiplied by the label: positive when the point is on its own side, and larger the farther it is from the boundary. Scaling $(\mathbf{w}, b) \to (c\mathbf{w}, cb)$ with $c > 0$ leaves the hyperplane and the margin unchanged but multiplies the score by c. A classifier that maximizes the score must therefore fix the scale of $\mathbf{w}$ (3).

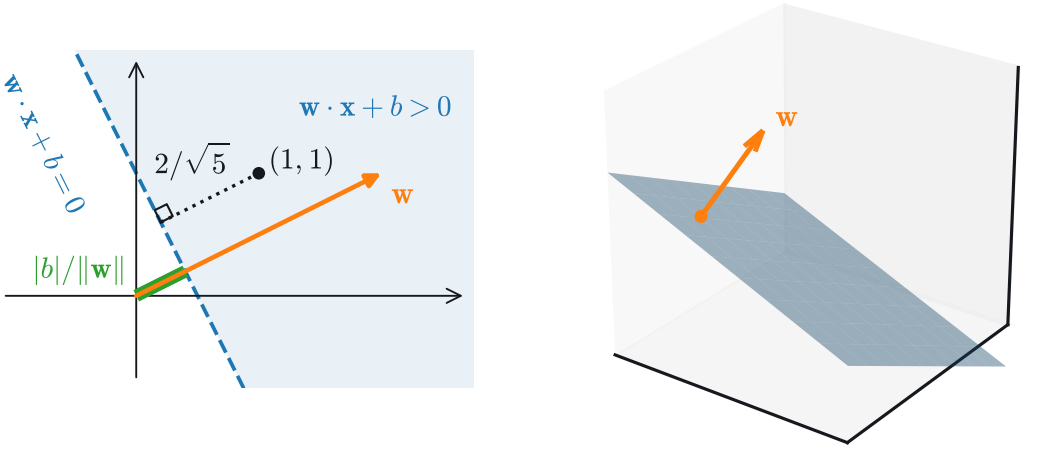


Figure 33.7: The hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ with normal $\mathbf{w}$. Left: for $\mathbf{w} = (2, 1)$ and $b = -1$ it is the line $2x + y = 1$ at distance $1/\sqrt{5}$ from the origin; the point $(1, 1)$ lies at signed distance $2/\sqrt{5}$ on the side $\mathbf{w}$ points to, and the shaded half-plane is $\mathbf{w} \cdot \mathbf{x} + b > 0$. Right: in three dimensions the same equation defines a plane.

33.1.4.2 The Nearest-Mean Classifier

Which hyperplane should we choose? The simplest classifier we can think of needs no training: assign $\mathbf{x}$ to the class whose mean is nearer. It turns out to be a hyperplane as well. Let $\bar{\mathbf{x}}_0$ and $\bar{\mathbf{x}}_1$ be the means of the two classes. Expanding both squared distances, the term $\|\mathbf{x}\|^2$ cancels:

$$\begin{aligned} \|\mathbf{x} - \bar{\mathbf{x}}_1\|^2 < \|\mathbf{x} - \bar{\mathbf{x}}_0\|^2 &\iff -2\,\mathbf{x} \cdot \bar{\mathbf{x}}_1 + \|\bar{\mathbf{x}}_1\|^2 < -2\,\mathbf{x} \cdot \bar{\mathbf{x}}_0 + \|\bar{\mathbf{x}}_0\|^2 \\ &\iff (\bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0) \cdot \mathbf{x} - \tfrac{1}{2}(\|\bar{\mathbf{x}}_1\|^2 - \|\bar{\mathbf{x}}_0\|^2) > 0. \end{aligned}$$

With $\|\mathbf{a}\|^2 - \|\mathbf{c}\|^2 = (\mathbf{a} - \mathbf{c}) \cdot (\mathbf{a} + \mathbf{c})$, the last line is $\mathbf{w} \cdot \mathbf{x} + b > 0$ for

$$\mathbf{w} = \bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0, \qquad b = -\mathbf{w} \cdot \tfrac{1}{2}\,(\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1)\,.$$

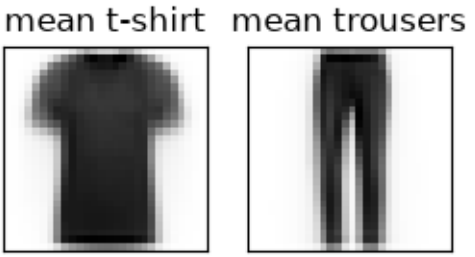
So "nearest mean" is one hyperplane: its normal points from one mean to the other, and it passes through their midpoint. Nothing about the threshold is chosen by hand. Nor does the rule depend on the units of the pixels: rescaling every pixel by $c > 0$ gives $\mathbf{w} \to c\mathbf{w}$ and $b \to c^2 b$ and multiplies every score by c^2, so no prediction changes (4 extends this to rotations and shifts, and finds a per-pixel rescaling that does change predictions).

Before applying the rule to t-shirts and trousers, we look at the two means and at the angle between them.

```
X0, X1 = X_train[y_train == 0], X_train[y_train == 1]    # t-shirts, trousers
mean_0, mean_1 = X0.mean(axis=0), X1.mean(axis=0)
d2l.show_images([mean_0.reshape(28, 28), mean_1.reshape(28, 28)], 1, 2,
                titles=['mean t-shirt', 'mean trousers'], cmap='Greys')
```

```
np.degrees(angle(mean_0, mean_1))
```

```
np.float64(33.20917997376519)
```



The means are blurry but recognizable. Their angle is about 33° (cosine 0.84), far from orthogonal: both are bright in the middle of the frame. With the two means only 33° apart, one might not expect much of the rule. But the classifier does not need the means to be orthogonal. It uses only their difference. We apply it to the $2{,}000$ test images of the two classes.

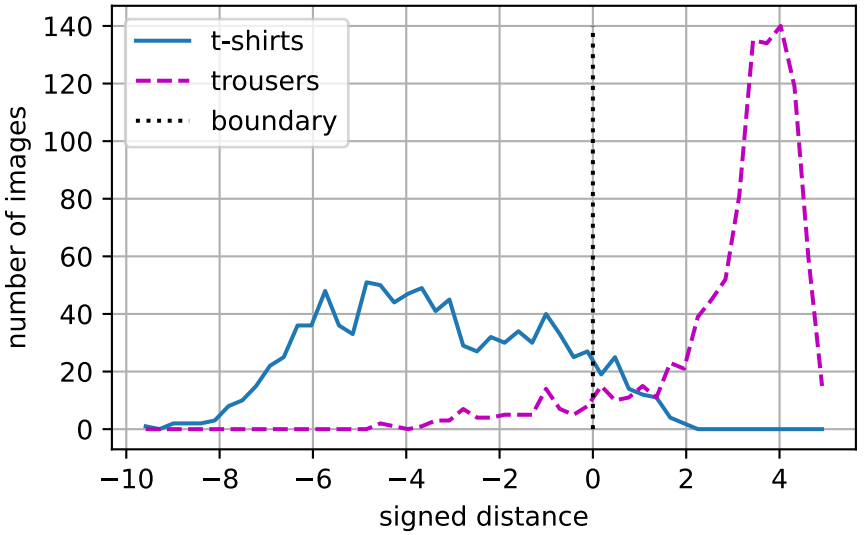
```
w = mean_1 - mean_0
b = -w @ (mean_0 + mean_1) / 2
mask = (y_test == 0) | (y_test == 1)
X2, y2 = X_test[mask], y_test[mask]
predictions = X2 @ w + b > 0                              # True: trousers
np.mean(predictions == (y2 == 1))
```

```
np.float64(0.9155)
```

The rule is right on about 92% of the test images: 169 of the $2{,}000$ lie on the wrong side. We computed two means and asked of each image only on which side of one hyperplane it lies.

To see where the errors come from, we reduce every test image to one number, its signed distance Equation 33.11 to the boundary, and plot the histogram of each class.

```
dist = (X2 @ w + b) / np.linalg.norm(w)         # signed distance to boundary
bins = np.linspace(dist.min(), dist.max(), 51)
counts = [np.histogram(dist[y2 == c], bins)[0] for c in (0, 1)]
top = max(counts[0].max(), counts[1].max())
d2l.plot([bins[:-1]] * 2 + [[0, 0]], counts + [[0, top]], 'signed distance',
         'number of images', fmts=('-', 'm--', 'k:'),
         legend=['t-shirts', 'trousers', 'boundary'], figsize=(5, 3))
```



The two classes form two modes whose tails overlap at the boundary. The t-shirts make a broad hump around -4; the trousers make a sharp peak near $+4$ with a long tail toward the boundary. On average each class sits near the signed distance of its training mean, $\mp\|\mathbf{w}\|/2 \approx \mp 3.2$. Two numbers per class make this precise: its standard deviation along $\mathbf{w}$, and its count on the wrong side.

```
(dist[y2 == 0].std(), dist[y2 == 1].std(),
 int((dist[y2 == 0] > 0).sum()), int((dist[y2 == 1] < 0).sum()))
```

```
(np.float64(2.356225995048113), np.float64(1.6627004538100882), 101, 68)
```

The t-shirts spread with a standard deviation of about 2.4 along $\mathbf{w}$, the trousers with about 1.7; 101 t-shirts and 68 trousers are on the wrong side. Each class sits less than two of its standard deviations from the boundary, so errors of several percent are no surprise. The midpoint rule uses only the two means. It has no way of representing the size or the shape of a class. Does it survive on pairs of classes that look alike, t-shirts against shirts and pullovers against coats?

```
# t-shirt/trouser, t-shirt/shirt, pullover/coat
for a, c in [(0, 1), (0, 6), (2, 4)]:
    m_a = X_train[y_train == a].mean(axis=0)
    m_c = X_train[y_train == c].mean(axis=0)
    w_ac, b_ac = m_c - m_a, -(m_c - m_a) @ (m_a + m_c) / 2
    pair = (y_test == a) | (y_test == c)
    acc = np.mean((X_test[pair] @ w_ac + b_ac > 0) == (y_test[pair] == c))
    dist_ac = (X_test[pair] @ w_ac + b_ac) / np.linalg.norm(w_ac)
    sd_a, sd_c = (dist_ac[y_test[pair] == k].std() for k in (a, c))
    names = data.text_labels([a, c])
    print(f'{names[0]:>8} vs {names[1]:<9}accuracy {acc:.3f}   '
          f'||w||/2 {np.linalg.norm(w_ac) / 2:.2f}   '
          f'std along w {sd_a:.2f}, {sd_c:.2f}')
```

```
 T-shirt vs trouser  accuracy 0.915   ||w||/2 3.23   std along w 2.36, 1.66
 T-shirt vs shirt    accuracy 0.786   ||w||/2 1.86   std along w 1.77, 2.38
pullover vs coat     accuracy 0.716   ||w||/2 1.24   std along w 1.88, 2.09
```

It does not. T-shirts against shirts drop to about 79%, and pullovers against coats to about 72%. The other columns show why. Each training mean lies $\|\mathbf{w}\|/2$ from the boundary, and what counts is this distance measured in standard deviations of the class along $\mathbf{w}$. For t-shirts and trousers it is 1.4 to 1.9 standard deviations. For the two harder pairs it is only 0.6 to 1.0, and a boundary that close to the middle of a class cuts off a large part of it. The remedy is to tilt $\mathbf{w}$ away from directions in which the classes spread, which is Fisher's discriminant (7). Softmax regression learns to move and tilt the same kind of boundary, and a deep network goes further: it learns a representation in which the classes are nearly separable by one hyperplane.

33.1.4.3 Layers as Stacked Hyperplanes

A linear layer computes many such tests at once. Entry i of the product $\mathbf{Ax}$ is the dot product of row i of $\mathbf{A}$ with $\mathbf{x}$, a score $\mathbf{a}_i \cdot \mathbf{x}$ against the normal $\mathbf{a}_i$, and adding a bias makes it the score of a hyperplane. The softmax layer of Section 3.1 for the ten Fashion-MNIST classes holds ten normals $\mathbf{w}_1, \dots, \mathbf{w}_{10}$ and ten offsets. It predicts the class with the largest score, so classes i and j are separated by the hyperplane $(\mathbf{w}_i - \mathbf{w}_j) \cdot \mathbf{x} + (b_i - b_j) = 0$ on which their scores tie.

Take the chapter's running matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ and $\mathbf{x} = (2, -1)$. Row one gives $1 \cdot 2 + 2 \cdot (-1) = 0$, so $\mathbf{x}$ lies on the hyperplane through the origin with normal $(1, 2)$. Row two gives $-2 - 3 = -5$, so $\mathbf{x}$ lies on the negative side of the hyperplane with normal $(-1, 3)$, at signed distance $-5/\sqrt{10} \approx -1.58$. Together, $\mathbf{Ax} = (0, -5)$.

The row view shows a layer as a stack of hyperplane tests, but it does not tell us which inputs the map destroys, or whether its action can be undone. Reading the same product by columns answers these questions in Section 33.2.

33.1.5 Summary

Dot products give lengths, angles, and projections by formulas that are the same in every dimension. The projection onto a line or a subspace is the closest point, and its residual is orthogonal to that line or subspace; from this follow Cauchy–Schwarz, which makes the angle well defined, and the normal equations of least squares. Gram–Schmidt turns a basis into an orthonormal one, the $\mathbf{Q}$ of the QR factorization. Random directions in $\mathbb{R}^d$ are nearly orthogonal, with cosines of size $1/\sqrt{d}$, but real images are not, so a cosine is large only relative to unrelated pairs from the same data. The nearest-mean rule is the hyperplane through the midpoint of the class means; it reaches about 92% on t-shirts against trousers but only 72% to 79% on classes that look alike.

33.1.6 Exercises

1. **Fitting a line by projection.** Four measurements $\mathbf{y} = (0, 8, 8, 20)$ are taken at the times $\mathbf{t} = (0, 1, 3, 4)$. Let $\mathbf{1} = (1, 1, 1, 1)$. This is the four-point version of the fit in the text.

    1. Project $\mathbf{y}$ onto the line through $\mathbf{1}$ with Equation 33.2. Check that the residual is orthogonal to $\mathbf{1}$, confirm Equation 33.3, and find the distance from $\mathbf{y}$ to the line.

Which statistic of the measurements is the coefficient of the projection?
2. Fit the line $C + Dt$ by least squares Equation 33.8, where $\mathbf{A}$ has the columns $\mathbf{1}$ and $\mathbf{t}$. Write down the normal equations Equation 33.9, solve them, and compute the four errors and the sum of their squares. Why does C differ from the coefficient of the first part?
3. Suppose the columns of $\mathbf{A}$ are linearly independent, so that the Gram–Schmidt process gives $\mathbf{A} = \mathbf{QR}$ with $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$ and an invertible upper triangular $\mathbf{R}$. Show that $\mathbf{A}^\top\mathbf{A}$ is invertible and that $\mathbf{A}(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top = \mathbf{QQ}^\top$. The normal equations therefore yield the projection $\mathbf{P}$ without an orthonormal basis.

Adapted from Gilbert Strang, MIT 18.065 Matrix Methods in Data Analysis, Signal Processing, and Machine Learning, Problem Sets, Problems for Lecture 10, Problems 12 and 17.

2. code **Cosines between random vectors.**

   1. Sample $1{,}000$ pairs of independent random unit vectors in $d = 10{,}000$ dimensions by drawing Gaussian vectors and normalizing them, and compute the cosine of each pair. Compare the empirical standard deviation with the prediction $1/\sqrt{d} = 0.01$ of the near-orthogonality proposition, and find the largest $|\cos\theta|$ among the pairs.
   2. A retrieval system compares one query with n keys and returns the key of largest cosine. Let the query and the keys be independent random unit vectors in $d = 256$ dimensions. Compute the largest of the n cosines for $n = 10$, 10^3, and 10^5, and compare it with $\sqrt{2\ln n/d}$, the leading-order estimate of the maximum of n independent Gaussian draws with variance $1/d$. Which cosine does an unrelated key reach as the top match among 10^5 keys? How should a threshold for declaring two embeddings related depend on n and d?

3. **Closest point and margin.**

   1. Show that the point of the hyperplane $\mathbf{w}\cdot\mathbf{x} + b = 0$ closest to $\mathbf{x}$ is $\mathbf{x} - \frac{\mathbf{w}\cdot\mathbf{x}+b}{\|\mathbf{w}\|^2}\,\mathbf{w}$, and that its distance from $\mathbf{x}$ is the absolute value of the signed distance Equation 33.11. (Hint: subtract from $\mathbf{x} - \mathbf{x}_0$ its shadow on $\mathbf{w}$, for a point $\mathbf{x}_0$ of the hyperplane.)
   2. For $\mathbf{w} = (3, 4)$ and $b = -10$, compute the signed distance of $\mathbf{x} = (1, 1)$, the side of the hyperplane on which $\mathbf{x}$ lies, and the closest point of the hyperplane.
   3. Replace $(\mathbf{w}, b)$ by $(c\mathbf{w}, cb)$ with $c > 0$. Which of the hyperplane, the score $\mathbf{w}\cdot\mathbf{x} + b$, and the signed distance change? Let training points $\mathbf{x}_i$ with labels $y_i \in \{-1, +1\}$ be separable: some $(\mathbf{w}, b)$ gives $y_i(\mathbf{w}\cdot\mathbf{x}_i + b) > 0$ for every i. A classifier chooses $(\mathbf{w}, b)$ to maximize the smallest score $\min_i y_i(\mathbf{w}\cdot\mathbf{x}_i + b)$. Why does this maximization have no solution unless the scale of $\mathbf{w}$ is fixed? Which scale-invariant quantity can be maximized instead?

4. **Nearest means and invariance.** The nearest-mean rule assigns $\mathbf{x}$ to class 1 when $\mathbf{w}\cdot\mathbf{x} + b > 0$, with $\mathbf{w} = \bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0$ and $b = -\mathbf{w}\cdot\frac{1}{2}(\bar{\mathbf{x}}_0 + \bar{\mathbf{x}}_1)$.

   1. Every training and test image is transformed by the same map $\mathbf{x} \mapsto c\mathbf{Q}\mathbf{x} + \mathbf{z}$ with $c > 0$, a square $\mathbf{Q}$ with $\mathbf{Q}^\top\mathbf{Q} = \mathbf{I}$, and a fixed vector $\mathbf{z}$, and the class means and b are recomputed. Show that no prediction changes, and relate the result to the rescaling $\mathbf{w} \to c\mathbf{w}$, $b \to c^2 b$ of the text.
   2. Standardizing every pixel to unit variance replaces $\mathbf{x}$ by $\mathbf{D}\mathbf{x}$ for a diagonal $\mathbf{D}$ with positive entries. Construct two class means and a point in $\mathbb{R}^2$ whose prediction changes under such a $\mathbf{D}$.
   3. With q classes and the rule “nearest mean”, show that the region assigned to class

i is the intersection of the $q-1$ half-spaces $(\bar{\mathbf{x}}_i - \bar{\mathbf{x}}_j)\cdot\mathbf{x} + b_{ij} > 0$, $j \neq i$, and find b_{ij}. Compare with the pairwise boundaries $(\mathbf{w}_i - \mathbf{w}_j)\cdot\mathbf{x} + (b_i - b_j) = 0$ of a softmax layer: which weights and biases make the layer predict the nearest mean?

5. **Orthogonal and oblique projectors.**
   1. Let $\mathbf{P}$ satisfy $\mathbf{P}^2 = \mathbf{P}$ and $\mathbf{P}^\top = \mathbf{P}$. Show that for every $\mathbf{x}$, $\mathbf{Px}$ lies in the column space of $\mathbf{P}$ and $\mathbf{x} - \mathbf{Px}$ is orthogonal to it. Conclude with Pythagoras that $\mathbf{Px}$ is the closest point of the column space, without using an orthonormal basis.
   2. The matrix $\mathbf{P}_{\text{o}} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ satisfies $\mathbf{P}_{\text{o}}^2 = \mathbf{P}_{\text{o}}$ but is not symmetric. For $\mathbf{x} = (1, 1)$, compute $\mathbf{P}_{\text{o}}\mathbf{x}$ and the orthogonal projection of $\mathbf{x}$ onto the same column space (the first coordinate axis), and the distance of each from $\mathbf{x}$. Along which direction does $\mathbf{P}_{\text{o}}$ move points?
   3. Show that any $\mathbf{P}$ with $\mathbf{P}^2 = \mathbf{P}$ splits every $\mathbf{x}$ as $\mathbf{Px} + (\mathbf{x} - \mathbf{Px})$, with the first piece in the column space of $\mathbf{P}$ and the second in its null space $\{\mathbf{z} : \mathbf{Pz} = \mathbf{0}\}$. Show that these two subspaces are orthogonal exactly when $\mathbf{P}^\top = \mathbf{P}$.

6. **The cosine in two and three dimensions.**
   1. In $d = 2$ the angle θ between a fixed direction and a uniformly random direction is uniform on $[0, \pi]$. Derive the density of $\cos\theta$, $f(c) = 1/(\pi\sqrt{1-c^2})$ on $(-1, 1)$, and use it to confirm the two probabilities quoted in the text for $|\cos\theta| < 0.1$ and $|\cos\theta| > 0.9$.
   2. In $d = 3$ show that $\cos\theta$ is uniform on $[-1, 1]$. (Hint: by a theorem of Archimedes, the area of the part of a sphere between two parallel planes that cut it depends only on the distance between the planes.)
   3. For general $d \geq 2$ the density of $\cos\theta$ is proportional to $(1-c^2)^{(d-3)/2}$ on $(-1, 1)$. Check, numerically or with a Beta integral, that its variance is $1/d$ for $d = 4$ and $d = 10$. For which d is $|\cos\theta| < 0.1$ more likely than $|\cos\theta| > 0.9$?

7. code **Tilting the boundary.** ⊠ The mean-difference normal $\Delta = \bar{\mathbf{x}}_1 - \bar{\mathbf{x}}_0$ ignores how the images of each class spread around their mean. Let $\mathbf{C}$ be the average of the two within-class covariance matrices of the training images. Fisher's linear discriminant (Fisher 1936) chooses the normal $\mathbf{w}$ that maximizes

$$J(\mathbf{w}) = \frac{(\mathbf{w}\cdot\Delta)^2}{\mathbf{w}^\top\mathbf{C}\mathbf{w}}, \tag{33.12}$$

the squared distance between the projected class means relative to the variance of the projections within a class. For a positive definite $\mathbf{C}$ its maximizer is $\mathbf{w} \propto \mathbf{C}^{-1}\Delta$. (This problem needs the inverse of Section 33.2.2.3, positive definiteness from Section 33.3.2.2, and the condition number of Section 33.5.3.2.)

   1. Show that $J(c\mathbf{w}) = J(\mathbf{w})$ for every $c \neq 0$. For $\mathbf{C} = s^2\mathbf{I}$, use Cauchy–Schwarz to show that Δ maximizes J.
   2. Use the arrays `X0` and `X1` of t-shirts and trousers from the text, compute $\mathbf{C}$, and let $\bar{s}^2 = \operatorname{tr}(\mathbf{C})/784$ be the average pixel variance. For these images $\mathbf{C}$ has a condition number of about 10^9: some pixels are almost constant. For $\gamma \in \{10^{-4}, 10^{-2}, 1, 10^2, 10^4\}$, take the normal $\mathbf{w}_\gamma = (\mathbf{C} + \gamma\bar{s}^2\mathbf{I})^{-1}\Delta$, put the boundary through the midpoint of the two means, and compute the training and test accuracy. Then hold out $1{,}000$ images of each class from the training images, recompute $\mathbf{C}$ and the class means from the rest, and choose γ by the accuracy on

the held-out images. Redraw the histogram of signed distances with $\mathbf{w}_\gamma$ for that γ.

3. Why does $\mathbf{w}_\gamma$ approach the direction of Δ as $\gamma \to \infty$? Why does the regularizer scale with $\bar{s}^2$ rather than use a fixed multiple of $\mathbf{I}$?

8. code **Gram–Schmidt in floating point.** ▯ The Gram–Schmidt process admits two orderings of the same subtractions. Classical Gram–Schmidt (CGS) forms $\mathbf{q}_j$ by normalizing $\mathbf{a}_j - \sum_{i<j} (\mathbf{q}_i \cdot \mathbf{a}_j)\, \mathbf{q}_i$, as in the two-vector example of the text. Modified Gram–Schmidt (MGS) starts from $\mathbf{v} = \mathbf{a}_j$, replaces $\mathbf{v}$ by $\mathbf{v} - (\mathbf{q}_i \cdot \mathbf{v})\, \mathbf{q}_i$ for $i = 1, \dots, j-1$ in turn, and normalizes the result.

   1. Show that CGS and MGS produce the same vectors in exact arithmetic.
   2. Implement both and apply them to the Hilbert matrices $\mathbf{H}_n$ with entries $1/(i + j - 1)$ for $n = 4, 6, 8, 10, 12$, whose columns are close to linearly dependent. Plot the Frobenius norm $\|\mathbf{Q}^\top \mathbf{Q} - \mathbf{I}\|_F$ against n for CGS, MGS, and `np.linalg.qr`. For each method, from which n on is $\mathbf{P} = \mathbf{Q}\mathbf{Q}^\top$ no longer a projection, as measured by $\|\mathbf{P}^2 - \mathbf{P}\|_F$?
   3. Run the CGS subtraction twice for every column, the second time on the vector that the first pass left. How close to orthonormal are the resulting columns, compared with one pass of CGS and with MGS? Explain the difference by comparing the size of the components along $\mathbf{q}_1, \dots, \mathbf{q}_{j-1}$ in the vector that enters the first pass and in the vector that enters the second.

9. [extended] **Packing nearly orthogonal directions.** ▯▯ $\mathbb{R}^d$ holds at most d mutually orthogonal directions but many more whose pairwise cosines are small. Elhage et al. (2022) use this fact to explain how a layer of width d can represent more than d features in *superposition*, as nearly orthogonal directions. Their features are *sparse*: each one is active on a small fraction of the inputs, so few features have to be read out at the same time. Let $\mathbf{u}_1, \dots, \mathbf{u}_n$ be unit vectors in $\mathbb{R}^d$, stacked as the rows of $\mathbf{U} \in \mathbb{R}^{n \times d}$, and let $\mu = \max_{i \neq j} |\mathbf{u}_i \cdot \mathbf{u}_j|$ be their largest interference.

   1. Expand $\sum_{i,j} (\mathbf{u}_i \cdot \mathbf{u}_j)^2$ in coordinates and show that it equals $\sum_{k,l} \big(\sum_i u_{ik} u_{il}\big)^2$, the sum of the squared entries of the $d \times d$ matrix $\mathbf{U}^\top \mathbf{U}$. Bound this sum from below by its d diagonal terms and Cauchy–Schwarz in the form $\sum_{k=1}^{d} a_k^2 \geq \big(\sum_k a_k\big)^2 / d$, using $\sum_k (\mathbf{U}^\top \mathbf{U})_{kk} = n$. Bound the original sum from above by $n + n(n-1)\mu^2$, and derive the *Welch bound* (Welch 1974)

$$\mu^2 \geq \frac{n - d}{d\,(n - 1)}. \tag{33.13}$$

      Evaluate it for $d = 32$ with $n = 64$ and $n = 512$.
   2. A union bound over the pairs shows that independent random directions keep $\mu \leq \varepsilon$ for n of order $e^{d\varepsilon^2/4}$ (Section 36.9). For $d = 32$ and $n \in \{64, 128, 256, 512\}$, compute μ for random unit vectors and compare it with the Welch bound.
   3. Reduce μ by optimization: minimize $\sum_{i \neq j} (\mathbf{u}_i \cdot \mathbf{u}_j)^p$ for an even power such as $p = 10$ by gradient steps on $\mathbf{U}$, and renormalize every row after each step. For $d \in \{16, 32, 48, 64\}$ and $n \in \{d, 2d, 4d, 8d, 16d\}$, plot μ against n for random vectors, for optimized vectors, and for the Welch bound.

4. Use the curves to estimate, for each d, the largest n whose optimized directions keep $\mu \leq 0.2$. How does this number grow with d, compared with the rate $e^{d\varepsilon^2/4}$ of random directions? Which of your conclusions would change if some features mattered more than others, or if many features were active at once?

33.2 Matrices as Linear Maps

In Section 33.1 we read $\mathbf{Ax}$ by rows: entry i is the dot product of row i with $\mathbf{x}$, one hyperplane test per output. That reading says what a matrix computes for one input. It does not say what the matrix does to all inputs at once, and in particular whether its effect can be undone.

The question arises as soon as a layer is narrower than its input. A fully connected layer from the 784 pixels of a Fashion-MNIST image to 256 features is a 256×784 matrix. Its output holds fewer numbers than its input. By a count we make precise below, it sends at least $784 - 256 = 528$ independent input directions to zero, so different images produce the same features before any nonlinearity sees them. A square matrix may or may not lose information. The running matrix of this chapter, $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$, sends the vector $(2, -1)$ to $(0, -5)$, and another matrix sends $(0, -5)$ back to $(2, -1)$. That matrix undoes $\mathbf{A}$ on every input, so $\mathbf{A}$ loses nothing. The matrix $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$ sends both $(1, 1)$ and $(0, 3)$ to $(3, 6)$. Seeing the output $(3, 6)$, we cannot tell which of the two inputs we started from, so no matrix can bring it back.

Whether a map can be undone decides whether a layer discards information, whether a linear system has one solution or many, and, through the determinant, how a probability density changes under a change of variables (Section 36.2.3.1). We answer the question by reading a matrix through its columns. Column j is the image of the j-th coordinate axis, and these images determine the whole map. The number of linearly independent columns, the *rank*, tells how many input dimensions survive. The signed volume of the image of the unit cube, the *determinant*, tells in one number whether any dimension is lost. We work first in two dimensions, where every statement is a picture, and then in n dimensions.

Three results carry the section: rank–nullity, the equivalence of independent columns and invertibility for a square matrix, and the multiplicativity of the determinant. The construction of the determinant in n dimensions (Section 33.2.3.2) is the most technical part and can be skimmed on a first reading.

```
import numpy as np
```

33.2.1 Linear Maps

33.2.1.1 Columns as Images of the Axes

A matrix acts on infinitely many inputs, yet it is a finite table of numbers. To see how the table determines every output, take a 2×2 matrix $\mathbf{A}$ with entries p, q, r, s and apply it to $\mathbf{x} = (x, y)$. Collecting the terms that multiply x and those that multiply y gives

$$\mathbf{Ax} = \begin{bmatrix} p & q \\ r & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} px + qy \\ rx + sy \end{bmatrix} = x \begin{bmatrix} p \\ r \end{bmatrix} + y \begin{bmatrix} q \\ s \end{bmatrix} = x\,\mathbf{Ae}_1 + y\,\mathbf{Ae}_2.$$

So $\mathbf{Ax}$ is a weighted sum of the two columns of $\mathbf{A}$, with the entries of $\mathbf{x}$ as weights, and the columns are the images of the standard basis vectors $\mathbf{e}_1 = (1, 0)$ and $\mathbf{e}_2 = (0, 1)$. The images of two vectors fix the image of every vector.

For the running matrix ($p = 1$, $q = 2$, $r = -1$, $s = 3$) the axes go to $\mathbf{Ae}_1 = (1, -1)$ and $\mathbf{Ae}_2 = (2, 3)$. The input $\mathbf{v} = (2, -1)$ is $2\mathbf{e}_1 - \mathbf{e}_2$, so its image is the same combination of the columns,

$$\mathbf{Av} = 2\,(1, -1) - (2, 3) = (0, -5).$$

The row view gives the same numbers: $(1, 2) \cdot (2, -1) = 0$ and $(-1, 3) \cdot (2, -1) = -5$. Both readings compute the same product. This section uses the columns, because they show where the map sends the whole plane. Figure 33.8 draws it: the unit square spanned by $\mathbf{e}_1$ and $\mathbf{e}_2$ goes to the parallelogram spanned by the two columns, and the grid point $\mathbf{v}$ goes to the grid point $\mathbf{Av}$ of the image grid.

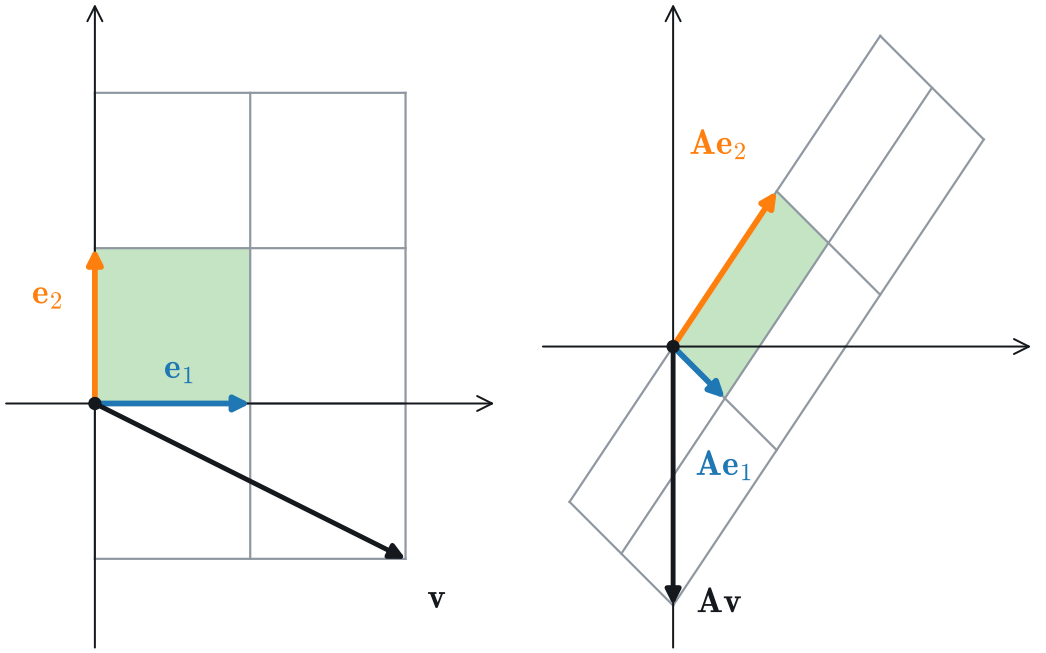


Figure 33.8: The running matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ acting on the plane. (a) The input grid, the shaded unit square, the basis vectors and the input $\mathbf{v} = (2, -1)$. (b) Its image: $\mathbf{e}_1$ goes to $(1, -1)$, $\mathbf{e}_2$ to $(2, 3)$, and $\mathbf{v}$ to $(0, -5)$. The grid is carried along: lines stay lines, the origin stays put, and equally spaced points stay equally spaced. The shaded square goes to the shaded parallelogram, whose area is the subject of the determinant. Panel (b) is drawn at one third the scale of (a).

The same holds for every size. An $m \times n$ matrix takes vectors with n entries to vectors with m entries, so it is a map from $\mathbb{R}^n$ to $\mathbb{R}^m$, and this map is *linear*:

$$\mathbf{A}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\,\mathbf{Ax} + \beta\,\mathbf{Ay} \quad \text{for all } \mathbf{x}, \mathbf{y} \in \mathbb{R}^n \text{ and } \alpha, \beta \in \mathbb{R}.$$

In words, the image of a weighted sum is the same weighted sum of the images. Writing $\mathbf{x} = \sum_j x_j \mathbf{e}_j$ gives $\mathbf{A}\mathbf{x} = \sum_j x_j \mathbf{A}\mathbf{e}_j$, and multiplying by the column vector $\mathbf{e}_j$ selects column j. Conversely, every map $f : \mathbb{R}^n \to \mathbb{R}^m$ with this linearity property is given by a matrix, the one whose column j is $f(\mathbf{e}_j)$, because $f(\mathbf{x}) = f(\sum_j x_j \mathbf{e}_j) = \sum_j x_j f(\mathbf{e}_j)$. Linear maps from $\mathbb{R}^n$ to $\mathbb{R}^m$ and $m \times n$ matrices are therefore the same objects.

Linearity also keeps the grid of Figure 33.8 a grid. The points $\mathbf{p} + t\mathbf{u}$ of a line go to $\mathbf{A}\mathbf{p} + t\,\mathbf{A}\mathbf{u}$, again a line (or a single point, when $\mathbf{A}\mathbf{u} = \mathbf{0}$), and $\mathbf{A}\mathbf{0} = \mathbf{0}$.

Nothing requires the matrix to be square. A 2×3 matrix sends $\mathbb{R}^3$ into the span of its three columns, which fills the whole output plane only when two of the columns are linearly independent. A 3×2 matrix sends the plane into $\mathbb{R}^3$, and its outputs form a plane through the origin, a line, or the origin alone, depending on its two columns. Every fully connected layer of a neural network is such a map between spaces of different dimensions, followed by a nonlinearity.

The gallery of Figure 33.9 shows six matrices acting on the plane, each as the image of the unit square, with the two columns drawn. The rotation by 90° sends $\mathbf{e}_1$ to $\mathbf{e}_2$ and $\mathbf{e}_2$ to $-\mathbf{e}_1$, so its columns are $(0, 1)$ and $(-1, 0)$. The shear keeps $\mathbf{e}_1$ fixed and tilts $\mathbf{e}_2$ to $(1, 1)$, so it slides each horizontal line sideways by its height. The last two panels lose a dimension: the whole square, and with it the whole plane, lands on a line.

A matrix records where the axes go. One familiar operation is not of this kind: the translation $\mathbf{x} \mapsto \mathbf{x} + \mathbf{c}$ with $\mathbf{c} \neq \mathbf{0}$ moves the origin, so it is not linear. A layer $\mathbf{W}\mathbf{x} + \mathbf{b}$ is a linear map followed by such a shift.

33.2.1.2 Composition

Two linear layers in a row, with no nonlinearity between them, form a single map. Which matrix represents it? Take the rotation by 90° and the shear of Figure 33.9, which we call $\mathbf{T}$ because it tilts the vertical axis:

$$\mathbf{R}_{90} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{T} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

Both return throughout the chapter. Shearing first sends $\mathbf{e}_1$ to $(1, 0)$, and rotating that gives $(0, 1)$; shearing sends $\mathbf{e}_2$ to $(1, 1)$, and rotating that gives $(-1, 1)$. The composite map "shear, then rotate" therefore has the columns $(0, 1)$ and $(-1, 1)$. In the other order, rotating sends $\mathbf{e}_1$ to $(0, 1)$, which the shear moves to $(1, 1)$, and $\mathbf{e}_2$ to $(-1, 0)$, which the shear leaves alone. The two composites are

$$\text{shear, then rotate: } \begin{bmatrix} 0 & -1 \\ 1 & 1 \end{bmatrix}, \qquad \text{rotate, then shear: } \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}.$$

The two orders give different maps, as Figure 33.10 shows.

The matrix product computes such composites. For any two matrices $\mathbf{A}$ and $\mathbf{B}$ whose sizes fit, apply $\mathbf{B}$ to $\mathbf{v}$, then $\mathbf{A}$ to the result, and write out entry i:

$$\big(\mathbf{A}(\mathbf{B}\mathbf{v})\big)_i = \sum_j a_{ij} (\mathbf{B}\mathbf{v})_j = \sum_j \sum_k a_{ij} b_{jk} v_k = \sum_k \Big(\sum_j a_{ij} b_{jk} \Big) v_k = \big((\mathbf{A}\mathbf{B})\mathbf{v}\big)_i. \quad (33.14)$$

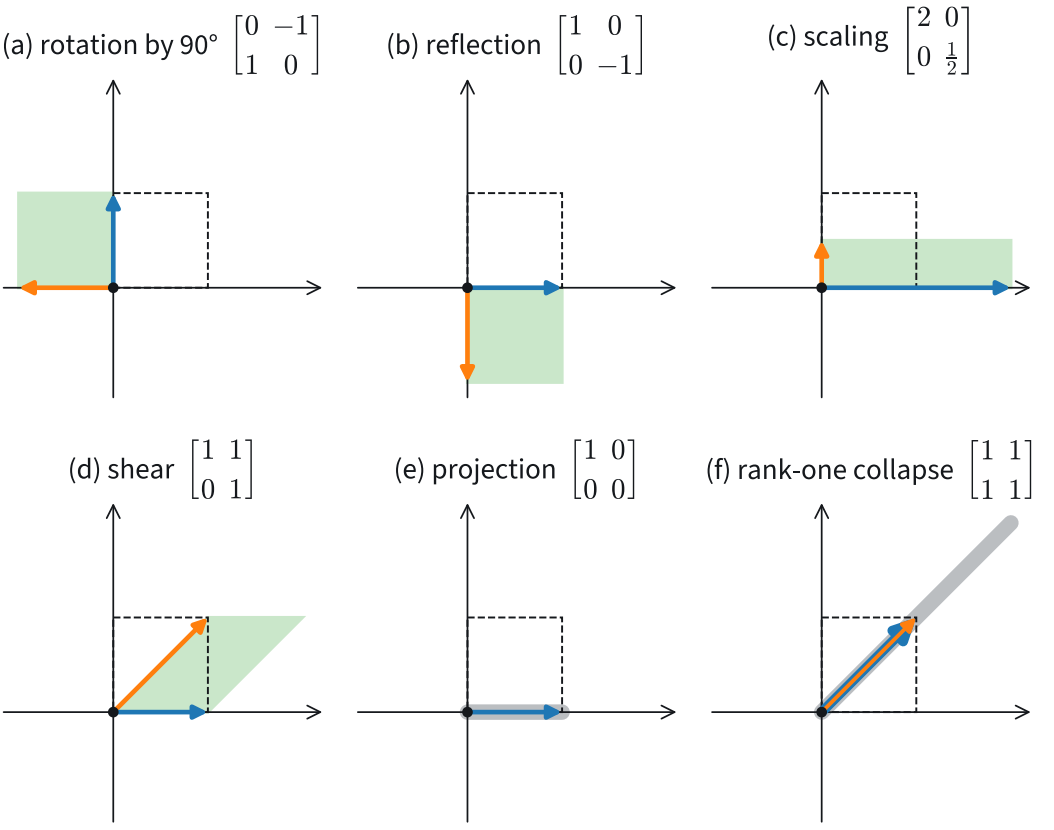


Figure 33.9: Six things a 2×2 matrix can do to the unit square (dashed). In each panel the columns, drawn as arrows, are the images of $\mathbf{e}_1$ (blue) and $\mathbf{e}_2$ (orange), and the shaded region is the image of the square. Panels (e) and (f) send the plane onto a line; in (f) both columns are $(1, 1)$.

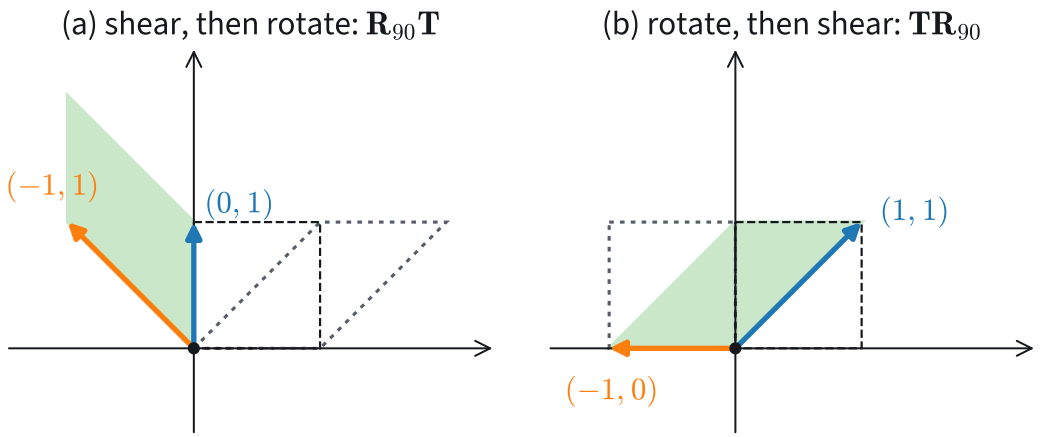


Figure 33.10: The order of composition matters. (a) The shear $\mathbf{T}$ (dotted outline: the square after the shear), then the rotation $\mathbf{R}_{90}$; the product is $\mathbf{R}_{90}\mathbf{T}$. (b) The rotation (dotted: the square after the rotation), then the shear; the product is $\mathbf{T}\mathbf{R}_{90}$. The labels give the images of $\mathbf{e}_1$ (blue) and $\mathbf{e}_2$ (orange). A product of matrices is read from right to left.

Applying the two maps in turn is the same as applying the single matrix $\mathbf{AB}$, whose entry (i, k) is row i of $\mathbf{A}$ times column k of $\mathbf{B}$. The product $\mathbf{AB}$ means "$\mathbf{B}$ first, then $\mathbf{A}$", read from right to left like $f(g(x))$. Taking $\mathbf{v} = \mathbf{e}_j$ in Equation 33.14 shows that column j of $\mathbf{AB}$ is $\mathbf{A}\mathbf{b}_j$, the image under $\mathbf{A}$ of column j of $\mathbf{B}$. We use this fact for the determinant.

Composition explains the multiplication rule. The product is row-times-column because that is what the composite of two linear maps computes. It is associative, $(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$, because composing functions is associative: both sides apply $\mathbf{C}$, then $\mathbf{B}$, then $\mathbf{A}$. And it is not commutative, because composing two functions in the other order generally gives another function. In code the product is the operator `@`. For the rotation and the shear, which the cell stores as `R90` and `S`, it returns the two composites that we found by following the axes.

```
R90 = np.array([[0.0, -1.0], [1.0, 0.0]])  # rotation by 90 degrees
S = np.array([[1.0, 1.0], [0.0, 1.0]])     # shear
print(R90 @ S)
print(S @ R90)
```

```
[[ 0. -1.]
 [ 1.  1.]]
[[ 1. -1.]
 [ 1.  0.]]
```

Some pairs of matrices do commute. Two rotations of the plane commute, since rotating by ϕ and then by ψ rotates by $\phi + \psi$ in either order. Two diagonal matrices commute, since each scales the axes independently. Exercise 5 finds every matrix that commutes with $\mathbf{R}_{90}$.

33.2.1.3 Orthogonal Matrices

Some maps move space rigidly: every length and every angle comes out unchanged. Which matrices do this? The rotation by an angle ϕ sends $\mathbf{e}_1$ to $(\cos\phi, \sin\phi)$ and $\mathbf{e}_2$ to

$(-\sin\phi, \cos\phi)$, so its matrix is

$$\mathbf{R}_\phi = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}.$$

Its columns have length $\sqrt{\cos^2\phi + \sin^2\phi} = 1$ and dot product $-\cos\phi\sin\phi + \sin\phi\cos\phi = 0$. They are orthonormal, which Section 33.1 wrote as $\mathbf{R}_\phi^\top \mathbf{R}_\phi = \mathbf{I}$. With $\phi = 90^\circ$ we get $\mathbf{R}_{90}$, which sends $(3, 2)$ to $(-2, 3)$; both have length $\sqrt{13}$. The reflection $\mathrm{diag}(1, -1)$ in the x-axis has orthonormal columns as well.

A square matrix $\mathbf{Q}$ is *orthogonal* when its columns are orthonormal,

$$\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}.$$

Columns that are only perpendicular are not enough: $\mathrm{diag}(1, 10)$ has perpendicular columns of lengths 1 and 10, and it changes the angle between $(1, 1)$ and $(1, -1)$ from 90° to $\arccos(-99/101) \approx 169^\circ$ (1). The name is also reserved for the square case. A rectangular $n \times k$ matrix with orthonormal columns, $k < n$, preserves lengths, but it maps $\mathbb{R}^k$ into the larger space $\mathbb{R}^n$ and has no two-sided inverse (Section 33.2.2.3).

An orthogonal map preserves every dot product,

$$(\mathbf{Q}\mathbf{x}) \cdot (\mathbf{Q}\mathbf{y}) = (\mathbf{Q}\mathbf{x})^\top (\mathbf{Q}\mathbf{y}) = \mathbf{x}^\top \mathbf{Q}^\top \mathbf{Q}\, \mathbf{y} = \mathbf{x}^\top \mathbf{y} = \mathbf{x} \cdot \mathbf{y}.$$

Taking $\mathbf{y} = \mathbf{x}$ shows $\|\mathbf{Q}\mathbf{x}\| = \|\mathbf{x}\|$, and since the angle between two vectors is computed from their dot product and their lengths, angles are preserved too. Rotating every embedding by the same orthogonal matrix therefore leaves all their cosine similarities unchanged.

The condition $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ says that $\mathbf{Q}^\top$ undoes $\mathbf{Q}$ when applied after it. For a square $\mathbf{Q}$, it also undoes $\mathbf{Q}$ when applied before it, $\mathbf{Q}\mathbf{Q}^\top = \mathbf{I}$, so that the rows are orthonormal too and $\mathbf{Q}^{-1} = \mathbf{Q}^\top$. The definition does not make this obvious. It is a corollary of the statement about square matrices that we prove in Section 33.2.2.3.

In the plane, rotations and reflections are the only orthogonal matrices. The first column of an orthogonal 2×2 matrix is a unit vector $(\cos\phi, \sin\phi)$, and the second is one of the two unit vectors perpendicular to it, $\pm(-\sin\phi, \cos\phi)$. The plus sign gives the rotation $\mathbf{R}_\phi$. The minus sign gives $\mathbf{R}_\phi\, \mathrm{diag}(1, -1)$, the reflection in the x-axis followed by a rotation, which is a reflection in a line through the origin. The determinant will tell the two kinds apart by its sign (Section 33.2.3.3).

33.2.1.4 Tensors and Einstein Summation

Attention in a transformer compares every query with every key, for every head and for every sequence in a batch. With a batch index b, a head index h, a query position i, a key position t, and a feature index j that runs over the width of a head, the scores are

$$s_{bhit} = \sum_j q_{bhij}\, k_{bhtj},$$

where q_{bhij} and k_{bhtj} are the entries of the query and key tensors (arrays with four axes, Section 1.1). Each score is a dot product along j, computed for every combination of the other four indices. Matrix notation has no name for this four-index object, and writing it as "a batched matrix product with a transpose" hides which axis is summed.

Every product we have met so far follows one pattern. The dot product, the matrix–vector product, the matrix product and the *trace* $\operatorname{tr}(\mathbf{A})$, the sum of the diagonal entries of a square matrix, are

$$\mathbf{v}\cdot\mathbf{w}=\sum_i v_i w_i, \qquad (\mathbf{A}\mathbf{v})_i=\sum_j a_{ij}v_j, \qquad (\mathbf{A}\mathbf{B})_{ik}=\sum_j a_{ij}b_{jk}, \qquad \operatorname{tr}(\mathbf{A})=\sum_i a_{ii}.$$

The trace is new here. It returns in Section 33.3 as the sum of the eigenvalues. Each of the four multiplies entries and sums over an index that appears twice. *Einstein notation* makes this the whole definition: write the indexed factors, drop the summation sign, and sum over every index that is repeated in a term. Thus $\mathbf{v}\cdot\mathbf{w}=v_i w_i$ and $(\mathbf{A}\mathbf{B})_{ik}=a_{ij}b_{jk}$. The attention scores show the limit of this classical rule: the indices b and h also appear in both factors, but they are kept, not summed. The library function `einsum` therefore names the kept indices explicitly. Its argument is a string of input indices, an arrow, and the output indices, and every index that does not appear after the arrow is summed. The four products become four strings.

```
A = np.array([[1.0, 2.0], [-1.0, 3.0]])    # the running matrix
v = np.array([2.0, -1.0])
(np.einsum('i,i->', v, v),         # dot product: v.v
 np.einsum('ij,j->i', A, v),       # matrix-vector product: Av
 np.einsum('ij,jk->ik', R90, S),   # matrix product: R90 S
 np.einsum('ii->', A))             # trace: tr(A)
```

```
(np.float64(5.0),
 array([ 0., -5.]),
 array([[ 0., -1.],
        [ 1.,  1.]]),
 np.float64(4.0))
```

The values are those computed by hand in the earlier subsections, and $\operatorname{tr}\mathbf{A}=1+3=4$. The attention scores take one line in the same notation. The cell compares them with the matrix product of the queries and the transposed keys and counts the multiply–adds.

```
rng = np.random.default_rng()
queries = rng.standard_normal((2, 3, 4, 8))   # (batch, heads, queries, width)
keys = rng.standard_normal((2, 3, 5, 8))      # (batch, heads, keys, width)
scores = np.einsum('bhij,bhtj->bhit', queries, keys)
(scores.shape, np.allclose(scores, queries @ keys.swapaxes(-1, -2)),
 scores.size * queries.shape[-1])             # multiply-adds
```

```
((2, 3, 4, 5), True, 960)
```

The scores have the shape $(2,3,4,5)$, one 4×5 table per sequence and head. The two computations agree, and each takes $2\cdot3\cdot4\cdot5\cdot8=960$ multiply–adds, the product of the sizes of all five indices.

Index notation also gives one-line proofs. For $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{n\times m}$, both products $\mathbf{AB}$ and $\mathbf{BA}$ are square, and their traces are

$$\operatorname{tr}(\mathbf{AB}) = a_{ij}b_{ji} = b_{ji}a_{ij} = \operatorname{tr}(\mathbf{BA}). \tag{33.15}$$

The middle equality only swaps two scalar factors, and the indices stay where they are. So the trace of a product does not depend on the order of the two factors, even when $\mathbf{AB} \neq \mathbf{BA}$, and even when the two products have different sizes. One consequence will matter for eigenvalues. If $\mathbf{V}$ has an inverse $\mathbf{V}^{-1}$, a matrix that undoes it (Section 33.2.2.3), then $\operatorname{tr}(\mathbf{VAV}^{-1}) = \operatorname{tr}(\mathbf{AV}^{-1}\mathbf{V}) = \operatorname{tr}(\mathbf{A})$. The matrix $\mathbf{VAV}^{-1}$ is the map $\mathbf{A}$ written in other coordinates, so the trace belongs to the map and not to the coordinates in which we write it down. The determinant will turn out to share this property (Section 33.2.3.3).

For three or more factors, `einsum` fixes the result but not the order in which the pairwise products are formed, and the order matters: for $n \times n$ matrices, $(\mathbf{AB})\mathbf{v}$ costs about n^3 multiply–adds and $\mathbf{A}(\mathbf{Bv})$ only $2n^2$.

33.2.2 Rank and Invertibility

33.2.2.1 Column Space and Null Space

Which outputs can a matrix produce, and which inputs does it erase? The matrix

$$\mathbf{B} = \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$$

has the columns $\mathbf{b}_1 = (2, 4)$ and $\mathbf{b}_2 = (1, 2)$, and $\mathbf{b}_1 = 2\mathbf{b}_2$. Every output is a combination of the columns, and such a combination collapses to a multiple of one column,

$$a_1\mathbf{b}_1 + a_2\mathbf{b}_2 = (2a_1 + a_2)\,\mathbf{b}_2.$$

All outputs of $\mathbf{B}$ therefore lie on the line through $(1, 2)$, the line $y = 2x$. Distinct inputs collide on that line: $\mathbf{B}(1, 1) = (3, 6)$ and $\mathbf{B}(0, 3) = (3, 6)$. By linearity their difference $(1, -2)$ goes to the difference of the outputs, $\mathbf{B}(1, -2) = \mathbf{0}$. The row view shows why: both rows, $(2, 1)$ and $(4, 2)$, are perpendicular to $(1, -2)$, so both hyperplane tests return zero. Figure 33.11 shows the two lines.

Two subspaces attach to every $m \times n$ matrix $\mathbf{A}$. The *column space* $\operatorname{col}\mathbf{A}$ is the span of the columns. Since $\mathbf{Ax}$ is the combination of the columns with weights $x_1, \dots, x_n$, the column space is the set of all outputs. The *null space*, also called the *kernel* and written $\ker\mathbf{A}$, is the set of inputs sent to zero,

$$\ker\mathbf{A} = \{\mathbf{x} \in \mathbb{R}^n : \mathbf{Ax} = \mathbf{0}\}.$$

It is a subspace: if $\mathbf{Ax} = \mathbf{Ay} = \mathbf{0}$, then $\mathbf{A}(\alpha\mathbf{x} + \beta\mathbf{y}) = \alpha\mathbf{Ax} + \beta\mathbf{Ay} = \mathbf{0}$. For $\mathbf{B}$ the column space is the line $y = 2x$ and the null space is the line through $(1, -2)$. The null space is a line of inputs and the column space a line of outputs; that both are lines in $\mathbb{R}^2$ is an accident of the square shape.

Three general facts follow the example.

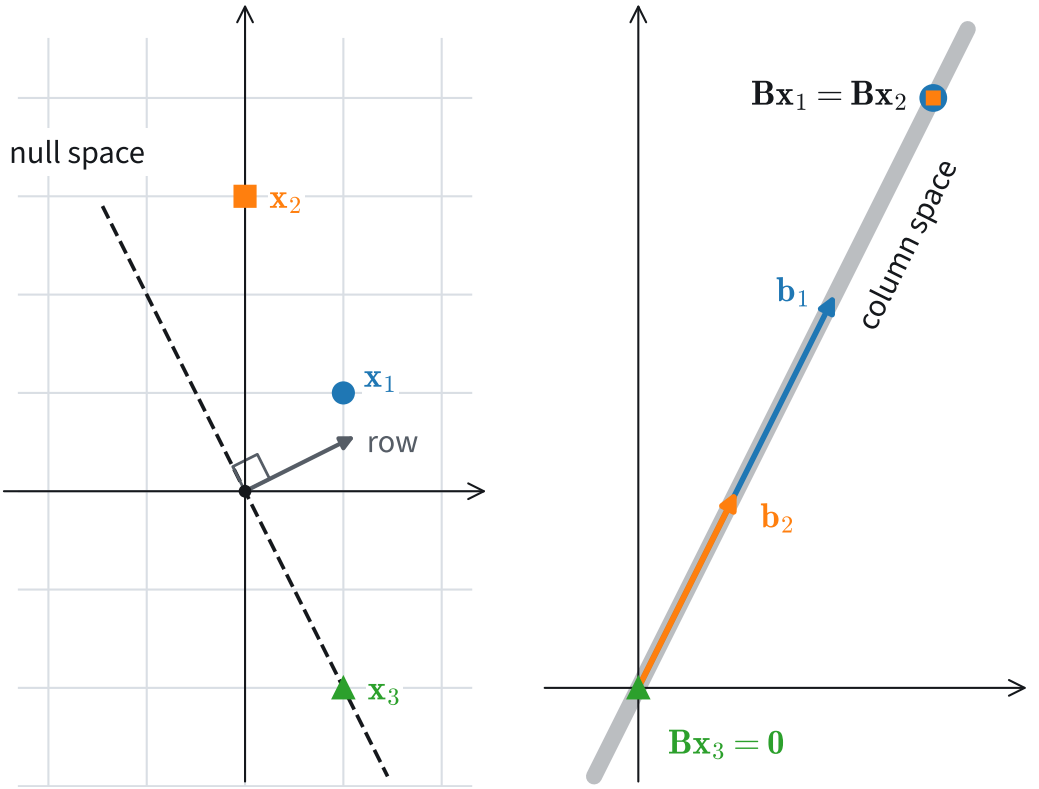


Figure 33.11: The matrix $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$ collapsing the plane. (a) The input plane, with the null-space line through $(1, -2)$ dashed and perpendicular to the row direction $(2, 1)$, and three marked inputs: $\mathbf{x}_1 = (1, 1)$, $\mathbf{x}_2 = (0, 3)$, and $\mathbf{x}_3 = (1, -2)$ on the null-space line. (b) The image plane, at the same scale: every output lies on the column space, the line $y = 2x$ spanned by the columns $\mathbf{b}_1$ and $\mathbf{b}_2$; $\mathbf{x}_1$ and $\mathbf{x}_2$ land on the same point $(3, 6)$, and $\mathbf{x}_3$ lands on the origin. Because distinct inputs have the same image, no inverse exists.

1. *The null space consists of the inputs perpendicular to every row.* This is the row view: $(\mathbf{A}\mathbf{x})_i$ is the dot product of row i with $\mathbf{x}$, and all of these vanish if and only if $\mathbf{x}$ is perpendicular to all rows.
2. *The columns are linearly dependent if and only if the null space contains a nonzero vector.* A dependence $\sum_j x_j \mathbf{a}_j = \mathbf{0}$ among the columns $\mathbf{a}_j$ of $\mathbf{A}$, with coefficients not all zero, says $\mathbf{A}\mathbf{x} = \mathbf{0}$, so its coefficients form a nonzero null vector, and the converse reads the same equation backwards. For $\mathbf{B}$, the dependence $\mathbf{b}_1 - 2\mathbf{b}_2 = \mathbf{0}$ has the coefficients $(1, -2)$.
3. *Two inputs have the same output if and only if they differ by a null vector,* because $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{y}$ means $\mathbf{A}(\mathbf{x} - \mathbf{y}) = \mathbf{0}$. The inputs that $\mathbf{B}$ sends to $(3, 6)$ form the line $(1, 1) + t(1, -2)$, which contains $(0, 3)$ at $t = -1$. A nonzero null vector therefore means that no inverse can exist.

33.2.2.2 Rank

How many input dimensions survive the map? The *rank* of a matrix is the dimension of its column space, equivalently the largest number of linearly independent columns:

$$\operatorname{rank} \mathbf{A} = \dim \operatorname{col} \mathbf{A}.$$

The matrix $\mathbf{B}$ has rank one: its two columns are dependent, and each column is nonzero. As a larger example, consider

$$\mathbf{C} = \begin{bmatrix} 1 & 3 & 0 & -1 & 0 \\ -1 & 0 & 1 & 1 & -1 \\ 0 & 3 & 1 & 0 & -1 \\ 2 & 3 & -1 & -2 & 1 \end{bmatrix},$$

which has rank two: the first two columns are linearly independent, while each of the $\binom{5}{3} = 10$ collections of three columns is linearly dependent (2).

The dimensions that survive and the dimensions sent to zero add up to the number of input dimensions.

Proposition (rank–nullity). *For an $m \times n$ matrix $\mathbf{A}$,*

$$\operatorname{rank} \mathbf{A} + \dim \ker \mathbf{A} = n. \tag{33.16}$$

Proof. Choose a basis $\mathbf{w}_1, \dots, \mathbf{w}_k$ of the null space and extend it to a basis $\mathbf{w}_1, \dots, \mathbf{w}_k, \mathbf{u}_1, \dots, \mathbf{u}_{n-k}$ of $\mathbb{R}^n$. (That an independent list extends to a basis is, like the equal size of all bases, a fact we use without proof.) The images $\mathbf{A}\mathbf{u}_1, \dots, \mathbf{A}\mathbf{u}_{n-k}$ span the column space, since every input is a combination of the basis vectors and the $\mathbf{w}_i$ contribute nothing. They are also independent: if $\sum_i c_i \mathbf{A}\mathbf{u}_i = \mathbf{0}$, then $\sum_i c_i \mathbf{u}_i$ lies in the null space, hence is a combination of the $\mathbf{w}_j$, which is impossible within a basis unless every $c_i = 0$. So the column space has dimension $n - k$. $\blacksquare$

For $\mathbf{B}$ the count is $1 + 1 = 2$. For the 256×784 layer of the introduction, the rank is at most 256, the number of output coordinates, so the null space has dimension at least $784 - 256 = 528$. For $\mathbf{C}$ the null space must have dimension $5 - 2 = 3$, so three independent null vectors are a basis of it. The cell tests the candidates $(3, -1, 3, 0, 0)$, $(1, 0, 0, 1, 0)$ and

$(0, 0, 1, 0, 1)$: it multiplies them by **C** and computes the rank of the matrix `Z` that holds them as columns.

```
B = np.array([[2.0, 1.0], [4.0, 2.0]])
C = np.array([[1, 3, 0, -1, 0], [-1, 0, 1, 1, -1],
              [0, 3, 1, 0, -1], [2, 3, -1, -2, 1]], dtype=float)
Z = np.array([[3, -1, 3, 0, 0], [1, 0, 0, 1, 0],
              [0, 0, 1, 0, 1]], dtype=float).T   # candidate null vectors
rank_C, rank_Z = np.linalg.matrix_rank(C), np.linalg.matrix_rank(Z)
np.linalg.matrix_rank(B), rank_C, np.abs(C @ Z).max(), rank_Z, rank_C + rank_Z
```

```
(np.int64(1), np.int64(2), np.float64(0.0), np.int64(3), np.int64(5))
```

The ranks of **B** and **C** are 1 and 2, and the three candidates are sent to zero and are independent (rank 3).

The same matrix can be read by rows. Its *row rank*, the largest number of linearly independent rows, counts the independent hyperplane tests the matrix performs. For **C**, row 3 is row 1 plus row 2 and row 4 is row 1 minus row 2, so the row rank is two, the same as the rank. This is no coincidence.

Proposition (row rank equals column rank). *For every matrix* **A***, the largest number of linearly independent rows equals the largest number of linearly independent columns.*

Proof. Let $r = \operatorname{rank} \mathbf{A}$ and put a basis of $\operatorname{col} \mathbf{A}$ into the columns of an $m \times r$ matrix **G**. We factor **A** through this basis. Every column of **A** is a combination of the columns of **G**, and writing the coefficients for column j into column j of an $r \times n$ matrix **H** gives

$$\mathbf{A} = \mathbf{G}\mathbf{H}. \tag{33.17}$$

Now read the same product by rows. Row i of **A** is $\sum_k g_{ik}\,(\text{row } k \text{ of } \mathbf{H})$, so all rows of **A** lie in the span of the r rows of **H**, and the row rank is at most r. Applying the same argument to $\mathbf{A}^{\top}$, whose columns are the rows of **A**, shows that the column rank is at most the row rank. ∎

The factorization says that a matrix of rank r first computes r numbers from its input (the map **H**) and then spreads them over the m outputs (the map **G**). With r far below m and n it stores $r(m + n)$ numbers in place of mn, the idea behind the low-rank updates of Section 33.5.4.1.

Finding the rank by checking subsets of columns takes exponentially long. By hand one uses Gaussian elimination, which this chapter does not teach. The routine `np.linalg.matrix_rank` instead counts the *singular values* (Section 33.5) that exceed a small threshold. It needs a threshold because rank is not a continuous function of the entries (Section 33.2.3.5).

33.2.2.3 Inverses

Given an output $\mathbf{A}\mathbf{x}$, can we recover the input $\mathbf{x}$? An *inverse* of a square matrix **A** is a matrix $\mathbf{A}^{-1}$ that undoes it from both sides,

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I},$$

so that $\mathbf{x} = \mathbf{A}^{-1}(\mathbf{A}\mathbf{x})$. For a 2×2 matrix with entries p, q, r, s and $ps - qr \neq 0$, the inverse is

$$\mathbf{A}^{-1} = \frac{1}{ps - qr} \begin{bmatrix} s & -q \\ -r & p \end{bmatrix}. \tag{33.18}$$

Multiplying out confirms the formula: the matrix $\left(\begin{smallmatrix} s & -q \\ -r & p \end{smallmatrix}\right)$ times $\mathbf{A}$, in either order, is $(ps - qr)\,\mathbf{I}$, and dividing by $ps - qr$ leaves $\mathbf{I}$. The hypothesis cannot be dropped. If $ps - qr = 0$, the product is the zero matrix, and multiplying it by an inverse of $\mathbf{A}$ would give $\left(\begin{smallmatrix} s & -q \\ -r & p \end{smallmatrix}\right) = \mathbf{0}$, that is, $\mathbf{A} = \mathbf{0}$, which has no inverse. The number $ps - qr$ returns as an area in Section 33.2.3.1. For the running matrix, $ps - qr = 3 + 2 = 5$, and

$$\mathbf{A}^{-1} = \frac{1}{5} \begin{bmatrix} 3 & -2 \\ 1 & 1 \end{bmatrix}, \qquad \mathbf{A}^{-1}(0, -5) = \tfrac{1}{5}\,(10, -5) = (2, -1).$$

The inverse undoes the worked example of Section 33.2.1.1.

A left inverse and a right inverse of the same matrix always agree, so the inverse is unique. If $\mathbf{L}\mathbf{A} = \mathbf{I}$ and $\mathbf{A}\mathbf{X} = \mathbf{I}$, then

$$\mathbf{L} = \mathbf{L}(\mathbf{A}\mathbf{X}) = (\mathbf{L}\mathbf{A})\mathbf{X} = \mathbf{X},$$

with associativity in the middle step.

When does an inverse exist? For a square matrix the answer needs only the null space and rank–nullity, and the rest of the chapter uses it more than any other result of this section.

Proposition (invertible matrices). *For an $n \times n$ matrix $\mathbf{A}$, the following are equivalent:* (i) $\mathbf{A}$ *is invertible;* (ii) *the columns of $\mathbf{A}$ are linearly independent;* (iii) $\ker \mathbf{A} = \{\mathbf{0}\}$*;* (iv) $\operatorname{rank} \mathbf{A} = n$.

Proof. Statements (ii) and (iii) are equivalent by fact 2 of Section 33.2.2.1, and (iii) and (iv) by rank–nullity. If $\mathbf{A}$ is invertible and $\mathbf{A}\mathbf{x} = \mathbf{0}$, then $\mathbf{x} = \mathbf{A}^{-1}\mathbf{A}\mathbf{x} = \mathbf{0}$, so (i) implies (iii). The converse takes more work, because the inverse has to be built. By (iv) the column space is an n-dimensional subspace of $\mathbb{R}^n$. Such a subspace is all of $\mathbb{R}^n$: a vector outside it, appended to its basis, would give $n+1$ independent vectors, and their extension to a basis of $\mathbb{R}^n$ would have more than n elements. So every target is hit. By (iii) and fact 3, no two inputs collide. Let column j of a matrix $\mathbf{X}$ be the unique solution of $\mathbf{A}\mathbf{x} = \mathbf{e}_j$. Then $\mathbf{A}\mathbf{X} = \mathbf{I}$ by construction. For any $\mathbf{x}$, $\mathbf{A}(\mathbf{X}\mathbf{A}\mathbf{x}) = (\mathbf{A}\mathbf{X})\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}$, and since no two inputs collide, $\mathbf{X}\mathbf{A}\mathbf{x} = \mathbf{x}$. So $\mathbf{X}\mathbf{A} = \mathbf{I}$ as well, and $\mathbf{X} = \mathbf{A}^{-1}$. ∎

The claim left open in Section 33.2.1.3 follows. If $\mathbf{Q}\mathbf{c} = \mathbf{0}$, then $\mathbf{c} = \mathbf{Q}^{\top}\mathbf{Q}\mathbf{c} = \mathbf{0}$, so an orthogonal $\mathbf{Q}$ is invertible. Its left inverse $\mathbf{Q}^{\top}$ is then its inverse, and $\mathbf{Q}\mathbf{Q}^{\top} = \mathbf{I}$.

A square matrix without an inverse is called *singular*; $\mathbf{B}$ is one. A matrix that is not square has no two-sided inverse at all. The rank of an $m \times n$ matrix is at most $\min(m, n)$. If $m > n$, the column space has dimension at most $n < m$, so the matrix is not *onto*: some targets in $\mathbb{R}^m$ are never hit. If $n > m$, rank–nullity leaves a null space of dimension at least $n - m > 0$, so the matrix is not *one-to-one*: some distinct inputs give the same output. The inverse runs the map backwards, and it exists when no two inputs collide and every target is hit. For a square matrix, the proposition says that either condition implies the other.

33.2.2.4 Solving Linear Systems in Practice

The inverse is a tool for reasoning, and in practice we rarely compute it. To solve $\mathbf{Ax} = \mathbf{b}$, prefer `np.linalg.solve(A, b)` to `np.linalg.inv(A) @ b`. One reason is cost. `solve` factors $\mathbf{A}$ once into triangular factors (an *LU factorization*, about $n^3/3$ multiply–adds) and then solves two triangular systems. Forming the inverse takes about n^3 multiply–adds before the product with $\mathbf{b}$ even starts. The other is accuracy. `solve` is *backward stable*: its answer is the exact solution of a system whose entries differ from those of $\mathbf{A}$ and $\mathbf{b}$ only by rounding. Its residual $\|\mathbf{Ax}-\mathbf{b}\|$ is therefore of the order of machine precision times $\|\mathbf{A}\|\,\|\mathbf{x}\|$ (with the spectral norm of Section 1.3.4). The product `inv(A) @ b` carries no such guarantee, and its residual can be larger by a factor up to the *condition number* $\kappa(\mathbf{A})$, a measure of how close $\mathbf{A}$ is to a singular matrix (Section 33.5.3.2).

How close a matrix is to singular is a property of the problem, not of the method. Both methods can lose up to about $\log_{10}\kappa(\mathbf{A})$ of the sixteen decimal digits of double precision in the solution itself. This is a bound for the worst case, not a prediction for a given right-hand side. How close does an example come to it? The *Hilbert matrix*, with entries $h_{ij} = 1/(i+j-1)$, is a classical example of a nearly singular matrix. We set $\mathbf{x} = \mathbf{1}$, compute the right-hand side $\mathbf{b} = \mathbf{H1}$ for the 10×10 Hilbert matrix $\mathbf{H}$, and solve for $\mathbf{x}$ again both ways.

```
H = 1 / (np.arange(10)[:, None] + np.arange(10) + 1)   # Hilbert matrix
x_true = np.ones(10)
b = H @ x_true
for x in (np.linalg.solve(H, b), np.linalg.inv(H) @ b):
    print(f'residual {np.linalg.norm(H @ x - b):.1e}, '
          f'error {np.linalg.norm(x - x_true):.1e}')
np.linalg.cond(H)
```

```
residual 5.7e-16, error 2.7e-04
residual 4.8e-05, error 9.0e-03
```

```
np.float64(16024416987428.36)
```

The last output is $\kappa(\mathbf{H}) \approx 1.6 \cdot 10^{13}$, so up to thirteen digits may be lost. The residual of `solve` is near machine precision, and that of `inv` is some eleven orders of magnitude larger, within the factor κ. A small residual does not mean a good solution, however. Both answers are wrong after the first few digits: either route loses twelve or more of the sixteen digits, close to what the bound allows, and `inv` is the worse of the two by more than an order of magnitude.

Sparse matrices give a further reason to avoid the inverse. A matrix with a million rows and columns but only five million nonzero entries is cheap to store, yet its inverse is typically dense, with about 10^{12} nonzero entries.

33.2.3 The Determinant

33.2.3.1 Area of a Parallelogram

By how much does $\mathbf{A}$ scale areas? In Figure 33.8 the unit square, with edges $\mathbf{e}_1$ and $\mathbf{e}_2$ and area one, goes to the parallelogram with edges $\mathbf{a}_1 = (1,-1)$ and $\mathbf{a}_2 = (2,3)$ and corners $(0,0)$, $(1,-1)$, $(3,2)$ and $(2,3)$. The parallelogram fits in the box $[0,3] \times [-1,3]$ of area 12. The part of the box outside it consists of four right triangles, with legs 1 and 1, 2 and 3, 1

and 1, and 2 and 3, of total area $\frac{1}{2}+3+\frac{1}{2}+3=7$. The parallelogram has area $12-7=5$ (Figure 33.12, panel (a)). The matrix multiplies the area of the unit square by five.

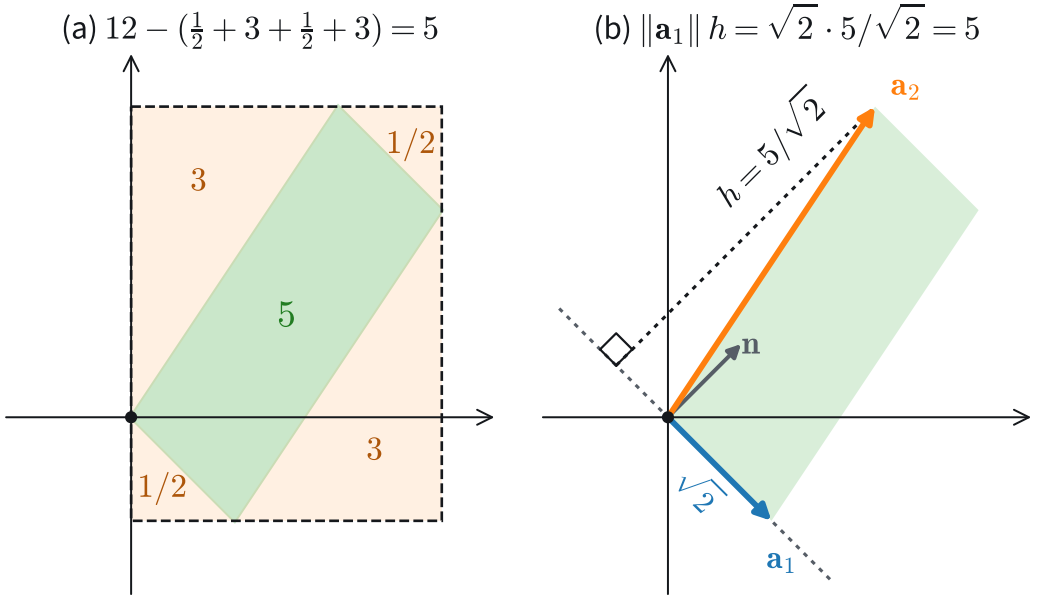


Figure 33.12: The image of the unit square under $\mathbf{A}$ has area 5. (a) The bounding box $[0,3]\times[-1,3]$, of area 12, minus four right triangles of total area 7. (b) Base times height: the base $\mathbf{a}_1=(1,-1)$ has length $\sqrt{2}$, the unit normal $\mathbf{n}=(1,1)/\sqrt{2}$ is perpendicular to it, and the height of the corner $\mathbf{a}_2=(2,3)$ above the line through $\mathbf{a}_1$ is $\mathbf{a}_2\cdot\mathbf{n}=5/\sqrt{2}$.

Counting triangles needs a new picture for every arrangement of the corners. Base times height works for all of them at once. Let the parallelogram have the edges $\mathbf{u}=(p,r)$ and $\mathbf{w}=(q,s)$, the two columns of a 2×2 matrix with entries p,q,r,s. Its base has length $\|\mathbf{u}\|$. The vector $(-r,p)$ is $\mathbf{u}$ rotated by 90°, that is, $\mathbf{R}_{90}\mathbf{u}$, so

$$\mathbf{n}=\frac{(-r,p)}{\|\mathbf{u}\|}$$

is a unit vector perpendicular to the base. By Section 33.1, the component of $\mathbf{w}$ along a unit vector $\mathbf{n}$ is $\mathbf{w}\cdot\mathbf{n}$. Here this component is the height of the parallelogram above its base, with a sign: positive when $\mathbf{w}$ lies on the counter-clockwise side of $\mathbf{u}$, negative on the clockwise side. Base times signed height is

$$\|\mathbf{u}\|\,(\mathbf{w}\cdot\mathbf{n})=\|\mathbf{u}\|\,\frac{-qr+ps}{\|\mathbf{u}\|}=ps-qr.$$

The lengths cancel, and what remains is a polynomial in the four entries. No case distinction on signs was needed. For the running matrix, $\|\mathbf{u}\|=\sqrt{2}$, $\mathbf{n}=(1,1)/\sqrt{2}$, and $\mathbf{w}\cdot\mathbf{n}=5/\sqrt{2}$, so the signed area is 5, as the count gave (panel (b)). It is positive because $\mathbf{a}_2$ lies counter-clockwise from $\mathbf{a}_1$.

The *determinant* of a 2×2 matrix is this signed area of the image of the unit square:

$$\det \begin{bmatrix} p & q \\ r & s \end{bmatrix} = ps - qr. \tag{33.19}$$

Its absolute value is the factor by which the matrix scales area, and its sign records whether the images of $\mathbf{e}_1$ and $\mathbf{e}_2$ keep their counter-clockwise order. For $\mathbf{A}$, $1 \cdot 3 - 2 \cdot (-1) = 5$. Exchanging the two columns of $\mathbf{A}$ gives $\left(\begin{smallmatrix} 2 & 1 \\ 3 & -1 \end{smallmatrix}\right)$, whose image is the same parallelogram with its edges in the other order, and the determinant is $-2-3 = -5$. For $\mathbf{B}$ it is $2 \cdot 2 - 1 \cdot 4 = 0$: the square goes to the segment from $\mathbf{0}$ to $(3, 6)$, which has no area.

The three cases are drawn in Figure 33.13: a positive determinant keeps the orientation, a negative one reverses it, and a zero determinant flattens the square.

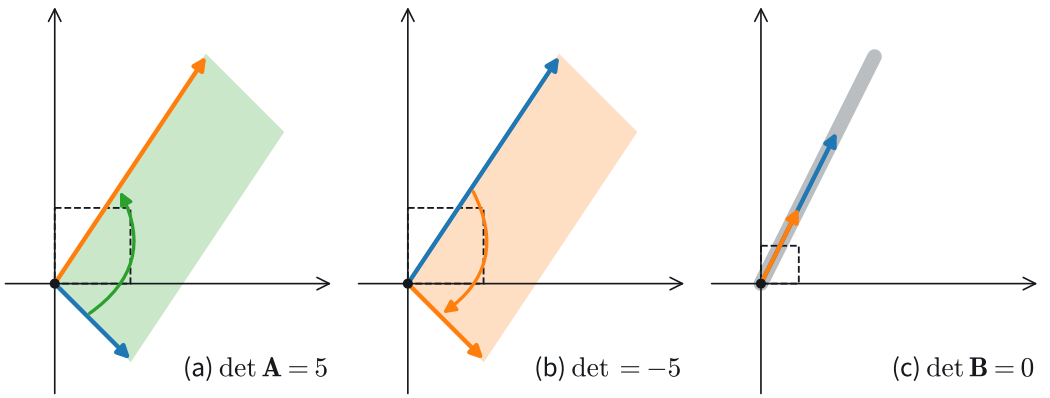


Figure 33.13: The determinant as signed area for the three matrices of this section. The unit square (dashed) goes to the region spanned by the columns. (a) $\mathbf{A}$, determinant 5: the columns keep their counter-clockwise order (curved arrow). (b) $\mathbf{A}$ with its columns exchanged, determinant -5: the same parallelogram with the orientation reversed. (c) $\mathbf{B}$, determinant 0: the square collapses to a segment of the line $y = 2x$. Panel (c) is drawn at half the scale of (a) and (b).

Base times height is a formula for the plane. For n columns we need the volume of a box in $\mathbb{R}^n$.

33.2.3.2 The Determinant in General

What is the signed volume of the box spanned by n vectors in $\mathbb{R}^n$? Rather than compute volumes in n dimensions directly, we list three properties of signed volume and let them determine the answer. Stated for the columns $\mathbf{a}_1, \ldots, \mathbf{a}_n$ of an $n \times n$ matrix, the determinant is

1. **Multilinear**: linear in each column separately, with the other columns held fixed. (Scaling one edge of a box scales its volume; splitting one edge as a sum splits the volume accordingly.)
2. **Alternating**: zero whenever two columns are equal. (The box then flattens into a lower-dimensional slab of zero volume.)
3. **Normalized**: $\det \mathbf{I} = 1$. (The unit cube has volume one.)

The area $ps - qr$ has all three properties: it is linear in (p, r) and in (q, s), it vanishes when $(p, r) = (q, s)$, and it is 1 for the identity. Two consequences follow, a rule for swapping columns and a formula, and the formula shows that at most one function can have all three properties.

First, **swapping two columns flips the sign of the determinant**. Place the vector $\mathbf{u} + \mathbf{w}$ in two column slots and expand by multilinearity:

$$\begin{aligned} 0 &= \det(\dots, \mathbf{u} + \mathbf{w}, \dots, \mathbf{u} + \mathbf{w}, \dots) \\ &= \det(\dots, \mathbf{u}, \dots, \mathbf{u}, \dots) + \det(\dots, \mathbf{u}, \dots, \mathbf{w}, \dots) + \det(\dots, \mathbf{w}, \dots, \mathbf{u}, \dots) + \det(\dots, \mathbf{w}, \dots, \mathbf{w}, \dots). \end{aligned}$$

The left-hand side, the first term and the last term vanish by the alternating property, which leaves

$$\det(\dots, \mathbf{u}, \dots, \mathbf{w}, \dots) = -\det(\dots, \mathbf{w}, \dots, \mathbf{u}, \dots).$$

Exchanging two edges reverses the orientation of the box, as in panel (b) of Figure 33.13.

Second, the three properties pin down a formula. Expand each column in the standard basis, $\mathbf{a}_j = \sum_i a_{ij}\,\mathbf{e}_i$, and apply multilinearity column by column. Then $\det \mathbf{A}$ becomes a sum of terms $a_{i_1 1} a_{i_2 2} \cdots a_{i_n n}\ \det(\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_n})$, one for every choice of a row index in each column. Whenever two of the chosen row indices coincide, the term vanishes by the alternating property. When all are distinct, the columns $\mathbf{e}_{i_1}, \dots, \mathbf{e}_{i_n}$ are those of the identity matrix in another order, and repeated column swaps turn the factor into ± 1. For a 2×2 matrix with columns $\mathbf{a}_1 = p\,\mathbf{e}_1 + r\,\mathbf{e}_2$ and $\mathbf{a}_2 = q\,\mathbf{e}_1 + s\,\mathbf{e}_2$, the expansion has four terms. The two with a repeated basis vector vanish, and

$$\det(\mathbf{a}_1, \mathbf{a}_2) = ps\ \det(\mathbf{e}_1, \mathbf{e}_2) + rq\ \det(\mathbf{e}_2, \mathbf{e}_1) = ps - qr,$$

the area we computed by hand. In general the determinant is a signed sum, over all ways of picking one entry from each row and each column, of the product of the picked entries. Writing $\sigma(j)$ for the row picked in column j, this is the *permutation formula*

$$\det \mathbf{A} = \sum_{\sigma} \operatorname{sgn}(\sigma)\, a_{\sigma(1)1}\, a_{\sigma(2)2} \cdots a_{\sigma(n)n}, \tag{33.20}$$

where σ runs over the $n!$ permutations of $\{1, \dots, n\}$ and $\operatorname{sgn}(\sigma) = \det(\mathbf{e}_{\sigma(1)}, \dots, \mathbf{e}_{\sigma(n)})$ is $+1$ or -1 according to whether σ is made of an even or an odd number of swaps. A function with the three properties must therefore be this one: there is at most one determinant.

That there is one we do not prove in full. The sign is well defined because no permutation is both an even and an odd number of swaps, a fact we take from (Horn and Johnson 2012). The formula then has the three properties: each term contains one entry from each column, which gives multilinearity, and for the identity only one term is nonzero. The alternating property needs the parity fact once more, and we omit the verification.

The derivation used the normalization only in its last step, which gives a sharper statement that the next subsection needs. Let f be *any* multilinear and alternating function

of the columns. The swap rule holds for f, the expansion is the same, and each factor $f(\mathbf{e}_{\sigma(1)}, \dots, \mathbf{e}_{\sigma(n)})$ equals $\operatorname{sgn}(\sigma)\, f(\mathbf{I})$. Hence

$$f(\mathbf{A}) = f(\mathbf{I})\ \det \mathbf{A}.$$

Every multilinear alternating function of the columns is a multiple of the determinant.

Grouping the terms of the formula by the row picked in column 1 gives the *cofactor expansion* used for hand computation: the entry a_{i1} multiplies the determinant of the submatrix without row i and column 1, with the sign $(-1)^{i+1}$. For the 3×3 example

$$\mathbf{M} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 3 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

it reads

$$\det \mathbf{M} = 2 \det \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix} - 1 \det \begin{bmatrix} 1 & 0 \\ 1 & 2 \end{bmatrix} + 0 \det \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} = 2 \cdot 5 - 1 \cdot 2 + 0 = 8. \tag{33.21}$$

So $\mathbf{M}$ maps the unit cube to a box of volume 8.

For a triangular matrix the determinant is the product of the diagonal entries. Of all the rules for computing determinants this is the one to remember. Suppose $\mathbf{U}$ is upper triangular, so $u_{ij} = 0$ whenever $i > j$. Each term of the permutation formula picks one entry from every column, all in distinct rows. To be nonzero, the entry picked from column 1 must come from row 1, since every lower entry vanishes. The entry from column 2 must then come from row 2, because row 1 is taken and rows $3, \dots, n$ hold zeros. Continuing across the columns, only the diagonal choice survives, and it needs no swaps, so

$$\det \mathbf{U} = u_{11} u_{22} \cdots u_{nn}.$$

The same argument, run from column n backwards, handles lower triangular matrices, and it extends to blocks. Take a *block upper triangular* matrix $\left(\begin{smallmatrix} \mathbf{D}_1 & * \\ \mathbf{0} & \mathbf{D}_2 \end{smallmatrix}\right)$ with square diagonal blocks $\mathbf{D}_1$ and $\mathbf{D}_2$, which need not be triangular themselves. A nonzero term must pick its entries in the columns of $\mathbf{D}_1$ from the rows of $\mathbf{D}_1$, since everything below them is zero. That uses up those rows, so the entries in the remaining columns come from the rows of $\mathbf{D}_2$, and the block marked $*$ never enters. The surviving permutations are therefore pairs, one permutation for each block. The sign of a pair is the product of the two signs, and the sum factors:

$$\det \begin{bmatrix} \mathbf{D}_1 & * \\ \mathbf{0} & \mathbf{D}_2 \end{bmatrix} = \det \mathbf{D}_1\ \det \mathbf{D}_2.$$

These two rules matter beyond hand computation. A normalizing flow transforms a random variable by an invertible map and needs the logarithm of the determinant of the map's Jacobian at every evaluation, because the density of the transformed variable is the old density divided by the factor by which the map scales volume (Section 36.2.3.1). Flows are therefore

built from layers whose Jacobians are triangular or block triangular. The log-determinant is then a sum of n logarithms, where a general determinant costs about $n^3/3$ multiply–adds.

Transposing does not change the determinant, $\det \mathbf{A}^\top = \det \mathbf{A}$. The permutation formula for $\mathbf{A}^\top$ has the terms $\operatorname{sgn}(\sigma) \prod_j a_{j\sigma(j)}$. Substituting $j = \sigma^{-1}(i)$ rewrites the product as $\prod_i a_{\sigma^{-1}(i)\,i}$, the term of the formula for $\mathbf{A}$ that belongs to σ^{-1}, and $\operatorname{sgn}(\sigma^{-1}) = \operatorname{sgn}(\sigma)$, because undoing a sequence of swaps takes the same number of swaps. The two formulas have the same terms, so everything the determinant says about columns holds for rows as well.

The permutation formula has $n!$ terms and is never used beyond tiny sizes. Libraries factor the matrix into triangular factors, as in `solve`, and multiply their diagonal entries, with a sign for the row exchanges, at a cost of about $n^3/3$ multiply–adds.

33.2.3.3 Multiplicativity

If $\mathbf{B}$ scales areas by $\det \mathbf{B}$ and $\mathbf{A}$ scales them by $\det \mathbf{A}$, then the composite $\mathbf{AB}$ should scale them by the product. For the running matrix, $\mathbf{A}^2 = \left(\begin{smallmatrix} -1 & 8 \\ -4 & 7 \end{smallmatrix}\right)$ and $\det \mathbf{A}^2 = -7 + 32 = 25 = 5^2$. For the composite of Section 33.2.1.2, $\det(\mathbf{R}_{90}\mathbf{T}) = 0 \cdot 1 - (-1) \cdot 1 = 1$, the product of $\det \mathbf{R}_{90} = 1$ and $\det \mathbf{T} = 1$.

The intuition comes from tiling. Cover a planar figure with small squares. A linear map sends each square to a parallelogram, and all of these parallelograms have the same area, $|\det \mathbf{A}|$ times that of a square, because the map acts in the same way everywhere. So a linear map scales the area of every figure by the same factor $|\det \mathbf{A}|$. Applying $\mathbf{B}$ to a figure of area V gives the area $|\det \mathbf{B}|\, V$, and applying $\mathbf{A}$ next gives $|\det \mathbf{A}|\,|\det \mathbf{B}|\, V$. The composite is a single linear map, so its own factor must be this product. The three properties turn the tiling argument into a proof, signs included.

Proposition (multiplicativity of the determinant). *For square matrices $\mathbf{A}$ and $\mathbf{B}$ of the same size,*

$$\det(\mathbf{AB}) = \det(\mathbf{A})\, \det(\mathbf{B}). \tag{33.22}$$

Proof. Fix $\mathbf{A}$ and view $D(\mathbf{B}) = \det(\mathbf{AB})$ as a function of the columns of $\mathbf{B}$ alone. By Section 33.2.1.2, column j of $\mathbf{AB}$ is $\mathbf{A}\mathbf{b}_j$. Because $\mathbf{A}(\cdot)$ is linear and det is multilinear, D is multilinear in the columns of $\mathbf{B}$. If two columns of $\mathbf{B}$ are equal, so are the corresponding columns of $\mathbf{AB}$, and $D(\mathbf{B}) = 0$; so D is alternating. Every multilinear alternating function is a multiple of the determinant (Section 33.2.3.2), so

$$\det(\mathbf{AB}) = D(\mathbf{B}) = D(\mathbf{I})\, \det(\mathbf{B}) = \det(\mathbf{A})\, \det(\mathbf{B}). \qquad \blacksquare$$

Four consequences of Equation 33.22 recur in the following sections.

1. *Determinants commute even when matrices do not.* Since the two determinants are scalars, $\det(\mathbf{AB}) = \det \mathbf{A} \det \mathbf{B} = \det \mathbf{B} \det \mathbf{A} = \det(\mathbf{BA})$. For the rotation and the shear, $\mathbf{R}_{90}\mathbf{T} \neq \mathbf{T}\mathbf{R}_{90}$, but both products have determinant 1.
2. *Similarity invariance.* For an invertible $\mathbf{V}$, applying consequence 1 to the pair $\mathbf{VA}$ and $\mathbf{V}^{-1}$ gives $\det(\mathbf{VAV}^{-1}) = \det(\mathbf{V}^{-1}\mathbf{VA}) = \det \mathbf{A}$. The matrix $\mathbf{VAV}^{-1}$ describes the map $\mathbf{A}$ in the coordinates whose axes are the columns of $\mathbf{V}^{-1}$, and a change of coordinates does not change the volume factor.

3. *Inverses.* If $\mathbf{A}$ is invertible, taking $\mathbf{B} = \mathbf{A}^{-1}$ gives $\det \mathbf{A} \det \mathbf{A}^{-1} = \det \mathbf{I} = 1$. Hence an invertible matrix has $\det \mathbf{A} \neq 0$, and $\det \mathbf{A}^{-1} = 1/\det \mathbf{A}$; for the running matrix, $1/5 = 0.2$.
4. *Orthogonal matrices.* Combining multiplicativity with $\det \mathbf{Q}^\top = \det \mathbf{Q}$ and applying both to $\mathbf{Q}^\top \mathbf{Q} = \mathbf{I}$ gives

$$\det(\mathbf{Q})^2 = \det(\mathbf{Q}^\top)\det(\mathbf{Q}) = \det(\mathbf{Q}^\top \mathbf{Q}) = \det(\mathbf{I}) = 1,$$

so $\det \mathbf{Q} = \pm 1$. A rigid motion leaves the size of every volume unchanged, and the sign records whether it keeps the orientation: it is $+1$ for the rotations and -1 for the reflections of the plane (Section 33.2.1.3).

33.2.3.4 Invertibility and the Determinant

The determinant adds one more equivalent statement to the four of Section 33.2.2.3, the only one that is a single number. In two dimensions it agrees with the others on sight. The signed area $ps - qr$ is zero if and only if the two edges lie on one line through the origin (or one of them is $\mathbf{0}$), that is, if and only if the columns are dependent. The same holds in every dimension.

Proposition (invertibility and the determinant). *An $n \times n$ matrix $\mathbf{A}$ is invertible if and only if $\det \mathbf{A} \neq 0$. Equivalently, $\det \mathbf{A} = 0$ if and only if the columns of $\mathbf{A}$ are linearly dependent.*

Proof. An invertible matrix has $\det \mathbf{A} \det \mathbf{A}^{-1} = 1$ by consequence 3 of multiplicativity, so $\det \mathbf{A} \neq 0$. If $\mathbf{A}$ is not invertible, its columns are dependent by the proposition of Section 33.2.2.3, so one of them is a combination of the others, $\mathbf{a}_j = \sum_{i \neq j} c_i \mathbf{a}_i$. Multilinearity in column j gives $\det \mathbf{A} = \sum_{i \neq j} c_i \det(\dots, \mathbf{a}_i, \dots)$, where $\mathbf{a}_i$ now stands in slot j. Every term has two equal columns, so it vanishes by the alternating property. ■

For the running matrix, $\det \mathbf{A} = 5 \neq 0$, and for $\mathbf{B}$, with its dependent columns, $\det \mathbf{B} = 0$. A nonzero determinant certifies that the inverse exists. It does not say that the inverse is safe to compute (Section 33.2.3.5).

33.2.3.5 Limits of the Determinant

A nonzero determinant says that a matrix can be undone. It does not tell us how *close* the matrix is to collapsing space, or in which direction. Two examples show this from opposite sides. The matrix $0.1\,\mathbf{I}_{400}$ is as harmless as a matrix can be, with inverse $10\,\mathbf{I}_{400}$, yet its determinant 10^{-400} is far below the smallest positive double-precision number. The shear $\left(\begin{smallmatrix} 1 & 10^8 \\ 0 & 1 \end{smallmatrix}\right)$ has determinant 1, yet a change of 10^{-8} in its lower left entry makes it singular (7). The cell computes both determinants, the first also through `slogdet`, which returns the sign and the logarithm of the absolute value, and the singular values of the shear. Before running it, guess the rank that the library reports for the unchanged shear.

```
sign, logdet = np.linalg.slogdet(0.1 * np.eye(400))
print(np.linalg.det(0.1 * np.eye(400)), sign, logdet)
shear = np.array([[1.0, 1e8], [0.0, 1.0]])
print(np.linalg.det(shear), np.linalg.matrix_rank(shear),
      np.linalg.svd(shear, compute_uv=False))
shear[1, 0] = 1e-8
```

```
print(np.linalg.det(shear), np.linalg.matrix_rank(shear))
```

```
0.0 1.0 -921.0340371976151
1.0 1 [1.e+08 1.e-08]
0.0 1
```

The determinant of $0.1\,\mathbf{I}_{400}$ underflows to 0.0, while its logarithm, $400 \ln 0.1 \approx -921.03$, is computed without trouble. The shear has determinant 1, and the changed shear has determinant 0 up to rounding. Yet `matrix_rank` reports rank one for both. In exact arithmetic the shear has rank two, but its singular values are about 10^{8} and 10^{-8}. The routine treats a singular value as zero when it is below the largest one times the matrix size times the machine precision, here $10^{8} \cdot 2 \cdot 2.2 \cdot 10^{-16} \approx 4 \cdot 10^{-8}$. The entries of the shear are stored exactly, but a computation with them commits rounding errors of about this size, the machine precision times the largest entry, so the routine does not tell the shear apart from a singular matrix.

One number cannot describe how a matrix stretches space in different directions. Eigenvalues and singular values add the directions. For the running matrix $\mathbf{A}$, the eigenvalues are $2 \pm i$, whose product is $5 = \det \mathbf{A}$ (Section 33.3), and the singular values are about 3.618 and 1.382, whose product is $5 = |\det \mathbf{A}|$ (Section 33.5). This holds for every square matrix: the determinant is the product of the eigenvalues, counted with multiplicity and possibly complex, and its absolute value is the product of the singular values. The ratio of the singular values, not the determinant, measures how close a matrix is to being singular.

33.2.4 Summary

A matrix is a linear map, fixed by its columns, the images of the coordinate axes, and the matrix product is the composition of maps. Einstein notation and `einsum` write every such product, and the attention scores of a transformer, as one index rule. The column space is the set of outputs, the null space is the set of inputs sent to zero, and rank–nullity, $\operatorname{rank} \mathbf{A} + \dim \ker \mathbf{A} = n$, ties their dimensions together: a 256×784 layer sends at least 528 input directions to zero. A square matrix is invertible if and only if its columns are independent, if and only if its determinant is nonzero. The determinant is the signed factor by which the map scales volume; it is multiplicative under composition and, for a triangular matrix, the product of the diagonal entries. It certifies that an inverse exists but not that the matrix is far from singular, which the singular values measure.

33.2.5 Exercises

1. **Maps that preserve angles.** Rotating every embedding vector by the same orthogonal matrix leaves all cosine similarities unchanged. This problem finds every linear map with that property.
    1. Find the condition on the columns of a square matrix $\mathbf{A}$ under which $\mathbf{A}\mathbf{e}_1$ and $\mathbf{A}\mathbf{e}_2$ are orthogonal. Show that $\mathbf{A} = \operatorname{diag}(1, 10)$ satisfies it but changes the angle between $(1, 1)$ and $(1, -1)$.
    2. Suppose that an invertible $\mathbf{A}$ preserves the angle between every pair of nonzero vectors. Apply this to the pairs $\mathbf{e}_i, \mathbf{e}_j$ and $\mathbf{e}_i + \mathbf{e}_j, \mathbf{e}_i - \mathbf{e}_j$ for $i \neq j$ to show that $\mathbf{A}^{\top}\mathbf{A} = c^2\mathbf{I}$ for some $c > 0$, so that $\mathbf{A} = c\mathbf{Q}$ with $\mathbf{Q}$ orthogonal. Conversely, show that every such matrix preserves all cosine similarities.

   3. Suppose instead that $\|\mathbf{Ax}\| = \|\mathbf{x}\|$ for every $\mathbf{x}$. Expand $\|\mathbf{x}+\mathbf{y}\|^2$ and $\|\mathbf{x}-\mathbf{y}\|^2$ to show that $\mathbf{A}$ preserves every dot product, and conclude that $\mathbf{A}$ is orthogonal.
2. **Rank and nullity of C.** The 4×5 matrix $\mathbf{C}$ of Section 33.2.2.2 has rank two. Write $\mathbf{c}_1, \dots, \mathbf{c}_5$ for its columns.
   1. Express $\mathbf{c}_2$, $\mathbf{c}_4$, and $\mathbf{c}_5$ as linear combinations of $\mathbf{c}_1$ and $\mathbf{c}_3$, and check that $\mathbf{c}_1$ and $\mathbf{c}_3$ are linearly independent. Why does this show that every collection of three columns is linearly dependent?
   2. Turn the three relations into three vectors of the null space of $\mathbf{C}$. Show that they are linearly independent, and use rank–nullity to show that they form a basis of the null space.
   3. Express the third and the fourth row of $\mathbf{C}$ as linear combinations of the first two rows, and read off the row rank. Then write down the factorization $\mathbf{C} = \mathbf{GH}$ of Equation 33.17 in which $\mathbf{G}$ has the columns $\mathbf{c}_1$ and $\mathbf{c}_3$.
3. **Rank of a product.** A linear autoencoder sends a d-dimensional input through a hidden layer of width $k < d$, $\mathbf{x} \mapsto \mathbf{W}_2\mathbf{W}_1\mathbf{x}$ with $\mathbf{W}_1 \in \mathbb{R}^{k\times d}$ and $\mathbf{W}_2 \in \mathbb{R}^{d\times k}$.
   1. Let $\mathbf{A}$ be an $m \times n$ matrix and $\mathbf{B}$ an $n \times p$ matrix. Show that the column space of $\mathbf{AB}$ lies in the column space of $\mathbf{A}$, and that the null space of $\mathbf{B}$ lies in the null space of $\mathbf{AB}$. Use the first fact, and rank–nullity with the second, to prove $\operatorname{rank}(\mathbf{AB}) \le \min(\operatorname{rank}\mathbf{A}, \operatorname{rank}\mathbf{B})$.
   2. Conclude that $\mathbf{W}_2\mathbf{W}_1$ has rank at most k. What is the smallest possible dimension of its null space? What is the largest dimension of a subspace on which the autoencoder reconstructs every input exactly?
   3. Construct a 2×3 matrix and a 3×2 matrix, both of rank two, whose 2×2 product has rank one.
4. code **Rank of a sum and of a low-rank update.** Fine-tuning methods often change a large weight matrix by adding the product of two thin matrices. How much can such an update change the rank?
   1. Show that every column of $\mathbf{A}+\mathbf{B}$ lies in the span of the columns of $\mathbf{A}$ and $\mathbf{B}$ together, and conclude $\operatorname{rank}(\mathbf{A}+\mathbf{B}) \le \operatorname{rank}\mathbf{A} + \operatorname{rank}\mathbf{B}$.
   2. A 4096×4096 weight matrix $\mathbf{W}$ is changed by adding $\Delta = \mathbf{UV}^\top$ with $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{4096\times 8}$. Show that Δ has rank at most 8, and that the rank of $\mathbf{W}+\Delta$ differs from that of $\mathbf{W}$ by at most 8 in either direction (hint: apply the first part also to $\mathbf{W} = (\mathbf{W}+\Delta) + (-\Delta)$).
   3. Give 2×2 examples with $\operatorname{rank}\mathbf{A} = \operatorname{rank}\mathbf{B} = 1$ and $\operatorname{rank}(\mathbf{A}+\mathbf{B}) = 0$, and with $\operatorname{rank}(\mathbf{A}+\mathbf{B}) = 2$.
   4. Build a 50×50 matrix of rank 10 as the product of a 50×10 and a 10×50 matrix with independent standard Gaussian entries, add a rank-3 product of the same kind, and check that `np.linalg.matrix_rank` returns 13. Explain why this happens for every draw (hint: write the sum as one product of a 50×13 and a 13×50 matrix, and use that Gaussian columns are linearly independent with probability one).
5. **Matrices that commute with a rotation.** Matrix products do not commute in general. This problem finds the exceptions for $\mathbf{R}_{90}$.
   1. Find all 2×2 matrices $\mathbf{X}$ with $\mathbf{XR}_{90} = \mathbf{R}_{90}\mathbf{X}$, and show that they are $\left(\begin{smallmatrix} p & -q \\ q & p \end{smallmatrix}\right) = p\mathbf{I} + q\mathbf{R}_{90}$.
   2. Using $\mathbf{R}_{90}^2 = -\mathbf{I}$, show that $(p\mathbf{I}+q\mathbf{R}_{90})(r\mathbf{I}+s\mathbf{R}_{90}) = (pr-qs)\,\mathbf{I} + (ps+qr)\,\mathbf{R}_{90}$, so these matrices multiply like the complex numbers $p+qi$.
   3. For $(p,q) \neq (0,0)$, show that $p\mathbf{I} + q\mathbf{R}_{90} = \sqrt{p^2+q^2}\,\mathbf{R}_\phi$ for the angle ϕ with $(\cos\phi, \sin\phi) = (p,q)/\sqrt{p^2+q^2}$, a scaled rotation. Compute its determinant,

and conclude that it is invertible unless $p = q = 0$.

4. Does the running matrix $\mathbf{A}$ commute with $\mathbf{R}_{90}$? Compute both products.

6. **Householder reflections.** Let $\mathbf{u}$ be a unit vector in $\mathbb{R}^n$ and $\mathbf{H} = \mathbf{I} - 2\mathbf{u}\mathbf{u}^\top$.
   1. Show that $\mathbf{H}$ is symmetric and orthogonal, that $\mathbf{H}\mathbf{u} = -\mathbf{u}$, and that $\mathbf{H}\mathbf{x} = \mathbf{x}$ for every $\mathbf{x}$ perpendicular to $\mathbf{u}$, so that $\mathbf{H}$ reflects in the hyperplane with normal $\mathbf{u}$.
   2. Write $\mathbf{H} = \mathbf{I} - 2\mathbf{P}$ with $\mathbf{P} = \mathbf{u}\mathbf{u}^\top$, and explain the formula with the projection onto $\mathbf{u}$ from Section 33.1: what does $\mathbf{x} - 2\mathbf{P}\mathbf{x}$ do to the component of $\mathbf{x}$ along $\mathbf{u}$?
   3. Prove $\det \mathbf{H} = -1$. Extend $\mathbf{u}$ to an orthonormal basis, the columns of an orthogonal matrix $\mathbf{Q} = [\mathbf{u}\, \mathbf{q}_2 \cdots \mathbf{q}_n]$, compute $\mathbf{Q}^\top \mathbf{H} \mathbf{Q}$, and use multiplicativity.
   4. Find the unit vector $\mathbf{u}$ whose reflection sends $(3, 4)$ to $(5, 0)$ (hint: $\mathbf{u}$ is parallel to $(3, 4) - (5, 0)$), write down $\mathbf{H}$, and check $\det \mathbf{H} = -1$. Reflections of this kind, applied column by column, are how libraries factor a matrix into an orthogonal and a triangular factor.

7. code **Determinant and near-singularity.** A square matrix is invertible if and only if its determinant is nonzero. How well does the size of the determinant measure how close a matrix is to being singular?
   1. Use multilinearity to prove $\det(c\mathbf{A}) = c^n \det \mathbf{A}$ for an $n \times n$ matrix $\mathbf{A}$ and a scalar c, and check the identity in code for the matrix $\mathbf{M}$ of Equation 33.21 with $c = 2$.
   2. Draw an $n \times n$ lower triangular matrix with independent standard Gaussian entries on and below the diagonal, for $n = 20$, 100, and 1000. Compare the logarithm that `np.linalg.slogdet` returns with the sum of the logarithms of the absolute diagonal entries, which the triangular rule makes exact up to rounding. From which n on do the two disagree, and by how much at $n = 1000$? Compute `np.linalg.cond` for $n = 20$ and $n = 100$, and explain the failure. (The routine does not know that the matrix is triangular. It factors it with row exchanges, like any other matrix.)
   3. The text states that a change of 10^{-8} in the lower left entry makes the shear $\left(\begin{smallmatrix} 1 & 10^8 \\ 0 & 1 \end{smallmatrix}\right)$ singular. Verify it, and show that no smaller change of a single entry does.

8. code **Area of the image of the unit disk.** The determinant scales the area of every region, not only of squares.
   1. Use the tiling argument of Section 33.2.3.3 to explain why $\mathbf{A}$ maps any region of area V to a region of area $|\det \mathbf{A}|\, V$, so that the running matrix maps the unit disk to a region of area $5\pi \approx 15.71$.
   2. Show that the image lies in the rectangle $[-\sqrt{5}, \sqrt{5}] \times [-\sqrt{10}, \sqrt{10}]$ (hint: by Cauchy–Schwarz, $|(\mathbf{A}\mathbf{x})_i|$ is at most the length of row i times $\|\mathbf{x}\|$).
   3. Estimate the area by Monte Carlo. Draw $N = 10^5$ points $\mathbf{p}$ uniformly in the rectangle with `np.random.default_rng()`, count those whose preimage $\mathbf{A}^{-1}\mathbf{p}$ lies in the unit disk, and multiply the fraction by the area $4\sqrt{50} \approx 28.28$ of the rectangle. Show that the standard error of the estimate is about 0.04, and check that the interval of two standard errors around your estimate contains 5π, as it will in about 95% of runs.
   4. Why does this method fail for $\mathbf{B}$? What is the area of the image of the unit disk under $\mathbf{B}$, and how does it relate to $\det \mathbf{B}$?

9. **Determinant of a rank-one update.** ⊠ A *planar flow* (Rezende and Mohamed 2015) transforms $\mathbf{z} \in \mathbb{R}^d$ by $f(\mathbf{z}) = \mathbf{z} + \mathbf{u}\, h(\mathbf{w} \cdot \mathbf{z} + b)$ for vectors $\mathbf{u}, \mathbf{w}$, a scalar b, and a scalar nonlinearity h. Computing the density of $f(\mathbf{z})$ requires the determinant of the Jacobian $\mathbf{I} + h'(\mathbf{w} \cdot \mathbf{z} + b)\, \mathbf{u}\mathbf{w}^\top$.
   1. Use multilinearity and the alternating property to show $\det(\mathbf{u}\mathbf{w}^\top) = 0$ for $n \times n$ matrices with $n \geq 2$.

2. Verify the two factorizations

$$\begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{w}^\top & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{w}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{0}^\top & 1 + \mathbf{w} \cdot \mathbf{u} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & -\mathbf{u} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} + \mathbf{u}\mathbf{w}^\top & \mathbf{0} \\ \mathbf{w}^\top & 1 \end{bmatrix}. \quad (33.23)$$

   Use the permutation formula Equation 33.20 to show that a matrix whose last column is $\mathbf{e}_{n+1}$ has the determinant of its upper-left $n \times n$ block. Deduce $\det(\mathbf{I} + \mathbf{u}\mathbf{w}^\top) = 1 + \mathbf{w} \cdot \mathbf{u}$ from Equation 33.22 and the triangular rule.
3. Write the logarithm of the absolute Jacobian determinant of a planar flow, and compare the cost of evaluating it with the cost of a general $d \times d$ determinant. For $h = \tanh$, which values of $\mathbf{u} \cdot \mathbf{w}$ keep the Jacobian invertible at every $\mathbf{z}$?

10. code **Einstein summation for attention.** Let the query, key and value tensors have the shapes (n_b, n_h, n_q, d_h), (n_b, n_h, n_k, d_h) and (n_b, n_h, n_k, d_h).
    1. Write the `einsum` strings for the scores $s_{bhit} = \sum_j q_{bhij} k_{bhtj}$ and for the weighted sum of values $o_{bhij} = \sum_t p_{bhit}\, v_{bhtj}$, where p_{bhit} are the attention weights, and check both against `@` with `swapaxes`.
    2. Count the multiply–adds of each contraction as a product of the sizes. Compare them with the multiply–adds of applying one $d_h \times d_h$ linear map to every query, and find the number of keys n_k beyond which the scores cost more.
    3. Write $\mathrm{tr}(\mathbf{A}^\top \mathbf{A})$, $\mathrm{tr}(\mathbf{A}\mathbf{A}^\top)$ and $\sum_{ij} a_{ij}^2$ as one `einsum` each for a random 3×5 matrix, and check that they agree. Which of the two equalities is an instance of Equation 33.15, and why does the other one hold?

33.3 Eigendecompositions

In Section 33.2 we asked what a matrix does to space when it is applied once. Many computations apply the same matrix again and again. A deep linear network with equal layers computes $\mathbf{A}^{50}\mathbf{x}$ after 50 layers, and gradient descent on a quadratic multiplies its error by one fixed matrix per step. In each case we want to know whether the signal grows, dies out, or settles, and how quickly. Multiplying out the powers gives the numbers but does not explain them.

A random walk between two web pages is the smallest instance of the question. From page 1 a surfer clicks to page 1 or to page 2 with probability $1/2$ each; from page 2 she stays with probability $3/4$ and returns to page 1 with probability $1/4$. Let $\mathbf{p}_k$ hold the probabilities of being on each page after k clicks. One click multiplies it by a fixed matrix,

$$\mathbf{p}_{k+1} = \mathbf{P}\mathbf{p}_k, \qquad \mathbf{P} = \begin{pmatrix} 1/2 & 1/4 \\ 1/2 & 3/4 \end{pmatrix},$$

whose columns are the click probabilities from each page, so each column sums to one. Starting on page 1, $\mathbf{p}_0 = (1, 0)$, the distribution reads $(0.5, 0.5)$, $(0.375, 0.625)$, $(0.344, 0.656)$, $(0.336, 0.664)$. The changes shrink from click to click, as if the walk were settling on a fixed distribution. Where is the surfer after 50 clicks, and how fast does the answer settle? PageRank asks the same question of the whole web.

The answer is to write the starting vector in the coordinates of a few special directions, the eigenvectors, along which the matrix acts as multiplication by a number. Each coordinate then evolves by its own scalar power λ^k.

In Section 1.3 we met eigenvectors as the directions that a matrix only stretches: the symmetric $\mathbf{S} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ stretches $(1,1)$ by 3 and $(1,-1)$ by 1, and for a symmetric matrix $\mathbf{A}^k\mathbf{x} = \sum_i c_i \lambda_i^k \mathbf{v}_i$. This section drops the symmetry. We find the eigenvalues of any square matrix, see when a full set of eigenvectors exists and what replaces it when it does not, and then prove what symmetry guarantees.

Three results matter most: the expansion Equation 33.27 of $\mathbf{A}^k\mathbf{x}$ in eigenvectors, the spectral theorem, and the quadratic form as a weighted sum of squares, Equation 33.35. A reader in a hurry can go from Section 33.3.1.3 straight to Section 33.3.2.1. Two things are left to Section 33.4: how eigenvalues are located and computed when a matrix is large, and the cases in which they mislead. We compute with NumPy throughout; its `eig` returns complex eigenvalues when a real matrix has them.

```
import numpy as np
```

33.3.1 Eigenvalues and Diagonalization

33.3.1.1 Eigenvectors and Eigenvalues

Some inputs are only rescaled by $\mathbf{P}$, as if it were a number. The vector $(1,2)$ is mapped to $\mathbf{P}(1,2) = (\frac{1}{2}+\frac{1}{2},\ \frac{1}{2}+\frac{3}{2}) = (1,2)$, and $(1,-1)$ is mapped to $(\frac{1}{2}-\frac{1}{4},\ \frac{1}{2}-\frac{3}{4}) = \frac{1}{4}(1,-1)$. Along the first direction $\mathbf{P}$ multiplies by 1, along the second by $1/4$.

In general, a nonzero vector $\mathbf{v}$ is an *eigenvector* of a square matrix $\mathbf{A}$, with *eigenvalue* λ, if

$$\mathbf{A}\mathbf{v} = \lambda \mathbf{v}. \tag{33.24}$$

The matrix only stretches an eigenvector by the factor λ, and reverses it when $\lambda < 0$. For $\mathbf{P}$, the pairs are $(1,(1,2))$ and $(\frac{1}{4},(1,-1))$. The zero vector is excluded as an eigenvector because it satisfies Equation 33.24 for every λ. All solutions of $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ for one λ, together with $\mathbf{0}$, form a subspace, the *eigenspace* of λ.

The eigenvalue $\lambda = 0$ is allowed. It means $\mathbf{A}\mathbf{v} = \mathbf{0}$ for some $\mathbf{v} \neq \mathbf{0}$, so a square matrix is singular if and only if 0 is one of its eigenvalues. Not every real matrix has a real eigenvector: a rotation by 90° turns every direction. Its eigenvalues are complex, and we return to them in Section 33.3.1.6.

A 2×2 invertible matrix maps the unit circle to an ellipse. Are the axes of that ellipse the eigenvectors? Figure 33.14 compares two matrices. For the symmetric $\mathbf{S}$ the eigenvectors lie on the axes. For the upper-triangular $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 2 \end{smallmatrix}\right)$ they do not: its eigenvectors are $(1,0)$ with $\lambda = 1$ and $(1,1)$ with $\lambda = 2$, while the ellipse axes point at 58° and -32° (Section 33.5 shows how to compute them). The ellipse axes are the directions of largest and smallest stretch; the eigenvectors are the directions that are not turned. For symmetric matrices the two coincide, as the spectral theorem of Section 33.3.2.1 proves. For other matrices they need not.

33.3.1.2 The Characteristic Polynomial

We found the eigenvectors of $\mathbf{P}$ by inspection. To find the eigenvalues of any matrix, move $\lambda\mathbf{v}$ to the left side of Equation 33.24 and factor out the vector:

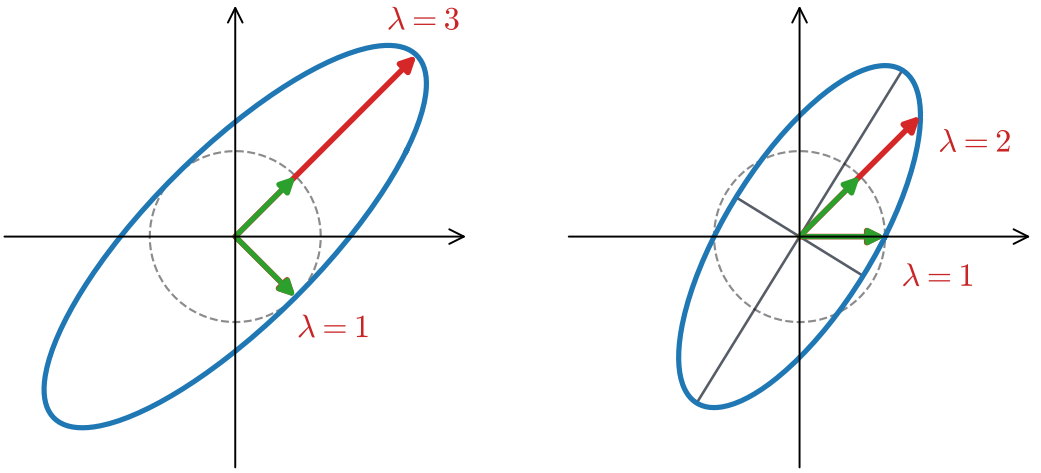


Figure 33.14: The unit circle (dashed) and its image (blue). Left: the symmetric $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$; its unit eigenvectors (green) are the ellipse axes, and their images (red) are scaled by 3 and 1 (for $\lambda = 1$ the two arrows coincide). Right: the upper-triangular $\left(\begin{smallmatrix} 1 & 1 \\ 0 & 2 \end{smallmatrix}\right)$; its eigenvectors $(1, 0)$ and $(1, 1)$ are not the ellipse axes (grey).

$$(\mathbf{A} - \lambda \mathbf{I})\mathbf{v} = \mathbf{0}. \tag{33.25}$$

A nonzero solution $\mathbf{v}$ exists if and only if $\mathbf{A} - \lambda\mathbf{I}$ has a nontrivial null space, that is, if and only if it is singular and $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. Expanding the determinant gives a polynomial of degree n in λ, the *characteristic polynomial* $p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I})$. Its roots are the eigenvalues. For each root, solving Equation 33.25 gives the eigenspace.

For the surfer's matrix,

$$\det(\mathbf{P} - \lambda\mathbf{I}) = \left(\tfrac{1}{2} - \lambda\right)\left(\tfrac{3}{4} - \lambda\right) - \tfrac{1}{4} \cdot \tfrac{1}{2} = \lambda^2 - \tfrac{5}{4}\lambda + \tfrac{1}{4} = (\lambda - 1)\left(\lambda - \tfrac{1}{4}\right),$$

so the eigenvalues are 1 and $1/4$, the two factors we found by inspection. The eigenvectors follow from one row of each singular system. For $\lambda = 1$ the first row of $\mathbf{P} - \mathbf{I}$ reads $-\frac{1}{2}v_1 + \frac{1}{4}v_2 = 0$, so $v_2 = 2v_1$ and $\mathbf{v} = (1, 2)$. For $\lambda = \frac{1}{4}$ the first row of $\mathbf{P} - \frac{1}{4}\mathbf{I}$ reads $\frac{1}{4}v_1 + \frac{1}{4}v_2 = 0$, so $\mathbf{v} = (1, -1)$. Each eigenvector is determined only up to a nonzero factor: if $\mathbf{v}$ is an eigenvector, so is $c\mathbf{v}$ for any $c \neq 0$, since both sides of Equation 33.24 scale by c.

In practice a library does this work. NumPy's `eig` returns the eigenvalues and, as the columns of a matrix, one eigenvector for each; the last line of the cell checks Equation 33.24 for all pairs at once.

```
P = np.array([[0.5, 0.25], [0.5, 0.75]])
lam, V = np.linalg.eig(P)
print('eigenvalues:', lam)
print('eigenvectors (columns):\n', V)
print('max |P V - V diag(lam)|:', np.abs(P @ V - V * lam).max())
```

```
eigenvalues: [0.25 1.  ]
eigenvectors (columns):
 [[-0.70710678 -0.4472136 ]
 [ 0.70710678 -0.89442719]]
max |P V - V diag(lam)|: 0.0
```

Order, scale and sign are the library's choice: the eigenvalues come as $1/4$ and 1, and the columns are parallel to $(1,-1)$ and $(1,2)$, scaled to length one.

Libraries do not solve the characteristic polynomial. For $n \geq 5$ there is no general formula for its roots in radicals, and the roots are highly sensitive to rounding errors in the coefficients. Instead, `eig` applies a sequence of orthogonal changes of basis that drives the matrix to triangular form without changing its eigenvalues. A triangular matrix shows its eigenvalues on its diagonal: subtracting $\lambda\mathbf{I}$ leaves it triangular, and the determinant of a triangular matrix is the product of its diagonal entries (Section 33.2.3.2). For matrices too large to factor at all, power iteration (Section 33.4.1.2) needs only matrix–vector products.

33.3.1.3 Diagonalization

We return to the opening question, $\mathbf{P}^{50}\mathbf{p}_0$. Collect the two eigenvectors as the columns of $\mathbf{V}$ and the eigenvalues on the diagonal of Λ:

$$\mathbf{V} = \begin{pmatrix} 1 & 1 \\ 2 & -1 \end{pmatrix}, \qquad \Lambda = \begin{pmatrix} 1 & 0 \\ 0 & 1/4 \end{pmatrix}.$$

The two eigenvector equations $\mathbf{P}\mathbf{v}_i = \lambda_i \mathbf{v}_i$, written side by side as columns, read $\mathbf{P}\mathbf{V} = \mathbf{V}\Lambda$. Since $\det \mathbf{V} = -1 - 2 = -3 \neq 0$, the matrix $\mathbf{V}$ is invertible, with $\mathbf{V}^{-1} = \frac{1}{3}\left(\begin{smallmatrix} 1 & 1 \\ 2 & -1 \end{smallmatrix}\right)$, and multiplying by $\mathbf{V}^{-1}$ on the right gives

$$\mathbf{P} = \mathbf{V}\Lambda\mathbf{V}^{-1}. \tag{33.26}$$

In words: change to eigenvector coordinates ($\mathbf{V}^{-1}$), scale each coordinate by its eigenvalue (Λ), and change back ($\mathbf{V}$). A square matrix $\mathbf{A}$ that can be written as $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$, with an invertible $\mathbf{V}$ and a diagonal Λ, is called *diagonalizable*; the factorization is its *eigendecomposition*. It exists if and only if the matrix has n linearly independent eigenvectors to fill the columns of $\mathbf{V}$.

Powers of a diagonalizable matrix reduce to powers of numbers. In a product of k copies of $\mathbf{V}\Lambda\mathbf{V}^{-1}$, each inner pair $\mathbf{V}^{-1}\mathbf{V}$ cancels:

$$\mathbf{A}^k = \overbrace{(\mathbf{V}\Lambda\mathbf{V}^{-1})\cdots(\mathbf{V}\Lambda\mathbf{V}^{-1})}^{k \text{ times}} = \mathbf{V}\,\overbrace{\Lambda\cdots\Lambda}^{k \text{ times}}\,\mathbf{V}^{-1} = \mathbf{V}\Lambda^k\mathbf{V}^{-1}.$$

Applied to a vector $\mathbf{x}$, with $\mathbf{c} = \mathbf{V}^{-1}\mathbf{x}$ the coordinates of $\mathbf{x}$ in the eigenbasis, so that $\mathbf{x} = \sum_i c_i \mathbf{v}_i$, this reads

$$\mathbf{A}^k\mathbf{x} = \mathbf{V}\Lambda^k\mathbf{c} = \sum_{i=1}^{n} c_i \lambda_i^k \mathbf{v}_i. \tag{33.27}$$

Each eigen-coordinate evolves on its own, as the scalar sequence $c_i \lambda_i^k$. For the surfer, $\mathbf{c} = \mathbf{V}^{-1}\mathbf{p}_0 = \frac{1}{3}(1,2) = (\frac{1}{3}, \frac{2}{3})$, and therefore

$$\mathbf{p}_k = \mathbf{P}^k \mathbf{p}_0 = \frac{1}{3}\begin{pmatrix}1\\2\end{pmatrix} + \frac{2}{3}\left(\frac{1}{4}\right)^k \begin{pmatrix}1\\-1\end{pmatrix}. \tag{33.28}$$

This answers the opening question. The distribution approaches $\pi = (1/3, 2/3)$: in the long run the surfer spends twice as much time on page 2 as on page 1. The error is $\frac{2}{3}\,4^{-k}$ in each entry and points along the eigenvector $(1,-1)$. Every click divides it by 4, so after 10 clicks it is 6×10^{-7} and after 50 clicks below 10^{-30}. At $k = 3$ the formula gives $(\frac{1}{3} + \frac{1}{96}, \frac{2}{3} - \frac{1}{96}) = (0.34375, 0.65625)$, the third iterate of the opening. Figure 33.15 shows the geometry: every $\mathbf{p}_k$ has entries summing to one, so the iterates slide along the line $x+y=1$ toward the point where the eigenvector of $\lambda = 1$ meets it.

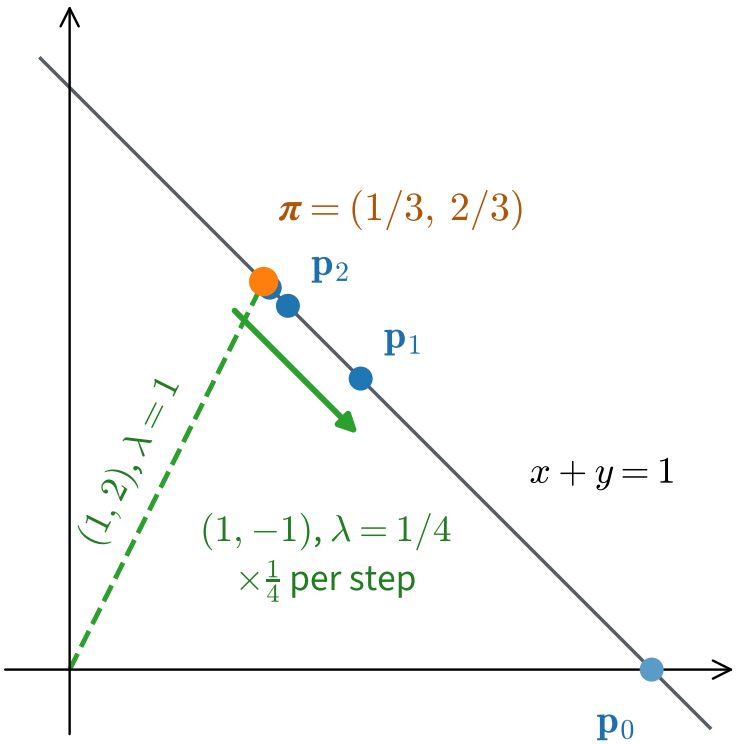


Figure 33.15: The surfer's distributions $\mathbf{p}_k$ slide along $x + y = 1$ toward $\pi = (1/3, 2/3)$, where the eigenvector $(1,2)$ of $\lambda = 1$ meets the line. The error points along the eigenvector $(1,-1)$ of $\lambda = 1/4$ and shrinks by a factor of 4 per click.

Fifty matrix multiplications and the two eigen-coordinates must give the same $\mathbf{p}_{50}$. The cell computes both and prints the ratio of consecutive errors.

```
p0 = np.array([1.0, 0.0])
c = np.linalg.solve(V, p0)              # coordinates of p0 in the eigenbasis
```

```
print('P^50 p0            :', np.linalg.matrix_power(P, 50) @ p0)
print('sum c_i lam_i^50 v_i:', V @ (c * lam ** 50))
pi = np.array([1 / 3, 2 / 3])
errs = np.array([np.abs(np.linalg.matrix_power(P, k) @ p0 - pi).max()
                 for k in range(8)])
print('error ratios:', np.round(errs[1:] / errs[:-1], 4))
```

```
P^50 p0            : [0.33333333 0.66666667]
sum c_i lam_i^50 v_i: [0.33333333 0.66666667]
error ratios: [0.25 0.25 0.25 0.25 0.25 0.25 0.25]
```

The two computations agree, and every error ratio is 0.25. The eigenvalue 1 belongs to the part of $\mathbf{p}_0$ that survives, the eigenvalue $1/4$ to the part that dies out.

The same recipe handles other functions of a diagonalizable matrix. If no eigenvalue is zero, $\mathbf{A}^{-1} = \mathbf{V}\Lambda^{-1}\mathbf{V}^{-1}$: invert each eigenvalue. The matrix exponential $e^{\mathbf{A}t} = \mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}$, which solves the differential equation $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, is obtained in the same way; Section 38.1 develops it.

Similarity. The eigendecomposition says that $\mathbf{P}$ and Λ are the same map written in two bases. Two matrices related by $\mathbf{A}' = \mathbf{B}\mathbf{A}\mathbf{B}^{-1}$ for an invertible $\mathbf{B}$ are called *similar*. The next proposition shows that a change of basis cannot change the eigenvalues. It uses the multiplicativity of the determinant from Section 33.2,

$$\det(\mathbf{X}\mathbf{Y}) = \det\mathbf{X}\,\det\mathbf{Y},$$

which also gives $\det(\mathbf{B}^{-1}) = 1/\det\mathbf{B}$ from $\mathbf{B}\mathbf{B}^{-1} = \mathbf{I}$.

Proposition (similar matrices share the characteristic polynomial). *For every square* $\mathbf{A}$ *and every invertible* $\mathbf{B}$*,*

$$\det(\mathbf{B}\mathbf{A}\mathbf{B}^{-1} - \lambda\mathbf{I}) = \det(\mathbf{A} - \lambda\mathbf{I}). \tag{33.29}$$

Proof. Write $\lambda\mathbf{I} = \mathbf{B}(\lambda\mathbf{I})\mathbf{B}^{-1}$ and factor: $\det(\mathbf{B}\mathbf{A}\mathbf{B}^{-1} - \lambda\mathbf{I}) = \det(\mathbf{B}(\mathbf{A} - \lambda\mathbf{I})\mathbf{B}^{-1}) = \det\mathbf{B}\,\det(\mathbf{A} - \lambda\mathbf{I})\,\det\mathbf{B}^{-1} = \det(\mathbf{A} - \lambda\mathbf{I})$. ■

Similar matrices therefore have the same eigenvalues with the same multiplicities as roots, and every quantity computed from the characteristic polynomial is a *similarity invariant*. The two most useful ones are the determinant and the trace. For a diagonalizable matrix,

$$\det(\mathbf{A}) = \det(\mathbf{V}\Lambda\mathbf{V}^{-1}) = \det(\mathbf{V})\det(\Lambda)\det(\mathbf{V}^{-1}) = \det(\Lambda) = \lambda_1\lambda_2\cdots\lambda_n. \tag{33.30}$$

The determinant is the product of the eigenvalues: whatever $\mathbf{V}$ does to volumes, $\mathbf{V}^{-1}$ undoes, so only the scaling by Λ remains. The trace is cyclic (Section 33.2.1.4),

$$\operatorname{tr}(\mathbf{X}\mathbf{Y}) = \operatorname{tr}(\mathbf{Y}\mathbf{X}),$$

for any $\mathbf{X}$ and $\mathbf{Y}$ whose products are square. With $\mathbf{X} = \mathbf{V}$ and $\mathbf{Y} = \Lambda\mathbf{V}^{-1}$,

$$\operatorname{tr}(\mathbf{A}) = \operatorname{tr}(\mathbf{V}\Lambda\mathbf{V}^{-1}) = \operatorname{tr}(\Lambda\mathbf{V}^{-1}\mathbf{V}) = \operatorname{tr}(\Lambda) = \lambda_1 + \cdots + \lambda_n. \tag{33.31}$$

The trace is the sum of the eigenvalues. On the surfer's matrix both identities check by eye: $\operatorname{tr}\mathbf{P} = \frac{1}{2} + \frac{3}{4} = \frac{5}{4} = 1 + \frac{1}{4}$ and $\det\mathbf{P} = \frac{3}{8} - \frac{1}{8} = \frac{1}{4} = 1 \cdot \frac{1}{4}$. For a 2×2 matrix the two numbers determine both eigenvalues, as the roots of $\lambda^2 - (\operatorname{tr})\lambda + \det$.

Both identities hold for every square matrix, diagonalizable or not, if the eigenvalues are counted over $\mathbb{C}$ with their multiplicity as roots. The *fundamental theorem of algebra*, which we use without proof, says that a polynomial of degree n factors completely over $\mathbb{C}$:

$$\det(\mathbf{A} - \lambda\mathbf{I}) = \prod_{i=1}^{n}(\lambda_i - \lambda).$$

Setting $\lambda = 0$ gives $\det\mathbf{A} = \prod_i \lambda_i$. For the trace, compare the coefficients of λ^{n-1}. On the right it is $(-1)^{n-1}\sum_i \lambda_i$. On the left, only the product of the diagonal entries of $\mathbf{A} - \lambda\mathbf{I}$ reaches degree $n-1$, so the coefficient is $(-1)^{n-1}\sum_i a_{ii}$. Hence $\operatorname{tr}\mathbf{A} = \sum_i \lambda_i$. The complex eigenvalues of a real matrix come in conjugate pairs, so the product and the sum are real.

Finally, the rank of a diagonalizable matrix is the number of its nonzero eigenvalues, counted with multiplicity: multiplying by the invertible $\mathbf{V}$ and $\mathbf{V}^{-1}$ changes no dimensions, so $\operatorname{rank}\mathbf{A} = \operatorname{rank}\Lambda$. Exercise 2 treats the general case.

33.3.1.4 Algebraic and Geometric Multiplicity

The surfer's matrix had two distinct eigenvalues and two independent eigenvectors. The second fact follows from the first.

Proposition (distinct eigenvalues give independent eigenvectors). *Eigenvectors belonging to distinct eigenvalues are linearly independent. In particular, an $n \times n$ matrix with n distinct eigenvalues is diagonalizable.*

Proof. Suppose, for contradiction, that some eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_m$ with distinct eigenvalues $\lambda_1, \dots, \lambda_m$ are dependent, and take a dependence relation $\sum_{i=1}^{m} c_i\mathbf{v}_i = \mathbf{0}$ with the fewest nonzero coefficients; relabel so that $c_m \neq 0$. Such a relation has at least two nonzero coefficients, because $c_m\mathbf{v}_m = \mathbf{0}$ alone would force $\mathbf{v}_m = \mathbf{0}$. Apply $\mathbf{A}$ to the relation and subtract λ_m times the relation:

$$\mathbf{0} = \mathbf{A}\sum_i c_i\mathbf{v}_i - \lambda_m\sum_i c_i\mathbf{v}_i = \sum_{i=1}^{m-1} c_i(\lambda_i - \lambda_m)\,\mathbf{v}_i.$$

The $\mathbf{v}_m$ term cancels, so this relation is shorter. It is still nontrivial: the second nonzero coefficient c_i, $i < m$, survives as $c_i(\lambda_i - \lambda_m) \neq 0$ because the eigenvalues are distinct. This contradicts minimality. With n distinct eigenvalues we obtain n independent eigenvectors, a basis of $\mathbb{C}^n$ (of $\mathbb{R}^n$ when the eigenvalues are real), so $\mathbf{V}$ is invertible. ■

With repeated eigenvalues there are two counts. The number of times λ appears as a root of the characteristic polynomial is its *algebraic multiplicity* $a(\lambda)$. The dimension of its eigenspace, the number of independent eigenvectors it contributes, is its *geometric multiplicity* $g(\lambda)$.

Proposition. *For every eigenvalue λ of a square matrix, $1 \le g(\lambda) \le a(\lambda)$.*

Proof. Since $\mathbf{A}-\lambda\mathbf{I}$ is singular, its null space is not zero, so $g \geq 1$. For the upper bound, take a basis $\mathbf{v}_1, \dots, \mathbf{v}_g$ of the eigenspace, extend it to a basis of $\mathbb{C}^n$, and collect the basis as the columns of $\mathbf{B}$. The first g columns of $\mathbf{AB}$ are $\lambda\mathbf{v}_1, \dots, \lambda\mathbf{v}_g$, so in this basis the matrix is block upper triangular,

$$\mathbf{B}^{-1}\mathbf{AB} = \begin{pmatrix} \lambda\mathbf{I}_g & \mathbf{X} \\ \mathbf{0} & \mathbf{Y} \end{pmatrix}.$$

By Equation 33.29 it has the characteristic polynomial p of $\mathbf{A}$. We write the variable of p as t, because λ now denotes the fixed eigenvalue. Subtracting $t\mathbf{I}$ keeps the matrix block upper triangular, with the diagonal blocks $(\lambda - t)\mathbf{I}_g$ and $\mathbf{Y} - t\mathbf{I}$. The determinant of such a matrix is the product of the determinants of its diagonal blocks (Section 33.2.3.2), so $p(t) = (\lambda - t)^g \det(\mathbf{Y} - t\mathbf{I})$. Hence $(\lambda - t)^g$ divides the characteristic polynomial, and $a \geq g$. ■

An eigenvalue can never have more independent eigenvectors than its count as a root, but it can have fewer. Now choose a basis of every eigenspace. The chosen vectors are jointly independent: if a combination of them vanishes, group it by eigenvalue; each group is zero or an eigenvector, and the proposition on distinct eigenvalues forces every group to be zero. A matrix therefore has $\sum_\lambda g(\lambda)$ independent eigenvectors and no more. Over $\mathbb{C}$ the algebraic multiplicities add up to n, and $g \leq a$, so *the eigenvectors of* $\mathbf{A}$ *span* $\mathbb{C}^n$ *if and only if* $g(\lambda) = a(\lambda)$ *for every complex eigenvalue* λ.

A real matrix has a basis of real eigenvectors if and only if, in addition, all of its eigenvalues are real. The rotation by $90°$, $\mathbf{R}_{90} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$, shows that the second condition is needed: its eigenvalues $\pm i$ each have $g = a = 1$, so it has a complex eigenbasis, but no real eigenvector.

When eigenvalues repeat, diagonalizability can fail. The simplest example is the *shear* of Section 33.2.1.2,

$$\mathbf{T} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}. \tag{33.32}$$

Its characteristic polynomial is $(1-\lambda)^2$, so $\lambda = 1$ has algebraic multiplicity 2. But $(\mathbf{T}-\mathbf{I})\mathbf{v} = \mathbf{0}$ reads $\left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)\mathbf{v} = \mathbf{0}$, that is, $v_2 = 0$, so the eigenspace is the line spanned by $(1, 0)$. The geometric multiplicity is 1, there is no eigenbasis, and the shear is not diagonalizable. A matrix with $g(\lambda) < a(\lambda)$ for some eigenvalue is called *defective*. Figure 33.16 shows the shortage: the shear slides every horizontal line sideways by an amount proportional to its height, and the horizontal axis is the only line through the origin that keeps its direction. A diagonalizable 2×2 matrix with a repeated eigenvalue λ would be $\lambda\mathbf{I}$ and would keep every direction.

Powers of the shear show what defectiveness does to iteration: $\mathbf{T}^k = \left(\begin{smallmatrix} 1 & k \\ 0 & 1 \end{smallmatrix}\right)$, so $\mathbf{T}^k\mathbf{e}_2 = (k, 1)$ grows without bound although the only eigenvalue is 1. No eigen-expansion can produce the factor k; the next subsection explains where it comes from.

33.3.1.5 The Jordan Normal Form

When the geometric multiplicity falls short, no eigenbasis exists. What structure remains? Later text needs one consequence of the answer: a defective eigenvalue multiplies λ^k by a polynomial in k. Over the complex numbers, every square matrix is similar to a block-diagonal matrix whose diagonal blocks are *Jordan blocks*

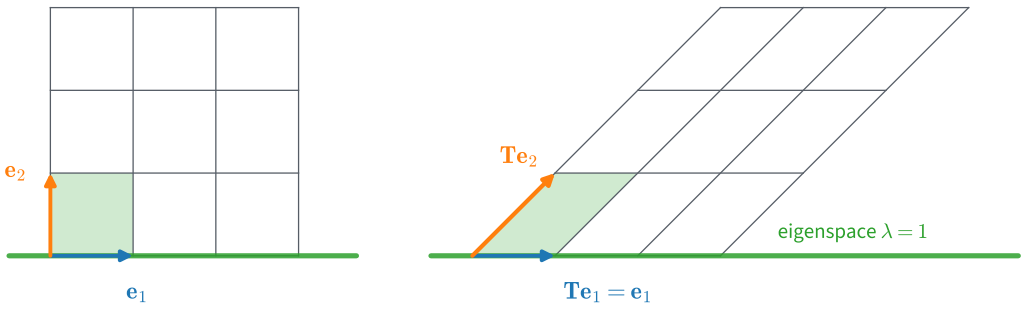


Figure 33.16: The defective shear acting on the plane. Every horizontal line slides to the right by an amount proportional to its height, so the grid tilts, but the x-axis (green) is carried to itself: $\mathbf{Te}_1 = \mathbf{e}_1$, the only eigendirection, with $\lambda = 1$. The other basis vector picks up a horizontal component, $\mathbf{Te}_2 = (1, 1)$, and no second direction is preserved: geometric multiplicity 1 against algebraic multiplicity 2.

$$\mathbf{J}_m(\lambda) = \begin{pmatrix} \lambda & 1 & & \\ & \lambda & \ddots & \\ & & \ddots & 1 \\ & & & \lambda \end{pmatrix},$$

$m \times m$ blocks with one eigenvalue λ repeated down the diagonal and 1s just above it. This is the *Jordan normal form* (Horn and Johnson 2012), which we use without proof. By Equation 33.29 it carries the same eigenvalues as the original matrix.

Each block contributes one eigenvector, its first basis vector. The geometric multiplicity of λ is therefore the number of blocks carrying λ, the algebraic multiplicity is their total size, and a matrix is diagonalizable if and only if every block is 1×1. The shear $\mathbf{T}$ of Equation 33.32 is the single block $\mathbf{J}_2(1)$.

Under iteration a Jordan block does more than scale. Write $\mathbf{J}_m(\lambda) = \lambda\mathbf{I} + \mathbf{N}$, where $\mathbf{N}$ has 1s just above the diagonal and zeros elsewhere. $\mathbf{N}$ is a shift: it sends each basis vector to the previous one, so $\mathbf{N}^m = \mathbf{0}$ (a matrix with a vanishing power is called *nilpotent*). Because $\mathbf{I}$ and $\mathbf{N}$ commute, the binomial theorem applies, and for $m = 2$ every term beyond the first two contains $\mathbf{N}^2 = \mathbf{0}$:

$$\mathbf{J}_2(\lambda)^k = (\lambda\mathbf{I} + \mathbf{N})^k = \lambda^k\mathbf{I} + k\lambda^{k-1}\mathbf{N}.$$

The eigenvalue still decides between growth and decay in the long run, but it is multiplied by a polynomial in k (of degree $m - 1$ for a block of size m). With $|\lambda| = 1$, as for the shear, the iterates drift linearly. With $|\lambda|$ slightly below 1, the corner entry $k\lambda^{k-1}$ first grows, peaks near $k \approx -1/\log|\lambda|$, and only then decays. For $\lambda = 0.99$ this puts the peak near $k \approx 100$, where the corner entry is $100 \cdot 0.99^{99} \approx 37$. The cell checks the prediction.

```
J = np.array([[0.99, 1.], [0., 0.99]])   # the Jordan block J_2(0.99)
for k in [1, 50, 100, 200, 800]:
```

```
    Jk = np.linalg.matrix_power(J, k)
    pred = k * 0.99 ** (k - 1)   # predicted corner entry, k * lambda^(k-1)
    print(f'k = {k:3d}   ||J^k|| = {np.linalg.norm(Jk, 2):8.3f}   '
          f'k * 0.99^(k-1) = {pred:8.3f}')
```

```
k =   1   ||J^k|| =    1.609   k * 0.99^(k-1) =    1.000
k =  50   ||J^k|| =   30.568   k * 0.99^(k-1) =   30.556
k = 100   ||J^k|| =   36.977   k * 0.99^(k-1) =   36.973
k = 200   ||J^k|| =   27.067   k * 0.99^(k-1) =   27.067
k = 800   ||J^k|| =    0.260   k * 0.99^(k-1) =    0.260
```

Every eigenvalue of $\mathbf{J}$ is 0.99, yet the norm of its powers climbs to about 37 at $k = 100$ and is still 27 at $k = 200$ before the factor 0.99^k dominates. The eigenvalues alone do not reveal this transient growth, to which Section 33.4.2.3 returns.

Jordan blocks are one way in which a real matrix falls short of a real eigenbasis: it has too few eigenvectors. The second way is different in kind. A real matrix can have no real eigenvector at all in some plane, because it rotates that plane.

33.3.1.6 Complex Eigenvalues

Consider the planar rotation by an angle θ (Section 33.2.1.3),

$$\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}.$$

If θ is not a multiple of π, $\mathbf{R}_\theta$ turns every nonzero real vector, so it has no real eigenvector. Its characteristic polynomial $\lambda^2 - 2\cos\theta\,\lambda + 1$ has the complex conjugate roots $\lambda = \cos\theta \pm i\sin\theta = e^{\pm i\theta}$. The modulus $|\lambda| = 1$ records that $\mathbf{R}_\theta$ preserves lengths, and the argument $\pm\theta$ records the angle.

The chapter's running 2×2 matrix has no visible rotation in its entries:

$$\mathbf{A} = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \qquad \operatorname{tr}\mathbf{A} = 4, \quad \det\mathbf{A} = 5.$$

Its characteristic polynomial is $\lambda^2 - 4\lambda + 5$, with roots $\lambda = 2 \pm i = \sqrt{5}\,e^{\pm i\theta}$ and $\theta = \arctan\frac{1}{2} \approx 26.6^\circ$. The eigenvector for $2 + i$ comes from the second row of $\mathbf{A} - (2+i)\mathbf{I}$, which reads $-w_1 + (1-i)w_2 = 0$. With $w_2 = 1$,

$$\mathbf{w} = \begin{pmatrix} 1-i \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}_{\mathbf{u}} + i\,\underbrace{\begin{pmatrix} -1 \\ 0 \end{pmatrix}}_{\mathbf{v}}.$$

Where is the rotation hidden? Take any real matrix with an eigenpair $\mathbf{A}\mathbf{w} = \lambda\mathbf{w}$, $\lambda = a + ib$, $b \neq 0$, and split the eigenvector into real and imaginary parts, $\mathbf{w} = \mathbf{u} + i\mathbf{v}$. Expanding $\mathbf{A}(\mathbf{u} + i\mathbf{v}) = (a + ib)(\mathbf{u} + i\mathbf{v})$ and matching real and imaginary parts gives

$$\mathbf{A}\mathbf{u} = a\mathbf{u} - b\mathbf{v}, \qquad \mathbf{A}\mathbf{v} = b\mathbf{u} + a\mathbf{v}.$$

Both images land back in $\operatorname{span}\{\mathbf{u},\mathbf{v}\}$, so this span is an invariant subspace. It is a plane. If $\mathbf{u}$ and $\mathbf{v}$ were dependent, their span would be a line (not zero, since $\mathbf{w} \neq \mathbf{0}$) mapped into itself, on which $\mathbf{A}$ multiplies by a real number μ. Then $\mathbf{Aw} = \mu\mathbf{w}$, and $\lambda = \mu$ would be real, contradicting $b \neq 0$. Collecting the two identities as columns with $\mathbf{B} = [\mathbf{v} \quad \mathbf{u}]$ gives $\mathbf{AB} = \mathbf{B}\left(\begin{smallmatrix} a & -b \\ b & a \end{smallmatrix}\right)$, that is,

$$\mathbf{B}^{-1}\mathbf{A}\mathbf{B} = \begin{pmatrix} a & -b \\ b & a \end{pmatrix} = \underbrace{\sqrt{a^2+b^2}}_{r} \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \qquad \theta = \arg(a+ib). \tag{33.33}$$

In the basis $(\mathbf{v},\mathbf{u})$ of its invariant plane, the matrix is a rotation–scaling block $r\mathbf{R}_\theta$: it scales by the modulus and rotates by the argument. The block does not depend on the scale or phase of $\mathbf{w}$, since the derivation used only $\mathbf{Aw} = \lambda\mathbf{w}$. The correspondence also reads from right to left. Identify $(x,y) \in \mathbb{R}^2$ with the complex number $x+iy$. The block maps (x,y) to $(ax-by,\ bx+ay)$, which is multiplication by $a+ib$.

For the running matrix, $\mathbf{B} = \left(\begin{smallmatrix} -1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ and $\mathbf{B}^{-1}\mathbf{AB} = \left(\begin{smallmatrix} 2 & -1 \\ 1 & 2 \end{smallmatrix}\right)$: scale by $\sqrt{5}$ and rotate by 26.6°. If that were the whole description, every step would lengthen a vector by the factor $\sqrt{5} \approx 2.24$. We compute the block from the eigenvector that `eig` returns and then follow the lengths of the iterates $\mathbf{A}^k(1,0)$.

```
A = np.array([[1., 2.], [-1., 3.]])
lam, V = np.linalg.eig(A)
j = np.argmax(lam.imag)                  # the eigenvalue a + ib with b > 0
r, theta = np.abs(lam[j]), np.angle(lam[j])
u, v = V[:, j].real, V[:, j].imag
B = np.column_stack([v, u])              # basis (v, u) of the invariant plane
print(f'lambda = {lam[j]:.4f}   r = {r:.4f}   theta = {theta:.4f}')
print('B^-1 A B =\n', np.round(np.linalg.inv(B) @ A @ B, 4))
xs = [np.array([1., 0.])]
for _ in range(8):
    xs.append(A @ xs[-1])
norms = np.linalg.norm(xs, axis=1)
print('norm ratios:', np.round(norms[1:] / norms[:-1], 2))
x20 = np.linalg.matrix_power(A, 20) @ np.array([1., 0.])
print(f'average rate ||A^20 x||^(1/20) = {np.linalg.norm(x20) ** (1 / 20):.3f}')
```

```
lambda = 2.0000+1.0000j   r = 2.2361   theta = 0.4636
B^-1 A B =
 [[ 2. -1.]
 [ 1.  2.]]
norm ratios: [1.41 2.92 3.45 2.76 2.27 1.88 1.5  1.54]
average rate ||A^20 x||^(1/20) = 2.252
```

The block is $\left(\begin{smallmatrix} 2 & -1 \\ 1 & 2 \end{smallmatrix}\right)$, whatever eigenvector `eig` returned. The lengths, however, do not grow by 2.24 per step. Along the iterates $(1,0) \to (1,-1) \to (-1,-4) \to (-9,-11) \to (-31,-24)$ the ratios swing between about 1.4 and 3.5. The scaling is uniform only in the coordinates $(\mathbf{v},\mathbf{u})$, which are not orthonormal. In the original coordinates the iterates spiral outward along stretched ellipses, and a full turn takes $2\pi/\theta \approx 13.6$ steps. Only the average rate $\|\mathbf{A}^k\mathbf{x}\|^{1/k}$ settles at $\sqrt{5}$ (it is 2.25 at $k = 20$). The motion is a rigid rotation and scaling only when the matrix is normal, a property defined in Section 33.4.2.2.

Every real square matrix is built, up to a real change of basis, from the three behaviours we have now seen, alone or combined: stretch along a line, shear within a Jordan block, and rotation with scaling in a plane. This is the *real Jordan form* (Horn and Johnson 2012).

33.3.2 Symmetric Matrices and Quadratic Forms

33.3.2.1 The Spectral Theorem

For which matrices is a real, orthonormal eigenbasis guaranteed? Symmetry rules out both failures we have met, too few eigenvectors and complex ones. The symmetric $\mathbf{S}$ of Section 1.3 shows what to expect. Its unit eigenvectors $(1,1)/\sqrt{2}$ and $(1,-1)/\sqrt{2}$ are perpendicular, and with them

$$\mathbf{S} = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} = \underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}_{\mathbf{V}} \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \underbrace{\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}}_{\mathbf{V}^{\top}}.$$

The eigenvector matrix is orthogonal, $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$, so its inverse is its transpose. The spectral theorem states that this holds for every symmetric matrix. It is the theorem of this chapter: positive definiteness, the Rayleigh quotient, and all of Section 33.5 rest on it.

Theorem (spectral theorem). *Every real symmetric matrix $\mathbf{A}$ has real eigenvalues and an orthonormal basis of real eigenvectors. Equivalently,*

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{\top}, \tag{33.34}$$

where $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ is real and $\mathbf{V}$ is orthogonal.

We prove three claims in turn: the eigenvalues are real, eigenvectors of distinct eigenvalues are orthogonal, and an orthonormal eigenbasis exists. For complex vectors we use the inner product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^{*}\mathbf{y}$, where $\mathbf{x}^{*}$ is the conjugate transpose; on real vectors it is the dot product. The only fact we borrow is that the characteristic polynomial has at least one complex root (the fundamental theorem of algebra), so at least one eigenpair exists over $\mathbb{C}$. Parts (i) and (ii) are two-line computations; the work is in part (iii).

Proof, part (i): the eigenvalues are real. Let $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$, allowing complex entries. Since $\mathbf{A}$ is real and symmetric, $\mathbf{A}^{*} = \mathbf{A}^{\top} = \mathbf{A}$, so

$$\lambda\, \mathbf{v}^{*}\mathbf{v} = \mathbf{v}^{*}\mathbf{A}\mathbf{v} = (\mathbf{A}\mathbf{v})^{*}\mathbf{v} = \bar{\lambda}\, \mathbf{v}^{*}\mathbf{v}.$$

Because $\mathbf{v}^{*}\mathbf{v} = \|\mathbf{v}\|^2 > 0$, we may divide by it and obtain $\lambda = \bar{\lambda}$: the eigenvalue is real. The eigenvector can then be taken real as well, since it solves the real system $(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$. $\blacksquare$

Proof, part (ii): eigenvectors of distinct eigenvalues are orthogonal. Let $\mathbf{A}\mathbf{u} = \lambda\mathbf{u}$ and $\mathbf{A}\mathbf{v} = \mu\mathbf{v}$ with $\lambda \neq \mu$, both real by (i), and $\mathbf{u}, \mathbf{v}$ real. Using $\mathbf{A} = \mathbf{A}^{\top}$ to move $\mathbf{A}$ from one side of the inner product to the other,

$$\lambda\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{A}\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{v} \rangle = \mu\langle \mathbf{u}, \mathbf{v} \rangle,$$

so $(\lambda - \mu)\langle \mathbf{u}, \mathbf{v} \rangle = 0$, and since $\lambda \neq \mu$, $\langle \mathbf{u}, \mathbf{v} \rangle = 0$. $\blacksquare$

Proof, part (iii): an orthonormal eigenbasis. We induct on n; for $n = 1$ there is nothing to prove. For $n > 1$, part (i) gives a real eigenpair $(\lambda_1, \mathbf{v}_1)$ with $\|\mathbf{v}_1\| = 1$. Let $U = \mathbf{v}_1^{\perp}$ be the orthogonal complement, of dimension $n - 1$. The matrix maps U into itself: for any $\mathbf{x} \in U$,

$$\langle \mathbf{A}\mathbf{x}, \mathbf{v}_1 \rangle = \langle \mathbf{x}, \mathbf{A}\mathbf{v}_1 \rangle = \lambda_1 \langle \mathbf{x}, \mathbf{v}_1 \rangle = 0.$$

Let the columns of $\mathbf{Q} \in \mathbb{R}^{n \times (n-1)}$ be an orthonormal basis of U. Since the columns of $\mathbf{AQ}$ lie in U, we can write $\mathbf{AQ} = \mathbf{QC}$ with $\mathbf{C} = \mathbf{Q}^{\top}\mathbf{AQ}$, the matrix of $\mathbf{A}$ restricted to U. It is symmetric, $\mathbf{C}^{\top} = \mathbf{Q}^{\top}\mathbf{A}^{\top}\mathbf{Q} = \mathbf{C}$, and of size $n - 1$, so by the inductive hypothesis $\mathbf{C} = \mathbf{V}'\Lambda'\mathbf{V}'^{\top}$ with $\mathbf{V}'$ orthogonal. The columns of $\mathbf{QV}'$ are orthonormal, since $(\mathbf{QV}')^{\top}\mathbf{QV}' = \mathbf{V}'^{\top}\mathbf{V}' = \mathbf{I}$. They lie in U, hence are orthogonal to $\mathbf{v}_1$. They are also eigenvectors, since $\mathbf{AQV}' = \mathbf{QCV}' = \mathbf{QV}'\Lambda'$. Together with $\mathbf{v}_1$ they form the orthogonal matrix $\mathbf{V} = [\mathbf{v}_1 \ \mathbf{QV}']$, and $\mathbf{AV} = \mathbf{V}\Lambda$. ■

The factorization Equation 33.34 reads as three steps. First $\mathbf{V}^{\top}$ turns the orthonormal eigenvectors onto the coordinate axes. This is a rotation, possibly combined with a reflection; for $\mathbf{S}$ above, $\mathbf{V}$ has determinant -1. Then Λ stretches each axis by its eigenvalue, and $\mathbf{V}$ turns the axes back. This is the left panel of Figure 33.14, now proved for every symmetric matrix.

Every matrix $\mathbf{A}$, square or not, yields a symmetric matrix $\mathbf{A}^{\top}\mathbf{A}$. Its quadratic form $\mathbf{x}^{\top}\mathbf{A}^{\top}\mathbf{A}\mathbf{x} = \|\mathbf{A}\mathbf{x}\|^2$ is never negative, a property that the next subsection names and tests.

33.3.2.2 Positive Definiteness

Is the quadratic form $\mathbf{x}^{\top}\mathbf{A}\mathbf{x}$ ever negative? The question decides whether a critical point of a loss is a minimum, whether a matrix can be a covariance, and whether a least-squares problem has a unique solution. A symmetric matrix $\mathbf{A}$ is *positive semidefinite* (PSD), written $\mathbf{A} \succeq 0$, if

$$\mathbf{x}^{\top}\mathbf{A}\mathbf{x} \geq 0 \quad \text{for all } \mathbf{x},$$

and *positive definite* (PD), written $\mathbf{A} \succ 0$, if the inequality is strict for every $\mathbf{x} \neq \mathbf{0}$. For example, $\mathbf{x}^{\top}\mathbf{S}\mathbf{x} = 2x_1^2 + 2x_1x_2 + 2x_2^2 = x_1^2 + x_2^2 + (x_1 + x_2)^2$ is positive unless $\mathbf{x} = \mathbf{0}$, so $\mathbf{S} \succ 0$, while $\left(\begin{smallmatrix} 1 & 0 \\ 0 & -1 \end{smallmatrix}\right)$ is neither, since $\mathbf{e}_2$ gives -1. Completing squares by hand does not scale, but the eigenvalues decide the question for every symmetric matrix.

Proposition (definiteness from eigenvalues). *For symmetric* $\mathbf{A}$*,*

$$\mathbf{A} \succeq 0 \iff \lambda_i \geq 0 \text{ for all } i, \qquad \mathbf{A} \succ 0 \iff \lambda_i > 0 \text{ for all } i.$$

Proof. Write $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{\top}$ by the spectral theorem and change variables to the eigencoordinates $\mathbf{y} = \mathbf{V}^{\top}\mathbf{x}$, a rotation (possibly with a reflection) that preserves lengths. Then

$$\mathbf{x}^{\top}\mathbf{A}\mathbf{x} = \mathbf{x}^{\top}\mathbf{V}\Lambda\mathbf{V}^{\top}\mathbf{x} = \mathbf{y}^{\top}\Lambda\mathbf{y} = \sum_{i=1}^{n} \lambda_i \, y_i^2 = \sum_{i=1}^{n} \lambda_i \, (\mathbf{v}_i^{\top}\mathbf{x})^2. \tag{33.35}$$

The quadratic form is a weighted sum of squares with the eigenvalues as weights. If every $\lambda_i \geq 0$, the sum is non-negative. Conversely, $\mathbf{x} = \mathbf{v}_j$ gives $\mathbf{y} = \mathbf{e}_j$ and $\mathbf{v}_j^{\top}\mathbf{A}\mathbf{v}_j = \lambda_j$, which

forces $\lambda_j \geq 0$. The PD statement is the same argument with strict inequalities, using that $\mathbf{y} \neq \mathbf{0}$ whenever $\mathbf{x} \neq \mathbf{0}$. ■

For $\mathbf{S}$ and $\mathbf{x} = \mathbf{e}_1$, the eigen-coordinates are $\mathbf{y} = \mathbf{V}^\top \mathbf{e}_1 = (1/\sqrt{2}, 1/\sqrt{2})$, and Equation 33.35 gives $3 \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} = 2$, which is $\mathbf{e}_1^\top \mathbf{S} \mathbf{e}_1 = s_{11}$. Three consequences cover most places where definiteness appears in deep learning.

- **PD implies invertible.** If $\mathbf{A} \succ 0$ and $\mathbf{A}\mathbf{x} = \mathbf{0}$, then $\mathbf{x}^\top \mathbf{A}\mathbf{x} = 0$, which forces $\mathbf{x} = \mathbf{0}$.
- **Gram and covariance matrices are PSD.** For any matrix $\mathbf{X}$, $\mathbf{x}^\top \mathbf{X}^\top \mathbf{X}\mathbf{x} = \|\mathbf{X}\mathbf{x}\|^2 \geq 0$, so $\mathbf{X}^\top \mathbf{X} \succeq 0$, and it is PD if and only if $\mathbf{X}$ has full column rank (so that $\mathbf{X}\mathbf{x} = \mathbf{0}$ forces $\mathbf{x} = \mathbf{0}$). The sample covariance of centred data is such a Gram matrix divided by the number of samples, so it is always PSD.
- **A PD Hessian gives a strict local minimum.** Let f be twice continuously differentiable with $\nabla f(\mathbf{x}^*) = \mathbf{0}$, and let its Hessian $\nabla^2 f(\mathbf{x}^*)$, the symmetric matrix of second partial derivatives, be PD. For a small displacement δ, the second-order Taylor expansion of Section 34.2.3.2 gives $f(\mathbf{x}^* + \delta) = f(\mathbf{x}^*) + \frac{1}{2}\delta^\top \nabla^2 f(\mathbf{x}^*)\delta + o(\|\delta\|^2)$, and by Equation 33.35 the quadratic term is at least $\frac{1}{2}\lambda_{\min}\|\delta\|^2 > 0$. So $\mathbf{x}^*$ is a strict local minimum. The condition is sufficient, not necessary: $f(x) = x^4$ has a strict minimum at 0 with Hessian 0.

Positive definiteness also has an algorithmic certificate, the *Cholesky factorization*: $\mathbf{A} \succ 0$ if and only if $\mathbf{A} = \mathbf{L}\mathbf{L}^\top$ for a lower-triangular $\mathbf{L}$ with positive diagonal. One direction is immediate: such an $\mathbf{L}$ is invertible, so $\mathbf{x}^\top \mathbf{A}\mathbf{x} = \|\mathbf{L}^\top \mathbf{x}\|^2 > 0$ for $\mathbf{x} \neq \mathbf{0}$. Attempting the factorization is the standard numerical test for definiteness, and it is cheaper than computing the eigenvalues. The same factor turns independent standard Gaussian samples $\mathbf{z}$ into samples $\mathbf{L}\mathbf{z}$ with covariance $\mathbf{L}\mathbf{L}^\top = \mathbf{A}$.

Geometrically, the signs of the eigenvalues set the shape of the surface $z = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ (Figure 33.17): a bowl when all are positive, a trough that is flat along the eigenvector of a zero eigenvalue, and a saddle for mixed signs.

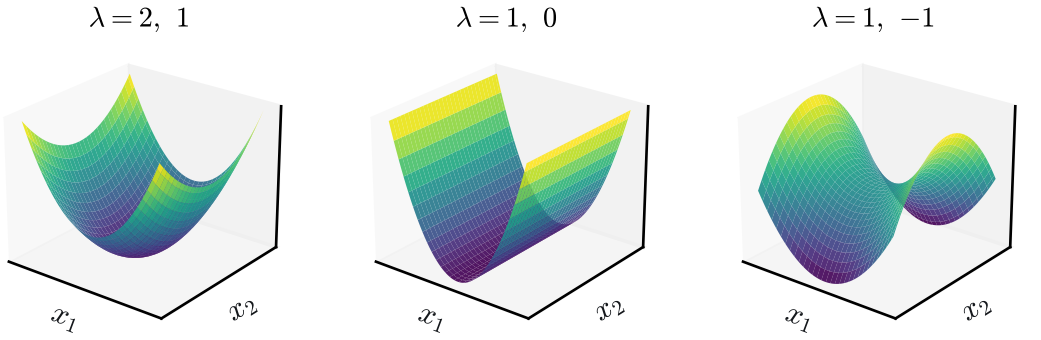


Figure 33.17: The quadratic-form surface $z = \mathbf{x}^\top \mathbf{A}\mathbf{x}$ for three diagonal matrices takes its shape from the eigenvalue signs: a bowl when all $\lambda_i > 0$ (positive definite), a trough when one $\lambda_i = 0$ (positive semidefinite), and a saddle for mixed signs (indefinite).

The rank condition for Gram matrices can be read off their eigenvalues. Take a tall $\mathbf{X}$ with independent columns, whose $\mathbf{X}^\top \mathbf{X}$ should be PD, and one whose second column is twice the first, for which it should be PSD with a zero eigenvalue.

```
X_full = np.array([[1., 0.], [1., 1.], [0., 1.], [2., 1.]])  # rank 2
X_dep = np.array([[1., 2.], [1., 2.], [0., 0.], [2., 4.]])   # col 2 = 2*col 1
for X, name in [(X_full, 'full column rank'), (X_dep, 'dependent columns')]:
    eigvals = np.linalg.eigvalsh(X.T @ X).round(4)
    eigvals[np.abs(eigvals) < 1e-9] = 0.0  # clean up tiny round-off / -0.0
    print(f'{name}: eigenvalues of X^T X = {eigvals}')
```

```
full column rank: eigenvalues of X^T X = [1.1459 7.8541]
dependent columns: eigenvalues of X^T X = [ 0. 30.]
```

Both eigenvalues are positive in the first case. In the second, one is zero: the form vanishes along $(2, -1)$, the direction that $\mathbf{X}$ maps to zero.

33.3.2.3 The Rayleigh Quotient

Identity Equation 33.35 does more than test signs: it identifies the largest and smallest eigenvalues as the solutions of optimization problems. Define the *Rayleigh quotient* of a symmetric $\mathbf{A}$,

$$R(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{A}\mathbf{x}}{\mathbf{x}^\top \mathbf{x}}, \qquad \mathbf{x} \neq \mathbf{0}.$$

It is the value of the quadratic form per unit squared length, and on an eigenvector it equals the eigenvalue. In this subsection the eigenvalues are ordered by their signed values, $\lambda_1 \geq \cdots \geq \lambda_n$, not by modulus.

Proposition (Rayleigh). *For symmetric* $\mathbf{A}$ *with eigenvalues* $\lambda_1 \geq \cdots \geq \lambda_n$,

$$\lambda_1 = \max_{\mathbf{x}\neq\mathbf{0}} R(\mathbf{x}), \qquad \lambda_n = \min_{\mathbf{x}\neq\mathbf{0}} R(\mathbf{x}),$$

attained at the corresponding eigenvectors.

Proof. Scaling $\mathbf{x}$ leaves R unchanged, so restrict to unit vectors. With $\mathbf{y} = \mathbf{V}^\top \mathbf{x}$ we have $\|\mathbf{y}\| = \|\mathbf{x}\| = 1$ and, by Equation 33.35, $R(\mathbf{x}) = \sum_i \lambda_i y_i^2$: an average of the eigenvalues with non-negative weights y_i^2 that sum to 1. An average lies between the smallest and the largest value, so $\lambda_n \leq R(\mathbf{x}) \leq \lambda_1$, and the bounds are attained by putting all weight on one coordinate, at $\mathbf{x} = \mathbf{v}_1$ or $\mathbf{x} = \mathbf{v}_n$. ■

For $\mathbf{S}$ this means that R ranges over $[1, 3]$, with the maximum at the eigenvector direction $45°$ and the minimum at $135°$. Every direction between the two extremes mixes the eigenvalues 1 and 3 in proportion to its squared eigen-coordinates.

Maximizing over subspaces recovers every eigenvalue in between: the *Courant–Fischer theorem* states that $\lambda_j = \max_{\dim U = j} \min_{\mathbf{0}\neq\mathbf{x}\in U} R(\mathbf{x})$, the best guaranteed stretch over all j-dimensional subspaces U (Exercise 4d asks for the proof and gives a hint).

Two uses of the extreme case recur in the book. First, for every matrix $\mathbf{A}$, the largest Rayleigh quotient of $\mathbf{A}^\top\mathbf{A}$ is $\max_{\mathbf{x}\neq\mathbf{0}} \|\mathbf{A}\mathbf{x}\|^2/\|\mathbf{x}\|^2$, the square of the largest factor by which $\mathbf{A}$ stretches a vector. Second, gradient descent on a convex quadratic $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ with $\mathbf{A} \succ 0$ updates $\mathbf{x} \leftarrow (\mathbf{I} - \eta\mathbf{A})\mathbf{x}$. By Equation 33.27 each eigen-coordinate is multiplied by $1 - \eta\lambda_i$ per step, so the iteration converges from every start if and only if $|1 - \eta\lambda_i| < 1$ for

all i, that is, $0 < \eta < 2/\lambda_{\max}$. The largest eigenvalue limits the step size (it is the smoothness constant L of Section 35.1), and at step sizes $\eta \le 1/\lambda_{\max}$ the smallest sets the slowest rate, $1 - \eta\lambda_{\min}$. Their ratio $\lambda_{\max}/\lambda_{\min}$ is the condition number of $\mathbf{A}$, which Section 33.5.3.2 defines for every matrix.

33.3.2.4 Sample Covariance and the Marchenko–Pastur Law

When we look for structure in data, we look at the large eigenvalues of its sample covariance. To tell which eigenvalues are large, we need to know what they are when the data have no structure at all. Draw n samples of pure noise in d dimensions, with independent coordinates of mean 0 and variance 1, so that the true covariance is $\mathbf{I}$, and stack them as the rows of $\mathbf{X} \in \mathbb{R}^{n\times d}$. Do the eigenvalues of the sample covariance $\mathbf{C} = \frac{1}{n}\mathbf{X}^\top\mathbf{X}$ sit near 1? For a unit vector $\mathbf{w}$, the Rayleigh quotient $\mathbf{w}^\top\mathbf{C}\mathbf{w}$ is the sample variance of the data along $\mathbf{w}$. The extreme eigenvalues of $\mathbf{C}$ are therefore the sample variances of the most and the least variable directions, the quantities that principal component analysis reads as structure (Section 33.5.2.2).

They do not sit near 1. Let $d, n \to \infty$ with the *aspect ratio* $c = d/n \le 1$ fixed. Then the histogram of the eigenvalues of $\mathbf{C}$ converges to a fixed density supported on the interval

$$\left[(1-\sqrt{c})^2,\ (1+\sqrt{c})^2\right],$$

the *Marchenko–Pastur bulk* (Marchenko and Pastur 1967), and with a finite fourth moment the extreme eigenvalues converge to its edges. For $c > 1$ the matrix has rank at most $n < d$, so in addition at least $d-n$ eigenvalues are zero. With four samples per dimension, $c = 1/4$, the noise eigenvalues already spread over $[0.25, 2.25]$: the sample variance along the most variable direction is more than twice the true variance, along the least variable one a quarter of it. At $n = 4000$ and $d = 1000$ the limit is already accurate.

```
n, d = 4000, 1000                       # aspect ratio c = d / n = 1/4
X = np.random.randn(n, d)               # pure noise: true covariance I
ev = np.linalg.eigvalsh(X.T @ X / n)
c = d / n
print(f'eigenvalues of C: min {ev.min():.3f}, max {ev.max():.3f}')
print(f'Marchenko-Pastur edges: {(1 - c ** 0.5) ** 2:.3f}, '
      f'{(1 + c ** 0.5) ** 2:.3f}')
```

```
eigenvalues of C: min 0.247, max 2.238
Marchenko-Pastur edges: 0.250, 2.250
```

On every run the smallest and largest eigenvalues land within a few hundredths of the predicted edges 0.25 and 2.25. An eigenvalue inside the bulk is therefore indistinguishable from noise. Only an eigenvalue clearly above the upper edge indicates a direction of real structure. How strong a direction must be to show up is the subject of Exercise 5.

Two things were taken for granted throughout this section: that the eigenvalues of a matrix can be computed, and that they say what its powers do after a few steps as well as after many. For large matrices, and for matrices far from symmetric, neither is safe. Section 33.4 takes up both.

33.3.3 Summary

In eigenvector coordinates a matrix acts as a list of numbers, $\mathbf{A}^k\mathbf{x} = \sum_i c_i \lambda_i^k \mathbf{v}_i$, which explains repeated application in a Markov chain, a deep linear network, or gradient descent on a quadratic. An eigenbasis exists over $\mathbb{C}$ if and only if every eigenvalue has as many independent eigenvectors as its multiplicity as a root; otherwise Jordan blocks add polynomial factors in k. Symmetric matrices always have real eigenvalues and an orthonormal eigenbasis. Their quadratic forms are weighted sums of squares, so the eigenvalue signs decide definiteness and the extreme eigenvalues are the extremes of the Rayleigh quotient. The eigenvalues of a sample covariance of pure noise spread over the Marchenko–Pastur interval, so only an eigenvalue above its upper edge is evidence of structure.

33.3.4 Exercises

1. **Trace, determinant, and definiteness.** The identities Equation 33.30 and Equation 33.31 fix both eigenvalues of a 2×2 matrix through their sum and product.
    1. Find the eigenvalues of $\left(\begin{smallmatrix} 4 & 1 \\ 1 & 4 \end{smallmatrix}\right)$ from its trace and determinant alone, without forming the characteristic polynomial. Then find its eigenvectors. Do the same for the surfer's matrix $\mathbf{P}$, whose trace is $5/4$ and determinant $1/4$.
    2. Show that a symmetric 2×2 matrix is positive definite if and only if its trace and its determinant are both positive. Which sign conditions on trace and determinant characterize positive semidefiniteness?
    3. Each of the matrices $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$, $\left(\begin{smallmatrix} 1 & 2 \\ 2 & 1 \end{smallmatrix}\right)$, and $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ is the Hessian of the quadratic $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ at its critical point $\mathbf{0}$. Use (b) to decide for each f whether the origin is a strict minimum, a minimum that is not strict, or a saddle point. For the saddle point, give a direction along which f decreases.
    4. Show that the Cholesky factor $\mathbf{L}$ of $\left(\begin{smallmatrix} a & b \\ b & c \end{smallmatrix}\right)$ has the diagonal entries $\sqrt{a}$ and $\sqrt{(ac - b^2)/a}$. Why do the conditions under which a factor with positive diagonal exists coincide with the test of (b)?

2. **Rank and zero eigenvalues.** The section shows that the rank of a diagonalizable matrix equals the number of its nonzero eigenvalues, counted with multiplicity.
    1. Find the rank and the eigenvalues of $\mathbf{N} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$. Which step of the argument for diagonalizable matrices fails for $\mathbf{N}$?
    2. Let g_0 be the geometric multiplicity of the eigenvalue 0 of an $n \times n$ matrix $\mathbf{A}$, with $g_0 = 0$ when 0 is not an eigenvalue. Show that $\operatorname{rank} \mathbf{A} = n - g_0$ for every square matrix. Conclude that the rank is at least the number of nonzero eigenvalues counted with algebraic multiplicity, with equality if and only if the eigenvalue 0 is not defective.
    3. The *singular values* of a matrix are $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^\top \mathbf{A})}$, the square roots of the eigenvalues of the PSD matrix $\mathbf{A}^\top \mathbf{A}$. Find the singular values of $\mathbf{N}$ and of the Jordan block $\mathbf{J}_m(0)$ of Section 33.3.1.5. Compare the number of nonzero singular values with the rank and with the number of nonzero eigenvalues.

3. **The centering matrix.** Let $\mathbf{1} \in \mathbb{R}^n$ be the all-ones vector and $\mathbf{H} = \mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top$.
    1. Show that $n\mathbf{I} - \mathbf{1}\mathbf{1}^\top$ is positive semidefinite by writing $\mathbf{x}^\top(n\mathbf{I} - \mathbf{1}\mathbf{1}^\top)\mathbf{x}$ as a sum of squares of the differences $x_i - x_j$. Then find all eigenvalues of $n\mathbf{I} - \mathbf{1}\mathbf{1}^\top$ and their eigenvectors.

2. Show that $\mathbf{H}$ is symmetric with $\mathbf{H}^2 = \mathbf{H}$, and deduce its eigenvalues and its rank from (a).
3. Let the rows of $\mathbf{X} \in \mathbb{R}^{n \times d}$ be n samples. Subtracting the sample mean from every row gives $\mathbf{HX}$, so that the sample covariance is $\mathbf{C} = \frac{1}{n}\mathbf{X}^\top \mathbf{HX}$. Show that $\mathbf{C} \succeq 0$ and rank $\mathbf{C} \le n - 1$. How many zero eigenvalues does the sample covariance of 512-dimensional embeddings computed from a minibatch of 256 examples have at least? This is the case $c = d/n > 1$ of Section 33.3.2.4. Which operations on $\mathbf{C}$ does it rule out?

Adapted from Gilbert Strang, MIT 18.065 Problem Sets, Problems for Lecture 13, Problem 4.

4. code **Accuracy of computed eigenvalues.** ⊠ The algorithm behind `np.linalg.eigvals` is backward stable: it returns the exact eigenvalues of a matrix $\mathbf{A} + \mathbf{E}$ with $\|\mathbf{E}\|_2$ of the order of $\epsilon_{\text{mach}}\|\mathbf{A}\|_2$, where $\epsilon_{\text{mach}} \approx 2.2 \times 10^{-16}$ in double precision (Golub and Van Loan 1996). Exercise 1 shows how strongly such a perturbation can move a defective eigenvalue.
   1. Let $\mathbf{e}_1, \dots, \mathbf{e}_m$ be the standard basis vectors. Show that $\mathbf{J}_m(\lambda) + \varepsilon\, \mathbf{e}_m \mathbf{e}_1^\top$, which places ε in the lower-left corner of the Jordan block, has the eigenvalues $\lambda + \varepsilon^{1/m}\omega$, where ω ranges over the m-th roots of unity.
   2. For $m = 1, \dots, 4$, form $\mathbf{Q}\mathbf{J}_m(2)\mathbf{Q}^\top$ with an orthogonal $\mathbf{Q}$ taken from the QR factorization of a Gaussian matrix, and record the median over 20 draws of $\max_i |\widehat{\lambda}_i - 2|$ for the computed eigenvalues $\widehat{\lambda}_i$. Compare with $\epsilon_{\text{mach}}^{1/m}$. Repeat with the symmetric matrix $\mathbf{Q}(2\mathbf{I})\mathbf{Q}^\top$.
   3. Let $\mathbf{A}$ and $\mathbf{E}$ be symmetric. Use the Rayleigh proposition of Section 33.3.2.3 to show that $\lambda_{\max}(\mathbf{A} + \mathbf{E}) \le \lambda_{\max}(\mathbf{A}) + \|\mathbf{E}\|_2$ and $\lambda_{\min}(\mathbf{A} + \mathbf{E}) \ge \lambda_{\min}(\mathbf{A}) - \|\mathbf{E}\|_2$. How does this explain the symmetric result of (b)?
   4. Prove the Courant–Fischer theorem $\lambda_j = \max_{\dim U = j} \min_{\mathbf{0} \neq \mathbf{x} \in U} R(\mathbf{x})$ for symmetric $\mathbf{A}$ with $\lambda_1 \ge \cdots \ge \lambda_n$. For the upper bound, intersect U with $\operatorname{span}(\mathbf{v}_j, \dots, \mathbf{v}_n)$, which has dimension $n - j + 1$, so the intersection is at least a line; for the lower bound take $U = \operatorname{span}(\mathbf{v}_1, \dots, \mathbf{v}_j)$. Deduce Weyl's inequality $|\lambda_j(\mathbf{A} + \mathbf{E}) - \lambda_j(\mathbf{A})| \le \|\mathbf{E}\|_2$ for every j, which extends (c) to all eigenvalues.

5. code **Spikes above the bulk.** ⊠ Section 33.3.2.4 treats the Marchenko–Pastur bulk as the range of pure-noise eigenvalues. Suppose that the data contain one real direction: the n samples have independent Gaussian coordinates of variance 1, except the first coordinate, whose variance is $1 + \beta$. For $n \to \infty$ with $c = d/n$ fixed, Baik and Silverstein (2006) show that the largest eigenvalue of $\mathbf{C} = \frac{1}{n}\mathbf{X}^\top \mathbf{X}$ converges to

$$\lambda_{\max} \to \begin{cases} (1 + \sqrt{c})^2, & \beta \le \sqrt{c}, \\ (1 + \beta)\,(1 + c/\beta)\,, & \beta > \sqrt{c}. \end{cases}$$

   1. For $n = 2000$ and $d = 500$, compute the largest eigenvalue of $\mathbf{C}$ for $\beta \in \{0, 0.3, 0.5, 0.7, 1, 2\}$, averaged over five draws, and compare with the prediction.
   2. Fix $d = 500$ and $\beta = 0.4$. Use the formula to find the sample size n above which the largest eigenvalue leaves the bulk. Then increase the number of samples through $n \in \{500, 1000, 2000, 4000, 8000\}$, and compare the largest eigenvalue with the bulk edge, with the prediction, and with the largest eigenvalue for $\beta = 0$ at the same n. Why is the separation hard to see at $n = 4000$?

3. A practitioner keeps the principal components whose eigenvalues exceed the bulk edge. How large must the variance of a direction be at $c = 1/4$ for it to be kept? How many samples per dimension are needed to detect a direction whose variance exceeds the others by 10%?

33.4 Spectra in Practice

The matrices of Section 33.3 were small enough to solve by hand, and we trusted what their eigenvalues told us: the surfer's error shrank by the factor $1/4$ at every click, as the second eigenvalue said it would. Both comforts end outside the textbook. The link matrix of the web has billions of rows, and nobody writes down its characteristic polynomial. And eigenvalues promise less than they seem to. A Jordan block whose eigenvalues are both 0.99 grew to norm 37 before it decayed (Section 33.3.1.5), and backpropagation through a recurrent network multiplies the gradient by fifty different matrices, where the surfer applied one matrix fifty times.

This section takes the two problems in turn. First, what can we learn about the eigenvalues of a matrix that is too large to factor? Surprisingly much: the entries alone confine the eigenvalues to a few discs, and repeated multiplication by the matrix, the one operation that stays cheap, finds the largest eigenvalue and its eigenvector. PageRank was computed this way. Second, what do eigenvalues leave unsaid about a finite number of steps, or about a product of different matrices? Here the answer is sobering. Eigenvalues describe the long run of the powers of one matrix. For anything else we need norms, and in the end the singular values of Section 33.5.

The topics are not of equal weight. Three recur in deep learning. Power iteration (Section 33.4.1.2) estimates the largest stretch of a layer for spectral normalization (Section 33.5.4.3). Transient growth and products of Jacobians (Section 33.4.2.3, Section 33.4.2.5) are the mathematics of the exploding and vanishing gradients of Section 8.6. The random matrices of Section 33.4.2.6 are the weight matrices of Section 4.4 at initialization. Gershgorin's theorem and PageRank are classical, and deep learning uses them less often; a reader in a hurry may skim both. Pseudospectra (Section 33.4.2.4) are a diagnostic that few readers will compute, and can be skipped as well. The computations use NumPy.

```
import numpy as np
```

33.4.1 Locating and Computing Eigenvalues

33.4.1.1 Gershgorin Discs

Can we say where the eigenvalues of a matrix lie without computing them? For a diagonal matrix they are the diagonal entries. If the off-diagonal entries are small, the eigenvalues should stay near the diagonal, and the following theorem says how near.

Theorem (Gershgorin (Gershgorin 1931)**).** *Let $\mathbf{A} = (a_{ij})$ be any $n \times n$ matrix and let $r_i = \sum_{j \neq i} |a_{ij}|$ be the sum of the absolute values of the off-diagonal entries in row i. Let $\mathcal{D}_i$ be the disc in the complex plane with centre a_{ii} and radius r_i. Then every eigenvalue of $\mathbf{A}$ lies in the union $\bigcup_i \mathcal{D}_i$.*

Proof. Let $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$, and pick the index i of an entry of largest magnitude, so that $|v_i| \geq |v_j|$ for all j and $v_i \neq 0$. Row i of $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ reads

$$\sum_{j} a_{ij} v_j = \lambda v_i \quad \Longrightarrow \quad (\lambda - a_{ii})\, v_i = \sum_{j \neq i} a_{ij} v_j .$$

Divide by v_i and take absolute values; since each $|v_j / v_i| \leq 1$,

$$|\lambda - a_{ii}| = \Big| \sum_{j \neq i} a_{ij} \, \tfrac{v_j}{v_i} \Big| \leq \sum_{j \neq i} |a_{ij}| = r_i ,$$

so $\lambda \in \mathcal{D}_i$. $\blacksquare$

The proof places each eigenvalue in the disc of the row where its eigenvector is largest, which we do not know in advance; the theorem therefore guarantees only the union. Consider the non-symmetric matrix

$$\mathbf{A} = \begin{pmatrix} 4 & 1 & 0 \\ -1 & 4 & 0.5 \\ 0.5 & 0 & 1 \end{pmatrix} . \tag{33.36}$$

The discs have centres 4, 4, 1 and radii 1, 1.5, 0.5. The two discs centred at 4 overlap (one contains the other), and the disc centred at 1 is disjoint from them, since $|4 - 1| = 3 > 1.5 + 0.5$. The eigenvalues are $3.99 \pm 0.96i$ and 1.03: the complex pair lies inside the two discs centred at 4, and the real eigenvalue inside the small disc. Figure 33.18 draws the discs and the eigenvalues.

The split, two eigenvalues in the overlapping pair and one in the small disc, is not a coincidence. *If a union of m discs is connected and disjoint from the other discs, it contains exactly m eigenvalues*, counted with multiplicity, because eigenvalues move continuously as the off-diagonal entries grow from zero and cannot jump the gap between separated groups of discs (Horn and Johnson 2012). The cell computes the centres, radii and eigenvalues quoted above.

```
A = np.array([[4.0, 1.0, 0.0],
              [-1.0, 4.0, 0.5],
              [0.5, 0.0, 1.0]])
centres = np.diag(A)
radii = np.abs(A).sum(axis=1) - np.abs(centres)
print('centres:', centres, '  radii:', radii)
for z in np.linalg.eigvals(A):
    print(f'eigenvalue {z:.3f}   distances to the centres: '
          f'{np.abs(z - centres).round(2)}')
```

```
centres: [4. 4. 1.]   radii: [1.  1.5 0.5]
eigenvalue 3.987+0.961j   distances to the centres: [0.96 0.96 3.14]
eigenvalue 3.987-0.961j   distances to the centres: [0.96 0.96 3.14]
eigenvalue 1.025+0.000j   distances to the centres: [2.97 2.97 0.03]
```

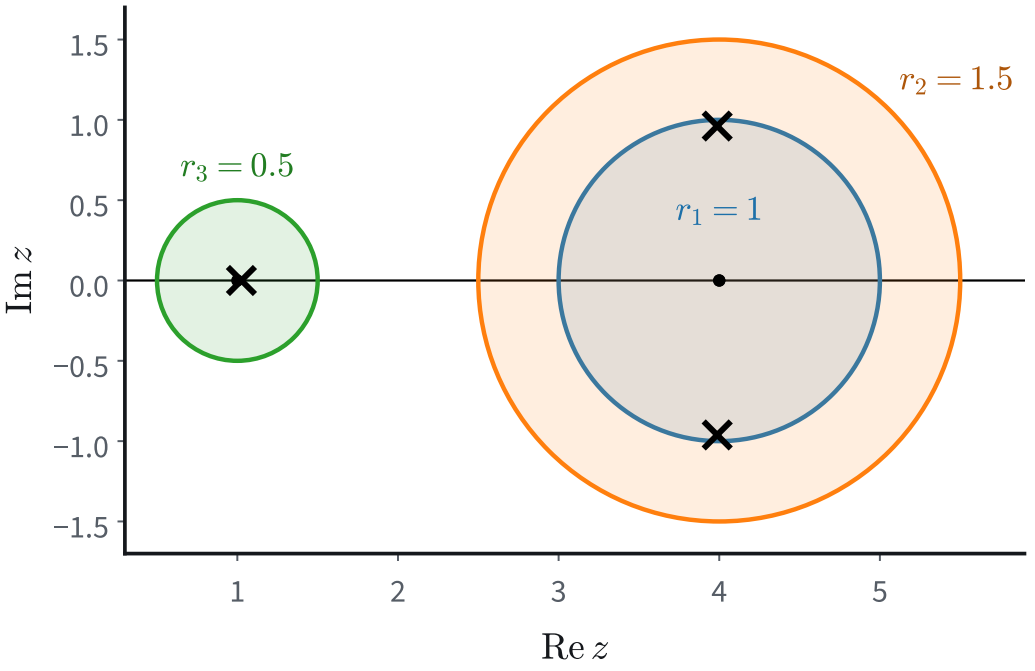


Figure 33.18: Gershgorin discs of the 3×3 matrix in the text (centres at the diagonal entries, radii the sums of the absolute off-diagonal entries of each row) and its eigenvalues (crosses). The two discs centred at 4 overlap and together hold two eigenvalues, the complex pair; the disjoint disc centred at 1 holds the third.

The complex pair is at distance 0.96 from 4, close to the rim of the disc of radius 1: the theorem is nearly sharp here.

A corollary needs no computation at all. If $\mathbf{A}$ is *strictly diagonally dominant* with positive diagonal, $a_{ii} > r_i$ for every i, then every disc lies in the open right half-plane, 0 is not an eigenvalue, and $\mathbf{A}$ is invertible. If such a matrix is also symmetric, its eigenvalues are real and positive, so it is positive definite.

33.4.1.2 Power Iteration

A matrix with a billion rows cannot be factored, but it can be multiplied by a vector if it is sparse. What does $\mathbf{A}^k\mathbf{v}_0$ tell us about its eigenvalues? The expansion Equation 33.27 answers this. Let $\mathbf{A}$ be diagonalizable with eigenvectors $\mathbf{v}_i$, ordered by modulus with a *strictly dominant* eigenvalue, $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. (For a real matrix, strict dominance forces λ_1 to be real, since a complex eigenvalue would share its modulus with its conjugate.) Write $\mathbf{v}_0 = \sum_i c_i\mathbf{v}_i$ and factor out λ_1^k:

$$\mathbf{A}^k\mathbf{v}_0 = \sum_{i=1}^{n} c_i\lambda_i^k\mathbf{v}_i = \lambda_1^k\Big(c_1\mathbf{v}_1 + \sum_{i\geq 2} c_i\Big(\tfrac{\lambda_i}{\lambda_1}\Big)^k\mathbf{v}_i\Big). \tag{33.37}$$

Every ratio $|\lambda_i/\lambda_1|$ is below 1, so if $c_1 \neq 0$ the bracket converges to $c_1\mathbf{v}_1$. The direction of $\mathbf{A}^k\mathbf{v}_0$ therefore approaches the eigenvector $\mathbf{v}_1$, with an error that decays like $|\lambda_2/\lambda_1|^k$, and the length grows by a factor approaching $|\lambda_1|$ per step. Multiplying repeatedly and rescaling to length 1 after each step, so that the numbers stay in range, is *power iteration* (Golub and Van Loan 1996). For a symmetric matrix, the Rayleigh quotient of the iterate estimates λ_1 and converges faster than the direction, at the squared rate $|\lambda_2/\lambda_1|^{2k}$. The eigenvectors are orthogonal, so a misalignment of size δ changes the Rayleigh quotient only by an amount of order δ^2 (Exercise 3).

Section 1.3 computed the iterates of $\mathbf{S}$ from $\mathbf{e}_1$: $(2,1)$, $(5,4)$, $(14,13)$, in general $((3^k+1)/2, (3^k-1)/2)$. The expansion explains them. With the unit eigenvectors $\mathbf{v}_1 = (1,1)/\sqrt{2}$ and $\mathbf{v}_2 = (1,-1)/\sqrt{2}$, we have $\mathbf{e}_1 = (\mathbf{v}_1+\mathbf{v}_2)/\sqrt{2}$, so

$$\mathbf{S}^k\mathbf{e}_1 = \frac{3^k\mathbf{v}_1 + \mathbf{v}_2}{\sqrt{2}}, \qquad \tan\angle(\mathbf{S}^k\mathbf{e}_1, \mathbf{v}_1) = 3^{-k}, \qquad 3 - R(\mathbf{S}^k\mathbf{e}_1) = \frac{2}{9^k+1}.$$

The direction error falls by the factor $|\lambda_2/\lambda_1| = 1/3$ per step: the sine of the angle to $\mathbf{v}_1$ is 0.316, 0.110, 0.037 for $k = 1, 2, 3$. The error of the Rayleigh quotient falls by $1/9$: it is 0.2, 0.024, 0.0027. Figure 33.19 plots both on a logarithmic scale.

Two conditions can fail. The start needs $c_1 \neq 0$; a random start satisfies this with probability one. And the dominance must be strict. If the dominant eigenvalues are a complex pair of equal modulus, the dominant part of the iterate keeps rotating. The ratio of consecutive norms then oscillates forever, as it did for the running matrix of Section 33.3.1.6. Even then, the average rate $\|\mathbf{A}^k\mathbf{v}_0\|^{1/k}$ converges to the largest modulus.

A matrix with positive entries is safe: it has a real, simple, strictly dominant eigenvalue, by the Perron–Frobenius theorem of Section 33.4.1.3. We run 200 steps of power iteration on a random 5×5 matrix with entries uniform in $[0,1]$, from a random start, and compare the final norm ratio with the largest eigenvalue modulus.

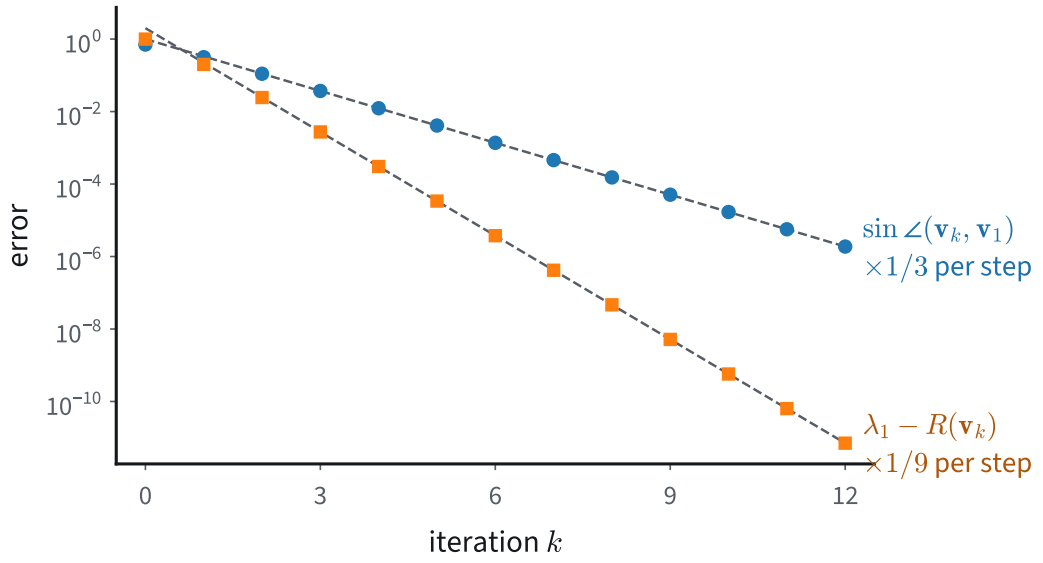


Figure 33.19: Power iteration on $\mathbf{S} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ from $\mathbf{e}_1$. The direction error falls by a factor of 3 per step and the error of the eigenvalue estimate by a factor of 9: the squared rate, because the eigenvectors of a symmetric matrix are orthogonal. Dashed lines: 3^{-k} and $2 \cdot 9^{-k}$.

```
A = np.random.rand(5, 5)                # positive entries
v = np.random.randn(5)
for _ in range(200):
    Av = A @ v
    ratio = np.linalg.norm(Av) / np.linalg.norm(v)
    v = Av / np.linalg.norm(Av)
mods = np.sort(np.abs(np.linalg.eigvals(A)))[::-1]
print(f'norm ratio after 200 steps = {ratio:.10f}')
print(f'largest |eigenvalue|       = {mods[0]:.10f}')
print(f'|lambda_2| / |lambda_1|    = {mods[1] / mods[0]:.3f}')
```

```
norm ratio after 200 steps = 2.5254088042
largest |eigenvalue|       = 2.5254088042
|lambda_2| / |lambda_1|    = 0.299
```

The norm ratio agrees with the largest eigenvalue modulus in every printed digit, on every run. One draw does not show how typical this is, so we compute the gap ratio $|\lambda_2/\lambda_1|$ for 10,000 such matrices. For contrast we also take 10,000 matrices with Gaussian entries, which have no reason to be safe, and count how often their dominant eigenvalues are a complex pair.

```
mods = np.sort(np.abs(np.linalg.eigvals(np.random.rand(10000, 5, 5))), axis=1)
print('uniform entries, gap ratio |lambda_2| / |lambda_1|, 5% and 95% quantiles:',
      np.quantile(mods[:, -2] / mods[:, -1], [0.05, 0.95]).round(2))
lam = np.linalg.eigvals(np.random.randn(10000, 5, 5))
top = lam[np.arange(10000), np.abs(lam).argmax(axis=1)]   # a dominant eigenvalue
```

```
print('Gaussian entries, fraction with a complex dominant pair:',
      np.mean(top.imag != 0).round(2))
```

```
uniform entries, gap ratio |lambda_2| / |lambda_1|, 5% and 95% quantiles: [0.14 0.37]
Gaussian entries, fraction with a complex dominant pair: 0.43
```

For uniform entries the gap ratio lies between about 0.14 and 0.37 in nine draws of ten, so 200 steps shrink the direction error far below rounding level. With Gaussian entries about four draws in ten have a complex dominant pair, and for those the norm ratio never settles.

33.4.1.3 PageRank

For non-negative matrices the dominant eigenvector is often the quantity we want, and power iteration is how it is computed. A non-negative matrix is *irreducible* if, in the graph with an edge $j \to i$ whenever $a_{ij} > 0$, every node reaches every other. The *Perron–Frobenius theorem* (Horn and Johnson 2012) states that such a matrix has a real, positive eigenvalue equal to the largest modulus of its eigenvalues, that this eigenvalue is simple, and that its eigenvector can be chosen with positive entries. If every entry of the matrix is positive, this eigenvalue is also strictly dominant: every other eigenvalue has smaller modulus.

Google's original PageRank (Brin and Page 1998) is such an eigenvector. Consider three pages with links $1 \to 2$, $2 \to 1$, and $2 \to 3$. A surfer follows a random outgoing link; page 3 has none, so we repair it by letting the surfer jump from page 3 to a uniformly random page. Here $\mathbf{P}$ is the repaired link matrix, column-stochastic with column j holding the click probabilities from page j:

$$\mathbf{P} = \begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{3} \\ 1 & 0 & \frac{1}{3} \\ 0 & \frac{1}{2} & \frac{1}{3} \end{pmatrix}.$$

The long-run fraction of time on each page is the stationary distribution π with $\mathbf{P}\pi = \pi$. By hand, rows 1 and 3 give $\pi_1 = \frac{1}{2}\pi_2 + \frac{1}{3}\pi_3 = \pi_3$, and then row 1 gives $\pi_2 = \frac{4}{3}\pi_3$; normalizing to sum 1 gives $\pi = (0.3, 0.4, 0.3)$. Page 2 ranks first because page 1 sends it every click, while page 2 splits its clicks between pages 1 and 3. The eigenvalues of $\mathbf{P}$ are 1, $-\frac{2}{3}$ and 0, so power iteration from any distribution converges at the rate $2/3$ per step.

Every column-stochastic matrix has 1 as an eigenvalue of largest modulus. First, because the columns sum to one, $\mathbf{1}^\top \mathbf{P} = \mathbf{1}^\top$, so 1 is an eigenvalue of $\mathbf{P}^\top$, and hence of $\mathbf{P}$: a matrix and its transpose have the same characteristic polynomial, since $\det \mathbf{X}^\top = \det \mathbf{X}$. Second, Gershgorin's theorem applied to $\mathbf{P}^\top$, whose rows are the columns of $\mathbf{P}$, puts every eigenvalue in a disc with centre $p_{ii} \geq 0$ and radius $1 - p_{ii}$, which lies inside the unit disc.

The web contains link cycles and groups of pages from which the surfer cannot escape. The repaired matrix can then fail to be irreducible or have several eigenvalues of modulus 1. A cycle of four pages has eigenvalues ± 1 and $\pm i$, and power iteration on it never settles (Exercise 6). PageRank therefore iterates the *damped* matrix

$$\mathbf{G} = \alpha \mathbf{P} + (1 - \alpha)\tfrac{1}{n}\mathbf{1}\mathbf{1}^\top, \qquad \alpha = 0.85,$$

which models a surfer who, at each click, jumps to a uniformly random page with probability 0.15. Every entry of $\mathbf{G}$ is positive, so Perron–Frobenius applies by construction. Haveliwala and Kamvar (2003) show that $|\lambda_2(\mathbf{G})| \le \alpha$, so the error of power iteration shrinks by at least the factor 0.85 per step, whatever the link structure. This bound is what made the computation feasible for the web: fifty sweeps reduce the error by a factor of more than 3000 ($0.85^{50} \approx 3 \times 10^{-4}$) on a graph of any size.

At web scale $\mathbf{P}$ is sparse and stored as one entry per link, and the dense $\mathbf{G}$ is never formed. For a distribution $\mathbf{p}$, $\mathbf{1}^\top \mathbf{p} = 1$ gives $\mathbf{G}\mathbf{p} = \alpha \mathbf{P}\mathbf{p} + (1-\alpha)\frac{1}{n}\mathbf{1}$, so each step needs one sparse product. The repair of pages without links is a rank-one term handled the same way.

33.4.2 Iteration and Stability

33.4.2.1 The Spectral Radius

Power iteration measured one number, the largest eigenvalue modulus. It has a name, the *spectral radius* $\rho(\mathbf{A}) = \max_i |\lambda_i|$, and it controls the long-run growth rate of the powers of one fixed matrix. Gelfand's formula makes this precise: for every square matrix and every matrix norm,

$$\lim_{k \to \infty} \|\mathbf{A}^k\|^{1/k} = \rho(\mathbf{A}).$$

The easy half, $\rho(\mathbf{A}) \le \|\mathbf{A}^k\|^{1/k}$ for every k, is Exercise 2. The formula says nothing about a fixed finite k, and we have already seen two matrices where that matters. The shear $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ has $\rho = 1$, yet its powers $\mathbf{T}^k = \left(\begin{smallmatrix} 1 & k \\ 0 & 1 \end{smallmatrix}\right)$ grow without bound, so $\rho \le 1$ does not bound the iterates. The Jordan block $\mathbf{J}_2(0.99)$ has $\rho < 1$, yet its powers reach norm 37 first, so $\rho < 1$ does not bound the transient. When do the eigenvalues control every step, not only the limit?

33.4.2.2 Normal Matrices

Both counterexamples, the shear and the Jordan block, have too few eigenvectors to form a basis. We therefore look first at the opposite extreme, matrices with an orthonormal basis of eigenvectors. Such a matrix acts on perpendicular directions independently, so its eigenvalues should tell the whole story. A matrix is *normal* when it commutes with its transpose, $\mathbf{A}^\top \mathbf{A} = \mathbf{A}\mathbf{A}^\top$ (for complex matrices, with the conjugate transpose $\mathbf{A}^*$ in place of $\mathbf{A}^\top$). Symmetric matrices, orthogonal matrices, and the rotation–scaling blocks $r\mathbf{R}_\theta$ of Section 33.3.1.6 are normal. The running matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ is not: $\mathbf{A}\mathbf{A}^\top = \left(\begin{smallmatrix} 5 & 5 \\ 5 & 10 \end{smallmatrix}\right)$ while $\mathbf{A}^\top \mathbf{A} = \left(\begin{smallmatrix} 2 & -1 \\ -1 & 13 \end{smallmatrix}\right)$. This is why its iterates did not move on circles.

The *complex spectral theorem* (Horn and Johnson 2012) states that a matrix is normal if and only if it has an orthonormal eigenbasis over $\mathbb{C}$, $\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^*$ with $\mathbf{U}$ unitary ($\mathbf{U}^*\mathbf{U} = \mathbf{I}$). Its proof follows the induction of Section 33.3.2.1.

Write $\|\mathbf{X}\|_2 = \max_{\|\mathbf{x}\|=1} \|\mathbf{X}\mathbf{x}\|$ for the *spectral norm*, the largest factor by which $\mathbf{X}$ stretches a vector. Since a unitary change of basis preserves lengths, for a normal matrix

$$\|\mathbf{A}^k\|_2 = \|\mathbf{U}\Lambda^k\mathbf{U}^*\|_2 = \max_i |\lambda_i|^k = \rho(\mathbf{A})^k. \tag{33.38}$$

For a normal matrix the eigenvalues control the largest possible amplification at every step, not only its long-run rate. A diagonalizable matrix that is not normal has a non-orthogonal eigenvector matrix in $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$, and submultiplicativity of the norm gives only

$$\|\mathbf{A}^k\|_2 \le \underbrace{\|\mathbf{V}\|_2\|\mathbf{V}^{-1}\|_2}_{\kappa_2(\mathbf{V})}\,\rho(\mathbf{A})^k. \tag{33.39}$$

The factor $\kappa_2(\mathbf{V})$, the condition number (Section 33.5.3.2) of the eigenvector matrix, is 1 for an orthogonal $\mathbf{V}$ and large when eigenvectors are nearly parallel. The growth it allows comes from cancellation. When eigenvectors are nearly parallel, an ordinary input can only be written as a combination of them with large coefficients that nearly cancel. The modes then decay at different rates, the cancellation is undone, and the iterate grows before the common decay takes over.

33.4.2.3 Transient Growth

The smallest example of such growth is the upper-triangular matrix

$$\mathbf{M} = \begin{pmatrix} 0.9 & 4 \\ 0 & 0.8 \end{pmatrix}. \tag{33.40}$$

Its eigenvalues are its diagonal entries 0.9 and 0.8, both inside the unit circle, and they are distinct, so $\mathbf{M}$ is diagonalizable. We find its eigenvectors and expand $\mathbf{e}_2 = (0, 1)$ in them. For $\lambda = 0.9$, the second row of $\mathbf{M} - 0.9\mathbf{I}$ reads $-0.1v_2 = 0$, so $\mathbf{v}_1 = (1, 0)$. For $\lambda = 0.8$, the first row of $\mathbf{M} - 0.8\mathbf{I}$ reads $0.1v_1 + 4v_2 = 0$, so $\mathbf{v}_2 = (-40, 1)$. The two lines are $\arctan(1/40) \approx 1.4^\circ$ apart. The unit input is $\mathbf{e}_2 = 40\mathbf{v}_1 + \mathbf{v}_2$, and by Equation 33.27

$$\mathbf{M}^k\mathbf{e}_2 = 40 \cdot 0.9^k\,\mathbf{v}_1 + 0.8^k\,\mathbf{v}_2 = \begin{pmatrix} 40\,(0.9^k - 0.8^k) \\ 0.8^k \end{pmatrix}. \tag{33.41}$$

A vector of length 1 is the difference of two components of length about 40. Both decay, but at different rates, so their difference first grows. The first entry $40(0.9^k - 0.8^k)$ peaks where its derivative in k vanishes, where $(9/8)^k = \ln 0.8 / \ln 0.9$, at $k \approx 6.4$, with value about 10.8. Figure 33.20 draws the two components at $k = 0$ and $k = 6$. The bound Equation 33.39 with unit eigenvectors has $\kappa_2(\mathbf{V}) = 80$ and gives $\|\mathbf{M}^6\|_2 \le 80 \cdot 0.9^6 \approx 42.5$. The bound holds, but it is four times the actual value 10.8.

The cell checks the derivation and compares $\mathbf{M}$ with the diagonal matrix $\Lambda = \mathrm{diag}(0.9, 0.8)$, which has the same eigenvalues and is normal.

```
M = np.array([[0.9, 4.0], [0.0, 0.8]])
Lam = np.diag([0.9, 0.8])                  # same eigenvalues, normal
lam, V = np.linalg.eig(M)                  # unit eigenvectors as columns
print('eigenvalues:', lam)
print('coefficients of e_2 on the unit eigenvectors:',
      np.linalg.solve(V, [0., 1.]))
ks = np.arange(61)
norm_M = [np.linalg.norm(np.linalg.matrix_power(M, k), 2) for k in ks]
norm_L = [np.linalg.norm(np.linalg.matrix_power(Lam, k), 2) for k in ks]
```

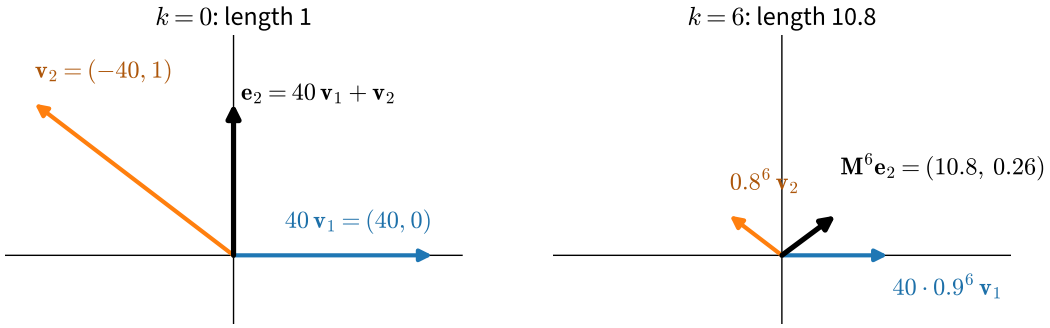


Figure 33.20: Growth of $\mathbf{M}^k\mathbf{e}_2$ before decay. The unit vector $\mathbf{e}_2$ is the difference of two eigencomponents of length about 40 that nearly cancel (left). Six steps later the components have decayed by 0.9^6 and 0.8^6, the cancellation is gone, and the result is ten times longer (right). The vertical axis is stretched about 30-fold; drawn to scale, $\mathbf{v}_1$ and $\mathbf{v}_2$ are 1.4° apart.

```
k_peak = int(np.argmax(norm_M))
print(f'max ||M^k|| = {norm_M[k_peak]:.3f} at k = {k_peak};  '
      f'max ||Lambda^k|| = {max(norm_L):.3f}')
print(f'kappa_2(V) = {np.linalg.cond(V):.1f}')
```

```
eigenvalues: [0.9 0.8]
coefficients of e_2 on the unit eigenvectors: [40.         40.01249805]
max ||M^k|| = 10.788 at k = 6;  max ||Lambda^k|| = 1.000
kappa_2(V) = 80.0
```

On the unit eigenvectors, $\mathbf{e}_2$ has two coefficients of size about 40, the lengths of $40\mathbf{v}_1$ and $\mathbf{v}_2$. The largest $\|\mathbf{M}^k\|_2$ is 10.8 at $k = 6$, while Λ never exceeds its starting norm 1. With the same eigenvalues, the normal matrix contracts from the first step, while $\mathbf{M}$ first grows ten-fold. Figure 33.21 (left) shows the two curves on a logarithmic scale; after the peak both fall with the same slope $\log 0.9$, as Gelfand's formula requires.

33.4.2.4 Pseudospectra

This subsection adds a diagnostic, not a new phenomenon, and can be skipped. The eigenvalues of a matrix that is not normal are also sensitive to perturbations. The ε-*pseudospectrum* collects every number that becomes an eigenvalue under a perturbation $\mathbf{E}$ of size at most ε, and is computed from the *resolvent* $(z\mathbf{I} - \mathbf{A})^{-1}$:

$$\Lambda_\varepsilon(\mathbf{A}) = \{z \in \mathbb{C} : z \text{ is an eigenvalue of } \mathbf{A} + \mathbf{E} \text{ for some } \|\mathbf{E}\|_2 \le \varepsilon\} = \{z \in \mathbb{C} : \|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 \ge 1/\varepsilon\}, \tag{33.42}$$

The resolvent norm counts as infinite at an eigenvalue; the equality of the two sets is Exercise 4. For a normal matrix the resolvent norm is the reciprocal of the distance from z to the nearest eigenvalue. For $\mathbf{M}$ it is far larger, as the norms at $z = 1$, a point on the unit circle, show.

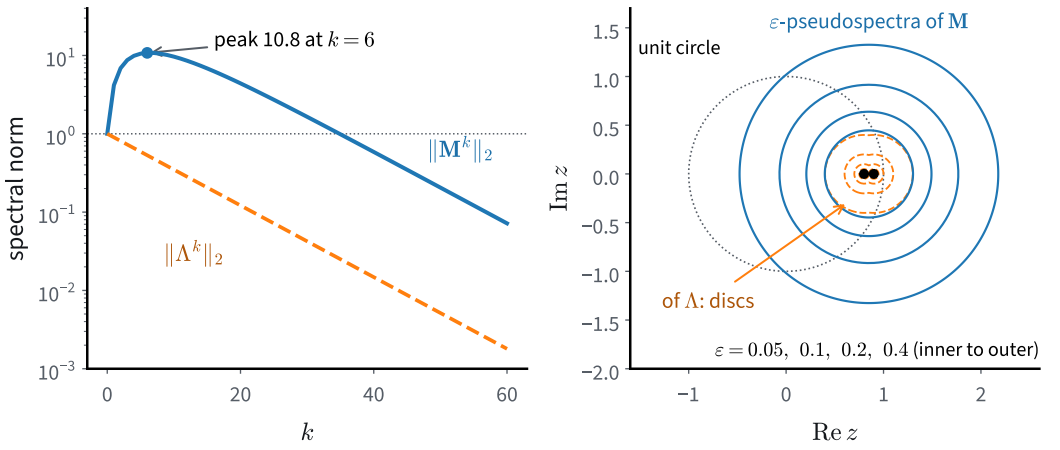


Figure 33.21: Left: $\|\mathbf{M}^k\|_2$ and $\|\Lambda^k\|_2$ for the same eigenvalues 0.9 and 0.8; $\mathbf{M}$ grows to 10.8 at $k = 6$ before both decay with slope $\log 0.9$. Right: the ε-pseudospectra for $\varepsilon = 0.05, 0.1, 0.2, 0.4$; around Λ they are discs of radius ε, around $\mathbf{M}$ they reach far past the unit circle.

```
I = np.eye(2)
print(f'||(I - M)^-1||      = {np.linalg.norm(np.linalg.inv(I - M), 2):.1f}')
print(f'||(I - Lambda)^-1|| = {np.linalg.norm(np.linalg.inv(I - Lam), 2):.1f}')
```

```
||(I - M)^-1||      = 200.3
||(I - Lambda)^-1|| = 10.0
```

For Λ the norm is 10, the reciprocal of the distance 0.1 to the eigenvalue 0.9. For $\mathbf{M}$ it is 200.3, so a perturbation of size $1/200.3 \approx 0.005$ already puts an eigenvalue of $\mathbf{M}$ on the unit circle, where Λ needs 0.1 (right panel of Figure 33.21). Rounding errors act as such perturbations, so the computed eigenvalues of a strongly non-normal matrix may be inaccurate. Section 33.6 continues with the Kreiss matrix theorem.

33.4.2.5 Products of Jacobians

A recurrent network (Section 8.6) computes its hidden state as $\mathbf{h}_t = \phi(\mathbf{W}\mathbf{h}_{t-1} + \cdots)$, with the same weights $\mathbf{W}$ at every step, and backpropagation through time multiplies the per-step Jacobians $\mathbf{D}_t = \partial \mathbf{h}_t / \partial \mathbf{h}_{t-1}$:

$$\nabla_{\mathbf{h}_0} \mathcal{L} = \mathbf{D}_1^\top \mathbf{D}_2^\top \cdots \mathbf{D}_T^\top \nabla_{\mathbf{h}_T} \mathcal{L}.$$

The gradient that reaches the first state is the last one multiplied by T matrices. These matrices differ from step to step, because the nonlinearity ϕ is evaluated at different points, so the theory of powers of one matrix does not apply. The spectral radii of the factors say nothing about the product. Take

$$\mathbf{D}_1 = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}, \qquad \mathbf{D}_2 = \begin{pmatrix} 0 & 0 \\ 2 & 0 \end{pmatrix}, \qquad \mathbf{D}_2\mathbf{D}_1 = \begin{pmatrix} 0 & 0 \\ 0 & 4 \end{pmatrix}.$$

Both factors are nilpotent with $\rho = 0$, and their product has $\rho = 4$: $\mathbf{D}_1$ moves the second coordinate into the first, doubled, and $\mathbf{D}_2$ moves it back, doubled again.

Three quantities describe iterated linear maps. The spectral radius gives the asymptotic rate of the powers of one fixed map. The spectral norm bounds every finite product, $\|\mathbf{D}_T \cdots \mathbf{D}_1\|_2 \le \prod_t \|\mathbf{D}_t\|_2$, here $4 \le 2 \cdot 2$. The *top Lyapunov exponent*

$$\mu = \lim_{T\to\infty} \frac{1}{T} \log \|\mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0\|$$

gives the average growth rate per step of a long product of changing maps. For independent, identically distributed random factors the limit exists with probability one under mild conditions (Furstenberg and Kesten 1960).

A gradient that is multiplied by maps with $\mu > 0$ explodes, and one with $\mu < 0$ vanishes (Bengio et al. 1994; Pascanu et al. 2013). The standard remedies act on these quantities rather than on eigenvalues. Orthogonal initialization (Saxe et al. 2014) makes the initial weight matrix orthogonal, so a linear recurrence preserves the length of every vector, a stronger property than $\rho = 1$. Gradient clipping caps the norm of the gradient after it has been computed. Gates, as in LSTMs and GRUs, change the Jacobians themselves so that some directions pass through with factors close to 1 (Section 8.6.2.1).

33.4.2.6 Random Matrices at Initialization

How large are these quantities for the random weight matrices of an untrained network? Let $\mathbf{A}$ be $n \times n$ with independent standard Gaussian entries, the real *Ginibre ensemble* (Ginibre 1965). As $n \to \infty$, the eigenvalues divided by $\sqrt{n}$ fill the unit disc in the complex plane uniformly: this is the *circular law*, which Tao and Vu (2010) prove for independent, identically distributed entries of mean 0 and variance 1. The circular law describes where almost all eigenvalues are; it does not exclude a few outliers. That none escapes, so that $\rho(\mathbf{A})/\sqrt{n} \to 1$ with probability one, is a separate theorem (Geman 1986). The spectral norm behaves differently: $\|\mathbf{A}\|_2/\sqrt{n} \to 2$ (Yin et al. 1988). This is the Marchenko–Pastur law of Section 33.3.2.4 at aspect ratio $c = 1$. The largest eigenvalue of $\frac{1}{n}\mathbf{A}^\top\mathbf{A}$ tends to the upper edge $(1+\sqrt{c})^2 = 4$, and $\|\mathbf{A}\|_2^2$ is n times that eigenvalue. At $n = 1000$ both limits are close.

```
n = 1000
A = np.random.randn(n, n)
print(f'rho(A) / sqrt(n)       = {np.abs(np.linalg.eigvals(A)).max() / n ** 0.5:.3f}')
print(f'sigma_max(A) / sqrt(n) = {np.linalg.norm(A, 2) / n ** 0.5:.3f}')
```

```
rho(A) / sqrt(n)       = 1.030
sigma_max(A) / sqrt(n) = 1.990
```

For large n the spectral radius is within a few percent of $\sqrt{n}$ and the largest stretch close to $2\sqrt{n}$: a random matrix stretches some vectors twice as much as its eigenvalues suggest. Initialization schemes divide random weights by $\sqrt{\text{fan-in}}$ for a different reason: with entries of variance $1/n$, $\mathbb{E}\|\mathbf{A}\mathbf{x}\|^2 = \|\mathbf{x}\|^2$ for every fixed $\mathbf{x}$, so squared lengths are preserved on average. A spectral radius near 1 is a side effect, and by Section 33.4.2.5 it does not control a product of such matrices. In fact the product of independent factors with this scaling

shrinks a typical vector, $\mu < 0$, even though every factor preserves squared length on average. The logarithm is concave, so for a unit vector the average of $\log \|\mathbf{A}\mathbf{x}\|^2$ lies below the logarithm of the average of $\|\mathbf{A}\mathbf{x}\|^2$, which is zero, and it is the logarithms that add along a product (Exercise 7).

The random matrix and **M** make the same point: eigenvalues say little about the stretch in a finite number of steps. Nor do they exist for rectangular matrices. The singular values of Section 33.5 measure stretch directly, for every matrix.

33.4.3 Summary

Without a factorization, Gershgorin discs bound the eigenvalues from the entries, and power iteration finds the dominant one at the rate $|\lambda_2/\lambda_1|$, as in PageRank. The spectral radius gives the long-run rate of the powers of one matrix and nothing more: a matrix that is not normal can grow before it decays, and the eigenvalues of the factors say nothing about a product of different matrices. There the spectral norm bounds every finite product, and the Lyapunov exponent gives the average rate. A Gaussian $n \times n$ matrix shows the gap between the two at initialization: its spectral radius is close to $\sqrt{n}$, its spectral norm close to $2\sqrt{n}$.

33.4.4 Exercises

1. **A defective limit.** For $\varepsilon \geq 0$, let

$$\mathbf{A}_\varepsilon = \begin{pmatrix} 2 & 1 \\ \varepsilon & 2 \end{pmatrix}.$$

   1. Find the eigenvalues and eigenvectors of $\mathbf{A}_\varepsilon$ for $\varepsilon > 0$ and for $\varepsilon = 0$. Which multiplicities differ at $\varepsilon = 0$?
   2. Collect the unit eigenvectors of $\mathbf{A}_\varepsilon$ in the columns of $\mathbf{V}_\varepsilon$. How do the angle between them and the condition number $\kappa_2(\mathbf{V}_\varepsilon)$ of Equation 33.39 behave as $\varepsilon \to 0$? Show that $\sqrt{\varepsilon}\,\kappa_2(\mathbf{V}_\varepsilon)$ tends to a constant as $\varepsilon \to 0$, and find it.
   3. By how much do the eigenvalues of $\mathbf{A}_0$ move when its lower-left entry changes from 0 to $\varepsilon = 10^{-10}$? Compare with the eigenvalues of the symmetric matrix $2\mathbf{I} + \varepsilon \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. This is the smallest case of Exercise 4a.

2. **Spectral radius and norms.** The spectral radius bounds every power of a matrix from below, and for normal matrices it also bounds them from above.

   1. Show that $\rho(\mathbf{A}) \leq \|\mathbf{A}^k\|_2^{1/k}$ for every square matrix $\mathbf{A}$ and every $k \geq 1$.
   2. The chain Equation 33.38 uses two facts without proof. Show that $\|\mathbf{U}\mathbf{X}\mathbf{U}^*\|_2 = \|\mathbf{X}\|_2$ for every unitary $\mathbf{U}$ and every square $\mathbf{X}$, and that the spectral norm of a diagonal matrix is the largest modulus of its diagonal entries.
   3. Show that the converse of Equation 33.38 fails: find a 3×3 matrix that is not normal but satisfies $\|\mathbf{A}^k\|_2 = \rho(\mathbf{A})^k$ for every $k \geq 1$.
   4. For a given $t > 0$, construct a diagonalizable 2×2 matrix with spectral radius $1/2$ and $\|\mathbf{A}\|_2 > t$.

3. code **Power-iteration rates.** Let **S** be the matrix of Figure 33.19.

   1. By Equation 33.37, the misalignment between the iterate and $\mathbf{v}_1$ decays at the rate $|\lambda_2/\lambda_1| = 1/3$. How many iterations reduce it by a factor of 100?

2. Let $\mathbf{A}$ be symmetric with unit eigenvector $\mathbf{v}_1$ for λ_1, and let $\mathbf{x} = \mathbf{v}_1 + \delta\mathbf{u}$ for a unit vector $\mathbf{u}$ orthogonal to $\mathbf{v}_1$. Show that the Rayleigh quotient of Section 33.3.2.3 satisfies $R(\mathbf{x}) - \lambda_1 = O(\delta^2)$, with no term linear in δ. At which rate does the Rayleigh quotient of the power iterates therefore approach λ_1? Where does the argument use that $\mathbf{A}$ is symmetric?
3. Repeat the experiment of Figure 33.19 for $\left(\begin{smallmatrix} 1 & 0.1 \\ 0.1 & 1 \end{smallmatrix}\right)$, which has the same eigenvectors as $\mathbf{S}$ and the eigenvalues 1.1 and 0.9, from a random start. Plot the sine of the angle to $\mathbf{v}_1$ and the error of the Rayleigh quotient on a logarithmic scale for 30 steps. Predict both slopes and the number of steps each error needs to shrink by a factor of 100, and explain why this matrix converges so much more slowly than $\mathbf{S}$.

4. code **Pseudospectra and transient growth.** ⊠ Let $\mathbf{M}$ be the non-normal matrix Equation 33.40 and $\Lambda = \operatorname{diag}(0.9, 0.8)$ the normal matrix with the same eigenvalues, as in Section 33.4.2.3.

   1. Prove the equality of the two sets in Equation 33.42. First show that $\|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 = 1/\sigma_{\min}(z\mathbf{I} - \mathbf{A})$, where $\sigma_{\min}(\mathbf{X}) = \min_{\|\mathbf{x}\|=1} \|\mathbf{X}\mathbf{x}\|$. For one inclusion, take unit vectors $\mathbf{u}$, $\mathbf{v}$ with $(z\mathbf{I} - \mathbf{A})\mathbf{v} = \sigma\mathbf{u}$, $\sigma = \sigma_{\min}(z\mathbf{I} - \mathbf{A})$, and show that z is an eigenvalue of $\mathbf{A} + \sigma\mathbf{u}\mathbf{v}^*$. For the other, apply $(z\mathbf{I} - \mathbf{A})\mathbf{x} = \mathbf{E}\mathbf{x}$ to an eigenvector $\mathbf{x}$ of $\mathbf{A} + \mathbf{E}$. Using the resolvent norms printed in Section 33.4.2.4, how large a perturbation makes $z = 1$ an eigenvalue of a perturbed $\mathbf{M}$, and how large for Λ?
   2. Suppose that $\|\mathbf{A}^k\|_2 \le K_0$ for all $k \ge 0$. Show that for $|z| > 1$ the series $\sum_{k\ge 0} \mathbf{A}^k / z^{k+1}$ converges to $(z\mathbf{I} - \mathbf{A})^{-1}$, and conclude that

$$\sup_{k\ge 0} \|\mathbf{A}^k\|_2 \;\ge\; K(\mathbf{A}) = \sup_{|z|>1} \left(|z| - 1\right) \|(z\mathbf{I} - \mathbf{A})^{-1}\|_2 .$$

      The quantity $K(\mathbf{A})$ is called the Kreiss constant. Restate the bound in terms of pseudospectra: if $\Lambda_\varepsilon(\mathbf{A})$ contains a point at distance δ outside the unit circle, how large must some power of $\mathbf{A}$ be?
   3. Evaluate $K(\mathbf{M})$ numerically on a grid of points z with $|z| > 1$, and compare it with the largest $\|\mathbf{M}^k\|_2$ printed in Section 33.4.2.3. Find $K(\Lambda)$ by hand.

5. code **Shifted inverse iteration.** ⊠ Power iteration finds only the dominant eigenvalue. *Inverse iteration* with shift μ runs power iteration with $(\mathbf{A} - \mu\mathbf{I})^{-1}$ instead, solving one linear system per step (Golub and Van Loan 1996). Parts (a) to (c) use the symmetric matrix

$$\mathbf{A}_4 = \begin{pmatrix} 1 & 0.1 & 0.1 & 0.1 \\ 0.1 & 3 & 0.2 & 0.3 \\ 0.1 & 0.2 & 5 & 0.5 \\ 0.1 & 0.3 & 0.5 & 9 \end{pmatrix},$$

   whose Gershgorin centres are 1, 3, 5, 9 and whose eigenvalues are about 0.99, 2.97, 4.95, and 9.08.

   1. Show that $(\mathbf{A} - \mu\mathbf{I})^{-1}$ has the same eigenvectors as $\mathbf{A}$, and find its eigenvalues. Which eigenpair of $\mathbf{A}$ does inverse iteration find, and at which rate does it converge?

2. Run inverse iteration on $\mathbf{A}_4$ with each of the four disc centres as the shift. How many iterations does each run need until the sine of the angle between the iterate and the eigenvector returned by `np.linalg.eigh` falls below 10^{-10}? Compare with plain power iteration for the largest eigenvalue.
3. *Rayleigh quotient iteration* replaces the fixed shift by the Rayleigh quotient of the current iterate at every step. Repeat (b) with the disc centres as initial shifts. Relate the iteration counts to Exercise 3b.
4. Run inverse iteration on the 3×3 matrix Equation 33.36 with the real shift $\mu = 4$. Why does it never converge? Replace the shift by $4 + i$ (complex arithmetic) and explain the observed rate of about 0.02 per step from the distances $|\lambda - \mu|$ of the three eigenvalues.

6. code **Damping in PageRank.** ⊠ Let $\mathbf{P}$ be a column-stochastic $n \times n$ matrix and $\mathbf{G} = \alpha \mathbf{P} + (1 - \alpha)\frac{1}{n}\mathbf{1}\mathbf{1}^{\top}$ with $0 < \alpha < 1$, the damped matrix of Section 33.4.1.3.

   1. Let $\mathbf{P}\mathbf{x} = \lambda \mathbf{x}$. Show that $\mathbf{1}^{\top}\mathbf{x} = 0$ whenever $\lambda \neq 1$, and that $\mathbf{G}\mathbf{x} = \alpha\lambda\mathbf{x}$ whenever $\mathbf{1}^{\top}\mathbf{x} = 0$.
   2. The eigenvalue 1 of $\mathbf{P}$ can be repeated, as it is when the links split into disconnected pockets. You may use that every column-stochastic $\mathbf{P}$ has a non-negative eigenvector for the eigenvalue 1 (a stationary distribution). Show that the vectors $\mathbf{x}$ of this eigenspace with $\mathbf{1}^{\top}\mathbf{x} = 0$ form a subspace of dimension one less than the eigenspace. Conclude that for diagonalizable $\mathbf{P}$ the eigenvalues of $\mathbf{G}$ are 1 and $\alpha\lambda_2(\mathbf{P}), \dots, \alpha\lambda_n(\mathbf{P})$, which gives the bound $|\lambda_2(\mathbf{G})| \leq \alpha$ of Haveliwala and Kamvar (2003) .
   3. Four pages link in a cycle, $1 \to 2 \to 3 \to 4 \to 1$. Find the eigenvalues of their link matrix $\mathbf{P}$. Why does undamped power iteration started at page 1 never converge?
   4. With $\alpha = 0.85$ and the cycle of (c), run power iteration on $\mathbf{G}$ from page 1, and count the iterations until the ℓ_1 distance to the stationary distribution falls below 10^{-8}. Compare with the prediction $0.85^k \leq 10^{-8}$ from (b).
   5. Check numerically on the three-page example of Section 33.4.1.3 that the eigenvalues of $\mathbf{G}$ are α times those of $\mathbf{P}$, apart from the eigenvalue 1, and that the stationary distribution moves from $(0.3, 0.4, 0.3)$ to about $(0.303, 0.394, 0.303)$. Why does damping move it toward uniform?

7. code **Lyapunov exponents of random products.** ⊠⊠ Let $\mathbf{D}_1, \mathbf{D}_2, \dots$ be independent $n \times n$ matrices with independent $\mathcal{N}(0, 1/n)$ entries, a linear model of the Jacobians of a deep network at fan-in initialization, and let $\mathbf{x}_0$ be a unit vector. By Section 33.4.2.6, each factor has spectral radius close to 1 for large n. Section 33.4.2.5 summarizes the growth of the product by its top Lyapunov exponent

$$\mu = \lim_{T \to \infty} \frac{1}{T} \log \|\mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0\|,$$

   which exists with probability one by the theorem of Furstenberg and Kesten.

   1. Show that $\mathbb{E}\|\mathbf{D}\mathbf{x}\|^2 = \|\mathbf{x}\|^2$ for every fixed $\mathbf{x}$, so that this scaling preserves squared norms on average.
   2. Let $\mathbf{u}_0 = \mathbf{x}_0$ and $\mathbf{u}_t = \mathbf{x}_t / \|\mathbf{x}_t\|$ for the iterates $\mathbf{x}_t = \mathbf{D}_t \cdots \mathbf{D}_1 \mathbf{x}_0$. Given $\mathbf{D}_1, \dots, \mathbf{D}_{t-1}$, what is the distribution of $n\|\mathbf{D}_t \mathbf{u}_{t-1}\|^2$? Use the answer to write $\log \|\mathbf{D}_T \cdots \mathbf{D}_1 \mathbf{x}_0\|$

as a sum of independent terms. With $\mathbb{E}\log\chi^2_n = \psi(n/2) + \log 2$, where ψ is the digamma function, derive the closed form of Cohen and Newman (1984)

$$\mu = \tfrac{1}{2}(\psi(n/2) + \log 2 - \log n).$$

3. Show that $\mu < 0$ for every n, although (a) holds. Evaluate μ in closed form for $n = 1$, using $\psi(1/2) = -\gamma_{\mathrm{E}} - 2\log 2$ with Euler's constant $\gamma_{\mathrm{E}} \approx 0.5772$. Using $\psi(x) = \log x - \frac{1}{2x} + O(x^{-2})$, show that $\mu \approx -1/(2n)$ for large n.

4. Simulate $T = 10^4$ steps with renormalization for $n \in \{1, 2, 5, 50\}$, and compare the measured exponent with (b) and with the logarithms of the spectral radius and of the largest singular value of single factors. By which factor does a typical input shrink in a linear network of width 50 and depth 100 at this initialization? What does the comparison say about the three quantities of Section 33.4.2.5?

33.5 Singular Value Decomposition and Low-Rank Approximation

The eigendecomposition of Section 33.3 describes a square matrix by the directions it only stretches, and it turns the powers of that matrix into powers of numbers. Most matrices in deep learning are of another kind. A weight matrix maps inputs of one width to outputs of another, and a data matrix has one row per example and one column per feature. Neither is applied to its own output, and neither need be square. Even for a square matrix the eigenvalues can fail us. The shear $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ is *defective*, with a single eigenvector direction where the plane needs two, and Section 33.4 met a matrix that lengthens a vector ten-fold although both of its eigenvalues are below 1. We need a description that works for every matrix, that measures stretch directly, and that tells how few numbers suffice to describe a large matrix well.

A photograph shows why the last requirement matters. A grayscale picture of a cat, 400×500 pixels, is a matrix with 200,000 entries. Like any matrix, it can be written as a sum of rank-one pieces, each the outer product of a column and a row. Twenty well-chosen pieces, 18,020 numbers or 9% of the original, reproduce it with a relative error of 8% (Figure 33.25 shows the result). Which twenty pieces are these, and could any other twenty do better? The same question decides how many parameters a fine-tuning adapter for a large model needs, and how many directions of a dataset are worth keeping.

The answer is to stop asking one basis to serve as both the input and the output axes. Choose one orthonormal basis for the inputs and another for the outputs, and every matrix becomes a list of independent stretches $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$. Rank, the best low-rank approximation, principal component analysis, least squares and the sensitivity of a linear solve to errors are all read off that list. Section 1.3 defined two sizes of a matrix, the Frobenius norm $\|\mathbf{A}\|_F$ (the root of the sum of the squared entries) and the spectral norm $\|\mathbf{A}\|_2$ (the largest factor by which $\mathbf{A}$ stretches a vector). In this section both become functions of the σ_i.

Two results carry the section: the existence of the decomposition, and the Eckart–Young theorem, by which cutting the list short gives the best approximation of each rank. The pseudoinverse and the condition number follow from the first in a few lines. The last part, on the SVD in deep learning, can be skipped on a first reading. The computations use

NumPy. Throughout, **T** is the shear, and **A** stands for a general matrix unless it is named otherwise.

33.5.1 The Singular Value Decomposition

33.5.1.1 The Factorization and Its Geometry

The symmetric matrix $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ of Section 1.3 maps the unit circle to an ellipse whose axes lie along its eigenvectors $(1,1)$ and $(-1,1)$, with half-lengths 3 and 1. A non-symmetric matrix also maps the circle to an ellipse (flattened to a segment if the matrix is singular), as we prove below. The input directions that land on the axes of this ellipse, however, are no longer the axes themselves. There are two orthonormal frames, one for the inputs and one for the outputs, and the singular value decomposition names both.

Theorem (singular value decomposition). *Every real $m \times n$ matrix* **A** *can be written as*

$$\mathbf{A} = \mathbf{U}\,\Sigma\,\mathbf{V}^{\top}, \tag{33.43}$$

where $\mathbf{U} \in \mathbb{R}^{m\times m}$ *and* $\mathbf{V} \in \mathbb{R}^{n\times n}$ *are orthogonal (*$\mathbf{U}^{\top}\mathbf{U} = \mathbf{I}$, $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$*) and* $\Sigma \in \mathbb{R}^{m\times n}$ *is zero except for the entries* $\sigma_1 \ge \sigma_2 \ge \cdots \ge \sigma_{\min(m,n)} \ge 0$ *on its main diagonal.* The columns $\mathbf{v}_i$ of **V** are the *right singular vectors*, the columns $\mathbf{u}_i$ of **U** are the *left singular vectors*, and the σ_i are the *singular values*.

So a matrix of any shape is an orthogonal map, a diagonal stretch, and a second orthogonal map. Reading off column i of $\mathbf{AV} = \mathbf{U}\Sigma$ gives the same statement one direction at a time:

$$\mathbf{A}\mathbf{v}_i = \sigma_i\,\mathbf{u}_i, \qquad i \le \min(m,n). \tag{33.44}$$

The input direction $\mathbf{v}_i$ goes to the output direction $\mathbf{u}_i$, stretched by σ_i; when $n > m$, the remaining input directions $\mathbf{v}_{m+1}, \dots, \mathbf{v}_n$ are sent to zero. Read right to left, Equation 33.43 acts on a vector $\mathbf{x}$ in three stages:

1. $\mathbf{V}^{\top}$ rotates the input so that each right singular vector $\mathbf{v}_i$ lands on the coordinate axis $\mathbf{e}_i$;
2. Σ stretches axis i by σ_i and, when $m \neq n$, appends zero coordinates or drops coordinates to change the dimension;
3. **U** rotates the stretched axes so that axis i lies along $\mathbf{u}_i$.

"Rotation" here means an orthogonal map, which may include a reflection.

Figure 33.22 traces the three stages for the shear **T**. Its singular values are $\sigma_1 = 1.618$ and $\sigma_2 = 0.618$; the first right singular vector $\mathbf{v}_1$ points at 58.3° above the x-axis, and its image direction $\mathbf{u}_1$ at 31.7°. The two frames differ, so $\mathbf{U} \neq \mathbf{V}$. Section 33.5.1.2 derives these numbers.

In n dimensions the same three stages turn the unit sphere into an ellipsoid whose semi-axes are the vectors $\sigma_i\mathbf{u}_i$. When only $r < n$ of the input directions $\mathbf{v}_i$ are stretched by a positive σ_i, the other $n - r$ directions collapse, and the image is a filled r-dimensional ellipsoid lying in the span of $\mathbf{u}_1, \dots, \mathbf{u}_r$.

The compact SVD and the dyadic sum. Let r be the number of positive singular values; Section 33.5.1.5 shows that r is the rank of **A**. Every row and column of Σ beyond the first r is zero, and so is every contribution of the singular vectors that meet them. Dropping them

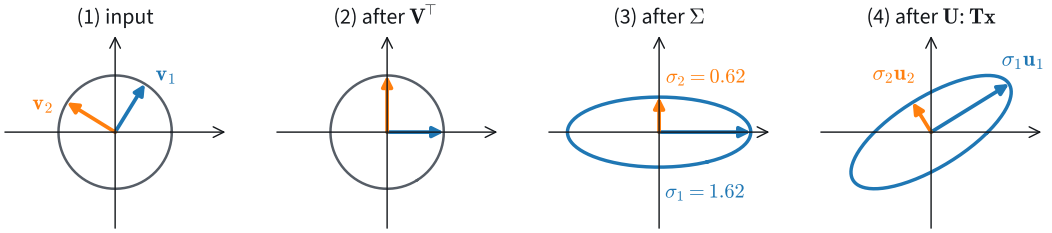


Figure 33.22: The shear $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right) = \mathbf{U}\Sigma\mathbf{V}^\top$ acting on the unit circle, read left to right: $\mathbf{V}^\top$ rotates by $-58.3°$, Σ stretches by $\varphi = 1.618$ along x and by $1/\varphi = 0.618$ along y, and $\mathbf{U}$ rotates by $31.7°$. The input frame $\{\mathbf{v}_i\}$ and the output frame $\{\mathbf{u}_i\}$ differ.

gives the *compact SVD* $\mathbf{A} = \mathbf{U}_r\Sigma_r\mathbf{V}_r^\top$, with $\mathbf{U}_r \in \mathbb{R}^{m\times r}$ and $\mathbf{V}_r \in \mathbb{R}^{n\times r}$ holding the first r singular vectors and $\Sigma_r = \operatorname{diag}(\sigma_1, \dots, \sigma_r)$. In between is the *thin SVD*, the form libraries return on request, with $\min(m, n)$ columns in $\mathbf{U}$ and $\mathbf{V}$; it still lists zero singular values when $r < \min(m, n)$. For a 3×2 matrix of rank 2 the full factors have the shapes 3×3, 3×2 and 2×2, the thin ones 3×2, 2×2 and 2×2, and the compact SVD is the thin one. Multiplying out the compact SVD column by row writes $\mathbf{A}$ as a sum of rank-one matrices, or *dyads*:

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i\, \mathbf{u}_i\mathbf{v}_i^\top. \tag{33.45}$$

Each term is the outer product of an output direction and an input direction, weighted by its singular value, and the terms come sorted by weight. For the shear, $\mathbf{T} = \varphi\,\mathbf{u}_1\mathbf{v}_1^\top + \varphi^{-1}\,\mathbf{u}_2\mathbf{v}_2^\top$, one piece of weight 1.618 and one of weight 0.618. Keeping only the heaviest pieces is the low-rank approximation of Section 33.5.2.1.

33.5.1.2 Existence

Where do the σ_i, $\mathbf{v}_i$ and $\mathbf{u}_i$ come from, and why is the output frame orthonormal? The spectral theorem of Section 33.3.2.1 gives every symmetric matrix an orthonormal basis of eigenvectors with real eigenvalues. We apply it to the *Gram matrix* $\mathbf{A}^\top\mathbf{A}$, the $n \times n$ matrix of dot products between the columns of $\mathbf{A}$. This matrix is symmetric, and it is positive semidefinite (Section 33.3.2.2) because $\mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x} = \|\mathbf{A}\mathbf{x}\|^2 \geq 0$.

Proof of Equation 33.43 (existence). By the spectral theorem, $\mathbf{A}^\top\mathbf{A}$ has an orthonormal basis of eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$ with eigenvalues $\lambda_1 \geq \cdots \geq \lambda_n$, all non-negative because $\lambda_i = \mathbf{v}_i^\top\mathbf{A}^\top\mathbf{A}\mathbf{v}_i = \|\mathbf{A}\mathbf{v}_i\|^2$. Define

$$\sigma_i = \sqrt{\lambda_i}, \qquad r = \#\{i : \sigma_i > 0\}.$$

The singular values are the square roots of the eigenvalues of the Gram matrix, and r counts the positive ones. For each $i \leq r$ set $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/\sigma_i$. The construction works because these vectors come out orthonormal, using only that $\mathbf{v}_j$ is an eigenvector and the $\mathbf{v}_i$ are orthonormal:

$$\mathbf{u}_i^\top\mathbf{u}_j = \frac{1}{\sigma_i\sigma_j}\,\mathbf{v}_i^\top\mathbf{A}^\top\mathbf{A}\,\mathbf{v}_j = \frac{\lambda_j}{\sigma_i\sigma_j}\,\mathbf{v}_i^\top\mathbf{v}_j = \frac{\lambda_j}{\sigma_i\sigma_j}\,\delta_{ij} = \delta_{ij}.$$

The orthonormality of the input frame passes through $\mathbf{A}^\top\mathbf{A}$ to the output frame. It remains to check the dimensions. The r orthonormal vectors $\mathbf{u}_1, \dots, \mathbf{u}_r$ lie in $\mathbb{R}^m$, so $r \le m$, and $r \le n$ because there are n eigenvalues; hence $r \le \min(m, n)$, and $\lambda_i = 0$ for every $i > r$. Extend $\mathbf{u}_1, \dots, \mathbf{u}_r$ to an orthonormal basis $\mathbf{u}_1, \dots, \mathbf{u}_m$ of $\mathbb{R}^m$ (Gram–Schmidt applied to any completion). For $r < i \le n$, $\|\mathbf{A}\mathbf{v}_i\|^2 = \lambda_i = 0$, so $\mathbf{A}\mathbf{v}_i = \mathbf{0}$. Compare this with $\mathbf{U}\Sigma$ column by column: column i of $\mathbf{U}\Sigma$ is $\sigma_i\mathbf{u}_i$ for $i \le \min(m, n)$, which is $\mathbf{0}$ when $i > r$, and it is $\mathbf{0}$ for $i > m$, where Σ has no diagonal entry. In every case column i of $\mathbf{A}\mathbf{V}$ equals column i of $\mathbf{U}\Sigma$, so $\mathbf{A}\mathbf{V} = \mathbf{U}\Sigma$, and right-multiplying by $\mathbf{V}^\top = \mathbf{V}^{-1}$ gives $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top$. ■

The proof is also an algorithm: diagonalize the symmetric matrix $\mathbf{A}^\top\mathbf{A}$, take square roots for the singular values, and push the eigenvectors through $\mathbf{A}$ for the left singular vectors.

The shear is small enough to take apart by hand. For $\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right)$ the Gram matrix is $\mathbf{T}^\top\mathbf{T} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 2 \end{smallmatrix}\right)$, with trace 3 and determinant 1, so its characteristic polynomial is $\lambda^2 - 3\lambda + 1$ and its eigenvalues are $(3 \pm \sqrt{5})/2$. With the golden ratio $\varphi = (1 + \sqrt{5})/2$, which satisfies $\varphi^2 = \varphi + 1$, these are φ^2 and φ^{-2}. The singular values are therefore $\sigma_1 = \varphi \approx 1.618$ and $\sigma_2 = 1/\varphi \approx 0.618$, and their product is $1 = |\det \mathbf{T}|$. For the first right singular vector (x, y), the first row of $(\mathbf{T}^\top\mathbf{T} - \varphi^2\mathbf{I})(x, y) = \mathbf{0}$ reads $(1 - \varphi^2)x + y = 0$, and $\varphi^2 - 1 = \varphi$, so $y = \varphi x$ and $\mathbf{v}_1 \propto (1, \varphi)$. Pushing it through $\mathbf{T}$ gives $\mathbf{T}(1, \varphi) = (1 + \varphi, \varphi) = \varphi\,(\varphi, 1)$, so $\mathbf{u}_1 \propto (\varphi, 1)$. Both vectors have length $\sqrt{1 + \varphi^2} = \sqrt{\varphi + 2}$, which gives $\mathbf{v}_1 \approx (0.526, 0.851)$ at the angle $\arctan\varphi = 58.3^\circ$ and $\mathbf{u}_1 \approx (0.851, 0.526)$ at 31.7°. The second pair follows by orthogonality, $\mathbf{v}_2 = (-\varphi, 1)/\sqrt{\varphi + 2}$ and $\mathbf{u}_2 = (-1, \varphi)/\sqrt{\varphi + 2}$, and indeed $\mathbf{T}\mathbf{v}_2 = (1 - \varphi, 1)/\sqrt{\varphi + 2} = \varphi^{-1}\mathbf{u}_2$ because $(1 - \varphi)\varphi = -1$. Both $\mathbf{V}$ and $\mathbf{U}$ are rotations $\mathbf{R}_\theta = \left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix}\right)$, and the shear factors as

$$\mathbf{T} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \mathbf{R}_{31.7^\circ} \begin{bmatrix} \varphi & 0 \\ 0 & 1/\varphi \end{bmatrix} \mathbf{R}_{58.3^\circ}^\top. \tag{33.46}$$

The shear rotates the plane by -58.3°, stretches it by 1.618 along the x-axis and by 0.618 along the y-axis, and rotates it back by only 31.7°. The two angles add up to 90° because $\mathbf{u}_1$ is $\mathbf{v}_1$ with its coordinates swapped, a consequence of the symmetry $\mathbf{T}^\top = \mathbf{J}\mathbf{T}\mathbf{J}$ for the swap $\mathbf{J} = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$.

A library call should return the same factors, though possibly with other signs (Section 33.5.1.3). The cell, which stores the shear as `A`, therefore first fixes the signs that put $\mathbf{u}_1$ in the first quadrant and make $\mathbf{U}$ a rotation.

```
A = np.array([[1., 1.], [0., 1.]])
U, s, Vt = np.linalg.svd(A)
if U[0, 0] < 0:                          # joint sign choice: u_1 in the first quadrant
    U[:, 0], Vt[0] = -U[:, 0], -Vt[0]
if np.linalg.det(U) < 0:                 # joint sign choice: U a rotation (then so is V)
    U[:, 1], Vt[1] = -U[:, 1], -Vt[1]
phi = (1 + np.sqrt(5)) / 2
print('singular values', s.round(4), '  phi, 1/phi', np.array([phi, 1 / phi]).round(4))
print('U =', U.round(3).tolist(), '  Vt =', Vt.round(3).tolist())
print('angles of u_1, v_1 (degrees):', np.degrees(np.arctan2(U[1, 0], U[0, 0])).round(1),
      np.degrees(np.arctan2(Vt[0, 1], Vt[0, 0])).round(1))
print('A v_1 = sigma_1 u_1:', np.allclose(A @ Vt[0], s[0] * U[:, 0]),
      '  sigma_1 sigma_2 =', s.prod().round(6))
```

```
singular values [1.618 0.618]   phi, 1/phi [1.618 0.618]
U = [[0.851, -0.526], [0.526, 0.851]]   Vt = [[0.526, 0.851], [-0.851, 0.526]]
angles of u_1, v_1 (degrees): 31.7 58.3
A v_1 = sigma_1 u_1: True   sigma_1 sigma_2 = 1.0
```

It does: the library returns φ and $1/\varphi$ and the two rotations of Equation 33.46.

The construction of the proof works unchanged for a rectangular matrix. For the 3×2 matrix with rows $(3,1)$, $(1,3)$, $(0,2)$, the Gram matrix is $\left(\begin{smallmatrix} 10 & 6 \\ 6 & 14 \end{smallmatrix}\right)$, with eigenvalues $12 \pm \sqrt{40}$. The next cell carries out the proof's three steps numerically (eigenvectors of $\mathbf{A}^\top\mathbf{A}$, square roots, $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/\sigma_i$) and sets the result beside the library's.

```
A = np.array([[3., 1.], [1., 3.], [0., 2.]])
lam, V = np.linalg.eigh(A.T @ A)
lam, V = lam[::-1], V[:, ::-1]                    # descending order
sigma = np.sqrt(lam)
U = A @ V / sigma                                  # u_i = A v_i / sigma_i
U_lib, s_lib, Vt_lib = np.linalg.svd(A, full_matrices=False)
print('sigma from eigh:', sigma.round(5), '  from svd:', s_lib.round(5))
print('U orthonormal:', np.allclose(U.T @ U, np.eye(2)), '  same U, V up to sign:',
      np.allclose(abs(U), abs(U_lib)), np.allclose(abs(V), abs(Vt_lib.T)))
print('rebuild error:', np.linalg.norm((U * sigma) @ V.T - A).round(12))
```

```
sigma from eigh: [4.28072 2.38232]   from svd: [4.28072 2.38232]
U orthonormal: True   same U, V up to sign: True True
rebuild error: 0.0
```

Both routes give $\sigma_1 = \sqrt{12 + \sqrt{40}} \approx 4.281$ and $\sigma_2 = \sqrt{12 - \sqrt{40}} \approx 2.382$. The pushed-through vectors are orthonormal, as the proof promised, and they agree with the library's up to sign.

33.5.1.3 Uniqueness and the Largest Stretch

The singular *values* are unique: they are the square roots of the eigenvalues of $\mathbf{A}^\top\mathbf{A}$, which $\mathbf{A}$ determines. The singular *vectors* have three kinds of freedom.

- For a simple $\sigma_i > 0$, the only choice is the joint sign flip $(\mathbf{u}_i, \mathbf{v}_i) \to (-\mathbf{u}_i, -\mathbf{v}_i)$. The flip must be joint, because flipping only one side would break $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$.
- For a repeated $\sigma > 0$ with index set $\mathcal{J}$, the columns $\mathbf{V}_\mathcal{J}$ may be replaced by $\mathbf{V}_\mathcal{J}\mathbf{Q}$ for any orthogonal $\mathbf{Q}$ of matching size, and then $\mathbf{U}_\mathcal{J}$ must become $\mathbf{U}_\mathcal{J}\mathbf{Q}$ with the *same* $\mathbf{Q}$, since $\mathbf{U}_\mathcal{J} = \mathbf{A}\mathbf{V}_\mathcal{J}/\sigma$.
- For $\sigma = 0$, the vectors $\mathbf{v}_{r+1}, \dots, \mathbf{v}_n$ and $\mathbf{u}_{r+1}, \dots, \mathbf{u}_m$ can be rotated independently, because they do not enter the dyadic sum Equation 33.45.

This freedom is why two libraries, or one library on two machines, may return different signs, and why the cells of this section fix a sign before printing vectors.

The top singular value is the largest factor by which $\mathbf{A}$ stretches any unit vector:

$$\sigma_1 = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\| = \|\mathbf{A}\|_2. \tag{33.47}$$

The reason is the Rayleigh proposition of Section 33.3.2.3: over unit vectors $\mathbf{x}$, the quadratic

form $\mathbf{x}^\top \mathbf{S}\mathbf{x}$ of a symmetric matrix $\mathbf{S}$ is maximized by the top eigenvector, with the top eigenvalue as its value. With $\mathbf{S} = \mathbf{A}^\top\mathbf{A}$, $\|\mathbf{A}\mathbf{x}\|^2 = \mathbf{x}^\top\mathbf{A}^\top\mathbf{A}\mathbf{x}$ is maximized at $\mathbf{x} = \mathbf{v}_1$ with value $\lambda_1 = \sigma_1^2$. So σ_1 is the spectral norm of Section 1.3: no unit vector is stretched by the shear by more than $\varphi = 1.618$, and $\mathbf{v}_1$ is stretched by that much. The other singular values solve the same problem with the earlier right singular vectors excluded, a procedure called *deflation*: $\sigma_k = \max\{\|\mathbf{A}\mathbf{x}\| : \|\mathbf{x}\| = 1,\ \mathbf{x} \perp \mathbf{v}_1, \dots, \mathbf{v}_{k-1}\}$. Exercise 2a runs the argument the other way and builds the SVD from the maximum stretch, without the spectral theorem.

33.5.1.4 Singular Values and Eigenvalues

Substituting Equation 33.43 into the two Gram matrices, and using $\mathbf{U}^\top\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$, gives

$$\mathbf{A}^\top\mathbf{A} = \mathbf{V}\,\Sigma^\top\Sigma\,\mathbf{V}^\top, \qquad \mathbf{A}\mathbf{A}^\top = \mathbf{U}\,\Sigma\Sigma^\top\,\mathbf{U}^\top. \tag{33.48}$$

Both are eigendecompositions: the right singular vectors are eigenvectors of $\mathbf{A}^\top\mathbf{A}$, the left singular vectors are eigenvectors of $\mathbf{A}\mathbf{A}^\top$, and the squares σ_i^2 are the nonzero eigenvalues the two share. For a symmetric positive semidefinite matrix the SVD and the eigendecomposition coincide, with $\mathbf{U} = \mathbf{V}$.

For any other matrix they differ, and singular values are not eigenvalue magnitudes. The chapter's running matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$ has the eigenvalues $2 \pm i$ of modulus $\sqrt{5} \approx 2.236$ (Section 33.3). Its Gram matrix $\mathbf{A}^\top\mathbf{A} = \left(\begin{smallmatrix} 2 & -1 \\ -1 & 13 \end{smallmatrix}\right)$ has trace 15 and determinant 25, so $\sigma_{1,2}^2 = (15 \pm 5\sqrt{5})/2$ and $\sigma_{1,2} = (5 \pm \sqrt{5})/2 \approx 3.618,\ 1.382$. One identity does hold for every square matrix:

$$|\det \mathbf{A}| = \prod_i \sigma_i = \prod_i |\lambda_i|.$$

The first equality holds because orthogonal factors have determinant ± 1, and the second because the determinant is the product of the eigenvalues. For a 2×2 matrix with a conjugate pair of eigenvalues, both of modulus $|\lambda|$, the identity forces $|\lambda| = \sqrt{\sigma_1\sigma_2}$; here $\sqrt{3.618 \cdot 1.382} = \sqrt{5}$. The identity is also the volume reading of the determinant from Section 33.2. Rotations preserve area, so the stretch Σ carries the whole change of area, and the running matrix turns the unit square into a parallelogram of area $3.618 \cdot 1.382 = 5$.

Computing the SVD. Libraries do not form $\mathbf{A}^\top\mathbf{A}$. Forming it squares the condition number of Section 33.5.3.2, and singular values far below σ_1 are then lost to rounding. Instead, they reduce $\mathbf{A}$ to bidiagonal form by orthogonal transformations (Golub–Kahan bidiagonalization) and diagonalize the result (Golub and Van Loan 1996).

33.5.1.5 The Four Fundamental Subspaces

Which inputs does $\mathbf{A}$ destroy, and which outputs can it reach? Section 33.2 named these sets the null space and the column space, and defined the rank as the number of independent columns. Because $\mathbf{U}$ and $\mathbf{V}$ are invertible, multiplying by them changes no dimension, so $\operatorname{rank}\mathbf{A} = \operatorname{rank}\Sigma$, the number of nonzero diagonal entries:

$$\operatorname{rank}\mathbf{A} = \#\{i : \sigma_i > 0\} = r. \tag{33.49}$$

The rank counts the stretches that do not collapse. With a tolerance for rounding (below), this is the numerically reliable way to compute rank. It costs more than elimination but is stable. A perturbation $\mathbf{E}$, such as the rounding error of the computation, moves no singular value by more than $\|\mathbf{E}\|_2$ (Weyl's inequality in Section 33.5.2.1). A clear gap between large and tiny singular values can therefore be trusted.

The dyadic sum Equation 33.45 makes the four fundamental subspaces explicit, splitting the singular vectors at the index r:

- the **column space** (range) of $\mathbf{A}$ is $\operatorname{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$, because every output $\mathbf{Ax} = \sum_{i \le r} \sigma_i (\mathbf{v}_i^\top \mathbf{x}) \mathbf{u}_i$ is a combination of these vectors, and each of them is reached, $\mathbf{u}_i = \mathbf{A}(\mathbf{v}_i/\sigma_i)$;
- the **null space** of $\mathbf{A}$ is $\operatorname{span}\{\mathbf{v}_{r+1}, \dots, \mathbf{v}_n\}$, because by the same formula $\mathbf{Ax} = \mathbf{0}$ exactly when $\mathbf{v}_i^\top \mathbf{x} = 0$ for every $i \le r$ (the $\mathbf{u}_i$ are independent and the σ_i positive), that is, when $\mathbf{x}$ is orthogonal to $\mathbf{v}_1, \dots, \mathbf{v}_r$;
- the **row space** (column space of $\mathbf{A}^\top$) is $\operatorname{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ and the **left null space** (null space of $\mathbf{A}^\top$) is $\operatorname{span}\{\mathbf{u}_{r+1}, \dots, \mathbf{u}_m\}$, by the same two arguments applied to $\mathbf{A}^\top = \mathbf{V}\Sigma^\top \mathbf{U}^\top$, whose SVD swaps the roles of the $\mathbf{u}_i$ and the $\mathbf{v}_i$.

The input space therefore splits into two orthogonal pieces, the row space of dimension r and the null space of dimension $n - r$. The output space splits into the column space of dimension r and the left null space of dimension $m - r$. On the row space, $\mathbf{A}$ sends each $\mathbf{v}_i$ to $\sigma_i \mathbf{u}_i$, a one-to-one map onto the column space; the null space goes to zero.

For the smallest case, take the singular matrix $\mathbf{B} = \left(\begin{smallmatrix} 2 & 1 \\ 4 & 2 \end{smallmatrix}\right)$ of Section 33.2. Its Gram matrix $\left(\begin{smallmatrix} 20 & 10 \\ 10 & 5 \end{smallmatrix}\right)$ has eigenvalues 25 and 0, so $\sigma_1 = 5$, $\sigma_2 = 0$, and $r = 1$. The four subspaces are the lines along

$$\mathbf{v}_1 = \tfrac{1}{\sqrt{5}}(2,1), \quad \mathbf{v}_2 = \tfrac{1}{\sqrt{5}}(-1,2), \quad \mathbf{u}_1 = \tfrac{1}{\sqrt{5}}(1,2), \quad \mathbf{u}_2 = \tfrac{1}{\sqrt{5}}(-2,1), \qquad \mathbf{B} = 5\,\mathbf{u}_1\mathbf{v}_1^\top.$$

The column space is the line $y = 2x$ along $\mathbf{u}_1$, and the left null space is the perpendicular line along $\mathbf{u}_2$. The null space is the line along $(1, -2)$, perpendicular to the row $(2, 1)$ that spans the row space. Figure 33.23 draws the general case; the pseudoinverse of Section 33.5.3.1 runs the map from the row space to the column space backwards.

Numerical rank. In floating-point arithmetic a matrix that is mathematically rank-deficient rarely has singular values that are exactly zero; rounding leaves values of order $\epsilon_{\text{mach}}\sigma_1$ instead. Rank is therefore decided by a threshold. `np.linalg.matrix_rank` counts the singular values above $\sigma_1 \max(m, n)\, \epsilon_{\text{mach}}$, a tolerance that scales with both the largest singular value and the size of the matrix. What do the "zero" singular values of a 4×4 matrix of rank two look like?

```
A = np.random.standard_normal((4, 2)) @ np.random.standard_normal((2, 4))   # rank <= 2
s = np.linalg.svd(A, compute_uv=False)
tol = s[0] * max(A.shape) * np.finfo(float).eps
print('singular values:', np.array2string(s, formatter={'float_kind': '{:.1e}'.format}))
print(f'tolerance {tol:.1e};  exact zeros: {(s == 0).sum()};  '
      f'numerical rank: {np.linalg.matrix_rank(A)}')
```

```
singular values: [4.4e+00 2.2e+00 4.7e-17 0.0e+00]
tolerance 3.9e-15;  exact zeros: 1;  numerical rank: 2
```

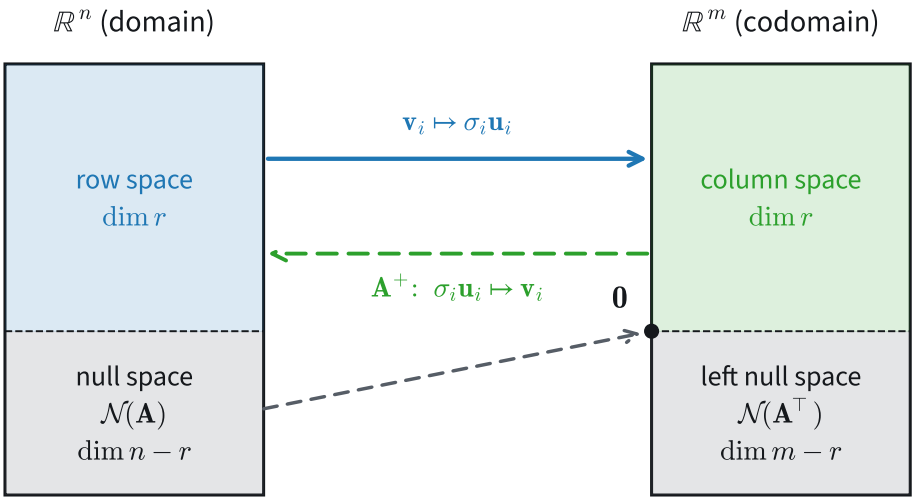


Figure 33.23: The four fundamental subspaces. $\mathbf{A}$ maps the row space $\text{span}\{\mathbf{v}_1, \dots, \mathbf{v}_r\}$ one-to-one onto the column space $\text{span}\{\mathbf{u}_1, \dots, \mathbf{u}_r\}$ by $\mathbf{v}_i \mapsto \sigma_i \mathbf{u}_i$, and maps the null space to zero. The pseudoinverse $\mathbf{A}^+$ (dashed) runs the map backwards, $\sigma_i \mathbf{u}_i \mapsto \mathbf{v}_i$, and sends the left null space to zero.

Two singular values are of order one. The other two are rounding noise, 10^{-15} or smaller and several times below the tolerance, but usually not exact zeros. A test for exact zeros would therefore report the wrong rank, while the threshold reports two.

33.5.2 Low-Rank Approximation

33.5.2.1 Best Low-Rank Approximation

Which twenty rank-one pieces of the photograph should we keep? The dyadic sum Equation 33.45 lists the pieces of $\mathbf{A}$ heaviest first, and the obvious candidate keeps the first k. This is the *truncated SVD*

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \, \mathbf{u}_i \mathbf{v}_i^\top . \tag{33.50}$$

It is a matrix of rank at most k stored as k singular values, k columns of length m and k of length n, that is $k(m + n + 1)$ numbers instead of mn. For the photograph and $k = 20$ that is 18,020 of 200,000 numbers. Two questions remain: how large is the error, and could a cleverer matrix of rank k do better?

Matrix norms as singular values. The error needs a size for matrices. The Frobenius norm of Section 1.3, $\|\mathbf{A}\|_F^2 = \sum_{ij} a_{ij}^2$, sums the squared column lengths, which are the diagonal entries of $\mathbf{A}^\top \mathbf{A}$, so $\|\mathbf{A}\|_F^2 = \text{tr}(\mathbf{A}^\top \mathbf{A})$. Inserting the SVD and moving $\mathbf{V}$ around the trace, which is cyclic (Section 33.2), gives

$$\|\mathbf{A}\|_F^2 = \operatorname{tr}(\mathbf{V}\Sigma^\top\Sigma\mathbf{V}^\top) = \operatorname{tr}(\Sigma^\top\Sigma\,\mathbf{V}^\top\mathbf{V}) = \sum_i \sigma_i^2. \tag{33.51}$$

The Frobenius norm is the root of the sum of the squared singular values, so orthogonal factors on either side never change it. For the shear, $1+1+0+1 = 3 = \varphi^2 + \varphi^{-2}$. The spectral norm is the largest singular value, $\|\mathbf{A}\|_2 = \sigma_1$, by Equation 33.47.

Theorem (Eckart–Young–Mirsky). *Let $\sigma_i = 0$ for $i > r$. For every $k \geq 0$, the smallest spectral-norm error of a matrix of rank at most k is*

$$\min_{\operatorname{rank}\mathbf{B}\leq k} \|\mathbf{A}-\mathbf{B}\|_2 = \sigma_{k+1}, \tag{33.52}$$

and the smallest squared Frobenius error is $\min_{\operatorname{rank}\mathbf{B}\leq k}\|\mathbf{A}-\mathbf{B}\|_F^2 = \sum_{i>k}\sigma_i^2$. Both minima are attained at $\mathbf{B} = \mathbf{A}_k$.

The best rank-k approximation misses by the first singular value it leaves out, or, in the Frobenius norm, by all of them together. Eckart and Young proved the Frobenius case (Eckart and Young 1936), and Mirsky extended the result to every norm that is unchanged by orthogonal factors on either side (Mirsky 1960). We prove the spectral-norm statement first. The Frobenius statement needs one more inequality, Weyl's, which the same argument gives.

Picture $n = 3$ and $k = 1$ first. A matrix $\mathbf{B}$ of rank one destroys a whole plane of inputs, and the top two right singular vectors of $\mathbf{A}$ span another plane. Two planes through the origin of $\mathbb{R}^3$ meet in at least a line. On that line $\mathbf{B}$ gives zero while $\mathbf{A}$ stretches by at least σ_2, so $\mathbf{A}-\mathbf{B}$ stretches some unit vector by at least σ_2 (Figure 33.24). The proof repeats this argument in n dimensions.

Proof (spectral norm). *The truncation achieves σ_{k+1}.* The residual $\mathbf{A}-\mathbf{A}_k = \sum_{i>k}\sigma_i\mathbf{u}_i\mathbf{v}_i^\top$ is itself written as a compact SVD, with largest singular value σ_{k+1}, so $\|\mathbf{A}-\mathbf{A}_k\|_2 = \sigma_{k+1}$ by Equation 33.47. By Equation 33.51, also $\|\mathbf{A}-\mathbf{A}_k\|_F^2 = \sum_{i>k}\sigma_i^2$.

No matrix of rank at most k does better. If $k \geq r$ there is nothing to prove, so let $k < r$ and $\operatorname{rank}\mathbf{B} \leq k$. By rank–nullity (Section 33.2), the null space $\mathcal{K} = \ker\mathbf{B}$ has dimension at least $n-k$. The span $\mathcal{V} = \operatorname{span}\{\mathbf{v}_1,\dots,\mathbf{v}_{k+1}\}$ of the top $k+1$ right singular vectors has dimension $k+1$. The main step shows that these two subspaces share a nonzero vector. Their dimensions add to more than n, and

$$\dim(\mathcal{K}\cap\mathcal{V}) = \dim\mathcal{K} + \dim\mathcal{V} - \dim(\mathcal{K}+\mathcal{V}) \geq (n-k)+(k+1)-n = 1,$$

where the equality is the dimension formula for two subspaces and the inequality uses $\dim(\mathcal{K}+\mathcal{V}) \leq n$. To see the formula, extend a basis of the intersection to a basis of $\mathcal{K}$ and, separately, to a basis of $\mathcal{V}$. The vectors obtained span $\mathcal{K}+\mathcal{V}$ and are independent, and there are $\dim\mathcal{K} + \dim\mathcal{V} - \dim(\mathcal{K}\cap\mathcal{V})$ of them. So there is a unit vector $\mathbf{x} \in \mathcal{K}\cap\mathcal{V}$. Write it as $\mathbf{x} = \sum_{i\leq k+1} c_i\mathbf{v}_i$ with $\sum_i c_i^2 = 1$. Then $\mathbf{B}\mathbf{x} = \mathbf{0}$, and since $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ with orthonormal $\mathbf{u}_i$,

$$\|\mathbf{A}\mathbf{x}\|^2 = \Big\|\sum_{i\leq k+1} c_i\sigma_i\mathbf{u}_i\Big\|^2 = \sum_{i\leq k+1} c_i^2\sigma_i^2 \geq \sigma_{k+1}^2\sum_{i\leq k+1} c_i^2 = \sigma_{k+1}^2.$$

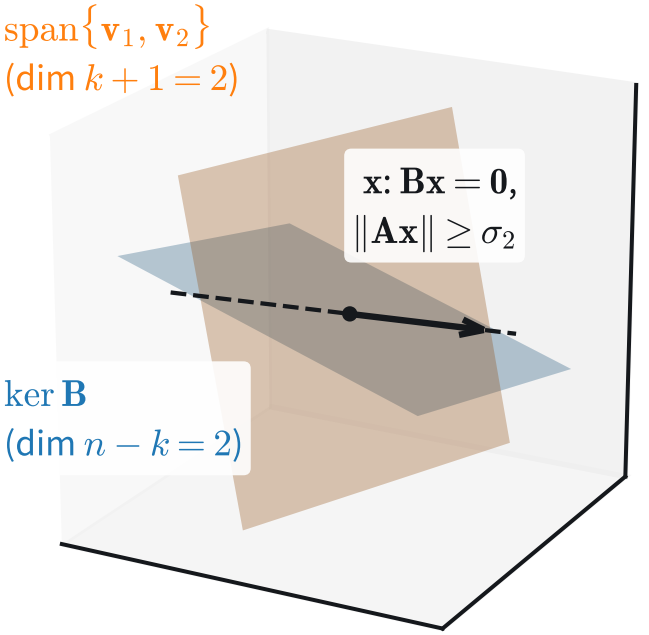


Figure 33.24: The dimension count behind Eckart–Young for $n = 3$, $k = 1$: the plane $\ker \mathbf{B}$ and the plane spanned by the top two right singular vectors must meet in a line. On that line $\mathbf{B}$ vanishes while $\mathbf{A}$ still stretches by at least σ_2.

Therefore $\|\mathbf{A}-\mathbf{B}\|_2 \geq \|(\mathbf{A}-\mathbf{B})\mathbf{x}\| = \|\mathbf{A}\mathbf{x}\| \geq \sigma_{k+1}$. ■

In words: $\mathbf{B}$ must annihilate some direction in the span of the top $k+1$ right singular vectors, and $\mathbf{A}$ stretches every direction of that span by at least σ_{k+1}. For the shear and $k=1$, the truncation $\mathbf{T}_1 = \varphi\,\mathbf{u}_1\mathbf{v}_1^\top = \frac{1}{\sqrt{5}}\begin{pmatrix}\varphi & \varphi^2\\ 1 & \varphi\end{pmatrix} \approx \begin{pmatrix}0.724 & 1.171\\ 0.447 & 0.724\end{pmatrix}$ misses by $1/\varphi = 0.618$ in both norms (the residual has rank one, so its two norms agree), and no matrix of rank one comes closer.

Half of the theorem is still unproved. The lower bound just shown yields the inequality on which the Frobenius case rests.

Lemma (Weyl's inequality for singular values). *For matrices* $\mathbf{X}, \mathbf{Y}$ *of the same shape and indices* $i, j \geq 1$ *with* $i + j - 1 \leq \min(m, n)$,

$$\sigma_{i+j-1}(\mathbf{X}+\mathbf{Y}) \leq \sigma_i(\mathbf{X}) + \sigma_j(\mathbf{Y}). \tag{33.53}$$

Proof. Let $\mathbf{X}_{i-1}$ and $\mathbf{Y}_{j-1}$ be the truncations Equation 33.50 (with $\mathbf{X}_0 = \mathbf{0}$). Their sum has rank at most $i+j-2$, so the lower bound just proved, applied to $\mathbf{X}+\mathbf{Y}$ with $k = i+j-2$, gives $\sigma_{i+j-1}(\mathbf{X}+\mathbf{Y}) \leq \|\mathbf{X}+\mathbf{Y}-\mathbf{X}_{i-1}-\mathbf{Y}_{j-1}\|_2$. By the triangle inequality and the achievability half, this is at most $\|\mathbf{X}-\mathbf{X}_{i-1}\|_2 + \|\mathbf{Y}-\mathbf{Y}_{j-1}\|_2 = \sigma_i(\mathbf{X}) + \sigma_j(\mathbf{Y})$. ■

Adding $\mathbf{Y}$ can raise the $(i+j-1)$-th singular value above the i-th singular value of $\mathbf{X}$ by at most $\sigma_j(\mathbf{Y})$. With $j=1$, applied in both directions, no singular value moves by more than $\|\mathbf{Y}\|_2$ (Horn and Johnson 2012); Exercise 3 works this out.

Proof (Frobenius norm). Apply the lemma with $\mathbf{X} = \mathbf{A}-\mathbf{B}$, $\mathbf{Y} = \mathbf{B}$ and $j = k+1$. The hypothesis $\operatorname{rank}\mathbf{B} \leq k$ enters here, as $\sigma_{k+1}(\mathbf{B}) = 0$, and gives $\sigma_{k+i}(\mathbf{A}) \leq \sigma_i(\mathbf{A}-\mathbf{B})$ for every i. Squaring and summing over i, with Equation 33.51,

$$\|\mathbf{A}-\mathbf{B}\|_F^2 = \sum_{i\geq 1}\sigma_i(\mathbf{A}-\mathbf{B})^2 \geq \sum_{i\geq 1}\sigma_{k+i}(\mathbf{A})^2 = \sum_{j>k}\sigma_j(\mathbf{A})^2,$$

which the truncation attains. ■

Uniqueness of the minimizer. In the spectral norm the best approximation is never unique when $k \geq 1$ and $\sigma_{k+1} > 0$. For $\mathbf{A} = \operatorname{diag}(3,2,1)$ and $k=1$, every $\mathbf{B} = \operatorname{diag}(3+t, 0, 0)$ with $|t| \leq 1$ has $\|\mathbf{A}-\mathbf{B}\|_2 = \max(|t|, 2, 1) = 2 = \sigma_2$. In the Frobenius norm, $\mathbf{A}_k$ is the unique minimizer exactly when $\sigma_k > \sigma_{k+1}$ or $\sigma_{k+1} = 0$ (then $\mathbf{A}_k = \mathbf{A}$) (Golub and Van Loan 1996). This is one reason principal component analysis, below, is stated in the Frobenius norm.

Now the photograph. The theorem fixes the spectral error of every truncation in advance, at σ_{k+1}. It does not say how fast the error falls with k, nor how far a plausible competitor of the same rank falls behind. The cell reads `img/cat1.jpg` and converts it to grayscale with the usual weights for red, green and blue. It forms the truncations with `einsum`, which multiplies its arrays and sums over every index that does not appear after the arrow. Since row i of `Vt` is $\mathbf{v}_i^\top$, the pattern `'i,mi,in->mn'` is the dyadic sum.

```
img = d2l.plt.imread('../img/cat1.jpg') @ np.array([0.299, 0.587, 0.114])  # 400 x 500
U, s, Vt = np.linalg.svd(img, full_matrices=False)
m, n = img.shape
fro = np.linalg.norm(img)
```

```
energy = np.cumsum(s**2) / (s**2).sum()                          # energy kept at each rank
slope = np.polyfit(np.log(np.arange(1, 101)), np.log(s[:100]), 1)[0]
print(f'{m} x {n} = {m * n} numbers;  slope of log sigma_i on log i: {slope:.2f}')
print(f'energy kept at rank 1: {energy[0]:.3f}, at rank 4: {energy[3]:.3f}')
for k in [1, 5, 20, 50]:
    A_k = np.einsum('i,mi,in->mn', s[:k], U[:, :k], Vt[:k])    # first k dyads
    print(f'k={k:2d}: ||A-A_k||_2={np.linalg.norm(img - A_k, 2):6.0f}  '
          f'sigma_{k + 1}={s[k]:6.0f}  rel. err {np.linalg.norm(img - A_k) / fro:.3f}  '
          f'stores {k * (m + n + 1) / (m * n):.1%}')
cols = np.argsort(-np.linalg.norm(img, axis=0))[:20]             # 20 largest columns
C = img[:, cols]
proj = C @ np.linalg.pinv(C) @ img
print(f'projection onto the 20 largest columns: rel. err {np.linalg.norm(img - proj) /
↪  fro:.3f}')
```

```
400 x 500 = 200000 numbers;  slope of log sigma_i on log i: -1.11
energy kept at rank 1: 0.886, at rank 4: 0.955
k= 1: ||A-A_k||_2=  9234  sigma_2=  9234  rel. err 0.337  stores 0.5%
k= 5: ||A-A_k||_2=  4048  sigma_6=  4048  rel. err 0.191  stores 2.3%
k=20: ||A-A_k||_2=   874  sigma_21=   874  rel. err 0.080  stores 9.0%
k=50: ||A-A_k||_2=   342  sigma_51=   342  rel. err 0.045  stores 22.5%
projection onto the 20 largest columns: rel. err 0.310
```

The spectral error equals σ_{k+1} at every rank, as it must, and rank 20 stores 9% of the numbers at a relative error of 8%. In Figure 33.25 the cat is a bright blur at rank 1, a pale body with four paws at rank 5, and plainly a cat lying on its back at rank 20. The left panel explains why so few pieces suffice: over the first hundred indices the singular values fall roughly as a power law, $\sigma_i \propto i^{-1.1}$.

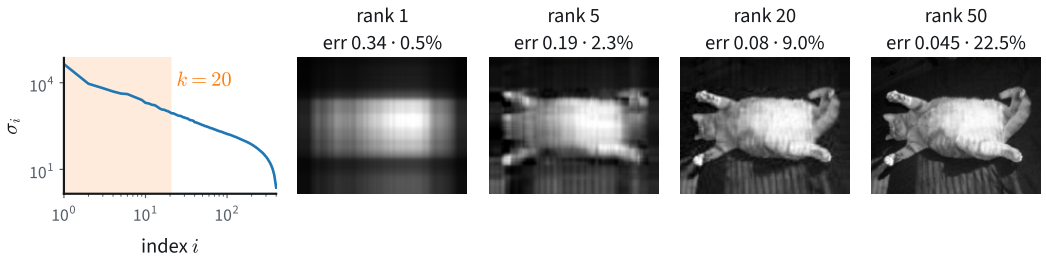


Figure 33.25: Eckart–Young on a 400×500 photograph. Left: the singular values fall roughly as a power law (log–log axes; the shaded band marks the twenty kept in the rank-20 panel). Right: the truncations $\mathbf{A}_k$ with their relative Frobenius error and the fraction $k(m+n+1)/mn$ of the original numbers they store.

The energy ratio. The Frobenius formula gives a rule for choosing the rank. The fraction of the "energy" $\sum_i \sigma_i^2$ that the truncation keeps is

$$\frac{\sum_{i \le k} \sigma_i^2}{\sum_i \sigma_i^2} = 1 - \frac{\|\mathbf{A} - \mathbf{A}_k\|_F^2}{\|\mathbf{A}\|_F^2}, \tag{33.54}$$

so the relative Frobenius error is the square root of one minus the energy ratio. Keeping

95% of the energy means a relative error of $\sqrt{0.05} \approx 22\%$, and keeping 99% means 10%. The ratio needs care. On the photograph σ_1^2 alone holds 89% of the energy. A constant image has rank one (all its rows are equal), and the photograph's mean brightness is large, so the first dyad is mostly that mean. Rank 4 already keeps 95%, at a relative error of 21%. Once the mean pixel value is subtracted from every pixel, 95% takes rank 8 instead of 4; the cell of Section 33.5.4.3 computes this count. Read the ratio with σ_1 in mind, or subtract the mean first, as principal component analysis does.

The competitor in the last line of the cell keeps 20 actual columns of the photograph, the 20 with the largest norms, and projects every column onto their span. This is the idea behind CUR decompositions, which approximate a matrix by its own rows and columns. Its relative error is 0.31 against 0.08 for the truncation. The theorem says only that no matrix of rank 20 does better; here a natural one is about four times worse.

Truncation as denoising. A second reason to truncate is statistical. Suppose the observed $m \times n$ matrix ($m \le n$) is a low-rank signal plus noise, $\mathbf{A} = \mathbf{B} + \mathbf{N}$, where the entries of $\mathbf{N}$ are independent with mean zero and standard deviation σ_{noise}. By the Marchenko–Pastur law of Section 33.3.2.4, applied to $\mathbf{N}\mathbf{N}^\top/n$, the singular values of a large noise matrix concentrate in the interval $[(\sqrt{n} - \sqrt{m})\sigma_{\text{noise}}, (\sqrt{n} + \sqrt{m})\sigma_{\text{noise}}]$. Signal components whose singular values rise well above the upper edge stand out from the noise; weaker ones are lost in it. For square matrices with a known noise level, Gavish and Donoho show that the asymptotically best hard threshold is $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}} \approx 2.309\sqrt{n}\,\sigma_{\text{noise}}$, somewhat above the edge $2\sqrt{n}\,\sigma_{\text{noise}}$ (Gavish and Donoho 2014). Exercise 4 shows the separation, and Exercise 11 derives the constant $4/\sqrt{3}$.

33.5.2.2 Principal Component Analysis

Let the rows of $\mathbf{X} \in \mathbb{R}^{n \times d}$ be data points; here n counts the points and d the features. Which unit direction captures the most variance of the data, and which k-dimensional affine subspace fits the points best?

Centering. Fix the directions, an orthogonal projector $\mathbf{P}$ onto a k-dimensional subspace, and ask which offset $\mathbf{c}$ makes the squared reconstruction error $\sum_i \|(\mathbf{I}-\mathbf{P})(\mathbf{x}_i-\mathbf{c})\|^2$ smallest. This error is a convex quadratic in $\mathbf{c}$, and its gradient $-2(\mathbf{I}-\mathbf{P})\sum_i(\mathbf{x}_i - \mathbf{c})$ vanishes at the mean $\mathbf{c} = \bar{\mathbf{x}}$, whatever $\mathbf{P}$ is. So the best affine fit passes through the mean, and we first *center* the data, $\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}\bar{\mathbf{x}}^\top$. The empirical covariance is the symmetric positive semidefinite matrix $\mathbf{C} = \frac{1}{n}\tilde{\mathbf{X}}^\top\tilde{\mathbf{X}}$. (The unbiased $\frac{1}{n-1}$ version rescales every eigenvalue by the same factor, so it changes neither the principal directions nor the explained-variance ratio.)

Proposition (PCA via Rayleigh). *The projected variance $\frac{1}{n}\|\tilde{\mathbf{X}}\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{C}\mathbf{w}$ is maximized over unit $\mathbf{w}$ by the top eigenvector of $\mathbf{C}$, which is the top right singular vector $\mathbf{v}_1$ of $\tilde{\mathbf{X}}$; the maximal variance is σ_1^2/n.*

Proof. The projection of a centered point onto $\mathbf{w}$ has the coordinate $\tilde{\mathbf{x}}^\top\mathbf{w}$, and the variance of these coordinates is $\frac{1}{n}\sum_i(\tilde{\mathbf{x}}_i^\top\mathbf{w})^2 = \frac{1}{n}\|\tilde{\mathbf{X}}\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{C}\mathbf{w}$. By the Rayleigh proposition of Section 33.3.2.3, this is maximized by the top eigenvector of $\mathbf{C}$, with value $\lambda_1(\mathbf{C})$. By Equation 33.48, $\mathbf{C} = \frac{1}{n}\mathbf{V}\Sigma^\top\Sigma\mathbf{V}^\top$, so the eigenvectors of $\mathbf{C}$ are the right singular vectors $\mathbf{v}_i$ of $\tilde{\mathbf{X}}$ and its eigenvalues are σ_i^2/n. ■

Among unit vectors orthogonal to $\mathbf{v}_1$, the variance is largest along $\mathbf{v}_2$, by the same argument on the remaining eigenvectors, and so on. The top k principal directions are there-

fore $\mathbf{v}_1, \dots, \mathbf{v}_k$, and component i explains the variance σ_i^2/n. These directions also give the best reconstruction. Projecting every centered point onto their span is right-multiplication by $\mathbf{V}_k\mathbf{V}_k^\top$, and since $\mathbf{V}^\top\mathbf{V}_k$ is the first k columns of the identity,

$$\tilde{\mathbf{X}}\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{U}\Sigma\mathbf{V}^\top\mathbf{V}_k\mathbf{V}_k^\top = \mathbf{U}_k\Sigma_k\mathbf{V}_k^\top = \tilde{\mathbf{X}}_k.$$

The projection onto the top k principal directions is the truncated SVD of the centered data. By Eckart–Young in the Frobenius norm, no other projection, and no other matrix of rank k, reconstructs the centered points with a smaller squared error. The discarded variance $\sum_{i>k}\sigma_i^2/n$ is the mean squared reconstruction error, and the explained-variance ratio is the energy ratio Equation 33.54 of $\tilde{\mathbf{X}}$.

Figure 33.26 draws a cloud of 300 points with standard deviations 3 and 0.8 along axes rotated by $\pi/5 = 36^\circ$, and its two principal axes. For such a cloud the first direction should hold about $9/9.64 = 93\%$ of the variance and lie along the long axis. How closely 300 points reproduce these population values is what the cell measures on a fresh draw.

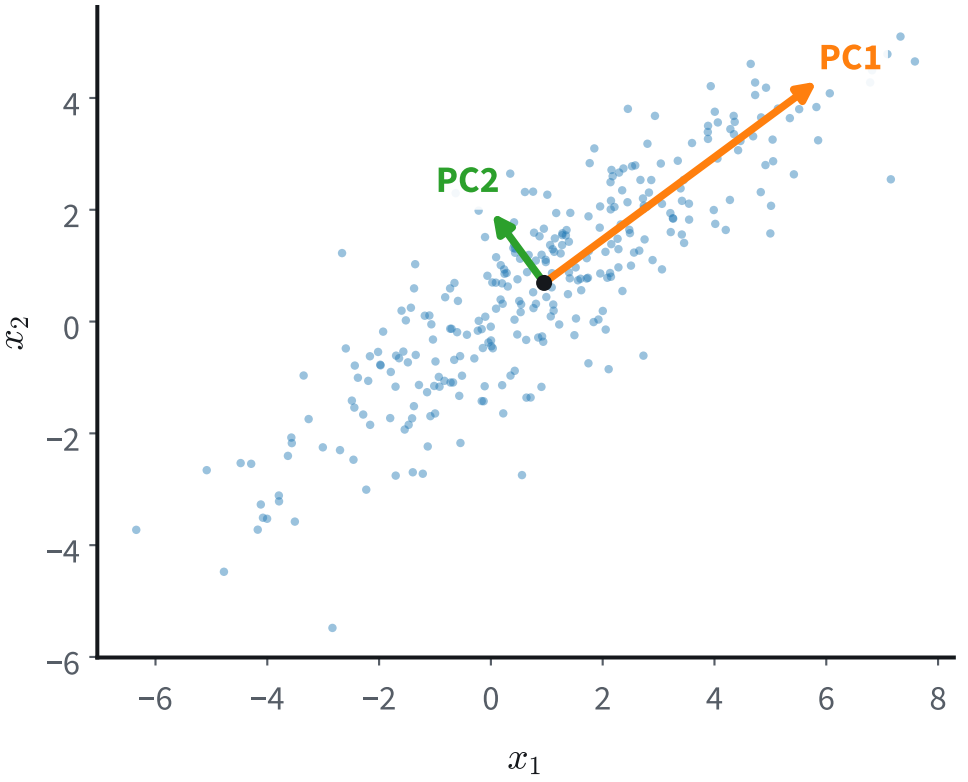


Figure 33.26: A cloud of 300 points centered at $(1, 0.6)$ with standard deviations 3 and 0.8 along axes rotated by 36°, and its principal axes drawn from the mean, each two standard deviations $2\sigma_i/\sqrt{n}$ long. The first axis points along the direction of largest variance.

```
theta = np.pi / 5
rot = np.array([[np.cos(theta), -np.sin(theta)], [np.sin(theta), np.cos(theta)]])
X = np.random.standard_normal((300, 2)) * np.array([3.0, 0.8]) @ rot.T   # the cloud
Xc = X - X.mean(0)
U, s, Vt = np.linalg.svd(Xc, full_matrices=False)
X1 = np.outer(U[:, 0] * s[0], Vt[0])                                    # rank-1 reconstruction
print('explained-variance ratio:', (s**2 / (s**2).sum()).round(3), ' (population 0.934,
↪  0.066)')
print('angle of v_1 to the long axis:',
      np.degrees(np.arccos(abs(Vt[0] @ rot[:, 0]))).round(1), 'degrees')
print('mean squared rank-1 error:', (((Xc - X1)**2).sum() / 300).round(4),
      '  sigma_2^2 / n:', (s[1]**2 / 300).round(4))
```

```
explained-variance ratio: [0.923 0.077]  (population 0.934, 0.066)
angle of v_1 to the long axis: 0.6 degrees
mean squared rank-1 error: 0.7111   sigma_2^2 / n: 0.7111
```

From draw to draw the first direction explains between about 91% and 95% of the variance, and it lies within a few degrees of the long axis. Without centering, the first singular vector turns toward the mean of the cloud once the offset of the mean exceeds the spread of the data. It then describes where the data are rather than how they vary, and Exercise 5 quantifies the turn.

33.5.3 Solving Linear Systems with the SVD

33.5.3.1 The Pseudoinverse and Least Squares

A system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with more equations than unknowns usually has no solution, and one with fewer equations has infinitely many. In both cases we ask for the least-squares solution of Section 33.1, with the target written $\mathbf{b}$ instead of $\mathbf{y}$,

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2,$$

and when several $\mathbf{x}$ minimize it, for the shortest one.

Take the smallest case first, one equation in two unknowns, $x_1 + x_2 = 2$. Its solutions form a line, and the residual is zero on all of it. Which point should we report? Here $\mathbf{A} = (1 \quad 1)$, with the single singular value $\sigma_1 = \sqrt{2}$, right singular vector $\mathbf{v}_1 = (1,1)/\sqrt{2}$ and $u_1 = 1$. The coordinate of $\mathbf{x}$ along $\mathbf{v}_1$ is fixed by the equation, $\sqrt{2}$. The coordinate along $\mathbf{v}_2 = (1,-1)/\sqrt{2}$ does not affect the residual at all. Setting it to zero gives $(1,1)$, the foot of the perpendicular from the origin to the line, with norm $\sqrt{2}$; every other solution $(1+t, 1-t)$ has squared norm $2 + 2t^2$ (Figure 33.27).

The general rule inverts the nonzero singular values and transposes the rotations. The *Moore–Penrose pseudoinverse* of $\mathbf{A}$ is

$$\mathbf{A}^{+} = \mathbf{V}\Sigma^{+}\mathbf{U}^{\top}, \qquad \Sigma^{+}_{ii} = \begin{cases} 1/\sigma_i & \sigma_i > 0, \\ 0 & \sigma_i = 0, \end{cases} \tag{33.55}$$

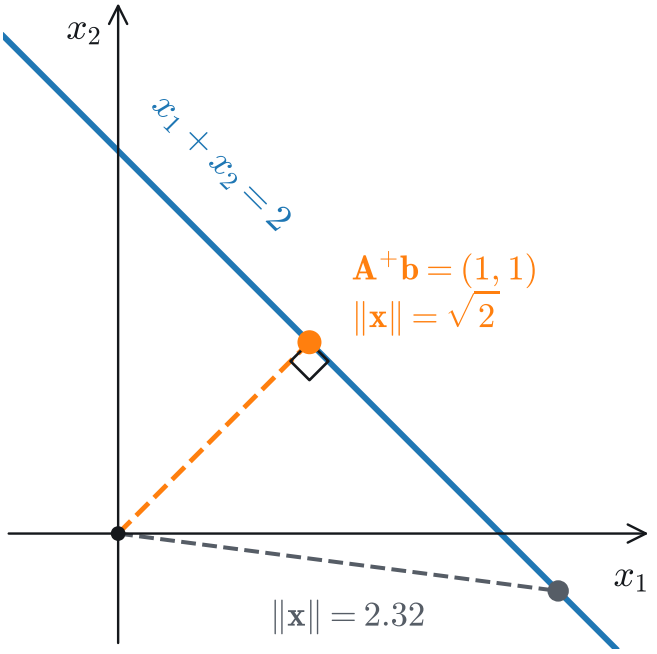


Figure 33.27: Least squares with one equation and two unknowns. Every point of the line solves $x_1 + x_2 = 2$; the pseudoinverse returns the one closest to the origin, the foot of the perpendicular.

where Σ^+ is $n \times m$, the transpose shape of Σ. It undoes each stretch that can be undone and sends the left null space to zero. For the shear, $\mathbf{T}^+ = \mathbf{T}^{-1} = \left(\begin{smallmatrix} 1 & -1 \\ 0 & 1 \end{smallmatrix}\right)$; for $\mathbf{A} = \begin{pmatrix} 1 & 1 \end{pmatrix}$, $\mathbf{A}^+ = \mathbf{v}_1 \sigma_1^{-1} u_1 = (\frac{1}{2}, \frac{1}{2})^\top$, and $\mathbf{A}^+ \cdot 2 = (1, 1)$.

Proposition (minimum-norm least squares). *Among all minimizers of $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|$, the one of smallest norm is $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{b}$.*

Proof. Rotate the unknown and the target into the singular bases, $\mathbf{y} = \mathbf{V}^\top \mathbf{x}$ and $\mathbf{c} = \mathbf{U}^\top \mathbf{b}$. Since $\mathbf{U}$ is orthogonal it preserves norms, so

$$\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 = \|\mathbf{U}^\top(\mathbf{A}\mathbf{x} - \mathbf{b})\|^2 = \|\Sigma\mathbf{y} - \mathbf{c}\|^2 = \sum_{i \le r} (\sigma_i y_i - c_i)^2 + \sum_{i > r} c_i^2.$$

The rotation has decoupled the problem into scalar problems. The first sum is minimized term by term at $y_i = c_i/\sigma_i$ for $i \le r$. The second sum is a constant, the part of $\mathbf{b}$ outside the column space, which no $\mathbf{x}$ can reduce. The coordinates y_i with $i > r$ do not appear in the residual at all, and since $\|\mathbf{x}\| = \|\mathbf{y}\|$, setting them to zero gives the unique minimizer of smallest norm. That choice is $\mathbf{y} = \Sigma^+\mathbf{c}$, that is, $\hat{\mathbf{x}} = \mathbf{V}\Sigma^+\mathbf{U}^\top\mathbf{b} = \mathbf{A}^+\mathbf{b}$. ■

In the singular bases, a small residual and a small norm concern disjoint coordinates, $i \le r$ and $i > r$, so both can be had at once. When $\mathbf{A}$ is square and invertible, every $\sigma_i > 0$ and $\mathbf{A}^+ = \mathbf{V}\Sigma^{-1}\mathbf{U}^\top = \mathbf{A}^{-1}$. The formula also shows where least squares goes wrong. A tiny σ_i is divided into c_i, so any noise in $\mathbf{b}$ along $\mathbf{u}_i$ is multiplied by $1/\sigma_i$.

Two remedies damp these components. *Ridge regression* minimizes $\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2 + \lambda\|\mathbf{x}\|^2$ with $\lambda > 0$ and has the solution $\mathbf{x}_\lambda = (\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^\top\mathbf{b}$. The *truncated pseudoinverse* solution $\mathbf{x}_k$ keeps only the first k singular components of $\mathbf{A}^+\mathbf{b}$. Inserting the SVD, with $\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I} = \mathbf{V}(\Sigma^\top\Sigma + \lambda\mathbf{I})\mathbf{V}^\top$, the two solutions read

$$\mathbf{x}_\lambda = \sum_{i \le r} \frac{\sigma_i^2}{\sigma_i^2 + \lambda} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i} \mathbf{v}_i, \qquad \mathbf{x}_k = \sum_{i \le k} \frac{\mathbf{u}_i^\top \mathbf{b}}{\sigma_i} \mathbf{v}_i. \tag{33.56}$$

Both start from the components $\mathbf{u}_i^\top\mathbf{b}/\sigma_i$ of the pseudoinverse solution. Ridge multiplies component i by the filter factor $\sigma_i^2/(\sigma_i^2 + \lambda)$, which is close to 1 for large σ_i and close to 0 for small ones; truncation multiplies it by exactly 1 or 0. At $\lambda = 1$, singular values 3, 1 and 0.3 get the factors 0.9, 0.5 and 0.083. For a linear model with squared loss, the weight decay of Section 2.7 is ridge regression, and the filter factors show which directions it shrinks.

The normal equations. The classical route solves $\mathbf{A}^\top\mathbf{A}\mathbf{x} = \mathbf{A}^\top\mathbf{b}$. For every $\mathbf{A}$ its solutions are the least-squares minimizers, since $\mathbf{A}^\top(\mathbf{A}\mathbf{x} - \mathbf{b}) = \mathbf{0}$ says the residual is orthogonal to the column space. The solution is unique when $\mathbf{A}$ has full column rank. Numerically this route is worse, because forming $\mathbf{A}^\top\mathbf{A}$ squares the condition number (Section 33.5.3.2). Libraries therefore solve least squares through the SVD or a QR factorization.

The cell applies the pseudoinverse to one system of each kind. The first has too many equations: it fits a line $b \approx x_1 + x_2 t$ to the values $\mathbf{b} = (6, 5, 7, 10)$ at $t = 1, 2, 3, 4$. The second is the single equation above.

```
A = np.array([[1., 1.], [1., 2.], [1., 3.], [1., 4.]])     # 4 equations, 2 unknowns
b = np.array([6., 5., 7., 10.])
```

```
x_pinv = np.linalg.pinv(A) @ b
x_lstsq = np.linalg.lstsq(A, b, rcond=None)[0]
print('overdetermined: pinv', x_pinv.round(4), ' lstsq', x_lstsq.round(4),
      ' residual', np.linalg.norm(A @ x_pinv - b).round(4))
A1 = np.array([[1., 1.]])                                  # 1 equation: x1 + x2 = 2
x1 = np.linalg.pinv(A1) @ np.array([2.])
print('underdetermined: pinv gives', x1.round(4), ' norm', np.linalg.norm(x1).round(4))
```

```
overdetermined: pinv [3.5 1.4]  lstsq [3.5 1.4]  residual 2.0494
underdetermined: pinv gives [1. 1.]  norm 1.4142
```

The pseudoinverse and the library's least-squares solver agree on the line $3.5 + 1.4t$, with residual $\sqrt{4.2} \approx 2.049$, and for the single equation the pseudoinverse returns the foot of the perpendicular, $(1, 1)$.

33.5.3.2 The Condition Number

The data of a linear system are measured or rounded, so before trusting a solution we need to know how much a small error in the data can change it. A 2×2 system shows how large the effect can be. Consider the system $\mathbf{K}\mathbf{x} = \mathbf{b}$ with $\mathbf{K} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1.0001 \end{smallmatrix}\right)$ and $\mathbf{b} = (2, 2.0001)$. Its solution is $\mathbf{x} = (1, 1)$. Change b_2 by 10^{-4}, to 2.0002, and the solution becomes $(0, 2)$. A change of about three parts in 10^5 in the data has changed the answer completely. The singular values of $\mathbf{K}$, about 2.00005 and 5×10^{-5}, explain it.

The *condition number* of a matrix of rank r is the ratio of its largest to its smallest nonzero singular value,

$$\kappa(\mathbf{A}) = \frac{\sigma_1}{\sigma_r}. \tag{33.57}$$

It says how unequal the stretches of $\mathbf{A}$ are; for $\mathbf{K}$ it is about 4×10^4, for the shear $\varphi^2 \approx 2.618$. For a symmetric positive definite matrix the singular values are the eigenvalues, so $\kappa = \lambda_{\max}/\lambda_{\min}$, the form used for quadratic bowls in Section 33.3.2.3. The condition number measures how much a linear solve can amplify an error in the data.

Proposition (error amplification). *Let $\mathbf{A}$ be square and invertible, $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{b} \neq \mathbf{0}$, and let a perturbation $\delta\mathbf{b}$ change the solution to $\mathbf{x} + \delta\mathbf{x}$. Then*

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \kappa(\mathbf{A}) \, \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}, \tag{33.58}$$

and for every $\mathbf{A}$ some $\mathbf{b}$ and $\delta\mathbf{b}$ attain the bound.

Proof. From $\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{b}$ we get $\delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b}$, and the largest singular value of $\mathbf{A}^{-1} = \mathbf{V}\Sigma^{-1}\mathbf{U}^{\top}$ is $1/\sigma_n$, so by Equation 33.47 $\|\delta\mathbf{x}\| \le \sigma_n^{-1}\|\delta\mathbf{b}\|$. From $\mathbf{b} = \mathbf{A}\mathbf{x}$ we get $\|\mathbf{b}\| \le \sigma_1\|\mathbf{x}\|$. Multiplying the two inequalities,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \le \frac{\sigma_1}{\sigma_n}\,\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|} = \kappa(\mathbf{A})\,\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

Both inequalities are equalities for $\mathbf{b} = \mathbf{u}_1$, whose solution $\mathbf{x} = \sigma_1^{-1}\mathbf{v}_1$ is as short as a unit right-hand side allows, and $\delta\mathbf{b} = \varepsilon\mathbf{u}_n$, whose effect $\delta\mathbf{x} = \varepsilon\sigma_n^{-1}\mathbf{v}_n$ is as large as a perturbation of that size allows. ■

The relative error of the data is multiplied by at most κ. With $\kappa = 10^4$, a relative error of 10^{-8} in $\mathbf{b}$ can become 10^{-4} in $\mathbf{x}$, a loss of four digits. In the example, the relative change of $\mathbf{b}$ is $10^{-4}/\|\mathbf{b}\| \approx 3.5 \times 10^{-5}$, so the bound allows a relative change of 1.41 in $\mathbf{x}$. The realized change is 1.0. The perturbation $(0, 10^{-4})$ has only $1/\sqrt{2}$ of its length along the most amplified direction $\mathbf{u}_2 \approx (1, -1)/\sqrt{2}$, while $\mathbf{b}$ lies almost along $\mathbf{u}_1$.

An orthogonal matrix has all singular values equal to 1, so $\kappa = 1$. Rotations and reflections therefore never amplify error. For this reason the factors $\mathbf{U}$ and $\mathbf{V}$ are safe to compute with, and a weight matrix initialized as an orthogonal matrix passes every input on with its length unchanged. At the other extreme, forming the Gram matrix squares the condition number,

$$\kappa(\mathbf{A}^\top\mathbf{A}) = \kappa(\mathbf{A})^2,$$

because Equation 33.48 is the SVD of $\mathbf{A}^\top\mathbf{A}$, with singular values σ_i^2. This holds for every $\mathbf{A}$, of any rank. For the shear, κ goes from $\varphi^2 \approx 2.62$ to $\varphi^4 \approx 6.85$. The cell solves the 2×2 example and computes both condition numbers for the line fit of the previous subsection.

```
K = np.array([[1., 1.], [1., 1.0001]])
b = np.array([2., 2.0001])
x = np.linalg.solve(K, b)
x2 = np.linalg.solve(K, b + np.array([0., 1e-4]))
print('x =', x.round(4), '  after changing b_2 by 1e-4:', x2.round(4) + 0.)
kappa = np.linalg.cond(K)
print(f'kappa(K) = {kappa:.0f};  bound {kappa * 1e-4 / np.linalg.norm(b):.2f};  '
      f'realized {np.linalg.norm(x2 - x) / np.linalg.norm(x):.2f}')
A = np.array([[1., 1.], [1., 2.], [1., 3.], [1., 4.]])
print(f'kappa(A) = {np.linalg.cond(A):.3f};  kappa(A^T A) = {np.linalg.cond(A.T @ A):.2f};  '
      f'kappa(A)^2 = {np.linalg.cond(A)**2:.2f}')
```

```
x = [1. 1.]   after changing b_2 by 1e-4: [0. 2.]
kappa(K) = 40002;  bound 1.41;  realized 1.00
kappa(A) = 7.469;  kappa(A^T A) = 55.78;  kappa(A)^2 = 55.78
```

The numbers are those computed by hand: the solution jumps from $(1, 1)$ to $(0, 2)$, with $\kappa(\mathbf{K}) \approx 40{,}002$. For the 4×2 matrix of the line fit, $\kappa = 7.47$, and the normal equations would work with $\kappa^2 = 55.8$.

Condition and the shape of a bowl. A large κ also means elongated level sets. For least squares with $\mathbf{A}$ of full column rank, $\frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$ has the Hessian $\mathbf{H} = \mathbf{A}^\top\mathbf{A}$, and its level sets are ellipsoids with semi-axes proportional to $1/\sigma_i$ along the $\mathbf{v}_i$. The contours therefore have axis ratio $\kappa(\mathbf{A})$, while the Hessian has condition number $\kappa(\mathbf{A})^2$. Gradient descent is slow on such a bowl (Figure 33.28), at a rate set by $\kappa(\mathbf{H})$ that Section 35.1 derives.

33.5.4 The SVD in Deep Learning

Three questions of large-model training reduce to the singular values of a weight matrix: how many parameters a fine-tuning adapter needs, which update direction treats every direction of the weight space equally, and how much a layer can stretch its input.

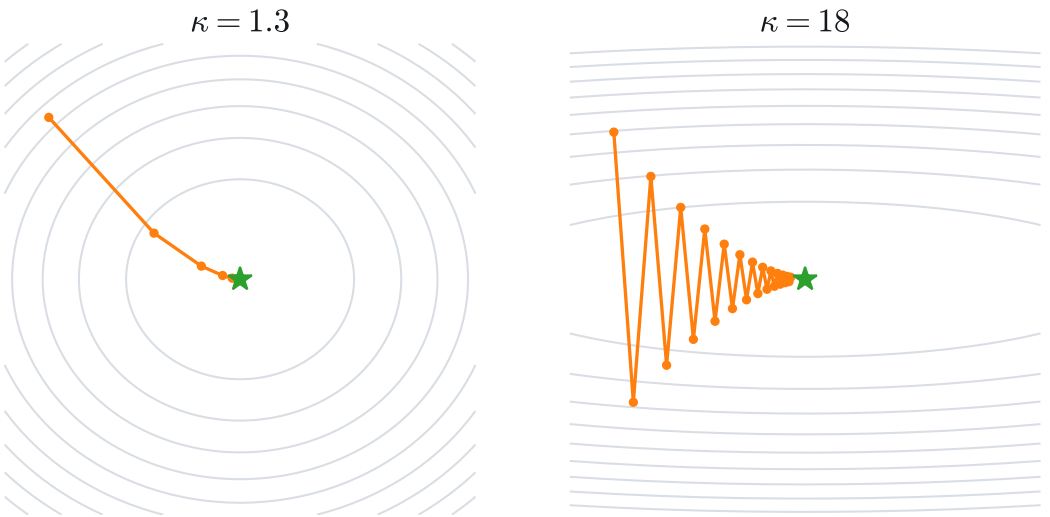


Figure 33.28: Gradient descent on a well-conditioned quadratic bowl ($\kappa = 1.3$, near-circular contours, an almost straight path) versus an ill-conditioned one ($\kappa = 18$, elongated contours, a zig-zag trajectory across the narrow valley).

33.5.4.1 Low-Rank Updates

Fine-tuning every weight of a large pretrained model is expensive. LoRA (Hu et al. 2021) freezes a pretrained weight $\mathbf{W} \in \mathbb{R}^{m\times n}$ and learns only a correction of rank k, $\Delta\mathbf{W} = \mathbf{BA}$ with $\mathbf{B} \in \mathbb{R}^{m\times k}$ and $\mathbf{A} \in \mathbb{R}^{k\times n}$; here $\mathbf{A}$ is the paper's $k \times n$ down-projection, not the generic matrix of this section. The adapter has $k(m+n)$ trainable parameters instead of mn. For a 4096×4096 layer and $k = 8$ that is $8 \cdot 8192 = 65{,}536$ of $16{,}777{,}216$, or 0.39%. Figure 33.29 shows the layer: the frozen weight and the low-rank bypass act on the same input, and their outputs add.

What does Eckart–Young say about this? It bounds how well any update of rank k can reproduce the update that full fine-tuning would have made. If that update has singular values $\sigma_1 \geq \sigma_2 \geq \cdots$, no adapter of rank k comes closer than σ_{k+1} in the spectral norm, or $(\sum_{i>k} \sigma_i^2)^{1/2}$ in the Frobenius norm. It says nothing about the adapter that training finds. That adapter is learned by gradient descent, and it succeeds when some low-rank update reaches a low loss, which need not be the one that best reproduces the full update. That such updates exist for many fine-tuning tasks is the empirical finding behind LoRA. Exercise 3 bounds how far an adapter of rank k can move the singular values of $\mathbf{W}$.

33.5.4.2 Orthogonalized Updates

The optimizer Muon (Jordan et al. 2024) steps along the *polar factor* of its momentum matrix $\mathbf{M}$, the matrix with the singular vectors of $\mathbf{M}$ and every stretch set to 1, instead of along $\mathbf{M}$ itself. Section 9.7 gives the reasons and the optimizer. Here we supply the mathematics it uses: what the polar factor is, and why matrix products alone can compute it.

The polar decomposition. For a square matrix $\mathbf{A}$, inserting $\mathbf{V}^\top\mathbf{V} = \mathbf{I}$ into the SVD splits it into an orthogonal factor and a stretch:

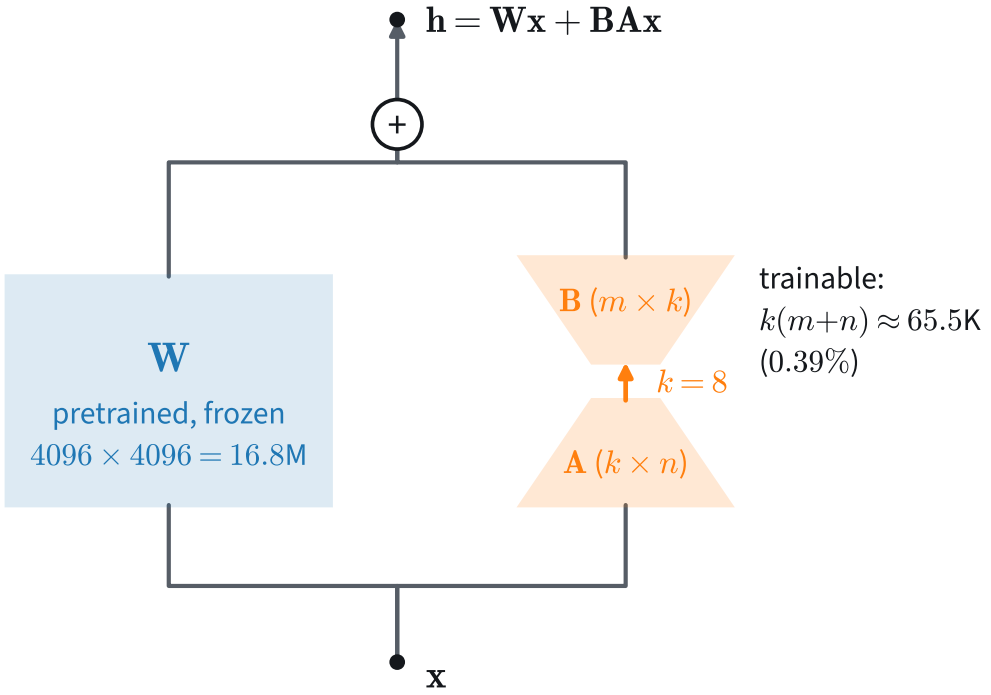


Figure 33.29: The LoRA architecture for one layer. The pretrained $\mathbf{W}$ (all $mn = 16.8\text{M}$ parameters for $m = n = 4096$) stays frozen; a parallel bypass first compresses the input to $k = 8$ dimensions with $\mathbf{A}$ and then expands back with $\mathbf{B}$, so only $k(m + n) \approx 65.5\text{K}$ parameters (about 0.39%) are trained. The two branches add: $\mathbf{h} = \mathbf{W}\mathbf{x} + \mathbf{B}\mathbf{A}\mathbf{x}$. After training, $\mathbf{B}\mathbf{A}$ can be merged into $\mathbf{W}$, so deployment costs nothing extra.

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^\top = (\mathbf{U}\mathbf{V}^\top)\,(\mathbf{V}\Sigma\mathbf{V}^\top) = \mathbf{Q}\,\mathbf{P}. \tag{33.59}$$

Here $\mathbf{Q} = \mathbf{U}\mathbf{V}^\top$ is orthogonal, as a product of orthogonal matrices, and $\mathbf{P} = \mathbf{V}\Sigma\mathbf{V}^\top$ is symmetric positive semidefinite, with the σ_i as its eigenvalues. Every square matrix is a symmetric stretch followed by an orthogonal map, the matrix version of writing a complex number as $z = e^{i\theta}|z|$. The orthogonal factor need not be a rotation: when $\mathbf{A}$ is invertible, $\det \mathbf{Q} = \operatorname{sign} \det \mathbf{A}$, so $\mathbf{Q}$ includes a reflection exactly when $\det \mathbf{A} < 0$. For the shear $\mathbf{T}$, Equation 33.46 gives $\mathbf{Q} = \mathbf{R}_{31.7^\circ}\mathbf{R}_{58.3^\circ}^\top = \mathbf{R}_{-26.6^\circ} = \frac{1}{\sqrt{5}}\left(\begin{smallmatrix} 2 & 1 \\ -1 & 2 \end{smallmatrix}\right)$ (note $\tan 26.6^\circ = \frac{1}{2}$) and $\mathbf{P} = \frac{1}{\sqrt{5}}\left(\begin{smallmatrix} 2 & 1 \\ 1 & 3 \end{smallmatrix}\right)$. Indeed $\mathbf{QP} = \frac{1}{5}\left(\begin{smallmatrix} 5 & 5 \\ 0 & 5 \end{smallmatrix}\right) = \mathbf{T}$, and $\mathbf{P}^2 = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 2 \end{smallmatrix}\right) = \mathbf{T}^\top\mathbf{T}$.

For a rectangular $\mathbf{M}$ with thin SVD $\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^\top$, the polar factor is $\mathbf{U}\mathbf{V}^\top$: it has the singular vectors of $\mathbf{M}$ and every singular value equal to 1, so its columns (if $m \geq n$) or rows (if $m \leq n$) are orthonormal. If $\mathbf{M}$ has rank $r < \min(m, n)$, the factor is not unique, and the natural choice is $\mathbf{U}_r\mathbf{V}_r^\top$ from the compact SVD. Exercise 8 shows that, for a square matrix, $\mathbf{U}\mathbf{V}^\top$ is an orthogonal matrix closest to it in the Frobenius norm.

Odd polynomials of a matrix. An SVD of every weight at every step would be too expensive, but matrix products can change the singular values without touching the singular vectors.

Proposition (odd polynomials act on singular values). *Let $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ be a thin SVD and $p(\sigma) = \sum_j c_j \sigma^{2j+1}$ an odd polynomial. Then*

$$p(\mathbf{X}) := \sum_j c_j\, \mathbf{X}(\mathbf{X}^\top\mathbf{X})^j = \mathbf{U}\, p(\Sigma)\, \mathbf{V}^\top. \tag{33.60}$$

Proof. $\mathbf{X}^\top\mathbf{X} = \mathbf{V}\Sigma^2\mathbf{V}^\top$, so $(\mathbf{X}^\top\mathbf{X})^j = \mathbf{V}\Sigma^{2j}\mathbf{V}^\top$ and $\mathbf{X}(\mathbf{X}^\top\mathbf{X})^j = \mathbf{U}\Sigma^{2j+1}\mathbf{V}^\top$; sum over j. ∎

A matrix polynomial of this form keeps the singular vectors and applies p to each singular value separately. To reach the polar factor we need a p whose repeated application sends every positive σ to 1.

Newton–Schulz. The iteration $\mathbf{X} \leftarrow \frac{1}{2}(3\mathbf{X} - \mathbf{X}\mathbf{X}^\top\mathbf{X})$ applies the cubic $p(\sigma) = \frac{1}{2}(3\sigma - \sigma^3)$ to every singular value. Its fixed points are 0 and ± 1. On $(0, 1]$ it increases every value without passing 1, since $p(\sigma) - \sigma = \frac{1}{2}\sigma(1 - \sigma^2) \geq 0$ and p rises to its maximum $p(1) = 1$. More generally, p maps the interval $(0, \sqrt{3})$ into $(0, 1]$, so every starting value in this interval, the *immediate basin* of 1, converges to 1. Larger values can end anywhere (Exercise 7). Starting from $\mathbf{X}_0 = \mathbf{M}/\|\mathbf{M}\|_F$ puts every singular value into $(0, 1]$, because $\|\mathbf{M}\|_F \geq \sigma_1(\mathbf{M})$. A zero singular value stays zero, since $p(0) = 0$, so a rank-deficient $\mathbf{M}$ yields $\mathbf{U}_r\mathbf{V}_r^\top$.

How many steps are needed? Near 0, $p(\sigma) \approx \frac{3}{2}\sigma$, so a singular value s needs about $\log_{3/2}(1/s)$ steps to reach order one. Near 1, $p(1 - e) = 1 - \frac{3}{2}e^2 + \frac{1}{2}e^3$, so the distance to 1 is roughly squared at each step, and from order one it reaches 10^{-6} in about four more steps. The count is therefore about $\log_{3/2}(1/s_{\min}) + 4$, set by the smallest normalized singular value $s_{\min} = \sigma_{\min}(\mathbf{M})/\|\mathbf{M}\|_F$. The estimate is rough, a linear phase glued to a quadratic one, so we test it on a random 6×4 matrix standing in for the momentum.

```
M = np.random.standard_normal((6, 4))              # a stand-in momentum matrix
X = M / np.linalg.norm(M)                        # Frobenius normalization: sigma in (0, 1]
s_min = np.linalg.svd(X, compute_uv=False)[-1]
```

```
for k in range(1, 41):
    X = 1.5 * X - 0.5 * X @ X.T @ X                  # p(sigma) = (3 sigma - sigma^3) / 2
    if np.abs(np.linalg.svd(X, compute_uv=False) - 1).max() < 1e-6:
        break
U, s, Vt = np.linalg.svd(M, full_matrices=False)
print(f'smallest scaled sigma {s_min:.3f};  predicted steps '
      f'{np.log(1 / s_min) / np.log(1.5) + 4:.1f}')
print(f'steps until max|sigma_i - 1| < 1e-6: {k};  distance to U V^T: '
      f'{np.linalg.norm(X - U @ Vt):.1e}')
```

```
smallest scaled sigma 0.146;  predicted steps 8.7
steps until max|sigma_i - 1| < 1e-6: 9;  distance to U V^T: 7.3e-11
```

The step count agrees with the prediction to within a step, and the iterate is within 10^{-5} of the polar factor computed by a full SVD. The count varies from draw to draw because $s_{\min}$ does; a nearly rank-deficient $\mathbf{M}$ needs more steps. Muon uses a tuned quintic in place of the cubic, trading exact convergence for speed (Section 9.7.2).

33.5.4.3 Measuring and Controlling the Spectrum

The last of the three questions asks how much a layer can stretch its input, and the largest singular value answers it. The whole spectrum of a weight matrix also says how close the matrix is to low rank. Training methods therefore bound the singular values, measure them, and compute them cheaply.

Spectral normalization. By Equation 33.47, a layer $\mathbf{x} \mapsto \mathbf{W}\mathbf{x}$ satisfies $\|\mathbf{W}\mathbf{x} - \mathbf{W}\mathbf{x}'\| \leq \sigma_1 \|\mathbf{x} - \mathbf{x}'\|$, with equality when $\mathbf{x} - \mathbf{x}'$ points along $\mathbf{v}_1$. The smallest factor that bounds the change of the output by the change of the input is the layer's *Lipschitz constant*, so here it is $\sigma_1 = \|\mathbf{W}\|_2$. Dividing $\mathbf{W}$ by σ_1 caps the stretch at 1, which stabilizes the training of GAN discriminators (Miyato et al. 2018). The value σ_1 is estimated by power iteration on $\mathbf{W}^\top\mathbf{W}$ (Section 33.4.1.2). In practice one alternates $\mathbf{u} \leftarrow \mathbf{W}\mathbf{v}$ and $\mathbf{v} \leftarrow \mathbf{W}^\top\mathbf{u}$, normalizing each; the two half-steps are one multiplication by $\mathbf{W}^\top\mathbf{W}$, and $\|\mathbf{W}\mathbf{v}\|$ is the estimate. Repeated multiplication turns $\mathbf{v}$ toward $\mathbf{v}_1$, with the error shrinking by about $(\sigma_2/\sigma_1)^2$ per step. Miyato et al. (2018) found empirically that one round per training step suffices when each round starts from the vector left by the previous one, and Exercise 9 tests this. For a convolution, σ_1 of the kernel reshaped into a matrix approximates the stretch of the convolution but is not equal to it.

Measuring the spectrum. The 0.39% of the LoRA example assumed that rank 8 suffices. Whether a matrix is close to low rank depends on its spectrum, so one measures before truncating. The cell does this for two matrices of the photograph's shape, 400×500: a Gaussian matrix, which has no structure at all, and the photograph itself with its mean pixel value subtracted, a matrix of real data.

```
G = np.random.standard_normal(img.shape)                    # no structure
for name, W in [('Gaussian', G), ('photograph', img - img.mean())]:
    s = np.linalg.svd(W, compute_uv=False)
    r95 = int(np.searchsorted(np.cumsum(s**2) / (s**2).sum(), 0.95)) + 1
    print(f'{name:10s}: sigma from {s[0]:.3g} to {s[-1]:.2g};  95% energy at rank {r95:3d};  '
          f'adapter {r95 * (m + n) / (m * n):.1%} of W')
```

```
Gaussian  : sigma from 42.2 to 2.5;  95% energy at rank 269;  adapter 121.0% of W
photograph: sigma from 2.65e+04 to 2.4;  95% energy at rank   8;  adapter 3.6% of W
```

The singular values of the Gaussian matrix fill the noise interval $[\sqrt{n}-\sqrt{m}, \sqrt{n}+\sqrt{m}] \approx [2.4, 42.4]$ of Section 33.5.2.1, with no gap. Keeping 95% of its energy takes rank about 270 of 400, and an adapter of that rank would have more parameters than the matrix itself. The centered photograph reaches 95% at rank 8, with an adapter of under 4% of the parameters. Neither matrix is a trained weight, and the pair shows only how far apart the answers can lie. For LoRA, the matrix in question is the update a task needs, and its spectrum has to be measured.

Randomized SVD. For matrices too large to factor in full, *randomized SVD* (Halko et al. 2011) computes an accurate rank-k truncation from a few products with $\mathbf{A}$. Its core is a *range finder*: multiply $\mathbf{A}$ by a Gaussian matrix Ω with slightly more than k columns, orthonormalize the columns of $\mathbf{A}\Omega$ into $\mathbf{Q}$, and factor only the small matrix $\mathbf{Q}^\top\mathbf{A}$. When the spectrum decays, the top singular directions dominate every random combination of the columns of $\mathbf{A}$, and $\mathbf{Q}$ captures them with high probability. Exercise 10 implements the method and its variant for slowly decaying spectra, and Section 33.6 places it among other sketching methods.

The SVD has three limits. A full SVD costs $O(mn\min(m,n))$ operations, of order $4096^3 \approx 7\times10^{10}$ for a single 4096×4096 weight, which is why the range finder and Newton–Schulz use only matrix products. Low rank cannot be assumed, as the Gaussian matrix showed. And the SVD finds only linear structure. Points on a circle in the plane form a data matrix of rank 2, yet the circle is one-dimensional, and describing such curved structure needs derivatives, the subject of the next chapter.

33.5.5 Summary

Every matrix, of any shape, has two orthonormal bases, $\mathbf{v}_i$ for the inputs and $\mathbf{u}_i$ for the outputs, between which it acts as a list of stretches, $\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$ with $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$. The factorization comes from the spectral theorem applied to $\mathbf{A}^\top\mathbf{A}$, with $\sigma_i = \sqrt{\lambda_i(\mathbf{A}^\top\mathbf{A})}$. From the list of σ_i one reads the rank, the norms $\|\mathbf{A}\|_2 = \sigma_1$ and $\|\mathbf{A}\|_F^2 = \sum_i \sigma_i^2$, the error of the best rank-k approximation, which is the truncation $\mathbf{A}_k$ (Eckart–Young; on centered data it is principal component analysis), and the condition number $\kappa = \sigma_1/\sigma_r$ that bounds how much a solve amplifies relative error. Inverting the nonzero σ_i gives the pseudoinverse and with it the shortest least-squares solution. In training, the same list sizes low-rank adapters, defines the polar factor that Muon steps along, and gives the Lipschitz constant that spectral normalization controls.

33.5.6 Exercises

1. **The SVD by hand.** Let $\mathbf{A} = \left(\begin{smallmatrix} 3 & 0 \\ 4 & 5 \end{smallmatrix}\right)$.
    1. Form $\mathbf{A}^\top\mathbf{A}$ and show that its eigenvalues are 45 and 5, so that $\sigma_1 = 3\sqrt{5}$ and $\sigma_2 = \sqrt{5}$.
    2. Find unit eigenvectors $\mathbf{v}_1, \mathbf{v}_2$ and compute $\mathbf{u}_i = \mathbf{A}\mathbf{v}_i/\sigma_i$. Check that $\mathbf{u}_1 \perp \mathbf{u}_2$ directly, without the proof of the text.
    3. Write the rank-one approximation $\mathbf{A}_1$ and compute $\|\mathbf{A}-\mathbf{A}_1\|_2$ and $\|\mathbf{A}-\mathbf{A}_1\|_F$. Why are the two equal here, and when are they not?
    4. Compute $\mathbf{A}^+$ from the SVD, check it against $\mathbf{A}^{-1}$, and compute $\kappa(\mathbf{A})$.

5. Let $\mathbf{A}$ be symmetric with eigendecomposition $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^\top$. Construct an SVD of $\mathbf{A}$ from $\mathbf{V}$ and Λ and conclude that its singular values are the $|\lambda_i|$. Apply the construction to $\mathrm{diag}(3, -2)$. Which factor absorbs the minus sign?
6. Exercise 2a of Section 33.4, with $k = 1$, gives $|\lambda| \le \sigma_1$ for every eigenvalue λ of a square matrix $\mathbf{A} \in \mathbb{R}^{n\times n}$. Show that $\sigma_n \le |\lambda|$ as well; for a complex eigenvector, write $\mathbf{x} = \mathbf{a} + i\,\mathbf{b}$ with real $\mathbf{a}$ and $\mathbf{b}$. Check both bounds on the chapter's running matrix $\left(\begin{smallmatrix} 1 & 2 \\ -1 & 3 \end{smallmatrix}\right)$.

2. **Norms and the maximum stretch.** The text obtained Equation 33.47 from the spectral theorem. The maximum stretch can also build the SVD on its own.

   1. Let $\sigma_1 = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|$, assume $\sigma_1 > 0$, and build an SVD without the spectral theorem. (i) Show that the maximum is attained at some unit $\mathbf{v}_1$, and set $\mathbf{u}_1 = \mathbf{A}\mathbf{v}_1/\sigma_1$. (ii) For a unit $\mathbf{w} \perp \mathbf{v}_1$, the function $t \mapsto \|\mathbf{A}(\mathbf{v}_1 + t\mathbf{w})\|^2/(1 + t^2)$ has a maximum at $t = 0$, so its derivative there vanishes. Deduce $\mathbf{u}_1^\top \mathbf{A}\mathbf{w} = 0$: $\mathbf{A}$ maps $\mathbf{v}_1^\perp$ into $\mathbf{u}_1^\perp$. (iii) Recurse on the restriction of $\mathbf{A}$ to $\mathbf{v}_1^\perp$. Where did compactness of the unit sphere enter?
   2. Show that $\|\mathbf{A}\|_2 \le \|\mathbf{A}\|_F \le \sqrt{r}\,\|\mathbf{A}\|_2$ with $r = \operatorname{rank}\mathbf{A}$. Which matrices attain equality on each side?
   3. The ratio $\|\mathbf{A}\|_F^2/\|\mathbf{A}\|_2^2$ is called the *stable rank* of $\mathbf{A}$; by (b) it lies between 1 and $\operatorname{rank}\mathbf{A}$. Compute it in closed form for the decaying spectrum $\sigma_i = e^{-i/12}$, $i = 0, \dots, 255$ (a geometric series), and show that rank 18 is needed to hold 95% of the energy of this spectrum. Then estimate the stable rank of the 400×500 Gaussian matrix of Section 33.5.4.3 from $\|\mathbf{G}\|_F^2 \approx mn$ and $\sigma_1 \approx \sqrt{m}+\sqrt{n}$, and compare it with the rank of about 270 that holds 95% of its energy. Why can the stable rank be far below the rank?

3. **Weyl perturbation bound.** An update Δ, for instance a rank-k LoRA update $\mathbf{B}\mathbf{A}$ (the paper's letters), changes the singular values of a weight matrix $\mathbf{W}$. The Weyl inequality Equation 33.53, proved in Section 33.5.2.1, limits how far they move.

   1. Set $j = 1$ in Equation 33.53 and apply the inequality in both directions to derive $|\sigma_i(\mathbf{W} + \Delta) - \sigma_i(\mathbf{W})| \le \sigma_1(\Delta)$ for every i.
   2. Take $\mathbf{W} = \mathrm{diag}(1, 2)$ and $\Delta = 0.1\,\mathbf{1}\mathbf{1}^\top$. Compute the singular values of $\mathbf{W} + \Delta$ and compare each change with $\sigma_1(\Delta)$.
   3. Now let $\operatorname{rank}\Delta \le k$. The Frobenius proof in Section 33.5.2.1 applies Equation 33.53 with $j = k + 1$ to a matrix of rank at most k. Apply the same step to Δ to obtain the interlacing inequalities $\sigma_{i+k}(\mathbf{W}) \le \sigma_i(\mathbf{W} + \Delta)$ and $\sigma_{i+k}(\mathbf{W} + \Delta) \le \sigma_i(\mathbf{W})$. Use them to show that the ranks of $\mathbf{W} + \Delta$ and $\mathbf{W}$ differ by at most k, and that at most k singular values of $\mathbf{W} + \Delta$ exceed $\sigma_1(\mathbf{W})$, however large $\sigma_1(\Delta)$ is. Why does the bound of
      (a) not give these conclusions?

4. code **Denoising by truncation.** Generate a rank-5 signal $\mathbf{B} = \mathbf{X}\mathbf{Y}^\top$ with $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{200\times5}$ standard Gaussian, and add noise $\mathbf{N}$ with i.i.d. $\mathcal{N}(0, 1)$ entries. This is the model $\mathbf{A} = \mathbf{B} + \mathbf{N}$ of Section 33.5.2.1 with $m = n = 200$ and $\sigma_{\text{noise}} = 1$.

   1. Plot the singular values of $\mathbf{A}$ on a log scale. Where does the noise bulk end, compared with the edge $(\sqrt{m} + \sqrt{n})\,\sigma_{\text{noise}}$, and how many singular values lie above it?
   2. Compute $\|\mathbf{A}_k - \mathbf{B}\|_F^2$ for $k = 0, \dots, 20$, where $\mathbf{A}_k$ is the rank-k truncation of $\mathbf{A}$. Which k minimizes the error, and how does the minimum compare with $\|\mathbf{N}\|_F^2$?

3. A rank-r matrix of size $m \times n$ has $r(m+n-r)$ degrees of freedom. Compare the minimal error with $\sigma_{\text{noise}}^2\, r(m+n-r)$ for $r = 5$. Why does only this part of the noise survive the truncation? By about how much does each component beyond the fifth increase the error, and which singular value of $\mathbf{A}$ predicts the increase?

5. code **Centering in PCA, and Fashion-MNIST.** Let the rows of $\mathbf{X} \in \mathbb{R}^{n\times d}$ be data points with mean row $\bar{\mathbf{x}}$ and covariance $\mathbf{C}$ as in the principal component analysis of Section 33.5.2.2.

   1. Show that $\frac{1}{n}\mathbf{X}^\top\mathbf{X} = \mathbf{C} + \bar{\mathbf{x}}\bar{\mathbf{x}}^\top$.
   2. Generate a cloud like the one of Section 33.5.2.2 (300 points with standard deviations 3 and 0.8 along axes rotated by $\pi/5$) and shift every point by $c\,(1,-1)/\sqrt{2}$ for $c \in \{0, 1, 3, 10, 30\}$. For each c, compute the top right singular vector of the *uncentered* data, its angle to the principal axis $(\cos\frac{\pi}{5}, \sin\frac{\pi}{5})$, and the ratio $\sigma_1^2/\sum_i \sigma_i^2$.
   3. Use (a) to predict, from the eigenvalues of $\mathbf{C}$, the offset at which the top singular vector turns from the principal axis toward the mean.
   4. Use (a) again to approximate the ratio $\sigma_1^2/\sum_i \sigma_i^2$ for large c, and find its limit as $c \to \infty$. At $c = 10$ the ratio is close to the centered explained-variance ratio. Does this agreement reflect the data or the chosen offset?
   5. Load the Fashion-MNIST test images of t-shirts and trousers through `d2l.FashionMNIST`, as in Section 33.1, flatten them to vectors of length 784, center them, and compute the top two right singular vectors. Plot the two classes in the plane of the first two principal components. What fraction of the variance do these two directions hold? Compute the angle between $\mathbf{v}_1$ and the difference of the class means $\bar{\mathbf{x}}_{\text{trouser}} - \bar{\mathbf{x}}_{\text{t-shirt}}$, and explain why the two directions are close but not equal. Finally, reconstruct one t-shirt from its first 5, 20 and 50 components and report the relative error of each.

6. code **Ridgeless limit and gradient descent.** An overparameterized linear model has more parameters than equations, so least squares has infinitely many minimizers. Part (a) holds for every shape and rank;

   (b) and (c) use $\mathbf{A} \in \mathbb{R}^{m\times n}$ with $m < n$.

   1. Use the filter form Equation 33.56 to show that $(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^\top \to \mathbf{A}^+$ as $\lambda \to 0^+$, for a matrix $\mathbf{A}$ of any shape and rank. Why does $(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$ not exist when $m < n$?
   2. Gradient descent on $\frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|^2$ with step size $0 < \eta < 2/\sigma_1^2$ updates $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\,\mathbf{A}^\top(\mathbf{A}\mathbf{x}_t - \mathbf{b})$. Show that $\mathbf{x}_t - \mathbf{x}_0$ stays in the row space of $\mathbf{A}$ and that $\mathbf{x}_t$ converges to $\mathbf{A}^+\mathbf{b} + (\mathbf{I} - \mathbf{A}^+\mathbf{A})\mathbf{x}_0$. Which solution does the initialization $\mathbf{x}_0 = \mathbf{0}$ select?
   3. Check (a) and (b) for a 20×50 Gaussian $\mathbf{A}$ and a Gaussian $\mathbf{b}$: run gradient descent from $\mathbf{x}_0 = \mathbf{0}$ and from a random $\mathbf{x}_0$, and compare both limits and their norms with $\mathbf{A}^+\mathbf{b}$.

7. code **Newton–Schulz basin.** The Newton–Schulz iteration $\mathbf{X} \leftarrow \frac{1}{2}(3\mathbf{X} - \mathbf{X}\mathbf{X}^\top\mathbf{X})$ of Section 33.5.4.2 starts from $\mathbf{X} = \mathbf{M}/\|\mathbf{M}\|_F$.

   1. Let $\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top$ be a thin SVD. Show directly, without Equation 33.60, that one step produces $\mathbf{U}\,p(\Sigma)\,\mathbf{V}^\top$ with $p(\sigma) = \frac{1}{2}(3\sigma - \sigma^3)$ applied to every diagonal entry.
   2. Show that $(0, \sqrt{3})$ is the immediate basin of 1: p maps it into $(0, 1]$, and the iterates $p(p(\cdots p(\sigma_0)))$ converge to 1 for every $\sigma_0 \in (0, \sqrt{3})$. Show that some start-

ing values beyond $\sqrt{3}$ also reach 1 (try 2.15) and others do not (try 2). Expand $p(1-e)$ in powers of e. How fast does the distance to 1 shrink once it is small?

3. For a 6×4 Gaussian $\mathbf{M}$, count the steps until $\|\mathbf{X} - \mathbf{U}\mathbf{V}^\top\|_F < 10^{-6}$ when the iteration starts from $\mathbf{M}/c$ with $c = \|\mathbf{M}\|_F$ and with $c = \sigma_1(\mathbf{M})$. Compare both counts with $\log_{3/2}(1/s_{\min}) + 4$, where $s_{\min}$ is the smallest singular value after scaling. Why does Muon divide by $\|\mathbf{M}\|_F$ although $\sigma_1(\mathbf{M})$ saves a step?
4. Start the iteration from $\mathbf{M}/c$ with $c = \sigma_1(\mathbf{M})/2$ and with $c = \sigma_1(\mathbf{M})/2.5$. What does it converge to in each case? Explain both outcomes with the polynomial p of (b).

8. code **Polar factor and Procrustes.** ⊠ The polar decomposition Equation 33.59 writes a square matrix as $\mathbf{A} = \mathbf{Q}\mathbf{P}$ with $\mathbf{Q}$ orthogonal and $\mathbf{P}$ symmetric positive semidefinite.

   1. Show that $\mathbf{P} = (\mathbf{A}^\top\mathbf{A})^{1/2}$ is determined by $\mathbf{A}$, and that $\mathbf{Q}$ is determined by $\mathbf{A}$ when $\mathbf{A}$ is invertible. Give two different polar decompositions of $\mathrm{diag}(1, 0)$.
   2. Show that $\mathbf{Q} = \mathbf{U}\mathbf{V}^\top$ is an orthogonal matrix closest to $\mathbf{A}$ in the Frobenius norm. Reduce the problem to maximizing $\mathrm{tr}(\mathbf{R}^\top\mathbf{A})$ over orthogonal $\mathbf{R}$.
   3. Two embedding spaces that differ by a rotation can be aligned from paired points $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{d\times N}$ by minimizing $\|\mathbf{R}\mathbf{X} - \mathbf{Y}\|_F$ over orthogonal $\mathbf{R}$. Use (b) to show that the minimizer is the polar factor of $\mathbf{Y}\mathbf{X}^\top$. Test it with $d = 50$, $N = 1000$, a random orthogonal $\mathbf{R}$ (the $\mathbf{Q}$ factor of a QR decomposition of a Gaussian matrix), Gaussian $\mathbf{X}$, and $\mathbf{Y} = \mathbf{R}\mathbf{X} + \mathbf{E}$, where $\mathbf{E}$ has i.i.d. $\mathcal{N}(0, s^2)$ entries for $s \in \{0, 0.1, 1, 3\}$. Compare $\|\hat{\mathbf{R}} - \mathbf{R}\|_F/\sqrt{d}$ for this estimate and for the unconstrained least-squares estimate $\mathbf{Y}\mathbf{X}^+$.

9. code **Warm-started power iteration.** ⊠ Spectral normalization divides a weight $\mathbf{W}$ by an estimate of $\sigma_1(\mathbf{W})$ at every training step. One round of the estimator updates $\mathbf{v} \leftarrow \mathbf{W}^\top\mathbf{u}/\|\mathbf{W}^\top\mathbf{u}\|$ and $\mathbf{u} \leftarrow \mathbf{W}\mathbf{v}/\|\mathbf{W}\mathbf{v}\|$ and reports $\hat{\sigma} = \mathbf{u}^\top\mathbf{W}\mathbf{v}$. Miyato et al. (2018) run one round per training step and start it from the $\mathbf{u}$ of the previous step.

   1. Show that one round performs one step of power iteration on $\mathbf{W}^\top\mathbf{W}$ for $\mathbf{v}$ and that $\hat{\sigma} = \|\mathbf{W}\mathbf{v}\|$. Why can $\hat{\sigma}$ never exceed σ_1? This is the one-round estimate whose adequacy Section 33.5.4.3 calls empirical; (c) tests it.
   2. For a 256×512 matrix with i.i.d. $\mathcal{N}(0, 1/512)$ entries, run 200 rounds from a random $\mathbf{u}$. Plot the relative error of $\hat{\sigma}$ and the squared sine of the angle between $\mathbf{v}$ and $\mathbf{v}_1$ against the round, and compute σ_2/σ_1. Since $\hat{\sigma}^2$ is the Rayleigh quotient of $\mathbf{W}^\top\mathbf{W}$ at $\mathbf{v}$, Exercise 3b of Section 33.4 bounds its error by the order of the squared sine. Why is the relative error of $\hat{\sigma}$ smaller than the squared sine by a further large factor?
   3. Simulate training: for 300 steps, add $\epsilon\,\mathbf{G}_k$ to $\mathbf{W}$, with a fresh $\mathbf{G}_k$ of i.i.d. $\mathcal{N}(0, 1/512)$ entries at each step and $\epsilon \in \{10^{-3}, 10^{-2}, 10^{-1}\}$, and run one round per step. Compare the relative error of $\hat{\sigma}$ when each round starts from the previous $\mathbf{u}$ and when it starts from a fresh random $\mathbf{u}$.

10. code **Randomized range finder.** ⊠ The randomized SVD of Section 33.5.4.3 multiplies $\mathbf{A}$ by a Gaussian $\Omega \in \mathbb{R}^{n\times(k+p)}$, orthonormalizes $\mathbf{Y} = \mathbf{A}\Omega$ into $\mathbf{Q}$, and truncates the SVD of $\mathbf{Q}^\top\mathbf{A}$ to rank k.

    1. Implement it. Build $\mathbf{A} \in \mathbb{R}^{1000\times500}$ as $\mathbf{U}\,\mathrm{diag}(\sigma_1, \dots, \sigma_{500})\mathbf{V}^\top$ with random orthonormal $\mathbf{U}$ and $\mathbf{V}$, once with the fast decay $\sigma_i = e^{-i/5}$ and once with the slow decay $\sigma_i = 1/i$. For $k = 20$ and $p \in \{0, 5, 10\}$, divide the spectral-norm error of the rank-k result by σ_{k+1}, the minimum of Equation 33.52, and average over

10 draws of Ω. Why does oversampling alone suffice for one spectrum but not for the other?

2. Replace $\mathbf{Y}$ by $(\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}\Omega$ for $q \in \{1, 2\}$. How do the ratios change for each spectrum? Explain the effect of q with the singular values of $(\mathbf{A}\mathbf{A}^\top)^q\mathbf{A}$. How many products with $\mathbf{A}$ or $\mathbf{A}^\top$ does each variant cost?

11. code **Optimal hard threshold.** ▣▣ The threshold $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}}$ of Gavish and Donoho (2014) follows from the behavior of a single signal component. Let $\mathbf{A} = \theta\,\mathbf{u}\mathbf{v}^\top + \mathbf{N}$ be $n \times n$, with unit vectors $\mathbf{u}, \mathbf{v}$, $\theta > 0$, and $\mathbf{N}$ with i.i.d. Gaussian entries $\mathcal{N}(0, \sigma_{\text{noise}}^2)$. Write $x = \theta/(\sqrt{n}\,\sigma_{\text{noise}})$ and let $(\sigma_1, \hat{\mathbf{u}}, \hat{\mathbf{v}})$ be the top singular triple of $\mathbf{A}$, with the sign of the pair chosen so that $\mathbf{u}^\top\hat{\mathbf{u}} \geq 0$. As $n \to \infty$, Benaych-Georges and Nadakuditi (2012) show that σ_1 tends to the bulk edge $2\sqrt{n}\,\sigma_{\text{noise}}$ and the singular vectors become orthogonal to $\mathbf{u}$ and $\mathbf{v}$ when $x \leq 1$, and that for $x > 1$

$$\frac{\sigma_1}{\sqrt{n}\,\sigma_{\text{noise}}} \to x + \frac{1}{x}, \qquad (\mathbf{u}^\top\hat{\mathbf{u}})^2 \to 1 - \frac{1}{x^2}, \qquad (\mathbf{v}^\top\hat{\mathbf{v}})^2 \to 1 - \frac{1}{x^2}.$$

The transition at $x = 1$ is the singular-value counterpart of the eigenvalue transition at $\beta = \sqrt{c}$ in exercise 5 of Section 33.3.

1. Two estimates of the signal $\theta\,\mathbf{u}\mathbf{v}^\top$ are the rank-one truncation $\sigma_1\hat{\mathbf{u}}\hat{\mathbf{v}}^\top$ and the zero matrix. Express the squared Frobenius error of each in terms of θ, σ_1, $\mathbf{u}^\top\hat{\mathbf{u}}$ and $\mathbf{v}^\top\hat{\mathbf{v}}$.
2. Use the limits to show that the truncation has the smaller asymptotic error exactly when $x > \sqrt{3}$, and that the singular value at $x = \sqrt{3}$ is $\frac{4}{\sqrt{3}}\sqrt{n}\,\sigma_{\text{noise}}$. Which signal strengths produce a singular value above the noise edge that should nevertheless be discarded?
3. Check (a) and (b) for $n = 500$ and $\sigma_{\text{noise}} = 1/\sqrt{n}$, with $x \in \{0.5, 1.2, 1.5, \sqrt{3}, 2, 3\}$ and 10 draws each: compare σ_1, the squared cosines, and both errors with their limits. How do the results bear on the choice of k in exercise 4?

12. [extended] **Matrix completion by soft-thresholding.** ▣▣ Matrix completion recovers a low-rank matrix $\mathbf{B} \in \mathbb{R}^{n\times n}$ from its entries in an observed index set Ω. Recommender systems face this problem with a sparsely observed ratings matrix. Instead of the rank, which is not convex, one penalizes the *nuclear norm* $\|\mathbf{Z}\|_* = \sum_i \sigma_i(\mathbf{Z})$, the convex stand-in for rank (Candès and Recht 2009), much as the ℓ_1 norm stands in for sparsity. Soft-Impute (Mazumder et al. 2010) minimizes $\frac{1}{2}\sum_{(i,j)\in\Omega}(Z_{ij} - B_{ij})^2 + \lambda\|\mathbf{Z}\|_*$.

1. Let $\mathbf{Y} = \mathbf{U}\Sigma\mathbf{V}^\top$. Show that $\frac{1}{2}\|\mathbf{Z} - \mathbf{Y}\|_F^2 + \lambda\|\mathbf{Z}\|_*$ is minimized by $S_\lambda(\mathbf{Y}) = \mathbf{U}\max(\Sigma - \lambda\mathbf{I}, \mathbf{0})\mathbf{V}^\top$, with the maximum taken entrywise on the diagonal. Use von Neumann's trace inequality $\operatorname{tr}(\mathbf{Z}^\top\mathbf{Y}) \leq \sum_i \sigma_i(\mathbf{Z})\sigma_i(\mathbf{Y})$ (Horn and Johnson 2012).
2. Let P_Ω keep the observed entries of a matrix and set the others to zero, and let $P_\Omega^\perp$ do the opposite. Soft-Impute iterates $\mathbf{Z} \leftarrow S_\lambda(P_\Omega(\mathbf{B}) + P_\Omega^\perp(\mathbf{Z}))$. Implement it for a decreasing sequence of λ that ends near $10^{-4}\,\sigma_1(P_\Omega(\mathbf{B}))$, starting each λ from the solution for the previous one.
3. Take $n = 100$, $\mathbf{B} = \mathbf{X}\mathbf{Y}^\top$ with Gaussian $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n\times r}$ for $r \in \{2, 5, 10\}$, and observe each entry independently with probability $p \in \{0.1, 0.2, 0.3, 0.4, 0.6\}$.

The deliverable is a table of the relative error on the unobserved entries for every (r, p). Compare the boundary between failure and recovery with the $r(2n - r)$ degrees of freedom of a rank-r matrix. Why can no method recover $\mathbf{B}$ from fewer observations than that?

33.6 Further Reading

The chapter stops where linear algebra turns into numerical analysis, probability, or current research. This section says what lies past that point, in the order of the chapter's sections; where to learn the chapter's own material is the business of the list in its introduction (Chapter 33). The entries are not of equal weight. For deep learning, read random projections, iterative solvers, random matrices and trainable depth, and sketching first, then the last two entries, which lead back into the main text. Tensor decompositions, the QR algorithm, the Kreiss theorem, and mixing times are for the curious. Nothing else in the book depends on this section.

33.6.1 Geometry and Linear Maps

33.6.1.1 Other Inner Products: Kernel Methods

The nearest-mean rule of Section 33.1.4.2 compares an image with two class means through one dot product, so its decision boundary is a hyperplane. It fails when a class is spread out or curved, as on t-shirts against shirts. Kernel methods keep the rule and change the inner product: $\mathbf{x} \cdot \mathbf{x}_i$ is replaced by a function $k(\mathbf{x}, \mathbf{x}_i)$ that is an inner product in some other, possibly infinite-dimensional, space. The class mean becomes a mean in that space, every training point votes with weight $k(\mathbf{x}, \mathbf{x}_i)$, and the boundary can bend. Choosing which training points to keep, and with what weights, leads to the support vector machine. Schölkopf and Smola (2002) start their book from exactly this classifier and develop the theory of kernels from it.

33.6.1.2 Random Projections

Section 33.1 showed that random directions in $\mathbb{R}^d$ are nearly orthogonal. Can this shrink the dimension of a dataset without changing its geometry? The Johnson–Lindenstrauss lemma (Johnson and Lindenstrauss 1984) says yes: for n points, a scaled random Gaussian map to k of order $\varepsilon^{-2} \log n$ dimensions preserves all $\binom{n}{2}$ pairwise distances within a factor $1 \pm \varepsilon$, with high probability. Unlike principal component analysis (Section 33.5.2.2), the map does not look at the data, and the target dimension does not depend on d, so the lemma pays off when d is far larger than $\varepsilon^{-2} \log n$. Dasgupta and Gupta (2003) give a short proof with explicit constants, and Vershynin (2018) places the lemma among the concentration results for random vectors.

33.6.1.3 Solving Linear Systems at Scale

Section 33.2.2.4 solved systems by an LU factorization and warned that the inverse of a large sparse matrix is dense. Trefethen and Bau (1997) explain why elimination with partial pivoting is stable in practice although it is not in the worst case, and Higham (2002) gives the full rounding-error analysis. Solvers for sparse matrices never form the inverse, and often not the LU factors either. Sparse direct methods reorder the unknowns so that elimination creates few new nonzeros. Iterative methods touch the matrix only through products $\mathbf{Ax}$, as

power iteration does (Section 33.4.1.2). For a symmetric positive definite matrix, the conjugate gradient method of Hestenes and Stiefel (1952) shrinks the error after k steps by at least the factor $2\big((\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)\big)^{k}$ in the norm defined by the matrix, where κ is the condition number of Section 33.5.3.2. Golub and Van Loan (1996) treat the iterative methods in detail.

33.6.1.4 Contraction Order and Tensor Decompositions

The `einsum` of Section 33.2.1.4 fixes the result of a product of several factors but not the order of the pairwise contractions. For $n \times n$ matrices, $(\mathbf{AB})\mathbf{x}$ costs about n^3 multiplications and $\mathbf{A}(\mathbf{Bx})$ costs $2n^2$. Finding the best order for many factors is a search over parenthesizations, which `np.einsum(..., optimize=True)` and the opt_einsum package (Smith and Gray 2018) carry out. Low rank also extends beyond matrices. A three-way tensor can be written as a sum of outer products $\mathbf{a}_i \otimes \mathbf{b}_i \otimes \mathbf{c}_i$ (the CP decomposition), as a small core tensor multiplied by a matrix along each mode (Tucker), or as a chain of three-way cores (the tensor train of Oseledets (2011)). Eckart–Young has no analogue for tensors: computing the CP rank is NP-hard, and a best approximation of a given CP rank need not exist. The survey of Kolda and Bader (2009) covers the CP and Tucker decompositions and their algorithms.

33.6.2 Eigenvalues and Spectra

33.6.2.1 How Libraries Compute Eigenvalues

No formula in radicals gives the roots of a general polynomial of degree five or more, so every eigensolver iterates, and none forms the characteristic polynomial of Section 33.3.1.2, whose roots are highly sensitive to its coefficients. `eig` first reduces the matrix by orthogonal similarity transformations to Hessenberg form, zero below the first subdiagonal, at a cost of $O(n^3)$. The shifted QR algorithm then factors $\mathbf{H} - \mu\mathbf{I} = \mathbf{QR}$, forms $\mathbf{RQ} + \mu\mathbf{I}$, and repeats. Each step is a similarity transformation, so the eigenvalues stay the same, while the subdiagonal entries go to zero and the diagonal converges to the spectrum (a complex pair of a real matrix remains as a 2×2 block). For a symmetric matrix the Hessenberg form is tridiagonal, which `eigh` exploits. Trefethen and Bau (1997) derive the algorithm from power iteration applied to whole subspaces, and Golub and Van Loan (1996) give the production details.

33.6.2.2 Pseudospectra and the Kreiss Matrix Theorem

Section 33.4.2.4 showed that the spectrum of a non-normal matrix does not predict its transient growth. The Kreiss matrix theorem makes the pseudospectra quantitative. With the Kreiss constant $K(\mathbf{A}) = \sup_{|z|>1} (|z|-1)\|(z\mathbf{I}-\mathbf{A})^{-1}\|_2$, it brackets the largest power, $K(\mathbf{A}) \le \sup_k \|\mathbf{A}^k\|_2 \le e\,n\,K(\mathbf{A})$ for an $n \times n$ matrix. The lower bound is an exercise of Section 33.4; the upper bound says that a matrix whose pseudospectra stay close to the unit disc cannot grow much. Trefethen and Embree (2005) develop the subject and its applications, among them the transition to turbulence in fluid flows whose linearization has only stable eigenvalues.

33.6.2.3 Mixing Times and the Spectral Gap

The error of the two-page chain of Section 33.3 shrinks by a factor of 4 per click, because its second eigenvalue is $1/4$. How long does a general chain take? For a reversible chain, let $\lambda_\star$

be the largest modulus among the eigenvalues other than 1. The number of steps needed to come within total variation distance ε of the stationary distribution is bounded above and below by multiples of $1/(1-\lambda_\star)$, the multiples depending on ε and on the smallest stationary probability. For a lazy chain, one that stays put with probability at least $1/2$, no eigenvalue is negative and $\lambda_\star$ is the second eigenvalue λ_2. Cheeger's inequality bounds the gap $1-\lambda_2$ from both sides by the chain's *conductance*, the smallest probability flow out of a set relative to the set's mass, so the gap can be read from the shape of the graph. Levin and Peres (2017) develop this theory, including the convergence facts that Section 33.4.1.3 cites from Horn and Johnson (2012) .

33.6.2.4 Random Matrices and Trainable Depth

Section 33.4.2.5 showed that the spectral radii of the factors do not control a product of Jacobians. At initialization, what does? Saxe et al. (2014) solve the learning dynamics of deep linear networks exactly and find that orthogonal initial weights make the learning time independent of depth, where scaled Gaussian weights do not. Pennington et al. (2017) use free probability to compute the distribution of all singular values of the input–output Jacobian of a deep nonlinear network. *Dynamical isometry*, all these singular values near 1, is reachable with orthogonal weights and tanh units but not with ReLU, and networks that have it train faster at large depth.

33.6.2.5 Spectra of Trained Weights

The Marchenko–Pastur law of Section 33.3.2.4 describes the spectrum of a matrix of independent noise, and the weights of an untrained network with independent random initialization follow it. Martin and Mahoney (2021) compare the spectra of trained weight matrices with this null model and find that training produces outliers and a heavy, power-law tail. They read the exponent of that tail as a measure of implicit self-regularization. The departure from Marchenko–Pastur is observed across many models; using the exponent to rank models by quality without test data rests on an empirical correlation, not a theorem.

33.6.3 Low Rank Beyond the Truncated SVD

33.6.3.1 Matrix Completion and the Nuclear Norm

A ratings matrix is observed in a few percent of its entries, and Eckart–Young (Section 33.5.2.1) needs the whole matrix. Matrix completion minimizes the *nuclear norm* $\|\mathbf{Z}\|_* = \sum_i \sigma_i$, the convex stand-in for rank in the way that ℓ_1 stands in for sparsity, subject to agreeing with the observed entries. Candès and Recht (2009) prove that this convex program recovers a low-rank matrix exactly, with high probability, from far fewer than all n^2 entries chosen at random, provided its singular vectors are spread out over the coordinates. Soft-Impute (Mazumder et al. 2010), the algorithm of the matrix-completion exercise of Section 33.5, solves the penalized form by alternating a fill-in step with soft-thresholding of the singular values.

33.6.3.2 CUR Decompositions and Sketching

The factors of the truncated SVD mix all rows and columns, so they are hard to interpret, and a full SVD of an $m \times n$ matrix costs $O(mn\min(m,n))$. A CUR decomposition keeps actual columns $\mathbf{C}$ and rows $\mathbf{R}$ of the data and fits a small core $\mathbf{U}$ between them. Section 33.5.2.1 showed the naive choice, the columns of largest norm, losing to the truncation by a wide

margin. Mahoney and Drineas (2009) sample columns and rows by their *leverage scores*, the squared lengths of their components along the top singular vectors, and come within a small constant factor of the Eckart–Young error with high probability. The range finder of Section 33.5.4.3 (Halko et al. 2011) is one instance of *sketching*: multiply by a random matrix and work with the small result. Woodruff (2014) applies the same idea to least squares and regression, which are solved to within a factor $1 + \varepsilon$ on a sketch with far fewer rows than the data.

33.6.3.3 Adapter Variants

LoRA (Section 33.5.4.1) starts its update $\mathbf{BA}$ with $\mathbf{B} = \mathbf{0}$ and a random $\mathbf{A}$, so the first updates see the input only through a random k-dimensional subspace, unrelated to the pretrained weight. In PiSSA (Meng et al. 2024), $\mathbf{BA}$ starts as the rank-k truncated SVD of the pretrained weight and the residual is frozen, so training adjusts the principal directions from the first step. DoRA (S.-Y. Liu et al. 2024) writes each column of the weight as a length times a unit direction, trains the lengths directly, and applies the low-rank update to the directions only.

33.6.3.4 Low Rank in Attention

Attention over n tokens forms an $n \times n$ matrix, and several methods taught in the attention chapters replace it, or the cache behind it, by a low-rank factorization. Linear attention (Section 10.5) writes the scores as $\phi(\mathbf{q}) \cdot \phi(\mathbf{k})$, so the matrix has rank at most the feature dimension, and multi-head latent attention (Section 11.3) (DeepSeek-AI 2024) caches one low-dimensional latent per token. Linformer (Wang et al. 2020) starts from the observation that trained attention matrices have rapidly decaying singular values and projects the keys and values along the sequence axis to a fixed length.

33.6.3.5 Steepest Descent under Other Norms

Section 33.5.4.2 computed the polar factor $\mathbf{UV}^\top$ of a matrix. It is the steepest-descent direction when steps are measured in the spectral norm, and Bernstein and Newhouse (2024) turn this into a way of classifying optimizers. With their moving averages switched off, Adam becomes sign descent, which is steepest descent under the ℓ_∞ norm, and Shampoo becomes steepest descent under the spectral norm. The norm, chosen per layer, is then the design decision, and the update rule follows from it. The optimizer built on the spectral norm is taught in Section 9.7.

Chapter 34

Calculus and Automatic Differentiation

Almost every model in this book was trained by the same loop: compute a loss, ask the framework for its gradient, and move every weight a small step against it. Two parts of that loop went unexplained. Suppose a run diverges after we double the learning rate. The gradient is still correct, but the step has left the region where the slope predicts the loss. How far can that prediction be trusted? The gradient also comes cheaply. Nudging a hundred million weights one at a time would take a hundred million network runs, yet one call returns every slope for the cost of a few. How?

Calculus answers these questions, although it was created for others. In the seventeenth century Newton and Leibniz asked how to describe a quantity that changes continuously, and how to add up infinitely many small pieces. Their answers, the derivative and the integral, undo each other. The derivative is the best linear model of a function near a point. Besides the loss value, it is all an optimizer knows. The integral enters once data or weights are random: a probability or an expected loss is an area under a curve, and a policy-gradient method (Section 14.6) differentiates one with respect to the distribution it samples from.

The Basics part stated the rules of differentiation (Section 1.4) and used automatic differentiation as a tool (Section 1.5); the main text needs no more. This chapter is additional reading, for the reader who wants the proofs. Three results matter most: the descent lemma, which turns a bound on the curvature into the safe step size that Section 9.2 quotes; the chain rule as a product of Jacobians, cheapest when multiplied from the loss backward, which is backpropagation (Section 4.3); and the two estimators for the gradient of an expectation, neither of which always wins.

The chapter has three movements. The first builds the linear and the quadratic model of a function, in one variable (Section 34.1) and in many (Section 34.2); these models bound a safe step and tell a minimum from a saddle. The second follows the chain rule from a sum over the paths of a graph (Section 34.3) to a product of layer Jacobians (Section 34.4) and to a program that multiplies them (Section 34.5). The third turns to the integral (Section 34.6) and ends where sampling meets differentiation, with the gradient of an expectation (Section 34.7).

The school calculus that opens several sections can be skimmed. Vector calculus beyond

the gradient (divergence, Stokes' theorem) and measure theory are left out; for differential equations see Chapter 38.

Resources and Further Reading

The resources below accompany the sections of this chapter; each names the section it goes with. Topics that the chapter does not teach, such as automatic differentiation at nonsmooth points, recursive checkpointing, and the Lebesgue integral, are collected in Section 34.8.

Books

- Mathematics for Machine Learning — Deisenroth, Faisal & Ong — free Cambridge text; its Chapter 5, "Vector Calculus," covers gradients, Jacobians, and the chain rule for machine learning, the material of Section 34.1 and Section 34.2.
- Calculus — Gilbert Strang, MIT OpenCourseWare — a free textbook (3rd edition) whose chapters on integrals, the fundamental theorem, and multiple integrals accompany Section 34.6.
- The Matrix Cookbook — Petersen & Pedersen — a reference table of matrix-derivative identities, useful for checking the results that Section 34.4 derives by the differential method.
- Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation — Griewank & Walther — the standard monograph (SIAM, 2nd ed.) on forward and reverse mode and checkpointing, the subject of Section 34.5.
- The Art of Differentiating Computer Programs — Naumann — an introduction to algorithmic differentiation from the compiler's point of view, with tangent and adjoint code written out by hand; it accompanies Section 34.5.

Courses and video lectures

- Essence of Calculus — 3Blue1Brown — a twelve-part visual introduction to derivatives, integrals, and the chain rule, for Section 34.1 and Section 34.6.
- Single Variable Calculus (18.01SC) — MIT OpenCourseWare — a self-study course with lecture videos and graded problem sets for Section 34.1.
- Multivariable Calculus (18.02SC) — MIT OpenCourseWare — partial derivatives, gradients, Lagrange multipliers, and double integrals, for Section 34.2.
- Matrix Calculus for Machine Learning and Beyond (18.S096) — Edelman & Johnson, MIT OCW — derivatives of matrix-valued maps treated as linear maps rather than entry by entry, the view of Section 34.4; lecture notes and videos included.

Tutorials and notes

- The Matrix Calculus You Need For Deep Learning — Parr & Howard — a free primer on the index method for the Jacobians of Section 34.4, assuming only Calculus 1.
- Calculus on Computational Graphs: Backpropagation — Christopher Olah — a short, diagram-driven post on the chain rule as a sum over paths, the backward sweep of Section 34.3.
- matrixcalculus.org — Laue, Mitterreiter & Giesen — an online symbolic calculator for vector and matrix derivatives, for checking the hand derivations of Section 34.4.

Automatic differentiation and gradient estimation

- Automatic Differentiation in Machine Learning: a Survey — Baydin, Pearlmutter, Radul & Siskind — the standard survey (JMLR, 2018) of the modes and their imple-

mentations, and of why automatic differentiation is neither symbolic nor numerical differentiation; it accompanies Section 34.5.

- The Autodiff Cookbook — JAX documentation — `grad`, JVPs, VJPs, Jacobians, and Hessian–vector products with runnable examples, the compositions of Section 34.5.
- Autodidax: JAX core from scratch — JAX documentation — builds forward- and reverse-mode autodiff (and `jit`/`vmap`) from basic components, a larger version of the engines built in Section 34.5.
- A Gentle Introduction to torch.autograd — PyTorch documentation — how PyTorch records a dynamic graph and replays it in reverse, the tape of Section 34.5.
- Monte Carlo Gradient Estimation in Machine Learning — Mohamed, Rosca, Figurnov & Mnih — a survey (JMLR, 2020) of the score-function and pathwise estimators of Section 34.7, with their variances and the conditions under which each applies.

34.1 Single Variable Calculus

The models of the Basics part were all trained by gradient descent. At each step the optimizer evaluates the loss and its slope at the current weights, moves the weights a little downhill, and repeats. It never sees the loss anywhere else, so every step rests on a prediction: the slope at one point is assumed to describe the loss a short distance away. How good is that prediction, and how large a step does it allow?

A single weight already shows what the answer looks like. Take one training example, one weight w, and the logistic loss $\ell(w) = \log(1 + e^{-w})$. At $w = 0$ the loss is 0.693147 and its slope is $-\frac{1}{2}$. If the weight moves to $w = 0.1$, a straight line through $w = 0$ with that slope predicts a loss of $0.693147 - 0.05 = 0.643147$. The actual loss is 0.644397, so the line is off by only 0.00125. The error has a simple form. To the digits shown it equals $\frac{1}{2} \cdot \frac{1}{4} \cdot 0.1^2$, where the number $\frac{1}{4}$ measures how much the graph of ℓ bends at $w = 0$. The slope makes the prediction, and the bend says how quickly the prediction goes wrong as the step grows. Both constants are derived once the rules of differentiation are available.

This section builds the line from the first derivative and, from the second, a parabola that also accounts for the bend. Three results carry it: the Mean Value Theorem, the Lagrange remainder, which bounds the error of the line and of the parabola, and the descent lemma, which turns a bound on the bend into a safe step size. Newton's method and the corner of ReLU, where no line fits and backpropagation must still return a slope, are met here in one variable; their theory belongs to Chapter 35.

The Basics (Section 1.4) defined the derivative, stated the rules of differentiation, and used automatic differentiation as a tool. We recall that material briefly and spend the words on the proofs and the error bounds. A reader who trusts the rules can start at Section 34.1.3. One variable is the place to begin because every loss contains it: holding all weights of $\ell(w_1, \dots, w_n)$ fixed except w_i gives a function of one variable, and one-variable calculus studies such slices.

Our running function is the *softplus* $s(x) = \log(1 + e^x)$. It is the one-variable slice $\log(e^0 + e^x)$ of the log-sum-exp function that this chapter uses throughout, and the logistic loss above is $\ell(w) = s(-w)$.

34.1.1 The Derivative

34.1.1.1 The Difference Quotient

Zoom in far enough on the graph of f near a point x, and the curve cannot be told apart from a straight line (Figure 34.1). The slope of that line is the derivative.

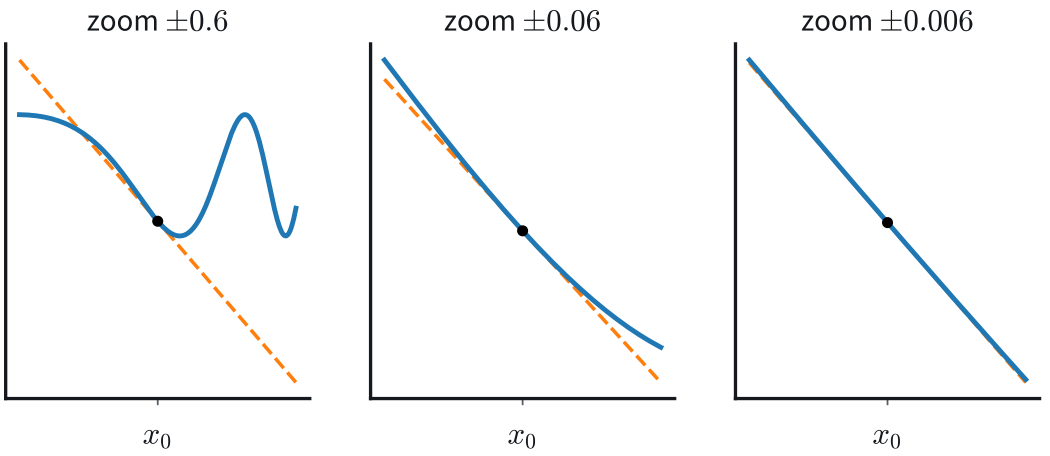


Figure 34.1: The curve $\sin(x^x)$ near $x_0 = 2$, in windows of half-width 0.6, 0.06 and 0.006. As the window shrinks, the curve (solid) becomes indistinguishable from its tangent line (dashed).

As in Section 1.4, we reach it through the *secant* line through $(x, f(x))$ and $(x+\epsilon, f(x+\epsilon))$, whose slope is the *difference quotient* $\big(f(x+\epsilon) - f(x)\big)/\epsilon$. As ϵ shrinks, the secant rotates toward the tangent (Figure 34.2).

How small must ϵ be before the secant slope can be trusted? That depends on the function. We choose a polynomial with a large cubic coefficient, $f(x) = x^2 + 1701(x-4)^3$, whose slope at $x = 4$ is 8, and print the secant slope there for shrinking ϵ.

```
def f(x):
    return x**2 + 1701*(x-4)**3

# Slope of the secant through x=4, for shrinking epsilon
for epsilon in [0.1, 0.001, 0.0001, 0.00001]:
    print(f'epsilon = {epsilon:.5f} -> {(f(4+epsilon) - f(4)) / epsilon:.5f}')
```

```
epsilon = 0.10000 -> 25.11000
epsilon = 0.00100 -> 8.00270
epsilon = 0.00010 -> 8.00012
epsilon = 0.00001 -> 8.00001
```

At $\epsilon = 0.1$ the secant slope is 25.11, three times the limit, and only from $\epsilon = 0.001$ on do the printed slopes settle near 8. Expanding $f(4+\epsilon) - f(4) = 8\epsilon + \epsilon^2 + 1701\epsilon^3$ shows why: the quotient is $8 + \epsilon + 1701\epsilon^2$, and the cubic term contributes 17 at $\epsilon = 0.1$ but only 1.7×10^{-7} at $\epsilon = 10^{-5}$. The large coefficient slows the approach to the limit without changing it. Automatic differentiation never meets this question, because it uses no ϵ: it

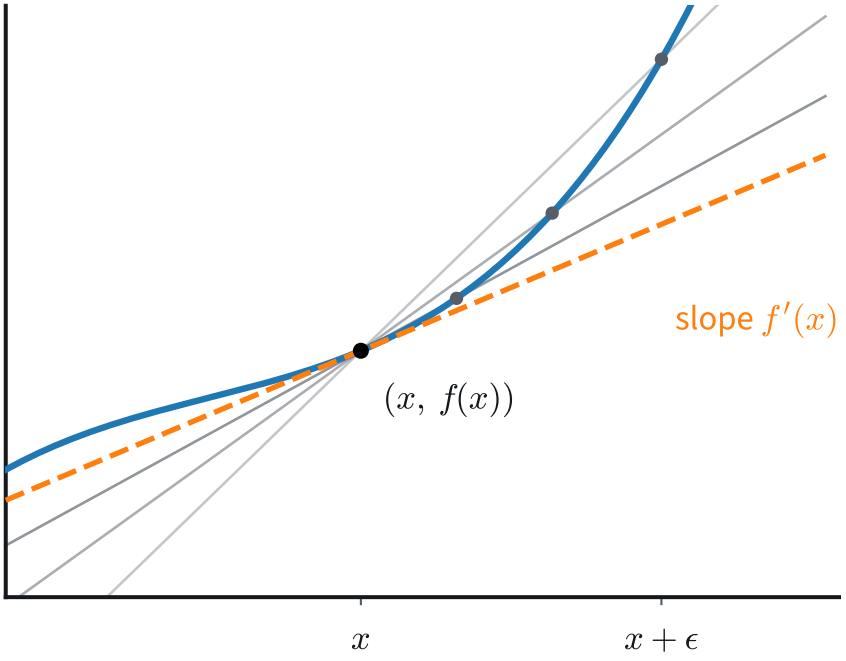


Figure 34.2: The difference quotient is the slope of the secant through $(x, f(x))$ and $(x + \epsilon, f(x + \epsilon))$. As $\epsilon \to 0$, this slope approaches the tangent slope $f'(x)$.

applies the rules of differentiation exactly (Section 34.5).

34.1.1.2 The Derivative and the Linear Approximation

The **derivative** of f at x is the limit of the secant slopes,

$$f'(x) = \frac{df}{dx}(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon}, \tag{34.1}$$

whenever the limit exists, and f is then called *differentiable* at x. For the cubic, $f'(4) = 8$. We write f' or $\frac{df}{dx}$, and $\frac{d}{dx}[\ldots]$ when the function is a bulky expression. The limit can fail to exist, as it does for $|x|$ at 0; Section 34.1.6 is about such corners.

It is instructive to rearrange the definition. Let $r(\epsilon) = f(x + \epsilon) - f(x) - \epsilon f'(x)$ be the error of the line; the definition says that $r(\epsilon)/\epsilon \to 0$. We write $o(\epsilon)$ for any quantity $r(\epsilon)$ with $r(\epsilon)/\epsilon \to 0$ as $\epsilon \to 0$, and obtain the **linear approximation**

$$f(x + \epsilon) = f(x) + \epsilon f'(x) + o(\epsilon). \tag{34.2}$$

A step ϵ changes the output by $\epsilon f'(x)$, up to an error that vanishes faster than ϵ. The converse holds as well: if $f(x + \epsilon) = f(x) + a\epsilon + o(\epsilon)$ for some number a, dividing by ϵ and letting $\epsilon \to 0$ shows that f is differentiable at x with $f'(x) = a$. The expansion and the derivative therefore carry the same information, and the proofs of the sum and chain rules use this direction.

Plotted against the input $x+\epsilon$, the line $f(x)+\epsilon f'(x)$ is the *tangent*, the limit of the rotating secants. It is the line that predicted the logistic loss at $w = 0.1$, and the gap of 0.00125 was its $o(\epsilon)$ term. Note that $o(\epsilon)$ promises little: such a term might shrink only slightly faster than ϵ. This one is of order ϵ^2, and Section 34.1.5.2 shows that this is the rule for twice differentiable functions.

34.1.2 Computing Derivatives

In practice nobody takes limits. We combine a short table of derivatives with rules for sums, products, and compositions, as Section 1.4 did. Here each rule is proved from the linear approximation. Of the table we derive the exponential, the logarithm and the power rule; the sine and cosine entries, and the existence of a base whose exponential has slope 1 at 0, are taken from a first calculus course.

34.1.2.1 A Table of Common Derivatives

The following entries hold wherever the right-hand sides are defined: the power rule for every real n when $x > 0$, and for integer n wherever x^n is defined; the logarithm for $x > 0$.

- **Derivative of constants.** $\frac{d}{dx} c = 0$.
- **Derivative of linear functions.** $\frac{d}{dx}(ax) = a$.
- **Power rule.** $\frac{d}{dx} x^n = n x^{n-1}$.
- **Derivative of exponentials.** $\frac{d}{dx} e^x = e^x$.
- **Derivative of the logarithm.** $\frac{d}{dx} \log(x) = \frac{1}{x}$.
- **Derivative of sine.** $\frac{d}{dx} \sin(x) = \cos(x)$.
- **Derivative of cosine.** $\frac{d}{dx} \cos(x) = -\sin(x)$.

The exponential entry rests on a one-line argument that also says what makes the base e special. For $a > 0$ the difference quotient of a^x factors as

$$\frac{a^{x+\epsilon} - a^x}{\epsilon} = a^x \, \frac{a^\epsilon - 1}{\epsilon},$$

and the second factor is the difference quotient at 0. So if a^x has a derivative at 0, call it c_a, then $\frac{d}{dx} a^x = c_a \, a^x$ for every x: the slope of an exponential is proportional to its height, with one constant per base. Numerically, $(2^\epsilon - 1)/\epsilon = 0.69315$ at $\epsilon = 10^{-6}$, and the same quotient for base 10 is 2.30259. The number e is the base whose constant is 1, which gives the table's entry $\frac{d}{dx} e^x = e^x$. Once the logarithm is available, the chain rule identifies c_a for every base.

34.1.2.2 Rules for Combining Derivatives

A loss combines the entries of the table by sums, products, quotients, and compositions. Let g and h be differentiable at x; for the chain rule, let h be differentiable at x and g at $h(x)$. Then:

- **Sum rule.** $\frac{d}{dx}\big(g(x) + h(x)\big) = g'(x) + h'(x)$.
- **Product rule.** $\frac{d}{dx}\big(g(x)\,h(x)\big) = g(x)h'(x) + g'(x)h(x)$.
- **Quotient rule.** $\frac{d}{dx}\frac{g(x)}{h(x)} = \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2}$, where $h(x) \neq 0$.

- **Chain rule.** $\frac{d}{dx} g(h(x)) = g'(h(x))\, h'(x)$.

The sum and product rules are routine. Only the chain rule needs care: the obvious argument divides by $h(x+\epsilon) - h(x)$, which may be zero. The proofs use one fact. **A function differentiable at x is continuous at x:** $g(x+\epsilon) - g(x) = \epsilon \cdot \frac{g(x+\epsilon)-g(x)}{\epsilon}$ tends to $0 \cdot g'(x) = 0$.

Sum rule. Adding the two linear approximations gives

$$g(x+\epsilon) + h(x+\epsilon) = g(x) + h(x) + \epsilon\big(g'(x) + h'(x)\big) + o(\epsilon),$$

because the sum of two quantities that vanish faster than ϵ also vanishes faster than ϵ. By the converse of the linear approximation, the coefficient of ϵ is the derivative: the two first-order changes add. ■

Product rule. Adding and subtracting $g(x+\epsilon)h(x)$ in the numerator splits the difference quotient of the product into two quotients we know,

$$\frac{g(x+\epsilon)h(x+\epsilon) - g(x)h(x)}{\epsilon} = g(x+\epsilon)\,\frac{h(x+\epsilon)-h(x)}{\epsilon} + h(x)\,\frac{g(x+\epsilon)-g(x)}{\epsilon}.$$

As $\epsilon \to 0$ the right-hand side tends to $g(x)h'(x) + h(x)g'(x)$, because $g(x+\epsilon) \to g(x)$ by continuity. ■

Chain rule. The argument feeds the change of the inner function into the linear approximation of the outer one. Since g is differentiable at $h(x)$, the quantity $\rho(t) = \frac{g(h(x)+t)-g(h(x))}{t} - g'(h(x))$ for $t \neq 0$, with $\rho(0) = 0$, tends to 0 as $t \to 0$. For every t, including $t = 0$,

$$g(h(x)+t) = g(h(x)) + \big(g'(h(x)) + \rho(t)\big)\, t.$$

Put $t = h(x+\epsilon) - h(x) = \epsilon h'(x) + o(\epsilon)$, which tends to 0 with ϵ, so that $\rho(t) \to 0$ as well. Then

$$g(h(x+\epsilon)) - g(h(x)) = \big(g'(h(x)) + \rho(t)\big)\big(\epsilon h'(x) + o(\epsilon)\big) = \epsilon\, g'(h(x))\, h'(x) + o(\epsilon),$$

since every product other than $g'(h(x))\,\epsilon h'(x)$ contains a factor that vanishes faster than ϵ. The converse of the linear approximation gives the rule. ■

Rates multiply along a composition. Applied layer by layer to a network, the chain rule produces a product of local slopes, and evaluating that product is what backpropagation does. The quotient rule follows from the product and chain rules applied to $g \cdot h^{-1}$ (exercise 1).

34.1.2.3 Inverse Functions

The chain rule also differentiates a function that undoes another. A function g with $g(f(x)) = x$ for all x near some point is an *inverse* of f there. Suppose such a g exists and is differentiable at $f(x)$. Both hold whenever f' is continuous and nonzero near x, by

the inverse function theorem (Rudin 1976), which we use without proof. Differentiating both sides with the chain rule gives

$$g'(f(x))\,f'(x) = 1, \qquad \text{so} \qquad g'(y) = \frac{1}{f'(g(y))} \quad \text{at } y = f(x).$$

The slope of the inverse is the reciprocal of the slope of f, read at the matching point. The graph of g is the graph of f reflected in the diagonal, and the reflection turns rise over run upside down.

The logarithm undoes e^x, so $\log'(y) = 1/e^{\log y} = 1/y$, the table's fifth entry. The hypothesis $f' \neq 0$ is needed. For x^3 we have $f'(0) = 0$, and the formula gives no slope at $y = 0$. The inverse $x^{1/3}$ has no derivative at 0: its quotient $\epsilon^{1/3}/\epsilon = \epsilon^{-2/3}$ grows without bound, and the graph has a vertical tangent there.

With the logarithm in hand, the chain rule settles the two entries of the table that were still open. Since $a^x = e^{x \log a}$, the constant of the exponential with base a is $c_a = \log a$, and $\log 2 = 0.693$ and $\log 10 = 2.303$ are the constants measured earlier. In the same way $x^n = e^{n \log x}$ for $x > 0$, so $\frac{d}{dx} x^n = e^{n \log x} \cdot \frac{n}{x} = n x^{n-1}$ for every real n.

34.1.2.4 The Softplus, the Sigmoid, and the Logit

We now apply the rules to the running function. The chain rule with outer function log and inner function $1 + e^x$ gives

$$s'(x) = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} = \sigma(x),$$

so the slope of the softplus is the *sigmoid* σ. Applying the chain rule again to $\sigma = (1+e^{-x})^{-1}$, with outer function u^{-1} and inner function $1 + e^{-x}$ of slope $-e^{-x}$, gives

$$\sigma'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)),$$

because $1 - \sigma(x) = e^{-x}/(1 + e^{-x})$. A product $p(1-p)$ is largest at $p = \frac{1}{2}$, so the sigmoid is steepest at the origin: $\sigma(0) = \frac{1}{2}$ and $\sigma'(0) = \frac{1}{4}$. The inverse of σ is the *logit* $\operatorname{logit}(p) = \log \frac{p}{1-p}$, and the inverse-function rule gives $\operatorname{logit}'(p) = 1/\sigma'(\operatorname{logit} p) = \frac{1}{p(1-p)}$. At $p = \frac{1}{2}$ this is $4 = 1/\frac{1}{4}$.

The two constants of the logistic loss are now within reach. The loss is $\ell(w) = s(-w)$, so the chain rule gives $\ell'(w) = -\sigma(-w)$ and $\ell'(0) = -\frac{1}{2}$. Differentiating the slope once more gives $\ell''(w) = \sigma(-w)(1 - \sigma(-w))$ and $\ell''(0) = \frac{1}{4}$, the bend. Does the gap between $\ell(\epsilon)$ and the line $\ell(0) - \frac{1}{2}\epsilon$ shrink like ϵ^2, with constant $\frac{1}{2}\ell''(0) = \frac{1}{8}$?

```
# The logistic loss l(w) = log(1 + e^{-w}) near w = 0, where l'(0) = -1/2
def ell(w):
    return np.log(1 + np.exp(-w))

for eps in [0.1, 0.01, 0.001]:
```

```
    line = ell(0) - 0.5 * eps          # the linear approximation
    gap = ell(eps) - line
    print(f'eps = {eps:<5}: l = {ell(eps):.6f}, line = {line:.6f}, '
          f'gap = {gap:.2e}, gap / eps^2 = {gap / eps**2:.4f}')
```

```
eps = 0.1  : l = 0.644397, line = 0.643147, gap = 1.25e-03, gap / eps^2 = 0.1249
eps = 0.01 : l = 0.688160, line = 0.688147, gap = 1.25e-05, gap / eps^2 = 0.1250
eps = 0.001: l = 0.692647, line = 0.692647, gap = 1.25e-07, gap / eps^2 = 0.1250
```

The gap drops a hundredfold for every tenfold smaller step, and gap$/\epsilon^2$ settles at $0.125 = \frac{1}{2}\ell''(0)$. The error of the linear approximation is of second order, with a constant set by the curvature. The first line prints 0.1249, not 0.1250: the gap is not quite $\epsilon^2/8$, and Section 34.1.5.2 finds the term that is missing.

What the slope at a point says about the function on a whole interval is a different kind of question.

34.1.3 The Mean Value Theorem

A positive slope means that the function climbs, and a point with slope zero is a candidate minimum. Gradient descent depends on both readings. Each infers the behavior of f on an interval from its derivative at single points, and neither is part of the definition. The second is the easier one.

Lemma (Fermat). *If f has a local maximum or a local minimum at an interior point ξ of an interval and is differentiable at ξ, then $f'(\xi) = 0$.*

Proof. At a local maximum, $f(\xi+\epsilon)-f(\xi) \leq 0$ for all small ϵ of either sign. The difference quotient is then ≤ 0 for $\epsilon > 0$ and ≥ 0 for $\epsilon < 0$. Since f is differentiable at ξ, both one-sided limits equal $f'(\xi)$, and a number that is both ≤ 0 and ≥ 0 is zero. A minimum is handled the same way. ■

The first statement needs the Mean Value Theorem. Every error bound in the rest of this section is this theorem applied to a suitable function. In a picture (Figure 34.3), the chord joining the endpoints of the graph over $[a, b]$ is parallel to the tangent at some interior point.

For $f(x) = x^2$ on $[0, 2]$ the chord has slope $(4-0)/2 = 2$, and the tangent slope $f'(\xi) = 2\xi$ equals 2 at $\xi = 1$, the midpoint. The theorem needs a derivative at every interior point. The function $|x|$ on $[-1, 1]$ has $f(-1) = f(1)$, so its chord is flat, yet no interior point has slope 0: the slopes are -1 on the left, $+1$ on the right, and at 0 there is none.

Proposition (Mean Value Theorem). *If f is continuous on $[a, b]$ and differentiable on (a, b), then there is a point $\xi \in (a, b)$ with*

$$f'(\xi) = \frac{f(b) - f(a)}{b - a}. \tag{34.3}$$

Proof. First take the flat case $f(a) = f(b)$, which is Rolle's theorem. We use one fact from real analysis, the **extreme value theorem**: a continuous function on a closed interval attains a maximum and a minimum (Rudin 1976). If both occur at the endpoints then f is constant and $f' \equiv 0$ inside. Otherwise an extremum sits at some interior ξ, and $f'(\xi) = 0$ by Fermat's lemma.

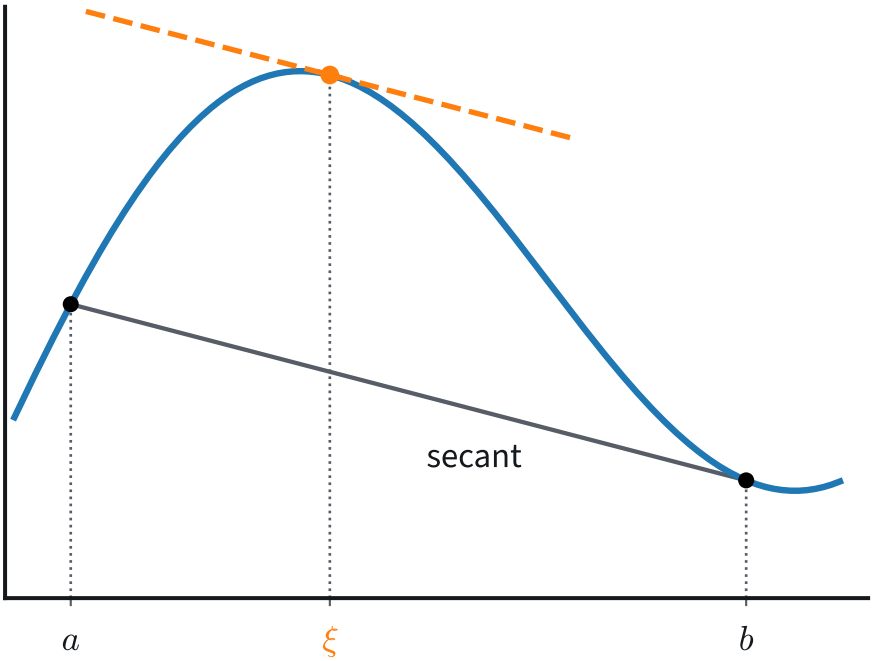


Figure 34.3: Over the interval from a to b, the secant chord through the endpoints has slope equal to the average rate of change; the Mean Value Theorem says some interior point ξ has a tangent of that same slope, parallel to the chord.

For the general case, subtract the chord. The tilted function $\tilde{f}(x) = f(x) - \frac{f(b)-f(a)}{b-a}(x-a)$ has $\tilde{f}(a) = \tilde{f}(b) = f(a)$, so the flat case gives a ξ with $\tilde{f}'(\xi) = 0$, which is Equation 34.3. ■

The average rate of change over an interval is attained as an instantaneous rate at some interior point. We will use two consequences again and again.

1. *If $f' > 0$ on an interval, then f is strictly increasing there.* For any $x_1 < x_2$ in the interval, Equation 34.3 gives $f(x_2) - f(x_1) = f'(\xi)(x_2 - x_1) > 0$ for some ξ between them. Likewise $f' < 0$ gives a strictly decreasing f, and $f' = 0$ a constant one.
2. *If $f' < 0$ on $(x_0 - \delta, x_0)$ and $f' > 0$ on $(x_0, x_0 + \delta)$ for some $\delta > 0$, and f is continuous at x_0, then x_0 is a strict local minimum.* Apply the first consequence on each side: f falls into x_0 and rises out of it.

For the running function, $s' = \sigma > 0$ everywhere, so the softplus is strictly increasing on all of $\mathbb{R}$.

34.1.4 Gradient Descent in One Variable

34.1.4.1 The Gradient-Descent Step

The linear approximation predicts the effect of any step ϵ: $f(x + \epsilon) \approx f(x) + \epsilon f'(x)$. To lower f we choose ϵ with the sign opposite to the slope. Taking $\epsilon = -\eta f'(x)$ with a *step size* $\eta > 0$, the line predicts the change

$$f(x - \eta f'(x)) - f(x) \approx -\eta\, f'(x)^2,$$

which is negative whenever $f'(x) \neq 0$. Whatever the sign of the slope, stepping against it lowers the linear model by an amount proportional to the slope squared (Figure 34.4).

Repeating the step gives one-dimensional **gradient descent**,

$$x_{t+1} = x_t - \eta\, f'(x_t). \tag{34.4}$$

Each iterate moves against the current slope, and the iteration stops moving at a *stationary point*, where $f'(x_t) = 0$. For a network, each coordinate of the weight update $w_i \leftarrow w_i - \eta\, \partial \ell / \partial w_i$ has this form, with all coordinates updated at once. On the logistic loss from $w = 0$, the slope is $\ell'(0) = -\frac{1}{2}$, so the step is $\epsilon = \eta/2$ and the predicted drop is $\eta/4$.

The prediction $-\eta f'(x)^2$ is negative for every η, however large. Yet a large step can overshoot the minimum and land higher than it started. The line does not register this, because it carries no information about how the graph bends. The simplest bowl, x^2, shows where its prediction fails.

34.1.4.2 The Quadratic Case

For $f(x) = x^2$ the slope is $f'(x) = 2x$, and the gradient-descent step becomes

$$x_{t+1} = x_t - \eta\,(2x_t) = (1 - 2\eta)\, x_t.$$

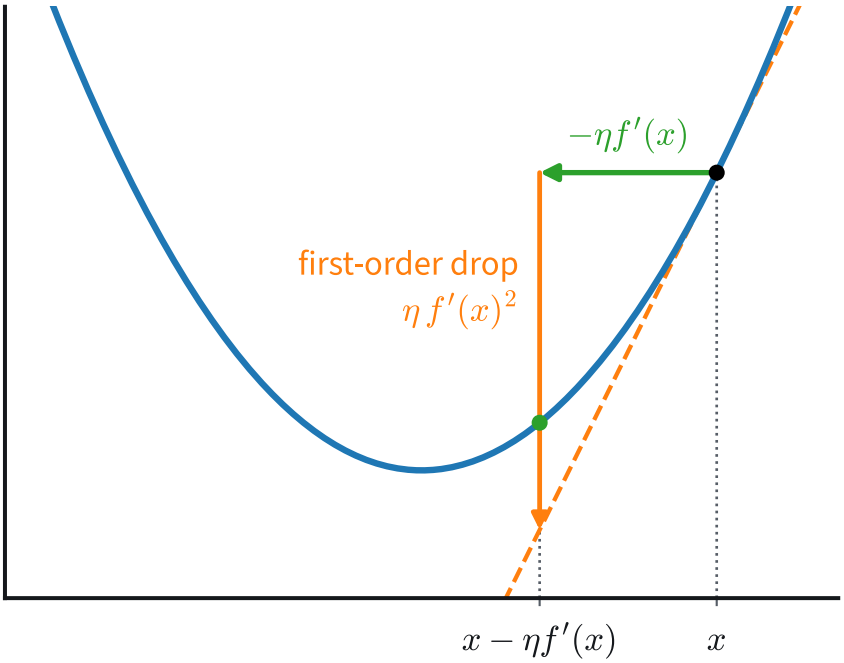


Figure 34.4: One gradient-descent step on a 1-D bowl. At x the tangent line has slope $f'(x)$; the step $-\eta f'(x)$ moves opposite the slope and lands at $x - \eta f'(x)$, lower on the curve, with the drop equal to $\eta f'(x)^2$ to first order.

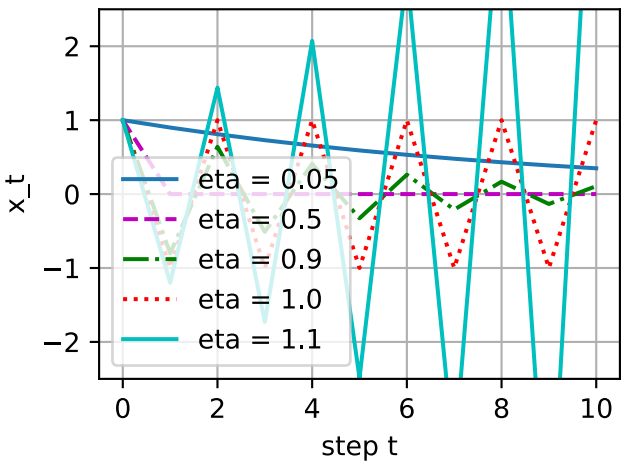
Each step multiplies the iterate by the same factor, so $x_t = (1 - 2\eta)^t x_0$. The iterates converge to the minimum $x = 0$ if and only if $|1 - 2\eta| < 1$, that is, for $0 < \eta < 1$. Inside this range a small η converges slowly and monotonically, $\eta = \frac{1}{2}$ reaches the minimum in one step, and $\frac{1}{2} < \eta < 1$ overshoots and alternates inward. At $\eta = 1$ the iterates alternate forever between $\pm x_0$, and for $\eta > 1$ they diverge.

Ten steps from $x_0 = 1$ at five step sizes show one run per regime.

```
# Gradient descent on f(x) = x^2 from x0 = 1, for several step sizes
def gd(eta, steps=10, x0=1.0):
    xs = [x0]
    for _ in range(steps):
        xs.append(xs[-1] - eta * (2 * xs[-1]))  # x <- x - eta * f'(x)
    return xs

etas = [0.05, 0.5, 0.9, 1.0, 1.1]
trajectories = [gd(eta) for eta in etas]
for eta, xs in zip(etas, trajectories):
    print(f'eta = {eta:.2f} -> x_10 = {xs[-1]:+.5f}')
d2l.plot(list(range(11)), trajectories, 'step t', 'x_t', ylim=[-2.5, 2.5],
         legend=[f'eta = {eta}' for eta in etas],
         fmts=('-', 'm--', 'g-.', 'r:', 'c-'))
```

```
eta = 0.05 -> x_10 = +0.34868
eta = 0.50 -> x_10 = +0.00000
eta = 0.90 -> x_10 = +0.10737
eta = 1.00 -> x_10 = +1.00000
eta = 1.10 -> x_10 = +6.19174
```



The printed values are the powers $(1-2\eta)^{10}$: the cautious $\eta = 0.05$ has reached only 0.349, $\eta = 0.9$ has alternated down to 0.107, $\eta = 1$ is back at 1, and $\eta = 1.1$ has grown to 6.19. The threshold $\eta = 1$ is $2/f''$, since $f'' \equiv 2$ here, and Section 34.1.5.4 shows that a bound L on the curvature gives the threshold $2/L$ for every function.

34.1.5 Curvature and Taylor Series

How far can the line be trusted, and how large a step is safe? Both answers need the rate at which the slope itself changes.

34.1.5.1 The Second Derivative

The derivative f' is itself a function, so we can differentiate it again. The result is the *second derivative*, $f'' = \frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right)$: the rate of change of the slope. Repeating gives the n-th derivative,

$$f^{(n)}(x) = \frac{d^n f}{dx^n} = \left(\frac{d}{dx}\right)^n f.$$

A positive f'' means the slope increases, so the graph bends upward; a negative f'' means it bends downward. At a stationary point this sign decides between a minimum and a maximum.

Proposition (second-derivative test). *Let $f'(x_0) = 0$ and let $f''(x_0)$ exist. If $f''(x_0) > 0$, then x_0 is a strict local minimum; if $f''(x_0) < 0$, it is a strict local maximum.*

Proof. Since $f'(x_0) = 0$, the definition of $f''(x_0)$ reads $f''(x_0) = \lim_{\epsilon \to 0} f'(x_0 + \epsilon)/\epsilon$. If this limit is positive, the quotient $f'(x_0 + \epsilon)/\epsilon$ is positive for all small $\epsilon \neq 0$, so $f'(x_0 + \epsilon)$ has the sign of ϵ: negative just left of x_0, positive just right of it. The second consequence of the Mean Value Theorem then makes x_0 a strict local minimum. The case $f''(x_0) < 0$ is the same with signs reversed. ■

The step that is not routine is the last one: a sign pattern of the slope at points becomes a minimum on an interval only through the Mean Value Theorem. The test certifies a neighborhood of x_0, not the whole function. With many variables, the signs of the eigenvalues of the matrix of second derivatives play the same role.

At a stationary point with $f''(x_0) = 0$ the test is silent (Figure 34.5, bottom row): x^3, x^4 and $-x^4$ all have $f'(0) = f''(0) = 0$, yet 0 is an inflection point, a minimum, and a maximum respectively. Exercise 5 derives the test that higher derivatives give in this case.

34.1.5.2 The Taylor Polynomial

The line matches the value and the slope of f at a base point x_0. A parabola can also match the curvature. A quadratic $q(x) = ax^2 + bx + c$ has

$$q'(x) = 2ax + b, \qquad q''(x) = 2a,$$

so its second derivative is the constant $2a$, and matching the curvature of f fixes $a = f''(x_0)/2$. Matching value and slope as well fixes b and c. For $f(x) = \sin(x)$, with $\sin' = \cos$ and $\sin'' = -\sin$, the best parabola at x_0 is

$$q(x) = \sin(x_0) + \cos(x_0)\,(x - x_0) - \tfrac{1}{2}\sin(x_0)\,(x - x_0)^2.$$

Because it also follows the bend, it stays close to the curve over a wider interval than the tangent (Figure 34.6, drawn at a base point where $\sin x_0 \neq 0$, so the parabola is curved).

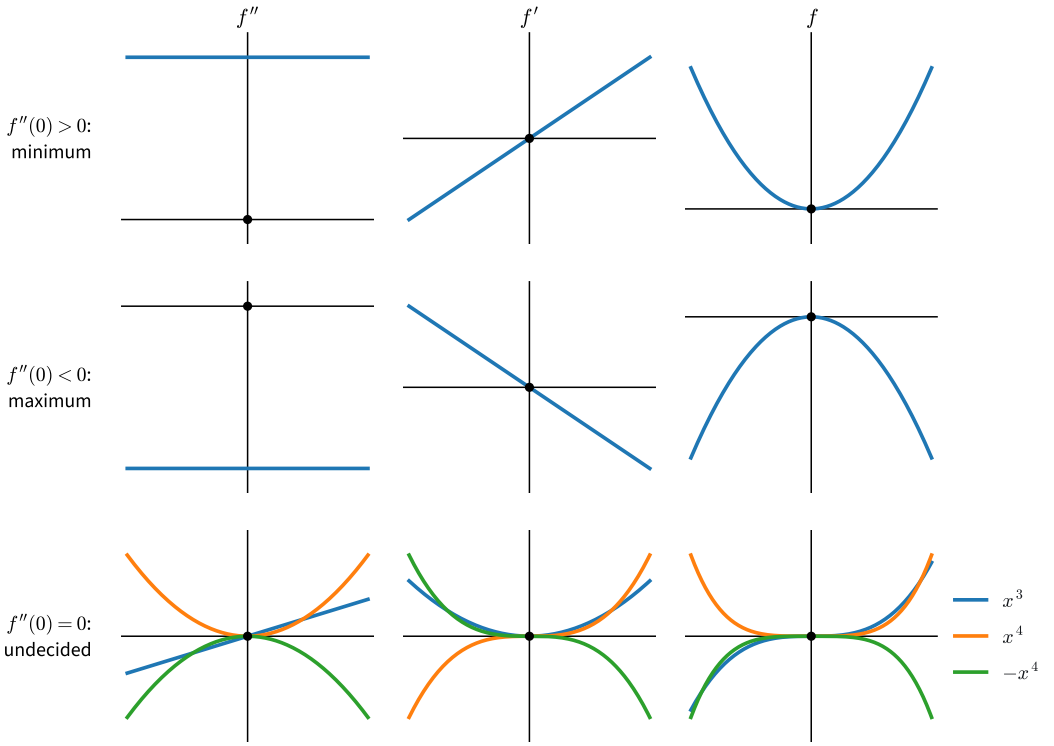


Figure 34.5: The second-derivative test. Rows: $f''(0) > 0$ (a bowl, minimum), $f''(0) < 0$ (a dome, maximum), and $f''(0) = 0$, where x^3, x^4 and $-x^4$ share $f'(0) = f''(0) = 0$ yet have an inflection point, a minimum and a maximum at 0. Columns: f'', f', f.

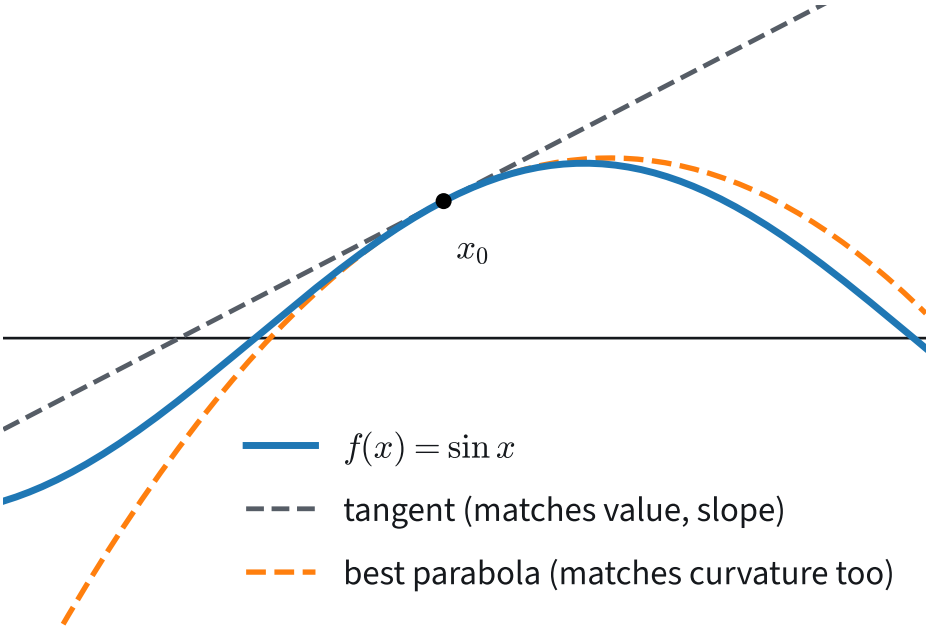


Figure 34.6: The sine curve with its two local models at x_0: the tangent line matches value and slope, while the best parabola also matches curvature and remains accurate over a wider interval.

For the running function at $x_0 = 0$, the values $s(0) = \log 2$, $s'(0) = \frac{1}{2}$ and $s''(0) = \frac{1}{4}$ give the best parabola $\log 2 + \frac{x}{2} + \frac{x^2}{8}$. Its quadratic coefficient $\frac{1}{8}$ is the constant at which gap$/\epsilon^2$ settled for the logistic loss $\ell(w) = s(-w)$.

The same construction works to any degree. The degree-n **Taylor polynomial** of f at x_0 is

$$P_n(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_0)}{i!}(x - x_0)^i,$$

the unique polynomial of degree at most n whose derivatives through order n match those of f at x_0. The factorials appear because differentiating $(x - x_0)^i$ i times at x_0 gives $i!$, while every other term gives 0. For P_1 this is the tangent and for P_2 the best parabola. For e^x at $x_0 = 0$, where every derivative equals 1, the Taylor polynomial is $P_n(x) = \sum_{i=0}^{n} x^i/i!$, and each added term lets the polynomial follow the curve a little farther from the base point. How far it can be trusted, and how fast the error grows beyond, is what the remainder tells us.

34.1.5.3 The Lagrange Remainder

The Mean Value Theorem gives an exact expression for the error of P_n, with one unknown point. This is the main result of the section: the descent lemma and Newton's rate are both its case $n = 1$.

Proposition (Lagrange remainder). *If f is $(n+1)$-times differentiable on an open interval containing x_0 and x, with $x \neq x_0$, then*

$$f(x) = P_n(x) + R_n(x), \qquad R_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\,(x-x_0)^{n+1}, \tag{34.5}$$

for some ξ strictly between x_0 and x. The case $n = 0$ is the Mean Value Theorem, $f(x) = f(x_0) + f'(\xi)(x-x_0)$.

Proof. The only idea is the choice of an auxiliary function; the rest is differentiation. Fix x and make the base point the variable: for t between x_0 and x, let

$$G(t) = f(x) - \sum_{k=0}^{n} \frac{f^{(k)}(t)}{k!}\,(x-t)^k - M\,\frac{(x-t)^{n+1}}{(n+1)!},$$

the error at x of the Taylor polynomial based at t, minus a term of the shape we expect that error to have. Choose the constant M so that $G(x_0) = 0$. The proof has three steps.

First, the endpoints vanish. The choice of M gives $G(x_0) = 0$, while $G(x) = 0$ because every positive power of $(x-t)$ vanishes at $t = x$ and the $k = 0$ term becomes $f(x) - f(x)$. Rolle's theorem therefore supplies a point ξ strictly between x_0 and x with $G'(\xi) = 0$.

Second, differentiate the sum. By the product rule, the k-th summand with $k \geq 1$ contributes

$$-\frac{f^{(k+1)}(t)}{k!}(x-t)^k + \frac{f^{(k)}(t)}{(k-1)!}(x-t)^{k-1}$$

to $G'(t)$, and the $k = 0$ term contributes $-f'(t)$. The terms cancel in pairs along the sum (it telescopes), leaving only the last derivative:

$$G'(t) = -\frac{f^{(n+1)}(t)}{n!}\,(x-t)^n + M\,\frac{(x-t)^n}{n!}.$$

Third, set $t = \xi$. Since $(x-\xi)^n \neq 0$, the equation $G'(\xi) = 0$ forces $M = f^{(n+1)}(\xi)$. Substituting this value into $G(x_0) = 0$ gives Equation 34.5. ∎

The remainder has the shape of the next Taylor term, with the derivative evaluated at an unknown interior point ξ instead of at x_0. When $f^{(n+1)}$ is bounded near x_0, the error therefore scales as $|x-x_0|^{n+1}$: each additional matched derivative raises the order by one.

For the running function, $s''' = \sigma(1-\sigma)(1-2\sigma)$ vanishes at 0 and $s''''(0) = -\frac{1}{8}$, so

$$P_4(x) = \log 2 + \frac{x}{2} + \frac{x^2}{8} - \frac{x^4}{192}.$$

At $x = 0.5$ this gives 0.974072 against $s(0.5) = 0.974077$, an error of 5×10^{-6}. The error is even smaller than the x^5 scale suggests, because $s(x) - \frac{x}{2} = \log(2\cosh\frac{x}{2})$ is an even

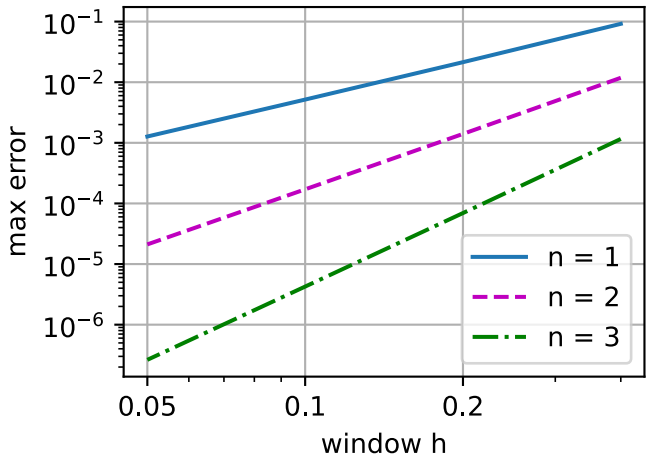
function and the next nonzero term is of sixth order. The quartic term is also what turned 0.1250 into the printed 0.1249 at $\epsilon = 0.1$: there $\frac{1}{8} - \frac{\epsilon^2}{192} = 0.12495$.

The scaling is a prediction we can test. If the error of P_n on the window $|x| \le h$ grows like h^{n+1}, halving the window should divide the worst error by 2^{n+1}: by 4 for the tangent, 8 for the parabola, and 16 for the cubic. Does it, for e^x at $x_0 = 0$?

```
# Max error of P_n for e^x on |x| <= h. Lagrange predicts error ~ h^(n+1),
# so halving h should divide it by 2^(n+1)
def taylor_error(n, h, m=1000):
    x = h * np.arange(-m, m + 1) / m
    P = sum(x**k / math.factorial(k) for k in range(n + 1))
    return np.max(np.abs(np.exp(x) - P))

hs = [0.4, 0.2, 0.1, 0.05]
errs = [[taylor_error(n, h) for h in hs] for n in (1, 2, 3)]
for n, e in zip((1, 2, 3), errs):
    ratios = ' '.join(f'{a / b:.1f}' for a, b in zip(e, e[1:]))
    print(f'n = {n}: ratios between successive windows {ratios}, '
          f'predicted {2**(n + 1)}')
d2l.plot(hs, errs, 'window h', 'max error', xscale='log', yscale='log',
         legend=['n = 1', 'n = 2', 'n = 3'])
```

```
n = 1: ratios between successive windows 4.3 4.1 4.1, predicted 4
n = 2: ratios between successive windows 8.4 8.2 8.1, predicted 8
n = 3: ratios between successive windows 16.7 16.3 16.2, predicted 16
```



The measured ratios sit near 4, 8, and 16, and on the log–log plot the three error curves are straight lines with slopes 2, 3, and 4, that is, $n + 1$. The ratios overshoot slightly on the wider windows because the factor e^{ξ} in the remainder is larger there.

The remainder also settles whether the infinite series reproduces the function. For e^x every derivative is e^x, so Equation 34.5 with $x_0 = 0$ gives

$$\left|e^x - P_n(x)\right| = \frac{e^{\xi}\,|x|^{n+1}}{(n+1)!} \le \frac{e^{|x|}\,|x|^{n+1}}{(n+1)!}.$$

For fixed x the right-hand side tends to 0 as $n \to \infty$, because factorials outgrow powers. Hence

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$$

on all of $\mathbb{R}$. Infinitely many derivatives do not guarantee this. For $f(x) = e^{-1/x^2}$ with $f(0) = 0$, every derivative vanishes at 0, so the Taylor series there is identically zero and agrees with f only at $x = 0$ (Figure 34.7). Nothing later depends on the series: we use the finite polynomials P_n, whose error Equation 34.5 bounds directly.

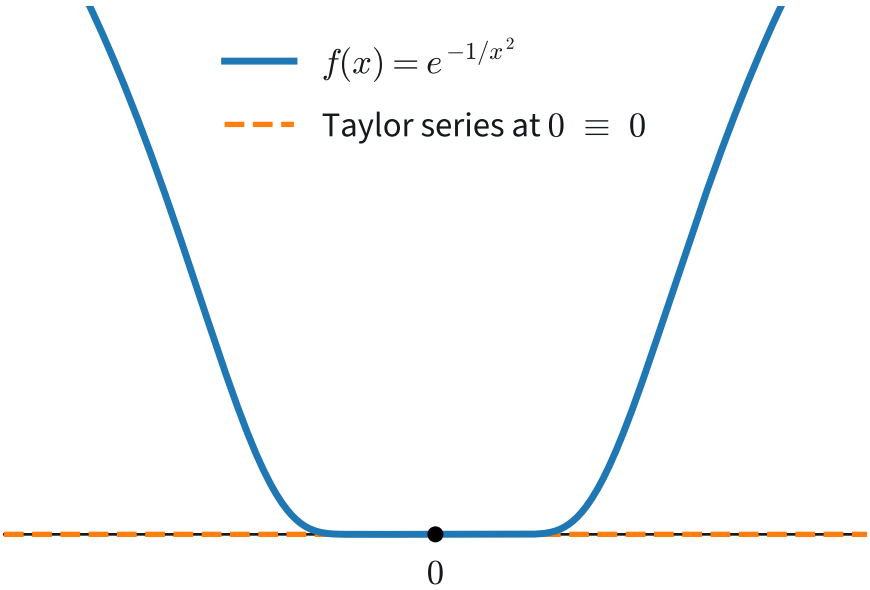


Figure 34.7: The smooth function $f(x) = e^{-1/x^2}$ has every derivative equal to zero at the origin, so its Taylor series there is identically zero. The series converges everywhere yet agrees with f only at $x = 0$: smooth does not imply analytic.

34.1.5.4 The Descent Lemma

Does a bound on the curvature guarantee descent for every function, as it did for x^2? The Lagrange remainder with $n = 1$ answers yes. Of the results in this section, this is the one that optimization uses most.

Proposition (descent lemma). *Let f be twice differentiable on an open interval I containing x and $x - \eta f'(x)$, with $|f''| \le L$ on I. Then*

$$f(x-\eta\,f'(x)) \;\le\; f(x)-\eta\Big(1-\frac{L\eta}{2}\Big)f'(x)^2. \tag{34.6}$$

In particular f strictly decreases whenever $0<\eta<2/L$ and $f'(x)\neq 0$, and the guaranteed decrease is largest at $\eta=1/L$, where it is $f'(x)^2/(2L)$.

Proof. For a step u, the Lagrange remainder with $n=1$ gives $f(x+u)=f(x)+f'(x)u+\frac{1}{2}f''(\xi)u^2\le f(x)+f'(x)u+\frac{L}{2}u^2$. Put $u=-\eta f'(x)$; the right-hand side becomes $f(x)-\eta(1-L\eta/2)f'(x)^2$. The bracket is positive for $\eta<2/L$, and the coefficient $\eta(1-L\eta/2)$ is largest at $\eta=1/L$. ■

The lemma replaces the actual bend $f''(\xi)$ by its worst case L. The resulting quadratic $f(x)+f'(x)u+\frac{L}{2}u^2$ is a ceiling that touches f at x and lies above it on I. Its minimizer is the step $\eta=1/L$ (Figure 34.8). This bound is the standard tool of smooth optimization (Nesterov 2018). It also holds when f' is only L-*Lipschitz*, meaning that slopes at two points differ by at most L times their distance, $|f'(u)-f'(v)|\le L|u-v|$. That is the form used for functions of many variables (Equation 35.5), and exercise 2 proves it in one variable.

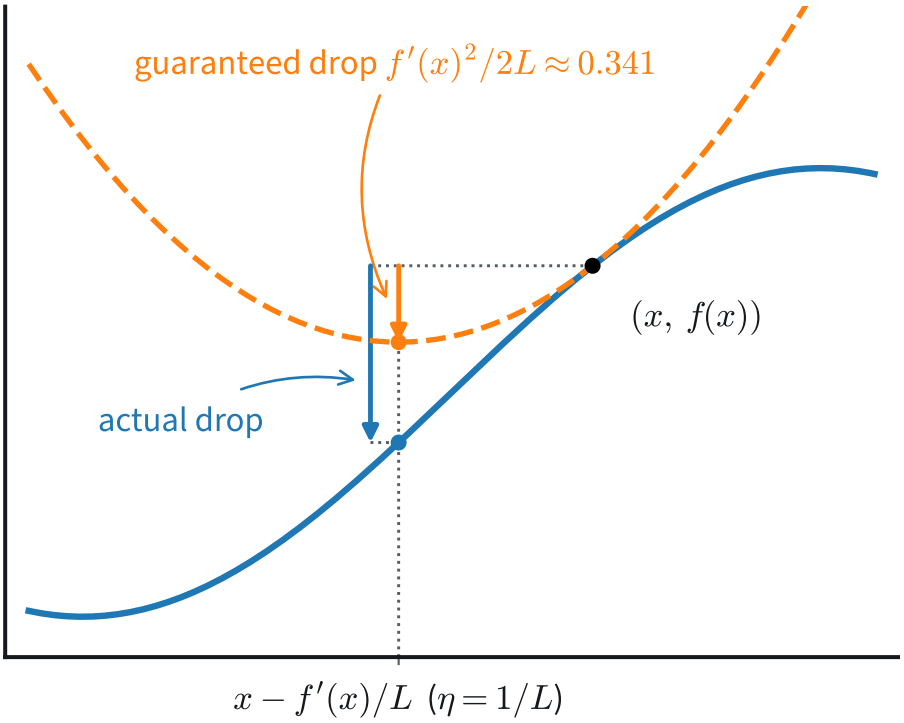


Figure 34.8: For $|f''|\le L$, $f(x)+f'(x)u+\frac{L}{2}u^2$ is a quadratic upper bound in the step u that equals f at the base point. The gradient step with $\eta=1/L$ minimizes this bound and guarantees a decrease of at least $\frac{1}{2L}f'(x)^2$. Here $f(x)=\sin(2x)$ with $L=4$, and the actual decrease exceeds the worst-case guarantee.

For x^2 we have $L=2$, and $2/L=1$ is the threshold found from the closed form. On the logistic loss, $\ell''(w)=\sigma(-w)(1-\sigma(-w))\le\frac{1}{4}$, with the maximum $\frac{1}{4}$ at $w=0$, so $L=\frac{1}{4}$

and every $\eta < 8$ decreases ℓ from any starting weight. From $w = 0$ with $\eta = 1$ the step lands at $w_1 = 0.5$. The bound promises $0.693147 - (1 - \frac{1}{8}) \cdot \frac{1}{4} = 0.4744$, and the actual value is $\ell(0.5) = 0.4741$. The bound is nearly tight because the curvature along this step is close to its maximum L. With $\eta = 4 = 1/L$ the bound is 0.1931 and the actual value $\ell(2) = 0.1269$: the curvature falls off away from 0, so the function descends further than the worst case allows for.

34.1.5.5 Newton's Method

The descent lemma uses the worst-case curvature L. Using the actual curvature $f''(x_t)$ at the current iterate gives a step that is exact on parabolas. The model of f at x_t is the Taylor polynomial P_2,

$$q(x) = f(x_t) + f'(x_t)\,(x - x_t) + \tfrac{1}{2} f''(x_t)\,(x - x_t)^2.$$

When $f''(x_t) > 0$ the parabola opens upward and has a unique minimizer. Solving $q'(x) = f'(x_t) + f''(x_t)(x - x_t) = 0$ gives the update of **Newton's method**:

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)}. \tag{34.7}$$

The step is a gradient step with the local step size $1/f''(x_t)$, and on any parabola it reaches the minimizer in one step. Figure 34.9 shows one step on $f(x) = x^4/4 - x$, whose minimizer is $x^\star = 1$. At $x_t = 2$ the values are $f = 2$, $f' = 7$, $f'' = 12$, so $x_{t+1} = 2 - 7/12 = 1.4167$.

The cell continues the iteration from $x_0 = 2$ and prints the error at each step. Its second loop runs Newton's method from $x_0 = 1.4$ and from $x_0 = 1.3$ on another function, $x \arctan x - \frac{1}{2}\log(1 + x^2)$, to which we come in a moment.

```
# Newton on f(x) = x^4/4 - x: solve f'(x) = x^3 - 1 = 0, root x* = 1
x = 2.0
for t in range(6):
    print(f't = {t}: x = {x:.12f}, error = {abs(x - 1):.1e}')
    x = x - (x**3 - 1) / (3 * x**2)  # x - f'(x) / f''(x)

# Newton on the convex f(x) = x arctan(x) - log(1 + x^2)/2,
# with f' = arctan(x) and f'' = 1/(1 + x^2)
for x0 in [1.4, 1.3]:
    x, xs = x0, []
    for t in range(6):
        x = x - math.atan(x) * (1 + x**2)
        xs.append(x)
    print(f'from x0 = {x0}:', ', '.join(f'{v:.2f}' for v in xs))
```

```
t = 0: x = 2.000000000000, error = 1.0e+00
t = 1: x = 1.416666666667, error = 4.2e-01
t = 2: x = 1.110534409842, error = 1.1e-01
t = 3: x = 1.010636768405, error = 1.1e-02
t = 4: x = 1.000111557304, error = 1.1e-04
```

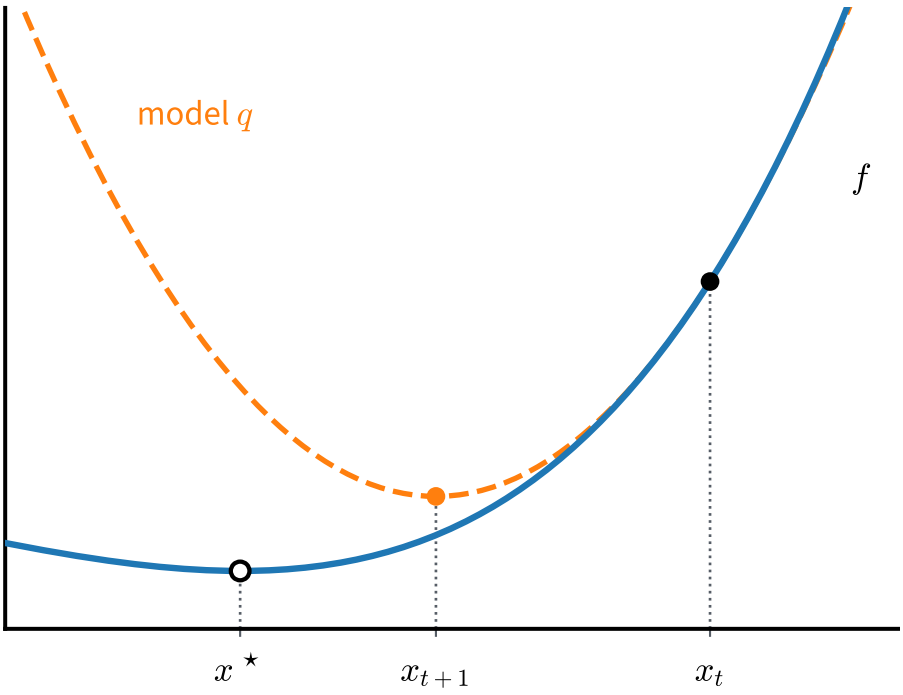


Figure 34.9: One Newton step on $f(x) = x^4/4 - x$ from $x_t = 2$. The parabola q matches the value, slope and curvature of f at x_t; its minimizer is the next iterate $x_{t+1} = x_t - f'(x_t)/f''(x_t) = 2 - 7/12$, closer to the minimizer $x^\star = 1$.

```
t = 5: x = 1.000000012443, error = 1.2e-08
from x0 = 1.4: -1.41, 1.45, -1.55, 1.85, -2.89, 8.71
from x0 = 1.3: -1.16, 0.86, -0.37, 0.03, -0.00, 0.00
```

On $x^4/4 - x$ the errors run 1.0, 0.42, 0.11, 1.1×10^{-2}, 1.1×10^{-4}, 1.2×10^{-8}: after the first steps, each step squares the previous error and doubles the number of correct digits. Gradient descent on x^2 shrank the error by the fixed factor $|1 - 2\eta|$ per step, which gains a fixed number of digits per step.

The Lagrange remainder explains the squaring. Let $x^\star$ be a stationary point and $e_t = x_t - x^\star$ the error. Applied with $n = 1$ to f' at the base point x_t and evaluated at $x^\star$, Equation 34.5 gives $0 = f'(x_t) - f''(x_t)\, e_t + \frac{1}{2} f'''(\xi_t)\, e_t^2$. Dividing by $f''(x_t) \neq 0$ and inserting the update Equation 34.7 leaves

$$e_{t+1} = \frac{f'''(\xi_t)}{2 f''(x_t)}\, e_t^2 \tag{34.8}$$

for some ξ_t between x_t and $x^\star$, when f is three times differentiable. For $x^4/4 - x$ the factor tends to $f'''(1)/(2f''(1)) = 6/(2 \cdot 3) = 1$, and the printed errors give $e_{t+1}/e_t^2 \approx 0.42, 0.64, 0.87, 0.99$. The general theorem, quadratic convergence from any start close enough to a minimizer with positive curvature, is Equation 35.18.

"Close enough" cannot be dropped, even for a strictly convex function. The function $f(x) = x \arctan x - \frac{1}{2}\log(1+x^2)$ has $f'(x) = \arctan x$ and $f''(x) = 1/(1+x^2) > 0$, so it is strictly convex with its unique minimum at 0. From $x_0 = 1.4$, the second loop of the cell printed the iterates $-1.41, 1.45, -1.55, 1.85, -2.89, 8.71$: Newton's method diverges. From $x_0 = 1.3$ it converges (-1.16, 0.86, -0.37, 0.03, ...), and the boundary between the two behaviors lies near $x_0 \approx 1.39$.

The model shows the cause. Far from 0 the curvature $1/(1 + x^2)$ is small, so the parabola q is flat and its minimizer lies far beyond $x^\star$. The remedy is to shorten the step by a factor chosen so that f decreases, which is called *damping*. Section 35.2.1 develops it, together with the cost of a Newton step in many dimensions.

34.1.6 Nondifferentiable Points

Every ReLU network has corners: $\mathrm{ReLU}(x) = \max(0, x)$ has no derivative at 0. Backpropagation still returns a number there. Which number is it, and is it a valid slope?

34.1.6.1 One-Sided Derivatives

The difference quotient Equation 34.1 shows the trouble directly. For $f(x) = |x|$ at $x = 0$,

$$\frac{|0 + \epsilon| - |0|}{\epsilon} = \frac{|\epsilon|}{\epsilon} = \begin{cases} +1 & \epsilon > 0, \\ -1 & \epsilon < 0. \end{cases}$$

The quotient is constant on each side, and the two constants differ. Coming from the right the function rises with slope $+1$; coming from the left it falls with slope -1. These are the **one-sided derivatives** $f'_+(0) = +1$ and $f'_-(0) = -1$, and the ordinary derivative exists only when they agree. For ReLU at 0 the same computation gives $f'_-(0) = 0$, flat on the left,

and $f'_+(0) = 1$ on the right. No slope is defined at these corners, and yet backpropagation must return a number. What does autograd return?

```
# What autograd returns at the corner, where no derivative exists
for name, fn in [('d|x|/dx', torch.abs), ("ReLU'", torch.relu)]:
    x = torch.tensor(0.0, requires_grad=True)
    fn(x).backward()
    print(f"{name} at 0: autograd returns {x.grad.item():.1f}")
```

```
d|x|/dx at 0: autograd returns 0.0
ReLU' at 0: autograd returns 0.0
```

For ReLU$'$ at 0 autograd returns 0, the left-hand slope. For $\frac{d}{dx}|x|$ at 0 it returns a fixed value chosen by convention.

Here it returns 0, the convention $\operatorname{sign}(0) = 0$.

34.1.6.2 Subgradients and Optimality

A corner has no single tangent, but it has many lines that touch the graph there and stay below it. For a *convex* function, one whose chords all lie on or above its graph (Section 35.3), these lines take over the role of the tangent. A number g is a **subgradient** of a convex function f at x if

$$f(y) \geq f(x) + g\,(y - x) \quad \text{for all } y, \tag{34.9}$$

that is, if the line through $(x, f(x))$ with slope g never rises above the graph. The set of all subgradients at x is the **subdifferential** $\partial f(x)$ (Rockafellar 1970). Where f is differentiable it contains only $f'(x)$. At a corner it is the interval between the one-sided derivatives, the fan of supporting lines in Figure 34.10:

$$\partial|x|(0) = [-1, 1], \qquad \partial\,\mathrm{ReLU}(0) = [0, 1].$$

Each value that autograd returned lies in the corresponding interval.

The running function shows where the interval comes from. The sharpened softplus $\mathrm{softplus}_\beta(x) = \frac{1}{\beta}s(\beta x)$ approaches $\mathrm{ReLU}(x)$ as $\beta \to \infty$, and its slope is $\sigma(\beta x)$ (Figure 34.11). At $x = \pm 0.1$ the slopes are 0.40 and 0.60 for $\beta = 4$, and 0.17 and 0.83 for $\beta = 16$. The subdifferential $[0, 1]$ is the set of values that these slopes approach as $\beta \to \infty$ and $x \to 0$ together.

Subgradients also give a test for a minimum. With $g = 0$, the definition Equation 34.9 reads $f(y) \geq f(x)$ for all y, so x is a global minimizer of a convex f if and only if

$$0 \in \partial f(x). \tag{34.10}$$

Both directions are this one sentence. For $|x|$, $0 \in [-1, 1]$ certifies the origin as the global minimum although no derivative exists there. Without convexity, $f'(x) = 0$ is only necessary (Fermat's lemma): x^3 has $f'(0) = 0$ and no minimum.

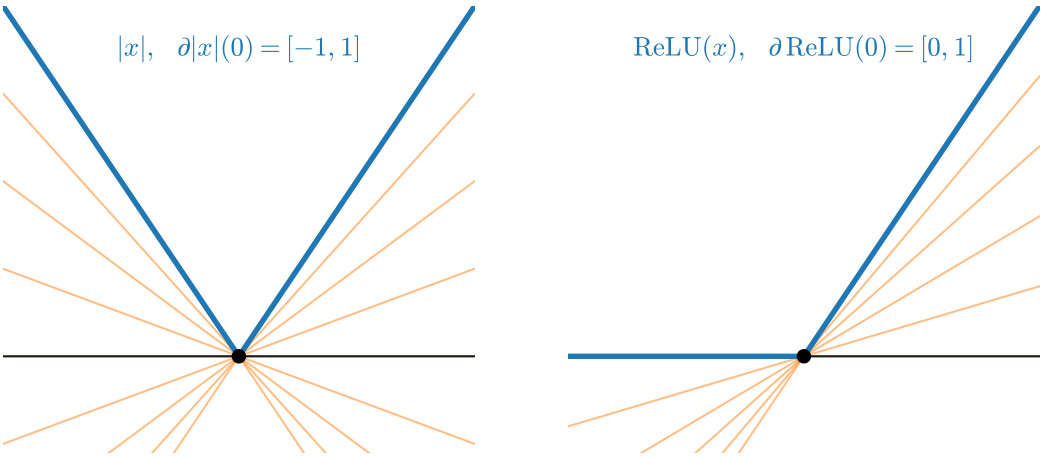


Figure 34.10: Two corners and their subdifferentials: the fan of supporting lines through the origin of $|x|$ (left) and of ReLU (right), with slopes running over $\partial|x|(0)$ and $\partial\,\mathrm{ReLU}(0)$ respectively. Off each corner there is a single tangent and the subdifferential collapses to the ordinary derivative.

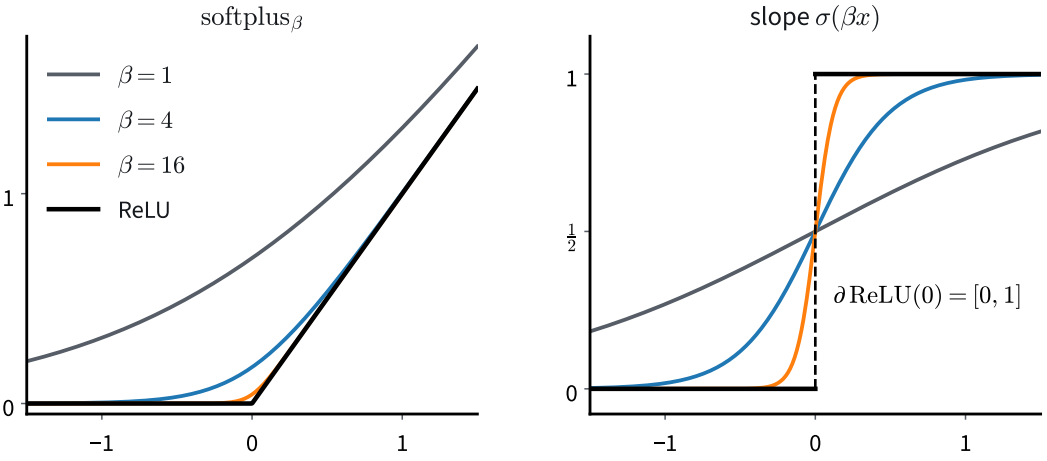


Figure 34.11: Smoothing the corner. Left: $\mathrm{softplus}_\beta(x) = \frac{1}{\beta}\log(1 + e^{\beta x})$ approaches ReLU as β grows. Right: its slope $\sigma(\beta x)$ steepens into the step from 0 to 1; near $x = 0$ the slopes of the smooth neighbours fill the interval $[0, 1]$, the subdifferential of ReLU at the corner.

Section 35.3 takes subgradients to many variables. One warning belongs here already: a subgradient step need not be a descent step. At the minimum of $|x|$ the subgradient $g = \frac{1}{2}$ is valid, yet the step to $x = -\eta/2$ increases $|x|$. The subgradient method converges by shrinking its step sizes, not by descending at every step (Nesterov 2018).

34.1.6.3 Kinks in a Network

Each kink in a network sits in a convex piece ($|x|$, ReLU, the hinge). Autograd returns one fixed element of that piece's subdifferential, such as $\mathrm{ReLU}'(0) = 0$, and the chain rule carries that choice through the surrounding composition, which is far from convex. Unfortunately, chaining the per-kink choices can produce a number that is not a subgradient of the composite function at all. The simplest case is the identity written in a roundabout way,

$$\varphi(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x) = x \quad \text{for all } x,$$

whose only correct slope at 0 is 1. The chain rule gives $\varphi'(x) = \mathrm{ReLU}'(x) + \mathrm{ReLU}'(-x)$, which is 1 everywhere except at the kink, where the convention makes it $0 + 0 = 0$. The frameworks follow the convention.

```
# phi(x) = relu(x) - relu(-x) is the identity, so the true slope at 0 is 1
x = torch.tensor(0.0, requires_grad=True)
phi = torch.relu(x) - torch.relu(-x)
phi.backward()
print(f"autograd: phi'(0) = {x.grad.item():.1f}  (true slope: 1.0)")
```

```
autograd: phi'(0) = 0.0  (true slope: 1.0)
```

Autograd reports slope 0 for a function that is the identity.

Such a wrong value needs a pre-activation of exactly 0. When the data and the initialization are drawn from distributions with densities, this happens with probability zero at any fixed step. In floating point it does happen. Bertoin et al. (2021) measured how often, on a small fully connected network: backpropagation with $\mathrm{ReLU}'(0) = 0$ and with $\mathrm{ReLU}'(0) = 1$ returned different gradients on some minibatch for every initialization in 16-bit arithmetic, for a little under half of the initializations in 32-bit, and for none in 64-bit.

How much this matters is a separate question, and the answer is: very little. At 32 bits only a few minibatches in a million are affected, and on those the two gradients differ by a relative amount of about 10^{-3}. The authors at first reported a large effect on training. Their corrected experiments find none, in training loss or in test accuracy, for any value of $\mathrm{ReLU}'(0)$ between 0 and 1. Training suffers only for absurd choices such as $\mathrm{ReLU}'(0) = 10^4$, and batch normalization and Adam tame even those. The convention is a real source of numerical differences between two runs, then, and a harmless one. Exercises 7 and 12 reproduce the measurements at small scale.

All of this was for one weight. A network has millions of weights that move together. Which joint direction lowers the loss, and what curvature means with many variables, are the questions of Section 34.2.

34.1.7 Summary

The derivative is the slope of the linear approximation $f(x+\epsilon) = f(x) + \epsilon f'(x) + o(\epsilon)$, and the sum, product, and chain rules, proved from this approximation, differentiate any function built from a short table, for example $s' = \sigma$ for the softplus. The Mean Value Theorem turns slopes at points into behavior on intervals, and the Lagrange remainder bounds the error of the degree-n Taylor polynomial by a multiple of $|x - x_0|^{n+1}$ when $f^{(n+1)}$ is bounded. With $|f''| \le L$ on the interval of the step and $f'(x) \neq 0$, every gradient step with $0 < \eta < 2/L$ decreases f (the descent lemma). Newton's step $-f'/f''$ uses the actual curvature and squares the error near a minimizer with $f'' > 0$, but can diverge far from it, even on a strictly convex function. At a corner the derivative does not exist. For convex f the subdifferential replaces it and $0 \in \partial f(x)$ characterizes the global minimum; autograd returns one fixed element of it per kink. A chained value can fail to be a subgradient, which in floating point does occur and does not disturb training.

34.1.8 Exercises

1. **Quotient rule.** Write a quotient as a product, $\frac{g(x)}{h(x)} = g(x)\,(h(x))^{-1}$, on an interval where $h(x) \neq 0$.
   1. Derive the quotient rule of Section 34.1.2 from the product rule, the chain rule, and the power rule with $n = -1$.
   2. Use $\tanh x = 2\sigma(2x) - 1$ and the chain rule to show $\tanh' = 1 - \tanh^2$. Check both sides numerically at $x = 0.7$, where they equal 0.6347.
2. **Smoothness of the logistic loss.** The descent lemma Equation 34.6 requires a bound L on the curvature.
   1. Use the Mean Value Theorem Equation 34.3 to show that if $|f''| \le L$ on an interval, then f' is L-Lipschitz there. Conversely, show from the definition Equation 34.1 that a twice differentiable f with an L-Lipschitz f' satisfies $|f''| \le L$.
   2. Logistic regression with one weight w, inputs $a_i \in \mathbb{R}$, and labels $y_i \in \{-1, 1\}$ minimizes

$$\ell(w) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y_i a_i w}). \tag{34.11}$$

      Express ℓ'' in terms of σ. Find the smallest L for which ℓ' is L-Lipschitz on all of $\mathbb{R}$, and the step sizes for which Equation 34.6 guarantees a decrease from every starting point.
   3. Prove the descent lemma when f is only differentiable and f' is L-Lipschitz on the interval of the step. For $u = -\eta f'(x)$ let $\psi(t) = f(x+tu) - f(x) - tf'(x)u - \frac{L}{2}t^2u^2$. Show that $\psi(0) = 0$ and $\psi'(t) = u\,(f'(x+tu) - f'(x)) - Ltu^2 \le 0$ for $t \in [0, 1]$, and conclude $\psi(1) \le 0$ from the Mean Value Theorem. This is the form of the lemma that optimization uses for functions of many variables.
3. **Entropy extremes.** The binary entropy of a coin that shows heads with probability p is

$$H(p) = -p \log p - (1-p) \log(1-p), \qquad 0 < p < 1. \tag{34.12}$$

1. Find the minimum of $f(x) = x \log x$ on $x \geq 0$, where $f(0) = 0$ is the limiting value. Why does the minimum not lie at the boundary $x = 0$?
2. Show that $H'(p) = -\log \frac{p}{1-p}$, the negative logit of p. Locate the stationary point of H and classify it with the second-derivative test.
3. Find the limit of $H'(p)$ as $p \to 0$. Then parameterize the coin by its logit z, so that $p = \sigma(z)$, and compute $\frac{d}{dz} H(\sigma(z))$. In which of the two parameterizations is the slope of H Lipschitz on the whole domain, so that Equation 34.6 certifies one step size of gradient descent on $-H$ for every starting point?

4. **Newton at degenerate minima.** For $p > 1$, the function $f_p(x) = |x|^p/p$ is convex with its unique minimum at $x = 0$.
   1. Show that f_p is differentiable everywhere, including at 0, with $f_p'(x) = \operatorname{sign}(x)\,|x|^{p-1}$, and compute $f_p''(x)$ for $x \neq 0$. For which p does $f_p''(0)$ exist, and what is its value?
   2. Show that Newton's method Equation 34.7 maps every $x_t \neq 0$ to $x_{t+1} = \frac{p-2}{p-1}\, x_t$. For which p does the iteration converge? How fast does it converge for $p = 4$?
   3. Which hypothesis of the local convergence statement in Section 34.1.5.5 fails for $p = 4/3$, and which fails for $p = 4$?
   4. For $p = 4/3$, the damped update $x_{t+1} = x_t - \alpha f_p'(x_t)/f_p''(x_t)$ shortens the Newton step by a factor $\alpha > 0$. For which α does it converge from every x_0, and at what rate?

   Adapted from MIT 18.01SC, Session 33 problems, "Cube Root of x".

5. **Flat stationary points.** The second-derivative test is silent at a stationary point with $f''(x_0) = 0$, as x^3, x^4, and $-x^4$ show.
   1. Suppose that $f^{(k)}(x_0) = 0$ for $1 \leq k \leq n-1$, that $f^{(n)}(x_0) \neq 0$, and that $f^{(n)}$ is continuous near x_0. Use the Lagrange remainder Equation 34.5 of degree $n-1$ to show that x_0 is a local minimum if n is even and $f^{(n)}(x_0) > 0$, a local maximum if n is even and $f^{(n)}(x_0) < 0$, and neither if n is odd.
   2. Classify the stationary point 0 of $x - \sin x$ and of $\cos x - 1 + x^2/2$.
   3. Apply the test to the running function with its best parabola removed, $s(x) - \log 2 - \frac{x}{2} - \frac{x^2}{8}$, at $x_0 = 0$. Which derivative decides, and what is the answer? Check it by evaluating the function at $x = \pm 0.5$.

6. code **Minimizers at kinks.** A support vector machine with one weight x, weight decay $\lambda > 0$, and a single training example whose label times input equals 1 minimizes

$$F(x) = \max(0, 1-x) + \frac{\lambda}{2} x^2. \tag{34.13}$$

   1. Compute the subdifferential of the hinge loss $\max(0, 1-x)$ at every x from Equation 34.9.
   2. Take as given that adding a differentiable convex function q to a convex function h adds q' to every subgradient, $\partial(q+h)(x) = q'(x) + \partial h(x)$. Use the optimality condition Equation 34.10 to find the minimizer of F for every $\lambda > 0$. For which λ does it sit at the kink?

3. Take $\lambda = 1/2$ and run the subgradient method $x_{t+1} = x_t - \eta_t g_t$ from $x_0 = 0$ for 10^4 steps, where g_t is the value that autograd returns for $F'(x_t)$ when the hinge is written as $\mathrm{ReLU}(1 - x)$. Compare the constant steps $\eta_t = 0.1$ and $\eta_t = 0.01$ with the decaying steps $\eta_t = 0.5/\sqrt{t+1}$. How does the final distance to the minimizer depend on the schedule?
4. The kink $\{x = 1\}$ is a single point, which a draw from a density hits with probability zero. What do (b) and (c) imply for the argument in Section 34.1.6.3 that training rarely meets a kink?

7. code **Chained subgradients.** Autograd returns $\mathrm{ReLU}'(0) = 0$ and chains this value through every expression, as in Section 34.1.6.3.
   1. Let f_1 and f_2 be convex, $g_1 \in \partial f_1(x)$, $g_2 \in \partial f_2(x)$, and $c_1, c_2 \geq 0$. Show from Equation 34.9 that $c_1 g_1 + c_2 g_2 \in \partial(c_1 f_1 + c_2 f_2)(x)$. Why does this argument not cover $\mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$?
   2. Show that for every real c,

$$k_c(x) = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x) + c\,(x - \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)) = |x|. \tag{34.14}$$

   Predict the value that autograd returns for $k_c'(0)$, and check the prediction for $c = 0$, 1, and 5. For which c is the returned value a subgradient of $|x|$ at 0?
   3. Run gradient descent with the derivative from autograd on $\varphi(x) = \mathrm{ReLU}(x) - \mathrm{ReLU}(-x)$ for 10 steps from $x_0 = 0.5$, once with $\eta = 0.125$ and once with $\eta = 0.1$. What happens in each run? Why do the two step sizes behave differently? What does the difference imply for the probability-zero argument of Section 34.1.6.3 in floating-point arithmetic?

8. **Quartic descent.** ⊠ The minimum of $f(x) = x^4/4$ is flat, since $f''(0) = 0$. Gradient descent Equation 34.4 on f reads $x_{t+1} = x_t - \eta x_t^3$.
   1. Using problem 2a, find the smallest L for which f' is L-Lipschitz on $[-R, R]$. Which starting points x_0 does the descent lemma Equation 34.6 cover for a given η? Why does no fixed η cover all of them?
   2. Show that the iterates converge to 0 if $\eta x_0^2 < 2$ and diverge if $\eta x_0^2 > 2$. What happens at $\eta x_0^2 = 2$? Compare this threshold with the starting points covered in (a).
   3. Show that $1/x_{t+1}^2 = 1/x_t^2 + 2\eta + O(x_t^2)$, and conclude that $x_t^2 \approx 1/(x_0^{-2} + 2\eta t)$. Check the prediction for $\eta = 0.1$ and $x_0 = 1$ at $t = 10$, 100, and 1000. How does this rate compare with gradient descent on x^2, and with Newton's method on $x^4/4$ in problem 4b?

9. code **Central differences.** ⊠ The forward difference $D_+ f(x) = \frac{f(x+\epsilon) - f(x)}{\epsilon}$ and the central difference $D_0 f(x) = \frac{f(x+\epsilon) - f(x-\epsilon)}{2\epsilon}$ both approximate $f'(x)$.
   1. Use the Lagrange remainder Equation 34.5 to show $D_+ f(x) - f'(x) = \frac{1}{2} f''(\xi)\,\epsilon$ and $D_0 f(x) - f'(x) = \frac{1}{6} f'''(\xi')\,\epsilon^2$ for some ξ, ξ' near x, where f is three times continuously differentiable (the continuity of f''' merges two remainder points into one). Expand $f(x \pm \epsilon)$ to the needed order and subtract.

2. For $f(x) = x^2 + 1701(x-4)^3$ at $x = 4$, show that the two errors are $\epsilon + 1701\epsilon^2$ and $1701\epsilon^2$. At $\epsilon = 0.1$ they are 17.11 and 17.01, nearly equal. The central difference raises the order of the error, which helps only once ϵ is small. At which ϵ is the central error a hundred times smaller than the forward one?

In floating point, rounding puts a floor under ϵ. The gradient-check exercise of Section 34.3 balances it against the truncation error of (a).

10. code **Argument reduction for exp.** ▯ Libraries do not evaluate a Taylor polynomial of e^x at the input itself. They reduce the argument to a small interval and evaluate a polynomial with controlled error there. Write $x = k \log 2 + r$ with an integer k and $|r| \le \frac{1}{2}\log 2$, so that $e^x = 2^k e^r$ and only e^r needs a polynomial. Let P_n be the degree-n Taylor polynomial of e^x at $x_0 = 0$ (Section 34.1.5.2).

    1. Use the Lagrange remainder Equation 34.5 to find the smallest degree n for which $P_n(r)$ approximates e^r with relative error below $2^{-53} \approx 1.1 \times 10^{-16}$, the unit roundoff of float64, for every $|r| \le \frac{1}{2}\log 2$. Repeat the calculation for $P_n(x)$ on $|x| \le 10$ without reduction.
    2. Evaluate $P_n(-10)$ in float64 with the unreduced degree of (a), and compare the result with `math.exp(-10)`. Explain the size of the error from the largest term of the sum.
    3. Implement $e^x = 2^k P_n(r)$ with the reduced degree of (a), and compare it with `math.exp` on a grid over $[-20, 20]$. Which step limits the accuracy of the implementation for large $|x|$?

11. code **Gradient descent past $2/L$.** ▯▯ On $f(x) = x^2$, every step size $\eta > 2/L$ diverges. On other functions, gradient descent past this threshold can settle into an oscillation instead (Chen and Bruna 2023), a one-dimensional version of the edge of stability (Cohen et al. 2021). Let $x^\star$ be a local minimizer of a four times continuously differentiable f with $f''(x^\star) > 0$, and write $T(x) = x - \eta f'(x)$ for one step of Equation 34.4.

    1. Use Equation 34.3 to show that iterates that start close enough to $x^\star$ converge to it if $\eta < 2/f''(x^\star)$. Show that if $\eta > 2/f''(x^\star)$, every iterate in some neighborhood of $x^\star$, other than $x^\star$ itself, is followed by one farther from $x^\star$.

    2. A period-2 orbit is a pair $a \ne b$ with $T(a) = b$ and $T(b) = a$. Show that for every period-2 orbit there is a point ξ strictly between a and b with $f''(\xi) = 2/\eta$.

    3. Let $\eta = (1+\delta)\, 2/f''(x^\star)$ with a small $\delta > 0$. Expand $T(T(x))$ around $x^\star$ to third order, and show that a period-2 orbit near $x^\star$ exists and attracts nearby iterates if

$$3 f'''(x^\star)^2 - f''(x^\star)\, f''''(x^\star) > 0. \tag{34.15}$$

    Derive the half-width $|a-b|/2$ of the orbit to leading order in δ.

    4. Run gradient descent on the double well $f(x) = \frac{1}{4}(x^2-1)^2$ from $x_0 = 0.5$ for 5000 steps for each step size $\eta \in [0.9, 1.5]$, and plot the last 100 iterates against η. Compare the half-width of the orbit near $\eta = 1$ with (c). Compare the range of η with a stable period-2 orbit with the range $1 < \eta < \sqrt{4.5} - 1$ in which Chen and Bruna (2023) prove convergence to the orbit from almost every $0 < x_0 < 1$.

5. Predict from (c) what gradient descent with $\eta = 2.02$ does on $\log\cosh x$ and on $x^2/2 + x^4/4$, both of which have $x^\star = 0$ and $f''(x^\star) = 1$. Confirm the predictions by running both from $x_0 = 0.01$ for 500 steps.

12. [extended] $\mathrm{ReLU}'(0)$ **in low precision.** ⊠⊠ In exact arithmetic, the parameters at which backpropagation with $\mathrm{ReLU}'(0) = 0$ and with $\mathrm{ReLU}'(0) = 1$ disagree form a set of measure zero, which a draw from a density hits with probability zero. Bertoin et al. (2021) call this set the bifurcation zone and measure how often floating-point computations meet it. This project reproduces the frequency measurement at small scale on Fashion-MNIST (Section 3.2).

    1. Write the forward pass and the backward pass (Section 4.3) of a fully connected network with three hidden ReLU layers of width 256 and a softmax cross-entropy loss by hand in NumPy. Make the value g_0 used for $\mathrm{ReLU}'(0)$ and the floating-point type arguments. Check the gradient in float64 against finite differences.
    2. Draw 10 parameter vectors with weights uniform on $[-\sqrt{6/m}, \sqrt{6/m}]$ for a layer with m inputs, as in the paper. For each vector, compute the gradient on each of the 40 minibatches of 256 images among the first $10{,}240$ training images with $g_0 = 0$ and with $g_0 = 1$, in float16, float32, and float64. The two backward passes can differ only on a minibatch whose forward pass produces an exact zero pre-activation. Report the fraction of parameter vectors with at least one disagreeing minibatch, the fraction of disagreeing minibatches, and the median relative ℓ_2 difference of disagreeing gradients. Table 1 of the corrected version of the paper reports these quantities for MNIST and 1000 parameter vectors. In float16, float32, and float64, respectively, 100%, 43%, and 0% of the parameter vectors have a disagreeing minibatch, and 0.005%, 0.0002%, and 0% of the minibatches disagree. The median relative difference is 2×10^{-3} in both float16 and float32. Compare your fractions with these.
    3. Repeat the float16 measurement with matrix products that accumulate in float32 and round only their results to float16. Explain the difference from (b).
    4. Count the exact zeros among the pre-activations in each setting. Suppose that the rate of exact zeros is proportional to the unit roundoff u, which is 2^{-11}, 2^{-24}, and 2^{-53} for the three formats. Predict the float32 and float64 counts from the float16 count, and compare.
    5. Train the network with minibatch SGD for one epoch in float32 twice, from the same initialization and with the same minibatch order, once with $g_0 = 0$ and once with $g_0 = 1$. Plot the ℓ_1 distance between the two parameter vectors against the iteration. Repeat with $g_0 = 10^4$. Compare with what Section 34.1.6.3 reports about the influence of $\mathrm{ReLU}'(0)$ on training.

34.2 Multivariable Calculus

The previous section studied a loss as a function of one weight, with every other weight held fixed. Training never moves one weight at a time. Each step of gradient descent moves all the weights a little at once, so it needs to know how the loss responds to a joint change, and which joint change lowers it fastest. A million separate slopes, one per weight, do not obviously answer either question.

They do, under one hypothesis. If the loss is differentiable, the vector of slopes, the *gradient*, predicts the effect of every joint move at once, to first order, and points in the direction of

fastest increase. A symmetric table of second derivatives, the *Hessian*, describes how the surface bends and tells a minimum from a saddle. This section is about what the two objects say. How a program computes a gradient is a separate question, and Section 34.3 takes it up.

Section 1.4 stated the rules of differentiation and used the gradient. Here we prove when the linear model holds and how second derivatives classify a critical point, as Section 34.1 did for functions of one variable. Two results carry the section: the linear model and the second-derivative test. Tangent planes, and the last part on Lagrange multipliers, can be skipped on a first reading. Our running example is the two-variable log-sum-exp $f(x, y) = \log(e^x + e^y)$ at the point $(0, \log 2)$. Its gradient will turn out to be the softmax.

34.2.1 Partial Derivatives and the Gradient

34.2.1.1 Partial Derivatives

Freeze every weight except w_1, and the loss becomes a function of one variable. Its derivative is the *partial derivative* $\partial \ell / \partial w_1$ of Section 1.4: the slope of the *slice* of the graph of ℓ along which only w_1 varies. As in one variable, it predicts the effect of a small change ϵ_1 of w_1 alone,

$$\ell(w_1 + \epsilon_1, w_2, \dots, w_N) \approx \ell(w_1, w_2, \dots, w_N) + \epsilon_1 \frac{\partial \ell}{\partial w_1}(w_1, w_2, \dots, w_N). \tag{34.16}$$

For the running example, holding y fixed and differentiating $f(x, y) = \log(e^x + e^y)$ in x gives $\partial f / \partial x = e^x / (e^x + e^y)$, and by symmetry $\partial f / \partial y = e^y / (e^x + e^y)$. At $(0, \log 2)$ we have $e^x = 1$ and $e^y = 2$, so $f = \log 3 \approx 1.0986$, $\partial f / \partial x = \frac{1}{3}$, and $\partial f / \partial y = \frac{2}{3}$.

34.2.1.2 Differentiability and the Gradient

The partial derivatives give one slope per coordinate. They do not by themselves say what happens when all coordinates move at once, say when we step from $(0, \log 2)$ by $\epsilon = (0.01, -0.03)$. It is natural to guess that the two changes add, $\frac{1}{3} \cdot 0.01 + \frac{2}{3} \cdot (-0.03)$, but this guess is a claim about f, and it needs a hypothesis.

The hypothesis is that, near $\mathbf{w}$, ℓ is a linear function up to an error smaller than the step. We call ℓ *differentiable* at $\mathbf{w}$ if there is a linear map Λ with

$$\ell(\mathbf{w} + \epsilon) = \ell(\mathbf{w}) + \Lambda(\epsilon) + o(\|\epsilon\|), \tag{34.17}$$

where $o(\|\epsilon\|)$ stands for any error that, divided by $\|\epsilon\|$, tends to zero as $\epsilon \to \mathbf{0}$.

If such a Λ exists, it is determined by the partial derivatives. Setting $\epsilon = h\mathbf{e}_i$ in Equation 34.17, dividing by h, and letting $h \to 0$ identifies $\Lambda(\mathbf{e}_i) = \partial \ell / \partial w_i$. Linearity then fixes Λ on every step,

$$\Lambda(\epsilon) = \Lambda\Big(\sum_{i=1}^{N} \epsilon_i \mathbf{e}_i\Big) = \sum_{i=1}^{N} \epsilon_i \frac{\partial \ell}{\partial w_i} = \epsilon \cdot \nabla_{\mathbf{w}} \ell, \qquad \nabla_{\mathbf{w}} \ell = \left[\frac{\partial \ell}{\partial w_1}, \dots, \frac{\partial \ell}{\partial w_N}\right]^{\top}.$$

The column vector $\nabla_{\mathbf{w}}\ell$ of partial derivatives is the *gradient* of ℓ. Substituting it into the definition gives the multivariable linear approximation

$$\ell(\mathbf{w}+\epsilon) = \ell(\mathbf{w}) + \epsilon \cdot \nabla_{\mathbf{w}}\ell(\mathbf{w}) + o(\|\epsilon\|). \tag{34.18}$$

This has the shape of the one-variable $f(x+\epsilon) \approx f(x) + \epsilon f'(x)$, with the slope replaced by a vector and the product by a dot product. The gradient is the unique vector whose dot product with a step gives the first-order change in ℓ. Of the ways to say what a gradient is, this is the one to remember. We treat the gradient as a column vector throughout.

At $(0, \log 2)$ the running example has $\nabla f = [\frac{1}{3}, \frac{2}{3}]^\top$, and Equation 34.18 predicts $f(0.01, \log 2 - 0.03) \approx \log 3 + \frac{1}{3}\cdot 0.01 - \frac{2}{3}\cdot 0.03 \approx 1.081946$. At every point the two entries of the gradient are positive and sum to one, because the gradient of log-sum-exp is the softmax $\mathbf{p} = \operatorname{softmax}(x, y)$, with entries $e^x/(e^x+e^y)$ and $e^y/(e^x+e^y)$.

Having all partial derivatives does not make a function differentiable. Take $f(x,y) = 2xy/(x^2+y^2)$ with $f(0,0) = 0$. On both axes f is zero, so both partials at the origin are 0, and the only candidate linear model is the constant 0. Yet $f = 1$ everywhere on the diagonal $x = y \neq 0$, however close to the origin, so no linear model holds there. This f is not even continuous at the origin, and Exercise 1 gives a continuous function with the same defect.

A convenient sufficient condition for differentiability at $\mathbf{w}$ is that the partial derivatives exist near $\mathbf{w}$ and are continuous at $\mathbf{w}$ (Rudin 1976). The partials of log-sum-exp are continuous everywhere, so it is differentiable everywhere.

The definition promises an error that is $o(\|\epsilon\|)$ and nothing more. Before running the next cell, guess by what factor the error of the linear model falls when the step $\epsilon = (0.01, -0.03)$ is made ten times smaller. The code is plain NumPy, the same in every framework.

```
import numpy as np
import torch
```

```
def lse(xy):  # f(x, y) = log(e^x + e^y), computed stably
    return np.logaddexp(xy[0], xy[1])

def grad_lse(xy):  # its gradient, the softmax of (x, y)
    return np.exp(xy - lse(xy))

xy0 = np.array([0.0, np.log(2)])
print('gradient:', grad_lse(xy0))
for scale in [1, 0.1, 0.01]:
    eps = scale * np.array([0.01, -0.03])
    linear = lse(xy0) + eps @ grad_lse(xy0)
    true = lse(xy0 + eps)
    print(f'|eps| {np.linalg.norm(eps):.1e}: linear {linear:.8f}, '
          f'true {true:.8f}, gap {true - linear:.3e}')
```

```
gradient: [0.33333333 0.66666667]
|eps| 3.2e-02: linear 1.08194562, true 1.08212418, gap 1.786e-04
|eps| 3.2e-03: linear 1.09694562, true 1.09694740, gap 1.779e-06
|eps| 3.2e-04: linear 1.09844562, true 1.09844564, gap 1.778e-08
```

At $\|\epsilon\| \approx 0.03$ the linear model is off by 1.79×10^{-4}, in the fourth decimal of f, and each tenfold smaller step makes the gap a hundredfold smaller. The error is of second order in the step, which is better than the definition requires. The Hessian will predict its size (Section 34.2.3.2).

34.2.1.3 Directional Derivatives

The gradient also gives the slope along any direction, not only along the axes. Put $\epsilon = h\mathbf{u}$ with a unit vector $\mathbf{u}$ into Equation 34.18, divide by h, and let $h \to 0$. The error term disappears, and we get

$$\lim_{h \to 0} \frac{\ell(\mathbf{w} + h\mathbf{u}) - \ell(\mathbf{w})}{h} = \mathbf{u} \cdot \nabla_{\mathbf{w}} \ell(\mathbf{w}). \tag{34.19}$$

The left side is the *directional derivative* of ℓ along $\mathbf{u}$, and the right side says that it is the projection of the gradient onto $\mathbf{u}$. One vector stores the slope in every direction.

Three directions of the running example at $(0, \log 2)$ show the range. Along the diagonal $\mathbf{u} = (1,1)/\sqrt{2}$ the slope is $(\tfrac{1}{3} + \tfrac{2}{3})/\sqrt{2} = 1/\sqrt{2} \approx 0.707$, and it is the same at *every* point: $f(x + t, y + t) = \log(e^t(e^x + e^y)) = f(x, y) + t$, so moving a distance $\sqrt{2}\,t$ along the diagonal raises f by t. Along $\mathbf{u} = -(1,2)/\sqrt{5}$, against the gradient, the slope is $-(\tfrac{1}{3} + \tfrac{4}{3})/\sqrt{5} = -\sqrt{5}/3 \approx -0.745$. Along $\mathbf{u} = (2,-1)/\sqrt{5}$ it is $(\tfrac{2}{3} - \tfrac{2}{3})/\sqrt{5} = 0$, so to first order f does not change in this direction.

34.2.2 The Geometry of Gradients

34.2.2.1 Steepest Descent

Gradient descent (Section 2.1) moves the parameters a short way in the direction along which ℓ falls fastest. By Equation 34.19 this is the unit vector $\mathbf{v}$ with the most negative projection $\mathbf{v} \cdot \nabla\ell$. Section 1.4 found it from the angle between $\mathbf{v}$ and the gradient. The Cauchy–Schwarz inequality also shows that it is unique.

Proposition (steepest ascent and descent). *At any point where $\nabla\ell \neq \mathbf{0}$, among all unit vectors $\mathbf{v}$ the directional derivative $\mathbf{v} \cdot \nabla\ell$ is largest for $\mathbf{v} = \nabla\ell/\|\nabla\ell\|$ and smallest for $\mathbf{v} = -\nabla\ell/\|\nabla\ell\|$. Thus $+\nabla\ell$ is the direction of steepest ascent and $-\nabla\ell$ the direction of steepest descent.*

Proof. The Cauchy–Schwarz inequality (Section 33.1) states that $|\mathbf{v} \cdot \mathbf{g}| \le \|\mathbf{v}\|\|\mathbf{g}\|$, with equality only when $\mathbf{v}$ and $\mathbf{g}$ are collinear. For a unit vector $\mathbf{v}$ and $\mathbf{g} = \nabla\ell$ it reads

$$-\|\nabla\ell\| \;\le\; \mathbf{v} \cdot \nabla\ell \;\le\; \|\nabla\ell\|.$$

Each bound is attained only by a unit vector collinear with $\nabla\ell$: the upper bound by $\mathbf{v} = \nabla\ell/\|\nabla\ell\|$ and the lower bound by $\mathbf{v} = -\nabla\ell/\|\nabla\ell\|$. $\blacksquare$

In the running example the steepest descent direction at $(0, \log 2)$ is $-(1,2)/\sqrt{5}$, with slope $-\|\nabla f\| = -\sqrt{5}/3 \approx -0.745$, the most negative of the three slopes computed above. Stepping along $-\nabla_{\mathbf{w}}\ell$ gives the gradient-descent update

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\, \nabla_{\mathbf{w}} \ell(\mathbf{w}).$$

Note that the update does not normalize the direction. Its step has length $\eta\|\nabla\ell\|$, which shrinks as the gradient does.

"Unit" in the proposition means Euclidean length, and this is a choice. If instead each coordinate may move by at most one, the steepest unit step is $-\operatorname{sign}(\nabla\ell)$, coordinate by coordinate; at $(0, \log 2)$ that is $(-1,-1)$, with slope $-(\frac{1}{3}+\frac{2}{3}) = -1$. Adam, which divides each coordinate of its step by an estimate of that coordinate's size (Chapter 9), resembles this sign step more than the Euclidean one. Exercise 5 develops steepest descent in other norms.

34.2.2.2 Gradients and Level Sets

The gradient also has a geometric meaning that does not refer to a step. A *level set* of ℓ is the set of points where ℓ takes a fixed value c. Moving along a level set does not change ℓ, so the slope along such a direction should vanish.

Proposition (gradients are normal to level sets). *Let ℓ be differentiable at $\mathbf{x}$. Then $\nabla\ell(\mathbf{x})$ is orthogonal to every direction tangent at $\mathbf{x}$ to the level set $\{\mathbf{x}' : \ell(\mathbf{x}') = c\}$ through $\mathbf{x}$.*

Proof. A direction $\mathbf{v}$ is *tangent* to the level set at $\mathbf{x}$ if $\mathbf{v} = \gamma'(0)$ for some differentiable curve $\gamma(t)$ lying in the level set with $\gamma(0) = \mathbf{x}$. Along such a curve $\ell(\gamma(t)) = c$ for all t, so the derivative of $t \mapsto \ell(\gamma(t))$ at $t = 0$ vanishes. To first order $\gamma(t) = \mathbf{x} + t\mathbf{v} + o(t)$, so by Equation 34.18 that derivative is $\mathbf{v}\cdot\nabla\ell(\mathbf{x})$. Hence $\mathbf{v}\cdot\nabla\ell(\mathbf{x}) = 0$. ■

The level set of the running example through $(0, \log 2)$ is the curve $e^x + e^y = 3$. Differentiating along it, $e^x\,dx + e^y\,dy = 0$, so its tangent direction is $(e^y, -e^x) = (2,-1)$, and indeed $(2,-1)\cdot(\frac{1}{3},\frac{2}{3}) = 0$. This is the direction of zero slope found above.

On a contour map the gradient crosses each contour at a right angle and points toward larger values. If neighboring contours differ in value by Δc and lie a distance d apart, the slope across them is about $\Delta c/d$, so the gradient is longest where the contours are packed most tightly (Figure 34.12).

34.2.2.3 Tangent Planes

Read without its error term as an equation for a height z, the linear model Equation 34.18 is the *tangent plane* to the graph of f at $\mathbf{x}_0$,

$$z = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)\cdot(\mathbf{x} - \mathbf{x}_0), \tag{34.20}$$

which for the running example is $z = \log 3 + \frac{1}{3}x + \frac{2}{3}(y - \log 2)$. Its normal in $(\mathbf{x}, z)$-space is $[\nabla f(\mathbf{x}_0), -1]^\top$, and dropping the height coordinate leaves $\nabla f(\mathbf{x}_0)$, the normal to the level curve in the base plane (Figure 34.13).

34.2.2.4 Critical Points

Losses in deep learning are too complex to minimize in closed form. The linear model still gives a condition that every minimum must satisfy, and gradient descent stops moving exactly where this condition holds. Throughout, f is defined on all of $\mathbb{R}^n$ and differentiable at the point $\mathbf{x}_0$ in question.

Suppose $\nabla f(\mathbf{x}_0) \neq \mathbf{0}$ and step against it, by $\epsilon = -\eta\nabla f(\mathbf{x}_0)$ with $\eta > 0$. By Equation 34.18 the change is $-\eta\|\nabla f(\mathbf{x}_0)\|^2 + o(\eta)$, which is negative for every sufficiently small η. So $\mathbf{x}_0$ is

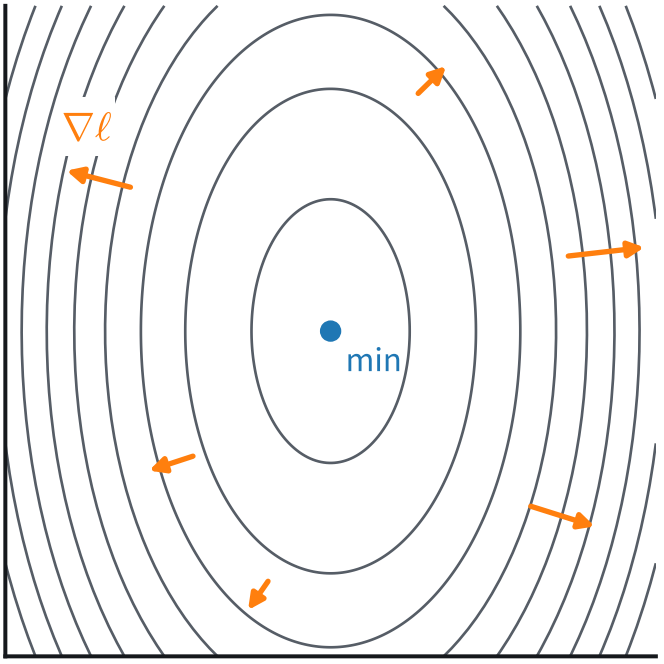


Figure 34.12: Contours of a loss ℓ (gray) with $\nabla \ell$ at several points. Each gradient is normal to its contour, points toward larger values, and is longest where the contours are packed most tightly.

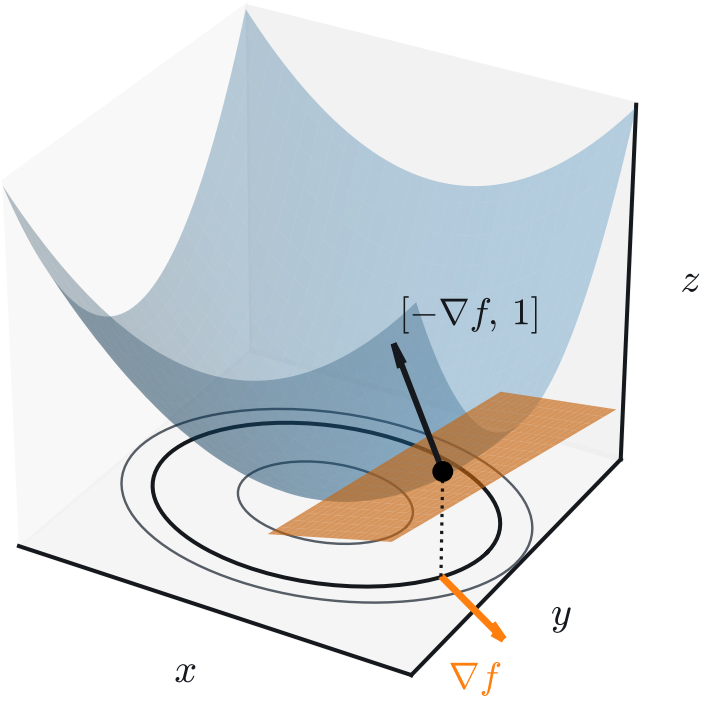


Figure 34.13: A surface $z = f(\mathbf{x})$ (blue) with its tangent plane at a point (orange). The vector $[-\nabla f, 1]$, the negative of $[\nabla f, -1]$, is normal to the plane (black arrow). Dropping the height coordinate leaves a multiple of ∇f, which on the base plane (orange arrow) is normal to the level curve through the point (black).

not a local minimum, and by the same argument with $+\eta$ it is not a local maximum either. Hence **at a local minimum or maximum, $\nabla f(\mathbf{x}_0) = \mathbf{0}$.** Points where the gradient vanishes are *critical points*. The condition is necessary, not sufficient, and the next example shows what else an argument by hand needs.

Take $\psi(x, y) = xe^{-x^2-y^2}$, a positive hill for $x > 0$ and a negative pit for $x < 0$. Its partial derivatives are

$$\frac{\partial \psi}{\partial x} = (1 - 2x^2)\, e^{-x^2-y^2}, \qquad \frac{\partial \psi}{\partial y} = -2xy\, e^{-x^2-y^2}.$$

Both vanish only if $y = 0$ (the second forces $x = 0$ or $y = 0$, and $x = 0$ makes the first equal to $e^{-y^2} \neq 0$) and $x = \pm 1/\sqrt{2}$. The two critical points have values $\psi(\mp 1/\sqrt{2}, 0) = \mp e^{-1/2}/\sqrt{2} \approx \mp 0.4289$. To conclude which is the global minimum we also need to know that a global minimum *exists*. Outside the disc of radius $R = 2$, $|\psi| \leq Re^{-R^2} < 0.04$, while inside it $\psi(-1/\sqrt{2}, 0) = -0.43$. A continuous function attains its minimum on the closed disc, and this minimum is below -0.04, so it lies in the interior and is a global minimum of ψ, hence a critical point. Of the two candidates, $(-1/\sqrt{2}, 0)$ has the smaller value: it is the global minimum, and by the same argument $(+1/\sqrt{2}, 0)$ is the global maximum. Without the existence step the argument fails: x^3 has a single critical point and no minimum at all.

The first-order condition cannot tell the minimum from the maximum, since both have zero gradient. Second derivatives do (Section 34.2.3.3).

34.2.3 The Hessian

34.2.3.1 Symmetry of the Hessian

The linear model is first order. Whether a critical point is a minimum, a maximum, or neither depends on how the surface bends, and the bending is measured by second derivatives. A function of n variables has n^2 second partial derivatives

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right),$$

collected into the *Hessian* matrix

$$\mathbf{H}_f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}. \tag{34.21}$$

Entry (i, j) is the rate at which the j-th slope changes as x_i moves. The Hessian is also written $\nabla^2 f$; when the function and the point are clear we abbreviate it to $\mathbf{H}$.

For log-sum-exp, differentiate the softmax entries $p_i = e^{x_i}/(e^{x_1} + e^{x_2})$ once more (here $(x_1, x_2) = (x, y)$). The quotient rule gives $\partial p_i/\partial x_j = p_i\delta_{ij} - p_i p_j$, where δ_{ij} is 1 for $i = j$ and 0 otherwise, so

$$\mathbf{H}_f = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}, \qquad \mathbf{H}_f(0, \log 2) = \begin{bmatrix} \frac{1}{3} - \frac{1}{9} & -\frac{2}{9} \\ -\frac{2}{9} & \frac{2}{3} - \frac{4}{9} \end{bmatrix} = \frac{2}{9}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{34.22}$$

The matrix has eigenvalue 0 along $(1,1)$ and $\frac{4}{9}$ along $(1,-1)$. The zero eigenvalue is the diagonal direction along which $f(x+t, y+t) = f(x,y) + t$ is linear and does not bend at all. This positive semidefinite, singular Hessian returns in the second-derivative test.

The n^2 entries are not independent. For twice continuously differentiable f the Hessian is symmetric (the Clairaut–Schwarz theorem (Rudin 1976)), and symmetry is what lets the spectral theorem and the definiteness criteria of Section 33.3 apply to it in the second-derivative test.

Proposition (symmetry of the Hessian; Clairaut–Schwarz). *Suppose the first partial derivatives of f and the mixed partials $\frac{\partial^2 f}{\partial x_i \partial x_j}$ and $\frac{\partial^2 f}{\partial x_j \partial x_i}$ exist near $\mathbf{x}$ and that the mixed partials are continuous at $\mathbf{x}$. Then*

$$\frac{\partial^2 f}{\partial x_i \partial x_j}(\mathbf{x}) = \frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x}).$$

If this holds for all i, j, then $\mathbf{H}_f = \mathbf{H}_f^{\top}$.

Proof. Both mixed partials are limits of the *same* quantity, the symmetric second difference

$$\Delta_h = \frac{f(\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x} + h\mathbf{e}_j) + f(\mathbf{x})}{h^2}, \tag{34.23}$$

which is symmetric in i and j by inspection. The work is to show that Δ_h equals a mixed partial at a nearby point. Write $\varphi(t) = f(\mathbf{x} + t\mathbf{e}_i + h\mathbf{e}_j) - f(\mathbf{x} + t\mathbf{e}_i)$, so that $h^2 \Delta_h = \varphi(h) - \varphi(0)$. Apply the mean value theorem Equation 34.3 twice: first to φ, and then along x_j to the difference of first partials that φ' is. With τ and σ between 0 and h,

$$\varphi(h) - \varphi(0) = h\,\varphi'(\tau) = h\Big[\frac{\partial f}{\partial x_i}(\mathbf{x} + \tau\mathbf{e}_i + h\mathbf{e}_j) - \frac{\partial f}{\partial x_i}(\mathbf{x} + \tau\mathbf{e}_i)\Big] = h^2\,\frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x} + \tau\mathbf{e}_i + \sigma\mathbf{e}_j).$$

Hence $\Delta_h = \frac{\partial^2 f}{\partial x_j \partial x_i}(\xi_h)$ at a point ξ_h of the square with corners $\mathbf{x}$ and $\mathbf{x} + h\mathbf{e}_i + h\mathbf{e}_j$, within distance $\sqrt{2}\,|h|$ of $\mathbf{x}$. Swapping the roles of i and j leaves Δ_h unchanged and gives $\Delta_h = \frac{\partial^2 f}{\partial x_i \partial x_j}(\xi_h')$ at another point of the same square. Let $h \to 0$: by continuity at $\mathbf{x}$ the two expressions converge to the two mixed partials at $\mathbf{x}$, and since they are equal for every h, so are the limits. ∎

Exercise 5 shows the hypothesis at work: a function whose mixed partials exist everywhere but are not continuous at the origin, where they differ.

34.2.3.2 The Second-Order Taylor Approximation

The gradient gives the best linear model of f; the Hessian gives the best *quadratic* model. To see how the coefficients enter, read them off an exact quadratic. Let $f(x_1, x_2) = a + b_1 x_1 + b_2 x_2 + c_{11} x_1^2 + c_{12} x_1 x_2 + c_{22} x_2^2$. Evaluating the value, gradient, and Hessian Equation 34.21 at the origin gives

$$f(0,0) = a, \qquad \nabla f(0,0) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \qquad \mathbf{H}_f(0,0) = \begin{bmatrix} 2c_{11} & c_{12} \\ c_{12} & 2c_{22} \end{bmatrix},$$

and these recover the polynomial: $f(\mathbf{x}) = f(\mathbf{0}) + \nabla f(\mathbf{0}) \cdot \mathbf{x} + \frac{1}{2}\mathbf{x}^\top \mathbf{H}_f(\mathbf{0})\,\mathbf{x}$. The factor $\frac{1}{2}$ undoes the 2 that differentiating a square produces.

For a general f the same assembly is no longer exact, and restricting f to a line shows how close it comes. Let f be twice continuously differentiable near the segment from $\mathbf{x}_0$ to $\mathbf{x}$, fix the displacement $\delta = \mathbf{x} - \mathbf{x}_0$, and let $\varphi(t) = f(\mathbf{x}_0 + t\delta)$, a function of the single variable t. Its derivative is the rate of change of f along δ, which the linear model Equation 34.18 gives as $\varphi'(t) = \sum_i \frac{\partial f}{\partial x_i}(\mathbf{x}_0 + t\delta)\,\delta_i$. Each partial derivative in this sum is itself differentiable, and the same step applied to it gives $\varphi''(t) = \sum_{i,j} \delta_j \frac{\partial^2 f}{\partial x_j \partial x_i}(\mathbf{x}_0 + t\delta)\,\delta_i = \delta^\top \mathbf{H}_f(\mathbf{x}_0 + t\delta)\,\delta$.

The Lagrange remainder Equation 34.5 of degree one, applied to φ between 0 and 1, gives $\varphi(1) = \varphi(0) + \varphi'(0) + \frac{1}{2}\varphi''(\tau)$ for some $\tau \in (0,1)$. Translated back to f,

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot \delta + \tfrac{1}{2}\delta^\top \mathbf{H}_f(\xi)\,\delta, \qquad \xi = \mathbf{x}_0 + \tau\delta.$$

This identity is exact, with the Hessian evaluated at an intermediate point ξ of the segment instead of at $\mathbf{x}_0$. Since the second partials are continuous, moving the evaluation point back to $\mathbf{x}_0$ changes the quadratic term by at most $\frac{1}{2}\|\mathbf{H}_f(\xi) - \mathbf{H}_f(\mathbf{x}_0)\|\,\|\delta\|^2 = o(\|\delta\|^2)$, which gives the *second-order Taylor approximation*

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0)^\top \mathbf{H}_f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + o(\|\mathbf{x} - \mathbf{x}_0\|^2). \tag{34.24}$$

The quadratic model is wrong only by an amount that vanishes faster than the squared step, and it is the only quadratic with this property. When f has three continuous derivatives, the error is of third order in the step.

Along a single direction, a step $t\mathbf{v}$ makes the quadratic term $\frac{t^2}{2}\mathbf{v}^\top \mathbf{H}\mathbf{v}$, and $\mathbf{v}^\top \mathbf{H}\mathbf{v} = \frac{d^2}{dt^2} f(\mathbf{x}_0 + t\mathbf{v})\big|_{t=0}$ is the *second directional derivative*, the curvature of the slice of f along $\mathbf{v}$.

The Hessian accounts for the gap that the linear model left in the first code cell. For the step $\epsilon = (0.01, -0.03)$, the Hessian Equation 34.22 predicts a gap between the true value and the linear model of $\frac{1}{2}\epsilon^\top \mathbf{H}\epsilon = \frac{1}{9}(\epsilon_1 - \epsilon_2)^2 = \frac{1}{9}(0.04)^2 \approx 1.78 \times 10^{-4}$, against the 1.786×10^{-4} that was measured.

```
p = grad_lse(xy0)               # the softmax at (0, log 2)
H = np.diag(p) - np.outer(p, p)  # Hessian of log-sum-exp
print('9 H =', (9 * H).round(6).tolist())
```

```
print('eigenvalues:', np.linalg.eigvalsh(H).round(6) + 0.0)  # no '-0.'
for scale in [1, 0.1]:
    eps = scale * np.array([0.01, -0.03])
    print(f'|eps| {np.linalg.norm(eps):.1e}: '
          f'predicted gap {0.5 * eps @ H @ eps:.4e}')
```

```
9 H = [[2.0, -2.0], [-2.0, 2.0]]
eigenvalues: [0.       0.444444]
|eps| 3.2e-02: predicted gap 1.7778e-04
|eps| 3.2e-03: predicted gap 1.7778e-06
```

The matrix and its eigenvalues 0 and $\frac{4}{9}$ are those of Equation 34.22. The predicted gaps, 1.7778×10^{-4} and 1.7778×10^{-6}, match the measured 1.786×10^{-4} and 1.779×10^{-6} to two or three digits. The remainder of about 10^{-6} in the first gap is the third-order term, and for the tenfold smaller step it falls about a thousandfold, below the printed digits.

In the log-sum-exp example one direction did not bend at all. In the next example both do. For $\psi(x, y) = xe^{-x^2-y^2}$ at $\mathbf{x}_0 = (-1, 0)$, the formulas for the partial derivatives above give $\psi = -e^{-1}$ and $\nabla\psi = (-e^{-1}, 0)$, and differentiating once more gives $\mathbf{H}_\psi = 2e^{-1}\mathbf{I}$. The quadratic model Equation 34.24 is therefore $q(x, y) = e^{-1}(-1 - (x + 1) + (x + 1)^2 + y^2)$. Figure 34.14 draws the surface against this quadratic: near $(-1, 0)$ they are nearly indistinguishable, and farther out they separate.

If the error of the quadratic model is of third order, doubling the step should multiply the gap by eight. The next cell evaluates ψ and q at steps of d in each coordinate from $(-1, 0)$.

```
def psi(x, y):
    return x * np.exp(-x**2 - y**2)

def quad(x, y):  # second-order Taylor model of psi at (-1, 0)
    return np.exp(-1) * (-1 - (x + 1) + (x + 1)**2 + y**2)

for d in [0.0, 0.05, 0.1, 0.3]:  # step d in each coordinate from (-1, 0)
    x, y = -1 + d, d
    print(f'step {d:.2f}: psi = {psi(x, y):.6f}, quadratic = {quad(x, y):.6f}, '
          f'gap = {abs(psi(x, y) - quad(x, y)):.6f}')
```

```
step 0.00: psi = -0.367879, quadratic = -0.367879, gap = 0.000000
step 0.05: psi = -0.384315, quadratic = -0.384434, gap = 0.000119
step 0.10: psi = -0.396388, quadratic = -0.397310, gap = 0.000921
step 0.30: psi = -0.391929, quadratic = -0.412025, gap = 0.020096
```

The gap vanishes at the base point, and doubling the step from 0.05 to 0.1 multiplies it by a little less than eight (0.000119 to 0.000921), as a third-order error should. Repeatedly fitting this quadratic and jumping to *its* minimum is Newton's method, which Section 34.1.5.5 introduced in one variable and Section 9.2.3 develops as an optimizer.

34.2.3.3 The Second-Derivative Test

At a critical point $\mathbf{x}_0$ the gradient term of Equation 34.24 vanishes, and near $\mathbf{x}_0$ the function is its quadratic term up to $o(\|\mathbf{x}-\mathbf{x}_0\|^2)$. Whether f rises or falls as we leave $\mathbf{x}_0$ is then decided

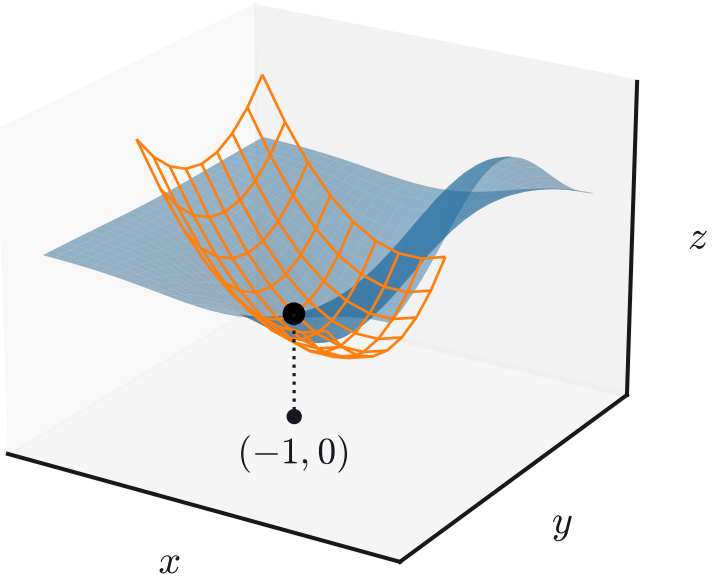


Figure 34.14: The surface $z = xe^{-x^2-y^2}$ (blue) and its second-order Taylor quadratic at the base point $(-1, 0)$ (orange). The two agree in value, slope, and curvature there, so they are nearly indistinguishable near the base point and separate farther out.

by the sign of $\mathbf{v}^\top \mathbf{H}\mathbf{v}$ in each direction $\mathbf{v}$, as long as this sign is not zero. The eigenvalues of $\mathbf{H}$ organize these signs. A symmetric $\mathbf{H}$ has orthonormal eigenvectors $\mathbf{q}_i$ with real eigenvalues λ_i, and in the eigenvector basis its quadratic form is a weighted sum of squares (the spectral theorem, Section 33.3):

$$\mathbf{v}^\top \mathbf{H}\mathbf{v} = \sum_{i=1}^{n} \lambda_i \, (\mathbf{q}_i^\top \mathbf{v})^2. \tag{34.25}$$

If every $\lambda_i > 0$, the curvature is positive in every direction, and at least $\lambda_{\min}\|\mathbf{v}\|^2$. If the eigenvalues have both signs, it is positive along some directions and negative along others.

Proposition (second-order conditions). *Let f be twice continuously differentiable near a critical point $\mathbf{x}_0$, and let $\mathbf{H} = \mathbf{H}_f(\mathbf{x}_0)$.*

1. *If $\mathbf{x}_0$ is a local minimum, then $\mathbf{H} \succeq 0$ (all eigenvalues ≥ 0).*
2. *If $\mathbf{H} \succ 0$ (all eigenvalues > 0), then $\mathbf{x}_0$ is a strict local minimum.*
3. *If $\mathbf{H}$ has eigenvalues of both signs, then $\mathbf{x}_0$ is neither a local minimum nor a local maximum, whatever the zero eigenvalues of $\mathbf{H}$: it is a saddle point.*

The statements for maxima follow by applying these to $-f$. A singular semidefinite $\mathbf{H}$ falls under neither 2 nor 3, and the test is then inconclusive.

Proof. (1) Suppose $\mathbf{v}^\top \mathbf{H}\mathbf{v} < 0$ for some $\mathbf{v}$. The slice $\varphi(t) = f(\mathbf{x}_0 + t\mathbf{v})$ has $\varphi'(0) = \nabla f(\mathbf{x}_0) \cdot \mathbf{v} = 0$ and $\varphi''(0) = \mathbf{v}^\top \mathbf{H}\mathbf{v} < 0$, so by the one-variable test it has a strict local maximum at $t = 0$, and f takes smaller values than $f(\mathbf{x}_0)$ arbitrarily close to $\mathbf{x}_0$. This contradicts the minimum.

(2) This is the step that needs care, because the error term must be controlled in all directions at once. By the exact form above with $\nabla f(\mathbf{x}_0) = \mathbf{0}$, $f(\mathbf{x}_0 + \delta) - f(\mathbf{x}_0) = \frac{1}{2}\delta^\top \mathbf{H}_f(\xi)\delta$ with ξ on the segment. Split $\mathbf{H}_f(\xi) = \mathbf{H} + (\mathbf{H}_f(\xi) - \mathbf{H})$. By Equation 34.25, $\delta^\top \mathbf{H}\delta \geq \lambda_{\min}\|\delta\|^2$, and by continuity of the second partials $\|\mathbf{H}_f(\xi) - \mathbf{H}\| \leq \frac{1}{2}\lambda_{\min}$ for all sufficiently small δ. Hence $f(\mathbf{x}_0 + \delta) - f(\mathbf{x}_0) \geq \frac{1}{2}(\lambda_{\min} - \frac{1}{2}\lambda_{\min})\|\delta\|^2 = \frac{1}{4}\lambda_{\min}\|\delta\|^2 > 0$ for small $\delta \neq \mathbf{0}$.

(3) Along an eigenvector with $\lambda > 0$ the slice φ has $\varphi'(0) = 0$ and $\varphi''(0) = \lambda > 0$, a strict local minimum, so f takes larger values than $f(\mathbf{x}_0)$ arbitrarily close to $\mathbf{x}_0$. Along an eigenvector with $\lambda < 0$ it takes smaller values. ■

Of the three statements, the second is the one to remember, and the only one whose proof takes work. The inconclusive case does occur. Along a zero eigenvector the quadratic model is flat, and higher-order terms decide: $x^2 + y^4$ and $x^2 - y^4$ both have Hessian $\operatorname{diag}(2, 0)$ at the origin, where the first has a strict minimum and the second a saddle (Exercise 4 goes further). Figure 34.15 shows the four cases on the quadratic models of this section's examples.

Panel (a) is the Hessian of ψ at $(-1/\sqrt{2}, 0)$, which the formulas for its partial derivatives give as $e^{-1/2}\operatorname{diag}(2\sqrt{2}, \sqrt{2})$; panel (b) is the Hessian at $(+1/\sqrt{2}, 0)$, its negative. Panel (c) is the origin of $x^4 + y^4 - 4xy$, with Hessian $\left(\begin{smallmatrix} 0 & -4 \\ -4 & 0 \end{smallmatrix}\right)$, and panel (d) is the log-sum-exp Hessian Equation 34.22, used here only as a matrix: log-sum-exp itself has no critical point, since its gradient, the softmax, never vanishes.

What does a numerical version of the test report at a flat direction? A Hessian assembled from second differences never contains an exact zero, so the next cell needs a tolerance

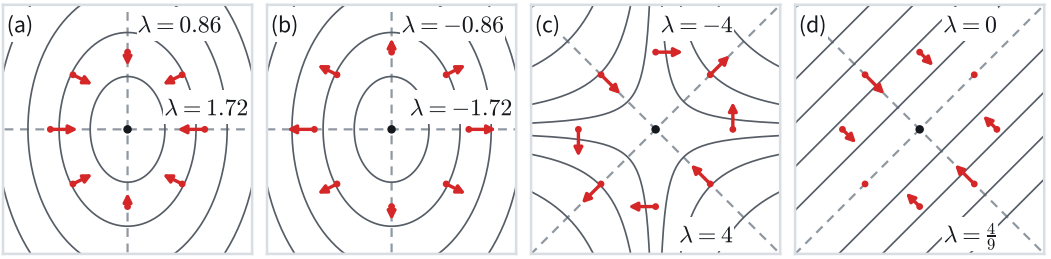


Figure 34.15: Contours of the quadratic model $\frac{1}{2}\mathbf{v}^\top \mathbf{H}\mathbf{v}$ (gray), its eigenvector directions with their eigenvalues (dashed), and descent arrows $-\mathbf{H}\mathbf{v}$ (red). (a) $\mathbf{H} \succ 0$: a minimum, every descent arrow points in. (b) $\mathbf{H} \prec 0$: a maximum. (c) Indefinite: a saddle; descent flows in along one eigenvector and out along the other. (d) Singular and semidefinite: the quadratic model is flat along the zero eigenvector, and second order is inconclusive.

below which an eigenvalue counts as zero. Its centered mixed difference `fxy` is the second difference Equation 34.23 with step $2h$ based at $(x-h, y-h)$.

```python
def hessian(f, x, y, h=1e-4):  # 2x2 Hessian from second differences
    fxx = (f(x + h, y) - 2 * f(x, y) + f(x - h, y)) / h**2
    fyy = (f(x, y + h) - 2 * f(x, y) + f(x, y - h)) / h**2
    fxy = (f(x + h, y + h) - f(x + h, y - h)
           - f(x - h, y + h) + f(x - h, y - h)) / (4 * h**2)
    return np.array([[fxx, fxy], [fxy, fyy]])

def classify(lo, hi, tol=1e-6):  # from the smallest, largest eigenvalue
    if lo > tol:
        return 'minimum'
    if hi < -tol:
        return 'maximum'
    if lo < -tol < tol < hi:
        return 'saddle'
    return 'inconclusive'

quartic = lambda x, y: x**4 + y**4 - 4 * x * y
trough = lambda x, y: x**2 + y**4
for fn, x0 in [(psi, -1 / np.sqrt(2)), (psi, 1 / np.sqrt(2)),
               (quartic, 0.0), (trough, 0.0)]:
    lo, hi = np.linalg.eigvalsh(hessian(fn, x0, 0.0))
    print(f'({x0:+.4f}, 0): eigenvalues {lo:+.3f}, {hi:+.3f} '
          f'-> {classify(lo, hi)}')
```

```
(-0.7071, 0): eigenvalues +0.858, +1.716 -> minimum
(+0.7071, 0): eigenvalues -1.716, -0.858 -> maximum
(+0.0000, 0): eigenvalues -4.000, +4.000 -> saddle
(+0.0000, 0): eigenvalues +0.000, +2.000 -> inconclusive
```

At $(-1/\sqrt{2}, 0)$ the eigenvalues are 0.858 and 1.716 ($\sqrt{2}e^{-1/2}$ and $2\sqrt{2}e^{-1/2}$): the global minimum found earlier. At $(+1/\sqrt{2}, 0)$ they are the negatives, a maximum. The origin of $x^4 + y^4 - 4xy$ has eigenvalues ± 4, a saddle whose directions of rise and fall, $(1, -1)$ and $(1, 1)$, are not the coordinate axes. The fourth point is the instructive one. At the origin

of $x^2 + y^4$ the finite difference returns $2h^2 = 2 \times 10^{-8}$ for the flat direction, below the tolerance, and the verdict is "inconclusive" although the origin is a strict minimum. Second-order information cannot decide this point, and only the tolerance makes the program say so. Exercise 3 classifies a saddle by hand.

34.2.3.4 Saddles in High Dimension

A local minimum needs all n eigenvalues of the Hessian to be nonnegative, and a nondegenerate one needs all of them positive. If the signs at a random critical point behaved like independent coin flips, all would be positive with probability 2^{-n}, which is tiny for large n. This heuristic *overestimates* the chance of a minimum. Consider Gaussian random symmetric matrices whose diagonal entries have twice the variance of the off-diagonal ones, the Gaussian orthogonal ensemble. For them the exact probability at $n = 2$ is $(1-1/\sqrt{2})/2 \approx 0.146$ rather than 0.25 (Exercise 6). For large n it decays like $\exp(-\frac{\ln 3}{4} n^2)$ (Dean and Majumdar 2006), because eigenvalues repel one another and rarely all sit on one side of zero.

For random landscapes, then, the critical points at typical (high) values of the function are almost all saddles. At low values the Hessian at a critical point is shifted toward positive eigenvalues, and minima dominate (Bray and Dean 2007). Neural networks show the same pattern: among the critical points that Newton's method finds on the loss of small networks, those of high loss are saddles (Dauphin et al. 2014). An optimizer that stalls at high loss is therefore more likely to be near a saddle than near a bad local minimum.

34.2.4 Constrained Optimization and Lagrange Multipliers

So far the variables were free to move in every direction. Many problems restrict them to a surface: weights normalized to unit length, probabilities that sum to one, the point on a curve closest to the origin. We then minimize $f(\mathbf{x})$ subject to $g(\mathbf{x}) = c$. The condition $\nabla f = \mathbf{0}$ no longer applies, because at the best point ∇f need not vanish. It only has to give no first-order decrease along moves that stay on the surface. What replaces it follows from the geometry of level sets, with no second derivatives.

Let f and g be differentiable. At a feasible point the admissible moves are the directions $\mathbf{v}$ tangent to the surface $\{g = c\}$. At a constrained local minimizer $\mathbf{x}^\star$ no admissible move lowers f to first order, so $\mathbf{v} \cdot \nabla f(\mathbf{x}^\star) \geq 0$ for every tangent $\mathbf{v}$. Since $-\mathbf{v}$ is tangent too, equality holds, and $\nabla f(\mathbf{x}^\star)$ is orthogonal to every tangent direction. By the level-set proposition, so is $\nabla g(\mathbf{x}^\star)$.

Proposition (Lagrange condition). *Let $\mathbf{x}^\star$ be a local minimizer of f on $\{\mathbf{x} : g(\mathbf{x}) = c\}$. Assume that f is differentiable at $\mathbf{x}^\star$, that g is continuously differentiable near $\mathbf{x}^\star$, and that $\nabla g(\mathbf{x}^\star) \neq \mathbf{0}$. Then there is a scalar μ, the Lagrange multiplier, with*

$$\nabla f(\mathbf{x}^\star) = \mu \, \nabla g(\mathbf{x}^\star). \tag{34.26}$$

Proof. The proof uses one fact about the constraint set. Under the last two hypotheses, $\{g = c\}$ is a smooth surface near $\mathbf{x}^\star$, and every $\mathbf{v}$ with $\nabla g(\mathbf{x}^\star) \cdot \mathbf{v} = 0$ is the velocity $\gamma'(0)$ of a differentiable curve γ in $\{g = c\}$ with $\gamma(0) = \mathbf{x}^\star$ (Rudin 1976). Along such a curve, $t \mapsto f(\gamma(t))$ has a local minimum at $t = 0$, so its derivative there vanishes. As in the level-set proof, that derivative is $\mathbf{v} \cdot \nabla f(\mathbf{x}^\star)$. Hence $\nabla f(\mathbf{x}^\star)$ is orthogonal to the hyperplane of all $\mathbf{v}$ orthogonal to $\nabla g(\mathbf{x}^\star)$. The vectors orthogonal to that hyperplane are the multiples of $\nabla g(\mathbf{x}^\star)$, so $\nabla f(\mathbf{x}^\star) = \mu \nabla g(\mathbf{x}^\star)$. $\blacksquare$

At the solution the gradients of f and g are parallel, so the level set of f through $\mathbf{x}^\star$ is tangent to the constraint surface. Together with $g(\mathbf{x}) = c$, the n equations of the condition give $n+1$ equations for the $n+1$ unknowns $\mathbf{x}$ and μ.

As a worked example, find the points of the hyperbola $xy = 1$ closest to the origin: minimize $f = x^2 + y^2$ subject to $g = xy = 1$. The gradients are $\nabla f = (2x, 2y)$ and $\nabla g = (y, x)$, and $\nabla g \neq \mathbf{0}$ on the hyperbola. The condition reads $2x = \mu y$ and $2y = \mu x$. Substituting the second into the first gives $4x = \mu^2 x$, so $x(4 - \mu^2) = 0$, and $x = 0$ is impossible on $xy = 1$. The multiplier $\mu = -2$ forces $y = -x$ and $xy = -x^2 = 1$, which has no solution. So $\mu = 2$, $x = y$, and the candidates are $(1, 1)$ and $(-1, -1)$, both with $f = 2$. A minimum exists, since on each branch $f = x^2 + 1/x^2$ grows without bound as $x \to 0$ or $|x| \to \infty$, so these are the closest points. At $(1, 1)$ the circle $x^2 + y^2 = 2$ touches the hyperbola, and $\nabla f = (2, 2) = 2\,\nabla g$ (Figure 34.16).

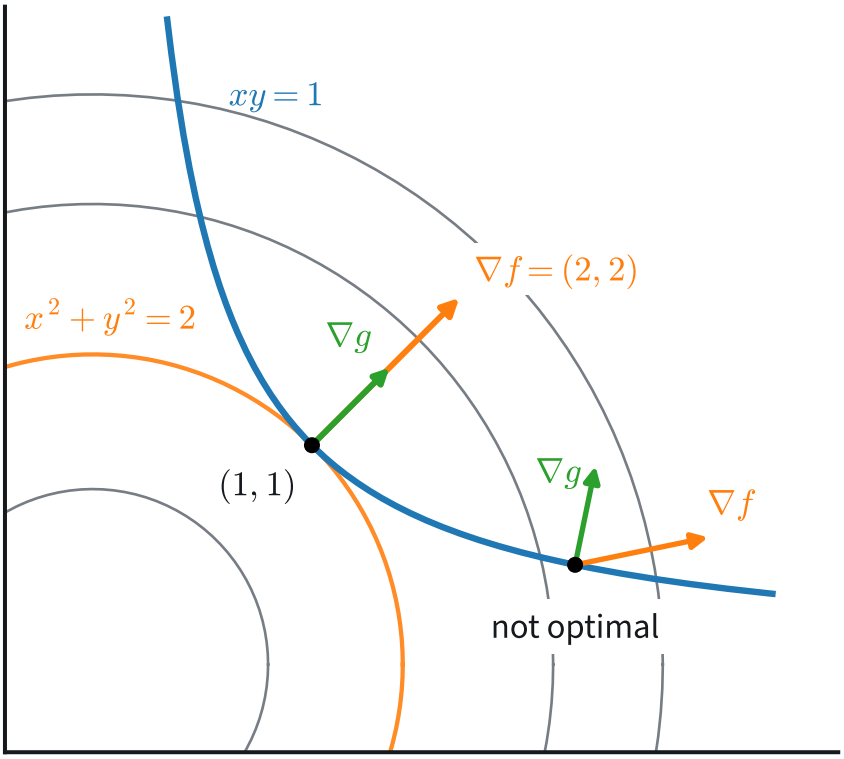


Figure 34.16: The closest point of $xy = 1$ to the origin. At $(1, 1)$ the circle $x^2 + y^2 = 2$ is tangent to the hyperbola and $\nabla f = (2, 2) = 2\nabla g$, so $\mu = 2$; the mirror solution $(-1, -1)$ is off the frame. At a feasible point that is not optimal the two gradients point in different directions, and ∇f keeps a component along the constraint.

The hyperbola was chosen for its picture. Closer to this book is the problem of maximizing $\mathbf{z} \cdot \mathbf{p} - \sum_i p_i \log p_i$ over probability vectors $\mathbf{p}$. With the constraint $\sum_i p_i = 1$ the Lagrange condition reads $z_i - \log p_i - 1 = \mu$, so $p_i \propto e^{z_i}$: the solution is the softmax, and the value attained is the log-sum-exp Equation 35.87.

Without the hypothesis $\nabla g(\mathbf{x}^\star) \neq \mathbf{0}$ a multiplier need not exist (Exercise 8 gives an example). Section 35.6 builds the KKT conditions on this condition, and Section 35.7 the dual problem.

Every statement in this section took the gradient as given. For a formula in two variables that is fair. For a network with a million weights, obtaining the gradient is the hard part.

34.2.5 Summary

A differentiable function changes, to first order, by the dot product of the step with its gradient; partial derivatives that merely exist do not guarantee this. The negative gradient is the direction of steepest descent, and the gradient is normal to the level sets. The symmetric Hessian supplies the quadratic model. At a critical point the signs of its eigenvalues tell a minimum, a maximum, and a saddle apart, and a singular semidefinite Hessian decides nothing. On a constraint surface the geometry of level sets gives the Lagrange condition $\nabla f = \mu \nabla g$.

34.2.6 Exercises

1. **Directional derivatives without a gradient.** Let $f(x, y) = x^2 y/(x^2 + y^2)$ for $(x, y) \neq (0, 0)$ and $f(0, 0) = 0$.
    1. Show that f is continuous at the origin.
    2. Compute the two partial derivatives of f at the origin and the directional derivative there along every unit vector $\mathbf{u} = [\cos\theta, \sin\theta]^\top$.
    3. Show that no vector $\mathbf{g}$ gives these directional derivatives as $\mathbf{u} \cdot \mathbf{g}$, so that f is not differentiable at the origin. Which hypothesis of the sufficient condition for differentiability in Section 34.2.1.2 fails? What does this function show that the counterexample $2xy/(x^2 + y^2)$ there does not?

2. **Gradients of an elongated bowl.** Let $f(x, y) = x^2 + \kappa y^2$ with $\kappa \geq 1$. Its minimizer is the origin, and κ measures how much more strongly f curves along y than along x.
    1. Find a direction tangent to the level curve of f through $(1, 1)$, check that it is orthogonal to $\nabla f(1, 1)$, and compute the angle between $-\nabla f(1, 1)$ and the direction $-[1, 1]^\top$ from $(1, 1)$ to the minimizer. At which points $(x, y) \neq (0, 0)$ does the negative gradient point at the minimizer?
    2. Find the smallest value of the cosine of this angle over all points $(x, y) \neq (0, 0)$, and the points where it is attained. (Hint: set $\rho = y/x$ and minimize over ρ.) Evaluate it for $\kappa = 10$. What does the result imply for a gradient-descent step on such a loss as κ grows?

3. **Classifying critical points.** Let $f(x, y) = x^3 - 3x + y^2$. Show that the critical points are $(\pm 1, 0)$. Determine whether each is a minimum, a maximum, or a saddle by examining f along the lines $y = 0$ and $x = \pm 1$, and confirm the classification with the eigenvalues of the Hessian.

4. **Flat directions.**
    1. Give three functions of (x, y) that have a critical point at the origin with Hessian $\mathrm{diag}(2, 0)$ there: one with a strict local minimum, one with a local minimum that is not strict, and one without a local minimum.
    2. The function $f(x, y) = (x - y^2)(x - 2y^2)$ also has Hessian $\mathrm{diag}(2, 0)$ at the origin. Show that its restriction to every line through the origin has a strict local minimum there. Sketch the region of the plane where $f < 0$, and show that the origin is nevertheless not a local minimum of f. What does this imply for testing a critical point by probing the loss along a few directions?

3. The *monkey saddle* $f(x, y) = x^3 - 3xy^2$ has a critical point at the origin. Compute its Hessian there, write f in polar coordinates, and find the directions along which f decreases. How does $\|\nabla f\|$ scale with the distance to the origin, compared with the saddle $x^2 - y^2$? What does this imply for gradient descent started near each of the two saddles?

5. **Mixed partials that disagree.** ▣ Second-order methods treat the Hessian as a symmetric matrix. Let $f(x, y) = xy(x^2 - y^2)/(x^2 + y^2)$ for $(x, y) \neq (0, 0)$ and $f(0, 0) = 0$.

   1. Compute $\frac{\partial f}{\partial x}(0, y)$ and $\frac{\partial f}{\partial y}(x, 0)$, and from them the two mixed partials at the origin in the notation of Equation 34.21.
   2. Evaluate the symmetric second difference Equation 34.23 at the origin for this f. Which step of the proof of the Clairaut–Schwarz theorem fails?
   3. What does the function `hessian` of this section return as the off-diagonal entry at the origin? What does a symmetric finite-difference formula miss here?

6. code **Random Hessians.** ▣ A local minimum requires all n eigenvalues of the Hessian to be nonnegative, and a nondegenerate one requires them all positive; the section's coin-flip heuristic puts the latter at 2^{-n}. Test this heuristic on random symmetric matrices $\mathbf{A} = (\mathbf{G} + \mathbf{G}^\top)/\sqrt{2}$, where the $n \times n$ matrix $\mathbf{G}$ has independent standard normal entries.

   1. For $n = 2$, show that $s = (a_{11} + a_{22})/2$, $d = (a_{11} - a_{22})/2$, and a_{12} are independent standard normal variables, and that $\mathbf{A} \succ 0$ exactly when $s > \sqrt{d^2 + a_{12}^2}$. Use the rotational symmetry of the standard normal distribution in three dimensions to compute $P(\mathbf{A} \succ 0)$ exactly, and compare it with 2^{-2}.
   2. Estimate $P(\mathbf{A} \succ 0)$ by Monte Carlo for $n = 1, \dots, 6$, with 10^6 draws for $n \leq 5$ and 10^7 draws, generated in batches, for $n = 6$. Plot $\log P$ against n together with $\log 2^{-n}$, and fit a quadratic in n to your estimates. Dean and Majumdar (2006) prove that $\log P \approx -\frac{\ln 3}{4} n^2$ for large n. Compare this coefficient with the quadratic coefficient of your fit. Does the heuristic over- or underestimate the probability of a minimum, and which property of the eigenvalues does it ignore?
   3. For large n the smallest eigenvalue of $\mathbf{A}$ lies close to $-2\sqrt{n}$. For $n = 5, 10, 20$, find the shift β at which half of the matrices $\mathbf{A} + \beta\mathbf{I}$ are positive definite, and compare it with $2\sqrt{n}$. In the random landscapes of Bray and Dean (2007), the Hessian at a critical point behaves like such a shifted matrix, with a shift that grows as the value of the random function at the critical point decreases. What does your result suggest about critical points of low loss?

7. [extended] **Saddles of a linear network.** ▣▣ Let $\mathbf{A}$ be a symmetric $n \times n$ matrix with eigenvalues $\lambda_1 > \lambda_2 > \cdots > \lambda_n > 0$ and orthonormal eigenvectors $\mathbf{q}_1, \dots, \mathbf{q}_n$. A linear network with one hidden unit, input weights $\mathbf{v}$, and output weights $\mathbf{u}$ maps $\mathbf{x}$ to $\mathbf{u}\mathbf{v}^\top\mathbf{x}$. Trained on whitened inputs ($\mathbb{E}[\mathbf{x}\mathbf{x}^\top] = \mathbf{I}$) with targets $\mathbf{A}\mathbf{x}$, half its expected squared error is

$$\ell(\mathbf{u}, \mathbf{v}) = \tfrac{1}{2}\|\mathbf{A} - \mathbf{u}\mathbf{v}^\top\|_F^2.$$

   Baldi and Hornik (1989) showed that the local minima of such linear networks are global. This problem derives the complete set of critical points and compares it with the relation between loss and index that Dauphin et al. (2014) measured for nonlinear networks; the *index* of a critical point is the fraction of negative eigenvalues of its

Hessian.

1. Compute $\nabla_{\mathbf{u}}\ell$ and $\nabla_{\mathbf{v}}\ell$. Show that the critical points are the origin and the points $\mathbf{u} = \alpha\mathbf{q}_i$, $\mathbf{v} = (\lambda_i/\alpha)\mathbf{q}_i$ with $\alpha \neq 0$, and compute ℓ at each of them.
2. Compute the eigenvalues of the $2n \times 2n$ Hessian at the balanced point $\mathbf{u} = \mathbf{v} = \sqrt{\lambda_i}\,\mathbf{q}_i$ and at the origin. (In the eigenvector basis, the Hessian splits into 2×2 blocks, one for each pair of coordinates along $\mathbf{q}_j$ in $\mathbf{u}$ and in $\mathbf{v}$.) How many eigenvalues are negative at each critical point? Explain the zero eigenvalue by a symmetry of ℓ, and conclude that every local minimum is global.
3. Take $n = 4$, $(\lambda_1, \dots, \lambda_4) = (4, 3, 2, 1)$, and a random orthonormal basis of eigenvectors. Run Newton's method from many random starting points of different scales, and record the loss and the index of every critical point it reaches. The Hessian is singular at every critical point other than the origin, so compute each Newton step by least squares. Plot the loss against the index, as in Figure 1 of Dauphin et al. (2014) . How often does Newton's method end at each kind of critical point, and where does gradient descent from small random initializations end?
4. Replace $\mathbf{u}\mathbf{v}^\top$ by $\mathbf{U}\mathbf{V}^\top$ with two hidden units, so that $\mathbf{U}$ and $\mathbf{V}$ are 4×2 matrices. Predict the loss values of the critical points from the rank-one analysis, and confirm them with Newton's method. How many zero eigenvalues does the Hessian have at the critical points, and which symmetry explains them? Is the index a monotone function of the loss?

8. **Lagrange multipliers and the Rayleigh quotient.** Let $\mathbf{A}$ be a symmetric $n \times n$ matrix. Maximize $f(\mathbf{x}) = \mathbf{x}^\top\mathbf{A}\mathbf{x}$ subject to $g(\mathbf{x}) = \|\mathbf{x}\|^2 = 1$.

   1. Write the Lagrange condition and show that it reads $\mathbf{A}\mathbf{x} = \mu\mathbf{x}$: every constrained critical point is a unit eigenvector, and its multiplier is the eigenvalue.
   2. Show that $f(\mathbf{x}) = \mu$ at every constrained critical point. Why does a constrained maximum exist? Conclude that the maximum is $\lambda_{\max}(\mathbf{A})$ and the minimum $\lambda_{\min}(\mathbf{A})$.
   3. For $\mathbf{A} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ find the maximizers and minimizers and their values, and check at a maximizer that the level set of f through it is tangent to the unit circle.
   4. Which hypothesis of the Lagrange proposition does $g(\mathbf{x}) = \|\mathbf{x}\|^2$ satisfy at every point of the unit sphere? Now minimize the linear function $\mathbf{c}^\top\mathbf{x}$, $\mathbf{c} \neq \mathbf{0}$, subject to $\|\mathbf{x}\|^2 = 0$. What is the minimizer, and why does no multiplier μ satisfy $\nabla f = \mu\nabla g$ there?

34.3 The Chain Rule on Computational Graphs

Every training step in this book began with one call, `backward()` or `grad`, and that one call returned the slope of the loss with respect to every weight. Section 34.2 said what these slopes mean: together they form the gradient, the vector that predicts the effect of any small joint move. It did not say how a program finds a million of them for the price of a few evaluations of the network. The obvious method, nudging one weight and running the network again, needs one run per weight.

The chain rule does much better, and the reason can be stated before any algebra. A network is a long composition of simple functions, and the derivatives with respect to its many weights share most of their work, since every one of them has to pass through the last layer. A formula for the composition hides this sharing. A graph shows it: draw a node for each

intermediate value and an edge for each direct dependence. On the graph the chain rule is a sum over paths, and the shared work becomes a choice of how to group the sum. If we group the paths from the output backward, one pass yields the slope with respect to every input. This grouping is backpropagation.

We work the idea out on graphs of scalars that are small enough to compute by hand, with integer values throughout, so that every claim can be checked. The section needs one fact from Section 34.2: near a point, a differentiable function changes by the dot product of the step with its gradient, the linear model Equation 34.18. Layers that map vectors to vectors, and the cost of the method on a real network, come in the two sections that follow.

34.3.1 The Chain Rule as a Sum over Paths

34.3.1.1 A Graph of Scalars

Our first example is a toy: four inputs w, x, y, z flow through intermediate quantities to a scalar output,

$$\begin{aligned} f(u,v) &= (u+v)^2, \\ u(a,b) &= (a+b)^2, \qquad v(a,b) = (a-b)^2, \\ a(w,x,y,z) &= (w+x+y+z)^2, \qquad b(w,x,y,z) = (w+x-y-z)^2. \end{aligned} \tag{34.27}$$

The dependencies form a graph (Figure 34.17): each node is a value, and each edge a direct functional dependence. Nothing about the squares is special; they keep the numbers whole. Throughout the section f denotes the output of the graph at hand.

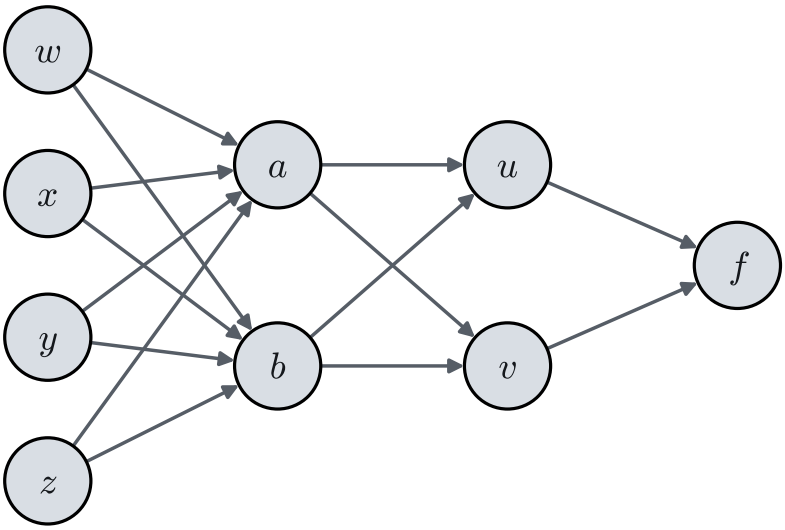


Figure 34.17: The function relations of Equation 34.27, drawn as a graph whose nodes are values and whose edges show direct functional dependence.

34.3.1.2 Two Paths through One Step

Take the last composite step, $f(u(a,b), v(a,b))$, and perturb a by a small ϵ. Each intermediate moves by its partial derivative, $u \to u + \epsilon \frac{\partial u}{\partial a}$ and $v \to v + \epsilon \frac{\partial v}{\partial a}$. Feeding these moves into the first-order expansion Equation 34.18 of f gives

$$f\big(u + \epsilon \tfrac{\partial u}{\partial a},\, v + \epsilon \tfrac{\partial v}{\partial a}\big) \approx f(u,v) + \epsilon \left[\frac{\partial f}{\partial u}\frac{\partial u}{\partial a} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial a} \right].$$

The coefficient of ϵ is the derivative of the composite with respect to a, and the result is the multivariable chain rule,

$$\frac{\mathrm{d}f}{\mathrm{d}a} = \frac{\partial f}{\partial u}\frac{\partial u}{\partial a} + \frac{\partial f}{\partial v}\frac{\partial v}{\partial a}.$$

The two kinds of d are deliberate, and this chapter keeps them apart from here on. The symbol ∂ is reserved for the derivative of one node's function with respect to one of its arguments, the others held fixed. The *total* derivative $\mathrm{d}f/\mathrm{d}a$ lets everything downstream of a respond. (For an input, the total is the ordinary partial derivative of f regarded as a function of the inputs alone.) There are two *paths* by which a influences f, $a \to u \to f$ and $a \to v \to f$. Each path contributes the product of the derivatives along its edges, and the total is the sum over the paths.

The argument is the one-variable proof of the chain rule again (Section 34.1.2): feed the change of the inner functions into the linear model of the outer one. Note, however, that it feeds two simultaneous moves into one linear expansion. That needs f to be *differentiable*, not merely to have partial derivatives. For the counterexample $2xy/(x^2+y^2)$ of Section 34.2.1.2 along the line $x = y = t$, the rule would predict derivative 0 at $t = 0$ from the two zero partials, yet the function jumps from 0 to 1 there.

34.3.1.3 Edge Derivatives and Total Derivatives

The rule extends to any graph. Call the *edge derivative* of a node with respect to one of its parents the partial derivative of the node's function with respect to that parent, holding its other parents fixed. Then for an output f and an input x,

$$\frac{\mathrm{d}f}{\mathrm{d}x} = \sum_{\text{directed paths } x \to \cdots \to f} \; \prod_{\text{edges on the path}} (\text{edge derivative}). \tag{34.28}$$

Each path multiplies the local rates along its edges, and the paths add. A graph in which one node feeds the output both directly and through a later node shows why edge derivatives must be kept apart from totals. Consider

$$\begin{aligned} p(x,y) &= x + y, \qquad q(x,y) = x - y, \\ r(p) &= p^2, \qquad s(q) = q^2, \\ f(r,p,s) &= r + p + s. \end{aligned} \tag{34.29}$$

The output f depends on r, on s, and also directly on p, a skip edge (Figure 34.18). The input y reaches f by three paths, $y \to p \to r \to f$, $y \to p \to f$, and $y \to q \to s \to f$. The edge

derivatives are $\partial f/\partial r = \partial f/\partial p = \partial f/\partial s = 1$, $\partial r/\partial p = 2p$, $\partial s/\partial q = 2q$, $\partial p/\partial y = 1$, and $\partial q/\partial y = -1$, so

$$\frac{\mathrm{d}f}{\mathrm{d}y} = \underbrace{1 \cdot 2p \cdot 1}_{y \to p \to r \to f} + \underbrace{1 \cdot 1}_{y \to p \to f} + \underbrace{1 \cdot 2q \cdot (-1)}_{y \to q \to s \to f} = 2p + 1 - 2q = 1 + 4y.$$

Direct substitution, $f = (x+y)^2 + (x+y) + (x-y)^2$, confirms the sum $1 + 4y$. At $(x, y) = (1, 2)$, $p = 3$ and $q = -1$, and the paths contribute $6 + 1 + 2 = 9 = 1 + 4 \cdot 2$. The *edge* derivative $\partial f/\partial p$ is 1, but the *total* derivative $\mathrm{d}f/\mathrm{d}p$, through the skip edge and through r, is $2p + 1 = 7$. Confusing the two is the usual way to get the multivariable chain rule wrong.

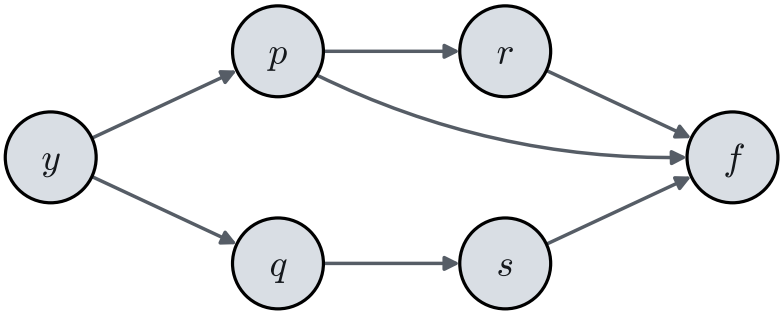


Figure 34.18: The dependency graph of Equation 34.29: the input y reaches f by three paths, $y \to p \to r \to f$, the skip $y \to p \to f$, and $y \to q \to s \to f$.

The sum over paths is how gradients flow through a network. Architectural choices that open or close paths, such as the gates of an LSTM (Section 12.1) or the skip connections of a residual block (Section 7.4), shape learning by controlling which paths carry gradient.

34.3.2 Backpropagation

As a formula the sum over paths is complete: it gives every derivative of every graph. As an algorithm it is hopeless, because the number of paths grows quickly with depth. A chain of m stages with two parallel nodes per stage already has 2^m paths from end to end. Luckily the paths share edges, and a product that many paths share needs to be computed only once. The way we group the shared products decides the cost.

34.3.2.1 Forward Mode

We compare two groupings on the four-input function Equation 34.27 at $(w, x, y, z) = (1, 1, -2, 1)$. A *forward pass* evaluates the nodes in order: $a = 1^2 = 1$, $b = 3^2 = 9$, $u = 10^2 = 100$, $v = (-8)^2 = 64$, and $f = 164^2 = 26896$. The edge derivatives at these values are

$$\frac{\partial f}{\partial u} = \frac{\partial f}{\partial v} = 2(u+v) = 328,$$
$$\frac{\partial u}{\partial a} = \frac{\partial u}{\partial b} = 2(a+b) = 20, \qquad \frac{\partial v}{\partial a} = 2(a-b) = -16, \qquad \frac{\partial v}{\partial b} = -2(a-b)$$
$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial x} = \frac{\partial a}{\partial y} = \frac{\partial a}{\partial z} = 2(w+x+y+z) = 2,$$
$$\frac{\partial b}{\partial w} = \frac{\partial b}{\partial x} = 2(w+x-y-z) = 6, \qquad \frac{\partial b}{\partial y} = \frac{\partial b}{\partial z} = -6.$$

Every edge derivative is a number we can compute locally, from the values of a node and its parents.

The four paths from w to f go through a or b and then through u or v. Pushing w forward through the graph collects them as

$$\begin{aligned}\frac{\mathrm{d}f}{\mathrm{d}w} &= \frac{\partial f}{\partial u}\left(\frac{\partial u}{\partial a}\frac{\partial a}{\partial w} + \frac{\partial u}{\partial b}\frac{\partial b}{\partial w}\right) + \frac{\partial f}{\partial v}\left(\frac{\partial v}{\partial a}\frac{\partial a}{\partial w} + \frac{\partial v}{\partial b}\frac{\partial b}{\partial w}\right) \\ &= 328\,(20 \cdot 2 + 20 \cdot 6) + 328\,(-16 \cdot 2 + 16 \cdot 6) = 328 \cdot 224 = 73472.\end{aligned}$$

The bracketed sums are $\mathrm{d}u/\mathrm{d}w = 160$ and $\mathrm{d}v/\mathrm{d}w = 64$, the total derivatives of the intermediates with respect to w. Carrying the dependence on one input forward through the graph in this way is called *forward mode*. In plain Python:

```
w, x, y, z = 1, 1, -2, 1
# Forward pass: node values
a, b = (w + x + y + z)**2, (w + x - y - z)**2
u, v = (a + b)**2, (a - b)**2
f = (u + v)**2
print('a, b, u, v, f =', a, b, u, v, f)

# Edge derivatives
df_du, df_dv = 2*(u + v), 2*(u + v)
du_da, du_db, dv_da, dv_db = 2*(a + b), 2*(a + b), 2*(a - b), -2*(a - b)
da_dw, db_dw = 2*(w + x + y + z), 2*(w + x - y - z)

# Forward mode: push the dependence on w through the graph
du_dw, dv_dw = du_da*da_dw + du_db*db_dw, dv_da*da_dw + dv_db*db_dw
df_dw = df_du*du_dw + df_dv*dv_dw
print('df/dw =', df_dw)
```

```
a, b, u, v, f = 1 9 100 64 26896
df/dw = 73472
```

Forward mode returns $\mathrm{d}f/\mathrm{d}w = 73472$, but nothing in it can be reused for x, y, or z: $\mathrm{d}u/\mathrm{d}w$ and $\mathrm{d}v/\mathrm{d}w$ belong to w alone. Training needs the derivative of one loss with respect to every parameter, and this grouping computes them one input at a time.

34.3.2.2 Adjoints and the Backward Pass

The remedy is to keep the output fixed instead of the input. Forward mode asked, for one input, how every later node responds to it. We now ask, for the one output, how it responds to every earlier node. The *adjoint* $\bar{n}$ of a node n is the total derivative $\mathrm{d}f/\mathrm{d}n$ of the output with respect to that node: the sum, over all paths from n to f, of the products of the edge derivatives. Every such path leaves n through one of its children c, the nodes that use n directly, and the paths from c onward sum to $\bar{c}$. Grouping the sum Equation 34.28 by this first edge turns it into a recursion,

$$\bar{f} = 1, \qquad \bar{n} = \sum_{\text{children } c \text{ of } n} \bar{c}\,\frac{\partial c}{\partial n}, \tag{34.30}$$

in which only edge derivatives and the adjoints of later nodes appear. Starting from $\bar{f} = 1$ and visiting the nodes from the output backward, each adjoint is ready when it is needed. In the skip-edge graph Equation 34.29 the recursion gives $\bar{r} = 1$ and $\bar{p} = \bar{r} \cdot 2p + \bar{f} \cdot 1 = 7$, the total found there. For the four-input graph:

- $\bar{u} = \bar{v} = 328$;
- $\bar{a} = \bar{u} \cdot 20 + \bar{v} \cdot (-16) = 1312$ and $\bar{b} = \bar{u} \cdot 20 + \bar{v} \cdot 16 = 11808$;
- $\bar{w} = \bar{x} = \bar{a} \cdot 2 + \bar{b} \cdot 6 = 73472$ and $\bar{y} = \bar{z} = \bar{a} \cdot 2 - \bar{b} \cdot 6 = -68224$.

One *backward pass* gives the derivatives with respect to all four inputs (Figure 34.19). The code reuses the edge derivatives of the forward-mode cell.

```
da_dy, db_dy = 2*(w + x + y + z), -2*(w + x - y - z)  # also for x, z
# Backward pass: adjoints from the output toward the inputs
u_bar, v_bar = df_du, df_dv
a_bar = u_bar*du_da + v_bar*dv_da
b_bar = u_bar*du_db + v_bar*dv_db
print(f'adjoints: u {u_bar}, v {v_bar}, a {a_bar}, b {b_bar}')
w_bar = a_bar*da_dw + b_bar*db_dw  # x enters a, b like w
y_bar = a_bar*da_dy + b_bar*db_dy  # z enters a, b like y
print('gradient:', {'w': w_bar, 'x': w_bar, 'y': y_bar, 'z': y_bar})
```

```
adjoints: u 328, v 328, a 1312, b 11808
gradient: {'w': 73472, 'x': 73472, 'y': -68224, 'z': -68224}
```

The derivatives with respect to w and x agree, and so do those with respect to y and z, because f depends on its inputs only through $w + x$ and $y + z$.

Counting multiplications shows the saving. Forward mode needs six per input (two each for $\mathrm{d}u/\mathrm{d}w$, $\mathrm{d}v/\mathrm{d}w$, and $\mathrm{d}f/\mathrm{d}w$), 24 for four inputs. The backward pass needs four for $\bar{a}$ and $\bar{b}$ and then two per input, 12 in all: it does the work on the shared part of the graph once. A factor of two is nothing to boast of. But the saving grows with the number of inputs: each further input costs forward mode a whole pass and the backward pass two multiplications.

34.3.2.3 Backpropagation in a Framework

Computing derivatives from the output back toward the inputs is *backpropagation* (Rumelhart et al. 1986), which Section 4.3 applies to a multilayer perceptron. Each framework's automatic differentiation runs the same two passes: it records values and edge derivatives

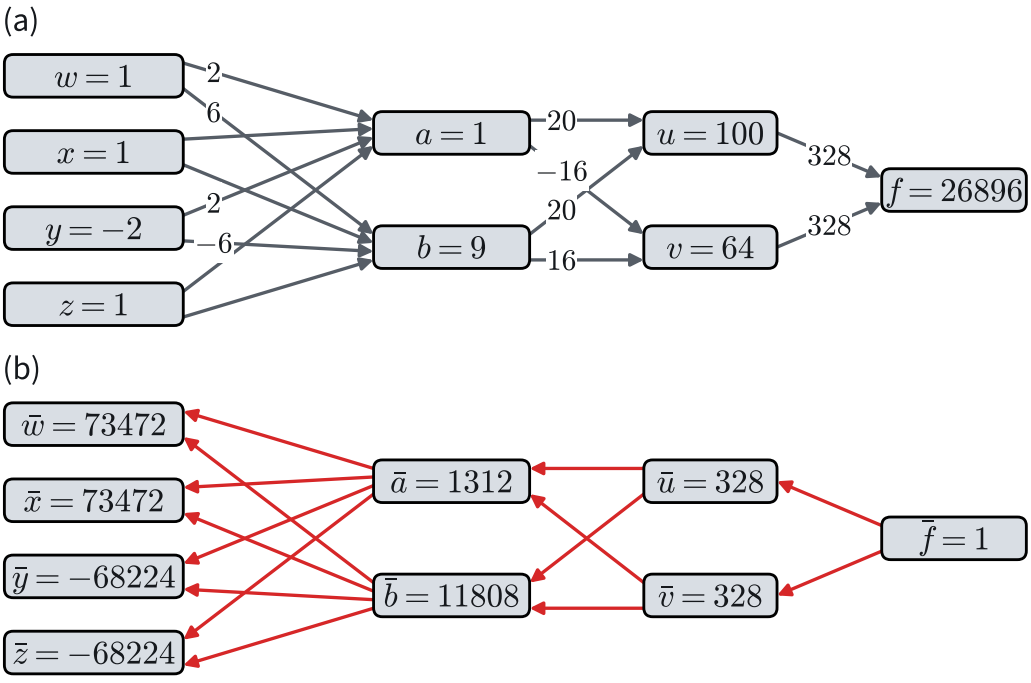


Figure 34.19: (a) The forward pass records every node value and every edge derivative. (b) The backward pass accumulates adjoints from $\bar{f} = 1$: $\bar{a} = 328 \cdot 20 + 328 \cdot (-16) = 1312$, and $\bar{w} = 1312 \cdot 2 + 11808 \cdot 6 = 73472$.

on the way forward and accumulates adjoints on the way back. Here it is on the same four inputs.

```
wxyz = torch.tensor([1., 1., -2., 1.], requires_grad=True)
w, x, y, z = wxyz
a, b = (w + x + y + z)**2, (w + x - y - z)**2
u, v = (a + b)**2, (a - b)**2
f = (u + v)**2
f.backward()  # one backward pass
print(wxyz.grad)
```

```
tensor([ 73472.,  73472., -68224., -68224.])
```

The library returns $(73472, 73472, -68224, -68224)$, the adjoints of our hand computation. The agreement is exact, not approximate: all the numbers involved are integers below 2^{24}, which single precision represents without rounding.

Every node here was a scalar, and we counted multiplications on a graph that fits on a napkin. A layer of a network maps a vector to a vector. Its edge derivatives are matrices, and the order in which matrices are multiplied matters far more than the order of numbers. Section 34.4 takes that step.

34.3.3 Summary

The derivative of an output with respect to an input is the sum, over the paths between them, of the products of the edge derivatives. The adjoint of a node, the total derivative of the output with respect to it, collects all paths leaving the node. Grouping them by their first edge gives a recursion that, run from the output backward, yields every adjoint in one pass. This is backpropagation. Carrying one input forward instead gives that input's derivative alone, so a loss with many inputs is differentiated backward.

34.3.4 Exercises

1. code **Backward pass for two tanh units.** A network with one input x, two hidden units, and one output computes, for a target t,

$$h_i = \tanh(w_i x + b_i) \quad (i = 1, 2), \qquad f = v_1 h_1 + v_2 h_2, \qquad \ell = \tfrac{1}{2}(f - t)^2.$$

   1. Draw the dependency graph of ℓ. In plain Python, write a forward pass that stores every intermediate value and a backward pass that returns the partial derivatives of ℓ with respect to all six parameters $w_1, b_1, w_2, b_2, v_1, v_2$. Check them against automatic differentiation at $x = 1.5$, $t = 0.5$, and $(w_1, b_1, w_2, b_2, v_1, v_2) = (0.5, 0, -1, 0.2, 1, -0.5)$. Which intermediate derivative does every one of the six partials reuse?
   2. Set $w_1 = 5$ and $x = 2$, keeping the other values. Compute $\partial \ell / \partial w_1$ with the plain-Python backward pass and explain its size from the product of edge derivatives along its path. Compare it with the value that automatic differentiation returns in single precision, and explain the difference. What does this imply for training a unit whose pre-activation is large?

3. Set $w_1 = w_2$, $b_1 = b_2$, and $v_1 = v_2$. Show from the backward pass that both units receive identical gradients, so that gradient descent keeps them identical at every step. What does this imply for the initialization of hidden units?

2. **Paths through a residual stack.** ◻ In a one-dimensional model of the residual networks of Section 7.4, scalar blocks compute $x_{l+1} = x_l + g_l(x_l)$ for $l = 0, \dots, m-1$ with differentiable functions g_l.

   1. Use the chain rule to write $\partial x_m / \partial x_0$ as a product of m factors.
   2. Expand the product and match each term to a directed path from x_0 to x_m in the dependency graph. How many paths are there, and how many of them pass through exactly k of the functions g_l? Compare the number of paths with the number of multiplications that a backward pass needs to compute $\partial x_m / \partial x_0$.
   3. For a trained network of $m = 54$ residual blocks, Veit et al. (2016) found that almost all of the gradient flows along the paths through 5 to 17 blocks. What fraction of all paths are these? Now assume that $g_l'(x_l) = c > 0$ for every block. Which value of c makes the paths through about 11 blocks contribute the most to $\partial x_m / \partial x_0$, and what share of the derivative do the paths through 5 to 17 blocks then carry?

3. code **Gradient checks in single precision.** ◻ A gradient check compares a backward pass with the central difference $(f(w + h, x, y, z) - f(w - h, x, y, z))/(2h)$. Let f be the function of Equation 34.27, whose exact partial derivative at $(w, x, y, z) = (1, 1, -2, 1)$ is $\partial f / \partial w = 73472$.

   1. Implement f in NumPy and compute the central difference at this point for $h = 10^{-1}, 10^{-2}, \dots, 10^{-12}$, once in `float64` and once in `float32`. Plot the relative error against h on log-log axes for both precisions, and read off the best h and the smallest relative error of each.
   2. The error of the central difference is a truncation part of order h^2 plus a rounding part of order $\varepsilon_{\text{mach}} |f| / h$, where the machine epsilon $\varepsilon_{\text{mach}}$ is about 2.2×10^{-16} for `float64` and 1.2×10^{-7} for `float32`. By which factors should the best h and the smallest relative error change from `float64` to `float32`? Compare the predicted factors with the plot.
   3. Which tolerance on the relative error can a `float32` gradient check use without rejecting a correct backward pass? Why are gradient checks run in double precision?
   4. Why does the `float32` estimate become exactly zero for the smallest values of h?

34.4 Matrix Calculus

The last two sections treated a loss as a function of scalars. Section 34.2 wrote its gradient as a column of partial derivatives, and Section 34.3 computed that column by a backward pass through a graph with one number per node. A network is not written in scalars. Each layer maps a vector to a vector, and its weights form a matrix. What is the derivative of the loss with respect to a vector or a matrix, and how do we compute it for a whole layer at once?

A network small enough to work by hand shows what is needed. It has two inputs, two ReLU hidden units and one output,

$$g(\mathbf{x}) = \mathbf{w}_2^\top \varphi(\mathbf{W}_1\mathbf{x} + \mathbf{b}), \qquad \mathbf{W}_1 = \begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} 0 \\ -4 \end{pmatrix}, \quad \mathbf{w}_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix},$$

where φ applies $\max(0, \cdot)$ to each entry. At $\mathbf{x} = (1, 1)$ the pre-activations are $\mathbf{W}_1\mathbf{x} + \mathbf{b} = (3, -2)$, so the second unit is off, the hidden vector is $(3, 0)$, and the output is $g = 6$. A training step asks three questions about this number. If the input moves by δ, the output changes by $2\delta_1 + 4\delta_2$, a row of two numbers, the transpose of the column gradient. If the weight matrix moves by $\Delta\mathbf{W}_1$, the output changes by $2\,\Delta W_{11} + 2\,\Delta W_{12}$ and not at all through the second row, so the derivative with respect to a matrix is again a matrix, $\left(\begin{smallmatrix} 2 & 2 \\ 0 & 0 \end{smallmatrix}\right)$. And the hidden layer $\mathbf{x} \mapsto \varphi(\mathbf{W}_1\mathbf{x} + \mathbf{b})$ maps a vector to a vector. Neither a slope nor a gradient describes its derivative, yet the output reaches $\mathbf{W}_1$ only through that layer.

Such derivatives are easy to get wrong by hand. A hand-derived gradient with a transposed square factor raises no error, and training goes wrong. Reference tables of matrix derivatives run to hundreds of entries, and they do not protect against that mistake.

This section answers the three questions with one object and one rule. The derivative of a vector-valued map is a matrix, the *Jacobian*, defined as the map's best linear approximation. Composing maps multiplies Jacobians, so the derivative of a network is a product of matrices, one per layer. The derivatives a layer needs (linear and quadratic forms, least squares, matrix products, softmax) all follow from one rule: write down the first-order change of the function and read the gradient off as its coefficient. The same rule produces the backward pass of each layer, which the main text uses in Section 4.3, and $\mathbf{p} - \mathbf{y}$, the gradient every classifier computes (Section 3.1.2.2).

Three results are worth remembering: the product of Jacobians Equation 34.36, the rule that a gradient has the shape of its variable, and the backward rule Equation 34.39 of a matrix product. The proof of the chain rule can be skipped on a first reading. The differential method (Section 34.4.3.1) cannot, since everything after it applies it.

We use the following imports throughout this section.

```
import numpy as np
import torch
```

34.4.1 The Jacobian

34.4.1.1 The Best Linear Approximation

Near $\mathbf{x} = (1, 1)$ the hidden layer of our network is affine, so its linear approximation is exact there and would leave no error to study. Let us therefore start with a curved map, the one this section and the next use throughout, and return to the network once the definition is in place:

$$\mathbf{f}(x, y) = \begin{pmatrix} x + \sin y \\ y + x^2/2 \end{pmatrix}, \qquad \mathbf{x}_0 = (0.5,\ 0.5).$$

At $\mathbf{x}_0$ its value is $\mathbf{f}(\mathbf{x}_0) = (0.9794,\ 0.6250)$. Each of the two outputs has two partial derivatives: $\partial f_1/\partial x = 1$, $\partial f_1/\partial y = \cos y = 0.8776$, $\partial f_2/\partial x = x = 0.5$ and $\partial f_2/\partial y = 1$. We arrange the four numbers in a matrix with one row per output and one column per input,

$$\mathbf{J}(\mathbf{x}_0) = \begin{pmatrix} 1 & \cos 0.5 \\ 0.5 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.8776 \\ 0.5 & 1 \end{pmatrix}.$$

Column j says how both outputs move when input j moves. This matrix is the derivative of $\mathbf{f}$ at $\mathbf{x}_0$, in the sense that $\mathbf{f}(\mathbf{x}_0 + \delta) \approx \mathbf{f}(\mathbf{x}_0) + \mathbf{J}\delta$ for small δ, and no other matrix does as well.

In one variable, the derivative is the slope of the best straight-line fit, $f(x + \delta) \approx f(x) + f'(x)\,\delta$. The vector-valued generalization uses the same local approximation. We say $\mathbf{f}$ is *differentiable* at $\mathbf{x}$ if there is a matrix $\mathbf{J} \in \mathbb{R}^{m \times n}$ such that

$$\mathbf{f}(\mathbf{x} + \delta) = \mathbf{f}(\mathbf{x}) + \mathbf{J}\,\delta + o(\|\delta\|), \tag{34.31}$$

that is, the error of the linear model $\mathbf{f}(\mathbf{x}) + \mathbf{J}\delta$ divided by $\|\delta\|$ tends to zero as $\delta \to \mathbf{0}$. That matrix, if it exists, is unique, is called the *Jacobian* $\mathbf{J}_\mathbf{f}(\mathbf{x})$, and its entries are the partial derivatives:

$$[\mathbf{J}_\mathbf{f}]_{ij} = \frac{\partial f_i}{\partial x_j}. \tag{34.32}$$

To see why, feed Equation 34.31 the perturbation $\delta = t\,\mathbf{e}_j$ (a small step along axis j) and read off component i: dividing by t and letting $t \to 0$ leaves $\partial f_i / \partial x_j$ in the (i, j) slot, whatever matrix we started with. The partial derivatives supply the entries of $\mathbf{J}$, but the linear approximation is what defines it.

Partials alone do not guarantee a Jacobian. The function $f(x, y) = 2xy/(x^2 + y^2)$ with $f(0, 0) = 0$ of Section 34.2.1.2 has both partials 0 at the origin and equals 1 on the diagonal $x = y$. The only candidate matrix is the zero row, and its error of 1 along the diagonal does not shrink with δ, so f has no Jacobian there. A sufficient condition is continuity of the partials: if every partial $\partial f_i / \partial x_j$ exists near $\mathbf{x}$ and is continuous at $\mathbf{x}$, then $\mathbf{f}$ is differentiable at $\mathbf{x}$ (Rudin 1976).

Figure 34.20 shows the definition at work: near $\mathbf{x}_0$, the map acts like its Jacobian. A circle of radius r around $\mathbf{x}_0$ maps nearly onto the ellipse $\mathbf{f}(\mathbf{x}_0) + \mathbf{J}\delta$, $\|\delta\| = r$, whose semi-axes are r times the singular values of $\mathbf{J}$ (a matrix maps the unit circle to an ellipse with these semi-axes, Section 33.5). Here the singular values are 1.706 and 0.329, so the circle of radius 0.35 becomes an ellipse with semi-axes 0.60 and 0.12. The largest gap between the true image and the ellipse is 0.061 at $r = 0.35$ and 0.015 at $r = 0.175$: halving the radius cuts the gap fourfold.

The definition says that the error of the linear model, divided by $\|\delta\|$, tends to zero. It does not say how fast. We shrink one perturbation tenfold at a time and print that ratio.

```
def f(v):                                   # the running map, R^2 -> R^2
    x, y = v
    return np.array([x + np.sin(y), y + x**2 / 2])

def J(v):                                   # its Jacobian, entry by entry
    x, y = v
    return np.array([[1.0, np.cos(y)], [x, 1.0]])
```

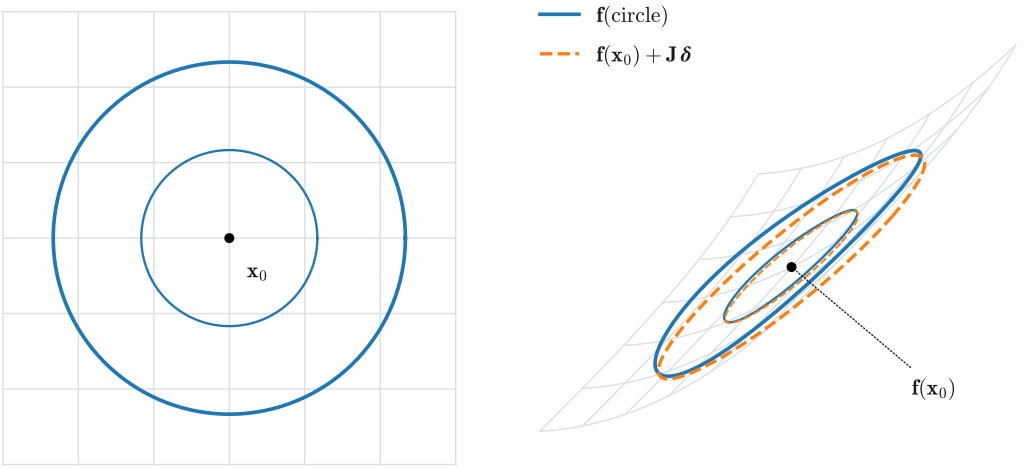


Figure 34.20: A differentiable map is locally its Jacobian. Two circles of radius 0.35 and 0.175 around $\mathbf{x}_0 = (0.5, 0.5)$ (left) and their images under $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ (right, blue) against the ellipses $\mathbf{f}(\mathbf{x}_0) + \mathbf{J}(\mathbf{x}_0)\delta$ (orange, dashed). The semi-axes of each ellipse are the radius times the singular values 1.71 and 0.33 of $\mathbf{J}$. Halving the radius cuts the largest gap between image and ellipse from 0.061 to 0.015, the $o(\|\delta\|)$ remainder.

```
x0 = np.array([0.5, 0.5])
for s in [1, 0.1, 0.01, 0.001]:
    delta = s * np.array([0.2, 0.1])
    err = np.linalg.norm(f(x0 + delta) - f(x0) - J(x0) @ delta)
    print(f'|delta| = {np.linalg.norm(delta):.1e}   '
          f'error / |delta| = {err / np.linalg.norm(delta):.1e}')
```

```
|delta| = 2.2e-01   error / |delta| = 9.0e-02
|delta| = 2.2e-02   error / |delta| = 9.0e-03
|delta| = 2.2e-03   error / |delta| = 9.0e-04
|delta| = 2.2e-04   error / |delta| = 9.0e-05
```

The ratio goes from 0.09 to 0.009, 0.0009 and 0.00009. It tends to zero, which is the $o(\|\delta\|)$ of the definition. It also falls tenfold for every tenfold step, so the error itself falls a hundredfold: the error is $O(\|\delta\|^2)$. The definition promises only the first property; the second holds because $\mathbf{f}$ has continuous second derivatives.

34.4.1.2 Gradients, Hessians, and the Shape of a Derivative

The gradient and the Hessian of Section 34.2 are both built from Jacobians. Seeing them this way settles a question of shape that every hand derivation runs into: whether a derivative is a row or a column, and how a derivative with respect to a matrix is arranged.

When $m = 1$, a scalar field $f : \mathbb{R}^n \to \mathbb{R}$, the Jacobian is a single row, $\partial f/\partial \mathbf{x} = [\partial f/\partial x_1, \dots, \partial f/\partial x_n]$, and Equation 34.31 is the first-order expansion $f(\mathbf{x} + \delta) \approx f(\mathbf{x}) + (\partial f/\partial \mathbf{x})\,\delta$. We wrote the gradient as a column. The two are transposes of each other,

$$\nabla_{\mathbf{x}} f = \Big(\frac{\partial f}{\partial \mathbf{x}}\Big)^{\top}, \tag{34.33}$$

and this book uses both, each for one job. Jacobians of vector-valued maps are $m \times n$, one row per output, so that the chain rule below composes without transposes. Gradients of scalars are columns, $\nabla_{\mathbf{x}} f$, the vector an optimizer subtracts from $\mathbf{x}$. For the network of the opening, $\partial g/\partial \mathbf{x} = (2,\ 4)$ is a 1×2 Jacobian and $\nabla_{\mathbf{x}} g = (2, 4)^{\top}$ is the column.

A scalar function of a matrix $\mathbf{W} \in \mathbb{R}^{p \times q}$ needs one more convention. Its gradient $\nabla_{\mathbf{W}} f$ is the $p \times q$ matrix with entries $\partial f / \partial W_{ij}$, so it has the shape of $\mathbf{W}$. Then the first-order expansion reads

$$f(\mathbf{W} + \mathbf{E}) = f(\mathbf{W}) + \langle \nabla_{\mathbf{W}} f,\ \mathbf{E} \rangle + o(\|\mathbf{E}\|), \qquad \langle \mathbf{A}, \mathbf{B} \rangle = \operatorname{tr}(\mathbf{A}^{\top}\mathbf{B}) = \sum_{ij} A_{ij} B_{ij}, \tag{34.34}$$

where $\langle \cdot, \cdot \rangle$ multiplies two matrices of the same shape entry by entry and adds the products, the dot product of the matrices read as long vectors. The change of f is the sum over all entries of "sensitivity times nudge". In the opening, $\nabla_{\mathbf{W}_1} g$ is the 2×2 matrix $\left(\begin{smallmatrix} 2 & 2 \\ 0 & 0 \end{smallmatrix}\right)$, and $\langle \nabla_{\mathbf{W}_1} g, \Delta \mathbf{W}_1 \rangle = 2\,\Delta W_{11} + 2\,\Delta W_{12}$ is the change we found there. One rule checks every formula in this section: a gradient has the shape of its variable. If a derived gradient does not, a transpose is missing.

Differentiating once more brings in the Hessian. The gradient field $\nabla f : \mathbb{R}^n \to \mathbb{R}^n$ is itself a vector-valued map, and its Jacobian is the matrix of second partials, the Hessian $\mathbf{H}_f$, with $[\mathbf{H}_f]_{ij} = \partial^2 f / \partial x_i \partial x_j$. The entry $[\mathbf{J}_{\nabla f}]_{ij} = \partial^2 f / \partial x_j \partial x_i$ takes the derivatives in the other order; the two agree, and $\mathbf{H}_f$ is symmetric, when the mixed partials are continuous (the Clairaut–Schwarz theorem of Section 34.2).

34.4.2 The Chain Rule for Jacobians

A network is a composition of layers. On a graph of scalars the chain rule summed over every path from an input to an output (Equation 34.28). For layers that are vector maps, that sum is a matrix product.

34.4.2.1 Composition as a Matrix Product

Proposition (chain rule). *If $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^p$ is differentiable at $\mathbf{x}$ and $\mathbf{g} : \mathbb{R}^p \to \mathbb{R}^m$ is differentiable at $\mathbf{f}(\mathbf{x})$, then $\mathbf{g} \circ \mathbf{f}$ is differentiable at $\mathbf{x}$ and*

$$\mathbf{J}_{\mathbf{g}\circ\mathbf{f}}(\mathbf{x}) = \mathbf{J}_{\mathbf{g}}\big(\mathbf{f}(\mathbf{x})\big)\,\mathbf{J}_{\mathbf{f}}(\mathbf{x}). \tag{34.35}$$

Proof. Write $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and let $\mathbf{A} = \mathbf{J}_{\mathbf{f}}(\mathbf{x})$, $\mathbf{B} = \mathbf{J}_{\mathbf{g}}(\mathbf{y})$. We substitute the linear model for $\mathbf{f}$ into the linear model for $\mathbf{g}$. By Equation 34.31, $\mathbf{f}(\mathbf{x} + \delta) = \mathbf{y} + \mathbf{A}\delta + o(\|\delta\|)$. Calling that displacement $\eta = \mathbf{A}\delta + o(\|\delta\|)$ and feeding it to $\mathbf{g}$,

$$\mathbf{g}\big(\mathbf{f}(\mathbf{x} + \delta)\big) = \mathbf{g}(\mathbf{y}) + \mathbf{B}\,\eta + o(\|\eta\|) = \mathbf{g}(\mathbf{y}) + \mathbf{B}\mathbf{A}\,\delta + o(\|\delta\|).$$

The second equality needs two facts. $\mathbf{B}$ applied to the $o(\|\delta\|)$ remainder is again $o(\|\delta\|)$. And $\|\eta\| \le \|\mathbf{A}\|\,\|\delta\| + o(\|\delta\|)$ is bounded by a constant multiple of $\|\delta\|$, so $\mathbf{g}$'s own remainder $o(\|\eta\|)$ is $o(\|\delta\|)$ as well; this is the one step that is not routine. The coefficient of δ is therefore a matrix with the property Equation 34.31, and by uniqueness it is the Jacobian of the composite: $\mathbf{J}_{\mathbf{g}\circ\mathbf{f}} = \mathbf{BA}$. ■

One entry of the product recovers the sum over paths: $[\mathbf{BA}]_{ik} = \sum_j (\partial g_i/\partial y_j)(\partial f_j/\partial x_k)$ sums, over the intermediate variables y_j, the product of the derivatives along each path from x_k to g_i.

34.4.2.2 The Jacobian of a Network

The network of the opening is a composition of three maps: the affine map $\mathbf{a} = \mathbf{W}_1\mathbf{x} + \mathbf{b}$, the ReLU $\mathbf{h} = \varphi(\mathbf{a})$, and the output $g = \mathbf{w}_2^\top \mathbf{h}$. The first map is its own linear approximation, so its Jacobian is $\mathbf{W}_1$; the bias shifts the value and not the slope. The ReLU acts on each entry separately, with slope 1 where $a_i > 0$ and 0 where $a_i < 0$, so its Jacobian is the diagonal matrix $\operatorname{diag}(\mathbf{m})$ of the *mask* $\mathbf{m}$, the vector of these slopes. The output map has Jacobian $\mathbf{w}_2^\top$. At $\mathbf{x} = (1, 1)$, where $\mathbf{a} = (3, -2)$ and $\mathbf{m} = (1, 0)$, the chain rule multiplies the three:

$$\mathbf{J}_g(\mathbf{x}) = \mathbf{w}_2^\top \operatorname{diag}(\mathbf{m})\,\mathbf{W}_1 = \begin{pmatrix} 2 & -1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 0 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 3 & -1 \end{pmatrix} = \begin{pmatrix} 2 & 4 \end{pmatrix}.$$

The mask zeroes the second entry of $\mathbf{w}_2^\top$, so the second row of $\mathbf{W}_1$ never reaches the output: the weights of an inactive unit play no part in the derivative (Figure 34.21). The row $(2, 4)$ is the change $2\delta_1 + 4\delta_2$ of the opening. At another input the mask changes and so does the Jacobian. At $\mathbf{x} = (2, -3)$ the pre-activations are $(-4, 5)$, the mask is $(0, 1)$, and $\mathbf{J}_g = (0,\ -1)\,\mathbf{W}_1 = (-3,\ 1)$. Because ReLU is piecewise linear, a finite difference along each axis reproduces these entries without error for steps small enough to keep the mask fixed (exercise 1).

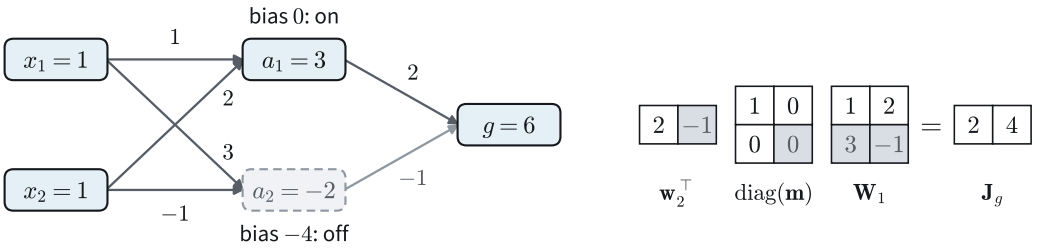


Figure 34.21: The Jacobian of the 2-2-1 ReLU network at $\mathbf{x} = (1, 1)$ is the product of three matrices. The second unit's pre-activation is -2, so its mask entry is 0 and the second row of $\mathbf{W}_1$ never reaches the output: $\mathbf{J}_g = \mathbf{w}_2^\top \operatorname{diag}(1, 0)\,\mathbf{W}_1 = (2,\ 4)$.

The same two kinds of factor make up every multilayer perceptron. A dense layer $\mathbf{x} \mapsto \mathbf{W}\mathbf{x}$ has $\mathbf{J} = \mathbf{W}$. An elementwise activation $\varphi(\mathbf{x}) = (\varphi(x_1), \dots, \varphi(x_n))$ couples no pair of coordinates, so $\mathbf{J}_\varphi(\mathbf{x}) = \operatorname{diag}(\varphi'(x_1), \dots, \varphi'(x_n))$. This requires φ to be differentiable at every x_i. ReLU has no derivative at 0, and frameworks use a fixed value there by convention (Section 34.1.6). A network of depth K is a composition $\mathbf{f} = \mathbf{f}_K \circ \cdots \circ \mathbf{f}_1$, and iterating Equation 34.35 turns its derivative into one product,

$$\mathbf{J} = \mathbf{J}_K \mathbf{J}_{K-1} \cdots \mathbf{J}_1, \tag{34.36}$$

with each factor evaluated at the input of its layer. For the alternation of dense layers and activations this is $\mathbf{J} = \operatorname{diag}(\varphi_K') \mathbf{W}_K \cdots \operatorname{diag}(\varphi_1') \mathbf{W}_1$: weight matrices interleaved with diagonal factors. Multiplying a vector by a diagonal factor scales each entry, which costs $O(n)$ operations instead of $O(n^2)$.

We now define the network in code, for reuse later in the section, and print the product $\mathbf{w}_2^\top \operatorname{diag}(\mathbf{m}) \mathbf{W}_1$ beside the gradient that a framework's automatic differentiation returns.

```
W1, b = torch.tensor([[1.0, 2.0], [3.0, -1.0]]), torch.tensor([0.0, -4.0])
w2, x = torch.tensor([2.0, -1.0]), torch.tensor([1.0, 1.0])

def g(x, W1, b, w2):                          # the 2-2-1 ReLU network
    return w2 @ torch.relu(W1 @ x + b)

m = (W1 @ x + b > 0).float()                  # the ReLU mask
print('by hand    :', (w2 @ torch.diag(m) @ W1).numpy())
print('by autograd:', torch.func.grad(g)(x, W1, b, w2).numpy())
```

```
by hand    : [2. 4.]
by autograd: [2. 4.]
```

Both lines print $(2, 4)$, as they must.

The product Equation 34.36 also says how large a derivative can become. When every factor is the same matrix $\mathbf{W}$, as in a recurrent network that applies one weight matrix at every step (ignoring the activations), the product is $\mathbf{W}^K$. For a diagonalizable $\mathbf{W} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ we have $\mathbf{W}^K = \mathbf{V}\Lambda^K\mathbf{V}^{-1}$, so for large K the norm of the product grows or shrinks like $|\lambda_{\max}|^K$, where $\lambda_{\max}$ is the eigenvalue of largest magnitude: $0.9^{50} \approx 0.005$ and $1.1^{50} \approx 117$. This is the vanishing and exploding gradient of deep and recurrent networks (Section 4.4, Section 8.6). The rate is only the long-run one. When the eigenvectors in $\mathbf{V}$ are far from orthogonal, the product can grow for many steps before it decays (Section 33.4.2.3), and a product of different factors is governed by singular values (Section 33.4.2.5).

The product has a limit that matters for training: it is the derivative with respect to the *input*. The weight matrices $\mathbf{W}_k$ appear in it as factors, never as variables, so it does not say how the output changes when a weight changes. Those gradients need one more rule per layer, which Section 34.4.3.2 derives and applies to this same network.

34.4.2.3 The Order of Multiplication

Matrix multiplication is associative, so the product Equation 34.36 may be bracketed in any order and every order gives the same matrix. The orders differ in cost by a factor that grows with the width of the network. Training needs the gradient of a scalar loss ℓ, so the chain starts on the left with a row vector, the $1 \times h$ Jacobian $\mathbf{u}^\top = \partial\ell/\partial\mathbf{h}_K$ of the loss with respect to the last layer's output $\mathbf{h}_K$. Multiplying from the left keeps every intermediate result a row vector, so every step multiplies a matrix by a vector. Multiplying from the right instead multiplies matrix by matrix until the row arrives at the end. The cheap order is the one in which every step is a matrix–vector product.

A count makes the difference concrete. Take $\mathbf{u}^\top \mathbf{J}_3\mathbf{J}_2\mathbf{J}_1$ with shapes $1 \times h$, $h \times h$, $h \times h$ and $h \times n$, and $h = n = 1000$, where a $p \times q$ matrix times a $q \times r$ matrix costs pqr multiplications. From the left there are three matrix–vector products, $h^2 + h^2 + hn = 3 \times 10^6$ multiplications. From the right, $\mathbf{J}_2\mathbf{J}_1$ and then $\mathbf{J}_3(\mathbf{J}_2\mathbf{J}_1)$ cost $h^2 n = 10^9$ each, and the final product with $\mathbf{u}^\top$ costs $hn = 10^6$, in all 2.001×10^9. Three million against two billion is a factor of about 670, or $2h/3$ (Figure 34.22).

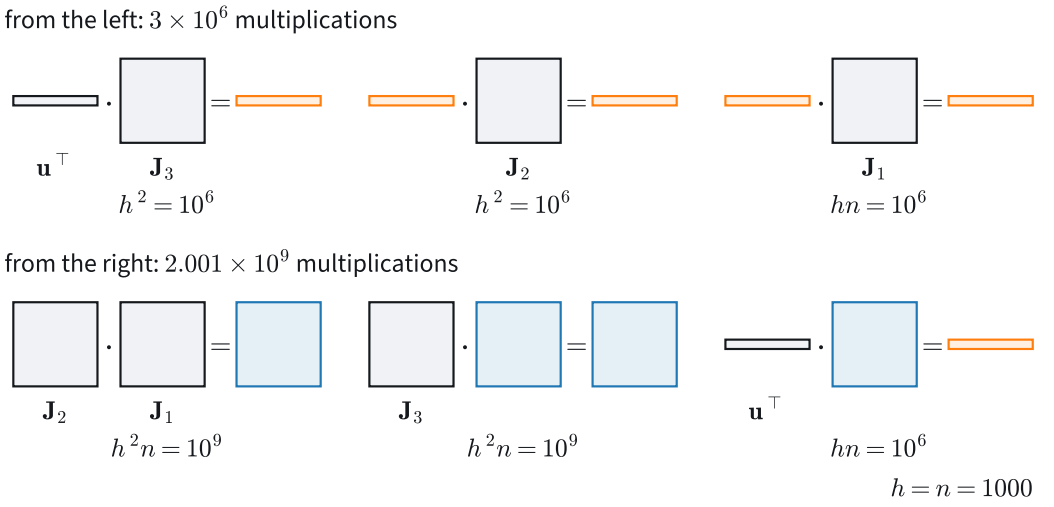


Figure 34.22: Two orders for $\mathbf{u}^\top \mathbf{J}_3\mathbf{J}_2\mathbf{J}_1$ with $h = n = 1000$. From the left every intermediate is a $1 \times h$ row and every step is a matrix–vector product: 3×10^6 multiplications. From the right the first two steps are matrix–matrix products of 10^9 multiplications each. The shapes, not the depth, decide the cost.

The two orders are two directions through the network. Multiplying from the left carries the sensitivity of the output backward through the layers, at the cost of one matrix–vector product per layer. Multiplying from the right is the cheap order when a column $\mathbf{v}$ stands at that end of the chain: it carries one input direction forward and yields $\mathbf{J}\mathbf{v}$. Programs that evaluate the two orders are called reverse-mode and forward-mode automatic differentiation (Section 34.5).

The count above uses factors of equal width. When the factors have different shapes, neither end order need be cheapest. The best bracketing of a chain is found by a dynamic program (exercise 3); for a general computation graph the problem is NP-complete (Naumann 2008).

34.4.3 Derivatives with Respect to Vectors and Matrices

A layer needs the gradient of a scalar with respect to vectors and matrices: its input, its weights, its bias. This part derives every gradient the section needs with one method, which produces each result in the shape of its variable.

34.4.3.1 The Differential Method

Write $d\mathbf{x}$ for a small change of the variable and df for the part of the resulting change of f that is linear in $d\mathbf{x}$, the term $\mathbf{J}\delta$ of Equation 34.31 with δ renamed $d\mathbf{x}$. By Equation 34.31 and Equation 34.34, this linear part is an inner product with the gradient,

$$df = \langle \nabla f,\ d\mathbf{x} \rangle, \tag{34.37}$$

for a vector variable ($\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^\top \mathbf{b}$) and a matrix variable alike. The equation can be read two ways. Given the gradient, it says how f responds to a nudge. Given f, it is a recipe for the gradient: compute df by the ordinary rules of differentiation, rearrange it into the form $\langle \mathbf{M},\ d\mathbf{x} \rangle$, and read off $\nabla f = \mathbf{M}$. The reading is unambiguous because a linear function of $d\mathbf{x}$ can be written as $\langle \mathbf{M}, d\mathbf{x} \rangle$ for only one $\mathbf{M}$: putting $d\mathbf{x}$ equal to the matrix with a single 1 in position (i, j) gives M_{ij}.

Four rules compute df. The differential is linear. It obeys the product rule $d(\mathbf{AB}) = d\mathbf{A}\,\mathbf{B} + \mathbf{A}\,d\mathbf{B}$, since $(\mathbf{A} + d\mathbf{A})(\mathbf{B} + d\mathbf{B})$ differs from $\mathbf{AB} + d\mathbf{A}\,\mathbf{B} + \mathbf{A}\,d\mathbf{B}$ by the second-order term $d\mathbf{A}\,d\mathbf{B}$. It commutes with transposition, $d(\mathbf{A}^\top) = (d\mathbf{A})^\top$. And a function applied to every entry obeys the scalar chain rule in every entry, $d\varphi(\mathbf{X}) = \varphi'(\mathbf{X}) \odot d\mathbf{X}$, where $\odot$ multiplies entry by entry.

Rearranging df into the form $\langle \mathbf{M}, d\mathbf{x} \rangle$ needs one rule more. The *move rule* shifts a factor next to $d\mathbf{x}$ to the other side of the inner product:

$$\langle \mathbf{A},\ \mathbf{BC} \rangle = \langle \mathbf{B}^\top \mathbf{A},\ \mathbf{C} \rangle = \langle \mathbf{AC}^\top,\ \mathbf{B} \rangle,$$

which follows from $\operatorname{tr}(\mathbf{XY}) = \operatorname{tr}(\mathbf{YX})$: $\operatorname{tr}(\mathbf{A}^\top \mathbf{BC}) = \operatorname{tr}\big((\mathbf{B}^\top \mathbf{A})^\top \mathbf{C}\big) = \operatorname{tr}(\mathbf{CA}^\top \mathbf{B}) = \operatorname{tr}\big((\mathbf{AC}^\top)^\top \mathbf{B}\big)$. A factor to the left of the variable moves across as its transpose on the left, a factor to the right as its transpose on the right. An entrywise factor moves across unchanged, $\langle \mathbf{A}, \mathbf{B} \odot \mathbf{C} \rangle = \langle \mathbf{A} \odot \mathbf{B}, \mathbf{C} \rangle$, since both sides are $\sum_{ij} A_{ij} B_{ij} C_{ij}$.

The two smallest cases take one line each. For the linear form, $d(\mathbf{a}^\top \mathbf{x}) = \mathbf{a}^\top d\mathbf{x} = \langle \mathbf{a}, d\mathbf{x} \rangle$, so $\nabla_{\mathbf{x}}\, \mathbf{a}^\top \mathbf{x} = \mathbf{a}$. For the quadratic form, the product rule hits both copies of $\mathbf{x}$:

$$d(\mathbf{x}^\top \mathbf{Ax}) = d\mathbf{x}^\top \mathbf{Ax} + \mathbf{x}^\top \mathbf{A}\, d\mathbf{x} = \mathbf{x}^\top \mathbf{A}^\top d\mathbf{x} + \mathbf{x}^\top \mathbf{A}\, d\mathbf{x} = \big\langle (\mathbf{A} + \mathbf{A}^\top)\mathbf{x},\ d\mathbf{x} \big\rangle.$$

The second equality transposes a 1×1 matrix: $d\mathbf{x}^\top \mathbf{Ax}$ is a number, so it equals its own transpose $\mathbf{x}^\top \mathbf{A}^\top d\mathbf{x}$. Reading off the coefficient, $\nabla_{\mathbf{x}}\, \mathbf{x}^\top \mathbf{Ax} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$, which is $2\mathbf{Ax}$ when $\mathbf{A}$ is symmetric.

Indices give a second route. Written entry by entry, the same computation is $\partial(\sum_{ij} x_i A_{ij} x_j)/\partial x_k = \sum_j A_{kj} x_j + \sum_i x_i A_{ik}$, the k-th entry of $(\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$ again. Indices are longer, but they leave no doubt about a transpose; the main text derives the matrix-product rule this way (Section 4.3).

A quick check of any matrix identity is to make every matrix 1×1: the identity must then reduce to the familiar scalar rule. The quadratic form passes, since $\frac{d}{dx}(ax^2) = 2ax = (a + a)x$. This 1×1 *check* catches a wrong coefficient or a missing term. It is blind to transposes and to the order of factors, because 1×1 matrices commute and equal their own transposes. The shape rule catches most of those errors, but unfortunately not when the

factor is square, which is the case the opening warned about. There only a numerical check helps. Let $\mathbf{A} = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$ and $\mathbf{x} = (1, 2)$. Then $\mathbf{x}^\top \mathbf{A}\mathbf{x} = x_1 x_2$, whose gradient $(x_2, x_1) = (2, 1)$ equals $(\mathbf{A} + \mathbf{A}^\top)\mathbf{x}$. The guess $2\mathbf{A}\mathbf{x} = (4, 0)$, which the scalar rule $2ax$ suggests, passes the 1×1 check, has the right shape, and is wrong.

The method handles matrix variables with no extra effort. For the least-squares loss $\ell = \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2$ with residual $\mathbf{r} = \mathbf{W}\mathbf{x} - \mathbf{y}$, we have $d\mathbf{r} = d\mathbf{W}\,\mathbf{x}$ when $\mathbf{W}$ varies, and the move rule puts $\mathbf{x}$ on the other side: $d\ell = 2\mathbf{r}^\top d\mathbf{r} = \langle 2\mathbf{r},\ d\mathbf{W}\,\mathbf{x}\rangle = \langle 2\mathbf{r}\mathbf{x}^\top,\ d\mathbf{W}\rangle$. Therefore

$$\nabla_{\mathbf{W}} \|\mathbf{W}\mathbf{x} - \mathbf{y}\|^2 = 2\,(\mathbf{W}\mathbf{x} - \mathbf{y})\,\mathbf{x}^\top, \tag{34.38}$$

a rank-one outer product of the residual with the input, with the shape of $\mathbf{W}$. When $\mathbf{x}$ varies instead, $d\mathbf{r} = \mathbf{W}\,d\mathbf{x}$ and $\nabla_{\mathbf{x}} \ell = 2\mathbf{W}^\top \mathbf{r}$. In both cases the gradient is twice the residual, multiplied by the transpose of the factor that multiplies the variable.

34.4.3.2 Backward Rules of a Layer

Inside a network, a layer $\mathbf{Y} = \mathbf{F}(\mathbf{X})$ is one step of a long computation that ends in a scalar loss ℓ. Its full Jacobian is too large to form: a 1000×1000 weight matrix applied to a minibatch of 1000 examples has 10^6 outputs and 10^6 weights, so the Jacobian of the outputs with respect to the weights has 10^{12} entries. The layer needs less. It turns the gradient of the loss with respect to its output into the gradients with respect to its inputs and weights. We write $\bar{\mathbf{Y}} = \nabla_{\mathbf{Y}} \ell$ for the adjoint of Section 34.3.2, now of a matrix: the total derivative of the loss with respect to $\mathbf{Y}$, arranged in the shape of $\mathbf{Y}$.

Proposition (backward rule of a layer). *Let $\mathbf{Y} = \mathbf{F}(\mathbf{X})$ be differentiable at $\mathbf{X}$ and let ℓ be a differentiable scalar function of $\mathbf{Y}$. Then the adjoint $\bar{\mathbf{X}}$ of the composite $\ell(\mathbf{F}(\mathbf{X}))$ is the unique matrix with*

$$\langle \bar{\mathbf{X}},\ d\mathbf{X}\rangle = \langle \bar{\mathbf{Y}},\ d\mathbf{Y}\rangle \quad \textit{for every } d\mathbf{X},$$

where $d\mathbf{Y}$ is the first-order change of $\mathbf{F}$ caused by $d\mathbf{X}$.

Proof. By the chain rule Equation 34.35 (with matrices read as long vectors), the linear part of the composite is the linear part of ℓ applied to the linear part of $\mathbf{F}$: $d\ell = \langle \bar{\mathbf{Y}}, d\mathbf{Y}\rangle$. Since $d\mathbf{Y}$ is linear in $d\mathbf{X}$, so is $d\ell$, and it can be written as $\langle \mathbf{M}, d\mathbf{X}\rangle$ for a unique $\mathbf{M}$. By Equation 34.37, $\mathbf{M} = \bar{\mathbf{X}}$. ■

In practice, applying the Proposition means applying the move rule. For a vector layer $d\mathbf{y} = \mathbf{J}\,d\mathbf{x}$, it gives $\langle \bar{\mathbf{y}}, \mathbf{J}\,d\mathbf{x}\rangle = \langle \mathbf{J}^\top \bar{\mathbf{y}}, d\mathbf{x}\rangle$, so $\bar{\mathbf{x}} = \mathbf{J}^\top \bar{\mathbf{y}}$: the backward rule of a layer is the transpose of its linearization. The transpose of a linear map is also called its adjoint, hence the name. Written as $\bar{\mathbf{x}}^\top = \bar{\mathbf{y}}^\top \mathbf{J}$, the rule is the left-to-right order of Equation 34.36 applied to one factor. A row vector times a Jacobian is called a *vector–Jacobian product*. The Proposition computes it for matrix variables without ever writing the Jacobian down.

The most used derivative in deep learning is that of a matrix product. The weights can stand on either side of the data, and since this section uses both layouts we derive them together. A minibatch $\mathbf{X} \in \mathbb{R}^{n \times d}$ with one example per row, the layout of the main text, passes through the weights $\mathbf{W} \in \mathbb{R}^{d \times q}$ to give $\mathbf{Y} = \mathbf{X}\mathbf{W}$. The product rule gives $d\mathbf{Y} = d\mathbf{X}\,\mathbf{W} + \mathbf{X}\,d\mathbf{W}$, and the move rule splits the inner product into one term per variable, $\langle \bar{\mathbf{Y}}, d\mathbf{X}\,\mathbf{W}\rangle + \langle \bar{\mathbf{Y}}, \mathbf{X}\,d\mathbf{W}\rangle = \langle \bar{\mathbf{Y}}\mathbf{W}^\top, d\mathbf{X}\rangle + \langle \mathbf{X}^\top \bar{\mathbf{Y}}, d\mathbf{W}\rangle$. Reading off both coefficients,

$$\bar{\mathbf{X}} = \bar{\mathbf{Y}}\,\mathbf{W}^\top, \qquad \bar{\mathbf{W}} = \mathbf{X}^\top \bar{\mathbf{Y}}. \tag{34.39}$$

For one example written as a column, $\mathbf{y} = \mathbf{W}\mathbf{x}$, the same computation gives

$$\bar{\mathbf{W}} = \bar{\mathbf{y}}\,\mathbf{x}^\top, \qquad \bar{\mathbf{x}} = \mathbf{W}^\top \bar{\mathbf{y}},$$

the pattern of Equation 34.38: the weight gradient is the outer product of the adjoint with the input. The two layouts are transposes of each other. With $\mathbf{x}_i^\top$ and $\bar{\mathbf{y}}_i^\top$ the rows of $\mathbf{X}$ and $\bar{\mathbf{Y}}$, the minibatch rule $\mathbf{X}^\top\bar{\mathbf{Y}} = \sum_i \mathbf{x}_i\bar{\mathbf{y}}_i^\top$ adds one such outer product per example.

The backward pass of a linear layer is therefore two matrix products, each of the size of the forward product: the forward pass costs ndq multiplications, the backward pass $2ndq$. The shapes check: $\bar{\mathbf{Y}}\mathbf{W}^\top$ is $n \times d$ like $\mathbf{X}$, and $\mathbf{X}^\top\bar{\mathbf{Y}}$ is $d \times q$ like $\mathbf{W}$. Rules for other matrix operations (inverses, determinants, decompositions) are derived the same way (Mike B. Giles 2008).

Two more layers complete a multilayer perceptron. An elementwise map $\mathbf{Y} = \varphi(\mathbf{X})$ has $d\mathbf{Y} = \varphi'(\mathbf{X}) \odot d\mathbf{X}$, so $\bar{\mathbf{X}} = \varphi'(\mathbf{X}) \odot \bar{\mathbf{Y}}$: the diagonal Jacobian of the previous part, applied as a mask. A bias $\mathbf{y} = \mathbf{z} + \mathbf{b}$ has $d\mathbf{y} = d\mathbf{b}$, so $\bar{\mathbf{b}} = \bar{\mathbf{y}}$. For a minibatch the bias is broadcast, $\mathbf{Z} = \mathbf{X}\mathbf{W} + \mathbf{1}\mathbf{b}^\top$, which is one more matrix product, and $\bar{\mathbf{b}} = \bar{\mathbf{Z}}^\top\mathbf{1}$ sums the adjoint over the examples. These three are the local rules from which Section 4.3 assembles the backward pass of a multilayer perceptron.

They also suffice to derive the matrix of the opening. Run the network of Figure 34.21 backward from $\bar{g} = 1$, one layer at a time, at $\mathbf{x} = (1, 1)$:

1. The output $g = \mathbf{w}_2^\top\mathbf{h}$ gives $\bar{\mathbf{h}} = \mathbf{w}_2 = (2, -1)$ and $\bar{\mathbf{w}}_2 = \mathbf{h} = (3, 0)$.
2. The ReLU gives $\bar{\mathbf{a}} = \mathbf{m} \odot \bar{\mathbf{h}} = (2, 0)$, and the bias $\bar{\mathbf{b}} = \bar{\mathbf{a}} = (2, 0)$.
3. The affine map $\mathbf{a} = \mathbf{W}_1\mathbf{x} + \mathbf{b}$ gives $\bar{\mathbf{W}}_1 = \bar{\mathbf{a}}\,\mathbf{x}^\top = \left(\begin{smallmatrix} 2 & 2 \\ 0 & 0 \end{smallmatrix}\right)$ and $\bar{\mathbf{x}} = \mathbf{W}_1^\top\bar{\mathbf{a}} = (2, 4)$.

The last line is the Jacobian row $(2, 4)$ of the chain-rule part, transposed, and $\bar{\mathbf{W}}_1$ is the matrix the opening found by nudging the weights. A framework differentiating `g` with respect to $\mathbf{W}_1$, $\mathbf{b}$ and $\mathbf{w}_2$ returns the same three arrays.

```
gW1, gb, gw2 = torch.func.grad(g, argnums=(1, 2, 3))(x, W1, b, w2)
h = torch.relu(W1 @ x + b)
show = lambda t: (t.numpy() + 0.0).tolist()     # + 0.0 turns -0.0 into 0.0
print('dg/dW1 = (w2*m) x^T:', show(torch.outer(w2 * m, x)),
      '  autograd:', show(gW1))
print('dg/db  = w2*m      :', show(w2 * m), '  autograd:', show(gb))
print('dg/dw2 = h         :', show(h), '  autograd:', show(gw2))
```

```
dg/dW1 = (w2*m) x^T: [[2.0, 2.0], [0.0, 0.0]]   autograd: [[2.0, 2.0], [0.0, 0.0]]
dg/db  = w2*m      : [2.0, 0.0]   autograd: [2.0, 0.0]
dg/dw2 = h         : [3.0, 0.0]   autograd: [3.0, 0.0]
```

What deserves a look is where the zeros are. The inactive unit shows up three times: a zero row in $\bar{\mathbf{W}}_1$, a zero in $\bar{\mathbf{b}}$, and a zero in $\bar{\mathbf{w}}_2$, because its output $h_2 = 0$ is what $w_{2,2}$ multiplies. A training step on this input leaves all four of the unit's parameters unchanged.

We still owe the numerical check that a transposed square factor calls for. A check that cannot fail tells us nothing, so we build this one to be able to: every factor below is square and random, hence not symmetric, and each rule is printed beside a plausible wrong guess

of the same shape. We test the quadratic form and the matrix product. For the product we differentiate the scalar $\langle \bar{\mathbf{Y}}, \mathbf{XW} \rangle$ for a random $\bar{\mathbf{Y}}$, whose gradients are, by the Proposition, $\bar{\mathbf{X}}$ and $\bar{\mathbf{W}}$.

```
A, x = torch.tensor(np.random.randn(3, 3)), torch.tensor(np.random.randn(3))
X, W, Ybar = (torch.tensor(np.random.randn(3, 3)) for _ in range(3))  # square
ok = lambda u, v: torch.allclose(u, v, atol=1e-5)
grad = torch.func.grad
gx = grad(lambda x: x @ A @ x)(x)
gX, gW = grad(lambda X, W: (Ybar * (X @ W)).sum(), argnums=(0, 1))(X, W)
print('grad x^T A x: (A + A^T) x', ok(gx, (A + A.T) @ x),
      '| wrong guess 2 A x       ', ok(gx, 2 * A @ x))
print('Xbar        : Ybar W^T   ', ok(gX, Ybar @ W.T),
      '| wrong guess Ybar W      ', ok(gX, Ybar @ W))
print('Wbar        : X^T Ybar   ', ok(gW, X.T @ Ybar),
      '| wrong guess (X^T Ybar)^T', ok(gW, (X.T @ Ybar).T))
```

```
grad x^T A x: (A + A^T) x True | wrong guess 2 A x        False
Xbar        : Ybar W^T    True | wrong guess Ybar W       False
Wbar        : X^T Ybar    True | wrong guess (X^T Ybar)^T False
```

The rules print `True` and the wrong guesses `False`. The design of the test matters more than its outcome. With rectangular factors, say $\mathbf{X}$ of shape 3×4 and $\mathbf{W}$ of shape 4×2, the two wrong guesses for the product would not even run: a misplaced transpose raises a shape error before any number is compared, which is the shape rule doing its work. With a symmetric $\mathbf{A}$ the guess $2\mathbf{Ax}$ would have been right, and the test would have passed a wrong rule. The dangerous case is the square factor that is not symmetric, so that is what a test must draw.

The matrix-product rule also handles a loss in which both factors of a product are unknown, such as the factorization loss $\|\mathbf{X} - \mathbf{UV}\|_F^2$ of exercise 4. In every case the backward rule multiplies the incoming adjoint by the transposes of whatever multiplied the variable on the way forward. The rules for the product and the activation also use forward values: $\mathbf{X}$ and $\mathbf{W}$ for the matrix product, $\varphi'(\mathbf{X})$ for the activation. The bias rule uses none. A backward pass must keep these values from the forward pass, and that memory is a cost which Section 34.5 measures.

34.4.3.3 Softmax and Cross-Entropy

The derivative a classifier evaluates most often is that of softmax followed by the cross-entropy loss. Softmax turns a vector of scores $\mathbf{z}$, the *logits*, into probabilities: with $S = \sum_k e^{z_k}$, $p_i = e^{z_i}/S$. The quotient rule that gave Equation 34.22 for two logits gives the Jacobian of softmax for any number of them. With $\delta_{ij} = 1$ if $i = j$ and 0 otherwise,

$$\frac{\partial p_i}{\partial z_j} = \frac{\delta_{ij} e^{z_i} S - e^{z_i} e^{z_j}}{S^2} = p_i(\delta_{ij} - p_j), \qquad \mathbf{J}_{\text{softmax}} = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top. \tag{34.40}$$

Raising logit i raises p_i at rate $p_i(1 - p_i)$ and lowers every other p_j at rate $p_i p_j$. One check: $\mathbf{J}\mathbf{1} = \mathbf{p} - \mathbf{p}(\mathbf{p}^\top \mathbf{1}) = \mathbf{0}$ because the probabilities sum to one. This must be so, since adding the same constant to every logit leaves $\mathbf{p}$ unchanged. The Jacobian is singular.

$\mathbf{J}$ is also symmetric, which a Jacobian need not be. The reason is that it is a Hessian. Softmax is the gradient of the log-sum-exp function $\mathrm{LSE}(\mathbf{z}) = \log \sum_k e^{z_k}$, the chapter's running function, because $\partial\,\mathrm{LSE}/\partial z_i = e^{z_i}/S = p_i$, and the Jacobian of a gradient is the Hessian. What this section adds to Equation 34.22 is a way to read the matrix. It is the covariance matrix of a one-hot vector drawn with probabilities $\mathbf{p}$, so for every $\mathbf{z}$ and $\mathbf{v}$ the number $\mathbf{v}^\top \mathbf{J}\mathbf{v} = \sum_i p_i v_i^2 - \left(\sum_i p_i v_i\right)^2$ is the variance of the entries of $\mathbf{v}$ under the weights $\mathbf{p}$. A variance is nonnegative: the Hessian is positive semidefinite everywhere and log-sum-exp is convex. A constant $\mathbf{v}$ has no variance, which is the null vector $\mathbf{1}$ again.

Now compose softmax with the cross-entropy loss $\ell = -\sum_i y_i \log p_i$ against a target distribution $\mathbf{y}$. The chain rule multiplies the row $\partial \ell / \partial \mathbf{p} = (-y_i/p_i)_i$ by Equation 34.40, and the algebra collapses:

$$\frac{\partial \ell}{\partial z_j} = -\sum_i \frac{y_i}{p_i}\, p_i(\delta_{ij} - p_j) = -y_j + p_j \sum_i y_i = p_j - y_j, \qquad \text{that is,} \qquad \nabla_{\mathbf{z}} \ell = \mathbf{p} - \mathbf{y}. \tag{34.41}$$

The gradient with respect to the logits is the predicted probability minus the target, the formula that Section 3.1.2.2 writes as $\hat{\mathbf{y}} - \mathbf{y}$. The last step used only $\sum_i y_i = 1$, so the result holds for any probability vector $\mathbf{y}$, not only one-hot labels: label smoothing and distillation targets give the same formula. For a target that does not sum to one the gradient is $\left(\sum_i y_i\right)\mathbf{p} - \mathbf{y}$. At $\mathbf{z} = (1, -0.5, 2)$, $\mathbf{p} = (0.254,\ 0.057,\ 0.690)$. With the second class as the target, the gradient is $(0.254,\ -0.943,\ 0.690)$.

Libraries fuse softmax and cross-entropy into one primitive for numerical reasons (Section 3.4.2, Section 35.8.2). The last cell computes the softmax Jacobian at $\mathbf{z}$ by automatic differentiation and prints its eigenvalues, which the variance argument says are nonnegative, with one of them zero. It then differentiates the fused loss for the target above and for the two-hot target $\mathbf{y} = (1, 1, 0)$, which does not sum to one.

```
z = torch.tensor([1.0, -0.5, 2.0])
p = torch.softmax(z, 0)
J = torch.func.jacrev(lambda z: torch.softmax(z, 0))(z)   # the Jacobian, by autograd
print('eigenvalues of J:', torch.linalg.eigvalsh(J).numpy().round(3) + 0.0)
loss = lambda z, y: -(y * torch.log_softmax(z, 0)).sum()
for y in [torch.tensor([0.0, 1.0, 0.0]), torch.tensor([1.0, 1.0, 0.0])]:
    print(f'y = {y.numpy()}: d loss / d z = {torch.func.grad(loss)(z, y).numpy().round(4)}, '
          f'p - y = {(p - y).numpy().round(4)}')
```

```
eigenvalues of J: [0.    0.079 0.378]
y = [0. 1. 0.]: d loss / d z = [ 0.2537 -0.9434  0.6897], p - y = [ 0.2537 -0.9434  0.6897]
y = [1. 1. 0.]: d loss / d z = [-0.4926 -0.8868  1.3793], p - y = [-0.7463 -0.9434  0.6897]
```

The eigenvalues are 0, 0.079 and 0.378: one zero, for the direction $\mathbf{1}$, and two positive ones. For the one-hot target the framework's gradient of the fused loss is $\mathbf{p} - \mathbf{y} = (0.2537, -0.9434, 0.6897)$, the vector found by hand. For the two-hot target it returns $(-0.4926, -0.8868, 1.3793)$, which is $2\mathbf{p} - \mathbf{y}$ and not the $\mathbf{p} - \mathbf{y}$ printed beside it. "Prediction minus target" is a property of targets that sum to one, not of the loss.

The product $\mathbf{J}_K \cdots \mathbf{J}_1$ was multiplied by hand here, for a chain of three factors. A program has to do it for a computation that is not a chain, keep the forward values the backward rules

need, and do it at a cost that stays a small multiple of evaluating the network, however many parameters it has. That program is automatic differentiation, the subject of Section 34.5.

34.4.4 Summary

The Jacobian of a map $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ is the $m \times n$ matrix of its best linear approximation, with entries $[\mathbf{J_f}]_{ij} = \partial f_i / \partial x_j$; the gradient (one row, written as a column) and the Hessian (the Jacobian of the gradient) are its special cases. A network's derivative with respect to its input is the product of its layers' Jacobians, $\mathbf{J}_K \cdots \mathbf{J}_1$, and the order of multiplication decides the cost: three million against two billion multiplications for three layers of width 1000. The rule $df = \langle \nabla f, d\mathbf{x} \rangle$ derives every gradient a layer needs in the shape of its variable, including the backward rule $\bar{\mathbf{X}} = \bar{\mathbf{Y}}\mathbf{W}^\top$, $\bar{\mathbf{W}} = \mathbf{X}^\top \bar{\mathbf{Y}}$ of a matrix product and the collapse of softmax with cross-entropy to $\mathbf{p} - \mathbf{y}$. The shape rule and the 1×1 check catch most errors of a hand derivation, but not a transposed square factor; that takes a numerical test on a matrix that is not symmetric.

34.4.5 Exercises

1. **Jacobian of a ReLU network.** Let $\mathbf{g}(\mathbf{x}) = \mathbf{W}_2\, \varphi(\mathbf{W}_1 \mathbf{x} + \mathbf{b})$ with $\mathbf{W}_1 \in \mathbb{R}^{h \times n}$, $\mathbf{b} \in \mathbb{R}^h$, $\mathbf{W}_2 \in \mathbb{R}^{m \times h}$, and the elementwise ReLU φ. Assume that no entry of the pre-activation $\mathbf{W}_1 \mathbf{x} + \mathbf{b}$ is zero. By Equation 34.35, $\mathbf{J_g}(\mathbf{x}) = \mathbf{W}_2 \mathbf{M}(\mathbf{x}) \mathbf{W}_1$, where the diagonal matrix $\mathbf{M}(\mathbf{x})$ has the entry 1 for each active hidden unit and 0 for each inactive one.
    1. Show that $\mathbf{g}(\mathbf{x}+\delta) = \mathbf{g}(\mathbf{x}) + \mathbf{J_g}(\mathbf{x})\, \delta$ holds exactly for every sufficiently small δ, so that the remainder in Equation 34.31 vanishes identically. For the 2-2-1 network of this section at $\mathbf{x} = (1, 1)$, find the largest r such that this holds for all δ with $\|\delta\|_\infty < r$. For $n = m = 2$, how does the image of a small circle under $\mathbf{g}$ differ from the one in Figure 34.20?
    2. Let k hidden units be active at $\mathbf{x}$. Show that $\operatorname{rank} \mathbf{J_g}(\mathbf{x}) \le k$. For $n = m = 2$ and $k = 1$, what is the image of a small circle around $\mathbf{x}$?
    3. Describe the set of inputs at which some pre-activation is zero. What happens to $\mathbf{J_g}$ when $\mathbf{x}$ crosses this set?

2. code **Area under the Jacobian.** The running map $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ carries the circle of radius r around $\mathbf{x}_0 = (0.5, 0.5)$ to the blue curves of Figure 34.20.
    1. Show that the ellipse $\mathbf{f}(\mathbf{x}_0) + \mathbf{J}\delta$, $\|\delta\| = r$, has semi-axes $r\sigma_1$ and $r\sigma_2$ along the left singular vectors of $\mathbf{J} = \mathbf{J_f}(\mathbf{x}_0)$, and area $\pi r^2 |\det \mathbf{J}|$.
    2. Compute the area enclosed by the true image of the circle (shoelace formula on 2,000 points) for $r = 0.35$ and $r = 0.175$, and divide it by $\pi r^2 |\det \mathbf{J}|$. Is the ratio above or below 1, and how fast does it approach 1 as r shrinks? Explain both with the average of $|\det \mathbf{J_f}|$ over the disk.
    3. At which points does the running map fail to be locally invertible? Solve $\det \mathbf{J_f}(x, y) = 0$.

3. code **Cheapest chain order.** ⊠ By Equation 34.36 the Jacobian of a chain of K maps is $\mathbf{J}_K \cdots \mathbf{J}_1$ with $\mathbf{J}_k \in \mathbb{R}^{d_k \times d_{k-1}}$. Multiplying a $p \times q$ matrix by a $q \times r$ matrix costs pqr multiplications.
    1. Give the costs of the forward order, which multiplies from the right, and of the reverse order, which multiplies from the left, as functions of $d_0, \dots, d_K$. Evaluate both for $(d_0, \dots, d_4) = (1000, 10, 1000, 10, 1000)$, a map whose layers alternate

between wide and narrow, and compare with the section's count for $(1 \times h)(h \times h)(h \times h)(h \times n)$.
2. Write a dynamic program over the sub-chains $\mathbf{J}_j \cdots \mathbf{J}_i$ that finds the cheapest parenthesization, and run it on the dimensions of (a). Which order does it choose, and how much cheaper is it than the better of the two orders of (a)?
3. For a scalar output, $d_K = 1$, compare the optimum with the reverse order on a few thousand random dimension vectors, with K between 2 and 6 and each of $d_0, \dots, d_{K-1}$ drawn uniformly from $\{1, \dots, 10\}$. Characterize the vectors for which some other order is cheaper.

4. code **Backward rules of a matrix product.**

1. For $\mathbf{Y} = \mathbf{XW}$, derive Equation 34.39 by the differential method, then derive it again by indices to confirm the transposes.
2. Verify both rules with your framework's vector–Jacobian product (`torch.func.vjp`, `jax.vjp`, `tape.gradient(..., output_gradients=...)` in TensorFlow, `backward(head_grad)` in MXNet) on $\mathbf{X} \in \mathbb{R}^{3\times4}$, $\mathbf{W} \in \mathbb{R}^{4\times2}$ and a random adjoint $\bar{\mathbf{Y}}$. Count the multiplications of the forward product and of the two backward products.
3. For the factorization loss $\|\mathbf{X} - \mathbf{UV}\|_F^2$ with $\mathbf{R} = \mathbf{X} - \mathbf{UV}$, check $\nabla_{\mathbf{U}} = -2\mathbf{RV}^\top$ and $\nabla_{\mathbf{V}} = -2\mathbf{U}^\top\mathbf{R}$ numerically. Then write the alternating least-squares step that sets one gradient to zero with the other factor fixed, and show that each step is one linear solve.

5. code **Gradient of softmax regression.** Softmax regression of Section 3.3 computes the logits $\mathbf{Z} = \mathbf{XW} + \mathbf{1b}^\top$ of a minibatch $\mathbf{X} \in \mathbb{R}^{n\times d}$, the row-wise softmax $\mathbf{P}$, and the loss $\ell = -\frac{1}{n}\sum_{ij} Y_{ij} \log P_{ij}$ with one-hot rows $\mathbf{Y}$.

1. Using Equation 34.41 row by row and the matrix-product rule Equation 34.39, derive $\bar{\mathbf{Z}} = (\mathbf{P} - \mathbf{Y})/n$, $\nabla_{\mathbf{W}}\ell = \mathbf{X}^\top(\mathbf{P} - \mathbf{Y})/n$ and $\nabla_{\mathbf{b}}\ell = (\mathbf{P} - \mathbf{Y})^\top\mathbf{1}/n$. *Hint:* $\mathbf{1b}^\top$ is a matrix product too.
2. Check all three against your framework's automatic differentiation on a random 6×4 matrix $\mathbf{X}$, a random 4×3 matrix $\mathbf{W}$ and random one-hot rows $\mathbf{Y}$.
3. Count the multiplications of the forward pass and of your gradient formulas, and state the ratio.

6. **Softmax vector–Jacobian product.** A backward pass never forms the softmax Jacobian Equation 34.40. It maps the adjoint $\mathbf{u} = \bar{\mathbf{p}}$ of the probabilities to the adjoint $\bar{\mathbf{z}} = \mathbf{J}_{\text{softmax}}^\top\mathbf{u}$ of the logits.

1. Show that $\mathbf{J}_{\text{softmax}}^\top\mathbf{u} = \mathbf{p} \odot \big(\mathbf{u} - (\mathbf{p}^\top\mathbf{u})\,\mathbf{1}\big)$, where $\odot$ is the elementwise product. For n classes, count the operations of this formula and those of forming $\mathbf{J}_{\text{softmax}}$ and multiplying it by $\mathbf{u}$.
2. Self-attention applies softmax to each row of an $n \times n$ score matrix $\mathbf{S}$ and produces the weights $\mathbf{P}$. How many entries does the Jacobian of $\mathbf{P}$ with respect to $\mathbf{S}$ have, and how many of them can be nonzero? For $n = 4096$ tokens in float32, compare the memory of the full Jacobian, of its nonzero entries, and of the adjoint of $\mathbf{S}$ that the formula of (a), applied row by row, returns.

7. code **The LayerNorm Jacobian.** Layer normalization (Ba et al. 2016) maps $\mathbf{x} \in \mathbb{R}^n$ to $\hat{\mathbf{x}} = (\mathbf{x} - \mu\mathbf{1})/\sigma$ with $\mu = \frac{1}{n}\sum_i x_i$ and $\sigma^2 = \frac{1}{n}\sum_i (x_i - \mu)^2 > 0$ (no learned scale and no ϵ).

1. Show by the differential method that $\mathbf{J} = \frac{1}{\sigma}\big(\mathbf{I} - \frac{1}{n}\mathbf{1}\mathbf{1}^\top - \frac{1}{n}\hat{\mathbf{x}}\hat{\mathbf{x}}^\top\big)$. *Hint:* $d\mu =$

$\frac{1}{n}\mathbf{1}^\top d\mathbf{x}$ and $d\sigma = \frac{1}{n}\hat{\mathbf{x}}^\top d\mathbf{x}$.

2. Show that $\mathbf{1}$ and $\hat{\mathbf{x}}$ are orthogonal and that $\|\hat{\mathbf{x}}\|^2 = n$. Conclude that $\sigma\mathbf{J}$ is the orthogonal projection onto the complement of $\operatorname{span}\{\mathbf{1}, \hat{\mathbf{x}}\}$ and that $\operatorname{rank}\mathbf{J} = n-2$ for $n \geq 2$. Which two changes of the input can LayerNorm not see, and why?
3. Check (a) and the two null vectors against your framework's Jacobian for a random $\mathbf{x} \in \mathbb{R}^5$, and print the eigenvalues of $\mathbf{J}$.

8. **Log-determinant gradient.** ▨ A normalizing flow (Rezende and Mohamed 2015) passes a random vector $\mathbf{X}$ through invertible layers. For one layer $\mathbf{Y} = \mathbf{g}(\mathbf{X})$, the log-density of $\mathbf{Y}$ at $\mathbf{g}(\mathbf{x})$ is $\log p_{\mathbf{X}}(\mathbf{x}) - \log|\det \mathbf{J}_{\mathbf{g}}(\mathbf{x})|$, by the change-of-variables formula for densities. For a linear layer $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ the correction is $-\log|\det \mathbf{A}|$, so maximum-likelihood training differentiates $\log|\det \mathbf{A}|$ with respect to $\mathbf{A}$.

   1. Expand $\det(\mathbf{I}+t\mathbf{M})$ to first order in t and use $\det(\mathbf{A}+d\mathbf{A}) = \det\mathbf{A}\,\det(\mathbf{I}+\mathbf{A}^{-1}d\mathbf{A})$ to derive Jacobi's formula $d\det\mathbf{A} = \det\mathbf{A}\,\operatorname{tr}(\mathbf{A}^{-1}d\mathbf{A})$ for invertible $\mathbf{A}$. Conclude with Equation 34.37 that $\nabla_{\mathbf{A}}\log|\det\mathbf{A}| = \mathbf{A}^{-\top}$. Why does the sign of $\det\mathbf{A}$ not matter?
   2. Check the identity on a random invertible 3×3 matrix by central differences of `np.linalg.slogdet`.
   3. How does the norm of the gradient behave as $\mathbf{A}$ approaches a singular matrix? What does this imply for a flow layer whose weight matrix is nearly singular?

9. code **Gradient through a linear solve.** Let $f(\mathbf{A}) = \mathbf{c}^\top\mathbf{A}^{-1}\mathbf{b}$ for invertible $\mathbf{A} \in \mathbb{R}^{n\times n}$ and fixed vectors $\mathbf{b}$, $\mathbf{c}$.

   1. Differentiate $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$ to show $d(\mathbf{A}^{-1}) = -\mathbf{A}^{-1}\,d\mathbf{A}\,\mathbf{A}^{-1}$.
   2. Show that $\nabla_{\mathbf{A}} f = -\mathbf{v}\mathbf{u}^\top$ with $\mathbf{u} = \mathbf{A}^{-1}\mathbf{b}$ and $\mathbf{v} = \mathbf{A}^{-\top}\mathbf{c}$, so the gradient costs two linear solves and no inverse. When does it reduce to $-\mathbf{u}\mathbf{u}^\top$?
   3. Check (b) for a random 3×3 matrix $\mathbf{A}$ (add $3\mathbf{I}$ to keep it well conditioned) against your framework's gradient of $\mathbf{c}^\top$ times the solution of $\mathbf{A}\mathbf{z} = \mathbf{b}$, and against the directional derivative $\langle \nabla_{\mathbf{A}} f, \mathbf{E}\rangle$ computed by central differences for a random $\mathbf{E}$.
   4. For $n = 10^4$ and a sparse $\mathbf{A}$ with 10^5 nonzeros, compare the cost of (b) with that of forming $\mathbf{A}^{-1}$.

   Adapted from Alan Edelman and Steven G. Johnson, Matrix Calculus for Machine Learning and Beyond, problem set 2, MIT 18.S096, IAP 2023, Problem 1.

10. **Gradient of the spectral norm.** ▨ Spectral normalization (Miyato et al. 2018) divides each weight matrix $\mathbf{W}$ of a network by its largest singular value $\sigma_1(\mathbf{W})$, so training differentiates σ_1.

    1. Let $\mathbf{S}$ be a real symmetric matrix with a simple eigenvalue λ and unit eigenvector $\mathbf{q}$, and perturb it to $\mathbf{S} + t\mathbf{E}$ with $\mathbf{E}$ symmetric. Take as given that $\lambda(t)$ and $\mathbf{q}(t)$, defined by $(\mathbf{S}+t\mathbf{E})\mathbf{q} = \lambda\mathbf{q}$ and $\mathbf{q}^\top\mathbf{q} = 1$, are differentiable near $t = 0$ (the implicit function theorem gives this because λ is simple). Differentiate both equations at $t = 0$ to show the Hellmann–Feynman theorem $\lambda'(0) = \mathbf{q}^\top\mathbf{E}\,\mathbf{q}$, and give $\nabla_{\mathbf{S}}\lambda$ for the inner product $\operatorname{tr}(\mathbf{A}^\top\mathbf{B})$.
    2. Apply (a) to $\sigma_1(\mathbf{W})^2 = \lambda_{\max}(\mathbf{W}^\top\mathbf{W})$ to show that $\nabla_{\mathbf{W}}\sigma_1 = \mathbf{u}_1\mathbf{v}_1^\top$ when $\sigma_1 > 0$ is simple, where $\mathbf{u}_1$ and $\mathbf{v}_1$ are the top left and right singular vectors. Why is $\sigma_1 > 0$ needed? Consider $\mathbf{W} = \mathbf{w} \in \mathbb{R}^{n\times 1}$ near $\mathbf{w} = \mathbf{0}$. Check the formula by central differences on a random 5×3 matrix in NumPy.
    3. Compute $\sigma_1(\mathbf{I} + t\,\operatorname{diag}(1,-1))$ for small t of both signs. Is σ_1 differentiable at the 2×2 identity, and which hypothesis of (a) fails there?

Adapted from Alan Edelman and Steven G. Johnson, Matrix Calculus for Machine Learning and Beyond, problem set 2, MIT 18.S096, IAP 2023, Problem 3.

34.5 Automatic Differentiation

In the Basics part, every gradient came from one call to the framework's automatic differentiation. The previous section showed what that call computes: the derivative of a network is a product of Jacobians, one per layer, and the order of multiplication decides the cost. A program, however, is a sequence of additions, multiplications and function calls, not a list of matrices. How does the framework find the factors from the code, and how does it multiply them in the cheap order?

How much the cheap order saves is easiest to see on a large network. A network with 10^8 parameters computes one number, its loss ℓ, and every training step needs the derivative of ℓ with respect to each of the 10^8 parameters. The direct method nudges one parameter, reruns the network, and divides the change in ℓ by the nudge. That is 10^8 runs per step. At 10 ms per run, one gradient takes 10^6 s, about 12 days, and the size of the nudge adds an error of its own. Backpropagation returns all 10^8 derivatives, correct up to rounding, for the cost of about three runs, roughly 30 ms. This section explains where the factor of 10^8 goes.

The short answer is that the product of Jacobians can be evaluated from either end. Starting from the input end, one pass carries one input direction, so 10^8 inputs need 10^8 passes. Starting from the loss end, one pass carries one output direction, and a loss has a single output.

Section 1.5 showed how to call both products in a framework. Here we build them ourselves in a few lines of Python, prove that they return derivatives, trace the backward pass by hand, and count its cost. Then we combine the two products to compute curvature along one direction and to differentiate the solution of an equation without differentiating the solver. Two results are worth remembering above the rest: arithmetic on dual numbers returns derivatives (the one proposition of the section), and a sweep costs a small multiple of one run of the program, forward mode needing one sweep per input direction and reverse mode one per output direction. The last part, Section 34.5.3, can wait for a second reading. Three small examples recur: the chapter's log-sum-exp $\ell(\mathbf{x}) = \log \sum_j e^{x_j}$, whose gradient is the softmax; the plane map $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ from the previous section; and the expression $y = (uv + u)^2$, which is small enough to trace by hand.

We use the following imports throughout this section.

```
import numpy as np
import torch
```

34.5.1 Forward Mode and Dual Numbers

34.5.1.1 Dual Numbers

We want the value of a program and its derivative from a single run, with no step size to choose and no formula for f' written down. The device is a small extension of the real numbers. Adjoin a symbol ε that is not zero but whose square is,

$$\varepsilon^2 = 0, \qquad \varepsilon \neq 0, \tag{34.42}$$

and compute with expressions $a + b\varepsilon$, called *dual numbers*, by the usual rules of algebra. Every differentiation rule of Section 34.1 discards second-order terms, and the rule $\varepsilon^2 = 0$ makes that step part of the arithmetic.

Let us see what the rule does to $3 + \varepsilon$. Squaring and cubing it gives

$$(3+\varepsilon)^2 = 9 + 6\varepsilon + \varepsilon^2 = 9 + 6\varepsilon, \qquad (3+\varepsilon)^3 = (9+6\varepsilon)(3+\varepsilon) = 27 + 27\varepsilon.$$

The ε-parts 6 and 27 are the derivatives of x^2 and x^3 at $x = 3$. No differentiation rule was applied; the derivatives came out of multiplication. The same happens for any product. Multiplying out and dropping the ε^2 term gives

$$(a + b\varepsilon)(c + d\varepsilon) = ac + (ad + bc)\,\varepsilon. \tag{34.43}$$

If b and d are the derivatives of two quantities whose values are a and c, then $ad + bc$ is the derivative of their product. Multiplication of dual numbers performs the product rule.

A reader may object that no number is nonzero with a zero square. No real number is, but a matrix can be. Take $\varepsilon = \left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)$. Then $\varepsilon^2 = \mathbf{0}$, the dual number $a + b\varepsilon$ is the matrix $\left(\begin{smallmatrix} a & b \\ 0 & a \end{smallmatrix}\right)$, and the matrix product $\left(\begin{smallmatrix} a & b \\ 0 & a \end{smallmatrix}\right)\left(\begin{smallmatrix} c & d \\ 0 & c \end{smallmatrix}\right) = \left(\begin{smallmatrix} ac & ad+bc \\ 0 & ac \end{smallmatrix}\right)$ is Equation 34.43. Dual numbers are as concrete as 2×2 matrices, just as complex numbers are as concrete as the rotations and scalings of the plane.

Programs also call functions such as sin, exp, and log. For a function φ given by its Taylor series around a, substituting $a+b\varepsilon$ into the series removes every term from $(b\varepsilon)^2$ on, which leaves

$$\varphi(a + b\varepsilon) = \varphi(a) + \varphi'(a)\, b\, \varepsilon.$$

So $e^{a+b\varepsilon} = e^a + e^a b\,\varepsilon$ and $\sin(a + b\varepsilon) = \sin a + (\cos a)\, b\,\varepsilon$ are forced by the algebra. For a primitive that is only differentiable at a, we take the display as the definition of its action on dual numbers. In both cases the ε-part follows the chain rule: the derivative b of the argument is multiplied by $\varphi'(a)$.

Proposition (dual numbers compute derivatives). *Let f be an expression built from constants, the identity, sums, products, and primitives φ (such as* sin*,* exp*,* log*). Let each primitive be differentiable at the argument it receives when f is evaluated at x (for* log*, a positive argument). Then evaluating the same expression on the dual number $x + \varepsilon$ gives*

$$f(x + \varepsilon) = f(x) + f'(x)\,\varepsilon. \tag{34.44}$$

Proof. We use induction on the expression. A constant c becomes $c + 0\varepsilon$ and the identity becomes $x + 1\varepsilon$; their ε-parts are their derivatives 0 and 1. Suppose two subexpressions satisfy $g(x + \varepsilon) = g(x) + g'(x)\varepsilon$ and $h(x + \varepsilon) = h(x) + h'(x)\varepsilon$. Their sum has ε-part $g' + h' = (g + h)'$. By Equation 34.43, their product has ε-part $g'h + gh' = (gh)'$. A primitive applied to g has, by the rule above with $a = g(x)$ and $b = g'(x)$, the ε-part

$\varphi'(g(x))\,g'(x) = (\varphi \circ g)'(x)$. Each way of building a larger expression from smaller ones preserves the property, so the whole expression has it. ■

To differentiate a program, we therefore run it on $x + \varepsilon$ and read the ε-part of the output. A program with loops and branches is covered as well: for a given input it executes one particular sequence of operations, which is an expression of this kind, and the dual run differentiates that sequence (Section 1.5.3.1 shows this for the frameworks). The input's ε-part, here 1, is called the *seed*. No formula for f' appears and no step size is chosen; the only error is the rounding of the arithmetic itself.

34.5.1.2 A Forward-Mode Engine

Operator overloading turns the algebra into code. The class below stores the value a and the derivative part b, which we call the *tangent* and write $\dot{v}$ for a quantity v. It implements Equation 34.43 for $*$ and gives sin, exp, and log the primitive rule. A class counter `ops` records how many primitive operations have run; we use it to measure cost in Section 34.5.2.4.

```
class Dual:
    """A dual number a + b*eps with eps^2 = 0; b carries the derivative."""
    ops = 0                                          # primitive operations run
    def __init__(self, a, b=0.0):
        self.a, self.b = a, b                        # value, tangent
    def __add__(self, o):
        o = o if isinstance(o, Dual) else Dual(o)
        Dual.ops += 1
        return Dual(self.a + o.a, self.b + o.b)
    __radd__ = __add__
    def __mul__(self, o):
        o = o if isinstance(o, Dual) else Dual(o)
        Dual.ops += 1
        return Dual(self.a * o.a, self.a * o.b + self.b * o.a)   # product rule
    __rmul__ = __mul__
    def sin(self):
        Dual.ops += 1
        return Dual(np.sin(self.a), np.cos(self.a) * self.b)     # chain rule
    def exp(self):
        Dual.ops += 1
        return Dual(np.exp(self.a), np.exp(self.a) * self.b)
    def log(self):
        Dual.ops += 1
        return Dual(np.log(self.a), self.b / self.a)
    def __repr__(self):
        return f'{np.round(self.a, 6)} + {np.round(self.b, 6)} eps'
```

A first run, on $f(x) = \sin(x^2) + e^x$ at $1.3 + \varepsilon$, with the hand derivative $f'(x) = 2x\cos(x^2) + e^x$ printed beside the ε-part:

```
def f(x):
    return (x * x).sin() + x.exp()                   # sin(x^2) + e^x, in dual algebra

x0 = 1.3
```

```
out = f(Dual(x0, 1.0))                              # seed the derivative with 1
exact = 2 * x0 * np.cos(x0**2) + np.exp(x0)         # f'(x) by hand
print('dual result :', out)
print('value f(x)  :', np.sin(x0**2) + np.exp(x0))
print("derivative  : dual %.6f  vs  exact %.6f" % (out.b, exact))
```

```
dual result : 4.6622 + 3.360101 eps
value f(x)  : 4.662200318713363
derivative  : dual 3.360101  vs  exact 3.360101
```

One run returns the value 4.6622 and the derivative 3.360101.

A function of one variable hides the weakness of the method. Log-sum-exp has three inputs at $\mathbf{x} = (1, -0.5, 2)$, and its gradient is the softmax, $\partial \ell / \partial x_j = e^{x_j} / \sum_k e^{x_k} = p_j$. How many runs does forward mode need for this gradient? Seeding input j with 1 and the other inputs with 0 makes the ε-part of the output the partial derivative $\partial \ell / \partial x_j$, and nothing more.

```
def lse_dual(xs):                                   # log-sum-exp, in dual algebra
    total = 0.0
    for x in xs:
        total = total + x.exp()
    return total.log()

x_lse = [1.0, -0.5, 2.0]
p = np.exp(x_lse) / np.exp(x_lse).sum()             # softmax, the known gradient
for j in range(3):                                  # one pass per input direction
    out = lse_dual([Dual(x, 1.0 if k == j else 0.0) for k, x in enumerate(x_lse)])
    print(f'seed e{j + 1}: value {out.a:.6f}, partial {out.b:.6f}, '
          f'softmax {p[j]:.6f}')
```

```
seed e1: value 2.371539, partial 0.253716, softmax 0.253716
seed e2: value 2.371539, partial 0.056612, softmax 0.056612
seed e3: value 2.371539, partial 0.689672, softmax 0.689672
```

It needs three. Each pass returns the value 2.371539 again and a single softmax entry, 0.253716, 0.056612, or 0.689672. A gradient over n inputs therefore costs n passes. For the network of the opening, with $n = 10^8$, that is again 10^8 runs of the program. Forward mode removes the step size but not the number of runs.

34.5.1.3 Vector Tangents and Jacobian Columns

A map $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^m$ has an $m \times n$ Jacobian $\mathbf{J}$. What does one forward pass compute for it? Seed the inputs with a direction $\mathbf{v}$, so that input j is $x_j + v_j \varepsilon$. Applying the proposition to the function $t \mapsto f_i(\mathbf{x} + t\mathbf{v})$ shows that output i comes out as $f_i(\mathbf{x}) + (\nabla f_i \cdot \mathbf{v})\,\varepsilon$, so the ε-parts of the outputs form the vector $\mathbf{Jv}$. This product is the *Jacobian–vector product* (JVP). With $\mathbf{v} = \mathbf{e}_j$ it is column j of $\mathbf{J}$. One pass carries one input direction.

Take the plane map $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ at $(0.5, 0.5)$. Its Jacobian, row i holding the partial derivatives of output i, is

$$\mathbf{J} = \begin{pmatrix} 1 & \cos y \\ x & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0.877583 \\ 0.5 & 1 \end{pmatrix}.$$

Two passes, seeded with $\mathbf{e}_1$ and then $\mathbf{e}_2$, return its two columns.

```
def plane_dual(x, y):
    return [x + y.sin(), y + 0.5 * (x * x)]         # (x + sin y, y + x^2/2)

z0 = (0.5, 0.5)
cols = []
for seed in ((1.0, 0.0), (0.0, 1.0)):               # e1, then e2
    out = plane_dual(Dual(z0[0], seed[0]), Dual(z0[1], seed[1]))
    cols.append([out[0].b, out[1].b])               # tangent = J @ seed, a column
J_fwd = np.array(cols).T                            # columns side by side
J_exact = np.array([[1.0, np.cos(0.5)], [0.5, 1.0]])
print('forward-mode J:\n', J_fwd.round(6))
print('matches the analytic Jacobian:', np.allclose(J_fwd, J_exact))
```

```
forward-mode J:
 [[1.       0.877583]
 [0.5      1.      ]]
matches the analytic Jacobian: True
```

Nothing in `Dual` requires the tangent to be a number. If the tangent of x is the array $(1, 0)$ and the tangent of y is $(0, 1)$, every operation carries both directions at once, and the tangent of output i becomes row i of $\mathbf{J}$. Can one pass return the whole matrix?

```
out = plane_dual(Dual(z0[0], np.array([1.0, 0.0])),     # identity seed:
                 Dual(z0[1], np.array([0.0, 1.0])))     # both directions at once
J_one = np.array([out[0].b, out[1].b])                  # row i = tangent of f_i
print('one-pass J:\n', J_one.round(6))
print('matches the analytic Jacobian:', np.allclose(J_one, J_exact))
```

```
one-pass J:
 [[1.       0.877583]
 [0.5      1.      ]]
matches the analytic Jacobian: True
```

It can, but the saving is smaller than it looks. Every tangent operation now works on arrays of length n, so the single pass does about n times the arithmetic of a scalar pass. Passes and work are different measures, and Section 34.5.2.4 counts work. For a scalar loss with 10^8 inputs, forward mode needs 10^8 passes, or one pass with tangents of 10^8 entries. Either way its cost is proportional to n runs of the program.

34.5.2 Reverse Mode and the Tape

34.5.2.1 Two Orders of One Product

To remove the factor n we need a different order of computation. Section 34.4 showed that the Jacobian of a program of K steps, $\mathbf{f} = \mathbf{f}_K \circ \cdots \circ \mathbf{f}_1$, is the product Equation 34.36 of the Jacobians of its steps,

$$\mathbf{J} = \mathbf{J}_K \mathbf{J}_{K-1} \cdots \mathbf{J}_1, \tag{34.45}$$

and that the order of multiplication decides the cost: for three layers of width 1000 with a row on the left, multiplying from the left takes 3×10^6 multiplications where the other order takes 2×10^9 (Figure 34.22). A program that differentiates never forms the factors. It holds a seed vector and passes it through one step after another. Forward mode multiplies a direction $\mathbf{v}$ in from the right; the alternative multiplies a row $\mathbf{u}^\top$ in from the left:

$$\mathbf{J}\mathbf{v} = \mathbf{J}_K\Big(\cdots(\mathbf{J}_2(\mathbf{J}_1\mathbf{v}))\Big), \qquad \mathbf{u}^\top\mathbf{J} = \Big(((\mathbf{u}^\top\mathbf{J}_K)\mathbf{J}_{K-1})\cdots\Big)\mathbf{J}_1. \tag{34.46}$$

With a seed, every step multiplies a matrix by a vector and never a matrix by a matrix. The left formula carries an input direction forward; it is the JVP of forward mode. The right formula carries an output direction backward. Its result $\mathbf{u}^\top\mathbf{J}$ is the *vector–Jacobian product* (VJP), a combination of the rows of $\mathbf{J}$ with weights $\mathbf{u}$; for $\mathbf{u} = \mathbf{e}_i$ it is row i.

The cost rule of automatic differentiation follows. Forward mode costs one sweep per input direction, and reverse mode one sweep per output direction. Each sweep costs a small constant multiple of one evaluation of the program. For programs built from $+, -, \times, \div$, Baur and Strassen showed that all partial derivatives together take at most a fixed multiple of the operations of the program itself, between three and five depending on which operations are counted (Baur and Strassen 1983). A scalar loss has a $1 \times n$ Jacobian, a single row, so one reverse sweep seeded with 1 returns the whole gradient. Figure 34.23 draws the rule as shapes.

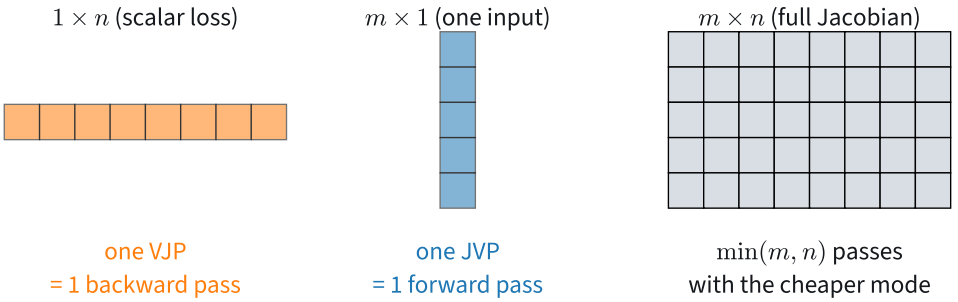


Figure 34.23: The shape of the Jacobian decides the cheaper mode. A scalar loss has a $1 \times n$ Jacobian: one row, one VJP, one backward pass. A map with one input has an $m \times 1$ Jacobian: one column, one JVP, one forward pass. A full $m \times n$ Jacobian costs $\min(m, n)$ passes with the cheaper mode.

The two modes are the same product read from opposite ends. Which end is cheaper depends only on the shape of $\mathbf{J}$: the cheap mode is the one whose number of seeds matches the short side. A loss, with one output and many inputs, has the most lopsided shape possible.

34.5.2.2 The Tape

Reverse mode does not carry the quantity that forward mode carries. In a program with input x and output y, the tangent $\dot{v}$ of an intermediate value v is the rate at which v changes

when the input moves in the direction $\dot{x}$. The adjoint $\bar{v}$ of Section 34.3 is the total derivative of the output y with respect to v, summed over all paths from v to y. A tangent is determined by the steps before v, an adjoint by the steps after it. For $y = v^2$ with $v = 3x$, for instance, $\dot{v} = 3\dot{x}$ and $\bar{v} = 2v$.

This difference decides how each mode runs (Figure 34.24). Tangents travel with the values, so forward mode finishes in the same sweep as the evaluation; `Dual` needs no memory of the past. Adjoints travel against the order of execution: the adjoint of a node needs the adjoints of the nodes that use it, and those exist only once the program has run to the end. Reverse mode therefore cannot finish in one forward sweep. It records the computation as it runs and then replays the record backward. The record is called the *tape*, or Wengert list, after the list of elementary operations that Wengert (1964) introduced for forward mode; running it backward is reverse mode, introduced by Linnainmaa (1970) .

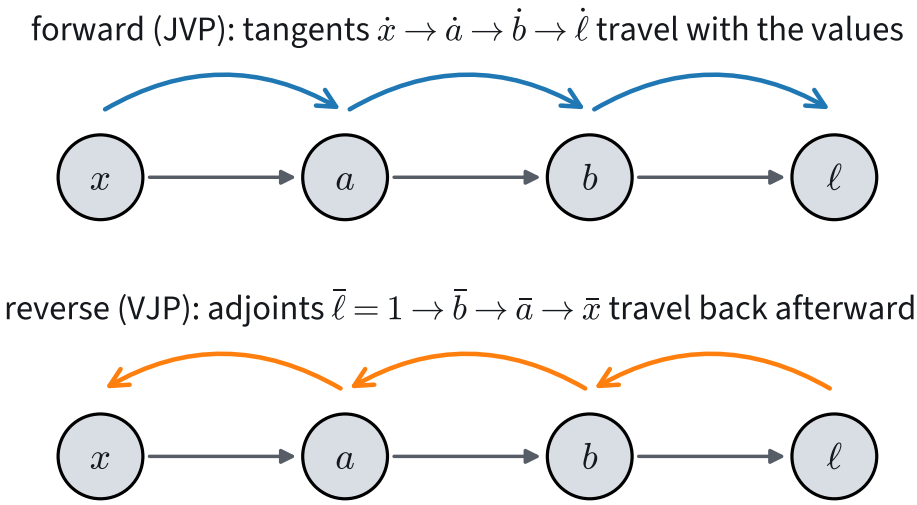


Figure 34.24: Forward and reverse mode on a chain $x \to a \to b \to \ell$. Both evaluate the chain from left to right (gray). Forward mode (top) carries the tangents $\dot{v}$ along with the values, one input direction per sweep. Reverse mode (bottom) runs the chain first and then carries the adjoints $\bar{v}$, the total derivatives of ℓ with respect to v, back from the seed $\bar{\ell} = 1$, one output direction per sweep, which for a scalar loss is the whole gradient.

A tape node stores its value, its parents, and a backward closure. Given the adjoint g of the node, the closure adds the node's contribution to the adjoints of its parents: for $c = a + b$ it adds g to both, and for $c = ab$ it adds $b\,g$ to $\bar{a}$ and $a\,g$ to $\bar{b}$. The function `backprop` sorts the nodes so that every node comes after its parents, seeds the output adjoint with 1, and runs the closures in reverse order. Like `Dual`, the class counts its operations, forward and backward.

```
class Var:
    """A scalar node on the autodiff tape; .grad accumulates the adjoint."""
    ops = 0                                          # recorded ops + backward steps
    def __init__(self, value, parents=(), backward=lambda g: None):
```

```
        self.value = value
        self.grad = 0.0
        self._parents = parents                    # nodes this one depends on
        self._backward = backward                  # pushes adjoint to parents
    def __add__(self, o):
        o = o if isinstance(o, Var) else Var(o)
        Var.ops += 1
        out = Var(self.value + o.value, (self, o))
        def back(g):                               # d(a+b) = 1*g to each parent
            Var.ops += 1
            self.grad += g; o.grad += g
        out._backward = back
        return out
    __radd__ = __add__
    def __mul__(self, o):
        o = o if isinstance(o, Var) else Var(o)
        Var.ops += 1
        out = Var(self.value * o.value, (self, o))
        def back(g):                               # product rule, reversed
            Var.ops += 1
            self.grad += o.value * g; o.grad += self.value * g
        out._backward = back
        return out
    __rmul__ = __mul__

def backprop(node):
    """Topologically sort the tape and replay it backward from `node`."""
    order, seen = [], set()
    def visit(v):
        if v not in seen:
            seen.add(v)
            for p in v._parents:
                visit(p)
            order.append(v)
    visit(node)
    node.grad = 1.0                                # seed the output adjoint
    for v in reversed(order):
        v._backward(v.grad)
```

`Var` implements only `+` and `*`, enough for the expression $y = (uv + u)^2$, which we first trace by hand. At $u = 2$, $v = -3$ it is recorded as three operations: $t = uv = -6$, then $r = t + u = -4$, then $y = r \cdot r = 16$. The backward sweep visits the nodes in an order such as y, r, t, u, v, in which every node comes after all the nodes that use it (`backprop` happens to take v before u, which is just as good). Each adjoint is the sum of the contributions along the node's outgoing edges, and each contribution is the adjoint of the consumer times the local derivative:

node	outgoing edges and their contributions	adjoint
y	none: the seed	$\bar{y} = 1$
r	both arguments of $y = r \cdot r$: $\bar{y}\,r + \bar{y}\,r$	$\bar{r} = -8$

node	outgoing edges and their contributions	adjoint
t	$r = t + u$: $\bar{r} \cdot 1$	$\bar{t} = -8$
u	$r = t + u$ and $t = uv$: $\bar{r} \cdot 1 + \bar{t} \cdot v$	$\bar{u} = -8 + 24 = 16$
v	$t = uv$: $\bar{t} \cdot u$	$\bar{v} = -16$

Differentiating $y = (uv + u)^2$ as a function of u and v confirms both results: $\partial y/\partial u = 2(uv + u)(v + 1) = 2(-4)(-2) = 16$ and $\partial y/\partial v = 2(uv + u)\,u = 2(-4)(2) = -16$. The sums matter at two nodes: r feeds both arguments of the last product, and u feeds both t and r (Figure 34.25). Overwriting instead of adding would give $\bar{r} = -4$, and even with $\bar{r}$ correct it would give $\bar{u} = -8$ or 24, depending on the order of the closures. An adjoint is the sum over outgoing edges, which is the chain rule's sum over paths, and this is why the closures use `+=`.

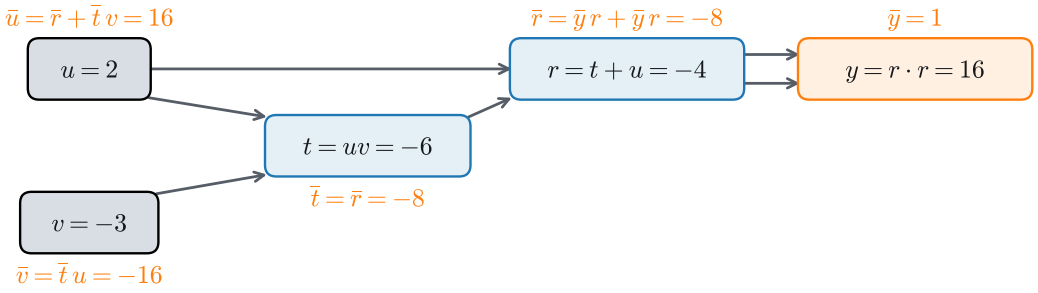


Figure 34.25: The tape of $y = (uv + u)^2$ at $u = 2$, $v = -3$. Each box shows the forward value of its node; the orange label shows its adjoint, the derivative of y with respect to that node. The graph is a diamond, not a chain: u feeds both $t = uv$ and $r = t + u$, and r feeds both arguments of $y = r \cdot r$, so the adjoints of r and u are sums of two contributions.

Recorded with `Var` and swept once, the same expression gives:

```
u, v = Var(2.0), Var(-3.0)
r = u * v + u                                   # t = uv, then r = t + u
y = r * r                                       # r feeds both arguments
backprop(y)
print('our tape  : dy/du = %.4f  dy/dv = %.4f' % (u.grad, v.grad))
```

```
our tape  : dy/du = 16.0000  dy/dv = -16.0000
```

These are $\bar{u}$ and $\bar{v}$ of the table. The framework's own reverse mode is the same procedure on a larger scale, and returns the same two numbers:

```
ut = torch.tensor(2.0, requires_grad=True)
vt = torch.tensor(-3.0, requires_grad=True)
rt = ut * vt + ut
(rt * rt).backward()                            # framework autograd
```

```
print('torch     : dy/du = %.4f  dy/dv = %.4f' % (ut.grad, vt.grad))
```

```
torch     : dy/du = 16.0000  dy/dv = -16.0000
```

A framework's tape differs from ours in three ways that leave the procedure unchanged. Its nodes hold tensors, so each backward closure is the VJP rule of a whole operation, such as Equation 34.39 for a matrix product. Its set of primitives is large, each with its own VJP rule. And it builds the graph in one of two ways. An eager tape records operations as the program runs, as `Var` does. A tracer runs the program once on placeholder inputs to obtain its sequence of operations, forms the forward-mode rule of each (a linear map of tangents), and reverses the sequence, replacing each linear map by its transpose. The result is the same backward sweep. Backpropagation is this procedure applied to a neural network (Rumelhart et al. 1986).

PyTorch records eagerly: every operation on a tensor with `requires_grad=True` adds a node, and `backward()` replays the nodes in reverse.

34.5.2.3 Parameter Gradients

The product Equation 34.36 is the derivative with respect to the input, and training needs the derivative with respect to the weights. They are read off during the same sweep by the backward rule of a layer (Section 34.4.3.2). For a layer $\mathbf{h}_k = \mathbf{W}_k \mathbf{h}_{k-1}$ with $\mathbf{W}_k$ of shape $d_k \times d_{k-1}$, the rule reads

$$\nabla_{\mathbf{W}_k} \ell = \bar{\mathbf{h}}_k \, \mathbf{h}_{k-1}^{\top}, \tag{34.47}$$

the outer product of the adjoint arriving from above and the activation arriving from below. A weight matters in proportion to how much its output matters to the loss and how large its input is. What concerns us here is the cost of the rule.

The outer product costs $d_k d_{k-1}$ multiplications, as many as the forward product $\mathbf{W}_k \mathbf{h}_{k-1}$. Passing the adjoint down, $\bar{\mathbf{h}}_{k-1}^{\top} = \bar{\mathbf{h}}_k^{\top} \mathbf{W}_k$, costs the same again. Counted in multiplications, the backward sweep therefore costs about two forward passes, and forward pass plus backward sweep about three, which is the 30 ms of the opening. Forward mode, in contrast, treats each weight as an input direction, so a single layer would need $d_k d_{k-1}$ passes. This is why no framework trains in forward mode.

34.5.2.4 Cost of the Two Modes

The two engines let us count operations instead of arguing about them. Take $f(\mathbf{x}) = (\sum_j x_j^2)^2$ with $n = 200$ inputs $x_j = 0.01\, j$ for $j = 1, \dots, 200$, built from $+$ and $\times$ alone so that `Var` can record it (log-sum-exp would need `exp` and `log` closures, the mirror of the `Dual` methods). Its gradient is $\nabla f = 4(\sum_j x_j^2)\mathbf{x}$, and one evaluation runs $2n + 1 = 401$ primitive operations: n squares, n additions, and the final square. Forward mode will need 200 passes of 401 operations. It is worth guessing the count for reverse mode before reading the output.

```
n = 200
def f(xs):                          # f: R^n -> R, from + and * alone
```

```
    total = 0.0
    for x in xs:
        total = total + x * x
    return total * total         # (sum of squares)^2

xs0 = [0.01 * (j + 1) for j in range(n)]
Dual.ops = 0
grad_fwd = [f([Dual(x, 1.0 if k == j else 0.0) for k, x in enumerate(xs0)]).b
            for j in range(n)]   # forward mode: one pass per input direction
Var.ops = 0
xs = [Var(x) for x in xs0]       # reverse mode: one recorded pass...
backprop(f(xs))                  # ...and one backward sweep for all n partials
grad_rev = [x.grad for x in xs]

gap = max(abs(a - b) for a, b in zip(grad_fwd, grad_rev))
print(f'forward mode: {Dual.ops:,} operations ({n} passes of {2 * n + 1})')
print(f'reverse mode: {Var.ops:,} operations (one recorded pass, one sweep)')
print(f'ratio {Dual.ops / Var.ops:.1f}, largest gradient difference {gap:.1e}')
```

```
forward mode: 80,200 operations (200 passes of 401)
reverse mode: 802 operations (one recorded pass, one sweep)
ratio 100.0, largest gradient difference 0.0e+00
```

Forward mode ran $80{,}200 = n(2n+1)$ operations and reverse mode $802 = 2(2n+1)$: 401 recorded on the forward pass and 401 backward closures. The ratio is $100 = n/2$, and the two gradients agree to every printed digit. The pass counts differ by a factor of n and the operation counts by $n/2$ for these engines; at $n = 10^8$ the ratio is the factor of the opening, up to the same constant.

The measured factor of two for a gradient seems to contradict the three of the opening. It does not, because the counter treats a backward closure as one operation. The closure of a product performs two multiplications, so the sweep does 402 multiplications against 201 on the forward pass, and gradient and value together cost three times the value, as for the dense layer. Counting the additions that accumulate the adjoints as well gives 1204 operations for the sweep and a factor of four, still within the bound of Baur and Strassen.

Reverse mode needs more memory instead. The tape keeps all 401 recorded values until the sweep consumes them, while forward mode, like the program itself, keeps only the current values, each with its tangent.

34.5.2.5 Gradient Checkpointing

A computation of T recorded steps keeps T values alive until the backward sweep reaches them. In a deep network these are the activations of every layer for every example in the batch, and this memory, not the arithmetic, often limits the batch size and the depth.

The simplest remedy cuts the tape into segments of s steps. The forward pass keeps only every s-th value, the T/s *checkpoints*, and discards the rest. When the backward sweep reaches a segment, it recomputes that segment's s values from its checkpoint onto a fresh tape, sweeps it, and frees it. At any moment the memory holds the checkpoints and one segment, and the best segment length balances the two:

$$M(s) = \frac{T}{s} + s, \qquad \min_s M(s) = M(\sqrt{T}) = 2\sqrt{T}. \tag{34.48}$$

The minimum follows from $M'(s) = 1 - T/s^2 = 0$, or from $T/s + s \geq 2\sqrt{T}$ by the inequality of arithmetic and geometric means. Memory grows like the square root of the length instead of the length itself. Each step is recomputed once, so the extra time is one forward pass. For $T = 400$, the segment lengths $s = 1, 5, 10, 20, 100, 400$ give $M = 401, 85, 50, 40, 104, 401$, smallest at $s = 20 = \sqrt{400}$. For $T = 10^4$, segments of $s = 100$ keep 200 values instead of 10,000 (Chen et al. 2016). Figure 34.26 draws the schedule.

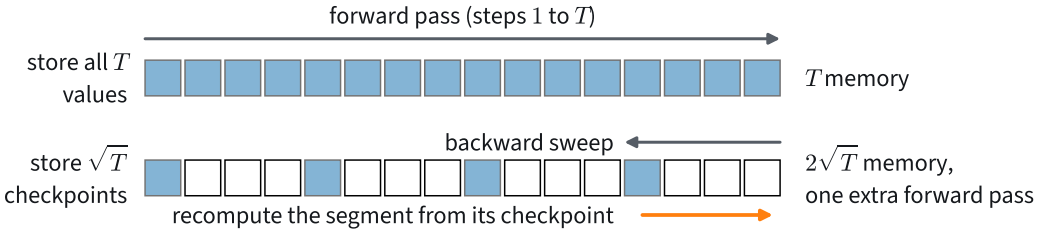


Figure 34.26: Gradient checkpointing on a tape of T steps. Plain reverse mode (top) keeps all T values until the backward sweep uses them. Checkpointing (bottom) keeps every s-th value; when the backward sweep reaches a segment, the segment is recomputed from its checkpoint and then swept. The memory is $T/s + s$, which is $2\sqrt{T}$ at $s = \sqrt{T}$, for one extra forward pass.

Checkpointing changes the time and the memory but not the gradient. The recomputed values are the same numbers, so the result equals that of plain reverse mode. Frameworks offer the same trade for the blocks of a deep network: a checkpointed block keeps only its input and is run a second time when the backward sweep reaches it. Recursive schedules, which checkpoint inside segments as well, reach $O(\log T)$ memory with $O(T \log T)$ recomputation.

34.5.3 Composing Jacobian Products

A framework exposes the two products our engines compute, $\mathbf{Jv}$ and $\mathbf{u}^\top\mathbf{J}$. Curvature along a direction, derivatives of solutions, and full Jacobians when they are needed are all built by composing them.

34.5.3.1 Jacobian–Vector and Vector–Jacobian Products

The framework calls for the two products are those of Section 1.5.4. For the plane map at $(0.5, 0.5)$ and $\mathbf{v} = \mathbf{u} = (1, 2)$, the Jacobian found by `Dual` gives $\mathbf{Jv} = (1 + 2 \cdot 0.8776,\ 0.5 + 2) = (2.7552,\ 2.5)$ and $\mathbf{u}^\top\mathbf{J} = (1 + 2 \cdot 0.5,\ 0.8776 + 2) = (2,\ 2.8776)$, and the calls return these numbers.

```
def plane(z):                                    # (x + sin y, y + x^2/2)
    return torch.stack([z[0] + torch.sin(z[1]), z[1] + 0.5 * z[0]**2])

z0 = torch.tensor([0.5, 0.5])
```

```
v, u = torch.tensor([1.0, 2.0]), torch.tensor([1.0, 2.0])
_, Jv = torch.func.jvp(plane, (z0,), (v,))          # forward mode: J v
_, vjp_fn = torch.func.vjp(plane, z0)               # reverse mode: records once,
uJ, = vjp_fn(u)                                     # then u^T J per seed
J = torch.func.jacrev(plane)(z0)                    # one VJP per row
print('J v   =', Jv.numpy().round(4))
print('u^T J =', uJ.numpy().round(4))
print('J     =', J.numpy().astype(float).round(4).tolist())
```

```
J v   = [2.7552 2.5   ]
u^T J = [2.     2.8776]
J     = [[1.0, 0.8776], [0.5, 1.0]]
```

A JVP and a VJP each cost one sweep, and the derivative information they carry along is a vector. The VJP also stores the forward intermediates, the tape. A dense $m \times n$ Jacobian costs $\min(m, n)$ sweeps with the cheaper mode, plus mn numbers of storage. Forming it is the right choice when n is small, as for this 2×2 map, or when its determinant is needed, as in the normalizing flows that change probability densities by $|\det \mathbf{J}|$.

34.5.3.2 Hessian–Vector Products

A model with $n = 10^6$ parameters has a Hessian of 10^{12} entries, 4 TB in float32. Newton's method needs the Hessian only to solve $\mathbf{H}\mathbf{d} = -\nabla \ell$ for its step $\mathbf{d}$, and Newton–CG, or truncated Newton, solves that system by conjugate gradients, an iterative method that touches $\mathbf{H}$ only through products $\mathbf{H}\mathbf{v}$ (Nocedal and Wright 2006; Martens 2010). Can we compute $\mathbf{H}\mathbf{v}$ without $\mathbf{H}$?

The Hessian is the Jacobian of the gradient map $\nabla \ell$, so $\mathbf{H}\mathbf{v}$ is a JVP of the gradient, the rate of change of the gradient along $\mathbf{v}$:

$$\mathbf{H}\mathbf{v} = \mathbf{J}_{\nabla \ell}\, \mathbf{v} = \frac{d}{dt}\Big|_{t=0} \nabla \ell(\mathbf{x} + t\mathbf{v}). \tag{34.49}$$

A reverse sweep computes $\nabla \ell$, and a forward-mode pass through that computation, with tangent $\mathbf{v}$, differentiates it along $\mathbf{v}$. This composition is called *forward over reverse* (Pearlmutter 1994). Its cost is a small constant multiple of the cost of one gradient evaluation, whatever n is.

Log-sum-exp gives a test with a known answer. Its gradient is the softmax $\mathbf{p}$, so its Hessian is the softmax Jacobian Equation 34.40 (it is also the covariance matrix of a one-hot vector $\mathbf{e}_J$ with J drawn from $\mathbf{p}$), and

$$\mathbf{H} = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^{\top}, \qquad \mathbf{H}\mathbf{v} = \mathbf{p} \odot \mathbf{v} - \mathbf{p}\,(\mathbf{p}^{\top}\mathbf{v}), \tag{34.50}$$

where $\odot$ multiplies entrywise. The product on the right needs only $3n$ multiplications and no $n \times n$ matrix. At $\mathbf{x} = (1, -0.5, 2)$ and $\mathbf{v} = (1, 0.5, -1)$ we have $\mathbf{p} = (0.2537, 0.0566, 0.6897)$ and $\mathbf{p}^{\top}\mathbf{v} = -0.4077$, so

$$\mathbf{H}\mathbf{v} = (0.3571,\ 0.0514,\ -0.4085).$$

The test is not vacuous. For the quadratic $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ the gradient at $\mathbf{v}$ equals $\mathbf{H}\mathbf{v}$, so a program that returned the wrong quantity would still pass; here $\mathbf{H}$ depends on $\mathbf{x}$, and the gradient at $\mathbf{v}$, $\mathrm{softmax}(\mathbf{v}) = (0.574, 0.348, 0.078)$, is far from $\mathbf{H}\mathbf{v}$.

Where a framework has both modes, their composition is one line.

```
x = torch.tensor([1.0, -0.5, 2.0])
v = torch.tensor([1.0, 0.5, -1.0])
lse = lambda x: torch.logsumexp(x, 0)
_, Hv = torch.func.jvp(torch.func.grad(lse), (x,), (v,))  # forward over reverse
p = torch.softmax(x, 0)
print('forward over reverse :', Hv.numpy().round(4))
print('p*v - p (p.v)        :', (p * v - p * (p @ v)).numpy().round(4))
```

```
forward over reverse : [ 0.3571  0.0514 -0.4085]
p*v - p (p.v)        : [ 0.3571  0.0514 -0.4085]
```

The printed products agree with Equation 34.50 to the printed precision, and no 3×3 matrix of second derivatives was formed. Exercise 4 builds the same product from `Dual` and `Var`, and compares forward over reverse with reverse over reverse.

34.5.3.3 Implicit Differentiation

Some outputs are defined by an equation and found by a solver: the fixed point of an iteration, the equilibrium of a physical model, the minimizer of an inner optimization problem. Taping twenty Newton iterations and running reverse mode through them works, but it stores every iteration and differentiates the path the solver happened to take. We want the derivative from the equation alone.

Start with one equation in one unknown, $g(x, \theta) = x^2 - \theta = 0$, solved by $x^\star = \sqrt{\theta}$. Differentiate the identity $x^\star(\theta)^2 - \theta \equiv 0$ in θ: $2x^\star \, dx^\star/d\theta - 1 = 0$, so

$$\frac{dx^\star}{d\theta} = \frac{1}{2x^\star} = \frac{1}{2\sqrt{\theta}},$$

which is the derivative of $\sqrt{\theta}$. At $\theta = 4$ it is 0.25. The formula for $x^\star$ was never used, only the equation and the value of the solution.

The general statement is the *implicit function theorem* (Rudin 1976), which we use without proof. Let $\mathbf{g}(\mathbf{x}, \theta)$ map $\mathbb{R}^d \times \mathbb{R}^p$ to $\mathbb{R}^d$, be continuously differentiable near a solution $(\mathbf{x}^\star, \theta)$ of $\mathbf{g} = \mathbf{0}$, and have an invertible Jacobian $\partial \mathbf{g}/\partial \mathbf{x}$ there. Then, near θ, the solution is a differentiable function $\mathbf{x}^\star(\theta)$. Differentiating the identity $\mathbf{g}(\mathbf{x}^\star(\theta), \theta) \equiv \mathbf{0}$ with the chain rule gives $\frac{\partial \mathbf{g}}{\partial \mathbf{x}} \frac{\partial \mathbf{x}^\star}{\partial \theta} + \frac{\partial \mathbf{g}}{\partial \theta} = \mathbf{0}$, hence

$$\frac{\partial \mathbf{x}^\star}{\partial \theta} = -\Big(\frac{\partial \mathbf{g}}{\partial \mathbf{x}}\Big)^{-1} \frac{\partial \mathbf{g}}{\partial \theta}. \tag{34.51}$$

Both Jacobians on the right are evaluated at the solution only. The derivative costs one linear solve with $\partial \mathbf{g}/\partial \mathbf{x}$ per parameter, however many iterations the solver ran. In VJP form it costs one solve for all parameters together: $\mathbf{u}^\top \partial \mathbf{x}^\star/\partial \theta = -\mathbf{a}^\top \partial \mathbf{g}/\partial \theta$, where $\mathbf{a}$ solves $(\partial \mathbf{g}/\partial \mathbf{x})^\top \mathbf{a} = \mathbf{u}$.

The adjoint method for differentiating through ODE solvers rests on the same idea (Section 38.2.2). Deep equilibrium models (Bai et al. 2019) define a whole layer as a fixed point $\mathbf{z}^\star = \mathbf{f}(\mathbf{z}^\star, \theta)$ and train it through Equation 34.51. Tuning a hyperparameter through an inner minimization is another instance.

We try Equation 34.51 on a nonlinear 2×2 system, $g_1 = x_1^2 + \theta x_2 - 2 = 0$ and $g_2 = x_1 x_2 + \theta - 1 = 0$, solved by twenty Newton steps, and compare it with a central difference of the solver's output.

```
# Implicit gradient dx*/dtheta = -(dg/dx)^{-1} dg/dtheta on a 2x2 system,
# checked against a finite difference of the solver output
def g(x, th):                       # g(x, theta) = 0 defines x*(theta)
    return np.array([x[0]**2 + th * x[1] - 2, x[0] * x[1] + th - 1])

def dg_dx(x, th):                   # Jacobian in x, evaluated at (x, th)
    return np.array([[2 * x[0], th], [x[1], x[0]]])

def solve(th):                      # Newton iteration, from a fixed start
    x = np.array([1.0, 1.0])
    for _ in range(20):
        x = x - np.linalg.solve(dg_dx(x, th), g(x, th))
    return x

th, h = 0.3, 1e-6
xs = solve(th)
dg_dth = np.array([xs[1], 1.0])     # dg/dtheta at the solution
implicit = -np.linalg.solve(dg_dx(xs, th), dg_dth)
fd = (solve(th + h) - solve(th - h)) / (2 * h)
print('implicit-function gradient:', implicit.round(8))
print('finite-difference check   :', fd.round(8))
```

```
implicit-function gradient: [-0.11311451 -0.69322129]
finite-difference check   : [-0.11311451 -0.69322129]
```

The two vectors agree to the precision of the finite difference, and the twenty Newton iterations never entered the implicit formula. The derivative depends only on $\mathbf{g}$ and the solution, so any converged solver gives the same gradient.

The theorem has two limits, and both can be seen. First, it needs $\partial \mathbf{g}/\partial \mathbf{x}$ to be invertible. For $x^2 = \theta$ at $\theta = 0$, $\partial g/\partial x = 2x^\star = 0$ and the formula divides by zero, matching the vertical tangent of $\sqrt{\theta}$ at 0. Second, the formula assumes that the solver has converged. What do the formula and the unrolled solver return when Newton's method on $x^2 = 4$, started at $x = 1$, stops after $k = 1, 2, 3$ steps? We evaluate the formula $1/(2\tilde{x}_k)$ at the unconverged $\tilde{x}_k$, and differentiate the k-step solver itself by a central difference in θ.

```
def newton(theta, k, x=1.0):                    # k Newton steps on x^2 = theta
    for _ in range(k):
        x = x - (x * x - theta) / (2 * x)
    return x

theta, h = 4.0, 1e-6
for k in (1, 2, 3):
```

```
    x_k = newton(theta, k)
    unrolled = (newton(theta + h, k) - newton(theta - h, k)) / (2 * h)
    print(f'k={k}: x = {x_k:.5f}, formula {1 / (2 * x_k):.4f}, '
          f'unrolled {unrolled:.4f}, exact 0.25')
```

```
k=1: x = 2.50000, formula 0.2000, unrolled 0.5000, exact 0.25
k=2: x = 2.05000, formula 0.2439, unrolled 0.2900, exact 0.25
k=3: x = 2.00061, formula 0.2499, unrolled 0.2509, exact 0.25
```

After one, two, and three steps the solver is at 2.5, 2.05, and 2.00061. The formula returns 0.2000, 0.2439, and 0.2499, and the unrolled derivative 0.5000, 0.2900, and 0.2509, against the true 0.25. Both are wrong before convergence. The formula inherits the error of the solver, and the unrolled derivative, which differentiates the path, is further off here. The remedy is to converge first.

One problem stays open. Automatic differentiation evaluates the product $\mathbf{J}_K \cdots \mathbf{J}_1$ to rounding error, but it cannot change what the product is, and over many steps the product can shrink or grow geometrically (Section 34.4). An exact gradient that is 10^{-20} is of little use. The remedies for vanishing and exploding gradients are choices of initialization and architecture, which begin in Section 4.4.

34.5.4 Summary

The derivative of a program is the product of the Jacobians of its steps, and automatic differentiation evaluates this product against a seed vector, so that every step multiplies a matrix by a vector. Forward mode runs the program on dual numbers and returns one Jacobian–vector product, one input direction, per pass; reverse mode records a tape, replays it backward, and returns one vector–Jacobian product, one output direction, per pass. The shape of the Jacobian decides the mode. A loss has a single output, so one backward sweep gives every parameter gradient at about three times the cost of the loss. On $(\sum_j x_j^2)^2$ with $n = 200$ inputs, the gradient took 802 operations in reverse mode against $80{,}200$ in forward mode. The tape's memory grows with the length T of the computation, and checkpointing reduces it to $2\sqrt{T}$ for one extra forward pass. Composing the two modes gives the Hessian–vector product at a small constant multiple of the cost of one gradient evaluation, and the implicit function theorem gives the derivative of a solver's output from one linear solve at the solution, provided the solver has converged.

34.5.5 Exercises

1. code **Dual and complex steps.** Take $\mathbf{f}(x, y) = (x + \sin y,\ y + x^2/2)$ at $\mathbf{x}_0 = (0.5, 0.5)$ and the second column $\mathbf{J}\mathbf{e}_2 = (\cos 0.5,\ 1)$ of its Jacobian. The central difference $(\mathbf{f}(\mathbf{x}_0 + t\mathbf{e}_2) - \mathbf{f}(\mathbf{x}_0 - t\mathbf{e}_2))/(2t)$ has an error of order t^2 from truncation plus an error of order u/t from rounding, where $u \approx 10^{-16}$ is the float64 unit roundoff.
    1. Predict the best step and the smallest error, then check the prediction for $t = 10^{-1}, 10^{-2}, \ldots, 10^{-12}$ (you should find about 2×10^{-11} near $t = 10^{-5}$). Compare with one pass of `Dual` seeded with $\mathbf{e}_2$. Why does no step size enter the dual computation?
    2. Repeat (a) for the first column $\mathbf{J}\mathbf{e}_1$, and explain why the central difference is exact there for every t, up to rounding. (Hint: $\mathbf{f}$ is a polynomial of degree two in x.)

3. Replacing ε by the imaginary unit i gives the complex-step estimate $\operatorname{Im} \mathbf{f}(\mathbf{x}_0 + it\mathbf{e}_2)/t$ (Squire and Trapp 1998), valid for an $\mathbf{f}$ that extends analytically to complex arguments, as this one does. Expand $\mathbf{f}(\mathbf{x}_0 + it\mathbf{e}_2)$ to third order in t and show that the error of the estimate is $O(t^2)$. Why does the estimate subtract no nearly equal numbers?
4. Evaluate the complex-step estimate with NumPy's complex arithmetic for $t = 10^{-1}, 10^{-2}, \dots, 10^{-20}$ and compare its errors with those of (a). Which difference between $\varepsilon^2 = 0$ and $i^2 = -1$ causes the $O(t^2)$ error of the complex step?

2. code **Compressed Jacobians.** A dense $m \times n$ Jacobian costs $\min(m, n)$ passes, and a sparse one can cost fewer. A JVP with the seed $\mathbf{e}_j + \mathbf{e}_k$ returns the sum of columns j and k of the Jacobian. If no row has nonzero entries in both columns, each nonzero entry of the sum belongs to exactly one of them.
    1. Show that one JVP recovers the whole Jacobian of an elementwise map $\varphi : \mathbb{R}^n \to \mathbb{R}^n$. Which seed does it use? How many VJPs does reverse mode need for the same matrix?
    2. Let $f_i(\mathbf{x}) = x_{i-1}x_i + \sin(x_{i+1})$ for $i = 1, \dots, n$, where every term with an index outside $1, \dots, n$ is dropped. Find three seeds from which all nonzero entries of $\mathbf{J_f}$ can be read off. Implement $\mathbf{f}$ for `Dual` inputs with $n = 10$, recover $\mathbf{J_f}$ from the three JVPs, and compare it with the analytic Jacobian. Can the three JVPs share a single pass, with tangents that are arrays of length three?
    3. How many seeds does a banded Jacobian need whose nonzero entries satisfy $|i - j| \le b$, and how does this number depend on n? A one-dimensional convolution with kernel size k, without padding, has nonzero entries at $0 \le j - i \le k - 1$; how many seeds does it need?

3. code **Hyperparameter gradient.** Ridge regression with one penalty $\lambda_j > 0$ per feature fits

$$\mathbf{w}^\star(\lambda) = \operatorname*{argmin}_{\mathbf{w}} \ \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \sum_{j=1}^{d} \lambda_j w_j^2$$

on training data $(\mathbf{X}, \mathbf{y})$, and its validation loss on data $(\mathbf{X}_\mathrm{v}, \mathbf{y}_\mathrm{v})$ is $V(\lambda) = \|\mathbf{X}_\mathrm{v}\mathbf{w}^\star(\lambda) - \mathbf{y}_\mathrm{v}\|^2$.
    1. Show that $\mathbf{w}^\star$ solves $\mathbf{g}(\mathbf{w}, \lambda) = (\mathbf{X}^\top\mathbf{X} + \operatorname{diag}(\lambda))\,\mathbf{w} - \mathbf{X}^\top\mathbf{y} = \mathbf{0}$, and use Equation 34.51 to derive $\partial\mathbf{w}^\star/\partial\lambda$.
    2. Show that $\nabla_\lambda V = -\mathbf{a} \odot \mathbf{w}^\star$, where $\mathbf{a}$ solves one linear system with the matrix $\mathbf{X}^\top\mathbf{X} + \operatorname{diag}(\lambda)$, and give its right-hand side. How many linear solves does the forward route through $\partial\mathbf{w}^\star/\partial\lambda$ need for d penalties, and how many does this adjoint route need?
    3. Implement V and the adjoint gradient in NumPy for random data with 50 training examples, 50 validation examples, and $d = 10$ features. Check the gradient against central differences of V.

    Adapted from Alan Edelman and Steven G. Johnson, Matrix Calculus for Machine Learning and Beyond, problem set 2, MIT 18.S096, IAP 2023, Problem 1.

4. code **Hessian–vector product.** ⊠ The function $f(\mathbf{x}) = \left(\sum_j x_j^2\right)^2$ of Section 34.5.2.4, evaluated at the inputs `xs0` with $n = 200$, is built from addition and multiplication

alone, so both `Dual` and `Var` can evaluate it.

1. Derive ∇f and the Hessian $\mathbf{H}$ of f, and write $\mathbf{Hv}$ in a form that never builds $\mathbf{H}$.
2. Draw a random direction $\mathbf{v}$, wrap each input x_j as `Var(Dual(x_j, v_j))`, and run `backprop`. Show that the value parts of the resulting adjoints form ∇f and that their ε-parts form $\mathbf{Hv}$, and compare both with
   (a) and with your framework's forward-over-reverse product applied to the same f. Which operations of the backward closures run in dual arithmetic, and why does this make the computation forward over reverse in the sense of Equation 34.49?
3. Nest the tape in itself instead. Wrap each input as `Var(Var(x_j))` and run `backprop`; the adjoints are now `Var` objects. Form $\sum_j (\nabla f)_j v_j$ from them and run `backprop` on it to obtain $\mathbf{Hv}$ by reverse over reverse. Count the `Var` objects that each method creates, for example with a counter in `Var.__init__`, and compare both counts with that of one plain gradient. Which method records the backward pass, and what does that cost in memory?

5. code **Checkpointing on the tape.** ▣ Let $x_k = x_{k-1} - 0.01\, x_{k-1}^2$ for $k = 1, \dots, T$ with $T = 400$ and $x_0 = 1$, and differentiate x_T with respect to x_0. `Var` has no subtraction, so record each step as `x + (-0.01) * x * x`.
   1. Record the chain on the tape with plain `backprop` and count the nodes that the tape holds when the backward sweep starts.
   2. `backprop` seeds the output adjoint with 1. Generalize it to accept any seed. Show that recording a segment $x_i \mapsto x_j$ from a fresh leaf for x_i and running the generalized `backprop` from x_j with the seed $\partial x_T / \partial x_j$ returns $\partial x_T / \partial x_i$ in the adjoint of the leaf.
   3. Implement the schedule of Figure 34.26 with segments of s steps: a forward pass in plain floats that stores $x_0, x_s, x_{2s}, \dots$, followed by a backward pass that re-records one segment at a time, from the last to the first, and applies (b). Check that the result equals the gradient of (a).
   4. For $s \in \{1, 5, 10, 20, 100, 400\}$, report the number of stored checkpoints, the largest tape in steps, and the number of evaluations of the step $x \mapsto x - 0.01x^2$, the first forward pass included. Confirm the memory $T/s + s$ of Equation 34.48, that is $401, 85, 50, 40, 104, 401$, and its minimizer $s = \sqrt{T} = 20$. Repeat the sweep for $T = 100$ and $T = 1600$ over the divisors $s \le 800$ of T (a single segment of 1600 steps exceeds Python's default recursion limit in `backprop`). How does the minimizing s scale with T, and how many step evaluations does the best schedule need?

6. [extended] **Equilibrium layer.** ▣▣ A deep equilibrium layer (Bai et al. 2019) outputs the fixed point

$$\mathbf{z}^\star = \tanh(\mathbf{W}\mathbf{z}^\star + \mathbf{c}), \tag{34.52}$$

where $\mathbf{c}$ depends on the layer's input. Let the loss be $\ell = \frac{1}{2}\|\mathbf{z}^\star - \mathbf{y}\|^2$, and write $\mathbf{D} = \operatorname{diag}(1 - (z_1^\star)^2, \dots, 1 - (z_d^\star)^2)$ for the derivative of tanh at the fixed point.

   1. Show that the map $\mathbf{z} \mapsto \tanh(\mathbf{W}\mathbf{z} + \mathbf{c})$ is a contraction when $\|\mathbf{W}\|_2 < 1$, so that the fixed point exists and is unique, and that $\mathbf{I} - \mathbf{DW}$ is then invertible. Use Equation 34.51 to derive $\nabla_{\mathbf{W}} \ell$, and show that it equals $(\mathbf{Da})(\mathbf{z}^\star)^\top$ for the adjoint $\mathbf{a}$ that

solves $(\mathbf{I} - \mathbf{DW})^\top \mathbf{a} = \mathbf{z}^\star - \mathbf{y}$.

2. Take $d = 20$, a standard normal $\mathbf{G} \in \mathbb{R}^{d\times d}$, $\mathbf{S} = \mathbf{G} + \mathbf{G}^\top$, and $\mathbf{W} = \rho\,\mathbf{S}/\|\mathbf{S}\|_2$ for $\rho \in \{0.5, 0.9, 0.99\}$, and draw the entries of $\mathbf{c}$ and $\mathbf{y}$ independently from $\mathcal{N}(0, 0.1^2)$. In NumPy, compute $\mathbf{z}^\star$ by fixed-point iteration and the gradient of (a) with one linear solve. Check a few of its entries against central differences of ℓ.
3. Unrolled differentiation runs K iterations from $\mathbf{z}_0 = \mathbf{0}$ and backpropagates through all of them. Implement this backward sweep by hand, one VJP per iteration. Plot against K the error of the unrolled gradient relative to the gradient of (b), together with the forward error $\|\mathbf{z}_K - \mathbf{z}^\star\|$. Compare the decay rate of both errors with the spectral radius of $\mathbf{DW}$. How does the ratio of the two errors behave as K grows, and why? How does the memory of each gradient scale with K?
4. Replace the linear solve by m steps of the iteration $\mathbf{a} \leftarrow (\mathbf{z}^\star - \mathbf{y}) + (\mathbf{DW})^\top \mathbf{a}$ from $\mathbf{a} = \mathbf{0}$, a truncated Neumann series that Geng et al. (2021) use as a *phantom gradient*. Show that this truncated adjoint gives the same gradient as backpropagation through m iterations started at $\mathbf{z}^\star$. Plot the cosine between the truncated and the exact gradient against m for each ρ. The gradient for $m = 1$ needs no linear solve and is the Jacobian-free backpropagation of Wu Fung et al. (2022) . Is it still a descent direction for ℓ?

7. code **A tensor tape.** Extend `Var` from scalars to NumPy arrays, so that each node holds an array and each backward closure applies a VJP rule.

   1. State the backward rule of each primitive as a VJP. For $\mathbf{Y} = \mathbf{XW}$, show $\bar{\mathbf{X}} = \bar{\mathbf{Y}}\mathbf{W}^\top$ and $\bar{\mathbf{W}} = \mathbf{X}^\top\bar{\mathbf{Y}}$. For a broadcast sum $\mathbf{Y} = \mathbf{X} + \mathbf{b}$ with a row vector $\mathbf{b}$, show that the adjoint of $\mathbf{b}$ sums $\bar{\mathbf{Y}}$ over the rows. For $s = \sum_{ij} X_{ij}$, show that the adjoint of $\mathbf{X}$ is $\bar{s}\,\mathbf{1}\mathbf{1}^\top$, and for relu that it is $\bar{\mathbf{Y}}$ times a mask.
   2. Implement a class `TVar` with `__matmul__`, `__add__`, `sum`, and `relu`, and reuse `backprop` unchanged. Why does `backprop` not need to know that the values are arrays?
   3. Build the loss $\frac{1}{2}\sum\big(\mathrm{relu}(\mathbf{XW}_1)\mathbf{W}_2 - \mathbf{Y}\big)^2$ for random $\mathbf{X} \in \mathbb{R}^{32\times 8}$, $\mathbf{W}_1 \in \mathbb{R}^{8\times 16}$, $\mathbf{W}_2 \in \mathbb{R}^{16\times 1}$, and $\mathbf{Y} \in \mathbb{R}^{32\times 1}$ (you will need a square or a product node), and check $\nabla_{\mathbf{W}_1}\ell$ and $\nabla_{\mathbf{W}_2}\ell$ against your framework's autograd to 10^{-6}.
   4. Count the multiplications of the forward product $\mathbf{XW}$ and of its two backward products, and conclude that the backward pass of a linear layer costs twice its forward pass.

8. code **Mixed mode through a bottleneck.** The text multiplies Equation 34.36 from one end or the other. Splitting it in the middle can beat both. Let $\mathbf{h}(\mathbf{x}) = \tanh(\mathbf{Ax})$ with $\mathbf{A} \in \mathbb{R}^{2\times 1000}$, let $\mathbf{g}(\mathbf{z}) = \tanh(\mathbf{Bz})$ with $\mathbf{B} \in \mathbb{R}^{1000\times 2}$, and let $\mathbf{f} = \mathbf{g}\circ\mathbf{h} : \mathbb{R}^{1000} \to \mathbb{R}^{1000}$.

   1. Show that $\operatorname{rank} \mathbf{J}_\mathbf{f} \le 2$.
   2. Count the passes needed to form the full 1000×1000 Jacobian in pure forward mode and in pure reverse mode. In mixed mode, two VJPs through $\mathbf{h}$ give $\mathbf{J}_\mathbf{h}$ (2×1000), two JVPs through $\mathbf{g}$ give $\mathbf{J}_\mathbf{g}$ (1000×2), and one product forms $\mathbf{J}_\mathbf{g}\mathbf{J}_\mathbf{h}$.
   3. Implement all three with your framework's `jvp` and `vjp` (in MXNet, with VJPs and the analytic diagonal factors of tanh) and compare with its full-Jacobian function. Count multiplications: about 5×10^6 for either pure mode (each pass multiplies by $\mathbf{A}$, by $\mathbf{B}$, and by the $\tanh'$ factors) against about 2×10^6 for mixed mode, and 1000 runs of the program against four runs of half of it.
   4. For a longer chain with one layer of width k in the middle, how many half-

program passes does mixed mode need? How would you choose where to split the product $\mathbf{J}_K \cdots \mathbf{J}_1$ in general?

9. code **Forward gradients.** Can forward mode train at all? Let $\ell : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$. One forward pass seeded with $\mathbf{v}$ returns $\nabla\ell^\top \mathbf{v}$; define the estimate $\hat{\mathbf{g}} = (\nabla\ell^\top \mathbf{v})\,\mathbf{v}$, which Baydin et al. (2022) call a forward gradient.
   1. Show that $\mathbb{E}[\hat{\mathbf{g}}] = \nabla\ell$.
   2. Show that $\mathbb{E}\|\hat{\mathbf{g}} - \nabla\ell\|^2 = (n+1)\|\nabla\ell\|^2$. (Hint: $\mathbb{E}[v_i v_j v_k^2]$ is 3 for $i = j = k$, 1 for $i = j \neq k$, and 0 when $i \neq j$.)
   3. With `Dual`, and the f and `xs0` of the cost cell ($n = 200$), average m independent estimates for $m = 1, 10, 10^2, 10^3, 10^4$ and plot the relative error against m on log–log axes. Show that the slope is $-\frac{1}{2}$ and that about 2×10^4 passes reach a relative error of 10%.
   4. Compare that pass count with the 200 coordinate passes and the one reverse sweep of the text. What does the random seed gain, and what does it cost?

10. code **Second derivatives in one forward pass.** Adjoin ε with $\varepsilon^3 = 0$ (but $\varepsilon^2 \neq 0$), and write $a + b\varepsilon + c\varepsilon^2$ as a triple (a, b, c).
    1. Derive the product $(a, b, c)(a', b', c') = (aa',\ ab' + ba',\ ac' + bb' + ca')$, and show that for an analytic φ, $\varphi(a + b\varepsilon + c\varepsilon^2) = \varphi(a) + \varphi'(a)\,b\,\varepsilon + (\varphi'(a)\,c + \frac{1}{2}\varphi''(a)\,b^2)\varepsilon^2$.
    2. Implement `Dual2` with `__add__`, `__mul__`, `sin`, and `exp`, and evaluate $f(x) = \sin(x^2) + e^x$ on $(1.3, 1, 0)$. Show that the result is $(f(1.3),\ f'(1.3),\ \frac{1}{2}f''(1.3))$, and check $f''(1.3) \approx -3.2806$ against the hand formula $f''(x) = 2\cos x^2 - 4x^2 \sin x^2 + e^x$.
    3. For $\ell : \mathbb{R}^n \to \mathbb{R}$ with inputs seeded by a direction $\mathbf{v}$, what does the ε^2-part return? (Answer: $\frac{1}{2}\mathbf{v}^\top \mathbf{H}\mathbf{v}$, a number.) How many such passes give the vector $\mathbf{H}\mathbf{v}$ by polarization, against one forward-over-reverse pass?
    4. Write ε as a 3×3 matrix with $\varepsilon^3 = \mathbf{0}$ and verify the product of (a) by matrix multiplication.

34.6 Integral Calculus

The main text used continuous random variables without asking how their probabilities are computed. A normal distribution was described by its density (Section 2.1.3), an expected loss was an average over that density, and both were taken as given. This section asks how we add up a quantity that is spread continuously over an interval.

The standard normal distribution, the bell curve, shows why the question needs care. A standard normal random variable Z has the density $p(z) = e^{-z^2/2}/\sqrt{2\pi}$. Its value at $z = 0$ is 0.40, yet $P(Z = 0) = 0$, and the same holds at every other single point. Intervals do carry probability: $P(-1 \le Z \le 1) \approx 0.68$. That number is the area under p between -1 and 1, and the constant $1/\sqrt{2\pi}$ is the one that makes the total area under p equal to 1. Where does the constant come from, and how is 0.68 computed?

The question is not special to the bell curve. Every probability and every expected loss over a continuous variable is an area of this kind, and training a model through random sampling, as variational autoencoders and policy-gradient methods do, needs the derivative of such an area with respect to a parameter.

The plan is to cut the area into thin slices, add them, and let the slices shrink; the limit is the integral. The fundamental theorem of calculus replaces the adding by the reverse of differentiation. That settles most integrals but not the bell curve, whose constant $\sqrt{2\pi}$ we find by a detour through two dimensions. Three results carry the section: the fundamental theorem Equation 34.55, the change of variables Equation 34.63, and the Gaussian integral Equation 34.65. A reader with first-year calculus at hand can skim as far as Section 34.6.2.2. The derivative of an area with respect to a parameter needs all of these tools and has a section of its own, Section 34.7. Measure theory and the Lebesgue integral are left out; Section 34.8 points to them.

34.6.1 The Definite Integral

34.6.1.1 Riemann Sums

A density gives probability per unit length, so over a short interval of width ϵ the probability is about density times width. Over a long interval we cut it into thin slices, form density times width on each, and add. The recipe works for any function.

We start with a case that can be done by hand. Take $f(x) = x^2$ on $[0, 1]$ and cut the interval into N slices of width $\epsilon = 1/N$. The i-th slice is $[i/N, (i+1)/N]$; give it the height of f at its left end, $(i/N)^2$. The rectangles add up to

$$\sum_{i=0}^{N-1} \frac{1}{N}\left(\frac{i}{N}\right)^2 = \frac{1}{N^3}\sum_{i=0}^{N-1} i^2 = \frac{(N-1)N(2N-1)}{6N^3} = \frac{1}{3} - \frac{1}{2N} + \frac{1}{6N^2},$$

where the middle step uses the classical formula $\sum_{i=0}^{N-1} i^2 = (N-1)N(2N-1)/6$. The sums have the limit $\frac{1}{3}$, and they miss it by about $\frac{1}{2N}$, an error of first order in the slice width ϵ.

The function `left_sum` implements this **left-endpoint rule** for any f: cut $[a, b]$ into slices of width ϵ and tag each slice at its left end, as in Figure 34.27. We use it for every grid sum of the section, and try it first on x^2.

```
def left_sum(f, a, b, eps):
    """Left-endpoint Riemann sum of f on [a, b] with slices of width eps."""
    x = a + eps * np.arange(round((b - a) / eps))   # left ends of the slices
    return float(np.sum(eps * f(x)))

for N in [10, 100, 1000]:
    print(f'N={N:<5} left sum={left_sum(lambda x: x**2, 0, 1, 1 / N):.6f}  '
          f'formula={1/3 - 1/(2*N) + 1/(6*N**2):.6f}')
```

```
N=10    left sum=0.285000  formula=0.285000
N=100   left sum=0.328350  formula=0.328350
N=1000  left sum=0.332834  formula=0.332834
```

The sums agree with the closed form to all printed digits.

The definition. Nothing in the recipe needed equal widths or left ends. A *partition* $P : a = x_0 < x_1 < \cdots < x_N = b$ cuts $[a, b]$ into slices, a *tag* $\xi_i \in [x_{i-1}, x_i]$ picks a point in each slice, and the *mesh* $\|P\| = \max_i (x_i - x_{i-1})$ is the width of the widest slice. A bounded

function f is **Riemann integrable** on $[a,b]$ if there is a number I that every tagged sum approaches as the mesh tends to zero, whatever the tags:

$$\int_a^b f(x)\,dx = I = \lim_{\|P\|\to 0} \sum_{i=1}^{N} f(\xi_i)(x_i - x_{i-1}). \tag{34.53}$$

The integral is the value that the slice sums approach as the slices become thin. The computation above gives $\int_0^1 x^2\,dx = \frac{1}{3}$. Every continuous function on $[a,b]$ is Riemann integrable, and so is every bounded function with finitely many jumps (Rudin 1976). Some bounded functions, such as the indicator of the rationals, have no Riemann integral; the Lebesgue integral of Section 34.8 repairs this, and it agrees with the Riemann integral whenever the latter exists.

Two properties follow from the definition, and the proofs below use both. The integral is **monotone**: if $f \le g$ on $[a,b]$, then every tagged sum of f is at most the same tagged sum of g, and limits preserve $\le$, so $\int_a^b f\,dx \le \int_a^b g\,dx$. It is **additive over subintervals**: $\int_a^c f\,dx = \int_a^b f\,dx + \int_b^c f\,dx$ for $a \le b \le c$, because partitions that include the point b split into a partition of $[a,b]$ and one of $[b,c]$.

The running example. The integral

$$\int_0^2 \frac{x}{1+x^2}\,dx = \tfrac{1}{2}\log 5 = 0.80472$$

is our running example as far as the Gaussian integral (the evaluation rule Equation 34.56 will show why its value is $\frac{1}{2}\log 5$). Figure 34.27 shows left-endpoint rectangles under this curve for three slice widths.

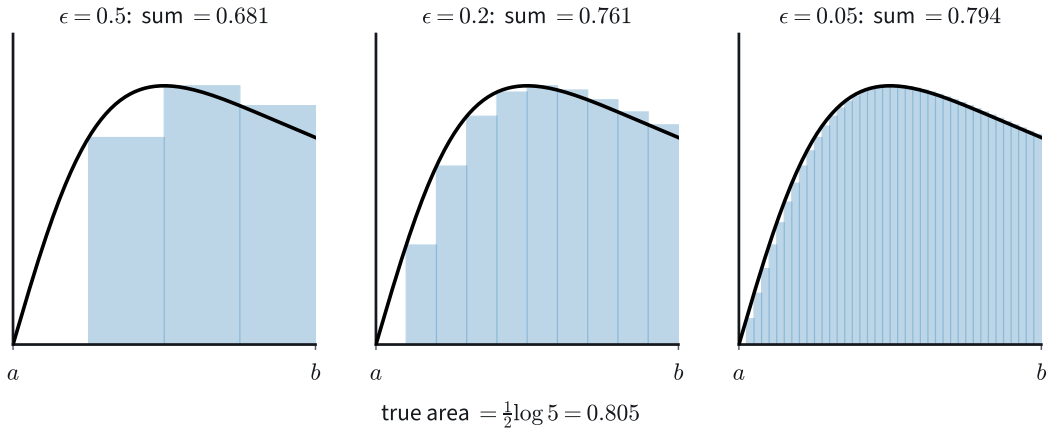


Figure 34.27: Left-endpoint rectangles under $f(x) = x/(1+x^2)$ on $[0,2]$ for slice widths $\epsilon = 0.5$, 0.2 and 0.05. Each slice takes the height of f at its left end; the rectangle sums approach the area $\frac{1}{2}\log 5$ as $\epsilon \to 0$.

How fast do the sums approach $\frac{1}{2}\log 5$? For x^2 the error was $\frac{1}{2N} = \frac{\epsilon}{2}$. If the first order is typical, slices ten times thinner should give an error ten times smaller, and the table prints the error beside $0.2\,\epsilon$ for comparison.

```
f = lambda x: x / (1 + x**2)
truth = math.log(5) / 2
for eps in [1e-1, 1e-2, 1e-3, 1e-4]:
    s = left_sum(f, 0, 2, eps)
    print(f'eps={eps:.0e}  sum={s:.6f}  truth-sum={truth - s:.2e}  '
          f'0.2*eps={0.2 * eps:.0e}')
print(f'truth = (1/2) log 5 = {truth:.6f}')
```

```
eps=1e-01  sum=0.783785  truth-sum=2.09e-02  0.2*eps=2e-02
eps=1e-02  sum=0.802710  truth-sum=2.01e-03  0.2*eps=2e-03
eps=1e-03  sum=0.804519  truth-sum=2.00e-04  0.2*eps=2e-04
eps=1e-04  sum=0.804699  truth-sum=2.00e-05  0.2*eps=2e-05
truth = (1/2) log 5 = 0.804719
```

Each time ϵ shrinks by ten, so does the error, and every sum lies below the truth. The constant 0.2 is half of $f(2) - f(0) = 0.4$. This is no accident, and we derive it once the fundamental theorem is available.

34.6.1.2 Signed Area

Rectangles below the axis have negative height $f(\xi_i) < 0$ and enter the sum with a minus sign, so the integral measures *signed* area: where $f < 0$ the area counts as negative, as in $\int_0^1 (-1)\,dx = -1$. Integrating from right to left negates the result, $\int_b^a f\,dx = -\int_a^b f\,dx$; this is the convention that keeps additivity true for every order of a, b, c. Figure 34.28 shows both rules on one curve.

34.6.2 The Fundamental Theorem

34.6.2.1 The Theorem

Adding slices gave $\frac{1}{3}$ only because $\sum i^2$ has a closed form. For the running example the left sum is $\sum_i \epsilon \cdot i\epsilon/(1 + i^2\epsilon^2)$, and no formula collapses it. We need a method that does not add slices at all.

The method comes from letting the upper limit vary. Fix a base point a and define the **area-so-far function**

$$F(x) = \int_a^x f(y)\,dy, \tag{34.54}$$

the signed area accumulated from a out to x. One function holds every definite integral of f: by additivity, the area over any interval $[x_1, x_2]$ is a difference of two accumulated areas,

$$\int_{x_1}^{x_2} f(x)\,dx = F(x_2) - F(x_1),$$

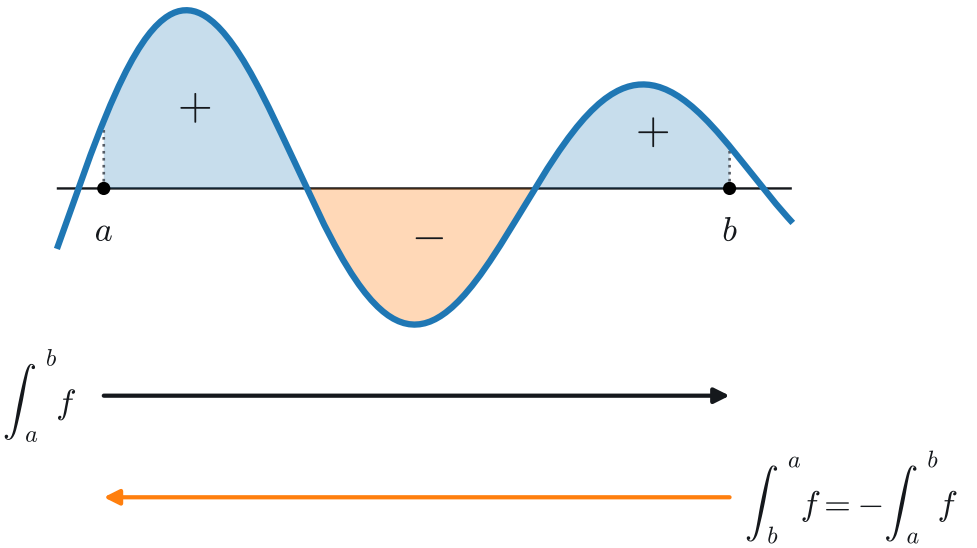


Figure 34.28: Signed area. Lobes above the axis count positive and the lobe below counts negative, so $\int_a^b f$ totals the shaded regions with their signs; reversing the limits traverses the same area right to left and negates the total.

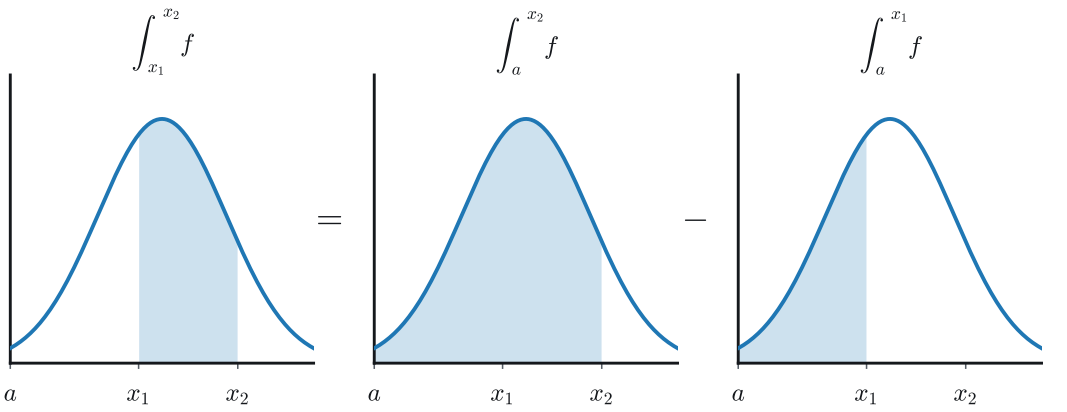


Figure 34.29: The area under f from x_1 to x_2 as the difference of two areas accumulated from the base point a at the plot's left edge: $\int_{x_1}^{x_2} f\,dx = \int_a^{x_2} f\,dx - \int_a^{x_1} f\,dx$.

as in Figure 34.29. So it suffices to find F.

How does F change when x moves a little to the right? Figure 34.30 shows the area out to x and the strip of width ϵ that moving to $x + \epsilon$ adds. The strip is nearly a rectangle of height $f(x)$, so $F(x + \epsilon) - F(x) \approx \epsilon f(x)$, and dividing by ϵ suggests $F'(x) = f(x)$.

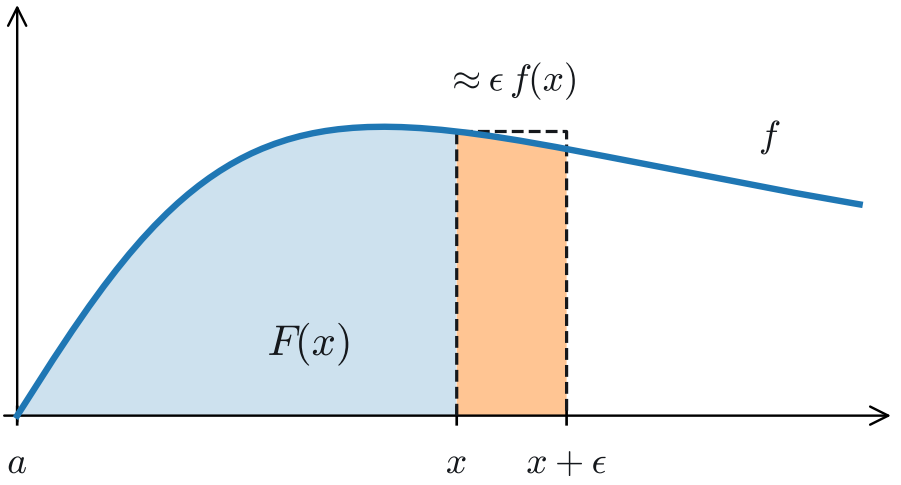


Figure 34.30: The area-so-far $F(x)$ under $f(y) = y/(1 + y^2)$ from $a = 0$ to $x = 1.2$ (light), and the strip that moving to $x + \epsilon$ adds (dark). The strip is close to the dashed rectangle of height $f(x)$, so $F(x + \epsilon) - F(x) \approx \epsilon f(x)$.

Theorem (Fundamental theorem of calculus). *Let f be Riemann integrable on $[a, b]$, let $F(x) = \int_a^x f(y)\, dy$, and let f be continuous at a point $x \in (a, b)$. Then F is differentiable at x and*

$$\frac{dF}{dx}(x) = f(x). \tag{34.55}$$

The rate at which area accumulates is the current height of the curve.

Proof. Nudge x by $\epsilon > 0$ and measure the new area. By Equation 34.54 and additivity, the increment is a single sliver,

$$F(x + \epsilon) - F(x) = \int_x^{x+\epsilon} f(y)\, dy.$$

Write $m_\epsilon = \inf_{[x,x+\epsilon]} f$ and $M_\epsilon = \sup_{[x,x+\epsilon]} f$ for the tightest constant bounds on f over the sliver; they are finite because an integrable function is bounded. Since $m_\epsilon \le f \le M_\epsilon$ on the sliver, monotonicity of the integral pins its area between two rectangles of width ϵ:

$$\epsilon\, m_\epsilon \;\le\; F(x + \epsilon) - F(x) \;\le\; \epsilon\, M_\epsilon.$$

Divide by ϵ. Because f is continuous at x, for any $\delta > 0$ the whole interval $[x, x+\epsilon]$ eventually lies where $|f(y)-f(x)| < \delta$. Hence m_ϵ and M_ϵ both converge to $f(x)$ as $\epsilon \to 0$, and the squeeze forces $\lim_{\epsilon\to 0} \frac{F(x+\epsilon)-F(x)}{\epsilon} = f(x)$. For $\epsilon < 0$ the sliver sits to the left of x, and the same bounds apply with both sign flips canceling in the quotient. That limit is $F'(x)$. ■

The proof needed only monotonicity and additivity, the two properties proved from the definition. The theorem turns the search for an area into the search for an **antiderivative**, a function G with $G' = f$, which we can read off the derivative table of Section 34.1.2. Any antiderivative gives the integral, through the **evaluation rule**: *if f is continuous on $[a, b]$ and $G' = f$ on all of $[a, b]$, then*

$$\int_a^b f(x)\,dx = G(b) - G(a). \tag{34.56}$$

The reason is that F and G have the same derivative f on (a, b), so $G - F$ has derivative zero there and is constant by the mean value theorem of Section 34.1.3; the constant cancels in $G(b) - G(a) = F(b) - F(a) = \int_a^b f\,dx$.

The hypothesis "on all of $[a, b]$" matters. Since $(-1/x)' = x^{-2}$, a careless use gives $\int_{-1}^{1} x^{-2}\,dx = [-1/x]_{-1}^{1} = -2$, a negative value for a positive integrand. The rule does not apply, because $G' = f$ fails at 0, where neither function is defined; the integral diverges (Section 34.6.2.2).

Each derivative read backwards is an integral: $(y^3)' = 3y^2$ gives $\int_0^1 y^2\,dy = \frac{1}{3}$ without a single slice, and $(e^t)' = e^t$ gives $\int_0^x e^t\,dt = e^x - 1$. For the running example take $G(x) = \frac{1}{2}\log(1 + x^2)$. By the chain rule $G'(x) = \frac{1}{2} \cdot \frac{2x}{1+x^2} = \frac{x}{1+x^2}$, so

$$\int_0^2 \frac{x}{1+x^2}\,dx = G(2) - G(0) = \tfrac{1}{2}\log 5 - \tfrac{1}{2}\log 1 = \tfrac{1}{2}\log 5,$$

the value the Riemann sums were approaching.

The error of the left rule. The evaluation rule also explains the constant 0.2 in the error of those sums. On the slice that starts at x_i, $f(x) \approx f(x_i) + f'(x_i)(x - x_i)$, so the area of the slice exceeds its rectangle $\epsilon f(x_i)$ by about $\frac{\epsilon^2}{2} f'(x_i)$. Summed over the slices, these excesses form $\frac{\epsilon}{2}$ times a left sum of f', which is close to $\frac{\epsilon}{2}\int_a^b f'\,dx = \frac{\epsilon}{2}(f(b) - f(a))$. To first order in ϵ, then,

$$\text{left sum} - \int_a^b f(x)\,dx \approx \frac{\epsilon}{2}(f(a) - f(b)).$$

For x^2 on $[0, 1]$ this is $-\frac{\epsilon}{2} = -\frac{1}{2N}$, the leading correction found by hand. For the running example, $f(0) = 0$ and $f(2) = 0.4$ give $-0.2\,\epsilon$. The formula is numerical analysis, not theory, but it accounts for the residue in three more outputs of this section, and it is the yardstick against which sampling is measured in Section 36.5.1.7.

The theorem says more than $\int_0^2 f = G(2) - G(0)$: the area-so-far equals $G(x) - G(0)$ at every x. To test this we build F as a cumulative left sum, one running total over the slices, and compare it with $G(x) - G(0)$ at every grid point. The two cannot agree exactly, because the left sum on $[0, x]$ carries the bias $\frac{\epsilon}{2}(f(0) - f(x))$. It is largest where f peaks, at $x = 1$ with $f(1) = \frac{1}{2}$, so the largest gap should be $\frac{\epsilon}{4}$.

```
G = lambda x: np.log(1 + x**2) / 2            # antiderivative of f
for eps in [1e-2, 1e-3, 1e-4]:
    x = eps * np.arange(round(2 / eps))       # left ends of the slices of [0, 2]
    F = np.cumsum(eps * f(x))                 # F[k]: left sum on [0, x_k + eps]
    gap = np.abs(F - (G(x + eps) - G(0))).max()
    print(f'eps={eps:.0e}  max over x of |F(x) - (G(x) - G(0))| = {gap:.2e}')
```

```
eps=1e-02  max over x of |F(x) - (G(x) - G(0))| = 2.51e-03
eps=1e-03  max over x of |F(x) - (G(x) - G(0))| = 2.50e-04
eps=1e-04  max over x of |F(x) - (G(x) - G(0))| = 2.50e-05
```

The largest gap is $\frac{\epsilon}{4}$ at each slice width, as predicted. The cumulative sum follows the antiderivative at every x, not only at $x = 2$.

34.6.2.2 Improper Integrals

The normal, exponential and many other densities live on unbounded domains, while Equation 34.53 defines integrals over a finite $[a, b]$ only. We extend the definition the way we made it, by a limit. The **improper integral** is

$$\int_a^\infty f(x)\,dx = \lim_{b\to\infty} \int_a^b f(x)\,dx, \tag{34.57}$$

the area out to b, in the limit of large b. If the limit exists and is finite the integral *converges*; otherwise it *diverges*. The integral over the whole line, $\int_{-\infty}^{\infty} f\,dx$, is split at any point into two one-sided improper integrals, and *both* must converge on their own. A symmetric limit is not enough: $\int_{-b}^{b} x\,dx = 0$ for every b, while $\int_0^\infty x\,dx = \infty$, so $\int_{-\infty}^{\infty} x\,dx$ does not exist.

The power law x^{-s} with exponent $s > 0$ is the standard test of whether a tail is thin enough to have finite area. The evaluation rule gives the definite integral over $[1, b]$, and then the limit:

$$\int_1^b x^{-s}\,dx = \frac{b^{1-s} - 1}{1 - s} \xrightarrow{b\to\infty} \begin{cases} \dfrac{1}{s-1}, & s > 1 \text{ (converges)}, \\ \infty, & s \le 1 \text{ (diverges)}. \end{cases}$$

(At $s = 1$ the formula reads $0/0$; the definite integral is $\log b$, its limit as $s \to 1$, and $\log b \to \infty$.) So $\int_1^\infty x^{-2}\,dx = 1$ while $\int_1^\infty x^{-1}\,dx = \infty$: the boundary between finite and infinite area sits at $s = 1$.

Power laws are the standard example of a **heavy-tailed** distribution (defined in Section 36.1, which also shows what such tails do to averages). For a density with tail $p(x) \propto x^{-s}$ the

normalizer is finite only if $s > 1$, and the mean, by the same test applied to $x \cdot x^{-s} = x^{1-s}$, only if $s > 2$.

The same idea handles an integrand that blows up at an endpoint: $\int_0^1 f\,dx = \lim_{a\to 0^+} \int_a^1 f\,dx$. For the power law the test reverses,

$$\int_0^1 x^{-s}\,dx = \lim_{a\to 0^+} \frac{1-a^{1-s}}{1-s} = \frac{1}{1-s} \quad \text{for } s < 1,$$

and the integral diverges for $s \geq 1$. At $s = \frac{1}{2}$ the spike $x^{-1/2}$ is infinite at 0, yet its area is 2. Near infinity a power must fall faster than $1/x$; near zero it must grow slower.

An improper integral computed numerically has two errors: we stop at a finite b, and we add slices of finite width. For $\int_0^\infty e^{-x}\,dx = 1$ the two can be told apart. The table prints the exact partial integrals $\int_0^b e^{-x}\,dx = 1 - e^{-b}$, read off the antiderivative $-e^{-x}$, beside a left sum of each with $\epsilon = 10^{-3}$.

```
for b in [1, 2, 5, 10, 20]:
    left = left_sum(lambda x: np.exp(-x), 0, b, 1e-3)
    print(f'b={b:<3}  left sum={left:.6f}  exact 1-e^-b={1 - math.exp(-b):.6f}')
```

```
b=1    left sum=0.632437  exact 1-e^-b=0.632121
b=2    left sum=0.865097  exact 1-e^-b=0.864665
b=5    left sum=0.993759  exact 1-e^-b=0.993262
b=10   left sum=1.000455  exact 1-e^-b=0.999955
b=20   left sum=1.000500  exact 1-e^-b=1.000000
```

The exact partials approach 1 quickly: the *truncation error* of stopping at b is e^{-b}, below the sixth decimal by $b = 20$. The left sums settle near 1.0005, not 1. That residue is *discretization error*, the left rule's bias $\frac{\epsilon}{2}(f(0) - f(b)) \to \frac{\epsilon}{2} = 0.0005$. To leading order the two errors add: a larger b removes the first, a smaller ϵ the second.

34.6.2.3 Integration by Parts

Every differentiation rule, read through the evaluation rule, becomes an integration rule. The product rule becomes integration by parts.

For continuously differentiable u and v on $[a, b]$,

$$\int_a^b u(x)\,v'(x)\,dx = \Big[\,u(x)\,v(x)\,\Big]_a^b - \int_a^b u'(x)\,v(x)\,dx, \tag{34.58}$$

where $[\,h(x)\,]_a^b$ abbreviates $h(b) - h(a)$. The proof is one line: by the product rule uv is an antiderivative of $u'v + uv'$, so Equation 34.56 gives $\int_a^b (u'v + uv')\,dx = [\,u(x)v(x)\,]_a^b$, and we subtract $\int_a^b u'v\,dx$ from both sides.

The formula moves the derivative from one factor to the other and adds a boundary term. It helps when the new integrand $u'v$ is simpler than the old one uv'. With $u(x) = x$ and $v'(x) = e^{-x}$, so that $u' = 1$ and $v = -e^{-x}$, apply Equation 34.58 on $[0, b]$ and let $b \to \infty$:

$$\int_0^\infty x\, e^{-x}\, dx = \Big[-x\, e^{-x} \Big]_0^\infty + \int_0^\infty e^{-x}\, dx = 0 + 1 = 1.$$

The boundary term vanishes because $xe^{-x} \to 0$ at both ends, and what remains is the improper integral computed above. The value is the mean of the exponential distribution (Section 36.3).

Expectations are integrals of products, and a common move is to shift a derivative from a factor we cannot handle onto one we can. The bell curve shows the move at its simplest. Take the bell weight $w(z) = e^{-z^2/2}$, which satisfies $w'(z) = -z\, w(z)$. Let g be continuously differentiable with $g(z)w(z) \to 0$ as $z \to \pm\infty$, and let the two integrals below converge. Integration by parts on $[-a, b]$, followed by the separate limits $a \to \infty$ and $b \to \infty$, gives

$$\int_{-\infty}^{\infty} g'(z)\, w(z)\, dz = \Big[g(z)\, w(z) \Big]_{-\infty}^{\infty} - \int_{-\infty}^{\infty} g(z)\, w'(z)\, dz = \int_{-\infty}^{\infty} z\, g(z)\, w(z)\, dz. \tag{34.59}$$

Under the bell weight, multiplying by z has the same effect as differentiating. Both conditions hold for every polynomial g. With $g(z) = z$ the identity reads $\int z^2 w\, dz = \int w\, dz$: the mean of z^2 under the bell weight is 1, whatever the normalizer $\int w\, dz$ turns out to be (it is $\sqrt{2\pi}$, by Equation 34.66). With $g(z) = z^3$ it reads $\int z^4 w = 3 \int z^2 w = 3 \int w$, so the mean of z^4 is 3. Once w is normalized into the standard normal density $p = w/\sqrt{2\pi}$, the identity reads $\mathbb{E}[g'(Z)] = \mathbb{E}[Z\, g(Z)]$, Stein's identity for the standard normal. Shifting a derivative from an unknown factor onto a known one is also the step that makes score matching work (Section 38.5, (Hyvärinen 2005)).

34.6.2.4 Substitution

The chain rule, read through the evaluation rule, becomes the substitution rule. Let u be continuously differentiable on $[a, b]$ and f continuous on an interval containing $u([a, b])$, with antiderivative G. The chain rule gives $(G \circ u)'(x) = f(u(x))\, u'(x)$, so

$$\int_a^b f(u(x))\, u'(x)\, dx = \int_a^b (G \circ u)'(x)\, dx = G(u(b)) - G(u(a)) = \int_{u(a)}^{u(b)} f(y)\, dy.$$

Read from right to left, this is the **change-of-variables formula**

$$\int_{u(a)}^{u(b)} f(y)\, dy = \int_a^b f(u(x))\, \frac{du}{dx}\, dx. \tag{34.60}$$

An integral over y can be computed as an integral over x, if the integrand carries the stretch factor du/dx. The map u need not be monotone (exercise 5).

A thin slice explains the factor. Cut $[a, b]$ into slices of width ϵ. The map u sends the slice at x to a slice of width about $\epsilon\, u'(x)$ at $y = u(x)$, where the integrand has height $f(y) = f(u(x))$. Its area, $f(u(x))\, u'(x)\, \epsilon$, is what the x-integral must add up (Figure 34.31).

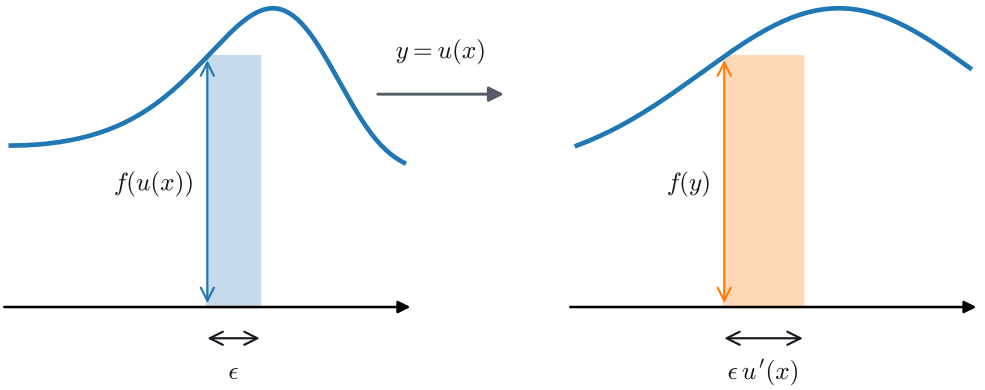


Figure 34.31: A thin sliver under the substitution $y = u(x)$. Left, the x side (the right-hand side of the formula): a sliver of width ϵ at x and height $f(u(x))$. Right, the y side: its image, of width $\epsilon\, u'(x)$ and the same height. Matching the two areas forces the factor du/dx.

The formula is used by recognizing the pattern $f(u(x))\, u'(x)$ in an integrand. In $\int_0^1 x\, e^{-x^2}\, dx$, take $u(x) = x^2$ with $u'(x) = 2x$: the integrand is $\frac{1}{2} e^{-u(x)} u'(x)$, so

$$\int_0^1 x\, e^{-x^2}\, dx = \frac{1}{2} \int_0^1 e^{-y}\, dy = \frac{1 - e^{-1}}{2} = 0.316.$$

The factor x supplied the $u'(x)$ that the rule requires. Without it, $\int e^{-x^2} dx$ has no elementary antiderivative, and this is the integral behind the normalizer $1/\sqrt{2\pi}$ of the bell curve. The Gaussian integral Equation 34.65 obtains the missing factor from a change to polar coordinates.

34.6.3 Multiple Integrals

From here on two theorems are quoted without proof, both from (Folland 1999): Fubini's theorem and the change of variables in many dimensions.

34.6.3.1 Double Integrals and Fubini's Theorem

The road to $\int e^{-x^2} dx$ leads through two dimensions, so we first extend the integral from intervals to regions of the plane. The bell in two dimensions, $f(x, y) = e^{-x^2 - y^2}$ over the square $[-2, 2]^2$, encloses a volume between its surface and the base plane. We compute it the way we computed areas: tile the base with $\epsilon \times \epsilon$ squares, stand on each square a box as tall as the surface, and add the box volumes (Figure 34.32). For f on a rectangle $R = [a, b] \times [c, d]$ the limit of these box sums as $\epsilon \to 0$ is the **double integral** $\iint_R f(x, y)\, dx\, dy$, the signed volume under the surface.

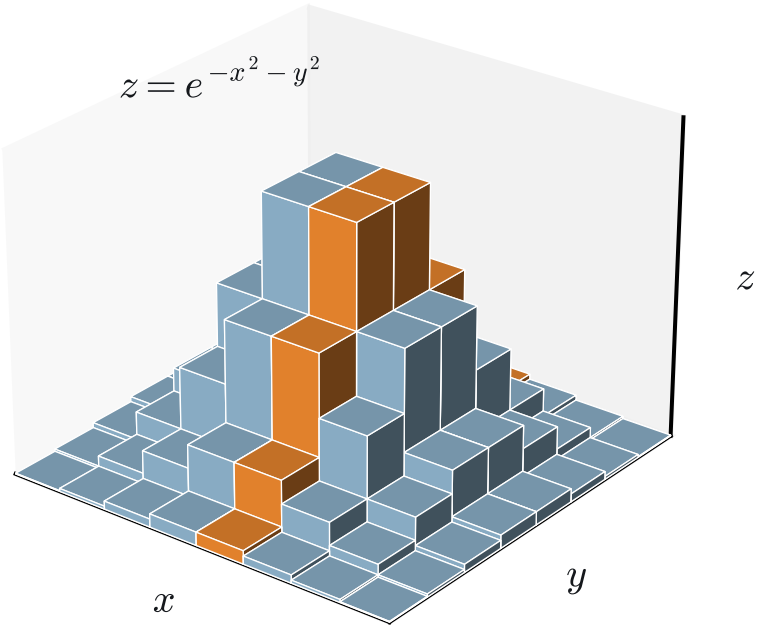


Figure 34.32: The bell surface $z = e^{-x^2-y^2}$ over the square $[-2, 2]^2$. Tiling the base with $\epsilon \times \epsilon$ squares, $\epsilon = 0.5$, and standing a box of height f on each approximates the volume under the surface; the highlighted column of boxes at one fixed x is one inner sum of Fubini's theorem.

How do we evaluate such a volume with one-dimensional tools? A finite sum can be totalled in any order. Index the squares by i along x and j along y, and total the boxes column by column (Figure 34.33):

$$\sum_{i,j} \epsilon^2 f(a+\epsilon i, c+\epsilon j) = \sum_i \epsilon \Big(\sum_j \epsilon\, f(a+\epsilon i, c+\epsilon j) \Big).$$

The inner sum is the left sum of $\int_c^d f(a+\epsilon i, y)\, dy$, the area of the slice of the surface at $x = a + \epsilon i$ (the highlighted column in Figure 34.32), and the outer sum is the left sum of that slice area over x.

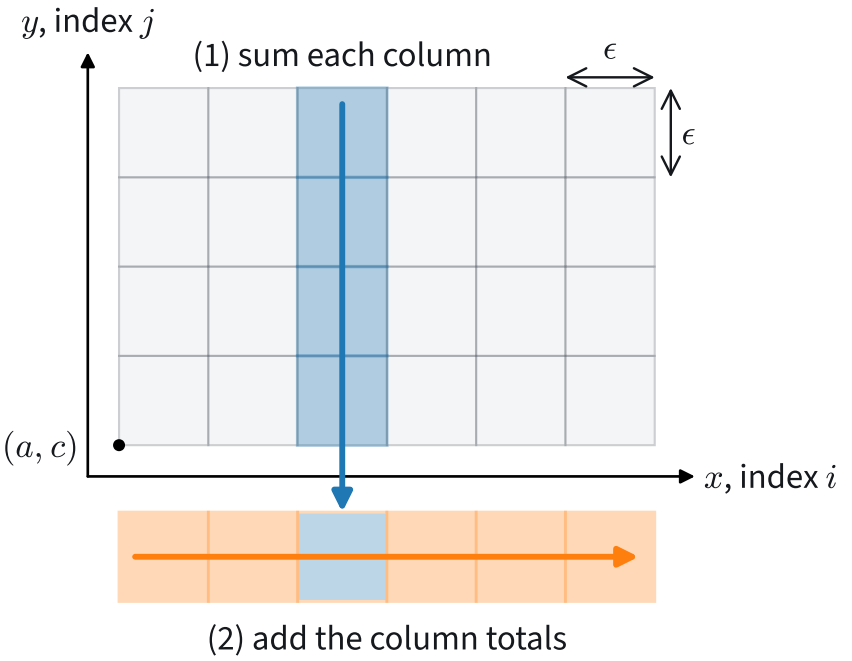


Figure 34.33: Totalling a grid of $\epsilon \times \epsilon$ cells with corner (a, c), index i along x and j along y. (1) Sum each column, at fixed i, over j. (2) Add the column totals from left to right. Reordering a finite sum changes nothing; this is the discrete fact behind iterated integration.

For finite grids the reordering is trivial. That it survives the limit $\epsilon \to 0$ is a theorem, and it needs a hypothesis. **Fubini's theorem** states: if f is **absolutely integrable** on R, that is $\iint_R |f(x,y)|\, dx\, dy < \infty$, then the double integral equals either iterated integral,

$$\iint_R f(x,y)\, dx\, dy = \int_c^d \left(\int_a^b f(x,y)\, dx \right) dy = \int_a^b \left(\int_c^d f(x,y)\, dy \right) dx. \tag{34.61}$$

A volume can be computed slice by slice, slicing in either direction. For $f \geq 0$ the same conclusion holds with no hypothesis, all three sides possibly $+\infty$ (*Tonelli's theorem*). Densities,

squared errors and every other nonnegative integrand therefore need no check; a signed integrand needs $\iint |f| < \infty$ first. The theorem is at work whenever a variable is integrated out of a joint density and whenever an expectation changes places with another integral.

The case used most often is a product integrand $f(x,y) = g(x)h(y)$. The inner integral $\int_c^d g(x)h(y)\,dy = g(x)\int_c^d h(y)\,dy$ is $g(x)$ times a constant, so

$$\iint_R g(x)\,h(y)\,dx\,dy = \int_a^b g(x)\,dx \int_c^d h(y)\,dy. \tag{34.62}$$

A product integrand integrates factor by factor. The bell is such a product, $e^{-x^2-y^2} = e^{-x^2}e^{-y^2}$, and the Gaussian integral below rests on this fact.

The hypothesis is needed. Take $f(x,y) = (x^2-y^2)/(x^2+y^2)^2$ on $[0,1]^2$. A short computation gives $\partial_y(y/(x^2+y^2)) = f(x,y)$, so by the evaluation rule $\int_0^1 f(x,y)\,dy = \frac{1}{1+x^2}$ for $x > 0$, and then $\int_0^1 \frac{dx}{1+x^2} = \arctan 1 = \frac{\pi}{4}$. Since $f(x,y) = -f(y,x)$, the other order gives $-\frac{\pi}{4}$. The two orders can disagree only because $\iint |f| = \infty$: near the origin $|f|$ grows like the inverse square of the distance, and its volume diverges.

When the region D is not a rectangle, $\int_D f(\mathbf{x})\,d\mathbf{x}$ is defined by *zero-extension*: set f to 0 outside D and integrate over any rectangle containing D.

34.6.3.2 Change of Variables in Many Dimensions

The one-dimensional stretch factor du/dx has a many-dimensional successor, and the linear case shows it. A linear map scales every area by the absolute value of its determinant (Section 33.2); the sign records orientation and plays no role in area. The map $\phi(u,v) = (au, bv)$ with $a, b > 0$ has determinant ab and sends the unit disc, of area π, to the ellipse $x^2/a^2 + y^2/b^2 \le 1$, whose area is therefore πab.

A nonlinear map is linear on a small scale, with its Jacobian as the linear part, so the scale factor varies from point to point. Let $\Omega \subseteq \mathbb{R}^n$ be open and let $\phi : \Omega \to \mathbb{R}^n$ be a C^1-**diffeomorphism** onto its image: ϕ is continuously differentiable, injective, and its Jacobian $\mathbf{J}_\phi$, the matrix of partial derivatives of Section 34.4, has $\det \mathbf{J}_\phi(\mathbf{x}) \neq 0$ everywhere on Ω. Injectivity stops the map from folding space onto itself, and the nonzero determinant keeps it from collapsing a region to lower dimension. Then, for absolutely integrable f,

$$\int_{\phi(\Omega)} f(\mathbf{y})\,d\mathbf{y} = \int_\Omega f(\phi(\mathbf{x}))\,\bigl|\det \mathbf{J}_\phi(\mathbf{x})\bigr|\,d\mathbf{x}. \tag{34.63}$$

A tiny cube of volume $d\mathbf{x}$ at $\mathbf{x}$ is sent to a tiny parallelepiped of volume $|\det \mathbf{J}_\phi(\mathbf{x})|\,d\mathbf{x}$ at $\mathbf{y} = \phi(\mathbf{x})$, and the factor in the integrand accounts for that distortion. Figure 34.34 puts the one- and two-dimensional pictures side by side.

The formula uses $|\det \mathbf{J}_\phi|$ and not $\det \mathbf{J}_\phi$ because its left side integrates over a set, not between oriented limits. In one dimension the order of the limits carries the sign: for a decreasing u the limits in Equation 34.60 run backwards and $u' < 0$, the two signs cancel, and the rule can be written over the image interval with $|u'|$. Regions in $\mathbb{R}^n$ have no order, so only this unsigned form generalizes (exercise 5).

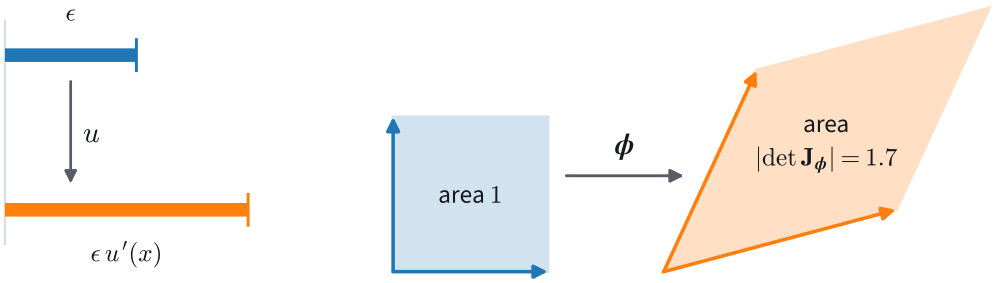


Figure 34.34: Left: the one-dimensional substitution scales a segment of length ϵ by $u'(x)$. Right: in two dimensions a linear ϕ sends the unit square to a parallelogram whose area is the volume factor $|\det \mathbf{J}_\phi|$.

Read with f a density, the formula says how a density changes under a map, which is how a normalizing flow computes the density of its output (Equation 36.30 in Section 36.2).

34.6.3.3 The Gaussian Integral

The normalizer $1/\sqrt{2\pi}$ of the bell curve comes from the one-dimensional integral $\int_{-\infty}^{\infty} e^{-x^2}\,dx$. The fundamental theorem gives no closed form for it. An antiderivative exists, since e^{-x^2} is continuous, but it is a new function, not a combination of the familiar ones. Surprisingly, the value over the whole line can be found all the same, by a detour through two dimensions. The argument takes three steps and is the most elegant of the section. (Normalized, the antiderivative is the **error function** $\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,dt$, which tends to 1 as $x \to \infty$ by the result below. Libraries provide it as `math.erf`.)

I. Square and factor. On the square $[-b, b]^2$ the product rule Equation 34.62 gives $\iint_{[-b,b]^2} e^{-x^2-y^2}\,dx\,dy = \left(\int_{-b}^{b} e^{-x^2}\,dx\right)^2$. Let $b \to \infty$ on both sides:

$$\left(\int_{-\infty}^{\infty} e^{-x^2}\,dx\right)^2 = \lim_{b\to\infty} \iint_{[-b,b]^2} e^{-x^2-y^2}\,dx\,dy = \iint_{\mathbb{R}^2} e^{-x^2-y^2}\,dx\,dy. \tag{34.64}$$

The square of the one-dimensional integral is the volume under the two-dimensional bell, and that bell is rotationally symmetric.

II. Polar coordinates. Rotational symmetry calls for the map $\phi(r, \theta) = (r\cos\theta, r\sin\theta)$ on the open set $\Omega = (0, \infty) \times (0, 2\pi)$, where it is injective. Its Jacobian determinant is

$$|\det \mathbf{J}_\phi| = \left|\det \begin{bmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{bmatrix}\right| = r(\cos^2\theta + \sin^2\theta) = r.$$

A polar cell with sides dr along the ray and $r\,d\theta$ along the arc has area $r\,dr\,d\theta$ (Figure 34.35): cells far from the origin are larger. This factor r is the missing factor x of the substitution example $\int_0^1 x\,e^{-x^2}\,dx$.

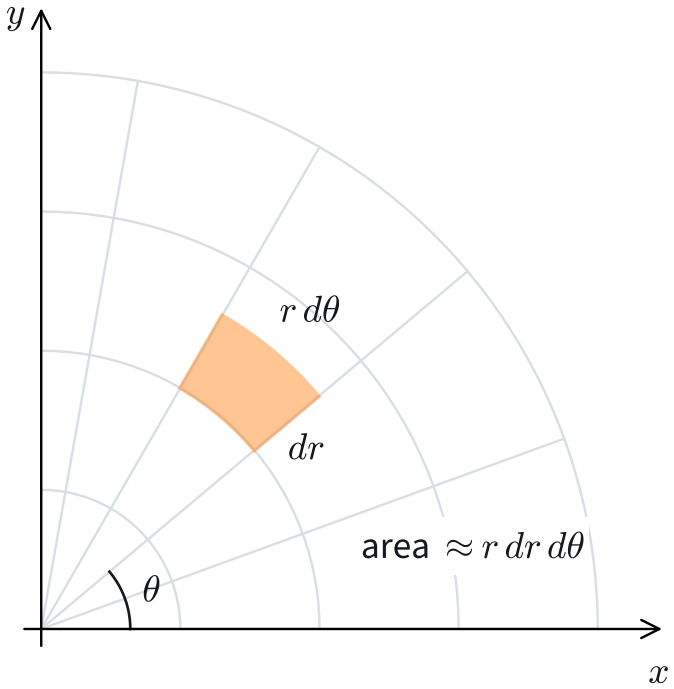


Figure 34.35: A polar grid with one cell filled: radial side dr, arc side $r\,d\theta$, so its area is about $r\,dr\,d\theta$. The change to polar coordinates multiplies the integrand by $|\det \mathbf{J}_\phi| = r$.

III. Integrate. By Equation 34.63, and because $-\frac{1}{2}e^{-r^2}$ is an antiderivative of re^{-r^2},

$$\iint_{\mathbb{R}^2} e^{-x^2-y^2}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} r\,e^{-r^2}\,dr\,d\theta = 2\pi\cdot\frac{1}{2} = \pi.$$

Taking the positive square root of Equation 34.64,

$$\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}. \tag{34.65}$$

The area under e^{-x^2} is $\sqrt{\pi} = 1.7725$.

Three points make the argument rigorous. For a nonnegative integrand the integrals over growing squares, or growing discs, increase to one common limit, because every disc lies inside some square and every square inside some disc; so the shape of the exhaustion does not matter. The polar map misses the ray $\theta = 0$, which has zero area and contributes nothing. And the change of variables is applied on the bounded sets $(0,b)\times(0,2\pi)$, where the integrand is bounded and continuous, before letting $b\to\infty$.

The normalizer of the bell curve now follows by one substitution. With $z = \sqrt{2}\,x$, Equation 34.60 gives

$$\int_{-\infty}^{\infty} e^{-z^2/2}\,dz = \sqrt{2}\int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{2\pi}. \tag{34.66}$$

Dividing the bell $e^{-z^2/2}$ by $\sqrt{2\pi} = 2.5066$ makes its total area 1. This is where the constant in the standard normal density comes from.

Do the squares exhaust the plane as step I claims, and is the limit π? The cell computes the grid sum over $[-b,b]^2$ at $\epsilon = 0.01$, the exact square value $\left(\int_{-b}^{b} e^{-x^2}dx\right)^2 = \pi\,\mathrm{erf}(b)^2$ from the product rule, and π.

```
eps = 0.01
for b in [1, 2, 3, 6]:
    g = -b + eps * np.arange(round(2 * b / eps))    # left ends of the slices
    x, y = np.meshgrid(g, g, indexing='ij')
    grid = np.sum(eps**2 * np.exp(-x**2 - y**2))
    print(f'b={b}  grid sum={grid:.6f}  pi*erf(b)^2={math.pi * math.erf(b)**2:.6f}'
          f'  pi={math.pi:.6f}')
```

```
b=1  grid sum=2.230949  pi*erf(b)^2=2.230985  pi=3.141593
b=2  grid sum=3.112266  pi*erf(b)^2=3.112270  pi=3.141593
b=3  grid sum=3.141454  pi*erf(b)^2=3.141454  pi=3.141593
b=6  grid sum=3.141593  pi*erf(b)^2=3.141593  pi=3.141593
```

The grid sums match $\pi\,\mathrm{erf}(b)^2$ and climb to π; by $b = 3$ the volume outside the square is about 10^{-4}. The agreement to four or more decimals at $\epsilon = 0.01$ has a reason: for an even integrand on a symmetric interval the left rule's first-order bias $\frac{\epsilon}{2}\big(f(-b) - f(b)\big)$ vanishes.

The probability $P(-1 \le Z \le 1) \approx 0.68$ of the standard normal is now a computation as well. The substitution $z = \sqrt{2}\,t$ turns the area under the density $p(z) = e^{-z^2/2}/\sqrt{2\pi}$ into an error function:

$$P(-1 \le Z \le 1) = \frac{1}{\sqrt{2\pi}} \int_{-1}^{1} e^{-z^2/2}\,dz = \frac{1}{\sqrt{\pi}} \int_{-1/\sqrt{2}}^{1/\sqrt{2}} e^{-t^2}\,dt = \operatorname{erf}(1/\sqrt{2}) = 0.683.$$

Both questions about the bell curve are thereby answered, and the puzzle that $p(0) = 0.40$ while $P(Z = 0) = 0$ dissolves: a density is probability per unit length, and a single point is an interval of width zero.

One question from the beginning of the section is still open. Training through random sampling needs to know how such an area changes when a parameter of the density moves, and for that we must differentiate an integral.

34.6.4 Summary

The definite integral is the limit of sums of thin slices, a signed area that the fundamental theorem evaluates: the area-so-far has derivative $F' = f$, so $\int_a^b f\,dx = G(b) - G(a)$ for any antiderivative G. Integration by parts and substitution are the product and chain rules read backwards, and in many dimensions the stretch factor becomes $|\det \mathbf{J}_\phi|$, the local volume factor; with polar coordinates it gives $\int e^{-x^2}\,dx = \sqrt{\pi}$, hence the normal constant $\sqrt{2\pi}$.

34.6.5 Exercises

1. code **Midpoint versus left rule.** The function `left_sum` evaluates the left-endpoint rule. The midpoint rule instead evaluates f at the center of each slice of width ϵ. On $[a, b]$ with $n = (b - a)/\epsilon$ slices and $x_i = a + i\epsilon$, it returns $\epsilon \sum_{i=1}^{n} f(x_{i-1} + \frac{\epsilon}{2})$.

    1. The text derives the error of the left rule on $[a, b]$, $\frac{\epsilon}{2}(f(a) - f(b))$, from a first-order expansion of f on each slice. The left sum of x^2 on $[0, 1]$ has error $-\frac{1}{2N} + \frac{1}{6N^2}$; which term does the expansion predict? Use the estimate to predict the errors that the text's cell prints at $\epsilon = 10^{-1}, \dots, 10^{-4}$ for $\int_0^2 \frac{x}{1+x^2}\,dx$. Do the sums lie above or below $\frac{1}{2}\log 5$?
    2. Implement both rules for an integrand f on $[a, b]$. Evaluate $\int_1^2 \frac{1}{x}\,dx$ with an antiderivative, and tabulate the error of both rules for $\epsilon \in \{10^{-1}, 10^{-2}, 10^{-3}\}$. By what factor does each error shrink when ϵ shrinks by a factor of ten?
    3. Expand f to second order about the center of each slice and show that the integral minus the midpoint sum is approximately $\frac{\epsilon^2}{24}(f'(b) - f'(a))$. Compare this prediction with your table for $\int_1^2 \frac{1}{x}\,dx$. For which integrands does the midpoint rule underestimate the integral?

    Adapted from MIT 18.01SC, Unit 3 exercises, Problems 3G-1 and 3G-5.

2. **Activations as accumulated areas.** Let H be the unit step, with $H(t) = 1$ for $t > 0$ and $H(t) = 0$ for $t \le 0$. Let $\phi(t) = e^{-t^2/2}/\sqrt{2\pi}$ be the standard Gaussian density

and $\Phi(x) = \int_{-\infty}^{x} \phi(t)\,dt$ its distribution function.

1. Show that $\int_{-\infty}^{x} H(t)\,dt = \mathrm{ReLU}(x)$. Which hypothesis of the fundamental theorem Equation 34.55 fails at $x = 0$? What are the one-sided derivatives of the integral there?
2. Write $\Phi(x) = \Phi(0) + \int_{0}^{x} \phi(t)\,dt$ to conclude that $\Phi' = \phi$. Then substitute $t = \sqrt{2}\,s$ to show that $\Phi(x) = \frac{1}{2}(1 + \mathrm{erf}(x/\sqrt{2}))$.
3. The GELU activation of Section 4.1 is $x\Phi(x)$. Derive its derivative, find the points where the derivative is largest and smallest, and evaluate it there with `math.erf`. In which interval does the slope of GELU lie, and how does that interval compare with the slopes 0 and 1 of ReLU?

3. **Heavy tails.** The *Pareto density* with tail index $\alpha > 0$ is $p(x) = \alpha x^{-\alpha-1}$ for $x \geq 1$ and $p(x) = 0$ otherwise. The *Cauchy density* is $p(x) = \frac{1}{\pi(1+x^2)}$ on $\mathbb{R}$.
   1. Show that the Pareto density is normalized for every $\alpha > 0$. For which α are $\mathbb{E}[X]$ and $\mathbb{E}[X^2]$ finite? Compute $\mathbb{E}[X]$ where it is finite.
   2. Show that the Cauchy density is normalized, using the antiderivative arctan x of $\frac{1}{1+x^2}$.
   3. Show that $\int_{-b}^{b} x\,p(x)\,dx = 0$ for the Cauchy density and every $b > 0$. Why does the Cauchy density nevertheless have no mean under the definition of $\int_{-\infty}^{\infty}$ that follows Equation 34.57? How fast does $\int_{0}^{b} x\,p(x)\,dx$ grow with b?

4. **Gamma function by parts.** The *Gamma function* is $\Gamma(t) = \int_{0}^{\infty} u^{t-1}e^{-u}\,du$. For $t < 1$ the integrand is unbounded at $u = 0$, and the integral near 0 is defined by a limit, $\int_{0}^{1} = \lim_{a\to 0^+} \int_{a}^{1}$, as in the subsection Section 34.6.2.2.
   1. For which t does the integral converge at $u = 0$? Compare the condition with the test for $\int_{0}^{1} x^{-s}\,dx$ in the subsection Section 34.6.2.2.
   2. Integrate by parts Equation 34.58 at a general $t > 0$ to show that $\Gamma(t+1) = t\,\Gamma(t)$, and conclude that $\Gamma(n+1) = n!$ for every integer $n \geq 0$. Where does the argument use $t > 0$? Use the result to compute the variance of the exponential density e^{-u} on $u \geq 0$, whose mean Section 34.6.2.3 computes.
   3. Substitute $u = x^2$ and use the Gaussian integral Equation 34.65 to evaluate $\Gamma(\frac{1}{2})$.
   4. The product rule Equation 34.62, applied d times, gives $\int_{\mathbb{R}^d} e^{-\|\mathbf{x}\|^2}\,d\mathbf{x} = \pi^{d/2}$. In polar form the same integral is $S_{d-1}\int_{0}^{\infty} r^{d-1}e^{-r^2}\,dr$, where S_{d-1} is the surface area of the unit sphere in $\mathbb{R}^d$. Show that the radial integral is $\frac{1}{2}\Gamma(d/2)$ and deduce $S_{d-1} = 2\pi^{d/2}/\Gamma(d/2)$. Integrate $S_{d-1}r^{d-1}$ over $r \in [0,1]$ to show that the unit ball has volume $V_d = \pi^{d/2}/\Gamma(d/2+1)$. Evaluate V_d for $d \in \{1,2,3,5,10,20\}$ and explain why $V_d \to 0$. A midpoint grid with three points per axis on the cube $[-1,1]^{10}$ has 3^{10} points; what fraction of them lies inside the unit ball, and what does that say about grids in high dimensions?

5. **Orientation in substitution.** The one-dimensional rule Equation 34.60 integrates between oriented limits and carries the factor $\frac{du}{dx}$. The n-dimensional rule Equation 34.63 integrates over a set and carries $\left|\det \mathbf{J}_{\phi}\right|$.
   1. Let u be continuously differentiable and strictly monotone on $[a,b]$, either in-

creasing or decreasing. Show that

$$\int_{u([a,b])} f(y)\,dy = \int_a^b f(u(x))\,|u'(x)|\,dx, \tag{34.67}$$

where the left side integrates over the interval $u([a,b])$ from its smaller to its larger endpoint. Check the decreasing case with $u(x) = e^{-x^2}$ and $f = 1$ on $[0,1]$.

2. Take $u(x) = x^2$ on $[-1,1]$ and $f(y) = y$. Evaluate both sides of Equation 34.60, and both sides of the absolute-value form of (a). Which of the two forms remains valid for this u, which is not injective? Explain the discrepancy in the other form.

6. **Swapping the order.** Fubini's theorem Equation 34.61, together with zero-extension for regions that are not rectangles, lets us choose the order of integration.

   1. The inner integral of $\int_0^1 (\int_y^1 e^{-x^2}\,dx)\,dy$ has no elementary antiderivative. Sketch the triangular region of integration, reverse the order, and evaluate the integral in closed form.
   2. Let $X \geq 0$ have density p. Write $x = \int_0^x dt$ inside $\mathbb{E}[X] = \int_0^\infty x\,p(x)\,dx$ and swap the two integrals to show that $\mathbb{E}[X] = \int_0^\infty P(X > t)\,dt$. Why is the swap valid even when $\mathbb{E}[X] = \infty$? Check the formula on the density $\lambda e^{-\lambda x}$ for $x \geq 0$.

7. **Gaussian normalizers and moments.** The text normalizes the standard normal and computes its moments with the Stein identity Equation 34.59 for the bell weight. Let $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\!\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$ with $\sigma > 0$.

   1. Substitute $x = \mu + \sigma z$ and use Equation 34.66 to show that p is normalized. Which μ and σ^2 does the density $\frac{1}{\sqrt{\pi}} e^{-x^2}$ have?
   2. Let X have density p, and let g be continuously differentiable with $g(x)\,p(x) \to 0$ as $x \to \pm\infty$. Use $p'(x) = -\frac{x-\mu}{\sigma^2}\,p(x)$ and integration by parts to show that

$$\mathbb{E}\big[(X-\mu)\,g(X)\big] = \sigma^2\,\mathbb{E}\big[g'(X)\big]. \tag{34.68}$$

   Deduce that $\mathbb{E}[(X-\mu)^2] = \sigma^2$ and $\mathbb{E}[(X-\mu)^4] = 3\sigma^4$.

   3. Let $\Sigma = \mathbf{L}\mathbf{L}^\top$ be a $d \times d$ covariance matrix with invertible $\mathbf{L}$. Use the change of variables $\mathbf{x} = \mathbf{L}\mathbf{z}$ in Equation 34.63 and the product rule Equation 34.62 to show that

$$\int_{\mathbb{R}^d} \exp\!\left(-\tfrac{1}{2}\mathbf{x}^\top \Sigma^{-1}\mathbf{x}\right) d\mathbf{x} = (2\pi)^{d/2}\,(\det \Sigma)^{1/2}. \tag{34.69}$$

34.7 Expectations and Their Gradients

Every loss we have minimized was an average: over a minibatch, over the masks of dropout, over the noise added to an input. As long as the randomness does not depend on the parameters, such an average is easy to differentiate. The derivative of an average is the average of

the derivatives, and minibatch gradient descent computes it without a second thought. The difficulty begins when the parameters control the distribution itself. A variational autoencoder draws its latent code from a distribution that its encoder outputs. A policy-gradient method (Section 14.6) draws actions from the very policy it is trying to improve. The loss is then an expectation over samples whose distribution moves with the parameters, and a sample, once drawn, is just a number: it does not remember the parameter that produced it.

Here is the smallest such problem. Let $X \sim \mathcal{N}(\theta, 1)$, a normal variable with mean θ, and let the loss be

$$\ell(\theta) = \mathbb{E}\big[X^2\big] = \theta^2 + 1.$$

The value follows by writing $X = \theta + Z$ with Z standard normal: $\mathbb{E}[(\theta + Z)^2] = \theta^2 + 2\theta\,\mathbb{E}[Z] + \mathbb{E}[Z^2]$, where $\mathbb{E}[Z] = 0$ by symmetry and $\mathbb{E}[Z^2] = 1$ (Section 34.6.2.3). At $\theta = 1.5$ the derivative is $\ell'(1.5) = 3$. A real model can draw samples of X, but it cannot write ℓ in closed form. How do we estimate $\ell'(\theta)$ from samples?

Calculus gives two answers, and they differ in kind. Both write the expectation as an integral (Section 34.6) and move the derivative inside it. The *score-function* estimator then differentiates the density and leaves the samples alone. The *pathwise* estimator differentiates the samples and leaves the density alone. Neither always wins, and comparing the two (Section 34.7.3) is the main business of this section. Before that we recall what an expectation is, and settle when a derivative may be moved inside an integral, which is the step that needs care.

34.7.1 Expectations as Integrals

Before anything is differentiated we fix what an expectation is. The probability chapter treats the two definitions below at length (Section 36.1); here they are a recall, in the language of the previous section. A probability **density** on a set $\mathcal{X}$ is a nonnegative function whose total integral is 1:

$$p(x) \geq 0, \qquad \int_{\mathcal{X}} p(x)\,dx = 1. \tag{34.70}$$

A random variable X with density p falls in an interval with probability equal to the area above it, $P(a \leq X \leq b) = \int_a^b p(x)\,dx$ (Equation 36.3). Being a probability per unit length and not a probability, a density can exceed 1: the uniform density on $[0, \frac{1}{2}]$ equals 2. Our standard example is the **standard normal** $\mathcal{N}(0, 1)$ with density $p(z) = e^{-z^2/2}/\sqrt{2\pi}$, whose constant is the Gaussian integral of Section 34.6.

An **expectation** averages a function of X, weighting each value by its density:

$$\mathbb{E}[X] = \int_{\mathcal{X}} x\,p(x)\,dx, \qquad \mathbb{E}[g(X)] = \int_{\mathcal{X}} g(x)\,p(x)\,dx. \tag{34.71}$$

The expectation $\mathbb{E}[g(X)]$ is defined when $\int_{\mathcal{X}} |g(x)|\,p(x)\,dx < \infty$; the Cauchy density of exercise 3 has no mean for this reason.

Most expectations in deep learning have no closed form, and many are over thousands of dimensions, where no grid is affordable. They are estimated by drawing samples $x_1, \dots, x_n$ from p and averaging, $\frac{1}{n}\sum_i g(x_i)$. This is the *Monte Carlo* estimate. If one sample $g(X)$ has standard deviation σ, the estimate has root-mean-square error $\sigma/\sqrt{n}$, whatever the dimension (Section 36.5.1.7). The gradient estimators of this section are Monte Carlo estimates of this kind. Two unbiased estimators of the same gradient differ only in the spread of a single sample, so its standard deviation σ is what we compare.

34.7.2 Differentiation under the Integral Sign

34.7.2.1 The Swap

The loss $\ell(\theta) = \mathbb{E}[X^2]$ is an integral that depends on a parameter. A parameter can enter an integral in two places: through the limits of integration, which the fundamental theorem Equation 34.55 handles, or through the integrand.

For the integrand, the derivative passes inside the integral:

$$\frac{d}{d\theta}\int f(x,\theta)\,dx = \int \frac{\partial f}{\partial \theta}(x,\theta)\,dx. \tag{34.72}$$

The rate of change of the area is the integral of the rate of change of the height. We call this exchange of derivative and integral the *swap*, and both gradient estimators rest on it. It looks innocent, since a derivative is a limit of difference quotients and an integral a limit of sums, and finite sums and differences commute. But two limits are being exchanged, and that needs a hypothesis. On a bounded range of x it holds when $\partial_\theta f$ is continuous in both variables. On an infinite range it needs in addition an integrable function of x that bounds $|\partial_\theta f(x,\theta')|$ for all θ' near θ (a consequence of dominated convergence, quoted without proof from (Folland 1999)). For $f(x,\theta) = x^2 p_\theta(x)$ with the normal density such a bound exists, because the Gaussian factor dominates every polynomial.

34.7.2.2 The Leibniz Rule

The swap alone is wrong when the limits of integration move with the parameter. The Leibniz rule supplies the correction: the two effects add (Figure 34.36).

Proposition (Leibniz rule). *Let f and $\partial_\theta f$ be continuous on a region containing $\{(x,\theta) : x \in [a(\theta), b(\theta)]\}$ for θ in an open interval, and let a and b be differentiable. Then*

$$\frac{d}{d\theta}\int_{a(\theta)}^{b(\theta)} f(x,\theta)\,dx = \int_{a(\theta)}^{b(\theta)} \frac{\partial f}{\partial \theta}(x,\theta)\,dx + f(b(\theta),\theta)\,b'(\theta) - f(a(\theta),\theta)\,a'(\theta). \tag{34.73}$$

Proof. Let $H(\alpha,\beta,\theta) = \int_\alpha^\beta f(x,\theta)\,dx$. By the fundamental theorem, $\partial_\beta H = f(\beta,\theta)$ and $\partial_\alpha H = -f(\alpha,\theta)$. By the swap on the bounded range $[\alpha,\beta]$, $\partial_\theta H = \int_\alpha^\beta \partial_\theta f\,dx$. All three partial derivatives are continuous, so the chain rule of Section 34.3 applies to $\theta \mapsto H(a(\theta),b(\theta),\theta)$ and adds the three contributions, weighted by $a'(\theta)$, $b'(\theta)$ and 1. ∎

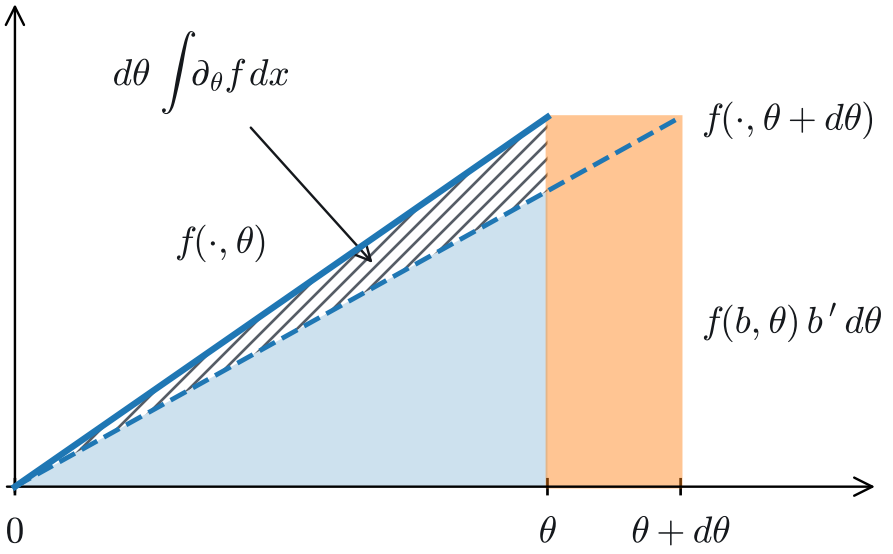


Figure 34.36: The Leibniz rule for $f(x, \theta) = x/\theta$ on $[0, \theta]$, the uniform example. When θ grows by $d\theta$ the curve changes to $f(\cdot, \theta + d\theta)$ (dashed; here it falls, so the hatched band between the curves is lost area), and the right edge moves, sweeping in the strip of height $f(b, \theta)$ and width $b' \, d\theta$.

Dropping the boundary term gives a wrong answer. Let X be uniform on $[0, \theta]$, with density $1/\theta$ there. Then $\mathbb{E}[X] = \int_0^\theta \frac{x}{\theta} \, dx = \frac{\theta}{2}$, with derivative $+\frac{1}{2}$. Differentiating only the integrand gives $\int_0^\theta -\frac{x}{\theta^2} \, dx = -\frac{1}{2}$, the wrong sign. The boundary term $f(\theta, \theta)\, b'(\theta) = \frac{\theta}{\theta} \cdot 1 = 1$ restores $-\frac{1}{2} + 1 = +\frac{1}{2}$. Whenever the support of a density moves with the parameter, the plain swap is not enough (exercise 1 does $\mathbb{E}[X^2]$).

34.7.2.3 Log-Integral-Exp

The swap also gives the continuous version of this chapter's log-sum-exp. Differentiating $\log \int e^{\theta x} p(x) \, dx$ with the chain rule and Equation 34.72 gives

$$\frac{d}{d\theta} \log \int e^{\theta x} p(x) \, dx = \frac{\int x \, e^{\theta x} p(x) \, dx}{\int e^{\theta x} p(x) \, dx},$$

the mean of x under the *tilted* density $e^{\theta x} p(x) / \int e^{\theta x} p$. For the standard normal, completing the square, $\theta z - \frac{1}{2} z^2 = \frac{1}{2}\theta^2 - \frac{1}{2}(z - \theta)^2$, shows that the tilted density is $\mathcal{N}(\theta, 1)$ and that $\log \mathbb{E}[e^{\theta Z}] = \theta^2/2$, whose derivative θ is indeed the tilted mean. The gradient of log-sum-exp is the softmax; the gradient of log-integral-exp is an expectation. This is the derivative $A'(\eta)$ of the log-partition function in Section 36.4.

34.7.3 Score-Function and Pathwise Gradients

By the swap, the derivative of an expectation is an integral, $\int g(x)\,\partial_\theta p_\theta(x)\,dx$. Unfortunately it is not an expectation: $\partial_\theta p_\theta$ is not a density, and we cannot draw samples from it. Two identities turn the integral into an expectation that samples can estimate.

34.7.3.1 The Score-Function Estimator

Write p_θ for a density that depends on parameters θ. The chain rule for log gives the **log-derivative identity**

$$\nabla_\theta\, p_\theta(x) = p_\theta(x)\,\nabla_\theta \log p_\theta(x). \tag{34.74}$$

The gradient of a density is the density times the gradient of its logarithm, the **score** $\nabla_\theta \log p_\theta$. When the swap is valid and the support of p_θ does not move with θ, the score has mean zero: $\mathbb{E}[\nabla_\theta \log p_\theta(X)] = \int \nabla_\theta p_\theta\,dx = \nabla_\theta \int p_\theta\,dx = \nabla_\theta 1 = \mathbf{0}$. The uniform density on $[0,\theta]$ shows why the support matters: its score is $\partial_\theta \log(1/\theta) = -1/\theta$, whose mean is $-1/\theta \neq 0$, because differentiating $\int_0^\theta p_\theta = 1$ also produces the boundary term $p_\theta(\theta) = 1/\theta$. The mean-zero score is the starting point of Fisher information (Section 36.6).

Multiplying the identity by g and integrating gives the **score-function identity**, for g that does not depend on θ:

$$\nabla_\theta\, \mathbb{E}_{X\sim p_\theta}[g(X)] = \int g(x)\,\nabla_\theta p_\theta(x)\,dx = \mathbb{E}\big[g(X)\,\nabla_\theta \log p_\theta(X)\big]. \tag{34.75}$$

The gradient of an expected loss is the average loss, weighted by how much the parameter raises the log-probability of each sample. It is estimated by Monte Carlo: sample x_i, average $g(x_i)\nabla \log p_\theta(x_i)$. This is the REINFORCE estimator of policy-gradient methods (Williams 1992). If g also depends on θ, add $\mathbb{E}[\nabla_\theta g(X,\theta)]$.

Because the score has mean zero, any **baseline** b that does not depend on x can be subtracted from g without changing the mean: $\mathbb{E}[(g(X) - b)\nabla \log p_\theta(X)] = \mathbb{E}[g(X)\nabla \log p_\theta(X)]$. A good b lowers the variance.

34.7.3.2 The Pathwise Estimator

The second identity gives the sample its memory back. Often a sample can be produced by transforming noise that does not depend on θ: $x = T_\theta(\mathbf{z})$ with $\mathbf{z} \sim q$. The normal $\mathcal{N}(\mu,\sigma^2)$ is $\mu + \sigma z$ with $z \sim \mathcal{N}(0,1)$. Then the expectation is an integral over $\mathbf{z}$, whose density q does not move, and the swap gives the **pathwise identity**

$$\nabla_\theta\, \mathbb{E}_{X\sim p_\theta}[g(X)] = \nabla_\theta \int g(T_\theta(\mathbf{z}))\,q(\mathbf{z})\,d\mathbf{z} = \mathbb{E}_{\mathbf{z}\sim q}\Big[\nabla_\theta\, g(T_\theta(\mathbf{z}))\Big]. \tag{34.76}$$

The estimator differentiates the sample itself: automatic differentiation follows the path from θ through T_θ to g. This *reparameterization* is how variational autoencoders backpropagate through a Gaussian latent variable (Kingma and Welling 2014). It needs T differentiable

in θ, g differentiable along the path (a Lipschitz g suffices when X has a density), and a dominating bound for the swap. No boundary term appears, even when the support of X moves, because the integral is over $\mathbf{z}$ and q stays put.

The hypothesis on g cannot be dropped. Take $g(x) = \mathbf{1}(x > 0)$, which is 1 for positive x and 0 otherwise, and $X = \theta + z$. Then $\mathbb{E}[g(X)] = P(z > -\theta)$ has derivative $\varphi(\theta) = e^{-\theta^2/2}/\sqrt{2\pi}$, which is 0.13 at $\theta = 1.5$. The pathwise sample $\partial_\theta \mathbf{1}(\theta + z > 0)$ is 0 for every z except $z = -\theta$, an event of probability zero, so the pathwise estimator returns 0. The score sample $\mathbf{1}(x > 0)\, z$ has mean $\int_{-\theta}^{\infty} z\,\varphi(z)\,dz = \big[-\varphi(z)\big]_{-\theta}^{\infty} = \varphi(\theta)$, the right answer. The score estimator never differentiates through the sample, which is why it also covers Bernoulli and categorical choices, where no differentiable path from θ to the sample exists.

34.7.3.3 The Two Estimators Compared

We now have two unbiased estimators of the same derivative, and the smallest problem is simple enough to compute the variance of both by hand. For $\ell(\theta) = \mathbb{E}[X^2]$ with $X = \theta + z$, one sample contributes $2X$ to the pathwise estimator, with variance $\mathrm{Var}(2X) = 4$. The score of the normal location family is $\partial_\theta \log p_\theta(X) = X - \theta = z$, so one sample contributes $X^2 z$ to the score-function estimator, and $(X^2 - b)z$ with a baseline. The variance of $X^2 z$ needs the Gaussian moments $\mathbb{E}[z^2] = 1$, $\mathbb{E}[z^4] = 3$ and $\mathbb{E}[z^6] = 15$, all from Stein's identity $\mathbb{E}[g'(Z)] = \mathbb{E}[Z\,g(Z)]$ (Section 34.6.2.3). The odd powers of z drop out of the second moment,

$$\mathbb{E}\big[(\theta + z)^4 z^2\big] = \theta^4\,\mathbb{E}[z^2] + 6\theta^2\,\mathbb{E}[z^4] + \mathbb{E}[z^6] = \theta^4 + 18\theta^2 + 15,$$

and subtracting the squared mean $(2\theta)^2$ leaves the variance $\theta^4 + 14\theta^2 + 15$. The variance-minimizing constant baseline is $b^\star = \theta^2 + 3$, and it leaves $8\theta^2 + 6$ (exercise 2). At $\theta = 1.5$ the three standard deviations are 2 for the pathwise estimator, 7.18 for the score function, and 4.90 with the baseline. The baseline helps, but it cannot close the gap for this g, since $8\theta^2 + 6 > 4$ for every θ.

A simulation can test all of this at once. The pathwise sample is $\partial_\theta(\theta + z_i)^2 = 2x_i$, so no automatic differentiation is needed. If the closed forms are right, the three sample standard deviations should come out near 2, 7.18 and 4.90, and each mean should lie within a few standard errors $\mathrm{sd}/\sqrt{n}$ of 3.

```
theta, n = 1.5, 200_000
z = np.random.default_rng().standard_normal(n)     # noise, independent of theta
x = theta + z                                      # samples of N(theta, 1)
pathwise = 2 * x                                   # d/dtheta (theta + z_i)^2
score = x**2 * z                                   # g(x) times the score x - theta
baseline = (x**2 - (theta**2 + 3)) * z             # with the optimal baseline
exact_sd = {'pathwise': 2, 'score': math.sqrt(theta**4 + 14 * theta**2 + 15),
            'score+baseline': math.sqrt(8 * theta**2 + 6)}
print(f'true derivative 2*theta = {2 * theta}')
for name, s in zip(exact_sd, [pathwise, score, baseline]):
    print(f'{name:15s} mean={s.mean():.4f}  sd={s.std():.3f} '
          f'(exact {exact_sd[name]:.3f})  se={s.std() / math.sqrt(n):.4f}')
```

```
true derivative 2*theta = 3.0
pathwise        mean=3.0041  sd=2.000 (exact 2.000)  se=0.0045
```

```
score           mean=3.0068  sd=7.146 (exact 7.181)  se=0.0160
score+baseline  mean=2.9961  sd=4.858 (exact 4.899)  se=0.0109
```

In most runs the three means lie within about two standard errors of 3, and the sample standard deviations match the closed forms to within a few percent. With the same number of samples, the pathwise estimate is about 3.6 times more precise than the plain score estimate. Put differently, the score estimate needs about thirteen times as many samples for the same precision.

It would be wrong to conclude that the pathwise estimator always has the smaller variance. Take $g(x) = \sin(\omega x)$, for which the derivative of the mean is $\omega \cos(\omega\theta) e^{-\omega^2/2}$. The pathwise sample $\omega \cos(\omega X)$ has standard deviation close to $\omega/\sqrt{2}$, while the score sample $\sin(\omega X)\, z$ has standard deviation at most $\sqrt{\mathbb{E}[z^2]} = 1$, because $|\sin| \leq 1$. An oscillating integrand has large derivatives but bounded values. We compare the two at $\omega = 1$ and $\omega = 3$, reusing the samples.

```
for omega in [1, 3]:
    true = omega * math.cos(omega * theta) * math.exp(-omega**2 / 2)
    path, score = omega * np.cos(omega * x), np.sin(omega * x) * z
    print(f'omega={omega}  true={true:+.4f}  pathwise {path.mean():+.4f} '
          f'(sd {path.std():.2f})  score {score.mean():+.4f} (sd {score.std():.2f})')
```

```
omega=1  true=+0.0429  pathwise +0.0414 (sd 0.66)  score +0.0432 (sd 0.54)
omega=3  true=-0.0070  pathwise -0.0067 (sd 2.12)  score -0.0080 (sd 0.71)
```

At $\omega = 1$ the score estimator already has the smaller spread, about 0.5 against 0.7; at $\omega = 3$ it wins by a factor of three, about 0.7 against 2.1. Both means agree with the true values within a few standard errors.

Neither estimator wins everywhere, then. The pathwise estimator differentiates g and suffers where g' is large; the score-function estimator multiplies g by noise and suffers where g itself is large. In both, the standard deviation σ of a single sample decides how many samples are needed. It depends on the distribution the samples are drawn from, and choosing that distribution to make σ small, which is called importance sampling, is taken up in Section 36.8.

34.7.4 Summary

When a parameter moves the distribution of the samples, the gradient of an expected loss has to be rewritten as an expectation before samples can estimate it. Both ways of doing so move the derivative inside the integral, which needs a dominating bound and, when the limits move with the parameter, the boundary term of the Leibniz rule. The score-function identity differentiates the density. It works for any g, including discrete choices, but its samples scale with g and can have high variance. The pathwise identity differentiates the sample. It needs a differentiable g and often, though not always, has the lower variance. Either estimate has root-mean-square error $\sigma/\sqrt{n}$ (Section 36.5.1.7), so the standard deviation σ of one sample is the quantity to compare.

34.7.5 Exercises

1. **A moving support.** Let X be uniform on $[0,\theta]$ with $\theta>0$, so that $\ell(\theta)=\mathbb{E}[X^2]=\int_0^\theta \frac{x^2}{\theta}\,dx=\frac{\theta^2}{3}$ and $\ell'(\theta)=\frac{2\theta}{3}$.
   1. Differentiate ℓ with the Leibniz rule Equation 34.73. Which part of the result does the plain swap Equation 34.72 produce, and which part does it miss?
   2. On the support, $\log p_\theta(x)=-\log\theta$. Compute the expectation of the score-function estimator $X^2\,\partial_\theta \log p_\theta(X)$ of Equation 34.75. Then let g be continuous on $[0,\infty)$. Show that the mean of $g(X)\,\partial_\theta \log p_\theta(X)$ equals the plain swap Equation 34.72 applied to $\mathbb{E}[g(X)]=\int_0^\theta g(x)\,p_\theta(x)\,dx$. What is its bias as an estimate of $\frac{d}{d\theta}\mathbb{E}[g(X)]$, in terms of $g(\theta)$ and θ?
   3. Write $X=\theta U$ with U uniform on $[0,1]$ and compute the expectation of the pathwise estimator of Equation 34.76. Why does it need no boundary term, although the support of X moves with θ?
2. code **Estimator variance for a quartic.** ▣ Let $X\sim\mathcal{N}(\theta,1)$, write $X=\theta+z$, and let $s(X)=\partial_\theta \log p_\theta(X)=z$ be the score.
   1. Take $g(X)=X^4$, so that $\frac{d}{d\theta}\mathbb{E}[X^4]=4\theta^3+12\theta$. Derive the single-sample pathwise estimator from Equation 34.76 and the single-sample score-function estimator from Equation 34.75. Use the identity of exercise 7 (b) to show that both have mean $4\theta^3+12\theta$.
   2. For a constant baseline b, show that the variance of $(g(X)-b)\,s(X)$ is minimized by $b^\star=\mathbb{E}[g(X)s(X)^2]/\mathbb{E}[s(X)^2]$. For $g(X)=X^2$ show that $b^\star=\theta^2+3$, and use $\mathbb{E}[z^2]=1$, $\mathbb{E}[z^4]=3$, $\mathbb{E}[z^6]=15$ to derive the three variances 4, $\theta^4+14\theta^2+15$ and $8\theta^2+6$ that the text quotes.
   3. Evaluate $b^\star$ for $g(X)=X^4$. With 10^6 samples at $\theta\in\{0,1,2\}$, estimate the standard deviations of the pathwise estimator, the score-function estimator, and the score-function estimator with the baseline. For which θ is the baseline's gain largest, and does it ever close the gap to the pathwise estimator? Explain the ordering from the sizes of g' and of g.
   4. Take $g(x)=\mathbf{1}(x>0)$ at $\theta=1.5$. The text shows that the pathwise estimator is 0 for almost every sample, while the score-function estimator $\mathbf{1}(x>0)\,z$ has the correct mean $\varphi(\theta)=0.1295$. Verify both by simulation.
3. [extended] **Gradients through a coin flip.** ▣▣ When x is a discrete choice there is no differentiable path from θ to the sample, so the pathwise estimator does not exist. Continuous relaxations such as Gumbel-softmax (Jang et al. 2017; Maddison et al. 2017) create a differentiable surrogate, and straight-through estimators (Bengio, Léonard, et al. 2013) use the hard sample in the forward pass and the surrogate in the backward pass; both are biased for the original objective. This problem measures that bias. Tucker et al. (2017) illustrate it on a toy problem. A single Bernoulli variable $b\in\{0,1\}$ has $P(b=1)=\sigma(\theta)$ for a logit θ, and the objective is $\ell(\theta)=\mathbb{E}[(b-t)^2]$ with target $t=0.45$. Let U be uniform on $(0,1)$, let $\zeta=\log U-\log(1-U)$, and, for a temperature $\tau>0$, let $z=\sigma((\theta+\zeta)/\tau)$, the two-class case of the Gumbel-softmax relaxation.
   1. Compute $\ell(\theta)$ and $\ell'(\theta)$ in closed form. Which probability $\sigma(\theta)$ minimizes ℓ, and does a finite θ attain it?
   2. Show that $b=\mathbf{1}(z>\frac{1}{2})$ satisfies $P(b=1)=\sigma(\theta)$ for every τ.
   3. Implement three single-sample estimators of $\ell'(\theta)$: REINFORCE Equation 34.75 with the baseline $\ell(\theta)$; the relaxed estimator $\frac{d}{d\theta}(z-t)^2$; and the straight-through

estimator $2(b-t)\frac{dz}{d\theta}$, which uses the hard sample in the loss and the relaxed sample in the derivative. Tabulate the mean and the standard deviation of each estimator against $\ell'(\theta)$ for $\theta \in \{-2, 0, 2\}$ and $\tau \in \{1, 0.5, 0.1\}$. How does the bias of each estimator depend on θ and τ?

4. Run stochastic gradient descent on θ from $\theta = 0$ with step size 1 for 5000 steps with REINFORCE and with the relaxed and straight-through estimators at each τ, replacing the REINFORCE baseline by an exponential moving average of past losses. Plot $\ell(\theta)$ along each run. Compute the relaxed objective $R_\tau(\theta) = \mathbb{E}[(z-t)^2]$ by quadrature over U, and compare its minimizer with the values of θ at which the relaxed runs settle. How does lowering τ trade the bias of the relaxed estimator against its variance?

34.8 Further Reading

The chapter proved its results under hypotheses that practice does not always meet: a ReLU network has kinks, a tape may not fit in memory, and an integral in a thousand dimensions cannot be put on a grid. The topics below say what is known when a hypothesis fails. Two bear on everyday training and are worth reading first: conservative fields, which say in what sense backpropagation through kinks is still correct, and recursive checkpointing. The rest is for lookup. Resources for what the chapter does teach are listed in its introduction (Chapter 34). This section can be skipped without loss.

34.8.1 Derivatives at Kinks

34.8.1.1 Conservative Fields

Section 34.1 shows that backpropagation with the convention $\mathrm{ReLU}'(0) = 0$ returns slope 0 for $\mathrm{ReLU}(x) - \mathrm{ReLU}(-x) = x$ at $x = 0$, a number that is neither the derivative nor a subgradient of the identity. The subdifferential cannot describe this, because its sum and chain rules hold only as inclusions once the pieces are not all convex. Bolte and Pauwels (2021) replace it by a *conservative field*, a set-valued derivative that obeys the chain rule by construction and agrees with the gradient outside a set of measure zero. Backpropagation with any fixed choice at each kink returns an element of such a field. That the exceptional set is small is Rademacher's theorem (Rademacher 1919): a Lipschitz function on $\mathbb{R}^n$ is differentiable almost everywhere, so the kinks of a ReLU network's loss form a set of measure zero.

The same paper answers whether stochastic gradient descent driven by these values converges. For losses built from the usual layers, with bounded iterates and the usual conditions on step sizes and noise, the iterates approach the set of points where the conservative field contains zero. The result holds in exact arithmetic. In floating point the kinks are hit. Section 34.1 reports how often, as measured by Bertoin et al. (2021) at three precisions, and how little the choice of $\mathrm{ReLU}'(0)$ then matters to training.

34.8.1.2 Subgradient Methods

At the minimum of $|x|$ the valid subgradient $\frac{1}{2}$ sends a step uphill, as Section 34.1 shows, so the descent lemma has no analogue for nonsmooth functions. The subgradient method converges anyway, because its analysis tracks the distance to a minimizer instead of the function value. Step sizes with $\sum_t \eta_t = \infty$ and $\sum_t \eta_t^2 < \infty$ make the best value found

converge to the minimum. The rate belongs to a different choice: for a convex function with subgradients bounded by G, started at distance R from a minimizer, the constant step $R/(G\sqrt{T})$ brings the best of T iterates within $RG/\sqrt{T}$ of the minimum. When the dimension exceeds T, no method that uses only subgradients can do better by more than a constant factor (Nesterov 2018). Rockafellar (1970) develops the calculus of subdifferentials, including when the sum rule holds with equality.

34.8.2 Derivatives of Vector and Matrix Maps

34.8.2.1 The Mean Value Theorem for Vector-Valued Maps

The one-variable mean value theorem of Section 34.1 writes $f(b) - f(a) = f'(\xi)(b-a)$ for one intermediate point ξ. For a curve $\gamma : \mathbb{R} \to \mathbb{R}^n$ no such point need exist: the circle $\gamma(t) = (\cos t, \sin t)$ returns to its start at $t = 2\pi$, yet $\gamma'(\xi) \neq \mathbf{0}$ for every ξ. Only the *mean value inequality* $\|\gamma(b) - \gamma(a)\| \le \sup_{\xi} \|\gamma'(\xi)\|\,|b-a|$ survives (Theorem 5.19 of Rudin (1976)). The Taylor bounds of Section 34.2 need no vector version of the theorem, because they restrict a scalar function to a line, where the one-variable theorem applies.

34.8.2.2 Jacobians of Matrix-Valued Maps

Section 34.4 differentiates scalar losses, so its gradients have the shape of the variable. The derivative of a matrix-valued function of a matrix has four indices. Flattening both matrices with vec, which stacks the columns, turns it into an ordinary Jacobian: a differential $d\mathbf{F} = \mathbf{A}\, d\mathbf{X}\, \mathbf{B}$ becomes $d\,\mathrm{vec}\,\mathbf{F} = (\mathbf{B}^{\top} \otimes \mathbf{A})\, d\,\mathrm{vec}\,\mathbf{X}$, and the chain rule multiplies Kronecker products. Mike B. Giles (2008) collects the forward and reverse rules for products, inverses, determinants, and matrix decompositions.

34.8.3 Automatic Differentiation

34.8.3.1 Optimal Jacobian Accumulation

Section 34.4 shows that the order in which a chain of Jacobians is multiplied changes the cost by a factor of about the layer width. For a chain, the cheapest bracketing is the matrix-chain dynamic program, which runs in $O(K^3)$ time for K factors. A computation graph that is not a chain allows many more orders: each intermediate node can be eliminated by multiplying the partial derivatives on its incoming and outgoing edges, in any sequence. Finding the sequence with the fewest multiplications is NP-complete (Naumann 2008). Frameworks do not attempt it: they offer forward mode, reverse mode, and their compositions, and leave the choice to the user. Griewank and Walther (2008) treat vertex, edge, and face elimination as the general operations and discuss the heuristics used in compilers of derivative code.

34.8.3.2 Recursive Checkpointing

The one-level schedule of Section 34.5 stores $T/s + s$ values, at best $2\sqrt{T}$, for one extra forward pass (Chen et al. 2016). Unrolled solvers and very long sequences need less memory still. Checkpointing recursively inside each recomputed segment reaches $O(\log T)$ memory for $O(T \log T)$ recomputation (Griewank 1992), and Griewank's binomial schedule is optimal for a given number of stored values when every step costs the same (Griewank and Walther 2008). `torch.utils.checkpoint` and `jax.checkpoint` apply one level per marked block; in JAX, nesting the marks gives a recursive schedule by hand.

34.8.3.3 Taylor Mode

Nesting forward mode k times to get the k-th derivative along a direction carries 2^k components per value. Taylor mode carries the truncated series $a_0 + a_1\varepsilon + \cdots + a_k\varepsilon^k$ with $\varepsilon^{k+1} = 0$ instead of a dual number. Each primitive propagates the $k+1$ coefficients in one pass: a product convolves the two coefficient sequences, at $O(k^2)$ arithmetic, and exp, log, and sin follow from recurrences of the same cost (Griewank and Walther 2008). The exercise on second derivatives in one forward pass in Section 34.5 is the case $k = 2$; `jax.experimental.jet` implements the general case.

34.8.3.4 Forward Gradients

Reverse mode needs a tape, and forward mode needs one pass per input direction. Baydin et al. (2022) take one forward pass with a random tangent $\mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and use $(\nabla \ell^\top \mathbf{v})\,\mathbf{v}$ as the gradient. The estimate is unbiased, because $\mathbb{E}[\mathbf{v}\mathbf{v}^\top] = \mathbf{I}$, and its mean squared error is $(n+1)\|\nabla \ell\|^2$ for n parameters (an exercise of Section 34.5 derives both facts). The tape is gone, but the error is about $\sqrt{n}$ times the length of the gradient, which is acceptable for small models and prohibitive for large ones.

34.8.4 Integration

34.8.4.1 The Lebesgue Integral

The indicator of the rationals has tagged sums that never settle, so it has no Riemann integral (Section 34.6), and a pointwise limit of Riemann-integrable functions need not be Riemann integrable. Lebesgue slices the range instead of the domain: it measures the set where f exceeds each level and adds up these measures. Sets of measure zero, such as the rationals or the ray $\theta = 0$ of polar coordinates, contribute nothing, so the indicator integrates to 0. Section 34.6 and Section 34.7 quote three results of this theory without proof: Fubini–Tonelli, the change of variables in n dimensions, and differentiation under an integral over an infinite range. All three rest on the monotone and dominated convergence theorems, which state when a limit passes through an integral: if $f_k \to f$ pointwise and $|f_k| \le g$ for one integrable g, then $\int f_k \to \int f$ (Folland 1999).

34.8.4.2 Quasi-Monte Carlo

Section 36.5.1.7 compares a second-order grid, with error $N^{-2/d}$, against Monte Carlo, with error $N^{-1/2}$ in every dimension. Random points leave gaps and clusters by chance. Low-discrepancy point sets (Sobol and Halton sequences, lattice rules) fill the cube more evenly by construction, and for integrands of bounded variation (in the sense of Hardy and Krause) their error is $O(N^{-1}(\log N)^d)$. As a guarantee this is weak: ignoring constants, the bound falls below $N^{-1/2}$ only beyond several thousand points for $d = 2$ and 10^{15} for $d = 5$. In practice quasi-Monte Carlo often wins in high dimension, which is usually explained by integrands that depend mostly on a few coordinates at a time. A deterministic point set has no error estimate; randomized quasi-Monte Carlo shifts or scrambles the points so that independent replicates give one. Owen (2013) covers both in Chapters 15–17.

34.8.4.3 Gradient Estimators beyond Score and Pathwise

Section 34.7 derives the score-function and pathwise estimators and reduces the variance of the first with a constant baseline. Other control variates use the model itself: REBAR

(Tucker et al. 2017) subtracts the pathwise gradient of a continuous relaxation from the score-function estimator and adds back its expectation, which keeps the estimate unbiased for discrete variables. The relaxations themselves, Gumbel-softmax (Jang et al. 2017; Maddison et al. 2017) and straight-through estimators (Bengio, Léonard, et al. 2013), have low variance but are biased for the original objective. A third family, measure-valued derivatives, writes the derivative of a density as a scaled difference of two densities and samples from both. Mohamed et al. (2020) compare all three families in one framework and state the conditions each needs.

Chapter 35

Optimization

A network is trained by one loop: compute the gradient of the loss, step against it, repeat. Chapter 9 showed on real networks what momentum, adaptive methods and schedules add to that loop, and left three questions to this chapter. Why does the learning rate have a ceiling? Why can a step below it still make almost no progress? Why do noisy gradients keep a run from settling?

The theory ties each answer to a property of the function. The steepest curvature sets the largest safe step, the ratio of the steepest to the flattest curvature sets the number of steps, and convexity decides whether the point where a run stops is the best one. The bowl $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$, the running example of the chapter, is a thousand times steeper in one direction than in the other. A step that is safe in the steep direction barely moves along the flat one, so gradient descent needs thousands of steps.

The two chapters divide the work: Chapter 9 measures and uses, this chapter proves. Small results, such as the ceiling $2/L$ or the Adam update, appear in both. Long derivations live only here, and the main text can be read without them.

The first three sections assume exact gradients. Section 35.1 proves the ceiling and the step count, and shows that momentum replaces the curvature ratio by its square root; Section 35.3 says when the point reached is the best one. These two are the ones to read. Between them, Section 35.2 treats Newton's method and its relatives, which model the curvature where gradient descent only respects it. Deep learning uses them less, and the section can wait for a second reading. The next two sections add noise. Section 35.4 shows that a fixed step stalls at a noise floor proportional to the step, the third result to remember, and Section 35.5 sets Adam against a tuned heavy ball, to see what its per-coordinate steps gain and where they fail. The remaining sections are best read on demand. Constraints and multipliers (Section 35.6, Section 35.7) are what Section 2.7 uses to read weight decay as a norm budget, and Section 35.8 derives why the low-precision arithmetic of Section 5.5 fails when it does.

We assume gradients and Hessians (Section 34.2) and eigenvalues (Section 33.3). The proofs assume smooth or convex functions. Deep-network losses are neither, so for them the results are models, and each section says where its model stops. Interior-point and operator-splitting methods are left out.

Notation

The chapter uses the symbols of Table 35.1. Each section restates the ones it needs, and symbols used in only one section are defined there.

Table 35.1: Symbols of the optimization chapter. A star $\star$ marks an optimum and an asterisk $*$ a convex conjugate. Matrices are bold.

Symbol	Meaning
f, $\ell(\mathbf{w})$	objective; training loss as a function of the weights
$\mathbf{x}^\star$, $f^\star$	a minimizer and the minimum value
∇f, $\nabla^2 f$, $\mathbf{g}_k$	gradient, Hessian; the gradient (or a stochastic gradient) used at step k
L, μ	smoothness and strong-convexity constants, $\mu\mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$
$\kappa = L/\mu$; $\kappa(\mathbf{A})$	condition number of the problem; of a matrix, $\sigma_{\max}(\mathbf{A})/\sigma_{\min}(\mathbf{A})$
λ_i, $\lambda_{\max}(\mathbf{A})$	eigenvalues of a Hessian or of a quadratic's matrix $\mathbf{A}$
η, β	step size (learning rate); momentum coefficient
k, t	iteration index; optimizer step in the adaptive methods
ϵ	target accuracy; in Adam, the constant that keeps a division finite
n, d, $\mathcal{B}$, b	number of examples; number of variables; a minibatch and its size
Σ	covariance of a per-example gradient
$\mathbf{m}_t$, $\mathbf{v}_t$; β_1, β_2	Adam's first- and second-moment estimates; their decay rates
$\mathbf{P}$	preconditioner: the step is $-\eta\,\mathbf{P}^{-1}\mathbf{g}$
g_i, h_j	inequality constraints $g_i(\mathbf{x}) \le 0$; equality constraints $h_j(\mathbf{x}) = 0$
λ, ν	multipliers of inequality and equality constraints; λ also weights the decay term $\frac{\lambda}{2}\lVert\mathbf{w}\rVert^2$
$\mathcal{L}$, q	Lagrangian; dual function
$p^\star$, $d^\star$	optimal values of the primal and the dual problem
f^*	convex conjugate of f
Π_C, prox_ψ	Euclidean projection onto a set C; proximal operator of ψ
u, $\varepsilon_{\text{mach}}$	unit roundoff and machine epsilon of a floating-point format, $u = \varepsilon_{\text{mach}}/2$

Resources and Further Reading

The resources below accompany the sections of this chapter; each names the section it goes with. Topics that the chapter does not teach, such as optimal methods on quadratics, mirror descent, interior-point methods, operator splitting, and probabilistic rounding-error analysis, are collected in Section 35.9.

Books

- Convex Optimization — Boyd & Vandenberghe — convex sets and functions, duality, and interior-point methods, with a free PDF; the reference for Section 35.3, Section 35.6, and Section 35.7.
- Numerical Optimization — Nocedal & Wright (2nd ed., Springer) — line search, trust regions, quasi-Newton and conjugate-gradient methods, and constrained optimization, in more depth than Section 35.1, Section 35.2, and Section 35.6.
- Lectures on Convex Optimization — Nesterov (2nd ed., Springer) — complexity bounds and accelerated first-order methods, with the general proofs that Section 35.1 and Section 35.3 cite.
- Optimization for Data Analysis — Wright & Recht (Cambridge, 2022) — gradient, accelerated, stochastic, and coordinate methods for data analysis, a companion to Section 35.1 through Section 35.5.
- Accuracy and Stability of Numerical Algorithms — Higham (2nd ed., SIAM) — rounding-error analysis, summation, and the conditioning of linear systems and least squares, the material of Section 35.8.
- Mathematics for Machine Learning — Deisenroth, Faisal & Ong — a free, self-contained chapter on gradient descent, constrained optimization, and convexity, at the level of Section 35.1 and Section 35.6.

Courses and video lectures

- Stanford EE364a: Convex Optimization I — Boyd — slides, homework, and software for the Boyd & Vandenberghe text, with 2023 lecture videos; it follows Section 35.3 and Section 35.7.
- CMU 10-725: Convex Optimization — Tibshirani — ML-oriented slides and scribed notes on subgradients, proximal and stochastic methods, duality, and ADMM, with Fall 2016 lecture videos; it accompanies Section 35.4, Section 35.6, and Section 35.7.

Tutorials, notes, and surveys

- Why Momentum Really Works — Goh (Distill) — interactive figures of momentum on convex quadratics, the analysis of Section 35.1.3.
- Optimization Methods for Large-Scale Machine Learning — Bottou, Curtis & Nocedal — the theory of stochastic-gradient methods for large-scale learning, the survey behind Section 35.4.
- An overview of gradient descent optimization algorithms — Ruder — momentum, Nesterov acceleration, AdaGrad, RMSProp, and Adam side by side, the methods of Section 35.5.
- What Every Computer Scientist Should Know About Floating-Point Arithmetic — Goldberg — the IEEE format, rounding, and cancellation, the opening material of Section 35.8.
- Optimization (Chapter 5), Patterns, Predictions, and Actions — Hardt & Recht — a concise account of optimization in supervised learning, a short alternative route through Section 35.1 and Section 35.4.

35.1 Gradient Descent and Curvature

A learning rate is usually found by trial: we try a value, watch the loss, and lower the value if the loss blows up. Each training step moves against the gradient of the loss, so the gradient fixes the direction of the step and the learning rate sets only its distance. Yet a poor choice of distance makes training either slow or unstable. How far can a step along the gradient be trusted, and how many steps does that limit force on us?

A function of two variables already shows the difficulty. Take the bowl $f(x, y) = \frac{1}{2}(x^2 + 10\,y^2)$, which curves ten times faster along y than along x, and start at the point $(2, 1)$, where $f = 7$ and the gradient is $\nabla f = (2, 10)$. A step of size η against the gradient moves to $(2 - 2\eta,\ 1 - 10\eta)$. With $\eta = 0.01$ the value drops only to 6.01. With $\eta = 0.1$ it drops to 1.62. With $\eta = 0.25$ it *rises* to 12.4. The direction was downhill every time; only the distance differed.

How far a step can go depends on curvature. A step short enough for the steep direction y makes slow progress along the flat direction x. Even the best fixed step needs 69 iterations to shrink the error by a factor of 10^6, that is, to gain six digits. Stretch the ratio to 1000, the chapter's running example $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$, and it needs $6{,}908$. Momentum brings that down to 315. We work by hand on the opening bowl, whose Hessian $\mathrm{diag}(1, 10)$ keeps every step checkable, and quote counts for the stiffer running example.

We first bound the safe step through smoothness, which turns the local linear model of Section 34.2 into a guarantee. We then solve gradient descent exactly on quadratics, where the ratio of curvatures appears as the condition number, and show how momentum changes the count. Three results carry the argument: the descent lemma Equation 35.5, the optimal step Equation 35.9, and the $\sqrt{\kappa}$ law of momentum Equation 35.14. Throughout, the methods of this section only respect the curvature. What a step can do once it models the curvature, as Newton's method does, is the subject of Section 35.2. Gradients are exact here, and the noisy gradients of training wait for Section 35.4. The code is plain NumPy, so each update can be read against its equation.

35.1.1 Descent Directions and Smoothness

A gradient step answers two questions, which direction and how far. The first has a clean answer from the first-order Taylor expansion. The second needs an assumption about how fast the gradient changes. That assumption turns the first-order approximation into a guarantee for a step of finite length.

35.1.1.1 The Steepest Direction

The opening stepped along $-\nabla f$ without asking why. To see that it is the right choice, expand f to first order. For a differentiable $f : \mathbb{R}^d \to \mathbb{R}$, a small step of size $\eta > 0$ along a direction $\mathbf{d}$ changes the value by

$$f(\mathbf{x} + \eta\,\mathbf{d}) = f(\mathbf{x}) + \eta\,\nabla f(\mathbf{x})^\top \mathbf{d} + o(\eta).$$

To first order the change is η times the directional derivative $\nabla f(\mathbf{x})^\top \mathbf{d}$. Call $\mathbf{d}$ a **descent direction** at $\mathbf{x}$ if $\nabla f(\mathbf{x})^\top \mathbf{d} < 0$.

Proposition (descent directions, and the steepest one). *Let f be differentiable at $\mathbf{x}$ with $\nabla f(\mathbf{x}) \neq \mathbf{0}$.*

1. *If $\nabla f(\mathbf{x})^{\top}\mathbf{d} < 0$, then $f(\mathbf{x}+\eta\,\mathbf{d}) < f(\mathbf{x})$ for every sufficiently small $\eta > 0$.*
2. *Among unit vectors, the directional derivative is minimized uniquely by the normalized negative gradient:*

$$\min_{\|\mathbf{d}\|=1} \nabla f(\mathbf{x})^{\top}\mathbf{d} = -\|\nabla f(\mathbf{x})\|, \qquad \text{attained only at } \mathbf{d} = -\frac{\nabla f(\mathbf{x})}{\|\nabla f(\mathbf{x})\|}. \tag{35.1}$$

Proof. For the first claim, the function $\varphi(\eta) = f(\mathbf{x}+\eta\,\mathbf{d})$ has $\varphi'(0) = \nabla f(\mathbf{x})^{\top}\mathbf{d} < 0$ by the chain rule, so the difference quotient $(\varphi(\eta)-\varphi(0))/\eta$ is negative for all small $\eta > 0$, that is, $\varphi(\eta) < \varphi(0)$. For the second, the Cauchy–Schwarz inequality gives, for every unit vector $\mathbf{d}$, $\nabla f(\mathbf{x})^{\top}\mathbf{d} \geq -\|\nabla f(\mathbf{x})\|\,\|\mathbf{d}\| = -\|\nabla f(\mathbf{x})\|$, with equality only when $\mathbf{d}$ is a negative multiple of $\nabla f(\mathbf{x})$. The only such unit vector is $-\nabla f(\mathbf{x})/\|\nabla f(\mathbf{x})\|$, and it attains the bound. ■

On the opening example the steepest unit direction at $(2,1)$ is $-(2,10)/\sqrt{104} = (-0.196,-0.981)$, with slope $-\|\nabla f\| = -10.198$. The axis direction $(-1,0)$ is also downhill, but its slope is only -2. Descent directions fill the open half-space of vectors that make an acute angle with $-\nabla f$; the gradient picks the steepest of them. All of this concerns slopes, the limit of short steps. For a step of finite length, $\eta = 0.01$, we set the slope beside the actual change along four unit directions.

```
def f(v):
    return 0.5 * (v[0]**2 + 10 * v[1]**2)

def grad(v):
    return np.array([v[0], 10 * v[1]])

x = np.array([2.0, 1.0])
g = grad(x)
steepest = -g / np.linalg.norm(g)
candidates = [('steepest -grad/|grad|', steepest),
              ('diagonal -(1,1)/sqrt(2)', -np.ones(2) / np.sqrt(2)),
              ('axis     (-1,0)', np.array([-1.0, 0.0])),
              ('uphill   +(1,1)/sqrt(2)', np.ones(2) / np.sqrt(2))]
eta = 0.01
for name, d in candidates:
    print(f'{name}:  slope = {g @ d:+8.4f},  '
          f'f(x + eta d) - f(x) = {f(x + eta * d) - f(x):+.6f}')
```

```
steepest -grad/|grad|:  slope = -10.1980,  f(x + eta d) - f(x) = -0.101498
diagonal -(1,1)/sqrt(2):  slope =  -8.4853,  f(x + eta d) - f(x) = -0.084578
axis     (-1,0):  slope =  -2.0000,  f(x + eta d) - f(x) = -0.019950
uphill   +(1,1)/sqrt(2):  slope =  +8.4853,  f(x + eta d) - f(x) = +0.085128
```

At this step length the actual changes are η times the slopes to within one percent, so here the slopes are a fair guide. Differentiability promises no more than that: a decrease for small enough steps. How small is the question of the next subsection.

35.1.1.2 Smoothness and the Descent Lemma

Gradient descent repeats the steepest step with a fixed **step size** (learning rate) $\eta > 0$:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k). \tag{35.2}$$

Throughout this section k counts iterations, and we abbreviate $\mathbf{g}_k = \nabla f(\mathbf{x}_k)$. The first-order model justified the direction. It says nothing about how far to trust it, because the gradient at $\mathbf{x}_k$ stops describing f once the step moves far enough for the gradient to change. The assumption we need bounds that change.

Definition (L-smoothness). A differentiable f is **L-smooth** if its gradient changes by at most L per unit of distance:

$$\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \;\le\; L\,\|\mathbf{x} - \mathbf{y}\| \qquad \text{for all } \mathbf{x}, \mathbf{y}. \tag{35.3}$$

For twice continuously differentiable f this is the same as every Hessian eigenvalue lying in $[-L, L]$. Bounded eigenvalues give Equation 35.3 by integrating the Hessian along the segment from $\mathbf{y}$ to $\mathbf{x}$, and the converse is a difference-quotient limit (exercise 2a). So L bounds the curvature in every direction. The opening example has the constant Hessian $\operatorname{diag}(1, 10)$, so $L = 10$. A function whose curvature grows without limit has no such constant: $\frac{1}{4}x^4$ has $f''(x) = 3x^2$, so no global L exists. The backtracking line search below handles that case.

Smoothness turns the first-order model into a one-sided guarantee, the quadratic ceiling

$$f(\mathbf{y}) \;\le\; f(\mathbf{x}) + \nabla f(\mathbf{x})^{\top}(\mathbf{y} - \mathbf{x}) + \tfrac{L}{2}\,\|\mathbf{y} - \mathbf{x}\|^2. \tag{35.4}$$

The graph of f never rises above its tangent plane by more than $L/2$ times the squared distance. Before proving it, insert a gradient step $\mathbf{y} = \mathbf{x} - \eta\,\mathbf{g}$ and compare the two terms. The first-order decrease $\eta\,\|\mathbf{g}\|^2$ grows linearly in η, while the curvature error $\frac{L}{2}\eta^2\|\mathbf{g}\|^2$ grows quadratically. For short steps the decrease dominates, and at $\eta = 2/L$ the two terms are equal.

Proposition (descent lemma). *If f is L-smooth, then the gradient step Equation 35.2 satisfies*

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\left(1 - \tfrac{L\eta}{2}\right)\|\mathbf{g}_k\|^2. \tag{35.5}$$

In particular f strictly decreases whenever $0 < \eta < 2/L$ and $\mathbf{g}_k \neq \mathbf{0}$, and the guaranteed decrease is largest at $\eta = 1/L$, where it reads $f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k) - \frac{1}{2L}\|\mathbf{g}_k\|^2$.

Proof. First the ceiling Equation 35.4. Restrict f to the segment, $\psi(t) = f(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))$ for $t \in [0, 1]$, so that $\psi'(t) = \nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))^{\top}(\mathbf{y} - \mathbf{x})$. By the fundamental theorem of calculus and then Cauchy–Schwarz with the Lipschitz bound Equation 35.3,

$$\begin{aligned} f(\mathbf{y}) - f(\mathbf{x}) - \nabla f(\mathbf{x})^{\top}(\mathbf{y} - \mathbf{x}) &= \int_0^1 \big(\nabla f(\mathbf{x} + t(\mathbf{y} - \mathbf{x})) - \nabla f(\mathbf{x})\big)^{\top}(\mathbf{y} - \mathbf{x})\,dt \\ &\le \int_0^1 L\,t\,\|\mathbf{y} - \mathbf{x}\|^2\,dt = \tfrac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2. \end{aligned}$$

Now insert the gradient step $\mathbf{y} = \mathbf{x}_k - \eta\,\mathbf{g}_k$:

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\,\|\mathbf{g}_k\|^2 + \tfrac{L}{2}\,\eta^2\,\|\mathbf{g}_k\|^2,$$

and the bracket $\eta(1 - L\eta/2)$ is positive exactly for $\eta < 2/L$ and maximized at $\eta = 1/L$. ■

The proof compares gradients only between $\mathbf{x}_k$ and points of the segment to $\mathbf{x}_{k+1}$, so it is enough that ∇f is L-Lipschitz on that segment. On the opening example $\|\mathbf{g}\|^2 = 104$ and $L = 10$. At $\eta = 1/L = 0.1$ the lemma promises $f \le 7 - 104/20 = 1.8$. At $\eta = 0.01$ it promises $f \le 7 - 104 \cdot 0.01 \cdot 0.95 = 6.012$. At $\eta = 0.25 > 2/L$ the bracket is negative and the bound exceeds the starting value, so the lemma promises nothing. How much do these promises give away against the values the three steps actually reach?

```
x, L = np.array([2.0, 1.0]), 10.0
g = grad(x)
for eta in (0.01, 0.1, 0.25):
    bound = f(x) - eta * (1 - L * eta / 2) * (g @ g)
    print(f'eta = {eta:4.2f}:  bound {bound:7.4f},  actual {f(x - eta * g):7.4f}')
```

```
eta = 0.01:  bound  6.0120,  actual  6.0102
eta = 0.10:  bound  1.8000,  actual  1.6200
eta = 0.25:  bound 13.5000,  actual 12.3750
```

Very little, for the two stable steps: 6.0102 under 6.012, and 1.62 under 1.8. The third bound, 13.5, is above the start $f = 7$, and the step indeed raised f, to 12.375. Figure 35.1 draws the two functions of η behind these numbers. Along the step, $\mathbf{g}^\top \mathrm{diag}(1, 10)\,\mathbf{g} = 1004$, so the objective is the parabola $7 - 104\,\eta + 502\,\eta^2$, and the ceiling is $7 - 104\,\eta + 520\,\eta^2$. The two are close because this gradient points almost along the stiff axis, where the curvature is nearly L.

The step $\eta = 1/L$ minimizes the ceiling, and f lies below the ceiling: that is all the guarantee says. Section 34.1 has the same argument in one variable.

35.1.1.3 Stationary Points Without Convexity

The descent lemma alone, with no convexity, already says something about a long run: the gradient cannot stay large.

Proposition (gradient descent finds approximate stationary points). *Let f be L-smooth and bounded below by f_{low}, and run Equation 35.2 with $\eta = 1/L$. Then for every $K \ge 1$,*

$$\min_{0 \le k < K} \|\nabla f(\mathbf{x}_k)\|^2 \;\le\; \frac{2L\,(f(\mathbf{x}_0) - f_{\text{low}})}{K}. \tag{35.6}$$

Proof. At $\eta = 1/L$ the descent lemma reads $\|\mathbf{g}_k\|^2 \le 2L\,(f(\mathbf{x}_k) - f(\mathbf{x}_{k+1}))$. Sum over $k = 0, \dots, K-1$; the right side telescopes to $2L\,(f(\mathbf{x}_0) - f(\mathbf{x}_K)) \le 2L\,(f(\mathbf{x}_0) - f_{\text{low}})$. The smallest of K numbers is at most their average. ■

Some iterate among the first K has squared gradient norm $O(1/K)$, and nothing beyond a lower bound and a smoothness constant was assumed. The bound does need a global L. ReLU networks are not smooth, so for them it describes an idealized model rather than a guarantee.

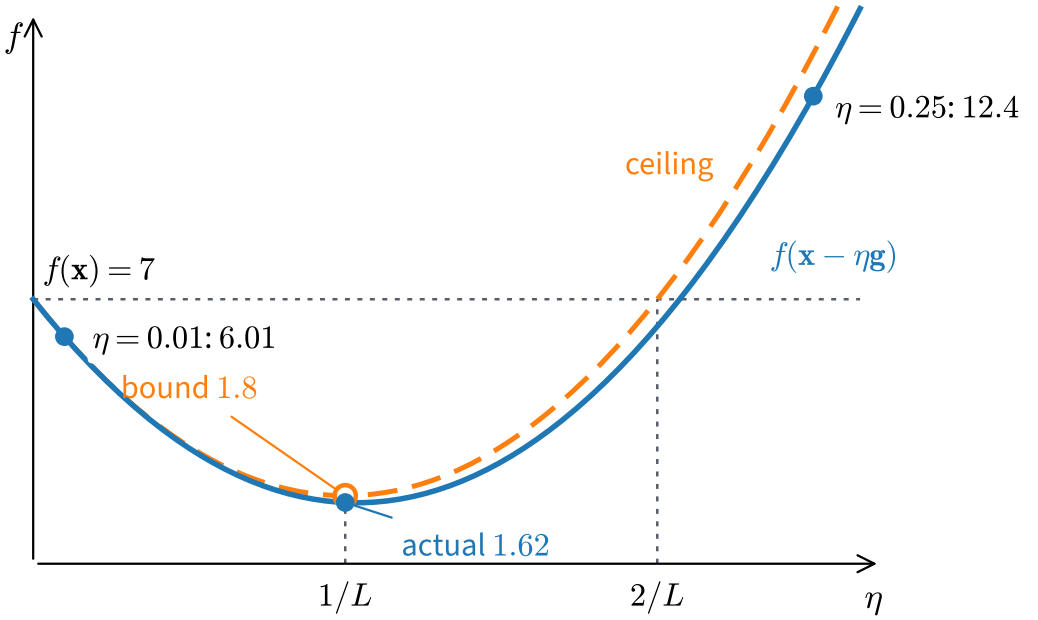


Figure 35.1: The descent lemma on the opening example. Along $-\mathbf{g}$ from $(2, 1)$ the objective is the lower parabola; the ceiling $f - \eta\|\mathbf{g}\|^2 + \frac{L}{2}\eta^2\|\mathbf{g}\|^2$ is the upper one, tangent at $\eta = 0$. Minimizing the ceiling ($\eta = 1/L = 0.1$) guarantees $f \le 1.8$; the step achieves 1.62. Past $2/L = 0.2$ the ceiling is above the starting value and promises nothing; at $\eta = 0.25$ the objective has risen to 12.4.

A small gradient does not mean a minimum. Take $f(x, y) = \frac{1}{2}x^2 + \cos y$, which is bounded below by -1 and 1-smooth (its Hessian $\mathrm{diag}(1, -\cos y)$ has eigenvalues in $[-1, 1]$). Gradient descent with $\eta = 1$ from $(1, 0)$ sets x to 0 in one step and never moves y, because $\partial f/\partial y = -\sin y$ vanishes at $y = 0$. The run stops at $(0, 0)$, where $f = 1$. This point is a saddle: f rises along x and falls along y. The minimum value -1 sits at $(0, \pi)$. The bound holds, since the gradient at the saddle is zero, but it says nothing about which stationary point is reached. Exercise 3 measures how long runs from nearby starts stay near the saddle. For convex f this cannot happen: every stationary point is a global minimum (Section 35.3).

35.1.1.4 Backtracking Line Search

The step $\eta = 1/L$ presumes that we know L. We rarely do, and the curvature that matters varies across the loss surface, so a single global constant is both unknown and too conservative. A **line search** chooses the step at run time. It tries a step and accepts it only if it achieves a fixed fraction $c \in (0, 1)$ of the decrease that the first-order model predicts, the **Armijo condition** (also called sufficient decrease) (Armijo 1966)

$$f(\mathbf{x} - \eta \nabla f(\mathbf{x})) \;\le\; f(\mathbf{x}) - c\,\eta \,\|\nabla f(\mathbf{x})\|^2. \tag{35.7}$$

The right side is the first-order prediction with its slope scaled down by c. **Backtracking** starts from an optimistic η_0 and halves η until Equation 35.7 holds. By the descent lemma the condition holds automatically once $1 - L\eta/2 \ge c$, that is, once $\eta \le 2(1-c)/L$. So the loop stops after finitely many halvings and accepts a step $\eta \ge \min\left(\eta_0,\, (1-c)/L\right)$, within a factor two of that threshold, without ever knowing L.

The argument needs L only on the region the method visits. Take the quartic $f(x) = \frac{1}{4}x^4$, which has no global L, from $x_0 = 3$ with $c = \frac{1}{2}$. Backtracking never increases f, so every iterate stays in $[-3, 3]$, where $f'' = 3x^2 \le 27$. A trial step with $\eta \le 1/27$ maps x to $x(1 - \eta x^2)$, which lies between $\frac{2}{3}x$ and x, so the whole segment stays inside $[-3, 3]$ as well, and $L = 27$ there is all the argument needs. The quartic's curvature is large far from the origin and vanishes near it, so no single fixed step suits the whole trajectory. We try both from this start: the fixed step $\eta = 0.3$, which would be harmless near the origin, and backtracking from $\eta_0 = 1$.

```
quartic = lambda v: 0.25 * (v ** 4).sum()
quartic_grad = lambda v: v ** 3

x = np.array([3.0])
for _ in range(6):                          # fixed step, too large at x0
    x = x - 0.3 * quartic_grad(x)
print(f'fixed eta = 0.3 from x0 = 3:  |x_6| = {abs(x[0]):.2e}  (diverged)')

x, etas = np.array([3.0]), []
for _ in range(25):                         # backtracking line search
    g = quartic_grad(x)
    eta = 1.0
    while quartic(x - eta * g) > quartic(x) - 0.5 * eta * (g * g).sum():
        eta *= 0.5                          # Armijo fails: halve the step
    x = x - eta * g
```

```
        etas.append(eta)
    print('accepted steps:', np.array(etas[:8]), '...')
    print(f'backtracking from x0 = 3:  f(x_25) = {quartic(x):.2e}  (monotone descent)')
```

```
fixed eta = 0.3 from x0 = 3:  |x_6| = 6.52e+103  (diverged)
accepted steps: [0.03125 0.0625  0.125   0.25    0.5     1.      1.      1.     ] ...
backtracking from x0 = 3:  f(x_25) = 1.09e-04  (monotone descent)
```

The fixed step diverges: after six steps the iterate sits at $|x_6| \approx 6.5 \times 10^{103}$. The update $x \mapsto x(1 - \eta x^2)$ shrinks $|x|$ if and only if $\eta x^2 < 2$, so from $x_0 = 3$ a fixed step converges for $\eta < 2/9 \approx 0.22$ and diverges above it (exercise 4b). Note that the threshold is not $2/f''(x_0) = 2/27$; that rule belongs to quadratics. A step toward the origin meets the average of f'' between 0 and x, which is x^2, not its largest value $3x^2$. The descent lemma has to assume the largest and certifies only a third of the safe range.

Backtracking accepts 0.031 on the first step. The halving before it, 0.0625, fails the Armijo test, which on this function holds if and only if $\eta x^2 \leq 0.456$, which at x_0 means η up to about 0.05. As the iterate approaches the origin the accepted steps grow, $0.0625, 0.125, 0.25, 0.5, 1.0$, each close to $0.3/x^2$: the line search measures the curvature by trial. Training rarely backtracks, because a minibatch loss is too noisy to compare two steps and each probe costs a forward pass. Practice fixes the step in advance by a schedule (Section 9.6).

35.1.2 The Quadratic Model and the Condition Number

The opening promised to explain the difference between 69 steps and $6{,}908$. Both functions are quadratics, and on a quadratic gradient descent can be solved exactly. The answer depends on a single number, the ratio of the largest to the smallest curvature.

35.1.2.1 Ill-Conditioned Valleys

Figure 35.2 shows gradient descent on two quadratics before any algebra. On a round bowl the path runs nearly straight to the minimum. In a long valley the walls are steep and the floor is flat. The step that the steep walls allow makes the iterate overshoot across the valley and alternate from wall to wall. Along the flat floor the same step barely moves, and the path zigzags. (The valley of the figure is $\mathrm{diag}(1, 20)$ with $\eta = 0.097$, just below its ceiling 0.1, which gives the two factors in the caption.)

Quadratics are worth solving because every smooth function looks like one near a minimum. Near a minimizer $\mathbf{x}^\star$ with positive-definite Hessian $\mathbf{A}$, the second-order Taylor expansion says that f is approximately $f(\mathbf{x}^\star) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^\star)^\top \mathbf{A}\,(\mathbf{x} - \mathbf{x}^\star)$. Moving the minimizer to the origin and dropping the constant leaves the model problem

$$f(\mathbf{x}) = \tfrac{1}{2}\,\mathbf{x}^\top \mathbf{A}\,\mathbf{x}, \qquad \mathbf{A} \text{ symmetric positive definite, with eigenvalues } 0 < \lambda_{\min} = \lambda_1 \leq \cdots \leq \lambda_d = \lambda_{\max}.$$

Its gradient is $\mathbf{Ax}$, its Hessian is $\mathbf{A}$, and so its smoothness constant is $L = \lambda_{\max}$. The ratio of the largest to the smallest curvature,

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}},$$

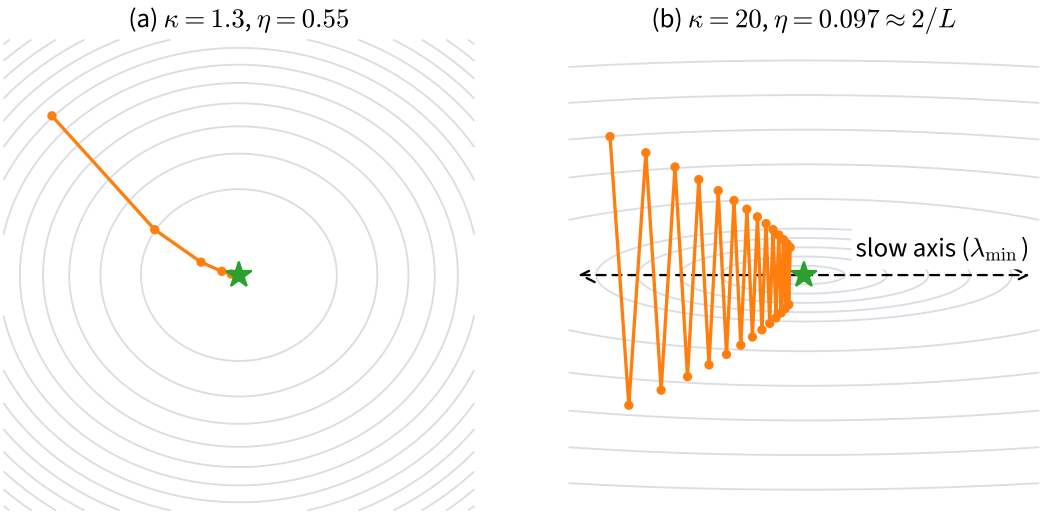


Figure 35.2: Gradient descent on two quadratics $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$. Left: a well-conditioned bowl; the path runs nearly straight to the minimum. Right: an ill-conditioned valley; a step near the stability ceiling makes the steep mode overshoot and alternate (factor -0.94) while the slow axis contracts by only 0.90 per step.

is the **condition number** of the problem. It is 10 for $\mathrm{diag}(1, 10)$ and 10^3 for the chapter's running example. For a symmetric positive definite $\mathbf{A}$ it equals the singular-value ratio $\kappa(\mathbf{A})$ of Section 33.5.3.2.

35.1.2.2 Per-Mode Contraction and the Stability Ceiling

The zigzag in the valley comes from a ceiling on the step that the steep direction imposes. To see where that ceiling lies, and what it costs the flat direction, we solve gradient descent on a quadratic exactly.

Start with one variable, $f(u) = \frac{\lambda}{2}u^2$ with $\lambda > 0$. The gradient step is $u_{k+1} = u_k - \eta\lambda u_k$, so

$$u_{k+1} = (1 - \eta\lambda)\, u_k, \qquad u_k = (1 - \eta\lambda)^k\, u_0.$$

Every step multiplies the error by the same factor. The iterates converge if and only if $|1 - \eta\lambda| < 1$, that is, $0 < \eta < 2/\lambda$. They land on 0 in one step if and only if $\eta = 1/\lambda$. For $1/\lambda < \eta < 2/\lambda$ the factor is negative, and the iterate alternates in sign while it shrinks. With $\lambda = 10$: $\eta = 0.1$ gives the factor 0, $\eta = 0.19$ gives -0.9, and $\eta = 0.21$ gives -1.1, which diverges.

The general quadratic keeps this form. Its gradient is $\mathbf{A}\mathbf{x}$, so the iteration Equation 35.2 is

$$\mathbf{x}_{k+1} = (\mathbf{I} - \eta\mathbf{A})\, \mathbf{x}_k.$$

Every step applies the same linear map, so the whole run is the matrix power $\mathbf{x}_k = (\mathbf{I} - \eta\mathbf{A})^k \mathbf{x}_0$. A symmetric matrix has an orthonormal basis of eigenvectors (Section 33.3). Write

$\mathbf{x}_k$ in the eigenbasis of $\mathbf{A}$ with coordinates $u_i^{(k)}$. The matrix $\mathbf{I}-\eta\mathbf{A}$ has the same eigenvectors, with eigenvalues $1-\eta\lambda_i$, so the coordinates evolve independently:

$$u_i^{(k+1)} = (1-\eta\lambda_i)\,u_i^{(k)}, \qquad \text{hence} \qquad u_i^{(k)} = (1-\eta\lambda_i)^k\,u_i^{(0)}. \tag{35.8}$$

Gradient descent on a quadratic is d copies of the one-variable recursion, one per eigenvalue, or **mode**. The factors $1-\eta\lambda_i$ decide convergence and speed:

- **Convergence** requires $|1-\eta\lambda_i| < 1$ for every mode, that is, $0 < \eta < 2/\lambda_{\max} = 2/L$. This is the **stability ceiling**. On quadratics, the $2/L$ that the descent lemma offered as sufficient is also necessary. Above it, the stiffest mode's factor is below -1, so that coordinate alternates with growing amplitude even while the other modes converge (exercise 4b builds such a half-converging, half-exploding run).
- **Speed** is set by the slowest mode. The error norm shrinks like $\rho(\eta)^k$, where $\rho(\eta) = \max_i |1-\eta\lambda_i|$ is the **spectral radius** of $\mathbf{I}-\eta\mathbf{A}$, its largest eigenvalue magnitude.

On $\mathbf{A} = \mathrm{diag}(1, 10)$ the ceiling is $\eta = 0.2$. Before reading the sweep below, guess which step leaves the smallest error after 60 steps from $(1, 1)$: the cautious 0.02, the step $0.1 = 1/L$ that the descent lemma favoured, or one closer to the ceiling.

```
A = np.diag([1.0, 10.0])                    # eigenvalues 1 and 10, so L = 10
x0 = np.array([1.0, 1.0])

def run_gd(A, x0, eta, steps):
    xs = [x0]
    for _ in range(steps):
        xs.append(xs[-1] - eta * (A @ xs[-1]))
    return np.array(xs)

print(' eta    |1-eta*1|  |1-eta*10|  spectral radius   |x_60|')
for eta in [0.02, 0.10, 2 / 11, 0.19, 0.20, 0.21]:
    f1, f10 = abs(1 - eta * 1.0), abs(1 - eta * 10.0)
    print(f'{eta:5.3f}    {f1:6.3f}     {f10:6.3f}        {max(f1, f10):6.3f}     '
          f'{np.linalg.norm(run_gd(A, x0, eta, 60)[-1]):9.1e}')
```

```
 eta    |1-eta*1|  |1-eta*10|  spectral radius   |x_60|
0.020     0.980      0.800         0.980       3.0e-01
0.100     0.900      0.000         0.900       1.8e-03
0.182     0.818      0.818         0.818       8.3e-06
0.190     0.810      0.900         0.900       1.8e-03
0.200     0.800      1.000         1.000       1.0e+00
0.210     0.790      1.100         1.100       3.0e+02
```

The cautious $\eta = 0.02$ is stable but slow ($\rho = 0.98$; after 60 steps the error is still 0.3). At $\eta = 0.1$ the stiff mode is eliminated in one step, yet the slow mode sets the rate, $\rho = 0.9$. The step $\eta = 0.19$ also has $\rho = 0.9$, for the opposite reason: its stiff mode alternates with factor -0.9. Between them, $\eta = 2/11 \approx 0.182$ balances the two extreme modes and wins by two orders of magnitude ($\rho = 0.818$, error 8.3×10^{-6}). At the ceiling $\eta = 0.2$ the stiff mode's factor is -1: the iterate alternates between ± 1 on that axis, and $\|\mathbf{x}_{60}\| = 1.0$. At $\eta = 0.21$, sixty steps increase the error to 3×10^2.

35.1.2.3 The Optimal Step

Within the stable range, which fixed η is fastest? The sweep suggests balancing the two extreme modes. For fixed η, the function $\lambda \mapsto |1-\eta\lambda|$ is a V with its vertex at $\lambda = 1/\eta$, so on $[\lambda_{\min}, \lambda_{\max}]$ its maximum sits at an endpoint, and $\rho(\eta) = \max\left(|1-\eta\lambda_{\min}|,\ |1-\eta\lambda_{\max}|\right)$. Once η exceeds $1/\lambda_{\max}$, raising it further lowers the value at $\lambda_{\min}$ and raises the one at $\lambda_{\max}$.

Proposition (optimal fixed step on a quadratic). *For $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ the convergence factor $\rho(\eta) = \max_i |1-\eta\lambda_i|$ is minimized by*

$$\eta^\star = \frac{2}{\lambda_{\min}+\lambda_{\max}}, \qquad \rho(\eta^\star) = \frac{\kappa-1}{\kappa+1}, \qquad \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}. \tag{35.9}$$

Proof. For every η, $\rho(\eta) \geq \max(1-\eta\lambda_{\min},\ \eta\lambda_{\max}-1)$, and a maximum of two numbers is at least any weighted average of them. Take the weights $\lambda_{\max}/(\lambda_{\min}+\lambda_{\max})$ and $\lambda_{\min}/(\lambda_{\min}+\lambda_{\max})$. The terms in η cancel:

$$\rho(\eta) \ \geq\ \frac{\lambda_{\max}(1-\eta\lambda_{\min}) + \lambda_{\min}(\eta\lambda_{\max}-1)}{\lambda_{\min}+\lambda_{\max}} = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} = \frac{\kappa-1}{\kappa+1}.$$

Equality requires the two terms to be equal, $1-\eta\lambda_{\min} = \eta\lambda_{\max}-1$, which is $\eta = \eta^\star$. At $\eta^\star$ both endpoint values are $(\kappa-1)/(\kappa+1)$, and the V keeps every interior eigenvalue below them, so the bound is attained. ■

Figure 35.3 is this proof drawn on $\mathrm{diag}(1, 10)$: the best V is the one whose endpoints are level. There $\eta^\star = 2/11$ and $\rho = 9/11 = 0.818$, so six digits take $\log 10^6 / \log(11/9) \approx 69$ steps. On the chapter's running example $\rho = 999/1001$, and six digits take $6{,}908$ steps.

When $\kappa = 1$ (a round bowl) the rate is 0 and one step solves the problem. As κ grows, $(\kappa-1)/(\kappa+1) \approx 1-2/\kappa$, so $\log(1/\rho) \approx 2/\kappa$, and reducing the error by a factor ϵ costs about $\frac{\kappa}{2}\log\frac{1}{\epsilon}$ iterations: **the cost of gradient descent is linear in the condition number**. Stability limits the step by the stiff mode, $\eta \lesssim 2/\lambda_{\max}$, and with that step the slow mode moves by only $\eta\,\lambda_{\min} \approx 2/\kappa$ per step.

At $\eta^\star$ the factor ρ is more than an asymptotic rate. On $\mathrm{diag}(1, 10)$ the two per-mode factors are $+9/11$ and $-9/11$ (the row $\eta = 2/11$ of the sweep): the slow mode contracts monotonically while the stiff mode alternates in sign. Both modes shrink by the same factor, so $\|\mathbf{x}_k\|$ shrinks by $(\kappa-1)/(\kappa+1)$ at every single step, not merely on average.

35.1.2.4 The Edge of Stability

The stability ceiling suggests a recipe: measure the **sharpness**, the largest Hessian eigenvalue $\lambda_{\max}(\nabla^2 f)$, and choose $\eta < 2/\lambda_{\max}$. On a network this is not enough, because the Hessian changes along the run. Under full-batch gradient descent the sharpness typically rises until it reaches about $2/\eta$ and then stays there, while the loss keeps falling over the run but rises on many single steps (Cohen et al. 2021). Section 9.1 runs the experiment on a small tanh network and discusses it. Here we ask only how much of it the quadratic model explains.

It explains half. In the eigenbasis of the Hessian the top mode's factor is $1-\eta\lambda_{\max}$, which is below -1 exactly when $\lambda_{\max} > 2/\eta$. That coordinate then grows and alternates in sign, which is how a loss that falls over the run can rise on single steps. On a fixed quadratic the

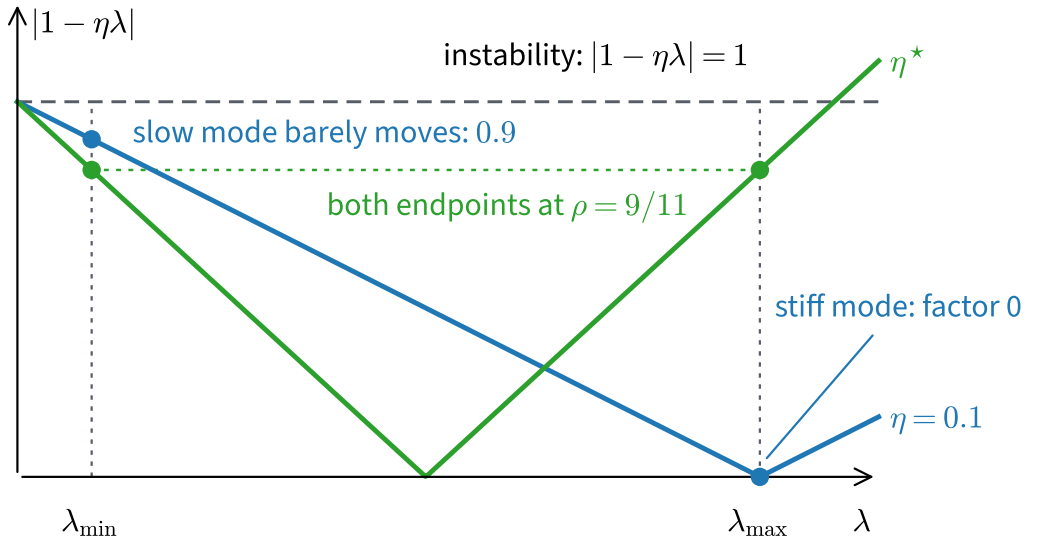


Figure 35.3: The per-mode contraction factors $|1-\eta\lambda|$ form a V with its vertex at $\lambda = 1/\eta$, and the convergence factor $\rho(\eta)$ is the larger of its values at the extreme eigenvalues $\lambda_{\min} = 1$, $\lambda_{\max} = 10$. The choice $\eta = 0.1$ eliminates the stiff mode but leaves the slow mode contracting at 0.9; the optimal $\eta^\star = 2/(\lambda_{\min}+\lambda_{\max})$ equalizes the two endpoint factors at $(\kappa-1)/(\kappa+1) = 9/11$. Lowering either endpoint would raise the other.

growth would end in divergence. Why it brings the sharpness back down instead lies outside the model: the third-order term of the Taylor expansion, which the quadratic model drops, turns the oscillation into a drift toward lower sharpness (Damian et al. 2023). Exercise 9 shows that with momentum the ceiling moves to $(2+2\beta)/\eta$.

35.1.2.5 From Quadratics to Convex Functions

What survives beyond quadratics is the linear dependence on κ. If f is L-smooth and μ-strongly convex Equation 35.26, it is sandwiched at every point between quadratics of curvature μ and L, the ratio $\kappa = L/\mu$ takes the place of $\lambda_{\max}/\lambda_{\min}$, and gradient descent with $\eta = 1/L$ satisfies

$$f(\mathbf{x}_k) - f^\star \le \left(1 - \frac{1}{\kappa}\right)^k (f(\mathbf{x}_0) - f^\star), \tag{35.10}$$

which again means $O(\kappa \log \frac{1}{\epsilon})$ steps. A function that is convex and smooth but not strongly convex still reaches its optimal value, at the slower rate $O(1/k)$, and without convexity only the stationarity bound Equation 35.6 remains. The proofs need convexity and are in Section 35.3.3.3, which also compares these bounds with what the quadratic achieves.

35.1.3 Momentum and Acceleration

Gradient descent needs a number of steps proportional to κ because one step size must serve both the stiff and the flat modes. Momentum keeps the gradient and changes the update so that the two modes can move at the same rate.

35.1.3.1 Heavy-Ball Momentum

The valley suggests the change. In the stiff direction successive gradients point in alternating directions (the overshoot of Figure 35.2), so averaging recent gradients would cancel the oscillation. In the slow direction successive gradients are small but agree, so accumulating them would build up speed. One mechanism does both: give the iterate a velocity with memory. **Heavy-ball momentum** (Polyak 1964) replaces the gradient step by

$$\mathbf{v}_{k+1} = \beta\,\mathbf{v}_k - \eta\,\nabla f(\mathbf{x}_k), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1}, \tag{35.11}$$

with momentum parameter $\beta \in [0, 1)$. The velocity $\mathbf{v}_k$ is a decaying sum of past gradients. Since $\mathbf{v}_k = \mathbf{x}_k - \mathbf{x}_{k-1}$, eliminating it gives

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla f(\mathbf{x}_k) + \beta\,(\mathbf{x}_k - \mathbf{x}_{k-1}).$$

Each step is a gradient step plus β times the previous step; with $\beta = 0$ it is gradient descent.

On the quadratic each mode again evolves on its own. With gradient λu_k in one mode, the eliminated form reads $u_{k+1} = (1 + \beta - \eta\lambda)\,u_k - \beta\,u_{k-1}$, so the new value depends on the last two. A second-order recurrence of this kind is the discrete version of a damped oscillator, a mass on a spring with friction $1 - \beta$, and β sets the damping.

Figure 35.4 shows the three regimes on $\mathrm{diag}(1, 10)$, all with the gradient-descent step $\eta = 2/11$. With too little momentum the path zigzags much like gradient descent. With the right amount it settles after one overshoot. With too much, the iterate loops around the minimum. The next subsection attaches a rate to each regime and shows that the best rate of all needs a larger step as well.

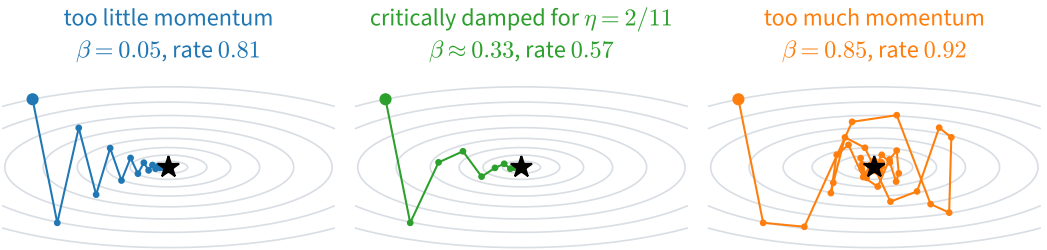


Figure 35.4: Heavy ball on the $\kappa = 10$ valley $\frac{1}{2}(x^2 + 10y^2)$, all three runs with the gradient-descent step $\eta = 2/11$. Too little momentum zigzags like gradient descent (the stiff mode's coefficient $1 + \beta - \eta\lambda_{\max}$ is negative, so a root is negative and real); the critically damped β for this step is fastest; too much momentum loops around the minimum. The $\sqrt{\kappa}$ rate 0.52 needs the larger step η_{HB} of Equation 35.13, at which $\beta = 0.05$ would diverge.

35.1.3.2 The $\sqrt{\kappa}$ Law

We now show that with the right pair (η, β) every mode contracts at the same rate $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$. For $\kappa = 10$ the pair is $\beta^\star \approx 0.27$, $\eta_{\text{HB}} \approx 0.23$, and the rate is 0.52 against gradient descent's 0.82: six digits in 27 steps instead of 69. Everything follows from one fact about a single mode.

Proposition (heavy ball contracts every mode at $\sqrt{\beta}$). *Let $0 < \beta < 1$. On $f(u) = \frac{\lambda}{2}u^2$ the heavy-ball iterates satisfy $u_{k+1} = (1+\beta-\eta\lambda)u_k - \beta u_{k-1}$. If $(1-\sqrt{\beta})^2 \le \eta\lambda \le (1+\sqrt{\beta})^2$, both roots of the characteristic polynomial have modulus $\sqrt{\beta}$, and $|u_k| \le C\, k\, \beta^{k/2}$ for $k \ge 1$ and a constant C that depends on u_0, u_1 and $\eta\lambda$. The factor k is needed only at the two endpoints, where the root is double.*

Proof. Substitute $u_k = z^k$ into the recurrence and divide by z^{k-1}:

$$z^2 - (1+\beta-\eta\lambda)\, z + \beta = 0. \tag{35.12}$$

The product of the two roots is β. The roots are complex exactly when the discriminant $(1+\beta-\eta\lambda)^2 - 4\beta$ is negative, that is, when $|1+\beta-\eta\lambda| < 2\sqrt{\beta}$, which rearranges to $(1-\sqrt{\beta})^2 < \eta\lambda < (1+\sqrt{\beta})^2$. Complex-conjugate roots $z, \bar{z}$ with $z\bar{z} = \beta$ each have modulus $\sqrt{\beta}$, whatever λ is, and the general solution $u_k = az^k + b\bar{z}^k$ is bounded by $(|a|+|b|)\beta^{k/2}$. At the two endpoints the discriminant is zero, and the double root is $(1+\beta-\eta\lambda)/2 = \pm\sqrt{\beta}$. A double root z gives the general solution $u_k = (a+bk)z^k$, which is the factor k. ■

Corollary (the tuning). Place the two extreme modes at the two ends of the interval: $\eta\lambda_{\min} = (1-\sqrt{\beta})^2$ and $\eta\lambda_{\max} = (1+\sqrt{\beta})^2$. Every other eigenvalue then lies inside, so every mode has modulus $\sqrt{\beta}$. Taking square roots, $\sqrt{\eta\lambda_{\max}} - \sqrt{\eta\lambda_{\min}} = 2\sqrt{\beta}$ and $\sqrt{\eta\lambda_{\max}} + \sqrt{\eta\lambda_{\min}} = 2$. The second equation gives η, and the ratio of the two gives $\sqrt{\beta} = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$. With the tuning

$$\eta_{\text{HB}} = \left(\frac{2}{\sqrt{\lambda_{\max}} + \sqrt{\lambda_{\min}}}\right)^2, \qquad \beta^\star = \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2, \tag{35.13}$$

heavy ball contracts every mode by the factor

$$\rho_{\text{HB}} = \sqrt{\beta^\star} = \frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1} \;\approx\; 1 - \frac{2}{\sqrt{\kappa}}. \tag{35.14}$$

This is gradient descent's rate with κ replaced by $\sqrt{\kappa}$, so the iteration count drops from $O(\kappa \log \frac{1}{\epsilon})$ to $O(\sqrt{\kappa} \log \frac{1}{\epsilon})$ (Polyak 1964). At finite k the double roots of the two extreme modes add a term. The error behaves like $k\rho^k$, and with $\log(1/\rho) \approx 2/\sqrt{\kappa}$ the count for a target ϵ solves $k \approx \frac{\sqrt{\kappa}}{2}(\log \frac{1}{\epsilon} + \log k)$, the asymptotic law plus a term of order $\sqrt{\kappa}\log\kappa$.

Figure 35.5 draws the proof for $\kappa = 10$: the interval on which the modulus is flat at $\sqrt{\beta}$, and the roots themselves, every pair on the circle of radius $\sqrt{\beta^\star} = 0.52$, while gradient descent's factors spread along the real axis up to ± 0.82. Momentum lets the stiff and the flat mode share one complex modulus instead of forcing the flat mode to crawl at $1 - 2/\kappa$.

The proposition also gives the rates of the three runs in Figure 35.4, whose step puts the two modes at $\eta\lambda = 0.18$ and 1.82. For $\beta = 0.85$ both lie inside the interval: the rate is $\sqrt{0.85} = 0.92$, and the complex roots make the iterate loop. The smallest β whose interval still holds the slow mode, $\beta = (1 - \sqrt{2/11})^2 \approx 0.33$, is the fastest for this step, with rate 0.57. For $\beta = 0.05$ the interval $[0.60, 1.50]$ holds neither mode. The slow one has the real root 0.81, and the stiff one a negative real root, which makes the iterate alternate, something a physical oscillator never does.

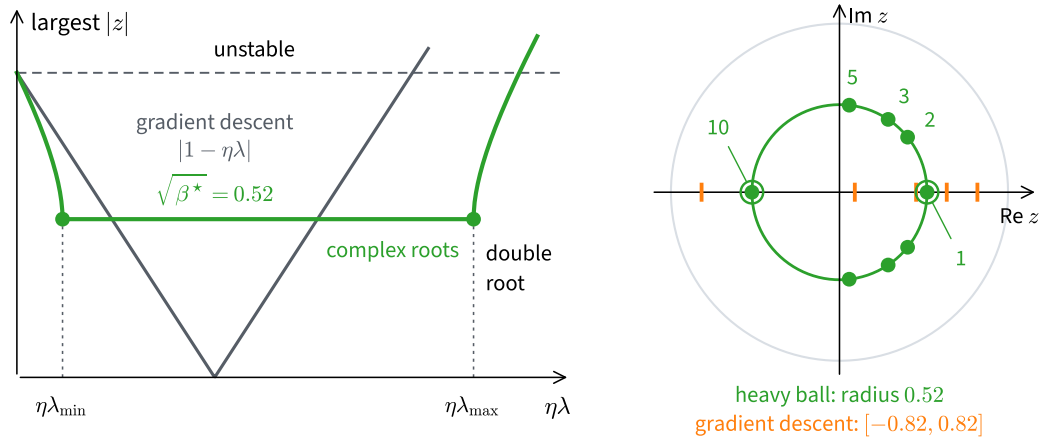


Figure 35.5: Heavy ball at the tuning of Equation 35.13 on the $\kappa = 10$ example. (a) For fixed β the modulus of the larger root is flat at $\sqrt{\beta}$ wherever the roots are complex; the tuning places the two extreme modes at the ends of that interval. (b) In the complex plane every mode's root pair (labeled by its eigenvalue λ) lies on the circle of radius $\sqrt{\beta^\star} = 0.52$, while gradient descent's factors spread over $[-0.82, 0.82]$ on the real axis. The extreme modes are double roots, which is the factor k in the error.

The law is exact for quadratics only. Lessard et al. (2016) constructed a one-dimensional, smooth, strongly convex function on which heavy ball with this tuning does not converge from some starting points: the iterates fall into a stable cycle (exercise 7). Momentum in deep learning (Sutskever et al. 2013) rests on the local quadratic model and on experience, not on a global theorem.

35.1.3.3 Nesterov's Look-Ahead

A variant that does carry a global guarantee evaluates the gradient at a look-ahead point displaced by the momentum. **Nesterov's accelerated gradient** (Nesterov 1983) is

$$\mathbf{v}_{k+1} = \beta\,\mathbf{v}_k - \eta\,\nabla f(\mathbf{x}_k + \beta\,\mathbf{v}_k), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{v}_{k+1}. \tag{35.15}$$

The only change from heavy ball is where the gradient is taken. When the momentum is about to overshoot, the gradient at the look-ahead point already points back and brakes the step.

Proposition (Nesterov on quadratics). *Let $\kappa = L/\mu$. With $\eta = 1/L$ and $\beta = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$, every mode of a quadratic with eigenvalues in $[\mu, L]$ has roots of modulus at most $1 - 1/\sqrt{\kappa}$, with a double root at the slow mode $\lambda = \mu$, so the error is at most $C\,k\,(1 - 1/\sqrt{\kappa})^k$.*

Proof. In one mode with eigenvalue λ, write $a = 1 - \eta\lambda$, which lies in $[0, 1 - 1/\kappa]$. The velocity update becomes $v_{k+1} = a\beta v_k - \eta\lambda u_k$, and $u_{k+1} = a u_k + a\beta v_k$; substituting $v_k = u_k - u_{k-1}$ gives $u_{k+1} = (1 + \beta)a\, u_k - \beta a\, u_{k-1}$. The characteristic polynomial $z^2 - (1 + \beta)a\, z + \beta a$ has discriminant $a\,[(1 + \beta)^2 a - 4\beta]$. With this β, $(1 + \beta)^2 = 4\kappa/(\sqrt{\kappa}+1)^2$. At the slow mode, $a = (\kappa-1)/\kappa$, the bracket is $4(\kappa-1)/(\sqrt{\kappa}+1)^2 - 4\beta = 4\beta - 4\beta = 0$: a double root at $(1+\beta)a/2 = (\sqrt{\kappa}-1)/\sqrt{\kappa} = 1 - 1/\sqrt{\kappa}$. Every stiffer mode has a smaller a, a negative bracket, and complex roots (or roots 0 when $a = 0$) of modulus $\sqrt{\beta a} < \sqrt{\beta(1 - 1/\kappa)} = 1 - 1/\sqrt{\kappa}$. ■

The rate is 0.684 for $\kappa = 10$, 0.900 for $\kappa = 100$, and 0.968 for $\kappa = 1000$, slower than heavy ball's 0.519, 0.818, and 0.939, but with the same $\sqrt{\kappa}$ dependence. Unlike heavy ball, Nesterov's method keeps its guarantees beyond quadratics, which we state without proof (Nesterov 2018). For μ-strongly convex, L-smooth f, the method with the same constant β satisfies $f(\mathbf{x}_k) - f^\star \le C\,(1 - 1/\sqrt{\kappa})^k$ on every function of the class. Mind the units: this bounds the gap $f - f^\star$, which scales as the square of the distance, so it has half the exponent that the proposition gives on a quadratic. For smooth convex f a schedule $\beta_k \to 1$ gives $f(\mathbf{x}_k) - f^\star = O(1/k^2)$, against gradient descent's $O(1/k)$. No method that forms its iterates from past gradients can beat $O(1/k^2)$ or the $\sqrt{\kappa}$ dependence in the worst case, as long as the dimension is large compared with the number of steps (Nemirovski and Yudin 1983). Exercise 8 builds the worst function.

It remains to collect the counts that the opening promised. On $\mathrm{diag}(1, \kappa)$ from $(1, 1)$, six digits should cost gradient descent $\frac{\kappa}{2}\log 10^6 \approx 6.9\,\kappa$ steps and heavy ball somewhat more than $6.9\sqrt{\kappa}$.

```
def iterations(lam, eta, beta=0.0, lookahead=False, tol=1e-6):
    """Steps until |x_k| <= tol |x_0| on f(x) = sum_i lam_i x_i^2 / 2."""
    x, v = np.array([1.0, 1.0]), np.zeros(2)
    n0 = np.linalg.norm(x)
    for k in range(1, 10**6):
        g = lam * (x + beta * v) if lookahead else lam * x
        v = beta * v - eta * g
        x = x + v
        if np.linalg.norm(x) <= tol * n0:
            return k

print('kappa      GD   heavy ball   Nesterov     GD/kappa   HB/sqrt(kappa)')
for kappa in [10.0, 100.0, 1000.0]:
    lam, sk = np.array([1.0, kappa]), np.sqrt(kappa)
    gd = iterations(lam, 2 / (1 + kappa))
    hb = iterations(lam, (2 / (1 + sk))**2, ((sk - 1) / (sk + 1))**2)
    na = iterations(lam, 1 / kappa, (sk - 1) / (sk + 1), lookahead=True)
    print(f'{kappa:5.0f} {gd:7d} {hb:9d} {na:10d}      '
          f'{gd / kappa:7.1f}     {hb / sk:7.1f}')
```

```
kappa      GD   heavy ball   Nesterov     GD/kappa   HB/sqrt(kappa)
   10      69         27         43          6.9          8.5
  100     691         93        155          6.9          9.3
 1000    6908        315        508          6.9         10.0
```

Gradient descent's count divided by κ sits at 6.9, so it needs $6{,}908$ steps at $\kappa = 1000$. Heavy ball needs 315 there. Its count divided by $\sqrt{\kappa}$ rises, 8.5, 9.3, 10.0, because each extreme

mode is a double root. Its 2×2 iteration matrix is then a Jordan block, a repeated eigenvalue with a single eigenvector (Section 33.3.1.5), whose k-th power carries the factor k. The formula $k \approx \frac{\sqrt{\kappa}}{2}(\log 10^6 + \log k)$ predicts 8.6, 9.2, 9.8, and at $\kappa = 10^4$ heavy ball needs $1{,}056$ steps, ratio 10.6 (exercise 6). Nesterov's 508 at $\kappa = 1000$ sits between the two, as its rate 0.968 against heavy ball's 0.939 predicts. All three methods use one gradient per step, so momentum's speedup costs no extra gradients. The square root is also as far as gradients will go: by the lower bound, no method built from past gradients does better in high dimension. To remove κ altogether a step has to know the curvature (Section 35.2).

35.1.4 Summary

A gradient step can be trusted up to $\eta = 2/L$. On a quadratic the best fixed step contracts the error by $(\kappa-1)/(\kappa+1)$ per step, so the number of steps grows linearly with the condition number κ: stability is set by the stiffest mode and progress by the flattest. Momentum makes every mode contract at the same rate $\sqrt{\beta}$, and its tuned rate $(\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ cuts the count to order $\sqrt{\kappa}$. On a network the sharpness moves to the ceiling $2/\eta$ of the chosen step, which the quadratic model explains only in part. Three questions remain open: what a step can do when it knows the curvature (Section 35.2), when a stationary point is the global minimum (Section 35.3), and what noisy gradients cost (Section 35.4).

35.1.5 Exercises

1. **Preconditioned descent directions.** The proposition of Section 35.1.1.1 measures the length of a direction by the Euclidean norm. Other norms give other steepest directions.
    1. Show that $\mathbf{d} = -\mathbf{P}^{-1}\nabla f(\mathbf{x})$ is a descent direction for every positive definite matrix $\mathbf{P}$.
    2. Measure length by $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^\top \mathbf{P}\,\mathbf{d}}$. Show that the minimizer of $\nabla f(\mathbf{x})^\top \mathbf{d}$ over $\|\mathbf{d}\|_{\mathbf{P}} = 1$ is proportional to $-\mathbf{P}^{-1}\nabla f(\mathbf{x})$, and find the $\mathbf{P}$ for which it is Newton's direction Equation 35.17. The substitution $\mathbf{u} = \mathbf{P}^{1/2}\mathbf{d}$ reduces the problem to the Euclidean case.
2. code **Smoothness constants.** The smoothness constant L of Equation 35.3 bounds the curvature of f, and for a logistic loss it can be computed from the data.
    1. Let f be twice continuously differentiable and L-smooth. Show that $\|\nabla^2 f(\mathbf{x})\,\mathbf{v}\| \le L\,\|\mathbf{v}\|$ for all $\mathbf{x}$ and $\mathbf{v}$, and conclude that every Hessian eigenvalue lies in $[-L, L]$. This is the converse direction of the equivalence stated after Equation 35.3.
    2. Let $f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i\,\mathbf{x}_i^\top \mathbf{w})\right) + \frac{\nu}{2}\|\mathbf{w}\|^2$ with labels $y_i = \pm 1$ and $\nu = 0.1$, the loss `loss` of the data cell of Section 35.2.1, where $\mathbf{x}_i^\top$ is the i-th row of $\mathbf{X}$. Show that $L = \lambda_{\max}(\mathbf{X}^\top\mathbf{X})/(4n) + \nu$ is a valid smoothness constant and that the Hessian attains this bound at $\mathbf{w} = \mathbf{0}$.
    3. Compute this L for the data of that cell. It comes out near 0.4 for every draw; explain why, using $\lambda_{\max}(\mathbf{X}^\top\mathbf{X})/n \approx (1 + \sqrt{d/n})^2$ for standard Gaussian entries. Count the steps that $\eta = 0.5$ (the reference's step) and $\eta = 1/L$ each need, from $\mathbf{w} = \mathbf{0}$, to reach $\|\nabla f(\mathbf{w})\| \le 10^{-10}$. Which is faster, and does $\eta = 0.5$ stay below $2/L$?

3. code **A saddle that attracts.** The stationarity guarantee Equation 35.6 does not distinguish minima from saddle points. Consider $f(x, y) = \frac{1}{2}x^2 + \cos y$, which the text ran from $(1, 0)$ into the saddle $(0, 0)$.
   1. Show that f is bounded below and L-smooth with $L = 1$. Find all stationary points and classify each one as a minimum or a saddle point.
   2. Run gradient descent Equation 35.2 with $\eta = 1/L$ from $(1, \delta)$ for $\delta \in \{10^{-2}, 10^{-4}, 10^{-8}, 10^{-12}\}$ and count the steps until $f < -0.99$. Linearize the update $y_{k+1} = y_k + \sin y_k \approx 2y_k$ near $y = 0$ to predict how this count grows as δ shrinks, and compare the prediction with your counts.
   3. What do these runs suggest about whether gradient descent from a random initialization converges to a saddle point? What do they suggest about the time it can spend near one?

4. code **Step size and the ceiling.** Let $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\,\mathbf{x}$ with $\mathbf{A}$ symmetric positive definite and condition number κ. Suppose the step $\eta = 1/\hat{L}$ is computed from an estimate $\hat{L}$ of the smoothness constant $L = \lambda_{\max}$, and write $\alpha = L/\hat{L}$, so that $\eta = \alpha/\lambda_{\max}$.
   1. Use Equation 35.8 to express the convergence factor $\rho(\eta) = \max_i |1 - \eta\lambda_i|$ through α and κ. Show that for $0 < \alpha \le 1$, reducing the error by a factor ϵ takes about $(\kappa/\alpha)\log(1/\epsilon)$ steps. What does overestimating L by a factor of two cost? What does underestimating it by a factor of two cost?
   2. On the quartic $f(x) = \frac{1}{4}x^4$ from $x_0 = 3$, find the largest fixed step for which gradient descent converges. Show that $|x_{k+1}| < |x_k|$ if and only if $0 < \eta x_k^2 < 2$, conclude that every $\eta < 2/9$ converges, and show that at $\eta = 2/9$ the iterate flips between 3 and -3 forever. Confirm both by a run. Backtracking accepted 0.031 on the first step and 1.0 later; explain why a fixed step must respect the curvature at the starting point for the whole run.

5. **Steepest descent in other norms.** ▣ Problem 1b measures length by a norm $\|\cdot\|_{\mathbf{A}}$ that comes from an inner product. Norms that do not come from an inner product give steepest directions of a different form.
   1. Find the minimizers of $\nabla f(\mathbf{x})^\top \mathbf{d}$ over $\|\mathbf{d}\|_\infty = 1$ and over $\|\mathbf{d}\|_1 = 1$, together with the minimal slopes, and state when each minimizer is unique.
   2. Let the parameter be a matrix whose gradient $\mathbf{G}$ has the reduced SVD $\mathbf{G} = \mathbf{U}\,\mathrm{diag}(\sigma_i)\,\mathbf{V}^\top$, let the slope of a direction $\mathbf{D}$ be $\mathrm{tr}(\mathbf{G}^\top\mathbf{D})$, and measure length by the spectral norm $\|\mathbf{D}\|_2$. Show that the minimal slope over $\|\mathbf{D}\|_2 \le 1$ is $-\sum_i \sigma_i$, attained at $\mathbf{D} = -\mathbf{U}\mathbf{V}^\top$, the polar factor that Muon applies to its momentum matrix (Section 9.7). Writing $\mathrm{tr}(\mathbf{G}^\top\mathbf{D}) = \sum_i \sigma_i\, \mathbf{u}_i^\top \mathbf{D}\,\mathbf{v}_i$ is a useful first step. This is the steepest-descent view of Bernstein and Newhouse (2024) .

6. code **Heavy ball: the double root and the factor k.** ▣ Apply heavy ball Equation 35.11 to $f(u) = \frac{\lambda}{2}u^2$.
   1. Write one step as a linear map of the state (u_k, u_{k-1}) and show that the characteristic polynomial of its 2×2 matrix is Equation 35.12 of the text.
   2. At the tuning Equation 35.13 the two extreme modes have the double eigenvalues $\sqrt{\beta^\star}$ and $-\sqrt{\beta^\star}$. Show that the matrix then has a single eigenvector, and that the k-th power of the Jordan block $\left(\begin{smallmatrix}\rho & 1\\ 0 & \rho\end{smallmatrix}\right)$ has the entry $k\rho^{k-1}$ above the diagonal.
   3. With $\|\mathbf{e}_k\| \approx k\rho^k$ and $\log(1/\rho) \approx 2/\sqrt{\kappa}$, solve $k = \frac{\sqrt{\kappa}}{2}(\log\frac{1}{\epsilon} + \log k)$ numerically for $\epsilon = 10^{-6}$ and $\kappa \in \{10, 100, 1000\}$, and compare $k/\sqrt{\kappa}$ with the mo-

mentum cell's column $8.5, 9.3, 10.0$.

4. Run the momentum cell's heavy ball at $\kappa = 10^4$ and compare its count with your prediction. Explain why the ratio $k/\sqrt{\kappa}$ never settles as κ grows.

7. code **Heavy ball beyond quadratics.** ⊠ Lessard et al. (2016) define a one-dimensional function through its gradient,

$$\nabla f(x) = \begin{cases} 25x, & x < 1, \\ x + 24, & 1 \le x < 2, \\ 25x - 24, & x \ge 2. \end{cases}$$

1. Show that f is μ-strongly convex with $\mu = 1$ and L-smooth with $L = 25$.
2. Run heavy ball with the tuning Equation 35.13 for 300 steps from $x_0 = 3.3$ with zero initial velocity, and run Nesterov's method Equation 35.15 with $\eta = 1/L$ and $\beta = (\sqrt{\kappa} - 1)/(\sqrt{\kappa} + 1)$ from the same start, where $\kappa = L/\mu$. Plot x_k against k for both methods.
3. The heavy-ball iterates approach a cycle of period three that visits the piece $x < 1$ twice and the piece $x \ge 2$ once. Write the heavy-ball update on these pieces as a linear system for the three points of the cycle, solve it, and compare the solution with the simulation.
4. Repeat the heavy-ball run from a grid of starting points on $[-6, 6]$ and report the fraction that end on the cycle. Every piece of f is a quadratic with curvature in $[\mu, L]$. Why does the per-mode analysis behind Equation 35.14 nevertheless not apply to f?

8. code **A lower bound for first-order methods.** ⊠ This problem derives the $O(1/k^2)$ part of the bound of Nemirovski and Yudin (1983) quoted in Section 35.1.3, the part for smooth convex functions, following the construction of Nesterov (2018) . A first-order method here is any method whose iterates satisfy $\mathbf{x}_k \in \mathbf{x}_0 + \operatorname{span}\{\nabla f(\mathbf{x}_0), \dots, \nabla f(\mathbf{x}_{k-1})\}$; take $\mathbf{x}_0 = \mathbf{0}$. For $p \le d$, let $\mathbf{A}_p \in \mathbb{R}^{d \times d}$ be zero except for its leading $p \times p$ block, which is tridiagonal with 2 on the diagonal and -1 on both off-diagonals, and let

$$f_p(\mathbf{x}) = \frac{L}{4}\left(\tfrac{1}{2}\,\mathbf{x}^\top \mathbf{A}_p\,\mathbf{x} - x_1\right).$$

1. Show that f_p is convex and L-smooth.
2. Show that if $\mathbf{x}$ is supported on the first j coordinates, then $\nabla f_p(\mathbf{x})$ is supported on the first $j + 1$. Conclude that the k-th iterate of every first-order method is supported on the first k coordinates.
3. Show that the minimizer of f_p has entries $x_i^\star = 1 - i/(p+1)$ for $i \le p$ and zero beyond, that the minimum is $-\frac{L}{8}(1 - \frac{1}{p+1})$, and that $\|\mathbf{x}^\star\|^2 \le (p+1)/3$.
4. Let $2k + 1 \le d$. The functions f_{2k+1} and f_k agree on vectors supported on the first k coordinates. Use this fact to show that every first-order method applied to f_{2k+1} satisfies, with $\mathbf{x}^\star$ the minimizer of f_{2k+1},

$$f_{2k+1}(\mathbf{x}_k) - \min f_{2k+1} \;\ge\; \frac{3L\,\|\mathbf{x}_0 - \mathbf{x}^\star\|^2}{32\,(k+1)^2}.$$

The construction needs the dimension $d \geq 2k+1$, so the bound governs about the first $d/2$ iterations. In a fixed dimension, long runs can do better asymptotically, but for networks with d in the millions the restriction is inactive.

5. For $k \in \{5, 10, 25, 50, 100, 200, 400\}$, run k steps of gradient descent with $\eta = 1/L$ and of Nesterov's update Equation 35.15 with $\eta = 1/L$ and $\beta_j = (j-1)/(j+2)$ at step $j = 1, 2, \ldots$ on f_{2k+1} with $d = 2k+1$, and divide each gap by the bound. Which ratio stays bounded as k grows?

9. code **Edge of stability with momentum.** ▯▯ Cohen et al. (2021) report that with momentum, the sharpness settles at a threshold that differs from $2/\eta$.

   1. Use the iteration matrix of problem 6a to show that heavy ball Equation 35.11 on $f(x) = \frac{\lambda}{2}x^2$ with $0 \leq \beta < 1$ converges if and only if $0 < \eta\lambda < 2 + 2\beta$. Derive the corresponding interval for Nesterov's update Equation 35.15.
   2. Modify the edge-of-stability experiment of Section 9.1 (cell `#optimization-intro-the-edge-of-stability-1`) to train with heavy ball and with Nesterov's update for $\beta \in \{0.25, 0.5\}$, choosing for each run a step size whose predicted threshold lies between 5 and 12. Track the sharpness against $2/\eta$ and against the threshold from (a).
   3. At which threshold does the sharpness settle? What does the answer imply for choosing η from a sharpness measured at initialization when training with momentum?

35.2 Curvature Methods: Newton, Quasi-Newton, and Trust Regions

Gradient descent never looks at the curvature of the function it descends. It only suffers from it: the steepest direction caps the step, the flattest direction then crawls, and the number of steps grows with the condition number κ, or with $\sqrt{\kappa}$ once momentum is added (Section 35.1.3). On the chapter's running example $\frac{1}{2}(x_1^2 + 1000\, x_2^2)$ that came to $6{,}908$ steps, or 315 with momentum. Yet anyone can solve this problem by inspection: divide each coordinate of the gradient by its curvature, 1 or 1000, and a single step lands on the minimum. What does a method gain when it measures the curvature instead of merely surviving it?

Newton's method, a century and a half older than gradient descent, is the classical answer. It fits a quadratic to the function at the current point and jumps to the minimum of the quadratic. We ask four questions of it. How fast is it? What does a step cost? Where does it go wrong? And how much of it can be recovered from gradients alone, when second derivatives are out of reach? In brief: it is very fast near a minimum, it is far too expensive for a network, it runs to saddle points as readily as to minima, and a good part of it can be rebuilt by comparing successive gradients. The remedy for the saddles is to limit each step to the region where the quadratic has predicted well.

For training networks this section matters less than the previous one, and we should say so. A Hessian is too large to form, and minibatch noise spoils curvature estimates built from differences of gradients. The ideas return in other forms, though: the per-coordinate steps of Adam are a diagonal stand-in for Newton's matrix (Section 35.5), and the damping that such methods add to a curvature matrix is the shift of a trust region. The result to remember is the quadratic convergence of Newton's method Equation 35.18. The part on quasi-Newton methods can be skipped on a first reading. From Section 35.1 we reuse two

things: the Armijo test of the backtracking line search, and the function $\frac{1}{2}x^2 + \cos y$, at whose saddle gradient descent stopped.

35.2.1 Newton's Method

Gradient descent rests on a linear model of f. A linear model has no minimum, which is why the method needs a step size from outside. A quadratic model has a minimum of its own, and with it comes a step that needs no tuning.

35.2.1.1 The Newton Step

Newton's method lets the curvature choose the step (Section 9.2.3). Model f near $\mathbf{x}_k$ by its second-order Taylor expansion in the step $\mathbf{p}$:

$$m_k(\mathbf{p}) = f(\mathbf{x}_k) + \mathbf{g}_k^\top \mathbf{p} + \tfrac{1}{2}\,\mathbf{p}^\top \nabla^2 f(\mathbf{x}_k)\,\mathbf{p}. \tag{35.16}$$

If $\nabla^2 f(\mathbf{x}_k)$ is positive definite, the model is a bowl with a unique minimizer, where $\nabla m_k = \mathbf{g}_k + \nabla^2 f(\mathbf{x}_k)\,\mathbf{p} = \mathbf{0}$. Solving for $\mathbf{p}$ and stepping there is **Newton's method**:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \left(\nabla^2 f(\mathbf{x}_k)\right)^{-1} \nabla f(\mathbf{x}_k). \tag{35.17}$$

No step size appears: the curvature has set it. Newton's direction belongs to the family $\mathbf{d} = -\mathbf{P}^{-1}\nabla f$ with a positive definite matrix $\mathbf{P}$, here $\nabla^2 f$. Every member of the family is a descent direction, since $\nabla f^\top \mathbf{d} = -\nabla f^\top \mathbf{P}^{-1} \nabla f < 0$. Newton's direction is the steepest one when lengths are measured in the Hessian's norm instead of the Euclidean one (exercise 1).

On the quadratic $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ the model is the function, so one step lands on the minimizer, $\mathbf{x}_1 = \mathbf{x}_0 - \mathbf{A}^{-1}\mathbf{A}\mathbf{x}_0 = \mathbf{0}$, for every κ. The reason goes deeper than the algebra. Newton's method is **affine-invariant**: it does not depend on the coordinates. Under a change of coordinates $\mathbf{x} = \mathbf{T}\mathbf{y}$, Newton's iterates for $f(\mathbf{T}\mathbf{y})$ are the images of Newton's iterates for f. Gradient descent's iterates, in contrast, change under every $\mathbf{T}$ that is not orthogonal; a multiple of an orthogonal matrix changes only the effective step (exercise 1). Since a change of coordinates can make any positive definite quadratic round, the condition number cannot matter to Newton. A long valley is, to Newton's method, a round bowl drawn on stretched paper.

35.2.1.2 Quadratic Convergence

On a function that is not quadratic the model is only an approximation, and one step no longer suffices. What survives is the speed. Near a minimizer Newton's method converges much faster than geometrically, and the proof is a single use of the fundamental theorem of calculus. We write $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}^\star$ for the error.

Proposition (quadratic convergence). *Let $\nabla f(\mathbf{x}^\star) = \mathbf{0}$, and suppose that on a ball around $\mathbf{x}^\star$ the Hessian satisfies $\nabla^2 f(\mathbf{x}) \succeq \mu \mathbf{I}$ and $\|\nabla^2 f(\mathbf{x}) - \nabla^2 f(\mathbf{y})\| \le L_2 \|\mathbf{x} - \mathbf{y}\|$. If $\mathbf{x}_k$ lies in the ball, then*

$$\|\mathbf{e}_{k+1}\| \;\le\; \frac{L_2}{2\mu}\,\|\mathbf{e}_k\|^2. \tag{35.18}$$

Proof. Write $\nabla^2 f_k = \nabla^2 f(\mathbf{x}_k)$. Since $\nabla f(\mathbf{x}^\star) = \mathbf{0}$, the Newton step gives $\mathbf{e}_{k+1} = \mathbf{e}_k - (\nabla^2 f_k)^{-1}\mathbf{g}_k = (\nabla^2 f_k)^{-1}\big[\nabla^2 f_k\,\mathbf{e}_k - (\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star))\big]$. By the fundamental theorem of calculus along the segment from $\mathbf{x}^\star$ to $\mathbf{x}_k$, the gradient difference is $\int_0^1 \nabla^2 f(\mathbf{x}^\star + t\mathbf{e}_k)\,\mathbf{e}_k\,dt$, so the bracket is $\int_0^1 \big[\nabla^2 f_k - \nabla^2 f(\mathbf{x}^\star + t\mathbf{e}_k)\big]\mathbf{e}_k\,dt$. The two points differ by $(1-t)\mathbf{e}_k$, so the bracket has norm at most $\int_0^1 L_2(1-t)\|\mathbf{e}_k\|^2\,dt = \tfrac{L_2}{2}\|\mathbf{e}_k\|^2$. Finally $\|(\nabla^2 f_k)^{-1}\| \le 1/\mu$. ∎

Multiplying both sides by $L_2/(2\mu)$ gives $\frac{L_2}{2\mu}\|\mathbf{e}_{k+1}\| \le \big(\frac{L_2}{2\mu}\|\mathbf{e}_k\|\big)^2$, so the scaled error squares at every step. If it starts below 1, the error shrinks and every later iterate stays in the ball. Once it is below 0.1, it goes to 10^{-2}, 10^{-4}, 10^{-8}, so the number of correct digits roughly doubles per step. Compare gradient descent on the running example, which gains one digit in about $1{,}150$ steps. The speed has a condition attached, and it is in the hypotheses: the bound says nothing until the iterate is already close.

Far from $\mathbf{x}^\star$ the full step can overshoot, even with a positive definite Hessian. The remedy is **damping**: step to $\mathbf{x}_k + \alpha_k \mathbf{p}_k$ along the Newton direction $\mathbf{p}_k = -(\nabla^2 f(\mathbf{x}_k))^{-1}\mathbf{g}_k$, with $\alpha_k \le 1$ chosen by backtracking on the Armijo test $f(\mathbf{x}_k + \alpha \mathbf{p}_k) \le f(\mathbf{x}_k) + c\,\alpha\,\mathbf{g}_k^\top \mathbf{p}_k$, which is Equation 35.7 for a general descent direction. Near a minimizer with positive definite Hessian the full step $\alpha = 1$ passes this test when $c < \tfrac{1}{2}$, so damping costs nothing once the fast phase begins.

The test problem for the cells below is ℓ_2-regularized logistic regression on $n = 200$ random examples in $d = 5$ dimensions,

$$f(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + \exp(-y_i\,\mathbf{x}_i^\top \mathbf{w})\right) + \frac{\nu}{2}\|\mathbf{w}\|^2, \qquad \nu = 0.1,$$

with labels $y_i = \pm 1$ drawn from a logistic model (the weight-decay coefficient is ν here because λ denotes eigenvalues). The data cell also computes a reference optimum $\mathbf{w}^\star$ with $4{,}000$ gradient steps of size $\eta = 0.5$, which is safe since $L = \lambda_{\max}(\mathbf{X}^\top\mathbf{X})/(4n) + \nu$ is about 0.4 here (exercise 2).

```
rng = np.random.default_rng()
n, d, nu = 200, 5, 0.1
w_true = np.array([1.0, -2.0, 1.5, 0.0, 2.0])
X = rng.normal(size=(n, d))
y = np.where(rng.random(n) < 1 / (1 + np.exp(-X @ w_true)), 1.0, -1.0)

def loss(w, X, y, nu):
    return np.logaddexp(0, -y * (X @ w)).mean() + nu / 2 * (w @ w)

def grad_rows(w, X, y, nu):                  # row i: gradient of example i
    s = 1 / (1 + np.exp(y * (X @ w)))        # sigmoid(-y_i x_i^T w)
    return -(y * s)[:, None] * X + nu * w

w_star = np.zeros(d)
for _ in range(4000):                        # full-batch reference optimum
    w_star = w_star - 0.5 * grad_rows(w_star, X, y, nu).mean(axis=0)
```

Two predictions can now be tested. On the chapter's running example $\mathrm{diag}(1, 1000)$ one Newton step should replace the $\log 10^6 / \log(1001/999) \approx 6{,}908$ steps of gradient descent at its best fixed step (Section 35.1.3). On the logistic loss, which is not a quadratic, the exponent of the gradient norm should roughly double per iteration once the fast phase has begun.

```
A2 = np.diag([1.0, 1000.0])                  # the running example, kappa = 1000
x = np.array([1.0, 1.0])
x_new = x - np.linalg.solve(A2, A2 @ x)      # one Newton step
rate = (1000 - 1) / (1000 + 1)
print(f'Newton on the quadratic: |x_1| = {np.linalg.norm(x_new):.1e} after one step')
print(f'GD at the optimal step contracts by {rate:.4f}; it needs '
      f'{int(np.ceil(np.log(1e-6) / np.log(rate)))} steps for |x_k| <= 1e-6 |x_0|')
w = np.zeros(d)                              # Newton on the logistic loss
for it in range(6):
    s = 1 / (1 + np.exp(y * (X @ w)))
    g = grad_rows(w, X, y, nu).mean(axis=0)
    hess = (X * (s * (1 - s))[:, None]).T @ X / n + nu * np.eye(d)
    w = w - np.linalg.solve(hess, g)
    print(f'Newton iteration {it + 1}:  |grad| = {np.linalg.norm(g):.1e}')
print(f'max |w - w_star| = {np.abs(w - w_star).max():.1e}')
```

```
Newton on the quadratic: |x_1| = 0.0e+00 after one step
GD at the optimal step contracts by 0.9980; it needs 6908 steps for |x_k| <= 1e-6 |x_0|
Newton iteration 1:  |grad| = 3.5e-01
Newton iteration 2:  |grad| = 4.1e-02
Newton iteration 3:  |grad| = 1.7e-03
Newton iteration 4:  |grad| = 3.0e-06
Newton iteration 5:  |grad| = 9.9e-12
Newton iteration 6:  |grad| = 1.6e-17
max |w - w_star| = 4.4e-16
```

Both hold. On the quadratic one Newton step leaves an error of exactly zero, where gradient descent at its best step contracts by 0.998 and needs its $6{,}908$ steps. On the logistic loss the gradient norm starts at a few tenths and is below 10^{-8} by the fifth iteration, its exponent roughly doubling each time once it is small. After six iterations the iterate agrees with the reference optimum of the data cell to about 10^{-15}, the precision of the reference itself. Six linear solves have done the work of the $4{,}000$ gradient steps that produced the reference.

35.2.1.3 The Cost of a Step

Unfortunately the comparison of step counts flatters Newton's method. The Hessian has d^2 entries, and a generic solve costs $O(d^3)$ operations per step. With $d = 5$ this is nothing. A network with a modest 10^8 weights has a Hessian of 10^{16} entries, which rules the method out before the first step is taken (Section 9.2.3). The cheap alternatives are the secant approximations of the next part and the diagonal or structured preconditioners of Section 35.5.

35.2.1.4 Saddle Points

Cost is not the only objection, and it is not the most instructive one. Newton's step solves $\nabla m_k = \mathbf{0}$, which finds the stationary point of the model whether it is a minimum or not. Return to the function $f(x, y) = \frac{1}{2}x^2 + \cos y$ of Section 35.1, which has a saddle

at the origin and its minimum value -1 at $(0, \pi)$, and start at $(0.5, 0.3)$. The gradient is $(0.5, -\sin 0.3) = (0.5, -0.296)$, and the Hessian $\mathrm{diag}(1, -\cos 0.3) = \mathrm{diag}(1, -0.955)$ is indefinite: the model curves up along x and down along y. Which way will Newton's step move y from here, and which way a plain gradient step with $\eta = 1$?

```
def f_s(v):
    return 0.5 * v[0]**2 + np.cos(v[1])

def grad_s(v):
    return np.array([v[0], -np.sin(v[1])])

def hess_s(v):
    return np.diag([1.0, -np.cos(v[1])])

x_n = x_g = np.array([0.5, 0.3])
ys_n, ys_g = [], []
for _ in range(6):
    x_n = x_n - np.linalg.solve(hess_s(x_n), grad_s(x_n))    # Newton
    x_g = x_g - 1.0 * grad_s(x_g)                            # GD, eta = 1
    ys_n.append(x_n[1])
    ys_g.append(x_g[1])
print('Newton y:', np.array(ys_n[:3]), f'  f = {f_s(x_n):.4f}')
print('GD     y:', np.array(ys_g).round(4), f'  f = {f_s(x_g):.4f}')
```

```
Newton y: [-9.33624961e-03  2.71275924e-07 -6.61744490e-21]   f = 1.0000
GD     y: [0.5955 1.1565 2.0718 2.9489 3.1404 3.1416]   f = -1.0000
```

Newton's first step sets x to 0 and y to $0.3 - \tan 0.3 = -0.0093$; the next sends y to 2.7×10^{-7}. This is Newton's fast local convergence (here even cubic, since $y - \tan y \approx -y^3/3$), to the saddle $(0, 0)$ with $f = 1$. Gradient descent moves y the other way, to 0.60, 1.16, 2.07, 2.95, 3.140, 3.1416, and reaches the minimum $(0, \pi)$ with $f = -1$. The better model has led to the worse point. The cause is that the model's curvature along y is negative, so its stationary point is a saddle, and Newton's method seeks any stationary point. A trust region (Section 35.2.3) repairs this.

35.2.2 Quasi-Newton Methods

Newton's method needs second derivatives. Can the curvature be inferred instead from gradients that have already been computed? A gradient that changes quickly along a step betrays a large curvature in that direction, and a method that remembers such changes can assemble a model of the Hessian without ever evaluating one.

35.2.2.1 The Secant Method

In one dimension the idea is elementary: the slope of f' between the last two iterates estimates f'',

$$h_k = \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}, \qquad x_{k+1} = x_k - \frac{f'(x_k)}{h_k}.$$

This is the **secant method**, Newton's step with f'' replaced by the slope of the chord through

the last two gradients. Take $f(x) = e^x - 2x$, whose minimizer is $\log 2 = 0.6931$, from $x_0 = 0$ and $x_1 = 1$. The first slope is $h_1 = (e-2) - (1-2) = e - 1 = 1.718$, so $x_2 = 1 - (e-2)/1.718 = 0.582$. The errors then run 1.1×10^{-1}, 1.6×10^{-2}, 9.3×10^{-4}, 7.7×10^{-6}, 3.6×10^{-9}. Each is about half the product of the previous two (the half is $f'''/(2f'')$ at the minimizer), so the number of correct digits grows by a factor approaching 1.6 per step. That is slower than Newton's factor 2, but each step costs one gradient and no second derivative. On a quadratic $\frac{\lambda}{2}x^2$ the secant slope is λ exactly, and the step from x_1 lands on the minimizer.

In d dimensions the analogous condition is one vector equation, which cannot determine a $d \times d$ matrix: one step reveals the curvature along one direction only. Something must fill in the rest.

35.2.2.2 BFGS

A **quasi-Newton** method keeps an approximation of the inverse Hessian and updates it from the steps and gradient changes it has observed. After moving from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$, record the secant pair

$$\mathbf{s}_k = \mathbf{x}_{k+1} - \mathbf{x}_k, \qquad \mathbf{y}_k = \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k).$$

On a quadratic $\mathbf{y}_k = \mathbf{A}\mathbf{s}_k$, so the inverse Hessian maps $\mathbf{y}_k$ back to $\mathbf{s}_k$. The new approximation $\mathbf{M}_{k+1}$ is required to do the same, which is the **secant equation**

$$\mathbf{M}_{k+1}\,\mathbf{y}_k = \mathbf{s}_k. \tag{35.19}$$

BFGS (Broyden, Fletcher, Goldfarb, and Shanno) fills in the rest by changing $\mathbf{M}_k$ as little as possible: in a suitably weighted Frobenius norm, the symmetric matrix closest to $\mathbf{M}_k$ that satisfies the secant equation is

$$\mathbf{M}_{k+1} = (\mathbf{I} - \gamma_k \mathbf{s}_k \mathbf{y}_k^{\top})\,\mathbf{M}_k\,(\mathbf{I} - \gamma_k \mathbf{y}_k \mathbf{s}_k^{\top}) + \gamma_k \mathbf{s}_k \mathbf{s}_k^{\top}, \qquad \gamma_k = \frac{1}{\mathbf{y}_k^{\top}\mathbf{s}_k}. \tag{35.20}$$

The formula can be read without deriving it (we drop the index k on the pair). The right factor $\mathbf{I} - \gamma \mathbf{y}\mathbf{s}^{\top}$ sends $\mathbf{y}$ to zero, since $\gamma\,\mathbf{s}^{\top}\mathbf{y} = 1$, and leaves every vector orthogonal to $\mathbf{s}$ unchanged. So the first term forgets what $\mathbf{M}_k$ said about $\mathbf{y}$, and the second writes in the new answer: $\mathbf{M}_{k+1}\mathbf{y} = \gamma\,\mathbf{s}\,(\mathbf{s}^{\top}\mathbf{y}) = \mathbf{s}$, the secant equation. The update also keeps the matrix positive definite, so that $-\mathbf{M}_{k+1}\nabla f(\mathbf{x}_{k+1})$ is again a descent direction. For $\mathbf{z} \neq \mathbf{0}$ put $\mathbf{w} = (\mathbf{I} - \gamma \mathbf{y}\mathbf{s}^{\top})\mathbf{z}$. Then $\mathbf{z}^{\top}\mathbf{M}_{k+1}\mathbf{z} = \mathbf{w}^{\top}\mathbf{M}_k\mathbf{w} + \gamma(\mathbf{s}^{\top}\mathbf{z})^2$. If $\mathbf{M}_k \succ 0$ and $\gamma > 0$, both terms are nonnegative, and they cannot both vanish: $\mathbf{w} = \mathbf{0}$ forces $\mathbf{z} = \gamma(\mathbf{s}^{\top}\mathbf{z})\,\mathbf{y}$ with $\mathbf{s}^{\top}\mathbf{z} \neq 0$. The requirement $\gamma > 0$ is the **curvature condition** $\mathbf{y}_k^{\top}\mathbf{s}_k > 0$: the slope along the step has increased. It holds for every step on a strictly convex function, and elsewhere the line search enforces it by a second test, the Wolfe condition (Nocedal and Wright 2006).

How much curvature do the secant pairs recover? We run BFGS from $(1, 1)$ on $\mathrm{diag}(1, 10)$ with $\mathbf{M}_0 = \mathbf{I}$ and the exact line search $\alpha = -\mathbf{g}^{\top}\mathbf{p}/(\mathbf{p}^{\top}\mathbf{A}\mathbf{p})$ of a quadratic, and print the iterate and $\mathbf{M}$ after each step.

```python
def bfgs_update(M, s, y):
    gamma = 1.0 / (y @ s)
    V = np.eye(len(s)) - gamma * np.outer(s, y)
    return V @ M @ V.T + gamma * np.outer(s, s)

A = np.diag([1.0, 10.0])
x, M = np.array([1.0, 1.0]), np.eye(2)
for k in range(2):
    g = A @ x
    p = -M @ g
    alpha = -(g @ p) / (p @ A @ p)            # exact line search
    s = alpha * p
    M = bfgs_update(M, s, A @ s)              # y = A s on a quadratic
    x = x + s
    print(f'step {k + 1}: x = {x.round(4)},  M = {M.round(4).tolist()}')
print(f'A^-1 = {np.linalg.inv(A).tolist()},  '
      f'max |M_2 - A^-1| = {np.abs(M - np.linalg.inv(A)).max():.0e}')
```

```
step 1: x = [ 0.8991 -0.009 ],  M = [[1.009, -0.0001], [-0.0001, 0.1]]
step 2: x = [0. 0.],  M = [[1.0, -0.0], [-0.0, 0.1]]
A^-1 = [[1.0, 0.0], [0.0, 0.1]],  max |M_2 - A^-1| = 1e-16
```

After one step $\mathbf{M}_1 \approx \operatorname{diag}(1.009, 0.100)$, already close to $\mathbf{A}^{-1}$ in the stiff direction it has seen. After the second step $\mathbf{M}_2 = \mathbf{A}^{-1}$ to rounding error and $\mathbf{x}_2 = \mathbf{0}$. The two-dimensional case is typical. On a quadratic with exact line searches, each secant equation, once satisfied, stays satisfied, and the steps are $\mathbf{A}$-conjugate ($\mathbf{s}_i^\top \mathbf{A}\mathbf{s}_j = 0$ for $i \neq j$). After d steps $\mathbf{M}_d\mathbf{A} = \mathbf{I}$ and the iterate is the minimizer (exercise 2).

Finite termination is a statement about quadratics with exact line searches, and it should not be read as a promise. On other functions the line search is inexact, the curvature changes from step to step, and noisy gradients corrupt $\mathbf{y}_k$: the difference of two minibatch gradients is mostly the difference of two noises. Full BFGS also stores a dense $d \times d$ matrix. **L-BFGS** keeps only the m most recent secant pairs and applies, in $O(md)$ time and memory, the matrix they define, a scaled identity plus a correction of rank at most $2m$ (Liu and Nocedal 1989). It suits deterministic medium-scale problems and is less natural when fresh minibatch noise makes gradient differences unreliable.

35.2.3 Trust Regions

Newton's method failed at $(0.5, 0.3)$ because it trusted the model everywhere. The model deserved no such trust: it was fitted at one point, and it describes the function only near that point. A trust region takes this literally. It keeps the curvature information, including the negative directions, and bounds the step instead.

35.2.3.1 The Subproblem

At $(0.5, 0.3)$ the model of $\frac{1}{2}x^2 + \cos y$ has curvature -0.955 along y, so it is unbounded below and "minimize the model" has no answer. A **trust-region method** minimizes the model over a ball whose radius Δ_k says how far the model is believed:

$$\min_{\|\mathbf{p}\| \le \Delta_k} m_k(\mathbf{p}) = f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^\top \mathbf{p} + \tfrac{1}{2}\mathbf{p}^\top \mathbf{B}_k \mathbf{p}. \tag{35.21}$$

Here $\mathbf{B}_k$ is the Hessian or a symmetric approximation of it. The radius makes the subproblem bounded even when $\mathbf{B}_k$ has a negative eigenvalue, because a continuous function on a ball has a minimum. The minimizers can be characterized exactly.

Proposition (solution of the subproblem). *A step* $\mathbf{p}$ *with* $\|\mathbf{p}\| \le \Delta$ *solves Equation 35.21 if and only if there is a* $\tau \ge 0$ *with*

$$(\mathbf{B} + \tau \mathbf{I})\,\mathbf{p} = -\mathbf{g}, \qquad \mathbf{B} + \tau \mathbf{I} \succeq \mathbf{0}, \qquad \tau\,(\Delta - \|\mathbf{p}\|) = 0.$$

This is Theorem 4.1 of Nocedal and Wright (2006); exercise 3 proves that the conditions are sufficient. A trust region adds the same amount τ to every curvature, enough to make the shifted model convex. If $\mathbf{B}$ is positive definite and the Newton step fits inside the ball, $\tau = 0$ and the step is Newton's. Otherwise the step lies on the boundary, and τ is found from the scalar equation $\|(\mathbf{B}+\tau\mathbf{I})^{-1}\mathbf{g}\| = \Delta$ with $\tau > -\lambda_{\min}(\mathbf{B})$, whose left side decreases as τ grows. The equation can fail to have a root only in the degenerate *hard case*, where $\mathbf{g}$ is orthogonal to the eigenvectors of the smallest eigenvalue of $\mathbf{B}$. Part d of exercise 3 meets it. The shift τ is a Lagrange multiplier for the constraint $\tfrac{1}{2}\|\mathbf{p}\|^2 \le \tfrac{1}{2}\Delta^2$, in the sense of Section 35.6.

The radius adapts to how well the model predicted the last step. After solving the subproblem, compare the actual with the predicted decrease:

$$r_k = \frac{f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{p}_k)}{m_k(\mathbf{0}) - m_k(\mathbf{p}_k)}. \tag{35.22}$$

A ratio near one says that the model was right. The rule used below accepts the step whenever $r_k > 0.1$. Independently, it shrinks the radius, $\Delta \leftarrow \Delta/4$, if $r_k < \tfrac{1}{4}$, and doubles it if $r_k > \tfrac{3}{4}$ and the step reached the boundary.

35.2.3.2 The Trust-Region Step at a Saddle

We solve the subproblem at $(0.5, 0.3)$ by hand, with $\Delta = 0.5$. There $\mathbf{g} = (0.5, -0.296)$ and $\mathbf{B} = \mathrm{diag}(1, -0.955)$. Positive semidefiniteness of $\mathbf{B} + \tau\mathbf{I}$ needs $\tau \ge 0.955$, and the Newton point $\tau = 0$ is excluded, so the step lies on the boundary. The equations are $(1 + \tau)p_1 = -0.5$ and $(\tau - 0.955)p_2 = 0.296$, with $p_1^2 + p_2^2 = 0.25$. Their solution is $\tau \approx 1.60$ and $\mathbf{p} \approx (-0.19, 0.46)$. The new point $(0.31, 0.76)$ has $f = 0.771$, down from 1.080. The model predicted a decrease of 0.316, the actual decrease is 0.309, and $r = 0.98$, so the radius doubles. The step moves *away* from the saddle, along the direction of negative curvature, where the model falls, while the Newton step $(-0.5, -0.31)$ moves toward it.

One step by hand does not show where the rule leads. We implement the step by bisection on τ and run six iterations, printing the iterate, f, the step length, τ, the ratio, and the radius in force.

```
def tr_step(g, B, Delta):
    """Minimize g p + p B p / 2 over |p| <= Delta; return (p, tau)."""
    eig_min = np.linalg.eigvalsh(B).min()
    if eig_min > 0 and np.linalg.norm(np.linalg.solve(B, -g)) <= Delta:
        return np.linalg.solve(B, -g), 0.0     # Newton step fits: tau = 0
    lo = hi = max(0.0, -eig_min)
    hi += 1e3
```

```
    for _ in range(100):                        # bisection for |p(tau)| = Delta
        tau = (lo + hi) / 2
        p = np.linalg.solve(B + tau * np.eye(len(g)), -g)
        lo, hi = (tau, hi) if np.linalg.norm(p) > Delta else (lo, tau)
    return p, tau
```

```
x, Delta = np.array([0.5, 0.3]), 0.5
print(' k        x            f      |p|    tau      r  Delta')
for k in range(6):
    g, B = grad_s(x), hess_s(x)
    p, tau = tr_step(g, B, Delta)
    r = (f_s(x) - f_s(x + p)) / -(g @ p + 0.5 * p @ B @ p)
    print(f'{k:2d} ({x[0]:6.3f},{x[1]:6.3f}) {f_s(x):7.4f} '
          f'{np.linalg.norm(p):6.3f} {tau:6.3f} {r:6.2f} {Delta:5.2f}')
    if r > 0.1:                                 # accept the step
        x = x + p
    if r < 0.25:
        Delta /= 4                              # poor prediction: shrink
    elif r > 0.75 and np.linalg.norm(p) > 0.99 * Delta:
        Delta *= 2                              # good, and on the boundary
print(f'final x = ({x[0]:.4f}, {x[1]:.4f}),  f = {f_s(x):.4f}')
```

```
 k        x            f      |p|    tau      r  Delta
 0 ( 0.500, 0.300)  1.0803  0.500  1.596   0.98  0.50
 1 ( 0.307, 0.761)  0.7711  1.000  1.419   0.87  1.00
 2 ( 0.180, 1.753) -0.1652  2.000  0.311   0.41  2.00
 3 ( 0.043, 3.749) -0.8205  0.696  0.000   0.88  2.00
 4 ( 0.000, 3.054) -0.9962  0.088  0.000   1.00  2.00
 5 ( 0.000, 3.142) -1.0000  0.000  0.000   1.00  2.00
final x = (0.0000, 3.1416),  f = -1.0000
```

The first row reproduces the hand computation. The radius grows from 0.5 to 1 and 2, and the third step, with ratio 0.41, is accepted without changing it. The fourth iterate, $(0.04, 3.75)$, lies in the region where the Hessian is positive definite. From there on the Newton step fits inside the ball and $\tau = 0$, so the method has become Newton's method. It converges quadratically to the minimum $(0, \pi)$, where $f = -1$. Six iterations suffice. Figure 35.6 shows both runs.

The exact subproblem needs a factorization of $\mathbf{B}_k + \tau\mathbf{I}$ for each trial τ. Large problems solve it approximately with Hessian–vector products, which cost about as much as a gradient (Section 34.5). Even so, neither BFGS nor a trust region replaces stochastic gradients on a large noisy loss.

The saddle of $\frac{1}{2}x^2 + \cos y$ has now misled two methods: gradient descent stopped at it, and Newton's method ran to it. A trust region escapes, but only by testing each model against the function. On some functions no such care is needed, because they have no saddle and no second basin, and every stationary point is the minimum. Which functions these are is the subject of Section 35.3.

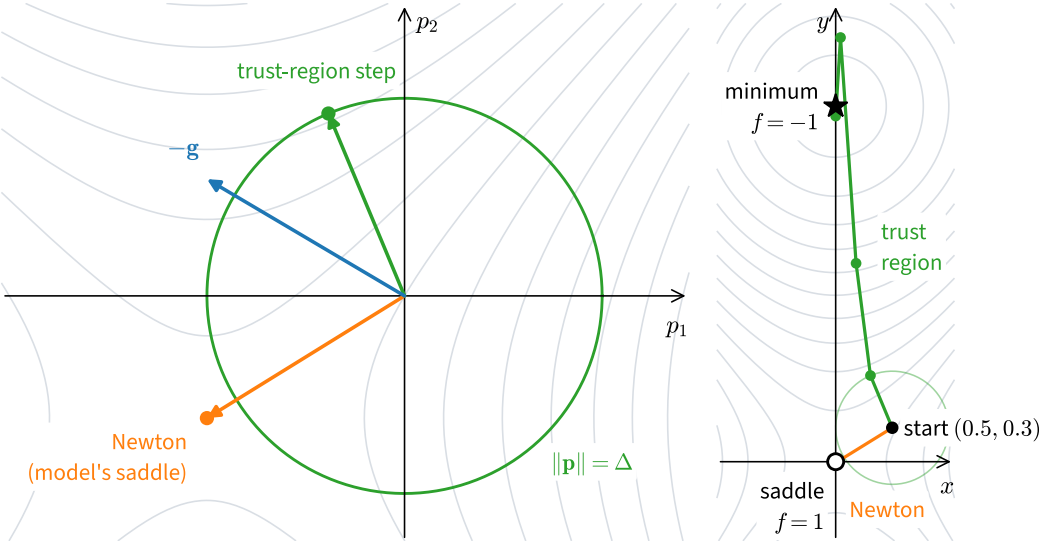


Figure 35.6: Newton and a trust region from $(0.5, 0.3)$ on $\frac{1}{2}x^2 + \cos y$, whose Hessian there is $\mathrm{diag}(1, -0.96)$. (a) The quadratic model is a saddle surface; Newton's step goes to its stationary point, while the step of length $\Delta = 0.5$ that minimizes the model moves along the negative-curvature direction, away from the saddle. (b) Newton converges quadratically to the saddle $(0, 0)$, where $f = 1$; the trust region, with its first radius drawn, reaches the minimum $(0, \pi)$, $f = -1$, in six steps, its radius growing $0.5, 1, 2$.

35.2.4 Summary

Newton's step minimizes the local quadratic model of f. It solves a quadratic in one step whatever its condition number, and near a minimizer with a positive definite, Lipschitz-continuous Hessian it squares the error at every step. A step costs $O(d^2)$ memory and $O(d^3)$ time, which no network can afford, and the method runs to saddles as readily as to minima. BFGS rebuilds the inverse Hessian from the changes of the gradient along past steps, and on a quadratic in d dimensions it has rebuilt all of it after d steps. A trust region makes any quadratic model safe to use: it bounds the step, compares the predicted decrease with the actual one, and adjusts the bound accordingly.

35.2.5 Exercises

1. **Reparametrization and preconditioning.** Let $\mathbf{T}$ be an invertible matrix and $h(\mathbf{y}) = f(\mathbf{T}\mathbf{y})$, so that $\mathbf{x} = \mathbf{T}\mathbf{y}$ is a change of coordinates.
   1. Express ∇h and $\nabla^2 h$ through the derivatives of f. Show that Newton's iterates Equation 35.17 for h, started at $\mathbf{y}_0 = \mathbf{T}^{-1}\mathbf{x}_0$, satisfy $\mathbf{y}_k = \mathbf{T}^{-1}\mathbf{x}_k$, where $\mathbf{x}_k$ are Newton's iterates for f.
   2. Run gradient descent with step η on h and map every iterate back by $\mathbf{x}_k = \mathbf{T}\mathbf{y}_k$. Show that the mapped iterates follow $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\mathbf{P}^{-1}\nabla f(\mathbf{x}_k)$ for a positive definite matrix $\mathbf{P}$, and find $\mathbf{P}$. By problem 1b this direction is the steepest one in the norm $\|\cdot\|_{\mathbf{P}}$. What is $\|\mathbf{x}\|_{\mathbf{P}}$ in the coordinates $\mathbf{y}$?
   3. For $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\,\mathbf{x}$ with $\mathbf{A} = \mathrm{diag}(1, 1000)$, the $\kappa = 1000$ quadratic of the Newton subsection, find a diagonal $\mathbf{T}$ for which gradient descent on h with $\eta = 1$

reaches the minimizer in one step. What does constructing such a $\mathbf{T}$ require for a general positive definite $\mathbf{A}$?

2. code **BFGS on a quadratic.** ⊠ Let $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}$ with $\mathbf{A}$ positive definite, and run BFGS Equation 35.20 with exact line searches from $\mathbf{M}_0 = \mathbf{I}$.
   1. Show that an exact line search along $\mathbf{p}_k$ gives $\nabla f(\mathbf{x}_{k+1})^\top \mathbf{p}_k = 0$, and that $\mathbf{y}_k = \mathbf{A}\mathbf{s}_k$.
   2. Show the hereditary property: if $\mathbf{M}_k \mathbf{y}_j = \mathbf{s}_j$ for all $j < k$ and the steps $\mathbf{s}_0, \dots, \mathbf{s}_{k-1}$, $\mathbf{s}_k$ are $\mathbf{A}$-conjugate, then $\mathbf{M}_{k+1}\mathbf{y}_j = \mathbf{s}_j$ for all $j \le k$. (Hint: $\mathbf{s}_j^\top \mathbf{y}_k = \mathbf{s}_j^\top \mathbf{A}\mathbf{s}_k = 0$ for $j < k$.) Conclude that after d linearly independent steps $\mathbf{M}_d \mathbf{A} = \mathbf{I}$.
   3. Run the BFGS loop of the text in $d = 6$ dimensions, on a randomly rotated quadratic with eigenvalues from 1 to 1000. Print $\max_{j \le k} \|\mathbf{M}_{k+1}\mathbf{y}_j - \mathbf{s}_j\|$ after each step, and confirm that it stays at rounding level and that $\mathbf{M}_6 = \mathbf{A}^{-1}$. How far from the minimizer is gradient descent at its best fixed step after the same six gradients?
   4. The symmetric rank-one update $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{(\mathbf{s} - \mathbf{M}_k\mathbf{y})(\mathbf{s} - \mathbf{M}_k\mathbf{y})^\top}{(\mathbf{s} - \mathbf{M}_k\mathbf{y})^\top \mathbf{y}}$ also satisfies the secant equation. Show by a 2×2 example that it need not stay positive definite even when $\mathbf{y}^\top \mathbf{s} > 0$, and say which of the two checks in the text fails for it.
3. code **Trust region at a saddle.** ⊠ Consider the model $m(\mathbf{p}) = \mathbf{g}^\top \mathbf{p} + \frac{1}{2}\mathbf{p}^\top \mathbf{B}\mathbf{p}$ with $\mathbf{B} = \operatorname{diag}(1, -1)$ and $\mathbf{g} = (\frac{1}{2}, -\frac{3}{10})$.
   1. Show that m is unbounded below on $\mathbb{R}^2$ and that the Newton point $-\mathbf{B}^{-1}\mathbf{g} = (-\frac{1}{2}, -\frac{3}{10})$ is a saddle of m, not a minimum.
   2. Prove that the conditions of the subproblem proposition are sufficient: if $(\mathbf{B} + \tau\mathbf{I})\mathbf{p} = -\mathbf{g}$ with $\tau \ge 0$, $\mathbf{B} + \tau\mathbf{I} \succeq \mathbf{0}$, and $\tau(\Delta - \|\mathbf{p}\|) = 0$, then every $\|\mathbf{q}\| \le \Delta$ satisfies $m(\mathbf{q}) \ge m(\mathbf{p}) + \frac{\tau}{2}(\|\mathbf{p}\|^2 - \|\mathbf{q}\|^2) \ge m(\mathbf{p})$. (Hint: expand $m(\mathbf{q}) - m(\mathbf{p})$ around $\mathbf{p}$ and use $\mathbf{g} = -(\mathbf{B} + \tau\mathbf{I})\mathbf{p}$.)
   3. For $\Delta = \frac{1}{2}$ find $\tau \ge 1$ by bisection on $\|(\mathbf{B} + \tau\mathbf{I})^{-1}\mathbf{g}\| = \Delta$ (the answer is $\tau \approx 1.65$, $\mathbf{p} \approx (-0.19, 0.46)$), and check that the step increases the coordinate with negative curvature while the Newton step decreases it.
   4. Replace $\mathbf{g}$ by $(\frac{1}{2}, 0)$. Show that $\|(\mathbf{B}+\tau\mathbf{I})^{-1}\mathbf{g}\| = \frac{1}{2}/(1+\tau) \le \frac{1}{4}$ for all $\tau > 1$, so the boundary equation has no solution with $\tau > 1$. Find the minimizers on the ball directly (they have $\tau = 1$ and $p_2 = \pm\sqrt{\Delta^2 - p_1^2}$), and explain in one sentence what the ball does with a direction of negative curvature that the gradient does not see.

35.3 Convexity

When a training run ends, we know one thing about the point where it stopped: the loss no longer decreases there. Gradient descent with a safe step reaches such a point, where the gradient is close to zero (Section 35.1), and Newton's method reaches one faster, even when it is a saddle (Section 35.2). Neither result says whether a better point exists elsewhere. This section asks when the end of a run is the best point there is, and how fast gradient descent gets there.

A one-dimensional experiment shows that the question is real. Run the same gradient-descent loop from 500 random starting points, once on the bowl $f(x) = (x - \frac{1}{2})^2$ and once on the tilted double well $\phi(x) = (x^2 - 1)^2 + x/2$. On the bowl every run ends at the same point. On the double well about half of the runs stop in a well whose value is almost

one unit above the best. Those runs have a zero gradient, which is all that a stationarity guarantee promises, and nothing in a stranded run tells us that a better point exists.

The two functions differ in one property. Every chord of the bowl lies above its graph, while the double well has chords that dip below it. Functions whose chords all lie on or above the graph are called *convex*, and for them the question has the best answer we could hope for: every local minimum is a global minimum. The proof takes three lines (Section 35.3.2.2). Behind it is a stronger fact: a tangent line of a convex function bounds it from below on its whole domain, so information at one point constrains the function everywhere. Summed along the steps of gradient descent, this bound gives a rate that holds from every start. Averaged over a random variable, it gives Jensen's inequality, which the probability and information chapters use throughout.

The section is long, and its results are not of equal weight. Three should be remembered: the tangent bound, that local minima are global, and the linear rate of gradient descent, whose proof rests on one inequality that survives without convexity. We define convex sets and functions, give three tests for convexity, derive the consequences, and certify the common losses as convex by a short calculus. Deep-network losses are not convex, and the last part says which conclusions survive. A reader in a hurry can take the chord and tangent conditions from Section 35.3.2 and go straight to Section 35.3.3. Two examples recur: the chapter's quadratic $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ for the rates, and the double well ϕ, which fails every test of convexity that we apply to it.

35.3.1 Convex Sets

A convex function is defined through the segments between points of its domain. That domain, the feasible region of a constrained problem, and the set of minimizers are all sets, so we begin with sets. We need little: the definition, three families of examples, and one rule.

35.3.1.1 Definition and Examples

A set is convex when the segment between any two of its points stays inside it. Formally, a set $C \subseteq \mathbb{R}^d$ of points with d coordinates is **convex** if

$$\theta\mathbf{x} + (1-\theta)\,\mathbf{y} \in C \qquad \text{for all } \mathbf{x}, \mathbf{y} \in C \text{ and all } \theta \in [0,1]. \tag{35.23}$$

As the weight θ runs from 1 to 0, the point $\theta\mathbf{x}+(1-\theta)\mathbf{y}$ traces the straight segment from $\mathbf{x}$ to $\mathbf{y}$. Figure 35.7 contrasts a convex set with a nonconvex crescent and shows the two convex sets that deep learning uses most, the probability simplex and a half-space.

An annulus shows what the definition excludes. The annulus $\{\mathbf{x} : 1 \le \|\mathbf{x}\| \le 2\}$ contains the points $(\pm\frac{3}{2}, 0)$, but their midpoint is the origin, which the annulus excludes. One missing midpoint is enough to violate Equation 35.23.

Three families will do for this chapter. A hyperplane $\{\mathbf{x} : \mathbf{a}^\top\mathbf{x} = b\}$ and a half-space $\{\mathbf{x} : \mathbf{a}^\top\mathbf{x} \le b\}$ are convex because $\mathbf{a}^\top\mathbf{x}$ is linear: an average of two numbers equal to b (or at most b) is again equal to b (or at most b). A norm ball $\{\mathbf{x} : \|\mathbf{x}\| \le r\}$ is convex by the triangle inequality and homogeneity, $\|\theta\mathbf{x}+(1-\theta)\mathbf{y}\| \le \theta\|\mathbf{x}\|+(1-\theta)\|\mathbf{y}\| \le r$. This covers the ℓ_2 balls of weight decay and gradient clipping and the ℓ_1 balls of sparsity-inducing regularization alike. The third family is the probability simplex $\Delta = \{\mathbf{p} \in \mathbb{R}^d : \mathbf{p} \succeq 0,\ \mathbf{1}^\top\mathbf{p} = 1\}$,

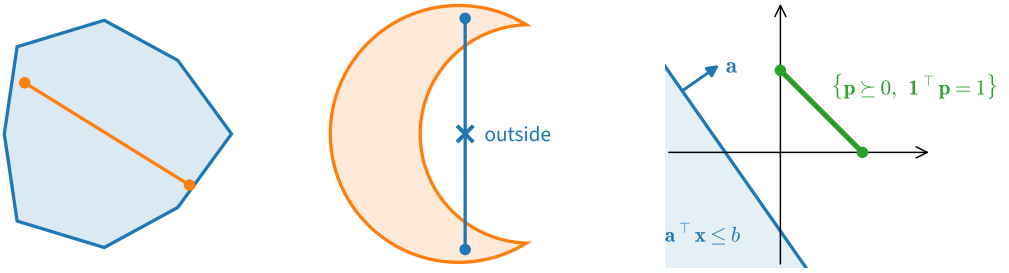


Figure 35.7: Convex versus nonconvex sets. Left: a convex set, where the segment between any two points stays inside. Middle: a nonconvex crescent, where a chord between two of its points passes outside the set. Right: the probability simplex $\{\mathbf{p} \succeq 0,\ \mathbf{1}^\top \mathbf{p} = 1\}$ and a half-space $\{\mathbf{x} : \mathbf{a}^\top \mathbf{x} \le b\}$, both convex.

where $\mathbf{p} \succeq 0$ means that every entry is nonnegative. It contains attention weights, class probabilities, and mixture weights, and it is convex by the rule that follows.

35.3.1.2 New Convex Sets from Old

The intersection of convex sets, finitely or infinitely many, is convex: the segment between two points of the intersection lies in each of the sets, hence in all of them. A convex set has no dents, and cutting a dent-free set with another dent-free set cannot create one. Unions have no such guarantee: $[0, 1] \cup [2, 3]$ is a union of two convex intervals whose chord from 1 to 2 leaves the set.

The simplex is the intersection of one hyperplane ($\mathbf{1}^\top \mathbf{p} = 1$) with d half-spaces ($-p_i \le 0$), hence convex. The symmetric positive semidefinite (PSD) matrices form a convex set for the same reason, as an intersection of infinitely many half-spaces: for each fixed $\mathbf{z}$, the condition $\mathbf{z}^\top \mathbf{A}\mathbf{z} \ge 0$ is linear in the entries of $\mathbf{A}$.

The opposite construction starts from points. The **convex hull** of a set S is the set of all weighted averages $\sum_i \theta_i \mathbf{x}_i$ of points $\mathbf{x}_i \in S$ with $\theta_i \ge 0$ and $\sum_i \theta_i = 1$. It is convex, and it is the smallest convex set that contains S (Exercise 2). The simplex is the convex hull of the coordinate vectors $\mathbf{e}_1, \dots, \mathbf{e}_d$: every probability distribution is an average of certainties.

35.3.2 Convex Functions

A convex function is one whose graph has no dents. We give three equivalent tests for this property, which need zero, one, and two derivatives respectively, and then two refinements: strong convexity, which controls the speed of gradient descent, and subgradients, which handle corners. If only one of the three tests is to be remembered, it should be the second, the tangent bound.

35.3.2.1 The Chord Condition

The first test needs no derivatives, so it applies to the absolute value and the hinge loss as well as to smooth losses. A function $f : C \to \mathbb{R}$ on a convex domain C is **convex** if its graph lies on or below every chord:

$$f\left(\theta \mathbf{x} + (1-\theta)\,\mathbf{y}\right) \;\le\; \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y}) \qquad \text{for all } \mathbf{x}, \mathbf{y} \in C,\ \theta \in [0,1]. \tag{35.24}$$

The left side evaluates f at a point of the segment; the right side is the height of the chord joining $(\mathbf{x}, f(\mathbf{x}))$ to $(\mathbf{y}, f(\mathbf{y}))$ above that same point. For $f(x) = x^2$ with $x = 0$, $y = 2$, and $\theta = \frac{1}{2}$, the graph gives $f(1) = 1$ and the chord gives $\frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 4 = 2$, and indeed $1 \le 2$. For general x, y, and θ, expanding the squares gives the gap

$$\theta x^2 + (1-\theta) y^2 - \left(\theta x + (1-\theta) y\right)^2 = \theta(1-\theta)(x-y)^2 \;\ge\; 0,$$

so the square is convex, with a gap that grows with the distance between the two points. This gap will return as the variance in Jensen's inequality.

If the inequality Equation 35.24 is strict whenever $\mathbf{x} \neq \mathbf{y}$ and $\theta \in (0,1)$, f is **strictly convex**; if $-f$ is convex, f is **concave**. The linear function $f(x) = x$ is convex but not strictly convex: its chords lie on its graph. The domain must be convex for the definition to parse, since the point $\theta\mathbf{x} + (1-\theta)\mathbf{y}$ has to be one where f is defined.

Convex functions and convex sets are linked by the **epigraph** $\{(\mathbf{x}, t) : t \ge f(\mathbf{x})\}$, the region on and above the graph. A function f is convex if and only if its epigraph is a convex set, because chords between points of the epigraph stay above the graph if and only if Equation 35.24 holds. The link also runs from functions to sets. Every sublevel set $\{\mathbf{x} \in C : f(\mathbf{x}) \le c\}$ of a convex f is convex: if $f(\mathbf{x}) \le c$ and $f(\mathbf{y}) \le c$, the chord inequality gives $f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \le \theta c + (1-\theta)c = c$. This is why a constraint $g(\mathbf{x}) \le 0$ with a convex g always describes a convex feasible region.

The converse fails. The function $\sqrt{|x|}$ has intervals $\{|x| \le c^2\}$ as sublevel sets, yet with $x = 0$, $y = 1$, and $\theta = \frac{1}{2}$ the graph gives $\sqrt{1/2} \approx 0.71$ and the chord gives 0.5, so the chord condition fails.

35.3.2.2 Local Minima Are Global

The chord condition already answers the question of the opening. Figure 35.8 draws the two situations: one basin on a convex objective, two on the double well.

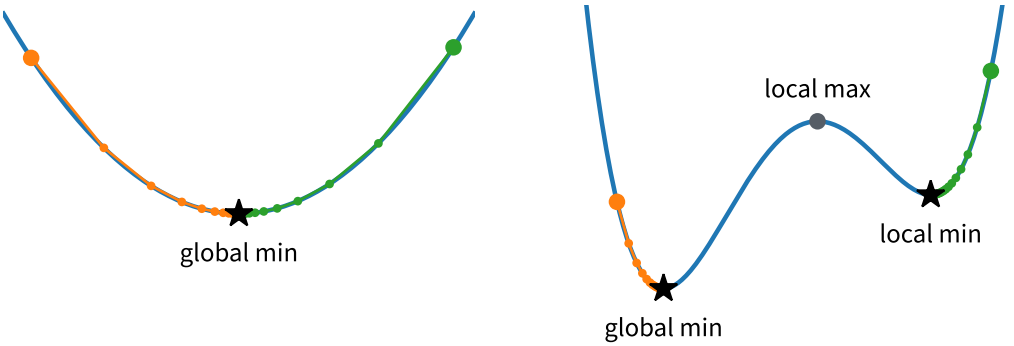


Figure 35.8: (a) On a convex objective, gradient descent with a small fixed step reaches the global minimizer from either start. (b) On a double well, the same update from different starts reaches different minima; the basins meet at the local maximum.

Proposition (local minima are global). *If f is convex on a convex set C, every local minimizer of f on C is a global minimizer.*

Proof. Let $\mathbf{x}^\star$ be a local minimizer and take any $\mathbf{y} \in C$. For small $\theta > 0$ the point $\mathbf{z}_\theta = (1-\theta)\mathbf{x}^\star + \theta\mathbf{y}$ on the chord toward $\mathbf{y}$ lies in the neighborhood where $\mathbf{x}^\star$ is minimal, so local minimality and the chord inequality give

$$f(\mathbf{x}^\star) \;\le\; f(\mathbf{z}_\theta) \;\le\; (1-\theta)f(\mathbf{x}^\star) + \theta f(\mathbf{y}).$$

Subtract $(1-\theta)f(\mathbf{x}^\star)$ and divide by θ: $f(\mathbf{x}^\star) \le f(\mathbf{y})$. ■

In words: if a better point $\mathbf{y}$ existed anywhere, the chord from $\mathbf{x}^\star$ toward it would run below the level $f(\mathbf{x}^\star)$ arbitrarily close to $\mathbf{x}^\star$, and $\mathbf{x}^\star$ would not be a local minimizer.

We now run the experiment of the opening, and with the proposition in hand we can say in advance what the bowl will do, and compute what the double well will do. The cell applies the same loop, with step 0.05 and 600 steps, to the bowl $(x - \frac{1}{2})^2$ and to the double well ϕ, from the same 500 starts drawn uniformly on $[-1.5, 1.5]$. With a step this small a run moves downhill without jumping over the hump, so it ends in the well on its own side of the local maximum of ϕ. The cell finds that maximum as a root of $\phi'(x) = 4x^3 - 4x + \frac{1}{2}$ and compares the share of starts on its right with the share of runs that end in the right well.

```
def gd(grad_fn, x, eta=0.05, steps=600):
    for _ in range(steps):
        x = x - eta * grad_fn(x)
    return x

phi = lambda x: (x**2 - 1)**2 + 0.5 * x        # tilted double well
dphi = lambda x: 4 * x * (x**2 - 1) + 0.5      # its derivative
rng = np.random.default_rng()
inits = rng.uniform(-1.5, 1.5, size=500)       # 500 random starting points
bowl = gd(lambda x: 2 * (x - 0.5), inits)      # convex: (x - 1/2)^2
well = gd(dphi, inits)
left, right = well[well < 0], well[well > 0]
print(f'bowl: all {bowl.size} runs end at x = {bowl.mean():.6f}'
      f' (spread {np.ptp(bowl):.0e})')
for name, w in (('left ', left), ('right', right)):
    print(f'well, {name}: {w.size} runs, x = {w.mean():.4f}, phi = {phi(w.mean()):.4f}')
x_max = np.sort(np.roots([4, 0, -4, 0.5]).real)[1]   # local max of phi
print(f'local max at {x_max:.4f}: predicted share right {(1.5 - x_max) / 3:.3f},'
      f' observed {right.size / inits.size:.3f}')
```

```
bowl: all 500 runs end at x = 0.500000 (spread 7e-16)
well, left : 263 runs, x = -1.0575, phi = -0.5148
well, right: 237 runs, x = 0.9304, phi = 0.4833
local max at 0.1271: predicted share right 0.458, observed 0.474
```

In every run the bowl's 500 endpoints agree to rounding. The double well splits the runs into two groups, at $x \approx -1.0575$ with $\phi \approx -0.515$ and at $x \approx 0.9304$ with $\phi \approx 0.483$, a gap of almost one unit. The local maximum that separates them is at $x \approx 0.127$, so a fraction $(1.5 - 0.127)/3 \approx 0.458$ of uniform starts goes right: a little under half. From draw to draw the observed share scatters around this prediction by a few percentage points, the sampling error of 500 starts.

The two runs use the same loop and differ only in the objective. Convexity rules out the second basin. It does not by itself carry the loop to the minimizer: that still takes smoothness and a safe step, as in Section 35.1.1.

35.3.2.3 The First-Order Condition

The chord condition needs no derivatives, but optimization proofs work with gradients. When f is differentiable, convexity has an equivalent first-order form: the first-order Taylor approximation at any point is a *global under-estimator*,

$$f(\mathbf{y}) \;\geq\; f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) \qquad \text{for all } \mathbf{x}, \mathbf{y} \in C. \tag{35.25}$$

For $f(x) = x^2$ at $x = 1$ the tangent line is $1 + 2(y-1) = 2y - 1$, and $y^2 - (2y-1) = (y-1)^2 \geq 0$ for every y. In words: one gradient, evaluated at one point $\mathbf{x}$, gives a lower bound on f at every other point of the domain. This is the property every proof below uses. Figure 35.9 shows the two conditions side by side: chords lie above the graph, tangents lie below it.

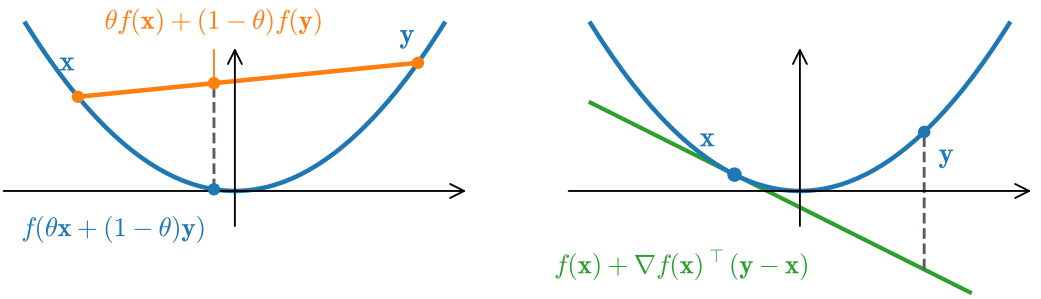


Figure 35.9: Two equivalent conditions. (a) The chord joining two points of the graph lies above the graph: $f(\theta\mathbf{x} + (1-\theta)\mathbf{y}) \leq \theta f(\mathbf{x}) + (1-\theta) f(\mathbf{y})$. (b) The tangent at any point lies below the graph: $f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$.

35.3.2.4 The Second-Order Condition

For smooth losses the easiest test is usually a matrix test. When f is twice differentiable on an **open** convex domain C, f is convex if and only if

$$\nabla^2 f(\mathbf{x}) \succeq 0 \qquad \text{for all } \mathbf{x} \in C,$$

where $\succeq 0$, applied to a symmetric matrix, says that it is PSD.

In one dimension this is the calculus criterion $f''(x) \geq 0$: the graph curves upward. In d dimensions it says that every one-dimensional slice curves upward, since $\mathbf{v}^\top \nabla^2 f(\mathbf{x})\, \mathbf{v}$ is the second derivative of f along the line through $\mathbf{x}$ in direction $\mathbf{v}$ (Section 34.2).

The test settles three common cases. The quadratic $\frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x} + \mathbf{b}^\top \mathbf{x}$ with $\mathbf{A}$ symmetric has the constant Hessian $\mathbf{A}$, so it is convex if and only if $\mathbf{A} \succeq 0$. The least-squares loss $\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ of a design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$ (n examples, d weights) has Hessian $\mathbf{X}^\top \mathbf{X}$, and $\mathbf{v}^\top \mathbf{X}^\top \mathbf{X} \mathbf{v} = \|\mathbf{X}\mathbf{v}\|^2 \geq 0$, so least squares is always convex (Section 33.3.2.2). And e^x, $-\log x$,

and $x \log x$ are convex on their domains because their second derivatives e^x, $1/x^2$, and $1/x$ are positive.

The domain is open for a reason: on a lower-dimensional domain only directions inside the domain count, and $x^2 - y^2$ is convex on the line $y = 0$ although its Hessian is indefinite.

The three tests describe the same functions. Of the four short parts of the proof, the second is the one to read: applying the tangent bound at the *intermediate* point of the chord makes the gradient terms cancel, a choice of point that returns in the proof of Jensen's inequality.

Proposition (the three characterizations agree). *Let f be differentiable on an open convex domain C. Then the chord condition Equation 35.24 and the first-order condition Equation 35.25 are equivalent. If f is twice differentiable, both are equivalent to $\nabla^2 f(\mathbf{x}) \succeq 0$ on C.*

Proof. *(Chord $\Rightarrow$ first-order.)* Fix $\mathbf{x}, \mathbf{y}$ and $\theta \in (0, 1]$. Write the chord inequality at the point $\mathbf{x} + \theta(\mathbf{y} - \mathbf{x})$, subtract $f(\mathbf{x})$, and divide by θ:

$$\frac{f(\mathbf{x} + \theta(\mathbf{y} - \mathbf{x})) - f(\mathbf{x})}{\theta} \;\le\; f(\mathbf{y}) - f(\mathbf{x}).$$

As $\theta \downarrow 0$ the left side converges to the directional derivative $\nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$, which gives Equation 35.25.

(First-order $\Rightarrow$ chord.) Let $\mathbf{z} = \theta \mathbf{x} + (1 - \theta)\mathbf{y}$ and apply the under-estimator at $\mathbf{z}$ twice, once toward each endpoint:

$$f(\mathbf{x}) \ge f(\mathbf{z}) + \nabla f(\mathbf{z})^\top (\mathbf{x} - \mathbf{z}), \qquad f(\mathbf{y}) \ge f(\mathbf{z}) + \nabla f(\mathbf{z})^\top (\mathbf{y} - \mathbf{z}).$$

Multiply by θ and $1-\theta$ and add. The gradient terms combine to $\nabla f(\mathbf{z})^\top (\theta \mathbf{x} + (1-\theta)\mathbf{y} - \mathbf{z}) = 0$, which leaves Equation 35.24.

(Second-order $\Rightarrow$ first-order.) Restrict f to the segment: $g(t) = f(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))$ has $g''(t) = (\mathbf{y} - \mathbf{x})^\top \nabla^2 f(\mathbf{x} + t(\mathbf{y} - \mathbf{x}))\,(\mathbf{y} - \mathbf{x}) \ge 0$. Taylor's theorem with the Lagrange remainder (Section 34.1) gives $g(1) = g(0) + g'(0) + \tfrac{1}{2} g''(\tau)$ for some $\tau \in (0, 1)$, and dropping the nonnegative remainder is Equation 35.25.

(First-order $\Rightarrow$ second-order.) Suppose instead that $\mathbf{v}^\top \nabla^2 f(\mathbf{x})\, \mathbf{v} < 0$ for some $\mathbf{x}$ and $\mathbf{v}$. Expanding along that direction,

$$f(\mathbf{x} + t\mathbf{v}) = f(\mathbf{x}) + t\, \nabla f(\mathbf{x})^\top \mathbf{v} + \frac{t^2}{2}\, \mathbf{v}^\top \nabla^2 f(\mathbf{x})\, \mathbf{v} + o(t^2),$$

so for small enough $t > 0$ the left side drops strictly below the tangent value $f(\mathbf{x}) + t \nabla f(\mathbf{x})^\top \mathbf{v}$, contradicting Equation 35.25. ■

35.3.2.5 A Numerical Check of the Three Tests

We run the three tests on two losses: the least-squares loss $f(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$ with $n = 20$ random examples and $d = 2$ weights, and the double well ϕ of the opening. For least squares the outcome is known in advance, and more than its sign: expanding the squares as we did for x^2 gives the chord gap, graph minus chord, as $-\tfrac{1}{2}\theta(1-\theta)\|\mathbf{X}(\mathbf{u} - \mathbf{v})\|^2$, and the tangent gap, tangent minus graph, as $-\tfrac{1}{2}\|\mathbf{X}(\mathbf{v} - \mathbf{u})\|^2$. For each loss the cell samples

a thousand chords and a thousand tangents between random points of $[-1.5, 1.5]^d$, prints the largest gap of each kind, and evaluates the Hessian at the sampled points.

```
X, y = rng.normal(size=(20, 2)), rng.normal(size=20)  # least squares, d = 2
losses = {'least squares': (2, lambda w: 0.5 * ((X @ w - y) ** 2).sum(),
                            lambda w: X.T @ (X @ w - y), lambda w: X.T @ X),
          'double well': (1, lambda w: phi(w[0]), dphi,  # phi'' = 12 x^2 - 4
                          lambda w: np.diag(12 * w**2 - 4))}
print('               worst chord gap  worst tangent gap  smallest Hessian eig')
for name, (d, f, grad, hess) in losses.items():
    U, V = rng.uniform(-1.5, 1.5, size=(2, 1000, d))  # 1000 random pairs
    T = rng.uniform(size=1000)
    chord = [f(t * u + (1 - t) * v) - (t * f(u) + (1 - t) * f(v))
             for u, v, t in zip(U, V, T)]             # graph minus chord
    tangent = [f(u) + grad(u) @ (v - u) - f(v)        # tangent minus graph
               for u, v in zip(U, V)]
    eig = min(np.linalg.eigvalsh(hess(u))[0] for u in U)
    print(f'{name:15s}{max(chord):15.3f}{max(tangent):19.3f}{eig:22.3f}')
i = int(np.argmax(chord))                             # the double well's worst
print(f'worst chord of the double well: from {U[i, 0]:.2f} to {V[i, 0]:.2f}')
```

```
               worst chord gap  worst tangent gap  smallest Hessian eig
least squares           -0.000             -0.003                24.375
double well              0.972              2.997                -4.000
worst chord of the double well: from -0.96 to 1.10
```

For least squares no sampled chord runs below the graph and no tangent above it, and the Hessian has positive eigenvalues, as the formulas require. The double well fails all three tests. Its worst chord runs below the graph by almost one unit, between points in the two wells, its tangents cut the graph, and its second derivative falls to nearly -4 near the hump, where $\phi''(0) = -4$. The three computations do not carry the same weight, though. Sampling can refute convexity, as it did for ϕ with one bad chord, but it cannot prove it: for least squares it only failed to find a counterexample. The Hessian check proves it there, since $\mathbf{X}^\top\mathbf{X}$ is the same PSD matrix at every $\mathbf{w}$.

35.3.2.6 Strict and Strong Convexity

Convexity alone allows flat stretches, and on a flat stretch gradient descent can be arbitrarily slow. A guaranteed speed needs curvature that is bounded away from zero. Three functions show the hierarchy, drawn in Figure 35.10(a). The line $f(x) = x$ is convex but not strictly convex, since its chords lie on its graph. The quartic x^4 is strictly convex (its second derivative $12x^2$ vanishes only at the single point 0, so its slope strictly increases), yet $f''(0) = 0$: it is flat at its minimizer. The square x^2 has $f'' \equiv 2$ and curves equally everywhere.

Strong convexity states that bound. For $\mu > 0$, f is **μ-strongly convex** if $f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$ is convex. Applying the first-order condition to $h(\mathbf{x}) = f(\mathbf{x}) - \frac{\mu}{2}\|\mathbf{x}\|^2$, whose gradient is $\nabla f(\mathbf{x}) - \mu\mathbf{x}$, and using $\|\mathbf{y}\|^2 - \|\mathbf{x}\|^2 - 2\mathbf{x}^\top(\mathbf{y}-\mathbf{x}) = \|\mathbf{y}-\mathbf{x}\|^2$, gives the equivalent form for differentiable f:

$$f(\mathbf{y}) \;\geq\; f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{\mu}{2}\,\|\mathbf{y}-\mathbf{x}\|^2 \qquad \text{for all } \mathbf{x}, \mathbf{y}. \tag{35.26}$$

So the graph lies above a parabola of curvature μ that touches it at $\mathbf{x}$, not only above the tangent line. Applying the second-order condition to h gives the third form: for twice-differentiable f, strong convexity means $\nabla^2 f(\mathbf{x}) \succeq \mu \mathbf{I}$ everywhere, where $\mathbf{I}$ is the identity. The square x^2 is 2-strongly convex; the quartic x^4 is strongly convex on no neighborhood of 0, because $f''(0) = 0 < \mu$ for every $\mu > 0$. Adding the ridge penalty $\frac{\lambda}{2}\|\mathbf{w}\|^2$ to any convex loss makes the sum λ-strongly convex, since subtracting $\frac{\lambda}{2}\|\mathbf{w}\|^2$ returns the convex loss. This is one more effect of weight decay.

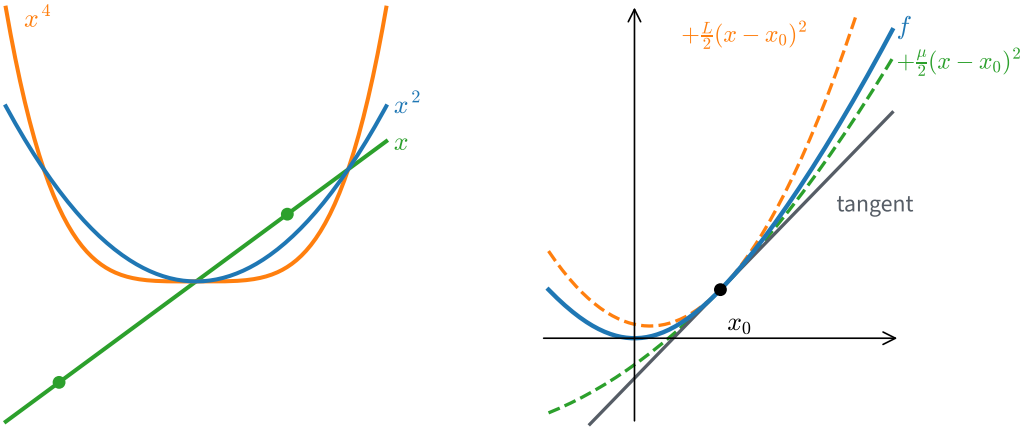


Figure 35.10: (a) Convex, strictly convex, strongly convex: x has chords on its graph, x^4 is flat at the origin, x^2 curves everywhere. (b) A μ-strongly convex, L-smooth function, here $f(x) = \frac{1}{2}x^2 + 3\log\cosh x$, lies between two parabolas tangent at $x_0 = 1$, with curvatures $\mu = 1$ and $L = 4$; their ratio is κ.

Strong convexity pairs with smoothness. Section 35.1.1 called f L-**smooth** when its gradient is L-Lipschitz, $\|\nabla f(\mathbf{x}) - \nabla f(\mathbf{y})\| \le L\|\mathbf{x} - \mathbf{y}\|$; for a convex twice-differentiable f this means $\nabla^2 f \preceq L\mathbf{I}$. Smoothness gives the quadratic ceiling $f(\mathbf{y}) \le f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$ of Section 35.1.1, and strong convexity gives the floor Equation 35.26. Together they sandwich the graph between two parabolas tangent at $\mathbf{x}$, with curvatures μ and L, as in Figure 35.10(b). The function in that panel, $f(x) = \frac{1}{2}x^2 + 3\log\cosh x$, has $f''(x) = 1 + 3/\cosh^2 x \in [1, 4]$, so $\mu = 1$ and $L = 4$.

The ratio $\kappa = L/\mu$ is the **condition number**, which Section 35.1.2 met as the ratio of the extreme eigenvalues of a quadratic. On the chapter's running example $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$, $\mu = 1$, $L = 1000$, and $\kappa = 10^3$.

35.3.2.7 Subgradients

The losses $\|\mathbf{x}\|_1$ and $\max(0, 1 - z)$ have corners, so Equation 35.25 cannot be checked as written: there is no gradient at the kink. What survives at a kink is the job that the gradient did in the tangent bound, and we take that job as the definition. A vector $\mathbf{g}$ is a **subgradient** of f at $\mathbf{x}$ if it gives the global under-estimate a gradient would:

$$f(\mathbf{y}) \;\ge\; f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{y} - \mathbf{x}) \qquad \text{for all } \mathbf{y}. \tag{35.27}$$

The set of all subgradients at $\mathbf{x}$ is the **subdifferential** $\partial f(\mathbf{x})$. Where a convex f is differentiable it is the single gradient, $\{\nabla f(\mathbf{x})\}$. At a corner it can contain many vectors (Figure 35.11). For $|x|$ at 0 every slope $g \in [-1, 1]$ qualifies, since $|y| \geq gy$ for all y, and for the hinge $\max(0, 1 - z)$ at $z = 1$ the subdifferential is $\partial f(1) = [-1, 0]$.

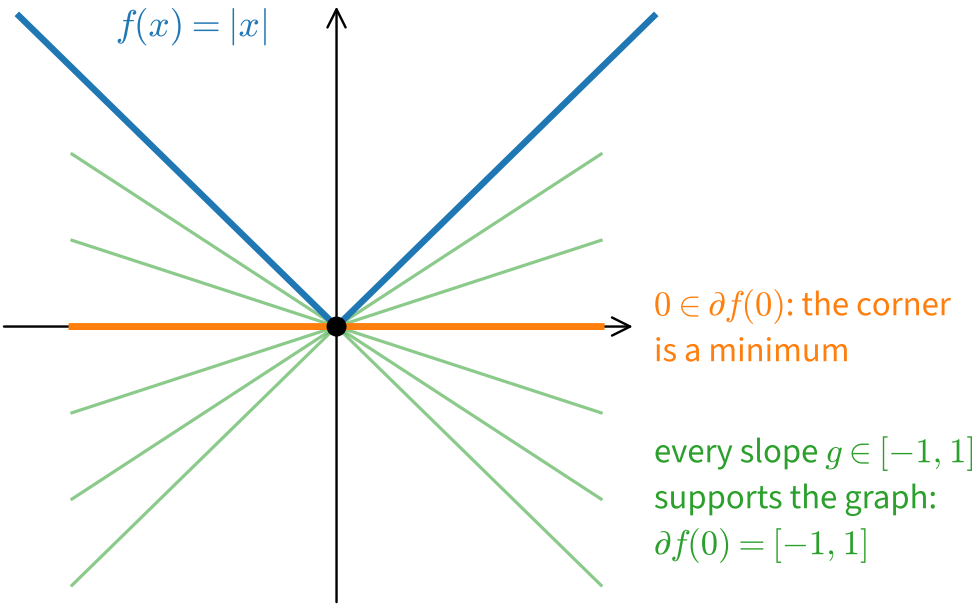


Figure 35.11: At the kink of $f(x) = |x|$ the gradient does not exist, but the subdifferential is a set: every slope $g \in (-1, 1)$ defines a supporting line below the V, and the two extreme slopes ± 1 lie along the branches themselves, so $\partial f(0)$ is the whole interval from -1 to 1. The zero-slope member (orange) is the optimality certificate $0 \in \partial f(0)$: the corner is a minimum certified without differentiability.

The optimality test needs no derivatives: $\mathbf{x}^\star$ minimizes f if and only if $\mathbf{0} \in \partial f(\mathbf{x}^\star)$. With $\mathbf{g} = \mathbf{0}$ the definition reads $f(\mathbf{y}) \geq f(\mathbf{x}^\star)$ for all $\mathbf{y}$, which is the statement that $\mathbf{x}^\star$ is a minimizer. The condition and the conclusion are the same inequality, so each implies the other.

Does a convex function always have a subgradient? At interior points of its domain it does, and Jensen's inequality rests on this. In one dimension the reason is short. For three points $a < b < c$ of an open interval on which f is convex, the chord inequality at b rearranges to

$$\frac{f(b) - f(a)}{b - a} \leq \frac{f(c) - f(a)}{c - a} \leq \frac{f(c) - f(b)}{c - b},$$

so slopes of chords increase from left to right. The difference quotient $q(x) = (f(x) - f(x_0))/(x - x_0)$ at a point x_0 of the interval is therefore nondecreasing, and its one-sided limits at x_0, the left and right derivatives $f'_-(x_0) \leq f'_+(x_0)$, exist and are finite. Every g between them is a subgradient: for $y > x_0$ we have $q(y) \geq f'_+(x_0) \geq g$, which is $f(y) \geq f(x_0) + g(y - x_0)$ after multiplying by $y - x_0$, and the case $y < x_0$ is symmetric.

In d dimensions the same holds at every interior point of the domain: a convex f has at least one subgradient there, so a supporting hyperplane fits under its graph (Rockafellar (1970);

Boyd and Vandenberghe (2004), section 2.5). We use this fact without proof. As a search direction a subgradient is weaker than a gradient: a subgradient step need not decrease f (at the minimum of $|x|$, the subgradient $\frac{1}{2}$ moves $x = 0$ to $-\eta/2$), and the subgradient method needs diminishing steps, with an error that falls only like $1/\sqrt{k}$.

35.3.3 Consequences of Convexity

Convexity has three consequences that the rest of the book uses, and all come from one source, a line or plane that bounds the function from below everywhere. Averaged over a random variable, the bound gives Jensen's inequality. At a stationary point it certifies the global minimum, which sharpens the answer that Section 35.3.2.2 drew from the chord condition alone. Summed along the steps of gradient descent, it gives rates. Jensen's inequality is independent of the other two, and a reader interested only in optimization can skip it.

35.3.3.1 Jensen's Inequality

Averages of losses, expected log-likelihoods, and KL divergences all compare a function of an average with the average of a function. The smallest case settles which is larger. Let $Z \in \{0, 2\}$ with probability $\frac{1}{2}$ each, and $f(x) = x^2$. Then $f(\mathbb{E}[Z]) = f(1) = 1$, while $\mathbb{E}[f(Z)] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 4 = 2$. The gap 1 is the variance of Z, and for $f = x^2$ the gap is always the variance, $\mathbb{E}[Z^2] - (\mathbb{E}[Z])^2$. The chord inequality of Figure 35.9(a) makes this comparison for a two-point distribution with weights θ and $1-\theta$. The gap $\theta(1-\theta)(x-y)^2$ computed for x^2 is that distribution's variance.

Nothing in the chord argument is special to two points. Applying Equation 35.24 repeatedly (Exercise 2) gives the finite form

$$f\left(\sum_i \theta_i \mathbf{x}_i\right) \;\le\; \sum_i \theta_i \, f(\mathbf{x}_i), \qquad \theta_i \ge 0, \; \sum_i \theta_i = 1, \tag{35.28}$$

and a set of nonnegative weights summing to one is a probability distribution on the points $\mathbf{x}_i$. The general form replaces the weighted average by an expectation.

Proposition (Jensen's inequality). *Let f be convex on a convex set $C \subseteq \mathbb{R}^d$, and let $\mathbf{Z}$ be a random vector with values in C, with $\mathbb{E}\|\mathbf{Z}\| < \infty$ and mean $\mathbf{m} = \mathbb{E}[\mathbf{Z}]$ in the interior of C. Then*

$$f\left(\mathbf{m}\right) \;\le\; \mathbb{E}\left[f(\mathbf{Z})\right]. \tag{35.29}$$

If f is strictly convex, equality holds if and only if $\mathbf{Z} = \mathbf{m}$ with probability one.

Proof. Pick a subgradient $\mathbf{g} \in \partial f(\mathbf{m})$, which exists because $\mathbf{m}$ is an interior point. The key step is that the supporting hyperplane at $\mathbf{m}$ holds for every outcome at once: by Equation 35.27,

$$f(\mathbf{Z}) \;\ge\; f(\mathbf{m}) + \mathbf{g}^{\top}(\mathbf{Z} - \mathbf{m}).$$

Take expectations of both sides. The linear term has mean zero, which leaves $\mathbb{E}[f(\mathbf{Z})] \ge f(\mathbf{m})$. For strictly convex f the supporting hyperplane touches the graph only at $\mathbf{m}$. Suppose it also touched at some $\mathbf{x} \neq \mathbf{m}$. Strict convexity would then give $f(\frac{\mathbf{x}+\mathbf{m}}{2}) < \frac{1}{2}f(\mathbf{x}) + \frac{1}{2}f(\mathbf{m})$,

where the right side is the value of the affine hyperplane at the midpoint; this contradicts that the hyperplane lies below f. So the pointwise inequality is strict on the event $\mathbf{Z} \neq \mathbf{m}$, and equality in expectation forces that event to have probability zero. ■

The same pointwise bound shows that $f(\mathbf{Z})$ is bounded below by an integrable random variable, so $\mathbb{E}[f(\mathbf{Z})]$ is always defined; it may be $+\infty$, in which case the inequality holds trivially. Spreading $\mathbf{Z}$ out around a fixed mean can only raise the average of a convex f. For concave f the inequality flips: $\mathbb{E}[f(\mathbf{Z})] \leq f(\mathbb{E}[\mathbf{Z}])$.

The most used corollary is the nonnegativity of the KL divergence; Section 37.1 restates it as Gibbs' inequality.

Corollary (nonnegativity of KL divergence). *For probability distributions p and q on a common finite alphabet, with $q(x) > 0$ wherever $p(x) > 0$,*

$$D_{\mathrm{KL}}(p \,\|\, q) \;=\; \sum_{x} p(x) \log \frac{p(x)}{q(x)} \;\geq\; 0,$$

with equality if and only if $p = q$.

Proof. Let $Z \sim p$ and apply Jensen's inequality to the strictly convex function $-\log$ and the positive random variable $R = q(Z)/p(Z)$:

$$D_{\mathrm{KL}}(p \,\|\, q) = \mathbb{E}_p\left[-\log R\right] \;\geq\; -\log \mathbb{E}_p[R] = -\log \sum_{x : p(x) > 0} p(x)\, \frac{q(x)}{p(x)} \;\geq\; -\log 1 = 0,$$

since $\sum_{x:p(x)>0} q(x) \leq 1$. By the equality case, the bound is tight if and only if R is constant with probability one and q puts no mass off the support of p, that is, $q = c\,p$ with $c = 1$. ■

The same argument with the concave log gives the **arithmetic–geometric mean inequality**: for positive $x_1, \dots, x_N$, concavity gives $\log(\frac{1}{N}\sum_i x_i) \geq \frac{1}{N}\sum_i \log x_i$, and exponentiating,

$$\frac{1}{N}\sum_{i=1}^{N} x_i \;\geq\; \left(\prod_{i=1}^{N} x_i\right)^{1/N},$$

with equality if and only if all x_i coincide, because log is strictly concave. The evidence lower bound of Section 36.7, a bound on the log of an expectation, is the same inequality for the concave log.

The cell computes two things. The first is the size of the gap in a case we can check by hand: for $Z \sim \mathcal{N}(0,1)$ and $f(x) = e^x$ the two sides are $\mathbb{E}[e^Z] = \sqrt{e} \approx 1.649$ and $e^{\mathbb{E}[Z]} = 1$. The second concerns equality in the KL bound: as q moves toward p along the mixture $(1-\varepsilon)p + \varepsilon q$, does the divergence fall in proportion to ε, or faster?

```
rng = np.random.default_rng()
z = rng.normal(size=1_000_000)                # Z ~ N(0, 1)
print(f'E[exp(Z)] = {np.exp(z).mean():.2f} vs exp(E[Z]) = {np.exp(z.mean()):.2f}'
      f'  (population: {np.exp(0.5):.4f} vs 1)')
kl = lambda p, q: (p * np.log(p / q)).sum(axis=1)
```

```
p = rng.dirichlet(np.ones(10), size=1000)      # 1000 random pairs, 10 symbols
q = rng.dirichlet(np.ones(10), size=1000)
print('median KL(p || (1-eps) p + eps q), eps = 1e-2, 1e-3, 1e-4:',
      *[f'{np.median(kl(p, (1 - e) * p + e * q)):.1e}' for e in (1e-2, 1e-3, 1e-4)])
```

```
E[exp(Z)] = 1.65 vs exp(E[Z]) = 1.00  (population: 1.6487 vs 1)
median KL(p || (1-eps) p + eps q), eps = 1e-2, 1e-3, 1e-4: 1.6e-04 1.6e-06 1.6e-08
```

The sample averages come out at about 1.65 against 1.00; more samples shrink the Monte Carlo error but not this gap. The divergence falls faster than ε: each tenfold smaller ε lowers the median about a hundredfold, because $D_{\mathrm{KL}}(p \,\|\, (1-\varepsilon)p + \varepsilon q) \approx \frac{\varepsilon^2}{2} \sum_x (q(x) - p(x))^2 / p(x)$ once $\varepsilon\, q(x)/p(x)$ is small for every x. (Pairs in which p has a nearly empty symbol reach this regime last, which is why the cell reports the median rather than the maximum.) Equality is approached quadratically, and it is reached only in the limit $q \to p$.

35.3.3.2 Local and Global Minima

Local minima are global (Section 35.3.2.2). Three more facts of the same kind have one-line proofs. The set of minimizers is convex: if $\mathbf{x}$ and $\mathbf{y}$ both attain the minimum value $f^\star$, the chord inequality puts every point between them at or below $f^\star$. For differentiable f on an open set, $\nabla f(\mathbf{x}) = \mathbf{0}$ implies $f(\mathbf{x}) = f^\star$, because the first-order condition with a zero gradient reads $f(\mathbf{y}) \geq f(\mathbf{x})$ for all $\mathbf{y}$. And a small gradient means a small gap: the first-order condition aimed at a minimizer gives $f(\mathbf{x}) - f^\star \leq \nabla f(\mathbf{x})^\top (\mathbf{x} - \mathbf{x}^\star) \leq \|\nabla f(\mathbf{x})\| \, \|\mathbf{x} - \mathbf{x}^\star\|$. So the guarantee of Section 35.1.1, that gradient descent with $\eta = 1/L$ drives the smallest gradient norm among its iterates to zero, becomes for convex f a guarantee about the value, provided that the iterates stay within a bounded distance of $\mathbf{x}^\star$. Section 35.3.3.3 proves that they do, and makes the guarantee quantitative.

A minimizer need not be unique (a convex f may be constant on a whole segment), and it need not exist (e^x has none). Strict convexity repairs the first defect and strong convexity both.

Proposition (uniqueness). *A strictly convex f has at most one minimizer. A continuous μ-strongly convex f on a nonempty closed convex set C with nonempty interior has exactly one.*

Proof. If $\mathbf{x} \neq \mathbf{y}$ were both minimizers with value $f^\star$, the strict chord inequality at $\theta = \frac{1}{2}$ would give $f(\frac{\mathbf{x}+\mathbf{y}}{2}) < f^\star$, a contradiction. Strong convexity implies strict convexity, so it remains to show existence. We do so for a differentiable f on $C = \mathbb{R}^d$, the case of the losses in this book; the general case runs the same argument with a subgradient of $f - \frac{\mu}{2}\|\cdot\|^2$ at an interior point of C in place of the gradient. Fix any $\mathbf{x}_0$. The linear term of Equation 35.26 at $\mathbf{x}_0$ is at least $-\|\nabla f(\mathbf{x}_0)\| \, \|\mathbf{y} - \mathbf{x}_0\|$, so $f(\mathbf{y}) > f(\mathbf{x}_0)$ whenever $\|\mathbf{y} - \mathbf{x}_0\| > 2\|\nabla f(\mathbf{x}_0)\|/\mu$. The sublevel set $\{\mathbf{y} : f(\mathbf{y}) \leq f(\mathbf{x}_0)\}$ is therefore bounded, and it is closed because f is continuous. A continuous function on a nonempty closed bounded set attains its minimum, and the minimum over this sublevel set is the minimum over $\mathbb{R}^d$. ■

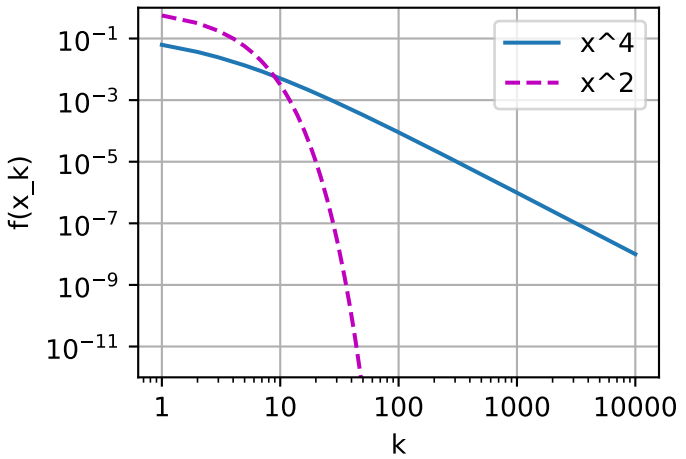
The hypotheses on C and on continuity exclude a function that jumps upward at the boundary of its domain, which no loss in this book does. The missing minimizer of e^x matters more in practice; Exercise 3 finds it in logistic regression on separable data.

35.3.3.3 Convergence Rates

Convexity says where a run ends. It remains to say how fast the run gets there, and on this convexity alone promises surprisingly little. Two functions show the range. Both x^2 and x^4 are strictly convex with minimizer 0, but only the square is strongly convex. We run the same loop on both, with step $\eta = \frac{1}{8}$ from $x_0 = 1$, for 10^4 steps.

```
eta, steps = 1 / 8, 10_000
x2, x4 = np.ones(steps + 1), np.ones(steps + 1)  # iterates from x_0 = 1
for j in range(steps):                           # same start, same step
    x2[j + 1] = x2[j] - eta * 2 * x2[j]          # f(x) = x^2
    x4[j + 1] = x4[j] - eta * 4 * x4[j]**3       # f(x) = x^4
k = np.arange(1, steps + 1)
print('x^2: first k with x_k < 1e-6:', k[x2[1:] < 1e-6][0])
print('x^4: x_k * sqrt(k) at k = 1e2, 1e3, 1e4:',
      (x4[1:] * np.sqrt(k))[[99, 999, 9999]].round(3))
d2l.plot(k, [x4[1:]**4, x2[1:]**2], 'k', 'f(x_k)', xscale='log',
         yscale='log', ylim=(1e-12, 1), legend=['x^4', 'x^2'])
```

```
x^2: first k with x_k < 1e-6: 49
x^4: x_k * sqrt(k) at k = 1e2, 1e3, 1e4: [0.973 0.996 1.   ]
```



On x^2 each step multiplies the iterate by $1 - 2\eta = 0.75$, so it falls below 10^{-6} at step 49 and the gap leaves the plot within fifty steps. On x^4 each step multiplies the iterate by $1 - 4\eta x_k^2$, a factor that approaches 1 as the iterate approaches the minimizer. The printed products $x_k\sqrt{k}$ approach 1, so the iterate behaves like $1/\sqrt{k}$ and is still 0.01 after 10^4 steps. On the log–log plot the gap x_k^4 is therefore a straight line of slope -2. Strict convexity allows at most one minimizer but promises no speed; a geometric rate needs $\mu > 0$. Exercise 10 proves the $1/\sqrt{k}$ law for the quartic.

The two propositions below make the contrast general. Every smooth strongly convex function behaves like the square, and without μ only a guarantee of order $1/k$ survives. Both proofs start from the **descent lemma** of Section 35.1.1. For an L-smooth f and the step $\mathbf{x}_{k+1} = \mathbf{x}_k - \frac{1}{L}\nabla f(\mathbf{x}_k)$, with k counting iterations,

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \frac{1}{2L}\,\|\nabla f(\mathbf{x}_k)\|^2.$$

One step lowers f by at least $\|\nabla f\|^2/(2L)$. By itself this guarantees only that gradients become small. Convexity ties the gradient to the gap: a large gap $f(\mathbf{x}_k) - f^\star$ forces a large gradient, so the steps cannot stall far from the minimum.

Proposition (gradient descent on smooth convex functions). *Let f be convex and L-smooth with a minimizer $\mathbf{x}^\star$. Gradient descent with $\eta = 1/L$ satisfies, for every $k \ge 1$,*

$$f(\mathbf{x}_k) - f(\mathbf{x}^\star) \;\le\; \frac{L\,\|\mathbf{x}_0 - \mathbf{x}^\star\|^2}{2k}. \tag{35.30}$$

The gap falls like $1/k$, scaled by the curvature bound and the squared distance from the start. With $L = 1$ and $\|\mathbf{x}_0 - \mathbf{x}^\star\| = 10$ the bound is $50/k$, so a gap of 10^{-2} is guaranteed after $5{,}000$ steps.

Proof. Combine the descent lemma with the first-order condition Equation 35.25 aimed at the optimum, $f(\mathbf{x}_k) - f(\mathbf{x}^\star) \le \nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star)$:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^\star) \;\le\; \nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2.$$

The non-routine step is to complete the square, so that the right side becomes a difference of two squared distances:

$$\nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2 = \frac{L}{2}\left(\|\mathbf{x}_k - \mathbf{x}^\star\|^2 - \left\|\mathbf{x}_k - \mathbf{x}^\star - \tfrac{1}{L}\nabla f(\mathbf{x}_k)\right\|^2\right),$$

and the second norm is $\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2$. Summing over the first k steps, the right side telescopes:

$$\sum_{j=0}^{k-1}\big(f(\mathbf{x}_{j+1}) - f(\mathbf{x}^\star)\big) \;\le\; \frac{L}{2}\left(\|\mathbf{x}_0 - \mathbf{x}^\star\|^2 - \|\mathbf{x}_k - \mathbf{x}^\star\|^2\right) \;\le\; \frac{L}{2}\,\|\mathbf{x}_0 - \mathbf{x}^\star\|^2.$$

The descent lemma makes $f(\mathbf{x}_j)$ nonincreasing, so the last term of the sum is the smallest, and $k\,(f(\mathbf{x}_k) - f(\mathbf{x}^\star))$ is at most the sum. This is Equation 35.30. ■

The bound contains no dimension: a model with a billion weights and the same L and starting distance gets the same guarantee. And the per-step inequality, before summing, has the left side $f(\mathbf{x}_{k+1}) - f^\star \ge 0$, so $\|\mathbf{x}_k - \mathbf{x}^\star\|$ never increases: the iterates move monotonically closer to every minimizer.

With strong convexity the $1/k$ sharpens to a geometric decay.

Proposition (linear rate under strong convexity). *Let f be μ-strongly convex and L-smooth, with minimizer $\mathbf{x}^\star$. Gradient descent with $\eta = 1/L$ satisfies*

$$f(\mathbf{x}_k) - f(\mathbf{x}^\star) \;\le\; \left(1 - \frac{\mu}{L}\right)^k \left(f(\mathbf{x}_0) - f(\mathbf{x}^\star)\right). \tag{35.31}$$

Each step removes at least the fraction $\mu/L = 1/\kappa$ of the remaining gap.

Proof. First, strong convexity ties the size of the gradient to the remaining gap. Minimize both sides of Equation 35.26 over $\mathbf{y}$: the left side's minimum is $f(\mathbf{x}^\star)$, and the right side is a quadratic in $\mathbf{y}$ minimized at $\mathbf{y} = \mathbf{x} - \frac{1}{\mu}\nabla f(\mathbf{x})$, where it equals $f(\mathbf{x}) - \frac{1}{2\mu}\|\nabla f(\mathbf{x})\|^2$. Hence

$$\frac{1}{2}\,\|\nabla f(\mathbf{x})\|^2 \;\ge\; \mu\,(f(\mathbf{x}) - f(\mathbf{x}^\star)) \qquad \text{for all } \mathbf{x}. \tag{35.32}$$

Second, substitute Equation 35.32 into the descent lemma:

$$f(\mathbf{x}_{k+1}) - f(\mathbf{x}^\star) \;\le\; f(\mathbf{x}_k) - f(\mathbf{x}^\star) - \frac{1}{2L}\|\nabla f(\mathbf{x}_k)\|^2 \;\le\; \left(1 - \frac{\mu}{L}\right)\left(f(\mathbf{x}_k) - f(\mathbf{x}^\star)\right),$$

and iterate k times. ■

The second half of the proof used no convexity, only the inequality Equation 35.32; we return to it in Section 35.3.5. Reaching a relative accuracy ϵ costs about $\kappa \log(1/\epsilon)$ iterations, against $O(1/\epsilon)$ without strong convexity. On the running example $\kappa = 10^3$, and six digits of the gap, $(1 - 10^{-3})^k \le 10^{-6}$, require $k \approx 13{,}800$ steps.

The bound must hold in every run; a measurement can tell us how much room it leaves. We build a least-squares problem whose Hessian $\mathbf{X}^\top\mathbf{X}$ has eigenvalues 1 and 10 by construction, so that $\mu = 1$, $L = 10$, and $\kappa = 10$, run gradient descent at $\eta = 1/L$, and print the ratio of successive gaps $f(\mathbf{w}_{k+1}) - f^\star$ over $f(\mathbf{w}_k) - f^\star$.

```
rng = np.random.default_rng()
Q, _ = np.linalg.qr(rng.normal(size=(20, 2)))  # orthonormal columns
R, _ = np.linalg.qr(rng.normal(size=(2, 2)))   # random rotation
X = Q @ np.diag([1.0, np.sqrt(10.0)]) @ R      # X^T X has eigenvalues 1, 10
y = rng.normal(size=20)
f = lambda w: 0.5 * ((X @ w - y) ** 2).sum()
mu, L = np.linalg.eigvalsh(X.T @ X)[[0, -1]]
w_star = np.linalg.lstsq(X, y, rcond=None)[0]
w, gaps = np.zeros(2), [f(np.zeros(2)) - f(w_star)]
for _ in range(60):                            # gradient descent, eta = 1/L
    w = w - X.T @ (X @ w - y) / L
    gaps.append(f(w) - f(w_star))
r = np.array(gaps[1:]) / np.array(gaps[:-1])   # per-step contraction of f - f*
print(f'mu = {mu:.4f}, L = {L:.4f}: bound 1 - mu/L = {1 - mu / L:.4f},'
      f' its square {(1 - mu / L)**2:.4f}')
print('first three ratios:', r[:3].round(4), ' largest ratio:', r.max().round(4))
```

```
mu = 1.0000, L = 10.0000: bound 1 - mu/L = 0.9000, its square 0.8100
first three ratios: [0.6084 0.81   0.81  ]  largest ratio: 0.81
```

The bound promises a factor 0.9 per step. The measured ratio is 0.81 at every step from the second on, which is $(1 - \mu/L)^2$; the first ratio depends on the start and varies from run to

run. The reason is visible in the eigenvector coordinates of $\mathbf{X}^\top\mathbf{X}$. With $\eta = 1/L$, each step multiplies the stiff component of $\mathbf{w} - \mathbf{w}^\star$ by $1 - L/L = 0$, so it disappears in the first step. In two dimensions only the slow component is left; its distance to the minimizer contracts by $1 - \mu/L$, and the gap, quadratic in the distance, by the square. With more dimensions the ratio climbs toward $(1 - \mu/L)^2$ as the faster components die out.

The proof gives away a square. A finer analysis shows that gradient descent with $\eta = 1/L$ contracts $f - f^\star$ by $(1 - \mu/L)^2$ per step on every L-smooth μ-strongly convex function (Taylor et al. 2018), and quadratics attain it; Exercise 5 proves the weaker $\kappa(1-\mu/L)^{2k}$. On the running example, six digits of the gap cost about $13{,}800$ steps by the bound and about $6{,}900$ at the squared factor. The best fixed step $2/(\mu + L)$ of Section 35.1, nearly twice as long, halves the count again to $3{,}454$. One factor of two belongs to the proof and one to the step.

35.3.4 Recognizing Convexity

The consequences above are worth having only if we can tell which losses are convex. Hessians are tedious and do not exist at kinks, so in practice convexity is assembled from parts.

35.3.4.1 A Calculus of Convex Functions

A small set of operations preserves convexity, and most losses are built from convex atoms by these operations.

1. **Nonnegative weighted sums.** If $f_1, \dots, f_m$ are convex and $w_i \geq 0$, then $\sum_i w_i f_i$ is convex: multiply the chord inequality for each f_i by w_i and add. An average of losses over a training set is such a sum.

2. **Affine pre-composition.** If f is convex then so is $\mathbf{x} \mapsto f(\mathbf{A}\mathbf{x} + \mathbf{b})$: affine maps send the segment between $\mathbf{x}$ and $\mathbf{y}$ to the segment between their images, with the same θ, so the chord inequality transfers unchanged. A loss that is convex in a model's *output* is therefore convex in the *weights of a linear model* producing that output.

3. **Pointwise maximum and supremum.** If each f_i is convex, so is $\max_i f_i$. The proof goes through epigraphs: the epigraph of a maximum is the intersection of the epigraphs, and intersections of convex sets are convex. The upper envelope of any family of lines is convex, however large the family.

4. **Monotone composition.** If g is convex and h is convex *and nondecreasing*, then $h \circ g$ is convex. Two inequalities prove it:

$$h(g(\theta\mathbf{x} + (1-\theta)\mathbf{y})) \;\leq\; h(\theta g(\mathbf{x}) + (1-\theta) g(\mathbf{y})) \;\leq\; \theta\, h(g(\mathbf{x})) + (1-\theta)\, h(g(\mathbf{y})),$$

   the first by convexity of g and monotonicity of h, the second by convexity of h. The monotonicity of the outer map is essential: $h(t) = t^2$ is convex but not nondecreasing, and with the convex $g(x) = x^2 - 1$ it composes to the double well $(x^2 - 1)^2$, which becomes ϕ after adding $x/2$.

These rules certify most of the convex losses in this book. The hinge loss $\max(0, 1 - y\,\mathbf{w}^\top\mathbf{x})$ is a maximum of two affine functions of $\mathbf{w}$ (rules 3 and 2). The ℓ_1 norm $\|\mathbf{w}\|_1 = \sum_i \max(w_i, -w_i)$ is a sum of maxima of affine functions (rules 1, 3, 2). The logistic loss

$\log(1 + e^{-y\mathbf{w}^\top \mathbf{x}})$ is the function $t \mapsto \log(1 + e^t)$, convex because its second derivative $e^t/(1 + e^t)^2$ is positive, pre-composed with an affine map (rule 2) and summed over data (rule 1). Adding the ridge penalty to any of these gives a strongly convex loss. A worked case: $\max(x^2, 1 - x)$ is convex by rule 3, but it is not strongly convex, because it equals the affine function $1 - x$ on the interval $(-1.618, 0.618)$ between the two roots of $x^2 = 1 - x$.

What the list lacks is composition with an inner map that is neither affine nor convex. A two-layer network $\mathbf{W}_2 \,\mathrm{relu}(\mathbf{W}_1 \mathbf{x})$ is such a composition in its weights, which is where convexity is lost (Section 35.3.5).

35.3.4.2 Log-Sum-Exp and the Softmax Covariance

The function behind every softmax cross-entropy loss is the log-sum-exp $\mathrm{lse}(\mathbf{x}) = \log \sum_{i=1}^{d} e^{x_i}$, a smooth version of $\max_i x_i$ that exceeds it by at most $\log d$. None of the four rules applies to it directly, but its Hessian has a probabilistic reading that settles the question.

Proposition (log-sum-exp is convex). *The Hessian of* lse *at* $\mathbf{x}$ *is*

$$\nabla^2 \mathrm{lse}(\mathbf{x}) = \mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top, \qquad \mathbf{s} = \mathrm{softmax}(\mathbf{x}), \tag{35.33}$$

which is the covariance matrix of the one-hot encoding of a class drawn from $\mathbf{s}$*, hence PSD, hence* lse *is convex.*

Proof. Differentiating once gives $\partial\,\mathrm{lse}/\partial x_i = e^{x_i}/\sum_k e^{x_k} = s_i$: the gradient of lse is the softmax. Differentiating $s_i = e^{x_i}/\sum_k e^{x_k}$ again by the quotient rule,

$$\frac{\partial s_i}{\partial x_j} = \frac{\delta_{ij}\, e^{x_i}}{\sum_k e^{x_k}} - \frac{e^{x_i}\, e^{x_j}}{\left(\sum_k e^{x_k}\right)^2} = s_i\,(\delta_{ij} - s_j),$$

which is the claimed matrix; Section 34.4 derives the same softmax Jacobian. Now let Y be a random class with $P(Y = i) = s_i$ and let $\mathbf{e}_Y$ be its one-hot vector. Then $\mathbb{E}[\mathbf{e}_Y] = \mathbf{s}$ and $\mathbb{E}[\mathbf{e}_Y \mathbf{e}_Y^\top] = \mathrm{diag}(\mathbf{s})$, so $\mathrm{Cov}(\mathbf{e}_Y) = \mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$ is the Hessian. Covariances are PSD; directly, for any $\mathbf{v}$,

$$\mathbf{v}^\top \nabla^2 \mathrm{lse}(\mathbf{x})\, \mathbf{v} = \sum_i s_i v_i^2 - \left(\sum_i s_i v_i\right)^2 = \mathbb{E}\left[v_Y^2\right] - \left(\mathbb{E}[v_Y]\right)^2 = \mathrm{Var}(v_Y) \;\geq\; 0. \;\blacksquare$$

The curvature of log-sum-exp in a direction $\mathbf{v}$ is the variance of the entry v_Y picked out by a random class.

This reading has three consequences. The cross-entropy loss in the logits, $\ell(\mathbf{z}) = \mathrm{lse}(\mathbf{z}) - z_y$ for the true class y, is log-sum-exp plus an affine function, hence convex. Softmax regression (Section 3.1) composes it with the affine map $\mathbf{z} = \mathbf{W}\mathbf{x} + \mathbf{b}$, so its loss is convex in the weights by rule 2: for fixed features, training only the linear classifier is a convex problem. And the formula predicts a *zero* eigenvalue in the direction $\mathbf{1}$, because $\mathrm{Var}(v_Y)$ vanishes

when all v_i are equal. This is the shift invariance $\mathrm{lse}(\mathbf{x} + c\mathbf{1}) = \mathrm{lse}(\mathbf{x}) + c$ behind the numerically stable softmax. The same Jacobian $\mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$ appears in the backward pass of every attention head, whose weights are a softmax of query–key scores (Section 10.2).

How large can this curvature be? No eigenvalue of the Hessian exceeds $\frac{1}{2}$, whatever the logits (Exercise 6 proves the bound), so $\frac{1}{2}$ is a smoothness constant of log-sum-exp. Where is it approached, and how large is the curvature at typical logits? The cell draws $10{,}000$ vectors of six logits, scales each by a temperature factor c, and prints the largest Hessian eigenvalue over the draws and its median.

```
z = rng.normal(size=(10_000, 6))              # 10,000 draws of six logits
print('  c   largest eigenvalue   median')
for c in (0.1, 1.0, 3.0, 10.0, 30.0):
    s = np.exp(c * z - (c * z).max(axis=1, keepdims=True))
    s /= s.sum(axis=1, keepdims=True)         # softmax of each row
    hess = s[:, :, None] * np.eye(6) - s[:, :, None] * s[:, None, :]
    top = np.linalg.eigvalsh(hess)[:, -1]     # largest eigenvalue per draw
    print(f'{c:5.1f}  {top.max():14.4f}  {np.median(top):12.4f}')
```

```
  c   largest eigenvalue   median
 0.1          0.2193        0.1843
 1.0          0.4546        0.3071
 3.0          0.4983        0.2995
10.0          0.5000        0.0165
30.0          0.5000        0.0000
```

The largest eigenvalue never exceeds $\frac{1}{2}$, so softmax cross-entropy is $\frac{1}{2}$-smooth in the logits at every temperature. Small logits ($c = 0.1$) give a nearly uniform softmax, and the curvature stays a little above $\frac{1}{6}$, its value when six classes share the mass equally. At moderate temperatures the typical curvature is highest, with a median near 0.3. As c grows further the typical draw becomes confident, its softmax nearly one-hot, and its curvature collapses: the median falls toward zero, so the loss is nearly flat around a confident prediction. The maximum over the draws moves the other way, toward $\frac{1}{2}$. It belongs to the rare draws whose two largest logits nearly tie, where the softmax splits its mass between two classes.

35.3.5 Beyond Convexity

The losses of deep networks are not convex in their weights. This part shows why with two weights, names the condition that keeps the linear rate of gradient descent without convexity, and describes how the optimizer chooses among many minimizers.

35.3.5.1 Nonconvex Networks

Take $f(a, b) = (ab - 1)^2$: a linear model with two stacked scalar weights and a squared loss, the smallest "deep" network. Figure 35.12 draws its contours. Its global minima form the hyperbola $\{ab = 1\}$, a nonconvex set, while the minimizer set of a convex function is convex (Section 35.3.3). So f cannot be convex. The average of the minimizers $(1, 1)$ and $(-1, -1)$ is the origin, where $f = 1$ sits strictly above the minimum 0. The origin is a stationary point but not a minimum: $\nabla^2 f(0, 0)$ has eigenvalues ± 2, so it is a saddle.

Real networks have the same structural source of nonconvexity. Permuting the hidden units of a layer, together with their incoming and outgoing weights, leaves the computed func-

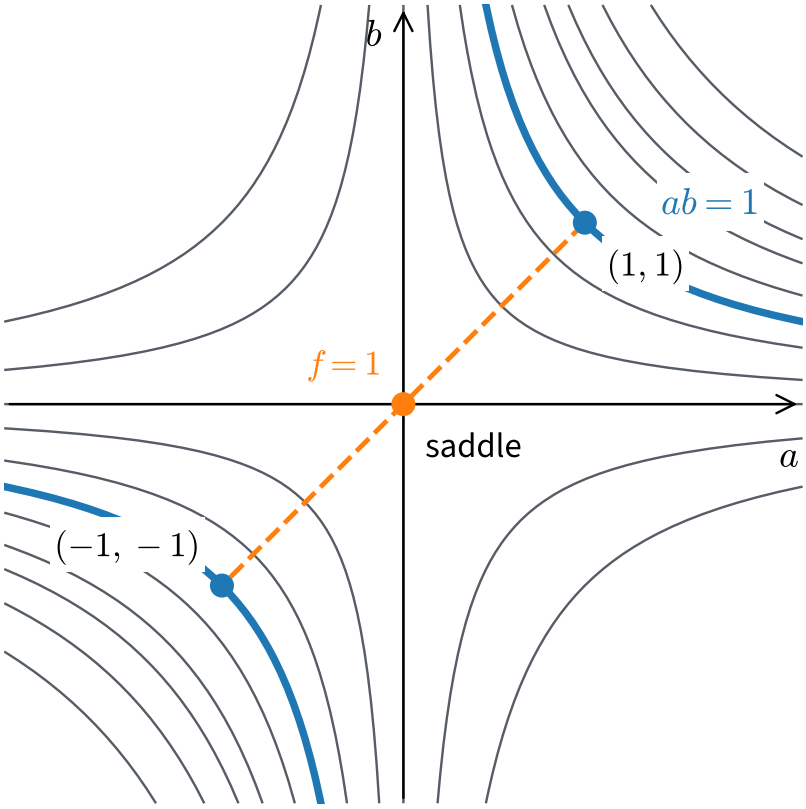


Figure 35.12: The two-weight model $(ab-1)^2$. Its minimizers form the hyperbola $ab=1$, which is not convex: the midpoint of $(1,1)$ and $(-1,-1)$ is the origin, where $f=1$ and the Hessian has eigenvalues ± 2.

tion unchanged, so every minimizer comes with many symmetric copies scattered across parameter space. Generically the copies are distinct, and averaging two of them changes the function and raises the loss, as the midpoint of $(1, 1)$ and $(-1, -1)$ did. A minimizer set that contains distinct copies but not their averages is not convex.

The smallest instance has two ReLU units. The weights $(w_1, v_1, w_2, v_2) = (1, 1, -1, 1)$ of $v_1 \operatorname{relu}(w_1 x) + v_2 \operatorname{relu}(w_2 x)$ compute $|x|$, and swapping the two units computes $|x|$ as well. The average of the two weight vectors has both input weights 0 and computes the zero function (Exercise 11).

35.3.5.2 The Polyak–Łojasiewicz Condition

The linear-rate proof used convexity only to derive Equation 35.32; its second half needed nothing else. That inequality has a name. A function f with infimum $f^\star$ satisfies the **Polyak–Łojasiewicz (PL) inequality** with constant $\mu > 0$ if $\frac{1}{2}\|\nabla f(\mathbf{x})\|^2 \geq \mu\,(f(\mathbf{x}) - f^\star)$ for all $\mathbf{x}$; Polyak and Łojasiewicz introduced it independently in 1963 (Polyak 1963; Łojasiewicz 1963). It says that the gradient can be small only where the value is nearly optimal.

Proposition (linear rate under PL). *Let f be L-smooth, bounded below with $f^\star = \inf f$, and satisfy the PL inequality with constant μ. Then gradient descent with $\eta = 1/L$ satisfies $f(\mathbf{x}_k) - f^\star \leq (1 - \mu/L)^k\,(f(\mathbf{x}_0) - f^\star)$.*

Proof. The second half of the proof of the linear rate under strong convexity, unchanged: the descent lemma and the PL inequality give $f(\mathbf{x}_{k+1}) - f^\star \leq (1 - \mu/L)(f(\mathbf{x}_k) - f^\star)$ (Karimi et al. 2016). ■

A PL function may have many minimizers, plateaus of minimizers, and negative curvature. Two properties are forced, though: every stationary point must be a global minimizer (a zero gradient forces $f(\mathbf{x}) = f^\star$), *and* $f - f^\star$ must grow at least quadratically with the distance to the set of minimizers. The quartic x^4 has only one stationary point, its global minimizer, yet it fails PL: $\frac{1}{2}f'(x)^2 = 8x^6$, and $8x^6 \geq \mu x^4$ breaks down as $x \to 0$ for every $\mu > 0$. That failure is the slow descent on x^4 measured in Section 35.3.3.3. Strong convexity implies PL (the first half of the proof there); the converse fails.

The standard example of a nonconvex PL function is $f(x) = x^2 + 3\sin^2 x$, whose second derivative $2 + 6\cos 2x$ dips to -4 and peaks at $L = 8$. The cell estimates its PL constant on $[-5, 5]$ and runs gradient descent from $x_0 = 4.5$ with the step $\frac{1}{16} = 1/(2L)$. (At $\eta = 1/L$ the local factor $1 - f''(0)\,\eta$ is 0, and the last ratios would say nothing about a rate.) At this step the quadratic ceiling guarantees a decrease of $\frac{3}{8L}f'(x_k)^2$ per step, and the PL inequality turns it into the factor $1 - \frac{3\mu}{4L}$.

```
f_pl = lambda x: x**2 + 3 * np.sin(x)**2       # nonconvex; f* = 0 at x = 0
df_pl = lambda x: 2 * x + 3 * np.sin(2 * x)
xs = np.linspace(-5, 5, 100001)
xs = xs[np.abs(xs) > 1e-3]                     # avoid 0/0 at the minimizer
mu = (0.5 * df_pl(xs)**2 / f_pl(xs)).min()
print(f'PL constant on [-5, 5]: mu = {mu:.4f}')
print(f"min f'' = {(2 + 6 * np.cos(2 * xs)).min():.2f}  (f is not convex)")
path = [4.5]
for _ in range(80):                            # gradient descent, eta = 1/16
    path.append(path[-1] - df_pl(path[-1]) / 16.0)
```

```
path = np.array(path)
ratios = f_pl(path[1:]) / f_pl(path[:-1])      # per-step contraction of f - f*
j = ratios.argmax()
print(f'guaranteed {1 - 3 * mu / 32:.4f}; largest ratio {ratios[j]:.4f}'
      f' from x = {path[j]:.2f}; last ratio {ratios[-1]:.4f}')
```

```
PL constant on [-5, 5]: mu = 0.1755
min f'' = -4.00  (f is not convex)
guaranteed 0.9835; largest ratio 0.9778 from x = 2.19; last ratio 0.2500
```

The PL constant on $[-5, 5]$ is $\mu \approx 0.1755$, and the minimum of f'' is -4: the function is nonconvex yet PL. The window suffices, because outside it the ratio $\frac{1}{2}f'(x)^2/f(x)$ stays well above this value and tends to 2 as $|x| \to \infty$; Exercise 4 proves the bound.

With this μ the guaranteed factor is about 0.98 per step. The largest measured ratio, about 0.978, comes from $x \approx 2.2$, where $\frac{1}{2}f'(x)^2/f(x)$ attains its minimum μ, so there the guarantee is nearly tight. Near the minimizer the ratio settles at 0.25. That limit comes from the local quadratic: near 0, $f \approx \frac{1}{2}f''(0)\,x^2$ with $f''(0) = 8$, each step multiplies x by $1 - 8/16 = \frac{1}{2}$, and the gap, quadratic in x, by $\frac{1}{4}$.

PL can also hold on part of the space. For $(ab-1)^2$ the gradient satisfies $\frac{1}{2}\|\nabla f\|^2 = 2(a^2 + b^2)f$, so PL holds with constant $2r^2$ on the region $\{a^2 + b^2 \geq r^2\}$ and with no constant near the saddle at the origin (Exercise 7). Analyses of wide networks establish such local PL inequalities near interpolating solutions (C. Liu et al. 2022), which gives a rate for the training loss and says nothing about generalization.

35.3.5.3 Implicit Bias

When a loss has many global minimizers, convex or not, the optimizer decides which one it reaches: it has an **implicit bias**. For deep learning this is not a curiosity. A network with more weights than training examples can fit its data in many ways, all with the same training loss and with different predictions elsewhere, so the choice among them decides how the model generalizes. The simplest case is underdetermined least squares, with fewer examples than weights and $\mathbf{X}$ of full row rank. Every gradient $\mathbf{X}^\top(\mathbf{X}\mathbf{w} - \mathbf{y})$ lies in the row space of $\mathbf{X}$, so iterates started from $\mathbf{w}_0 = \mathbf{0}$ never leave it. The limit is therefore the one interpolant in the row space, the *minimum-norm* interpolant that the pseudoinverse computes (Exercise 6 of Section 33.5). For logistic regression on separable data the loss has no minimizer at all, yet the direction of the iterates converges to the maximum-margin separator, the solution of the hard-margin support vector machine (Soudry et al. 2018) (Exercise 9). Which minimizer is selected depends on the model, the loss, the start, and the update rule. A fixed diagonal preconditioner $\mathbf{D}$, for instance, leads to the interpolant that minimizes $\mathbf{w}^\top\mathbf{D}^{-1}\mathbf{w}$ instead of $\|\mathbf{w}\|^2$ (Exercise 8), so the per-coordinate steps of Adam (Section 9.5) need not select the solution that gradient descent selects.

Every result of this section assumes the exact gradient and weights free to range over all of $\mathbb{R}^d$. Training uses minibatch gradients that are right only on average (Section 35.4), and a weight vector confined to a ball, a simplex, or a half-space needs the projections and multipliers of Section 35.6.

35.3.6 Summary

A convex function lies below its chords. For differentiable functions this is equivalent to lying above every tangent, and for twice-differentiable functions to a PSD Hessian; at a corner a subgradient takes the place of the gradient. Because one gradient bounds the function everywhere, a stationary point is a global minimizer, and the descent lemma turns into global rates for gradient descent: $L\|\mathbf{x}_0 - \mathbf{x}^\star\|^2/(2k)$ for smooth convex f, and a factor $1 - \mu/L$ per step with strong convexity, hence about $\kappa \log(1/\epsilon)$ steps. Jensen's inequality is the chord inequality for expectations and gives $D_{\mathrm{KL}} \geq 0$. Four closure rules certify the hinge, ℓ_1, and logistic losses; log-sum-exp, and with it the softmax cross-entropy, is convex because its Hessian is a covariance. Network losses are generically nonconvex. Where a PL inequality holds, the linear rate survives without convexity, and among many minimizers the optimizer's dynamics pick one.

35.3.7 Exercises

1. **Subdifferentials of norms and maxima.** Compute each subdifferential from the definition Equation 35.27.
    1. Compute $\partial\|\mathbf{x}\|_2$ at every $\mathbf{x} \in \mathbb{R}^d$, including $\mathbf{x} = \mathbf{0}$.
    2. Let $f(\mathbf{x}) = \max_i x_i$, and let $\mathcal{A}(\mathbf{x}) = \{i : x_i = f(\mathbf{x})\}$ be its set of active coordinates. Show that $\partial f(\mathbf{x})$ is the convex hull of $\{\mathbf{e}_i : i \in \mathcal{A}(\mathbf{x})\}$. Backpropagation through max pooling passes the incoming gradient to a single maximal input, so it uses one vertex $\mathbf{e}_i$ of this hull. Suppose that two or more inputs tie. Show that a small step along $-\mathbf{e}_i$ does not decrease f. For which $\mathbf{g} \in \partial f(\mathbf{x})$ is $-\mathbf{g}$ a descent direction, and which of them makes the directional derivative of f along $-\mathbf{g}/\|\mathbf{g}\|$ most negative?
    3. Use (a) and the optimality criterion $\mathbf{0} \in \partial(\cdot)$ to minimize $\lambda\|\mathbf{w}\|_2 + \frac{1}{2}\|\mathbf{w} - \mathbf{z}\|^2$ over $\mathbf{w} \in \mathbb{R}^d$, which gives the proximal operator of $\lambda\|\cdot\|_2$. Compare the zero pattern of the solution with that of coordinatewise soft-thresholding, $w_i = \operatorname{sign}(z_i)\max(|z_i| - \lambda, 0)$. What does this penalty do when $\mathbf{w}$ collects the outgoing weights of one hidden unit?

2. **Convex combinations and Jensen's inequality.** Write $\sum_{i=1}^N \theta_i \mathbf{x}_i$ with $\theta_i \geq 0$ and $\sum_i \theta_i = 1$ for a convex combination of N points.
    1. Show by induction on N that a convex set C contains every convex combination of points of C. This is the inclusion that completes the description of the convex hull in Section 35.3.1.
    2. Apply (a) to the epigraph of a convex function f to derive the finite form Equation 35.28 of Jensen's inequality.
    3. Let f be twice differentiable with $\alpha \leq f'' \leq \beta$ on an interval that contains the values of a random variable Z with mean m and variance s^2. Show that the Jensen gap $\mathbb{E}[f(Z)] - f(m)$ lies between $\alpha s^2/2$ and $\beta s^2/2$. Check the bounds on the two-point example of the text, $Z \in \{0, 2\}$ with $f(x) = x^2$, where $\alpha = \beta = 2$. Then apply the result to $f = -\log$ to bound $\log \mathbb{E}[Z] - \mathbb{E}[\log Z]$ for Z with values in $[a, b] \subset (0, \infty)$.

3. **Existence of minimizers.** Let $\ell(\mathbf{w}) = \sum_i \log(1 + e^{-y_i \mathbf{w}^\top \mathbf{x}_i})$ be the logistic loss of labels $y_i \in \{\pm 1\}$, and let $\mathbf{X}$ be the matrix with rows $\mathbf{x}_i^\top$. The uniqueness proposition of Section 35.3.3 states that a strictly convex function has at most one minimizer, and

that a continuous, strongly convex function on a nonempty closed convex set with nonempty interior has exactly one.

1. Suppose that some $\mathbf{w}$ satisfies $y_i \mathbf{w}^\top \mathbf{x}_i > 0$ for every i, that is, the data are linearly separable by a hyperplane through the origin. Show that $\inf_{\mathbf{w}} \ell(\mathbf{w}) = 0$ and that ℓ has no minimizer.
2. Show that ℓ is strictly convex whenever $\mathbf{X}$ has full column rank. Why does this not contradict (a)?
3. Show that $\ell(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2$ has exactly one minimizer for every $\lambda > 0$ and every dataset. Which hypothesis of the uniqueness proposition does the penalty supply?

4. **The PL condition beyond convexity.** The cell that studies $f(x) = x^2 + 3\sin^2 x$ measures its PL constant $\mu \approx 0.1755$ on the window $[-5, 5]$.

   1. Bound $|f'(x)|$ from below and $f(x)$ from above for $|x| \geq 5$. Conclude that the PL inequality Equation 35.32 holds on all of $\mathbb{R}$ with the constant measured on the window.
   2. Let $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $n < d$ have full row rank, and let $f(\mathbf{w}) = \frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$. Show that f is not strongly convex and has infinitely many minimizers, but satisfies the PL inequality with $\mu = \lambda_{\min}(\mathbf{X}\mathbf{X}^\top)$, the smallest eigenvalue of $\mathbf{X}\mathbf{X}^\top$. Which linear rate does gradient descent with $\eta = 1/L$ then achieve, and how is L related to $\mathbf{X}\mathbf{X}^\top$?

5. code **Contraction of the distance.** ⊠ The rate Equation 35.31 guarantees the per-step factor $1 - \mu/L$ for $f - f^\star$, but the rate-check cell measures $(1 - \mu/L)^2$. Let f be twice continuously differentiable with $\mu\mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$, and let $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \nabla f(\mathbf{x}_k)$. Two facts from Section 35.1 are used: the quadratic ceiling $f(\mathbf{y}) \leq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y} - \mathbf{x}) + \frac{L}{2}\|\mathbf{y} - \mathbf{x}\|^2$ and the step $\eta^\star = 2/(\mu + L)$, the best fixed step on a quadratic.

   1. Write $\nabla f(\mathbf{x}_k) = \bar{\mathbf{H}}_k(\mathbf{x}_k - \mathbf{x}^\star)$ with $\bar{\mathbf{H}}_k = \int_0^1 \nabla^2 f(\mathbf{x}^\star + s(\mathbf{x}_k - \mathbf{x}^\star))\, ds$. Show that

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \leq \max(|1 - \eta\mu|,\ |1 - \eta L|)\, \|\mathbf{x}_k - \mathbf{x}^\star\|. \tag{35.34}$$

   2. Set $\eta = 1/L$ and $\kappa = L/\mu$. Combine (a) with the lower bound Equation 35.26 and the quadratic ceiling, both taken at $\mathbf{x}^\star$, to show that $f(\mathbf{x}_k) - f^\star \leq \kappa\,(1 - \mu/L)^{2k}\,(f(\mathbf{x}_0) - f^\star)$. From which k on is this bound stronger than Equation 35.31?

   3. Predict the per-step contraction of $f - f^\star$ on the $\kappa = 10$ design of the rate-check cell at $\eta^\star = 2/11$. (Hint: both eigen-components are multiplied by numbers of absolute value $9/11$.) Verify the prediction by changing the step in that cell, and explain why, unlike at $\eta = 1/L$, the first ratio is no longer special.

6. **Smoothing the maximum.** ⊠ A maximum of affine functions, the shape of hinge-type and worst-case objectives, has a kink wherever two of its pieces tie. For a temperature $\tau > 0$, let $\mathrm{lse}_\tau(\mathbf{z}) = \tau\,\mathrm{lse}(\mathbf{z}/\tau)$ for $\mathbf{z} \in \mathbb{R}^m$. This smoothing technique is due to Nesterov (2005) .

   1. Show that $\max_j z_j \leq \mathrm{lse}(\mathbf{z}) \leq \max_j z_j + \log m$, and conclude that $\max_j z_j \leq \mathrm{lse}_\tau(\mathbf{z}) \leq \max_j z_j + \tau \log m$.

2. Use Equation 35.33 to show that the largest eigenvalue of $\nabla^2 \mathrm{lse}_\tau(\mathbf{z})$ is at most $1/(2\tau)$, by bounding $\mathrm{Var}(v_Y)$ by $\mathbb{E}[(v_Y - c)^2]$ for a well-chosen constant c. Is the bound attained?

3. Let $F(\mathbf{w}) = \max_j (\mathbf{A}\mathbf{w} + \mathbf{r})_j$ and $F_\tau(\mathbf{w}) = \mathrm{lse}_\tau(\mathbf{A}\mathbf{w} + \mathbf{r})$, and assume that F_τ has a minimizer within distance R of $\mathbf{w}_0$. Show with (b) that F_τ is L-smooth for $L = \|\mathbf{A}\|_2^2/(2\tau)$, and that k steps of gradient descent on F_τ with $\eta = 1/L$ give

$$F(\mathbf{w}_k) - \min F \le \tau \log m + \frac{\|\mathbf{A}\|_2^2 R^2}{4\tau k}. \tag{35.35}$$

Choose τ to minimize this bound, and give the number of steps that guarantees accuracy ϵ. How does the count change when an accelerated method with error $O(LR^2/k^2)$ replaces gradient descent?

7. **A factored scalar model.** ⊠ The two-weight model $f(a, b) = (ab - 1)^2$ of Section 35.3.5 is nonconvex, and its minimizers form the hyperbola $ab = 1$.

   1. Show that the origin is the only stationary point that is not a global minimum. Compute the eigenvalues of $\nabla^2 f(0, 0)$, classify the origin, and locate it in Figure 35.12, the contour plot of f.
   2. Show that $\frac{1}{2}\|\nabla f(a, b)\|^2 = 2(a^2 + b^2)\, f(a, b)$. On which sets does the PL inequality Equation 35.32 hold, and with which constant? Why does no constant $\mu > 0$ work on all of $\mathbb{R}^2$?
   3. Show that the imbalance $a^2 - b^2$ is constant along the gradient flow $(\dot{a}, \dot{b}) = -\nabla f(a, b)$ (Du et al. 2018), and that a gradient-descent step with step size η multiplies it by $1 - 4\eta^2(ab - 1)^2$.
   4. Use $a^2 + b^2 \ge |a^2 - b^2|$ and (b) to show that the gradient flow from (a_0, b_0) satisfies $f(a(t), b(t)) \le e^{-4|a_0^2 - b_0^2|\, t} f(a_0, b_0)$. What does this bound give for a balanced start $a_0 = b_0$, and why does gradient descent from a small balanced start still converge?

8. code **Implicit bias and the optimizer.** ⊠ Exercise 6 of Section 33.5 shows on paper that gradient descent on $\frac{1}{2}\|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$, for $\mathbf{X} \in \mathbb{R}^{n \times d}$ with $n < d$ and full row rank, started at $\mathbf{w}_0 = \mathbf{0}$, stays in the row space of $\mathbf{X}$ and converges to the minimum-norm interpolant $\mathbf{X}^+\mathbf{y}$. Here the start stays at $\mathbf{w}_0 = \mathbf{0}$ and the algorithm changes. Take a random 4×10 system, and compare the limit of each method and its norm with `np.linalg.pinv(X) @ y`.

   1. Run gradient descent with $\eta = 1/\lambda_{\max}(\mathbf{X}^\top\mathbf{X})$, and track the norm of the component of $\mathbf{w}_k$ orthogonal to the row space, computed with the projector $\mathbf{I} - \mathbf{X}^+\mathbf{X}$.
   2. Heavy-ball momentum updates $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\nabla f(\mathbf{w}_k) + \beta(\mathbf{w}_k - \mathbf{w}_{k-1})$. Show that its iterates stay in the row space, and predict their limit. Verify the prediction with the step of (a) and $\beta = 0.9$.
   3. Replace the update by the diagonally preconditioned step $\mathbf{w}_{k+1} = \mathbf{w}_k - \eta\,\mathbf{D}\mathbf{X}^\top(\mathbf{X}\mathbf{w}_k - \mathbf{y})$, where $\mathbf{D}$ is a fixed diagonal matrix with entries drawn uniformly from $[0.1, 10]$ and $\eta = 1/\lambda_{\max}(\mathbf{D}^{1/2}\mathbf{X}^\top\mathbf{X}\mathbf{D}^{1/2})$. Use the substitution $\mathbf{u} = \mathbf{D}^{-1/2}\mathbf{w}$ to find the limit in closed form, and verify it numerically. Is the limit still the minimum-norm interpolant? What does this suggest about the implicit bias of adaptive methods such as Adam (Section 9.5), whose per-coordinate step sizes act like $\mathbf{D}$?

4. Run exact cyclic coordinate descent for 1000 sweeps, with $\mathbf{A} = \mathbf{X}^\top\mathbf{X}$, $\mathbf{b} = \mathbf{X}^\top\mathbf{y}$, and the update $w_j \leftarrow (b_j - \sum_{i\neq j} A_{ji}w_i)/A_{jj}$ for $j = 1, \dots, d$ in turn. Check that the limit interpolates the data. Is it the minimum-norm interpolant, and which step of the row-space argument fails? Repeat the run three times with coordinates drawn uniformly at random. Does the limit depend on the draw?

9. [extended] **Max-margin bias of gradient descent.** ▯▯ On data that a hyperplane through the origin separates, the logistic loss has no minimizer (part (a) of Exercise 3). Soudry et al. (2018) prove that gradient descent on it nevertheless diverges in a definite direction: the iterates satisfy $\mathbf{w}_k = \hat{\mathbf{w}}\log k + \rho_k$, with ρ_k bounded for almost every dataset, where

$$\hat{\mathbf{w}} = \operatorname*{argmin}_{\mathbf{w}} \|\mathbf{w}\| \quad \text{subject to} \quad y_i\mathbf{w}^\top\mathbf{x}_i \geq 1 \text{ for all } i \tag{35.36}$$

is the hard-margin SVM solution, with margin $\gamma = 1/\|\hat{\mathbf{w}}\|$. The vector $\hat{\mathbf{w}}$ is characterized by $\hat{\mathbf{w}} = \sum_i \alpha_i y_i \mathbf{x}_i$ with $\alpha_i \geq 0$, where $\alpha_i = 0$ unless $y_i\hat{\mathbf{w}}^\top\mathbf{x}_i = 1$; the points with equality are the support vectors.

   1. Derive the direction heuristically. Replace the logistic loss by its tail e^{-q} in the margin $q = y_i\mathbf{w}^\top\mathbf{x}_i$, and suppose that $\mathbf{w}_k = g(k)\,\mathbf{u} + O(1)$ with $\|\mathbf{u}\| = 1$ and $g(k) \to \infty$. Show that the negative gradient is asymptotically a nonnegative combination of the points $y_i\mathbf{x}_i$ of smallest margin $y_i\mathbf{u}^\top\mathbf{x}_i$. Conclude that $\mathbf{u}$ is proportional to $\hat{\mathbf{w}}$.
   2. Derive the growth rate. Keep only the support vectors in the gradient flow $\dot{\mathbf{w}} = -\nabla\ell(\mathbf{w})$, show that $\dot{g} \approx c\,e^{-\gamma g}$ for a constant $c > 0$, and conclude that $\|\mathbf{w}(t)\| \approx \|\hat{\mathbf{w}}\|\log t$.
   3. Take the support vectors $\mathbf{x}_1 = (0.5, 1.5)$ and $\mathbf{x}_2 = (1.5, 0.5)$ with label $+1$, their negatives with label -1, and 12 random points, 6 per class, with $y_i\hat{\mathbf{w}}^\top\mathbf{x}_i > 1$ for $\hat{\mathbf{w}} = (\frac{1}{2}, \frac{1}{2})$. Let $\mathbf{X}$ be the 16×2 matrix with rows $\mathbf{x}_i^\top$. Run gradient descent on the summed logistic loss from $\mathbf{w}_0 = \mathbf{0}$ with $\eta = 1/\|\mathbf{X}\|_2^2$ for 10^6 steps, which takes seconds in vectorized NumPy. Plot, against k on a logarithmic axis, the norm $\|\mathbf{w}_k\|$, the training loss, the angle between $\mathbf{w}_k$ and $\hat{\mathbf{w}}$, and the margin gap $\gamma - \min_i y_i\mathbf{w}_k^\top\mathbf{x}_i/\|\mathbf{w}_k\|$. Compare the slope of $\|\mathbf{w}_k\|$ against $\log k$ with (b).
   4. The training error reaches zero within the first few steps. By what factor do the margin gap and the training loss change between $k = 10^3$ and $k = 10^6$? What does the comparison imply for stopping training as soon as the training error reaches zero?

10. **Gradient descent on x^4.** ▯ The quartic is strictly but not strongly convex, and the text measured $x_k\sqrt{k} \to 1$ for gradient descent with $\eta = \frac{1}{8}$ from $x_0 = 1$. Here you prove the rate. Let $x_{k+1} = x_k - 4\eta x_k^3$ with $0 < x_0 < 1/(2\sqrt{\eta})$.

    1. Show that $0 < x_{k+1} < x_k$ for every k and that $x_k \to 0$.
    2. Show that $1/x_{k+1}^2 = 1/x_k^2 + 8\eta + O(x_k^2)$, and conclude that $x_k\sqrt{k} \to 1/\sqrt{8\eta}$ and $f(x_k) = x_k^4 \sim 1/(64\eta^2k^2)$. Compare with the measured limit at $\eta = \frac{1}{8}$.
    3. On $[-1, 1]$, which contains every iterate from $x_0 = 1$, the quartic is 12-smooth, and the proof of Equation 35.30 uses smoothness only between iterates. At $\eta = 1/12$ compare the proposition's bound $12 \cdot x_0^2/(2k) = 6/k$ with the true gap at $k = 100$. Explain why no bound of the form $c\rho^k$ with $\rho < 1$ can hold. (Hint: $\frac{1}{2}f'(x)^2 = 8x^6$, against μx^4 in the PL inequality.)

11. [short-code] **Permutation symmetry and nonconvexity.** A network $h(x) = v_1 \,\mathrm{relu}(w_1 x) + v_2 \,\mathrm{relu}(w_2 x)$ with parameters $(w_1, v_1, w_2, v_2) = (1, 1, -1, 1)$ computes $|x|$.
    1. Write the parameter vector of the network with the two units swapped, and check that it computes the same function.
    2. Compute the function computed by the average of the two parameter vectors.
    3. Draw 1,000 inputs $x \sim \mathcal{N}(0, 1)$ with targets $y = |x|$, and print the mean squared error of the three networks. Explain why the third is close to $\mathbb{E}[x^2] = 1$. Which inequality does the midpoint violate, and what does this say about convexity of the loss in the parameters?
    4. For a hidden layer of m units with distinct parameters, how many copies of a minimizer does permutation symmetry produce? Show that the average of all copies gives every unit the same parameters, and explain why the convex hull of the copies therefore generally contains non-minimizers.
12. **Convexity of the log-partition function.** This exercise generalizes the log-sum-exp proposition to the log-partition functions that Section 36.4 builds on. For a finite set $\mathcal{X}$, weights $h(x) > 0$, statistics $\mathbf{T}(x) \in \mathbb{R}^p$, and natural parameters $\eta \in \mathbb{R}^p$ (not a step size here), let $A(\eta) = \log \sum_{x \in \mathcal{X}} h(x) \exp(\eta^\top \mathbf{T}(x))$ and $p_\eta(x) = h(x) \exp(\eta^\top \mathbf{T}(x) - A(\eta))$.
    1. Show that $\nabla A(\eta) = \mathbb{E}_\eta[\mathbf{T}]$ and $\nabla^2 A(\eta) = \mathrm{Cov}_\eta[\mathbf{T}]$, with expectation and covariance under p_η.
    2. Conclude that A is convex, and that it is strictly convex if and only if no nonzero $\mathbf{v}$ makes $\mathbf{v}^\top \mathbf{T}(x)$ constant on $\mathcal{X}$.
    3. Recover Equation 35.33 with $\mathcal{X} = \{1, \dots, d\}$, $\mathbf{T}(x) = \mathbf{e}_x$, and $h \equiv 1$, and name the direction in which A is flat.
    4. For observations $x_1, \dots, x_n$ with average statistic $\bar{\mathbf{T}} = \frac{1}{n} \sum_i \mathbf{T}(x_i)$, show that the average negative log-likelihood $A(\eta) - \eta^\top \bar{\mathbf{T}} - \frac{1}{n} \sum_i \log h(x_i)$ is convex in η, and that a minimizer satisfies $\mathbb{E}_\eta[\mathbf{T}] = \bar{\mathbf{T}}$ (moment matching). For the softmax case of (c), why does no minimizer exist when one class is never observed?

35.4 Stochastic Gradient Methods

Training a network almost never uses the exact gradient of its loss. Each step computes the gradient on a minibatch of a few hundred examples instead of the whole training set. Every section so far assumed exact gradients: curvature set the number of steps, and convexity made a stationary point a global minimizer. A minibatch gradient is right on average, but it is noisy, and the noise changes the question from how many steps a run needs to how close to the minimizer it can get. This section asks what the noise costs and how that cost can be removed.

The simplest loss already shows the cost. Take $f(x) = \frac{1}{2} x^2$, whose minimizer is 0, add noise of unit variance to every gradient, and use the step $\eta = 0.1$. With exact gradients each step would shrink x by a tenth, and the iterate would approach 0 as closely as we like. With noise it never settles: its mean square tends to $0.1/1.9 \approx 0.053$, so it stays about 0.23 away from the minimizer however many steps we take. Three numbers set the loss at which a run ends: the size of the noise, the step, and the number of steps we can afford.

We first measure the noise of a minibatch gradient. On a quadratic we then compute exactly

where a fixed step stalls and how long it takes to get there. Next we show how a decaying step removes the stall and what its constant must be, and we bound the loss that a constant step tuned to a fixed budget reaches on convex and on nonconvex losses. Finally, we cancel the noise on a finite training set with a stored gradient. Three results are worth remembering: the floor is proportional to the step, a $1/k$ schedule needs a large enough constant, and a budget of K steps calls for a step proportional to $1/\sqrt{K}$. Variance reduction, the last part, can be skipped on a first reading.

35.4.1 Minibatch Gradients

35.4.1.1 The Cost of a Full Gradient

A training loss averages the losses of n examples,

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} f_i(\mathbf{x}),$$

where f_i is the loss on example i and $\mathbf{x} \in \mathbb{R}^d$ holds the parameters. Its gradient $\nabla f = \frac{1}{n}\sum_i \nabla f_i$ is an average too, so computing it touches all n examples. With $n = 10^9$ examples, one step of gradient descent is one pass over the data, and gradient descent needs thousands of steps. On the chapter's running example $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$, whose two curvatures differ by a factor of 1000, even the best fixed step needs about $7{,}000$ of them to shrink the error a million-fold (Section 35.1). **Stochastic gradient descent** (SGD) replaces the average over all examples by the average over a random **minibatch** $\mathcal{B}$ of b indices,

$$\hat{\mathbf{g}}_b(\mathbf{x}) = \frac{1}{b}\sum_{i\in\mathcal{B}} \nabla f_i(\mathbf{x}), \qquad \mathbf{x}_{k+1} = \mathbf{x}_k - \eta_k\,\hat{\mathbf{g}}_b(\mathbf{x}_k). \tag{35.37}$$

A step now reads b examples instead of n. A minibatch of $b = 256$ out of 10^9 examples costs about four million times less than a full pass. Throughout the section, k counts optimizer steps, and the step size η_k may change with k. In the analysis, a fresh minibatch is drawn at every step and its b indices are independent and uniform on $\{1, \dots, n\}$, that is, drawn with replacement. Training loops usually shuffle the data and draw without replacement, which multiplies the variance below by $(n-b)/(n-1)$, a factor close to one when $b \ll n$ (Section 9.3 derives it).

35.4.1.2 Unbiasedness and the $1/b$ Law

How good is the estimate? Two numbers answer this: its mean, and its mean squared distance from the true gradient, which we call its noise power. The noise power depends on how much the example gradients spread around their average, which their covariance measures.

Proposition (minibatch gradients are unbiased with noise power $\operatorname{tr}\Sigma/b$**).** *Fix* $\mathbf{x}$ *and let*

$$\Sigma = \frac{1}{n}\sum_{i=1}^{n} \big(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\big)\big(\nabla f_i(\mathbf{x}) - \nabla f(\mathbf{x})\big)^{\top}$$

be the covariance of the gradient of one uniformly drawn example. If the b indices of $\mathcal{B}$ are independent and uniform on $\{1, \dots, n\}$, then

$$\mathbb{E}\big[\hat{\mathbf{g}}_b(\mathbf{x})\big] = \nabla f(\mathbf{x}), \qquad \mathbb{E}\big[\|\hat{\mathbf{g}}_b(\mathbf{x}) - \nabla f(\mathbf{x})\|^2\big] = \frac{\operatorname{tr}\Sigma}{b}. \tag{35.38}$$

Proof. A uniform index i gives $\mathbb{E}[\nabla f_i(\mathbf{x})] = \frac{1}{n}\sum_i \nabla f_i(\mathbf{x}) = \nabla f(\mathbf{x})$, so each of the b terms has mean $\nabla f(\mathbf{x})$, and so does their average. The terms are independent, each with covariance Σ. Covariances of independent terms add, so their sum has covariance $b\Sigma$ and their average $b\Sigma/b^2 = \Sigma/b$. The expected squared norm of a mean-zero vector is the trace of its covariance. ■

In a run, $\mathbf{x}$ is the current iterate $\mathbf{x}_k$ and the minibatch is drawn after it, so both statements hold conditionally on $\mathbf{x}_k$. Unbiasedness rests on uniform sampling: a sampler that draws example i with probability p_i must weight its gradient by $1/(np_i)$ to stay unbiased.

The noise power falls like $1/b$, so its root mean square falls only like $1/\sqrt{b}$: a hundred times more examples per step make the gradient ten times quieter. The signal ∇f has power $\|\nabla f\|^2$ at every batch size, and the two powers are equal at the batch size $b_{\text{noise}} = \operatorname{tr}\Sigma/\|\nabla f\|^2$, which the main text calls the gradient-noise scale (Section 9.8); exercise 1 derives the version weighted by curvature.

To see the size of this noise on data, we use a small logistic-regression problem, which the step-size experiment of the next part uses again. It has $n = 200$ examples $\mathbf{a}_i \in \mathbb{R}^5$ with standard normal entries, labels $y_i \in \{-1, +1\}$ drawn from the logistic model with weights $(1, -2, 1.5, 0, 2)$, and the loss

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \log\left(1 + e^{-y_i \mathbf{a}_i^\top \mathbf{x}}\right) + \frac{0.1}{2}\|\mathbf{x}\|^2.$$

The penalty makes f strongly convex with $\mu \geq 0.1$. The gradient of example i is $\nabla f_i(\mathbf{x}) = -y_i s_i \mathbf{a}_i + 0.1\,\mathbf{x}$, where $s_i = 1/(1 + e^{y_i \mathbf{a}_i^\top \mathbf{x}})$ is the probability that the model assigns to the wrong label. The cell defines the data, the loss, and all n example gradients at once, as the rows of an $n \times d$ matrix.

```
n, d, reg = 200, 5, 0.1                          # examples, parameters, l2 weight
x_true = np.array([1.0, -2.0, 1.5, 0.0, 2.0])
rng = np.random.default_rng()
A = rng.normal(size=(n, d))                       # row i: the features a_i
y = np.where(rng.random(n) < 1 / (1 + np.exp(-A @ x_true)), 1.0, -1.0)

def loss(x, A, y, reg):
    """The regularized logistic loss f(x)."""
    return np.mean(np.log1p(np.exp(-y * (A @ x)))) + 0.5 * reg * x @ x

def example_grads(x, A, y, reg):
    """The n x d matrix whose row i is the gradient of f_i at x."""
    s = 1 / (1 + np.exp(y * (A @ x)))             # probability of the wrong label
    return -(y * s)[:, None] * A + reg * x
```

The proposition is exact, so sampling can only agree with it. We sample once, at $\mathbf{x} = \mathbf{0}$ with $10{,}000$ minibatches of each size, mainly to learn how large $\operatorname{tr}\Sigma$ is on this problem.

```
G = example_grads(np.zeros(d), A, y, reg)        # per-example gradients at x = 0
g = G.mean(axis=0)                                # the full gradient
tr_sigma = ((G - g)**2).sum(axis=1).mean()
print(f'tr Sigma = {tr_sigma:.3f}')
for b in [1, 4, 16, 64]:
    idx = rng.integers(0, n, size=(10000, b))    # minibatches, with replacement
    err = ((G[idx].mean(axis=1) - g)**2).sum(axis=1).mean()
    print(f'b = {b:2d}: measured {err:.4f}, tr Sigma / b = {tr_sigma / b:.4f}, '
          f'ratio {err * b / tr_sigma:.3f}')
```

```
tr Sigma = 1.131
b =  1: measured 1.1188, tr Sigma / b = 1.1311, ratio 0.989
b =  4: measured 0.2847, tr Sigma / b = 0.2828, ratio 1.007
b = 16: measured 0.0699, tr Sigma / b = 0.0707, ratio 0.989
b = 64: measured 0.0177, tr Sigma / b = 0.0177, ratio 1.003
```

The single-example noise power $\operatorname{tr}\Sigma$ is about 1.1, and the measured error agrees with $\operatorname{tr}\Sigma/b$ to within the sampling error, a few percent, at every batch size. What the noise does to a whole run is the next question.

35.4.1.3 The Noise Floor

Does the noise average out over a run, or does it accumulate? Section 9.3 measured the cloud in which SGD settles on a valley and derived its size. Here we solve the whole trajectory, which also gives the time to get there. The smallest case suffices: one coordinate of a quadratic, $f(x) = \frac{1}{2}\lambda x^2$ with curvature $\lambda > 0$ and minimizer 0. A stochastic gradient is the true gradient λx_k plus noise ξ_k that has mean zero given x_k and variance σ^2. This σ^2 plays the role of a diagonal entry of Σ/b, so everything below that is proportional to σ^2 falls like $1/b$. One step reads

$$x_{k+1} = x_k - \eta\,(\lambda x_k + \xi_k) = \rho\, x_k - \eta\,\xi_k, \qquad \rho = 1 - \eta\lambda.$$

Each step shrinks the distance to the minimizer by the factor ρ, as gradient descent does, and then adds a random kick $-\eta\xi_k$. We follow the size of the iterate through its second moment $e_k = \mathbb{E}[x_k^2]$. When we square the step and take expectations, the cross term $-2\rho\eta\,\mathbb{E}[x_k\xi_k]$ vanishes because the noise has mean zero given x_k:

$$e_{k+1} = \rho^2 e_k + \eta^2\sigma^2.$$

The second moment contracts by the factor ρ^2 and gains the noise power $\eta^2\sigma^2$ at every step. Nearly everything in this section is read off this recursion. A linear recursion with a constant input is solved through its fixed point. The fixed point V satisfies $V = \rho^2 V + \eta^2\sigma^2$; subtracting the two equations shows that the remainder $e_k - V$ is multiplied by ρ^2 at every step. Hence

$$e_k = V + \rho^{2k}\,(e_0 - V) \tag{35.39}$$

with the **noise floor**

$$V = \frac{\eta^2 \sigma^2}{1 - \rho^2} = \frac{\eta\, \sigma^2}{\lambda\,(2 - \eta\lambda)} \approx \frac{\eta\, \sigma^2}{2\lambda} \qquad (\eta\lambda \ll 1). \tag{35.40}$$

The floor grows in proportion to the step and to the noise variance, and it shrinks as the curvature grows. These formulas hold for $0 < \eta < 2/\lambda$, below the stability ceiling of gradient descent on a quadratic. Above it the second moment grows even without noise.

The second moment has two parts. The transient $\rho^{2k}(e_0 - V)$ is the one gradient descent also has, and it decays geometrically. The floor V exists only because of the noise. Far from the minimizer the transient dominates and SGD moves like gradient descent. Near it, the noise kicks the iterate by the same amount at every step, while the pull $\eta\lambda x$ toward the minimizer weakens as the iterate approaches it, and the two balance at $e = V$. The iterate does not converge. Its distribution settles into a stationary one with mean square V, a **noise ball** around the minimizer whose radius, the root mean square distance of about $\sqrt{\eta\sigma^2/(2\lambda)}$, grows like $\sqrt{\eta}$. Figure 35.13 shows both phases on a two-dimensional bowl.

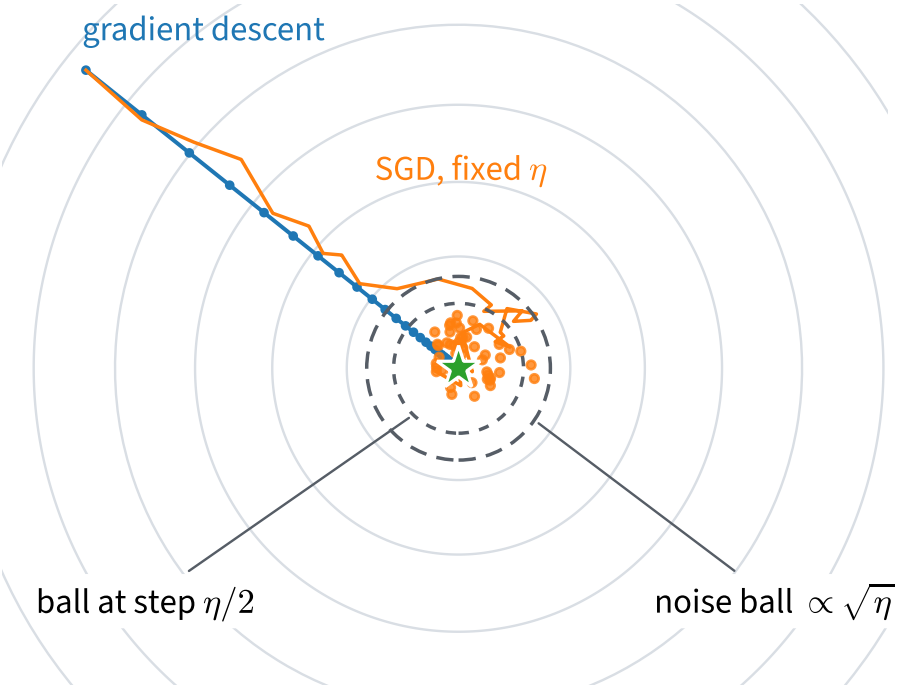


Figure 35.13: Gradient descent and fixed-step SGD on $\frac{1}{2}\|\mathbf{x}\|^2$ with $\eta = 0.15$ and gradient noise of standard deviation 0.6. The full-batch path converges to the minimizer; SGD descends like gradient descent and then fluctuates inside a noise ball. The dashed circles mark the noise ball at this step and at half of it; their radii, drawn at three times the per-coordinate root-mean-square distance $\sqrt{\eta\sigma^2/(2\lambda)}$ so that they enclose the scatter, differ by a factor $\sqrt{2}$.

Training is judged by the loss rather than by the distance. Since $\mathbb{E} f(x_k) = \frac{1}{2}\lambda e_k$, the loss

floor is

$$\tfrac{1}{2}\lambda V = \frac{\eta\,\sigma^2}{2\,(2-\eta\lambda)} \;\approx\; \frac{\eta\,\sigma^2}{4}.$$

The loss floor is proportional to the step and to the noise power and, for small steps, independent of the curvature: a flat direction lets the iterate wander farther, by exactly the factor that its flatness takes off the loss.

On a quadratic in d dimensions, each coordinate along an eigenvector of the Hessian obeys this scalar recursion with its own eigenvalue λ_i and the noise variance σ_i^2 along that eigenvector, so the loss floors $\eta\sigma_i^2/(2(2-\eta\lambda_i))$ add. The σ_i^2 are the diagonal entries of Σ/b in the eigenbasis and sum to $\operatorname{tr}\Sigma/b$. When every $\eta\lambda_i \ll 1$, the total floor is therefore about $\eta\operatorname{tr}\Sigma/(4b)$, and it depends on the step and the batch only through η/b. Halving the step and doubling the batch lower it by the same factor. This coupling is behind the learning-rate rules for large batches in Section 9.8.

How long does the run take to reach the floor? Take $\lambda = \sigma = 1$, as in the opening, where $\eta = 0.1$ gave $V = 0.053$. At $\eta = 0.3$ the floor is $V = 0.176$, a little more than three times as high, and the loss floor is 0.088. Starting from $x_0 = 3$, so $e_0 = 9$, the remainder $e_k - V$ falls below a tenth of V after 36 steps at $\eta = 0.1$ (where $\rho^2 = 0.81$) and after 9 steps at $\eta = 0.3$ (where $\rho^2 = 0.49$). A third of the step gives about a third of the floor and takes four times as many steps to reach it. Every remedy in this section manages this trade-off between the floor and the speed.

On the running example $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ we model the noise directly: to the gradient $(x_1, 1000\,x_2)$ we add noise of mean zero and variance $\sigma^2 = 1$ on each coordinate, drawn afresh at every step. The two coordinates then obey the scalar recursion with $\lambda_1 = 1$ and $\lambda_2 = 1000$. Take $\eta = 1/L = 10^{-3}$, where $L = 1000$ is the largest curvature; this step is half the stability ceiling $2/1000$. The stiff coordinate has $\rho_2 = 0$: one step erases its transient, and from then on it sits on its floor $V_2 = 10^{-3}/1000 = 10^{-6}$. The slow coordinate has $\rho_1 = 0.999$ and the floor $V_1 = 10^{-3}/1.999 = 5.0 \times 10^{-4}$, and its transient needs $\log 1000/(-2\log 0.999) \approx 3{,}450$ steps to fall a thousandfold. The loss floor is 7.5×10^{-4}: 2.5×10^{-4} from the slow coordinate and 5×10^{-4} from the stiff one. The coordinate that gradient descent handles in one step holds two thirds of the floor, because for this coordinate $\eta\lambda_2 = 1$, where the loss floor is $\eta\sigma^2/2$, twice the small-step value $\eta\sigma^2/4$. Ill-conditioning costs steps on the slow coordinate; noise costs loss on both.

The floor formula is exact only for a quadratic with additive noise. The next part tests the law "floor proportional to η" on the logistic loss.

35.4.2 Step-Size Decay

35.4.2.1 The Robbins–Monro Conditions

The floor is proportional to η, so reaching the minimizer requires steps that shrink to zero. Steps that shrink too fast, however, stop the iterate short: a gradient of unit size moves it by at most $\sum_k \eta_k$ in total. The conditions of Robbins and Monro (1951) state both demands,

$$\sum_k \eta_k = \infty, \qquad \sum_k \eta_k^2 < \infty. \tag{35.41}$$

The first condition lets the iterate travel any distance. The second keeps the total injected noise variance, $\sigma^2 \sum_k \eta_k^2$, finite. The steps $\eta_k = c/k$ satisfy both; $\eta_k = c/\sqrt{k}$ and a constant step satisfy only the first. Together with unbiased stochastic gradients whose variance grows at most quadratically with the distance from the minimizer, the two conditions make SGD converge to the minimizer of a strongly convex loss (Robbins and Monro 1951; Bottou et al. 2018).

On the quadratic model, mean-square convergence needs less. Unrolling the recursion shows that $e_k \to 0$ whenever $\eta_k \to 0$ and $\sum_k \eta_k = \infty$ (with $\eta_k < 2/\lambda$), and $\eta_k = c/\sqrt{k}$ gives $e_k \approx c\sigma^2/(2\lambda\sqrt{k})$, the floor of the current step (exercise 7). The square-summable condition is what the general theorem needs for its stronger claim, the convergence of almost every run.

Two things remain to be seen on a loss that is not quadratic: whether the floor is still proportional to η, and whether a decaying step goes below it. On the logistic problem we need the optimal value $f^\star$, which $4{,}000$ full-gradient steps supply, and a minibatch SGD loop with batch size b that records the gap $f(\mathbf{x}_k) - f^\star$ after every step.

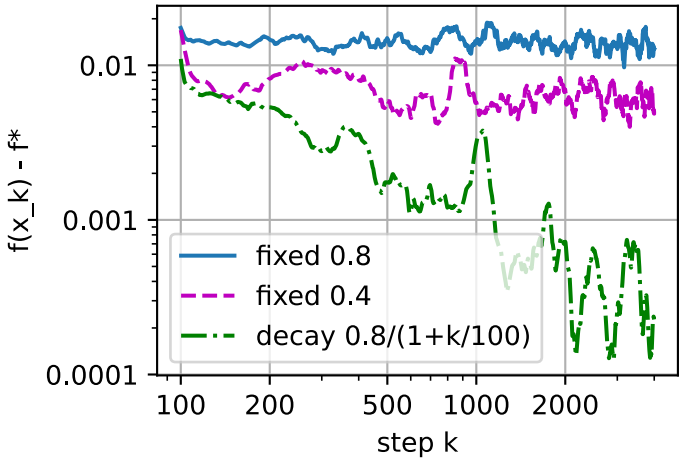
```
x_star = np.zeros(d)
for _ in range(4000):                              # full-gradient reference optimum
    x_star -= 0.5 * example_grads(x_star, A, y, reg).mean(axis=0)
f_star = loss(x_star, A, y, reg)

def sgd(schedule, A, y, reg, f_star, rng, steps=4000, b=8):
    """Minibatch SGD from x = 0; returns the gap f(x_k) - f* after every step."""
    x, gaps = np.zeros(A.shape[1]), np.empty(steps)
    for k in range(steps):
        idx = rng.integers(0, len(y), size=b)
        x -= schedule(k) * example_grads(x, A[idx], y[idx], reg).mean(axis=0)
        gaps[k] = loss(x, A, y, reg) - f_star
    return gaps
```

If the quadratic model is a guide, halving a fixed step should halve the gap at which the run stalls. We run $4{,}000$ steps with $b = 8$ at $\eta = 0.8$ and at $\eta = 0.4$, and with the decaying step $\eta_k = 80/(k + 100)$, which starts at 0.8 and which the code writes as $0.8/(1 + k/100)$. The cell prints the mean gap over the last $1{,}000$ steps of each run and two ratios, and plots the gaps averaged over windows of 100 steps.

```
rules = {'fixed 0.8': lambda k: 0.8, 'fixed 0.4': lambda k: 0.4,
         'decay 0.8/(1+k/100)': lambda k: 0.8 / (1 + k / 100)}
gaps = {name: sgd(s, A, y, reg, f_star, rng) for name, s in rules.items()}
tail = [gap[-1000:].mean() for gap in gaps.values()]
print('mean gap, last 1000 steps: ' + ', '.join(f'{t:.1e}' for t in tail))
print(f'floor(0.8) / floor(0.4) = {tail[0] / tail[1]:.2f}, '
      f'floor(0.8) / decay = {tail[0] / tail[2]:.0f}')
window = np.ones(100) / 100                        # plot 100-step moving averages
d2l.plot(np.arange(100, 4001), [np.convolve(gap, window, 'valid') for gap in gaps.values()],
         'step k', 'f(x_k) - f*', xscale='log', yscale='log', legend=list(gaps))
```

```
mean gap, last 1000 steps: 1.4e-02, 5.9e-03, 4.1e-04
floor(0.8) / floor(0.4) = 2.35, floor(0.8) / decay = 34
```



Both fixed steps stall. The mean gap over the last $1{,}000$ steps is about 1 to 2×10^{-2} at $\eta = 0.8$, and halving the step about halves it. The ratio scatters between about 1.7 and 2.8 across runs, around the factor 2 that the quadratic model predicts, although the loss is not quadratic. The decaying run ends roughly 20 to 90 times below the $\eta = 0.8$ floor, and on the log–log plot its curve still trends down, jagged with minibatch noise. Its early large steps cover the distance to the minimizer, and its late small steps shrink the noise ball.

35.4.2.2 The Constant in a $1/k$ Schedule

The schedule $\eta_k = c/(k + k_0)$ satisfies both Robbins–Monro conditions for every $c > 0$, and the shift k_0 keeps the first steps below the stability ceiling. Does every c give the rate $1/k$? The scalar model with curvature $\lambda = \mu$ answers exactly. Writing $t = k + k_0$, its recursion becomes

$$e_{k+1} = \left(1 - \frac{c\mu}{t}\right)^2 e_k + \frac{c^2\sigma^2}{t^2}.$$

The contraction now weakens like $1/t$, while the injected noise falls like $1/t^2$. Which of the two wins depends on c. If c is too small, the steps shrink faster than the iterate can use them, and the schedule limits the progress before the noise does.

Proposition (the constant of a $1/k$ schedule). *On the scalar model with curvature μ and steps $\eta_k = c/(k + k_0)$, if $2c\mu > 1$ then $k\, e_k \to A$ with*

$$A = \frac{c^2\sigma^2}{2c\mu - 1}. \tag{35.42}$$

If $2c\mu < 1$, then e_k decays like $k^{-2c\mu}$, more slowly than $1/k$, and at the boundary $2c\mu = 1$ like $\log k/k$. For every $c > 0$, $e_k \to 0$.

We do not prove the proposition here. We derive the constant by matching leading terms and argue the other two cases informally; exercise 2 supplies the proof of the two rates, by an induction above the threshold and by unrolling the recursion below it.

Derivation of the constant. Substitute $e_k \approx A/t$ into the recursion. The left side is $A/(t+1) \approx A/t - A/t^2$. On the right, $(1 - c\mu/t)^2 \approx 1 - 2c\mu/t$, so the right side is $A/t - 2c\mu A/t^2 + c^2\sigma^2/t^2$ up to terms of order t^{-3}. The $1/t$ terms agree, and matching the $1/t^2$ terms gives $-A = -2c\mu A + c^2\sigma^2$, that is, $A(2c\mu - 1) = c^2\sigma^2$. The step that is not routine is the shift on the left: moving from t to $t+1$ contributes $-A/t^2$, of the same order as the contraction, and it is the source of the threshold. Without it we would find $A = c\sigma^2/(2\mu)$ for every c.

Below the threshold (informally). The contractions decide the rate, both for the initial error and for the noise injected early: their product is $\prod_j (1 - c\mu/(j+k_0))^2 \approx (k_0/t)^{2c\mu}$, because $\log(1-u) \approx -u$ and $\sum_{j \le t} 1/j \approx \log t$, and for $2c\mu < 1$ this decays more slowly than $1/t$. At the boundary the noise injected at step j survives as about $c^2\sigma^2/(jt)$, and the sum over j produces the logarithm.

Take $\mu = \sigma = 1$. The constant $c = 1$ gives $A = 1$; iterating the recursion from $e_0 = 1$ with $k_0 = 10$ gives $k\,e_k = 1.0001$ at $k = 10^6$. The constant $c = 2$ gives $A = 4/3$. Below the threshold, $c = 0.25$ gives a log–log slope of -0.50 between $k = 10^5$ and 10^6, and $c = 0.1$ a slope of -0.20. The constant A is smallest at $c = 1/\mu$, where $A = \sigma^2/\mu^2$: the best $1/k$ schedule reaches $e_k \approx \sigma^2/(\mu^2 k)$. Averaging the iterates of a run with a small constant step reaches the same constant, with no decay at all (exercise 6). The threshold makes the schedule fragile: the curvature μ is rarely known, and a constant chosen too small loses the rate itself. This is one reason practical schedules decay more slowly than $1/k$.

On the running example the shift k_0 cannot be small either. The first step must respect the stiff coordinate's ceiling, $c/k_0 < 2/L$, so $k_0 > cL/2$, and the best constant for the slow coordinate, $c = 1/\mu = 1$, forces $k_0 > 500$. With $k_0 = 1000$ and $\mathbf{x}_0 = (1, 1)$, the slow coordinate's transient still holds a fraction $(k_0/(k+k_0))^2 \approx 0.008$ of its initial value at $k = 10^4$, and the expected loss ends at 4.2×10^{-3}, more than five times the constant step's floor of 7.5×10^{-4}. The small early steps that the stiff coordinate demands leave the slow coordinate far from its minimizer, and the condition number reappears in k_0.

35.4.2.3 Schedule Shapes under the Exact Recursion

Practice uses other shapes than $1/k$, above all cosine decay and warmup–stable–decay; Section 9.6 compares them on trained networks. No theorem ranks them there. On the quadratic model, however, the expected loss of any schedule can be computed exactly, with no runs and no random numbers. The derivation of the floor holds step by step when the step size changes, so

$$e_{k+1} = (1 - \eta_k \lambda)^2 e_k + \eta_k^2 \sigma^2, \qquad \mathbb{E} f(x_k) = \tfrac{1}{2} \lambda e_k.$$

Iterating this recursion computes the expected loss of a whole run in K multiplications. We compare four shapes over a budget of K steps, which Figure 35.14 draws. They are a constant step η_0; the $1/k$ decay $c/(k+k_0)$ with $c = \eta_0 k_0$, so that it starts at η_0; **cosine decay** (Loshchilov and Hutter 2016); and **warmup–stable–decay** (WSD) (Hu et al. 2024), which holds η_0 for the first 80% of the budget and then decays linearly to zero:

$$\eta_k^{\cos} = \tfrac{1}{2}\eta_0 \Big(1 + \cos \frac{\pi k}{K}\Big), \qquad \eta_k^{\mathrm{WSD}} = \begin{cases} \eta_0, & k < 0.8K, \\ \eta_0 \dfrac{K - k}{0.2K}, & k \ge 0.8K. \end{cases}$$

Both start at η_0 and end at zero; cosine decays throughout the run, WSD only at its end. The warmup of its name, a short initial ramp that Section 35.5.4 discusses, is left out of the computation, because the quadratic model is stable at η_0 from the first step. WSD is popular for an operational reason: any checkpoint of its constant phase can be decayed to any budget, so the budget need not be fixed at launch. The quadratic model ranks the final losses of the shapes, not such operational merits.

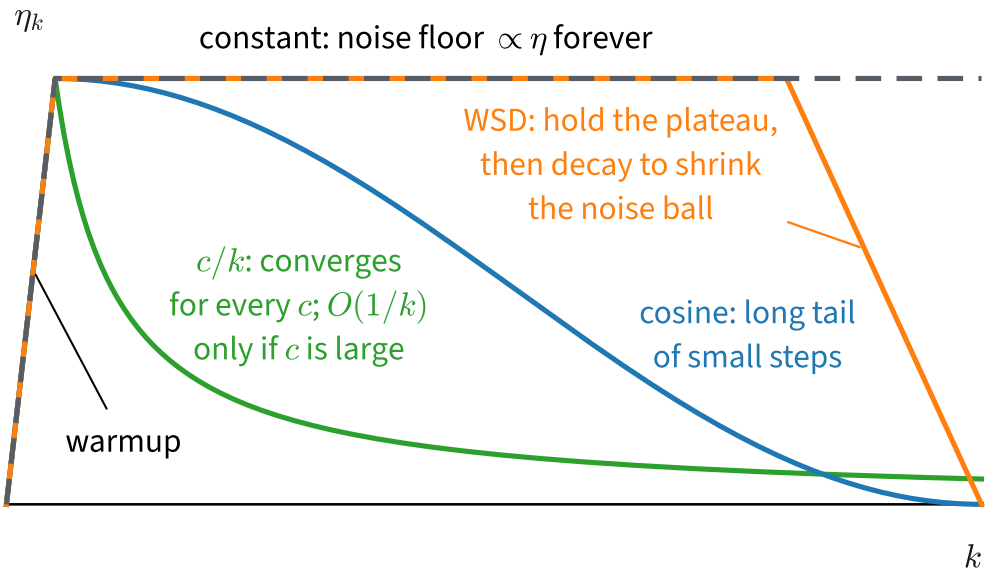


Figure 35.14: Schedule shapes over one budget K: constant, the $1/k$ decay (labelled c/k), cosine, and warmup–stable–decay (WSD), each preceded by a short linear warmup, which the computation below omits. The figure shows the step sequences only; the expected loss each produces on the quadratic model is computed below.

Which shape ends lowest? The reader may want to guess before reading on. The cell iterates the exact recursion with $\lambda = \sigma = 1$, $K = 2000$, $\eta_0 = 0.3$, $x_0 = 3$ and $k_0 = 50$ for the $1/k$ decay, prints the expected loss at 80% of the budget and at the end, and plots $\mathbb{E}f(x_k)$.

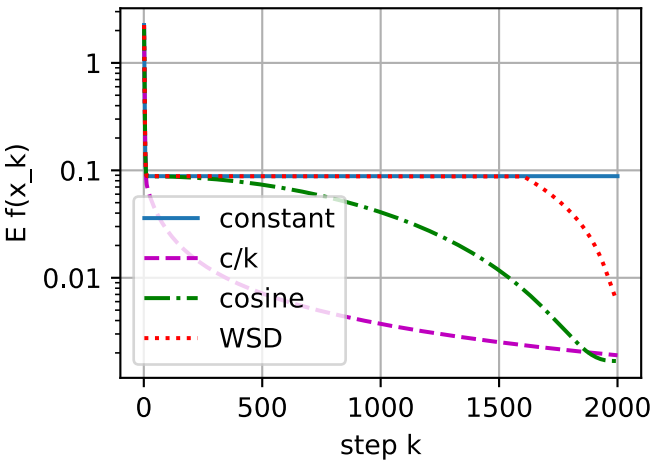
```
lam, sigma, K, eta0, x0 = 1.0, 1.0, 2000, 0.3, 3.0

def expected_loss(schedule, lam, sigma, K, x0):
    """E f(x_k) = lam e_k / 2 for k = 1..K from the exact second-moment recursion."""
    e, Ef = x0**2, np.empty(K)
    for k in range(K):
        eta = schedule(k)
        e = (1 - eta * lam)**2 * e + eta**2 * sigma**2
        Ef[k] = 0.5 * lam * e
    return Ef

schedules = {
    'constant': lambda k: eta0,
```

```
    'c/k': lambda k: eta0 / (1 + k / 50),
    'cosine': lambda k: 0.5 * eta0 * (1 + np.cos(np.pi * k / K)),
    'WSD': lambda k: eta0 if k < 0.8 * K else eta0 * (K - k) / (0.2 * K)}
traces = {name: expected_loss(s, lam, sigma, K, x0) for name, s in schedules.items()}
print(f'floor {eta0 * sigma**2 / (2 * (2 - eta0 * lam)):.4f}   '
      + ''.join(f'{name:>10s}' for name in traces))
for label, k in [('E f at 80%', int(0.8 * K) - 1), ('E f at end', K - 1)]:
    print(f'{label:15s}' + ''.join(f'{Ef[k]:10.2e}' for Ef in traces.values()))
d2l.plot(np.arange(1, K + 1), list(traces.values()), 'step k', 'E f(x_k)',
         yscale='log', legend=list(traces))
```

```
floor 0.0882     constant       c/k    cosine       WSD
E f at 80%       8.82e-02  2.36e-03  7.87e-03  8.82e-02
E f at end       8.82e-02  1.90e-03  1.68e-03  6.13e-03
```



The output is the same on every run. The constant step comes within a tenth of its loss floor 0.0882 by step 9 and stays there. WSD *is* the constant run until its decay begins, so at 80% of the budget it sits on the same floor. Its last 400 steps lower the loss 14.4 times, to 6.1×10^{-3}, so all of its improvement comes from the decay. Cosine and the $1/k$ decay end close together, at 1.7×10^{-3} and 1.9×10^{-3}, and WSD ends three to four times higher.

The value for the $1/k$ decay is the one the proposition predicts: here $c = 0.3 \cdot 50 = 15$, so $A = 225/29 = 7.8$ and $\mathbb{E}f(x_K) \approx \frac{1}{2}A/(K + k_0) = 1.9 \times 10^{-3}$. The ordering is exact for these parameters, and it need not carry over to a network. On the model itself the plateau does not help the final loss: lengthening WSD's decay phase lowers it steadily, down to 2.7×10^{-3} for a linear decay over the whole run, as exercise 8 computes.

35.4.3 Rates for a Fixed Budget

A decaying schedule is designed for a run of unlimited length. In practice the budget K is fixed in advance, and the question becomes which constant step to use and what loss it guarantees. Without strong convexity there is no contraction factor and no exact floor, so we bound instead of compute. Both results of this part have the same two-term shape: a transient that falls like $1/(\eta K)$ plus a floor that grows like η. The two terms balance at $\eta \propto 1/\sqrt{K}$, as Figure 35.15 shows.

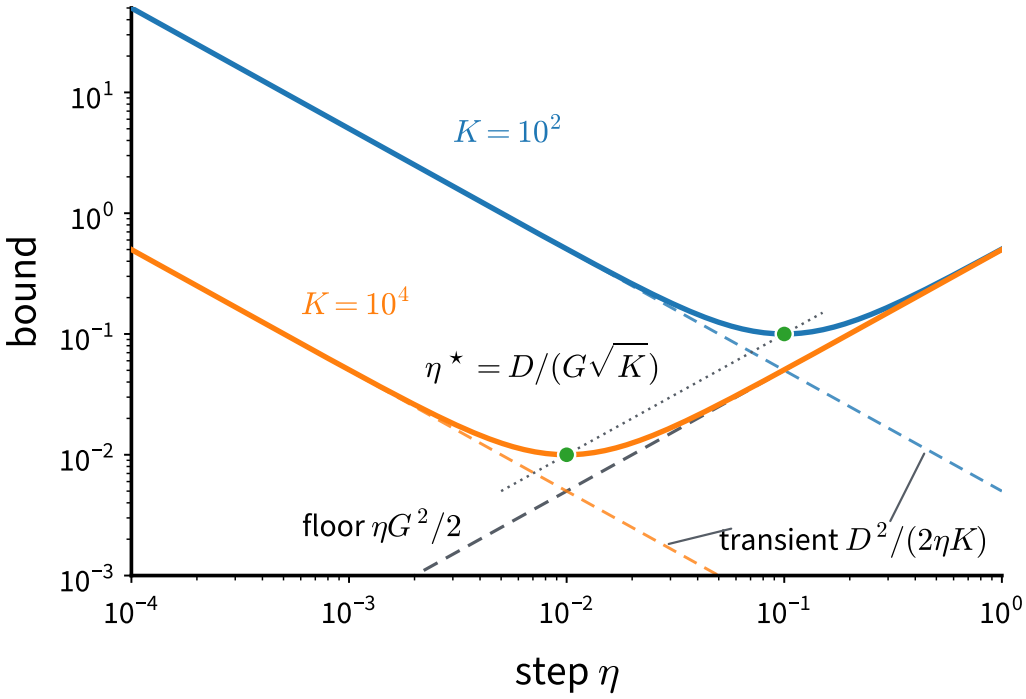


Figure 35.15: The two-term bound $D^2/(2\eta K) + \eta G^2/2$ of Equation 35.43 against the step η for $D = G = 1$ and two budgets. The transient falls with η, the floor rises with it, and the minimum $DG/\sqrt{K}$ sits at $\eta^\star = D/(G\sqrt{K})$: a $100\times$ larger budget moves the best step down $10\times$ and the bound down $10\times$.

35.4.3.1 Averaged Iterates on Convex Losses

The first result needs only convexity, through the subgradient inequality of Section 35.3: $f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{s}^\top(\mathbf{y}-\mathbf{x})$ for all $\mathbf{y}$ and every subgradient $\mathbf{s} \in \partial f(\mathbf{x})$, which is the gradient wherever f is differentiable.

Proposition (SGD on a convex loss, averaged iterate). *Let f be convex with a minimizer $\mathbf{x}^\star$ and $f^\star = f(\mathbf{x}^\star)$. Run $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\mathbf{g}_k$ from a point with $\|\mathbf{x}_0 - \mathbf{x}^\star\| \leq D$, where the stochastic gradient satisfies $\mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k] \in \partial f(\mathbf{x}_k)$ and $\mathbb{E}[\|\mathbf{g}_k\|^2 \mid \mathbf{x}_k] \leq G^2$. Then the average $\bar{\mathbf{x}}_K = \frac{1}{K}\sum_{k=0}^{K-1}\mathbf{x}_k$ satisfies*

$$\mathbb{E}\big[f(\bar{\mathbf{x}}_K)\big] - f^\star \;\leq\; \frac{D^2}{2\eta K} + \frac{\eta\,G^2}{2}, \tag{35.43}$$

and the step $\eta = D/(G\sqrt{K})$ gives $\mathbb{E}[f(\bar{\mathbf{x}}_K)] - f^\star \leq DG/\sqrt{K}$.

Proof. We follow the squared distance to the minimizer. Expanding the step,

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2 = \|\mathbf{x}_k - \mathbf{x}^\star\|^2 - 2\eta\,\mathbf{g}_k^\top(\mathbf{x}_k - \mathbf{x}^\star) + \eta^2\|\mathbf{g}_k\|^2.$$

Take the expectation given $\mathbf{x}_k$. The middle term becomes $-2\eta\,\mathbf{s}_k^\top(\mathbf{x}_k - \mathbf{x}^\star)$ with the subgradient $\mathbf{s}_k = \mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k]$, and the subgradient inequality at $\mathbf{y} = \mathbf{x}^\star$ gives $\mathbf{s}_k^\top(\mathbf{x}_k - \mathbf{x}^\star) \geq f(\mathbf{x}_k) - f^\star$. Everything rests on this inequality: the inner product of the gradient with the displacement from the minimizer is at least the loss gap, so a large gap forces the distance down. The last term is at most $\eta^2 G^2$. Taking total expectations and writing $r_k = \mathbb{E}\|\mathbf{x}_k - \mathbf{x}^\star\|^2$,

$$2\eta\,\mathbb{E}\big[f(\mathbf{x}_k) - f^\star\big] \;\leq\; r_k - r_{k+1} + \eta^2 G^2.$$

Summing over $k < K$ telescopes the distances, and $r_0 - r_K \leq D^2$, so $2\eta\sum_{k<K}\mathbb{E}[f(\mathbf{x}_k) - f^\star] \leq D^2 + K\eta^2 G^2$. Divide by $2\eta K$ and apply Jensen's inequality to the convex f, $f(\bar{\mathbf{x}}_K) \leq \frac{1}{K}\sum_k f(\mathbf{x}_k)$. The two terms of the bound are equal at $\eta = D/(G\sqrt{K})$, where each is $DG/(2\sqrt{K})$. ■

The transient $D^2/(2\eta K)$ is the distance that K steps of size η must cover, and the floor $\eta G^2/2$ is the noise that each step injects. The proof needs no smoothness, so the loss may have corners, as the hinge loss does. A hundred times the budget lowers both the best step and the bound tenfold. This is also the best possible order: over the class of the proposition, no method that sees only unbiased stochastic gradients with $\mathbb{E}\|\mathbf{g}\|^2 \leq G^2$ can guarantee less than a fixed positive multiple of $DG/\sqrt{K}$ in the worst case (Nemirovski and Yudin 1983).

Strong convexity changes the order. On the scalar quadratic, the best constant step for a budget K reaches $e_K = \Theta(\log K/K)$ (exercise 5), because the transient there decays geometrically, like ρ^{2K}, rather than like $1/(\eta K)$.

The proposition bounds the *average* of the iterates. The last iterate obeys a weaker bound, and for the last iterate the worst-case optimal schedule over a known budget decays linearly to zero (Defazio et al. 2023). The hypothesis $\mathbb{E}\|\mathbf{g}_k\|^2 \leq G^2$ also limits the result: it fails for a quadratic on all of $\mathbb{R}^d$, whose gradient is unbounded. It holds for Lipschitz losses with bounded features, such as the logistic loss without a quadratic penalty, or for iterates that

stay in a bounded set. As a worst-case bound it can be loose on a particular problem: on the logistic problem of this section, restricted to a ball, it overestimates the gap by about an order of magnitude, as exercise 10 measures.

35.4.3.2 Random Iterates on Nonconvex Losses

Networks have nonconvex losses *and* noisy gradients, and the results so far handle one of these at a time. For a function f whose gradient is L-Lipschitz (an L-smooth function), the graph lies below a parabola of curvature L tangent at any point,

$$f(\mathbf{y}) \;\le\; f(\mathbf{x}) + \nabla f(\mathbf{x})^\top(\mathbf{y}-\mathbf{x}) + \frac{L}{2}\|\mathbf{y}-\mathbf{x}\|^2 \quad \text{for all } \mathbf{x}, \mathbf{y}.$$

This quadratic ceiling bounds the loss after any step by the loss before it, a linear decrease, and a penalty for the length of the step. With exact gradients and $\eta = 1/L$, the ceiling gives $\min_{k<K} \|\nabla f(\mathbf{x}_k)\|^2 \le 2L\Delta/K$ without any convexity, where $f^\star$ is a lower bound of f and $\Delta = f(\mathbf{x}_0) - f^\star$ (Section 35.1). With noisy gradients on a quadratic, a fixed step parks on a floor. The theorem of Ghadimi and Lan (2013) combines the two. Model one SGD step as $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\mathbf{g}_k$, where the stochastic gradient satisfies

$$\mathbb{E}\left[\mathbf{g}_k \mid \mathbf{x}_k\right] = \nabla f(\mathbf{x}_k), \qquad \mathbb{E}\left[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|^2 \mid \mathbf{x}_k\right] \;\le\; \sigma^2. \tag{35.44}$$

On average the stochastic gradient points the right way, and its error has mean square at most σ^2 wherever the iterate is. A minibatch gradient meets both conditions with $\sigma^2 = \sup_{\mathbf{x}} \operatorname{tr}\Sigma(\mathbf{x})/b$ by Equation 35.38, provided this supremum is finite. It is finite for the bounded tanh model below; it is infinite for least squares on all of $\mathbb{R}^d$, whose gradient noise grows with the distance from the minimizer.

Proposition (SGD finds approximate stationary points). *Let f be L-smooth and bounded below by $f^\star$, write $\Delta = f(\mathbf{x}_0) - f^\star$, and run SGD with stochastic gradients satisfying Equation 35.44 and a constant step $\eta \le 1/L$. Then*

$$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla f(\mathbf{x}_k)\|^2\right] \;\le\; \frac{2\Delta}{\eta K} + L\eta\sigma^2, \tag{35.45}$$

and the step $\eta = \min\!\left(1/L,\ \sqrt{2\Delta/(L\sigma^2 K)}\right)$ gives, for an index R drawn uniformly from $\{0, \dots, K-1\}$ independently of the run,

$$\mathbb{E}\left[\|\nabla f(\mathbf{x}_R)\|^2\right] \;\le\; \frac{2L\Delta}{K} + 2\sigma\sqrt{\frac{2L\Delta}{K}}. \tag{35.46}$$

Proof. Apply the quadratic ceiling to the step actually taken, $\mathbf{y} - \mathbf{x} = -\eta\,\mathbf{g}_k$:

$$f(\mathbf{x}_{k+1}) \;\le\; f(\mathbf{x}_k) - \eta\,\nabla f(\mathbf{x}_k)^\top \mathbf{g}_k + \tfrac{L\eta^2}{2}\,\|\mathbf{g}_k\|^2.$$

Take the expectation given $\mathbf{x}_k$. Conditioning on the iterate does two things at once: it turns the middle term into $-\eta\,\|\nabla f(\mathbf{x}_k)\|^2$ by unbiasedness, and it splits the last term into signal

plus noise, $\mathbb{E}[\|\mathbf{g}_k\|^2 \mid \mathbf{x}_k] = \|\nabla f(\mathbf{x}_k)\|^2 + \mathbb{E}[\|\mathbf{g}_k - \nabla f(\mathbf{x}_k)\|^2 \mid \mathbf{x}_k] \le \|\nabla f(\mathbf{x}_k)\|^2 + \sigma^2$. Collecting,

$$\mathbb{E}\left[f(\mathbf{x}_{k+1}) \mid \mathbf{x}_k\right] \le f(\mathbf{x}_k) - \eta\left(1 - \tfrac{L\eta}{2}\right)\|\nabla f(\mathbf{x}_k)\|^2 + \tfrac{L\eta^2\sigma^2}{2}.$$

This is the descent lemma of gradient descent with an extra variance term $\frac{1}{2}L\eta^2\sigma^2$ per step. For $\eta \le 1/L$ the bracket is at least $\frac{1}{2}$. Take total expectations and sum over $k = 0, \dots, K-1$: the losses telescope to $\mathbb{E}[f(\mathbf{x}_K)] - f(\mathbf{x}_0) \ge -\Delta$, and dividing by $\eta K/2$ gives Equation 35.45. The average over k is the expectation at the uniformly random index R. For the balanced step there are two cases. If $\sqrt{2\Delta/(L\sigma^2 K)} \le 1/L$, the two terms of Equation 35.45 are equal, each $\sigma\sqrt{2L\Delta/K}$. Otherwise $\eta = 1/L$ and $\sigma^2 < 2L\Delta/K$, so the bound is $2L\Delta/K + \sigma^2 \le 2L\Delta/K + \sigma\sqrt{2L\Delta/K}$. Both are at most the stated bound. ■

With $\sigma = 0$ the second term vanishes, and the deterministic rate $2L\Delta/K$ returns. With noise and $K > 2L\Delta/\sigma^2$, the case in which $\eta < 1/L$, the second term is the larger one, and it decays like $K^{-1/2}$. With $L = \Delta = \sigma = 1$ and $K = 10^4$, the step is $\eta = 0.014$ and the bound is $2 \times 10^{-4} + 0.028$: the noise term is 140 times the deterministic rate $2L\Delta/K$. The bound controls the *squared* gradient norm: a hundred times the budget gives ten times less $\mathbb{E}\|\nabla f\|^2$, but a ten times smaller gradient norm needs 10^4 times the budget. With a biased estimator the proof leaves an extra inner product, and the bound acquires a term that does not shrink with K (exercise 4).

Why does the guarantee speak of a *random* iterate? With exact gradients the best iterate can be identified, since $\|\nabla f(\mathbf{x}_k)\|$ is observed. With stochastic gradients it is not, so an algorithm cannot pick the iterate with the smallest gradient; drawing R at random is a choice it can make, and the average over the run is what that choice controls.

Compared with the convex case, the bound now controls a squared gradient at a random iterate instead of a function value at the average. Without convexity a small gradient is all that can be certified, and a stationary point need not be a minimizer. This rate is optimal as well: no method with access only to such stochastic gradients can guarantee a better order in the worst case over this class (Arjevani et al. 2023).

35.4.3.3 The Rate on a Single tanh Neuron

The proposition is an upper bound, so an experiment cannot confirm its rate, only fail to violate it. We measure the random-iterate quantity, and beside it the best iterate, on a small nonconvex problem, a single tanh neuron fit by least squares to noisy labels: $n = 64$ examples in five dimensions and

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\left(\tanh(\mathbf{a}_i^\top \mathbf{x}) - y_i\right)^2.$$

Each term is bounded and smooth, and none is convex: tanh flattens for large arguments, so the loss has plateaus. The cell defines the data and the exact gradient for a batch of iterates, one per row of $\mathbf{X}$.

```
n_nc, d_nc = 64, 5                              # a small nonconvex problem
A_nc = rng.normal(size=(n_nc, d_nc))
y_nc = np.tanh(A_nc @ rng.normal(size=d_nc)) + 0.5 * rng.normal(size=n_nc)

def full_grad(X, A, y):
    """Exact gradient of mean_i (tanh(a_i^T x) - y_i)^2 at each row x of X."""
    t = np.tanh(X @ A.T)                        # (runs, n)
    return 2 * ((t - y) * (1 - t**2)) @ A / len(y)
```

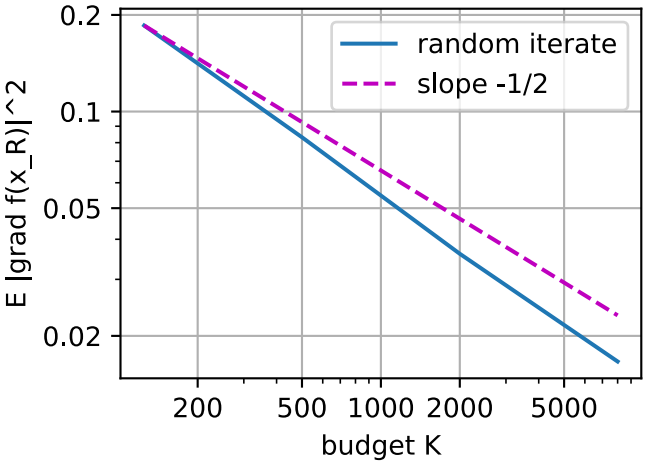
SGD with single-example gradients runs for the budgets $K = 125, 500, 2000, 8000$ at $\eta = 0.5/\sqrt{K}$, twenty runs at a time as the rows of `X`. The cell records the average of $\|\nabla f(\mathbf{x}_k)\|^2$ along each run, which is the random-iterate quantity, and its minimum, the best iterate, and prints their log–log slopes against K.

```
runs, Ks, avg, best = 20, np.array([125, 500, 2000, 8000]), [], []
for K in Ks:
    eta = 0.5 / np.sqrt(K)                      # the theorem's eta ~ 1/sqrt(K)
    X = np.zeros((runs, d_nc))                  # one row per independent run
    total, low = np.zeros(runs), np.full(runs, np.inf)
    for k in range(K):
        g2 = (full_grad(X, A_nc, y_nc)**2).sum(axis=1)   # |grad f(x_k)|^2
        total, low = total + g2, np.minimum(low, g2)
        i = rng.integers(0, n_nc, size=runs)             # one example per run
        t = np.tanh((X * A_nc[i]).sum(axis=1))
        X -= eta * (2 * (t - y_nc[i]) * (1 - t**2))[:, None] * A_nc[i]
    avg.append(total.mean() / K), best.append(low.mean())

def slope(v):
    return np.polyfit(np.log(Ks), np.log(v), 1)[0]

for name, v in [('random iterate', avg), ('best iterate', best)]:
    print(f'{name:15s}' + ' '.join(f'{u:.1e}' for u in v) + f'   slope {slope(v):.2f}')
d2l.plot(Ks, [avg, avg[0] * (Ks / Ks[0])**-0.5], 'budget K', 'E |grad f(x_R)|^2',
         xscale='log', yscale='log', legend=['random iterate', 'slope -1/2'])
```

```
random iterate 1.9e-01 8.3e-02 3.6e-02 1.7e-02   slope -0.58
best iterate   9.0e-03 2.1e-03 3.9e-04 4.9e-05   slope -1.25
```



The random-iterate quantity falls by a factor of about 12 (between roughly 9 and 16 across runs) for a 64 times larger budget, a log–log slope of about -0.6. The bound falls with slope $-1/2$; the measurement falls a little faster, as an upper bound permits. The best iterate of each run falls much faster, with a slope near -1 or steeper, but no algorithm that sees only noisy gradients can return it.

35.4.4 Variance Reduction

35.4.4.1 Control Variates

A decaying step removes the floor by shrinking η, at the cost of slower progress. On a *finite* sum there is another way: remove the noise itself. At the minimizer the full gradient is zero, but each $\nabla f_i(\mathbf{x}^\star)$ is not, and these nonzero example gradients are the noise that a fixed step keeps injecting.

One case needs no remedy. When a model fits every example exactly, the example gradients vanish at the minimizer, and on a strongly convex loss a fixed step converges linearly with no floor (exercise 9). Large networks can fit their training sets, which is one reason why constant-step SGD works better on them than the floor formula suggests.

Otherwise the noise must be cancelled. The idea is to subtract from the noisy $\nabla f_i(\mathbf{x})$ a quantity that moves with it and whose mean we know, and then add the mean back. A quantity of this kind is called a **control variate**. The baseline of policy gradients (Section 14.7) is the control variate that this book relies on most. On a finite sum another one is available: the gradient $\nabla f_i(\tilde{\mathbf{x}})$ of the same example at a stored snapshot $\tilde{\mathbf{x}}$, whose mean over i is the full gradient $\nabla f(\tilde{\mathbf{x}})$, computed once per pass over the data. The **SVRG** estimator (stochastic variance-reduced gradient) (Johnson and Zhang 2013) is

$$\tilde{\mathbf{g}}_i(\mathbf{x}) = \nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}}) + \nabla f(\tilde{\mathbf{x}}). \tag{35.47}$$

It corrects the full gradient at the snapshot by the change of one example's gradient between the snapshot and the current point. For a uniformly drawn index it is unbiased,

$$\mathbb{E}_i\big[\tilde{\mathbf{g}}_i(\mathbf{x})\big] = \nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}}) + \nabla f(\tilde{\mathbf{x}}) = \nabla f(\mathbf{x}),$$

and its advantage is its variance.

Proposition (the variance is bounded by the distance to the snapshot). *If each f_i has an L_i-Lipschitz gradient and $L_{\max} = \max_i L_i$, then for a uniformly drawn index i*

$$\mathbb{E}_i\left\|\tilde{\mathbf{g}}_i(\mathbf{x}) - \nabla f(\mathbf{x})\right\|^2 \;\le\; \mathbb{E}_i\left\|\nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}})\right\|^2 \;\le\; L_{\max}^2\,\|\mathbf{x} - \tilde{\mathbf{x}}\|^2. \tag{35.48}$$

Proof. The left side is the variance of the random vector $\nabla f_i(\mathbf{x}) - \nabla f_i(\tilde{\mathbf{x}})$, whose mean is $\nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}})$; a variance is at most the second moment. Each term of the second moment is at most $L_i^2\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$ by the Lipschitz gradient. ■

The noise power of the estimator is at most a multiple of the squared distance between the iterate and the snapshot, so it vanishes as both approach $\mathbf{x}^\star$, and a fixed step no longer has a floor. Intuitively, two gradients of the same example at nearby points share most of their noise. Their difference keeps the change, which is small, and discards the shared part, which is large.

35.4.4.2 SVRG and SAGA

The snapshot must follow the iterate, or the distance in Equation 35.48 stops shrinking, and a full gradient at every step would defeat the purpose. SVRG therefore works in epochs. At the start of each epoch it stores a snapshot $\tilde{\mathbf{x}}$ and computes the full gradient $\nabla f(\tilde{\mathbf{x}})$; it then takes m inner steps $\mathbf{x} \leftarrow \mathbf{x} - \eta\,\tilde{\mathbf{g}}_i(\mathbf{x})$ with fresh random indices, and the last inner iterate becomes the next snapshot. An epoch costs $n + 2m$ example gradients. The guarantee of Johnson and Zhang (2013) (Theorem 1) asks that every f_i be convex with an $L_{\max}$-Lipschitz gradient, that f be μ-strongly convex, and that $\eta < 1/(4L_{\max})$ with an inner loop long enough for the theorem's contraction factor to be below one. The expected gap $\mathbb{E}[f(\tilde{\mathbf{x}})] - f^\star$ at the snapshot then falls geometrically with the number of epochs. The theorem takes a random inner iterate as the next snapshot. Our code takes the last one, which in practice behaves the same.

We test SVRG on ridge regression, a finite sum with $f_i(\mathbf{x}) = \frac{1}{2}(\mathbf{a}_i^\top\mathbf{x} - y_i)^2 + \frac{0.1}{2}\|\mathbf{x}\|^2$, $n = 300$ examples and $d = 12$ parameters. Its minimizer solves the normal equations, and each f_i has an L_i-Lipschitz gradient with $L_i = \|\mathbf{a}_i\|^2 + 0.1$. The cell defines the data and the component and full gradients, and prints $L_{\max}$.

```
n_r, d_r, reg_r = 300, 12, 0.1                  # ridge regression, a finite sum
A_r = rng.standard_normal((n_r, d_r))
y_r = A_r @ rng.standard_normal(d_r) + 0.5 * rng.standard_normal(n_r)
H_r = A_r.T @ A_r / n_r + reg_r * np.eye(d_r)     # Hessian of f
x_star_r = np.linalg.solve(H_r, A_r.T @ y_r / n_r)

def component_grad(x, i, A, y, reg):
    """Gradient of f_i(x) = (a_i^T x - y_i)^2 / 2 + reg |x|^2 / 2 (i: index or array)."""
    return A[i] * (A[i] @ x - y[i])[..., None] + reg * x

def full_grad_sum(x, A, y, reg):
    """Gradient of the finite sum f = mean_i f_i."""
    return A.T @ (A @ x - y) / len(y) + reg * x

def objective(x, A, y, reg):
```

```
    """The ridge objective f(x)."""
    return 0.5 * np.mean((A @ x - y)**2) + 0.5 * reg * x @ x

L_max = (A_r**2).sum(axis=1).max() + reg_r       # largest per-example smoothness
print(f'L_max = {L_max:.0f}, 1 / (4 L_max) = {1 / (4 * L_max):.4f}')
```

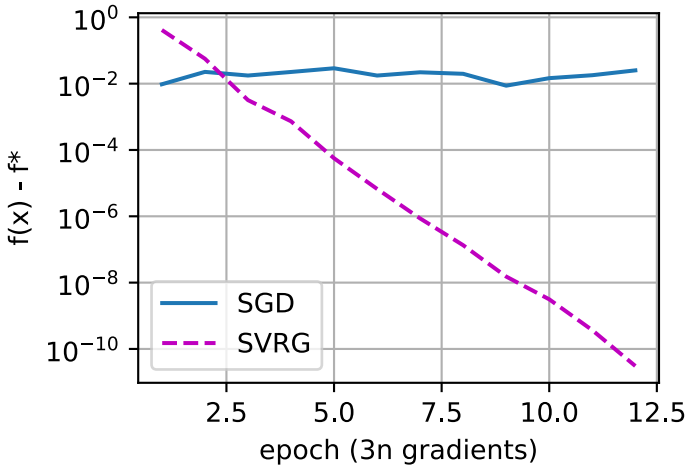
```
L_max = 29, 1 / (4 L_max) = 0.0088
```

$L_{\max}$ is about 30 (typically between 25 and 40 across draws of the data), so the theorem asks for η below $1/(4L_{\max})$, the second number printed, between about 0.006 and 0.01.

We run SVRG with $m = n$ and SGD with single-example gradients at the same fixed $\eta = 0.02$, more than twice the theorem's limit, for 12 epochs of $3n$ example gradients each. For SGD the first part of this section makes a prediction that the run can miss: its floor should be the sum of the loss floors $\eta\sigma_i^2/(2(2-\eta\lambda_i))$ over the eigenvectors of the Hessian, with σ_i^2 the variance of the example gradients at $\mathbf{x}^\star$ along eigenvector i.

```
eta, epochs = 0.02, 12
f_star_r = objective(x_star_r, A_r, y_r, reg_r)
snap, gaps_svrg = np.zeros(d_r), []
for _ in range(epochs):                          # n + 2n example gradients per epoch
    g_snap = full_grad_sum(snap, A_r, y_r, reg_r)
    x = snap.copy()
    for i in rng.integers(0, n_r, size=n_r):
        x -= eta * (component_grad(x, i, A_r, y_r, reg_r)
                    - component_grad(snap, i, A_r, y_r, reg_r) + g_snap)
    snap = x
    gaps_svrg.append(objective(snap, A_r, y_r, reg_r) - f_star_r)
x, gaps_sgd = np.zeros(d_r), []
for _ in range(epochs):
    for i in rng.integers(0, n_r, size=3 * n_r):  # the same 3n budget per epoch
        x -= eta * component_grad(x, i, A_r, y_r, reg_r)
    gaps_sgd.append(objective(x, A_r, y_r, reg_r) - f_star_r)
G_r = component_grad(x_star_r, np.arange(n_r), A_r, y_r, reg_r)  # noise at x*
ev, Q = np.linalg.eigh(H_r)                      # Hessian eigenvalues, eigenvectors
s2 = ((G_r @ Q)**2).mean(axis=0)                 # noise variance per eigenvector
floor = (eta * s2 / (2 * (2 - eta * ev))).sum()  # the loss floors add
print(f'SGD: predicted floor {floor:.1e}, mean gap in epochs 4-12 '
      f'{np.mean(gaps_sgd[3:]):.1e}')
print('SVRG: gap after epochs 1, 4, 12: '
      + ', '.join(f'{gaps_svrg[e - 1]:.0e}' for e in (1, 4, 12)))
d2l.plot(np.arange(1, epochs + 1), [gaps_sgd, gaps_svrg], 'epoch (3n gradients)',
         'f(x) - f*', yscale='log', legend=['SGD', 'SVRG'])
```

```
SGD: predicted floor 1.9e-02, mean gap in epochs 4-12 2.0e-02
SVRG: gap after epochs 1, 4, 12: 4e-01, 7e-04, 3e-11
```



SGD drops quickly and then fluctuates at a few times 10^{-2}, where the floor formula of the first part puts it. The plateau usually lies between about 10% below and 40% above the prediction, more often above. The formula counts only the noise at $\mathbf{x}^\star$, and the example gradients spread more away from it. SVRG loses early and wins late. After its first epoch, which takes only n steps with a control variate anchored at the starting point, its gap is still far above SGD's. From then on it falls by a factor of about eight per epoch (between roughly five and thirteen across runs), to between about 10^{-13} and 10^{-8} by epoch 12. It does so at a step more than twice the theorem's limit, so on this problem the theorem's step condition is not necessary. Three hypotheses carry the result: a finite sum, repeated passes over it, and strong convexity for the linear rate. On streaming data there is no sum to store.

SAGA (Defazio et al. 2014) replaces the snapshot by a table $\mathbf{t}_1, \dots, \mathbf{t}_n$ that holds the most recent gradient computed for each example. Its estimator is $\nabla f_i(\mathbf{x}) - \mathbf{t}_i + \frac{1}{n}\sum_j \mathbf{t}_j$, after which $\mathbf{t}_i$ is overwritten by $\nabla f_i(\mathbf{x})$. SAGA needs one new example gradient per step and no full passes, but its table holds $n \times d$ numbers, or one scalar per example for a generalized linear model (exercise 3).

On deep networks, unfortunately, these estimators have not given consistent gains: the control variate goes stale when the loss landscape changes under the iterate, for instance through data augmentation. A second limit has been with us since the valley: all of these methods share one step size across coordinates, which the stiff coordinate of the running example capped at $2/1000$. The next section lets each coordinate set its own step from the gradients it has seen.

35.4.5 Summary

A minibatch gradient is unbiased, and its error has power $\operatorname{tr}\Sigma/b$. On a quadratic, a fixed step η parks the second moment of each coordinate on the floor $\eta\sigma^2/(\lambda(2-\eta\lambda))$, reached geometrically at the rate $(1-\eta\lambda)^2$, and the loss floors $\eta\sigma^2/(2(2-\eta\lambda))$ add over coordinates. The floor is linear in η while the transient decays faster for larger η, so no fixed step reaches the minimizer, and every remedy trades the two. A Robbins–Monro schedule removes the floor, and $\eta_k = c/k$ reaches $O(1/k)$ only for $c > 1/(2\mu)$. For a fixed budget K, a constant step $\eta \propto 1/\sqrt{K}$ gives $DG/\sqrt{K}$ for the averaged iterate of a convex loss and, for large K, $O(\sigma\sqrt{L\Delta/K})$ for the squared gradient at a random iterate of a smooth nonconvex loss. On

a finite sum, the SVRG estimator is unbiased with variance at most $L_{\max}^2 \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$, so a fixed step converges linearly on a strongly convex loss.

35.4.6 Exercises

1. code **Noise scale and critical batch size.** ▣ How much does one minibatch step lower the loss, and how does the gain depend on b? Write $\mathbf{g} = \nabla f(\mathbf{x})$ and $\mathbf{H} = \nabla^2 f(\mathbf{x})$, and approximate the loss after the step $\mathbf{x} - \eta\,\hat{\mathbf{g}}_b$ of Equation 35.37 by its second-order Taylor expansion at $\mathbf{x}$. The analysis follows McCandlish et al. (2018) .
   1. Use the covariance Σ/b of $\hat{\mathbf{g}}_b$, established in the proof of Equation 35.38, to show that the expected loss after the step is minimized by $\eta_{\text{opt}}(b) = \eta_{\max}/(1 + B_{\text{noise}}/b)$, with expected improvement $\Delta_{\max}/(1 + B_{\text{noise}}/b)$, where
   $$\eta_{\max} = \frac{\|\mathbf{g}\|^2}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}, \qquad \Delta_{\max} = \frac{\|\mathbf{g}\|^4}{2\,\mathbf{g}^\top \mathbf{H}\mathbf{g}}, \qquad B_{\text{noise}} = \frac{\operatorname{tr}(\mathbf{H}\Sigma)}{\mathbf{g}^\top \mathbf{H}\mathbf{g}}.$$
   2. The linear scaling rule of Goyal et al. (2017) multiplies η by the same factor as b. For which batch sizes does this rule follow from $\eta_{\text{opt}}(b)$? What does a larger batch gain per step once $b \gg B_{\text{noise}}$?
   3. For $\mathbf{H}$ proportional to the identity, B_{noise} reduces to $B_{\text{simple}} = \operatorname{tr}\Sigma/\|\mathbf{g}\|^2$, the gradient-noise scale b_{noise} of the text. Compute B_{simple} and B_{noise} for the logistic-regression problem of this section at $\mathbf{x} = \mathbf{0}$ and after 5, 10, and 20 full-gradient steps with $\eta = 0.5$, using `example_grads` and the Hessian $\mathbf{H}(\mathbf{x}) = \frac{1}{n}\sum_i s_i(1 - s_i)\,\mathbf{a}_i\mathbf{a}_i^\top + 0.1\,\mathbf{I}$ with $s_i = 1/(1 + e^{y_i \mathbf{a}_i^\top \mathbf{x}})$. How does the noise scale change as the iterates approach the optimum? Treat B_{noise} as the critical batch size that separates the two regimes of (b). How should the batch size change over a training run?
2. code **The constant in a $1/k$ schedule.** ▣▣ The text found the constant A of Equation 35.42 to leading order. This exercise makes it rigorous and carries it to the running example.
   1. Let $t = k + k_0$ and suppose $u_{k+1} \le (1 - a/t)\,u_k + q/t^2$ for all $k \ge 0$, with $a > 1$, $q > 0$, $u_k \ge 0$ and $k_0 \ge a$. Prove by induction that $u_k \le \nu/(k + k_0)$ with $\nu = \max\{q/(a-1),\ k_0 u_0\}$. (Hint: $1 - a/t \ge 0$, and $1/t - 1/t^2 \le 1/(t+1)$.)
   2. Apply the lemma to the recursion $e_{k+1} = (1 - c\mu/t)^2 e_k + c^2\sigma^2/t^2$. Show that $(1 - c\mu/t)^2 \le 1 - (2c\mu - (c\mu)^2/k_0)/t$ for $t \ge k_0$, so that any $2c\mu > 1$ gives $e_k = O(1/k)$ once k_0 is large enough. Identify a and q, and show that $q/(a-1) \to A$ as $k_0 \to \infty$.
   3. For $2c\mu < 1$, show $e_k = \Theta(k^{-2c\mu})$ from the product $\prod_{j<k}(1 - c\mu/(j + k_0))^2$ and the unrolled noise sum.
   4. Iterate the recursion with $\mu = \sigma = 1$, $k_0 = 10$ and $e_0 = 1$ up to $k = 10^6$ for $c \in \{0.1, 0.25, 0.5, 1, 2\}$. Report the log–log slope between $k = 10^5$ and 10^6 and the value of $k\,e_k$ at 10^6. Which costs more, a constant twice the minimizer of A or half of it?
   5. On the running example $\frac{1}{2}(x_1^2 + 1000\,x_2^2)$ with unit noise per coordinate, stability at the first step forces $c/k_0 < 2/1000$. Take $c = 1/\mu = 1$, $k_0 = 1000$, $\mathbf{x}_0 = (1, 1)$ and $K = 10^4$. Compute the expected loss by the two-coordinate recursion (it is about 4.2×10^{-3}) and compare it with the constant step $\eta = 10^{-3}$ (7.5×10^{-4}) and with cosine decay from the same η_0 (1.0×10^{-4}). Which coordinate does the $1/k$ decay starve, and why does the condition number reappear in k_0?

3. code **SVRG inner loop and SAGA.** Use the ridge data and the functions `component_grad`, `full_grad_sum` and `objective` of this section with a budget of $36n = 10{,}800$ example-gradient evaluations, the budget behind the 12 reported epochs of the SVRG cell. Measure each run by the number of evaluations it needs to bring the gap below 10^{-10}, checking once per epoch.
   1. Let the inner loop run m steps, so that one epoch costs $n + 2m$ evaluations, and vary $m \in \{10, 30, 75, 300, 1200\}$ at $\eta \in \{0.01, 0.02\}$. Which m is best at each step size? Explain the trade-off between amortizing the snapshot gradient and letting the control variate go stale, and why the best m moves with η.
   2. Implement SAGA: keep a table with the most recent gradient of every example, filled at $\mathbf{x}_0$ for n evaluations, and use $\nabla f_i(\mathbf{x}) - \mathbf{t}_i + \frac{1}{n}\sum_j \mathbf{t}_j$ as the estimator, where $\mathbf{t}_i$ is the table entry of example i. Compare SAGA with the best SVRG run at both step sizes. How many numbers does the table hold for this problem, and how many would suffice if you stored only the scalar $\mathbf{a}_i^\top \mathbf{x} - y_i$ per example?
4. **Biased gradient estimators.** ▨ Keep the variance bound of Equation 35.44, now measured around the conditional mean, but replace unbiasedness by $\mathbb{E}[\mathbf{g}_k \mid \mathbf{x}_k] = \nabla f(\mathbf{x}_k) + \delta_k$ with $\|\delta_k\| \le \delta$.
   1. Redo the proof of Equation 35.45 for $\eta \le 1/L$ and show
   $$\frac{1}{K}\sum_{k=0}^{K-1} \mathbb{E}\left[\|\nabla f(\mathbf{x}_k)\|^2\right] \;\le\; \frac{2\Delta}{\eta K} + L\eta\sigma^2 + \delta^2. \tag{35.49}$$
   Use $L\eta \le 1$ on the term $\frac{L\eta^2}{2}\|\nabla f(\mathbf{x}_k) + \delta_k\|^2$ before expanding it.
   2. Show that the term δ^2 cannot be removed: give a one-dimensional f and a constant bias for which SGD with $\sigma = 0$ converges to a point where $|f'|^2 = \delta^2$.
   3. A single minibatch drawn uniformly without replacement is still an unbiased estimator of the full gradient. Which of the following estimators are biased in this sense: the example gradient at a step of random reshuffling (Section 9.3), which visits every example once per epoch in a fresh random order; a stochastic gradient clipped to a fixed maximum norm; the gradient evaluated at the iterate of τ steps earlier, as in asynchronous training?
5. code **The best constant step for a budget.** ▨ On the scalar model with $\lambda = \sigma = 1$ and $e_0 = 1$, Equation 35.39 gives $e_K = (1-\eta)^{2K}(1 - V) + V$ with $V = \eta/(2-\eta)$.
   1. Show that the minimizing step satisfies $\eta^\star = \Theta(\log K/K)$ and that $\min_\eta e_K = \Theta(\log K/K)$. (Hint: use $(1-\eta)^{2K} \approx e^{-2\eta K}$ and $V \approx \eta/2$, and set the derivative to zero.)
   2. Minimize numerically for $K = 10^2, 10^3, 10^4$ (you should find $\eta^\star \approx 0.030, 0.0041, 0.0005$ and $e_K \approx 1.7\times10^{-2}, 2.3\times10^{-3}, 2.9\times10^{-4}$), and check that $\eta^\star K/\log K$ and $e_K K/\log K$ stay of order one.
   3. Apply the budget rule $\eta = 1/\sqrt{K}$ of the convex bound to this quadratic anyway. Show that $e_K \approx 1/(2\sqrt{K})$, worse than (b) by a factor $\Theta(\sqrt{K}/\log K)$.
   4. Which term of the bound Equation 35.43 does strong convexity improve, and why does that change the best step from $\propto 1/\sqrt{K}$ to $\propto \log K/K$?
6. code **Iterate averaging.** ▨ Polyak and Juditsky (1992) showed that averaging the iterates of a slowly decaying or constant-step run reaches the best possible constant. Consider the scalar model with a constant step $\eta \in (0, 2/\lambda)$, started in its stationary distribution.
   1. Show that $\mathrm{Cov}(x_j, x_k) = V\rho^{|j-k|}$.

2. Show that the average $\bar{x}_K = \frac{1}{K}\sum_{k<K} x_k$ has $\mathbb{E}[\bar{x}_K^2] = \frac{V}{K}\cdot\frac{1+\rho}{1-\rho} - O(K^{-2})$, and that $V(1+\rho)/(1-\rho) = \sigma^2/\lambda^2$ for every such η: to leading order the averaged iterate does not depend on the step.
3. Check by simulation with $K = 2000$ and $4{,}000$ runs at $\eta = 0.3$ and $\eta = 0.1$, with $\lambda = \sigma = 1$. You should find $\mathbb{E}[\bar{x}_K^2] \approx 5\times10^{-4}$ in both cases, against last-iterate floors of 0.18 and 0.053.
4. Started at $x_0 = 3$ instead, the average remembers the transient. Show that its contribution to $\mathbb{E}[\bar{x}_K^2]$ is about $(x_0/(\eta\lambda K))^2$, and decide when averaging only the second half of the run is better.
5. Compare with the best $1/k$ schedule, whose constant is $A = \sigma^2/\mu^2$. What does each of the two routes to the rate $\sigma^2/(\mu^2 K)$ require?

7. **Which Robbins–Monro condition does the quadratic need?** ▣ On the scalar model with steps η_k and $\rho_j = 1 - \eta_j\lambda$:
   1. Unroll the recursion to $e_k = \prod_{j<k}\rho_j^2\, e_0 + \sigma^2\sum_{j<k}\eta_j^2\prod_{j<i<k}\rho_i^2$.
   2. Show that $e_k \to 0$ whenever $\eta_k \to 0$, $\sum_k \eta_k = \infty$ and $0 < \eta_k < 2/\lambda$: the square-summable condition is not needed for convergence in mean square on this model. (Hint: split the sum at an index m beyond which $\eta_j \le \epsilon$, and compare the tail with the floor of the step ϵ.)
   3. For $\eta_k = c/\sqrt{k}$, show that $e_k \approx c\sigma^2/(2\lambda\sqrt{k})$, and check numerically with $\lambda = \sigma = 1$ and $c = 0.5$ (2.5×10^{-3} at $k = 10^4$, 2.5×10^{-4} at 10^6).
   4. In words: what does $\sum_k \eta_k^2 < \infty$ provide in the general theorem, which claims convergence for almost every run with noise that need not be additive? (Consider the accumulated noise $\sum_k \eta_k\xi_k$.)
8. code **The decay fraction of WSD.** Use the exact recursion of the schedule cell ($\lambda = \sigma = 1$, $K = 2000$, $\eta_0 = 0.3$, $x_0 = 3$).
   1. Tabulate the final expected loss of WSD for the decay fractions $0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 0.9, 1$. (It falls steadily from 1.24×10^{-2} to 2.73×10^{-3}.)
   2. A linear decay over the whole run ends at 2.73×10^{-3}, cosine at 1.68×10^{-3}. Explain the difference from the shapes of the two schedules near the end of the run.
   3. On this model the plateau gives no advantage in final loss. State what it gives in practice, and why a model with one fixed curvature and stationary noise cannot show it.
9. code **Interpolation and linear convergence.** ▣▣ Suppose each f_i is convex with an L_i-Lipschitz gradient, f is μ-strongly convex, and every f_i is minimized at the same point $\mathbf{x}^\star$, as when a model fits every training example exactly. The analysis follows Vaswani et al. (2019) .
   1. Show that $\mathbb{E}_i\|\nabla f_i(\mathbf{x})\|^2 \le 2L_{\max}(f(\mathbf{x}) - f^\star)$. (Hint: a convex L-smooth h minimized at $\mathbf{x}^\star$ satisfies $\|\nabla h(\mathbf{x})\|^2 \le 2L(h(\mathbf{x}) - h(\mathbf{x}^\star))$.)
   2. Deduce that single-example SGD with a constant $\eta \le 1/L_{\max}$ satisfies $\mathbb{E}\|\mathbf{x}_{k+1} - \mathbf{x}^\star\|^2 \le (1-\eta\mu)\,\mathbb{E}\|\mathbf{x}_k - \mathbf{x}^\star\|^2$: linear convergence with no floor. What is $\operatorname{tr}\Sigma(\mathbf{x}^\star)$ in this case?
   3. Check it on least squares with $n = 50$ examples and $d = 100$ parameters, $\eta = 1/L_{\max}$ and $\mathbf{x}_0 = \mathbf{0}$. The iterates stay in the row space of $\mathbf{A}$, where f is strongly convex, so (b) applies there. Show that the gap falls linearly with consistent labels and also with labels corrupted by noise of standard deviation 0.5, and explain why the noise does not matter here.
   4. Repeat with $n = 200$ and $d = 100$. Now the noisy labels bring the floor back.

Why?

10. code **The convex bound on the logistic problem.** For the logistic-regression problem of this section:
    1. Set $D = \|\mathbf{x}^\star\|$ for the reference optimum `x_star`, and bound G^2 on the ball of radius $2D$ around the origin from $\|\nabla f_i(\mathbf{x})\| \le \|\mathbf{a}_i\| + 0.1\,\|\mathbf{x}\|$.
    2. Run SGD with $b = 1$, $\eta = D/(G\sqrt{K})$ and $K = 4000$, check that the iterates stay in the ball, and compare the gap of the averaged iterate with the bound $DG/\sqrt{K}$. By what factor is the bound loose?
    3. Which inequalities of the proof are loose here? The regularizer makes f strongly convex with $\mu \ge 0.1$; what rate does that allow instead?
    4. Compare the last iterate with the average at the same η.

35.5 Adaptive Methods

The transformers in this book, like most large networks, are trained with Adam or its variant AdamW (Section 9.5) rather than with plain stochastic gradient descent. One reason is the step size. The methods of Section 35.4 use one step size for every coordinate, and their schedules change that step only over time. A network has 10^7 to 10^{11} coordinates whose curvatures differ by orders of magnitude and are not known in advance. With one step size for all of them, the stiffest coordinate sets the step and the others crawl. This section asks how an optimizer can choose a step size for each coordinate from the gradients alone, and when that choice fails.

The chapter's running example shows how much is at stake. Take the valley $f(\mathbf{x}) = \frac{1}{2}(x_1^2 + 1000\,x_2^2)$ and start at $(1, 1)$. Gradient descent with the best single step size, $\eta = 2/1001$, needs 6,160 steps to bring f below 10^{-8}, because a step small enough to keep x_2 stable is a thousand times too small for x_1. The heavy-ball method of Section 35.1, which adds a fraction of the previous step to each new one, needs 296 steps. An optimizer that gives each coordinate its own step size, estimated only from the gradients it has seen, needs a few dozen. More telling than the count is what it does not depend on: make the valley a hundred times narrower still, and the count grows by six steps.

Counts like these are easy to produce and easy to misread, and a good part of this section goes into making the comparison fair. Heavy ball will run at its optimal settings, Adam at the best of a sweep of step sizes, and the valley will also be turned by 45°. Adam does not come out of that comparison as well as the first count suggests.

The idea itself is simple: divide each coordinate's step by the root mean square of its own past gradients, as AdaGrad, RMSProp and Adam do. We prove that the division makes Adam blind to the scale of each coordinate, and then follow that symmetry to the places where it ends: a rotation of the valley, a periodic sequence of convex losses, weight decay, the first steps of a run, and the matrix structure of a layer. Two results matter most, the scale-invariance proposition and the fixed points of AdamW. The regret bounds of AdaGrad and the structured preconditioners can be skipped on a first reading. Throughout, t counts optimizer steps, h_i is the curvature of coordinate i, and λ is the weight-decay coefficient.

35.5.1 Per-Coordinate Step Sizes

35.5.1.1 A Diagonal Quadratic

Gradient descent on a quadratic needs a number of steps proportional to the condition number $\kappa = h_{\max}/h_{\min}$, because its one step must stay below $2/h_{\max}$ (Section 35.1). The smallest case in which a cure is visible is the diagonal quadratic in two variables,

$$f(\mathbf{x}) = \tfrac{1}{2}(h_1 x_1^2 + h_2 x_2^2), \qquad \frac{\partial f}{\partial x_i} = h_i x_i.$$

Each partial derivative depends on its own coordinate only, so nothing forces the two coordinates to share a step. With a separate step η_i, coordinate i moves to $x_i - \eta_i h_i x_i = (1 - \eta_i h_i)\, x_i$, and the choice $\eta_i = 1/h_i$ sends every coordinate to zero in one step. For $h = (1, 1000)$ the two steps are 1 and 10^{-3}. This is Newton's method for a diagonal Hessian, and the condition number has disappeared from the iteration count.

Two obstacles stand between this observation and an algorithm. The curvatures h_i are not known and change along the run, and in training the gradients are noisy minibatch estimates. The step of each coordinate must therefore be estimated from the gradients themselves. The available estimate is the size of the gradient. On the diagonal quadratic the gradient entry $g_i = h_i x_i$ has size $h_i|x_i|$. When the coordinates have comparable magnitudes, a coordinate with large gradients therefore has large curvature, and dividing by the gradient size divides by the curvature. When they do not, $g_i/|g_i|$ still has size one: the division fixes the distance travelled per step at η, which is not the Newton step x_i.

35.5.1.2 AdaGrad

Curvature is one reason for per-coordinate steps. A second, independent one is that coordinates are used at very different rates. A global decaying step, $\eta/\sqrt{t}$, the classical choice in online and nonsmooth convex optimization, treats them all alike. Suppose coordinate 1 receives a gradient of ± 1 at every step and coordinate 2 only once every 10^4 steps, as the weight of a rare word does in a bag-of-words model. At $t = 10^4$ the global schedule gives both the step $\eta/100$, so the one piece of evidence about the rare coordinate moves it by a hundredth of η. If each coordinate accumulates its own squared gradients instead, the sums are 10^4 and 1, and dividing η by their square roots gives the busy coordinate $\eta/100$ and the rare one η.

This is **AdaGrad** (Duchi et al. 2011). Write $\mathbf{g}_t \odot \mathbf{g}_t$ for the vector of squared entries of the gradient (we abbreviate it as $\mathbf{g}_t^2$ from here on); square roots and divisions of vectors are also taken entry by entry. AdaGrad keeps the running sum $\mathbf{s}_t$ and divides by its square root:

$$\mathbf{s}_t = \mathbf{s}_{t-1} + \mathbf{g}_t \odot \mathbf{g}_t, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{s}_t} + \epsilon}\, \mathbf{g}_t, \tag{35.50}$$

with $\mathbf{s}_0 = \mathbf{0}$ and a small constant ϵ that keeps the division finite (Equation 9.26 in the main text puts ϵ inside the root, which makes no difference here). A coordinate's step decays with its own activity, not with the step count.

The metric view. Steepest descent depends on how step length is measured. Measuring it by $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^\top \mathbf{P}\, \mathbf{d}}$ for a positive definite $\mathbf{P}$ makes the steepest direction $-\mathbf{P}^{-1}\nabla f$ (Exercise 1 of Section 35.1). AdaGrad is steepest descent in the metric $\mathbf{P}_t = \operatorname{diag}(\sqrt{\mathbf{s}_t})$: moving along

coordinate i costs in proportion to the evidence $\sqrt{\sum_{\tau \le t} g_{\tau,i}^2}$ that its gradient has been large. A matrix such as $\mathbf{P}_t$, which rescales the gradient before the step, is called a **preconditioner**. Newton's method would use the true curvature $\mathrm{diag}(h_i)$, and AdaGrad replaces it by the gradient history (Exercise 1).

The regret view. AdaGrad was designed for **online convex optimization**. Convex losses $f_1, f_2, \dots$ arrive one at a time, and the algorithm must choose $\mathbf{x}_t$ before it sees f_t. Its performance after T rounds is the **regret**, $\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}} \sum_{t=1}^{T} f_t(\mathbf{x})$, the excess loss over the best fixed point chosen in hindsight. An algorithm learns, in this sense, when its regret divided by T tends to zero.

On a convex set of Euclidean diameter D_2, with every gradient bounded by $\|\mathbf{g}_t\| \le G$, online gradient descent with a tuned global step has regret of order $D_2 G \sqrt{T}$ (Exercise 7 proves it). AdaGrad has regret of order

$$D_\infty \sum_{i=1}^{d} \sqrt{\sum_{t=1}^{T} g_{t,i}^2}$$

(Duchi et al. 2011), where D_∞ is the diameter of the set in the ℓ_∞ norm. Each of the d coordinates is charged for its own gradient history, not for the largest gradient.

On the box $[-1,1]^d$, $D_2 = \sqrt{d}\, D_\infty$. When the gradient mass sits on a few coordinates, AdaGrad's bound is up to $\sqrt{d}$ times smaller; when the mass is spread evenly, the two bounds agree. For one-hot gradients that fall on coordinate i with probability p_i, the expected ratio of the bounds is at most $\sum_i \sqrt{p_i}/\sqrt{d}$ (Exercise 2). That is 1.00 for uniform p_i, even though every gradient is sparse, and 0.63 for word frequencies that follow Zipf's law with exponent 1 at $d = 10^4$. The gain therefore comes from concentration of the gradient mass, not from sparsity as such.

The square root in Equation 35.50 is the power that balances the two terms of the regret proof (Exercise 7c).

The accumulation is also AdaGrad's weakness outside this setting. Under persistent gradient noise of variance σ^2, $s_{t,i} \approx \sigma^2 t$, so the effective step is about $\eta/(\sigma\sqrt{t})$: AdaGrad carries its own decay schedule, tied to the step count and not to progress. That is the right rate for $\sqrt{T}$ regret on convex problems. On long nonconvex runs it is a drawback: the steps keep shrinking whether or not the loss is still falling, and a plateau met late is crossed with almost no step left.

35.5.1.3 RMSProp

RMSProp (Tieleman and Hinton 2012) replaces the all-time sum by an exponential moving average, so that the scale estimate follows the recent gradients:

$$\mathbf{v}_t = \beta_2\, \mathbf{v}_{t-1} + (1-\beta_2)\, \mathbf{g}_t^2, \qquad \mathbf{x}_{t+1} = \mathbf{x}_t - \frac{\eta}{\sqrt{\mathbf{v}_t} + \epsilon}\, \mathbf{g}_t. \tag{35.51}$$

Unrolled, $\mathbf{v}_t$ gives the squared gradient of k steps ago the weight $(1-\beta_2)\beta_2^k$, so the weights halve about every $\log 2/(1-\beta_2)$ steps: every 7 steps at RMSProp's customary $\beta_2 = 0.9$,

every 693 at $\beta_2 = 0.999$. The steps no longer decay to zero by construction. But a gradient seen once is now forgotten on that timescale, which the counterexample below exploits.

35.5.1.4 Adam and Its Bias Correction

Adam (Kingma and Ba 2015) applies the moving average of RMSProp to the gradient itself and corrects both averages for their zero start. In the notation of the main text (Equation 9.28, Equation 9.29), the update is

$$\begin{aligned} \mathbf{m}_t &= \beta_1\,\mathbf{m}_{t-1} + (1-\beta_1)\,\mathbf{g}_t, & \hat{\mathbf{m}}_t &= \mathbf{m}_t/(1-\beta_1^t), \\ \mathbf{v}_t &= \beta_2\,\mathbf{v}_{t-1} + (1-\beta_2)\,\mathbf{g}_t^2, & \hat{\mathbf{v}}_t &= \mathbf{v}_t/(1-\beta_2^t), \\ \mathbf{x}_{t+1} &= \mathbf{x}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t}+\epsilon}, \end{aligned} \tag{35.52}$$

with $\mathbf{m}_0 = \mathbf{v}_0 = \mathbf{0}$ and the defaults $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. The average $\mathbf{m}_t$ is the heavy-ball velocity of Section 35.1.3 with the factor $1-\beta_1$ in front, which turns a sum of gradients into a weighted average and smooths the minibatch noise that RMSProp steps along.

The division by $1-\beta_2^t$ corrects for the zero start. Unrolled from $\mathbf{v}_0 = \mathbf{0}$, the average is $\mathbf{v}_t = (1-\beta_2)\sum_{\tau\le t}\beta_2^{t-\tau}\,\mathbf{g}_\tau^2$. If the squared-gradient scale is stationary, $\mathbb{E}[\mathbf{g}_\tau^2] = \overline{\mathbf{g}^2}$ for every τ, the geometric series gives

$$\mathbb{E}[\mathbf{v}_t] = (1-\beta_2)\,\overline{\mathbf{g}^2}\sum_{\tau=1}^{t}\beta_2^{t-\tau} = (1-\beta_2^t)\,\overline{\mathbf{g}^2}. \tag{35.53}$$

So $\hat{\mathbf{v}}_t$ is unbiased for $\overline{\mathbf{g}^2}$ at every t under stationarity, and the same computation with β_1 handles $\hat{\mathbf{m}}_t$ (Exercise 3).

The deficit is large and it lasts: at the defaults $\mathbf{v}_{10}$ holds about 1% of its target, and $\mathbf{v}_{1000}$ still only 63% (Section 9.5). More interesting is the shape of the error. Without both corrections the update $\mathbf{m}_t/\sqrt{\mathbf{v}_t}$ is mis-scaled by the factor $(1-\beta_1^t)/\sqrt{1-\beta_2^t}$, which is 3.16 at $t=1$ and, perhaps surprisingly, keeps growing to a peak of 6.6 at $t=12$. The momentum average fills in about ten steps, while the second-moment average needs about a thousand. Figure 35.16 plots the deficit and the mis-scaling.

An unbiased $\hat{\mathbf{v}}_t$ still leaves the step size biased. Adam divides by $\sqrt{\hat{\mathbf{v}}_t}$, not by $\hat{\mathbf{v}}_t$, and the square root of an unbiased estimate is biased low: at $t=1$, $\sqrt{\hat{v}_1} = |g_1|$, and $\mathbb{E}|g| < \sqrt{\mathbb{E}[g^2]}$ unless $|g|$ is constant. This is why Adam's first step has size η in every coordinate, and not $\eta\,|g_1|/\sigma$ for a gradient of scale σ. Section 35.5.4 asks how large that is.

35.5.1.5 Scale Invariance

The valley of the opening has $h_2 = 1000\,h_1$, yet Adam crosses it as if the ratio were not there. The reason is a symmetry of the update, and it is the central result of this section.

Proposition (Adam ignores the scale of each coordinate). *Let $\epsilon = 0$ and let $\mathbf{C} = \operatorname{diag}(c_1,\dots,c_d)$ with every $c_i > 0$. Suppose two problems have gradients related by $\nabla\tilde{f}(\mathbf{x}) =$*

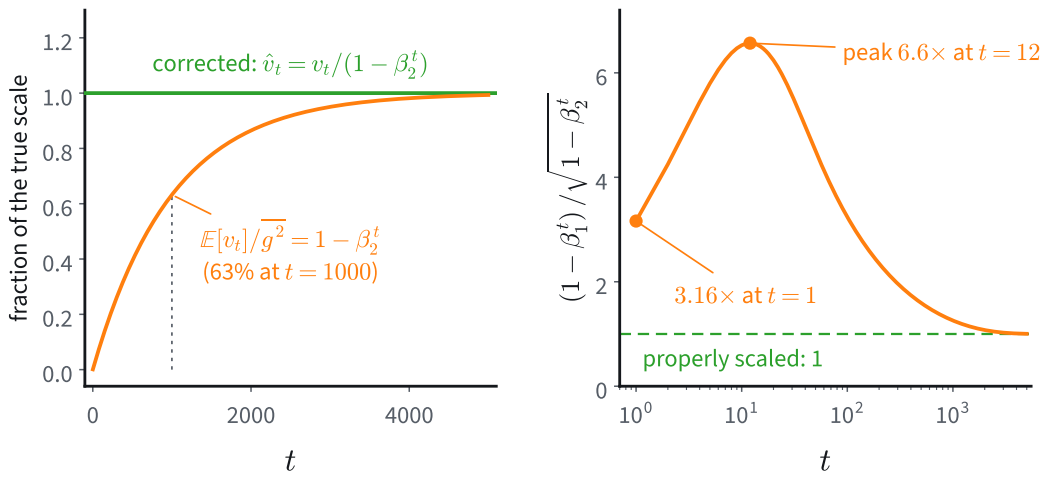


Figure 35.16: Bias correction in Adam. Left: with zero initialization, the running average $\mathbf{v}_t$ holds only the fraction $1-\beta_2^t$ of the squared-gradient scale; at $\beta_2 = 0.999$ it reaches 63% after 1000 steps, and dividing by $1-\beta_2^t$ restores the full scale at every t. Right: the mis-scaling of the uncorrected update $\mathbf{m}_t/\sqrt{\mathbf{v}_t}$, the factor $(1-\beta_1^t)/\sqrt{1-\beta_2^t}$. It is 3.16 at $t = 1$, peaks at 6.6 at $t = 12$ (the numerator fills in about ten steps, the denominator needs about a thousand), and returns to 1 only on the $1/(1-\beta_2)$ timescale. Uncorrected Adam takes its largest steps when its preconditioner rests on the fewest gradients.

C $\nabla f(\mathbf{x})$ *at every* $\mathbf{x}$. *Then Adam with the same* β_1, β_2, *step schedule and start produces the same iterates on both problems, as long as the updates are defined.*

The second problem is the first with the gradient of coordinate i reported in units c_i times smaller. We mark its quantities with a tilde.

Proof. By induction on t. Suppose the iterates agree up to $\mathbf{x}_t$. Then $\tilde{\mathbf{g}}_\tau = \mathbf{C}\mathbf{g}_\tau$ for every $\tau \le t$, and since the recursions for $\mathbf{m}$ and $\mathbf{v}$ are linear in $\mathbf{g}_\tau$ and $\mathbf{g}_\tau^2$, we get $\tilde{m}_{t,i} = c_i m_{t,i}$ and $\tilde{v}_{t,i} = c_i^2 v_{t,i}$. The bias corrections are the same scalars on both problems. Hence the corrected ratio $\hat{m}_{t,i}/\sqrt{\hat{v}_{t,i}}$ of the second problem is $c_i\hat{m}_{t,i}/(c_i\sqrt{\hat{v}_{t,i}}) = \hat{m}_{t,i}/\sqrt{\hat{v}_{t,i}}$, the updates coincide, and $\tilde{\mathbf{x}}_{t+1} = \mathbf{x}_{t+1}$. ■

On the diagonal quadratic, $\nabla f(\mathbf{x}) = \operatorname{diag}(h)\,\mathbf{x}$ is the gradient of the round bowl $\frac{1}{2}\|\mathbf{x}\|^2$ multiplied by $\mathbf{C} = \operatorname{diag}(h)$. So Adam on the $1000{:}1$ valley follows, step for step, the path of Adam on the round bowl, and on the bowl the two coordinates, started equal, stay equal. Each coordinate runs the same one-dimensional iteration on $\frac{1}{2}x^2$,

$$x_{t+1} = x_t\Big(1 - \frac{\eta}{\sqrt{\hat{v}_t}}\Big) \qquad (\beta_1 = 0),$$

where $\hat{v}_t$ averages the squares x_τ^2, $\tau \le t$. Since the earlier iterates were larger, $\sqrt{\hat{v}_t}$ exceeds $|x_t|$ from the second step on, so the iterate moves by less than η and contracts geometrically, by the factor $1-\eta/\sqrt{\hat{v}_t}$. For $\eta = 0.1$ this factor is 0.9 at $t = 1$ and 0.67 at $t = 48$, as $\hat{v}_t$ slowly follows the shrinking iterates. These iterates are the same for every ratio h_2/h_1.

Only the stopping test depends on the ratio, because $f(\mathbf{x}_t) = \frac{1}{2}(h_1 + h_2)\, x_t^2$ weighs the common iterate by the curvatures. A larger ratio should therefore cost a few more steps of a geometric contraction, not a factor κ.

What the count should depend on is the starting distance in units of η, because the first steps have length about η per coordinate whatever the curvature. A run in the next subsection measures both dependences.

Adam measures each coordinate in units of its own gradient size, so the scale of a coordinate is invisible to it. Three limits bound this statement. The constant ϵ breaks the invariance once a coordinate's gradients fall to the order of ϵ (Exercise 5). The proposition says nothing about a change of basis that mixes coordinates, which the rotated valley below tests. And the update is in general not sign descent. It is the **sign step** $\eta \operatorname{sign}(\mathbf{g}_t)$ in three cases only: at $t = 1$; for $\beta_1 = \beta_2 = 0$ (Exercise 8d); and, with $\beta_1 = 0$, when the gradient magnitude of each coordinate stays constant over the memory of $\hat{\mathbf{v}}$.

35.5.2 Coordinate Scaling and Its Limits

35.5.2.1 Aligned and Rotated Valleys

How much of gradient descent's 6,160 steps does coordinate scaling alone remove, and does the gain survive a rotation of the valley? To separate scaling from momentum we run four methods: gradient descent at the best single step; heavy ball with its tuned step and momentum; Adam with $\beta_1 = 0$, which only rescales; and Adam with its default $\beta_1 = 0.9$. The rotated problem replaces $\operatorname{diag}(h)$ by $\mathbf{A}_\theta = \mathbf{R}_\theta \operatorname{diag}(h)\, \mathbf{R}_\theta^\top$, where $\mathbf{R}_\theta$ is the rotation by θ, and rotates the start with it, $\mathbf{x}_0 = \mathbf{R}_\theta (1, 1)^\top$, so that the start has the same components along the two curvature directions. At $\theta = \pi/4$, $\mathbf{A} = \left(\begin{smallmatrix} 500.5 & -499.5 \\ -499.5 & 500.5 \end{smallmatrix}\right)$. The code is the problem, a loop that counts steps to the tolerance, and the three updates.

```
h = np.array([1.0, 1000.0])                    # curvatures h_i, kappa = 1000

def rotated(theta):
    """A = R diag(h) R^T and the start R (1, 1): the valley turned by theta."""
    c, s = np.cos(theta), np.sin(theta)
    R = np.array([[c, -s], [s, c]])
    return R @ np.diag(h) @ R.T, R @ np.ones(2)

def run(step, A, x0, K=40000, tol=1e-8):
    """First t with f(x_t) = x_t^T A x_t / 2 < tol, or None within K steps."""
    x, state = x0.copy(), {}
    for t in range(1, K + 1):
        x = step(x, A @ x, t, state)
        if 0.5 * x @ A @ x < tol:
            return t
    return None

def gd(eta):
    return lambda x, g, t, state: x - eta * g

def heavy_ball(eta, beta):
    def step(x, g, t, state):
        state['d'] = beta * state.get('d', 0.0) - eta * g    # velocity
```

```
            return x + state['d']
        return step

    def adam(eta, beta1=0.9, beta2=0.999, eps=1e-8):
        def step(x, g, t, state):
            state['m'] = beta1 * state.get('m', 0.0) + (1 - beta1) * g
            state['v'] = beta2 * state.get('v', 0.0) + (1 - beta2) * g * g
            m_hat, v_hat = state['m'] / (1 - beta1**t), state['v'] / (1 - beta2**t)
            return x - eta * m_hat / (np.sqrt(v_hat) + eps)
        return step
```

Before reading the counts, note what each method is given. Gradient descent and heavy ball run at their optimal settings, which the theory supplies in closed form from the two curvatures 1 and 1000. This is generous, since nobody knows the extreme curvatures of a network. Adam has no such formula, so it gets what a practitioner would give it, a sweep: seven step sizes spaced by factors of about three, from 0.002 to 2, with each row read at its best entry. The grid avoids $\eta = 1$ on purpose. Adam's first step has length η in every coordinate (Section 35.5.4), so from the start $(1, 1)$ the step $\eta = 1$ would land exactly on the minimizer, and Adam would be credited with solving the problem in one step. That would be an accident of where we started, and a comparison should not rest on it.

```
mu, L = h.min(), h.max()
rho = (np.sqrt(L) - np.sqrt(mu)) / (np.sqrt(L) + np.sqrt(mu))
tuned = {'GD, eta = 2/(mu + L)': gd(2 / (mu + L)),
         'heavy ball, tuned': heavy_ball(4 / (np.sqrt(L) + np.sqrt(mu))**2,
                                         rho**2)}
valleys = {'aligned': rotated(0.0), 'rotated 45': rotated(np.pi / 4)}
print(f'{"steps to f < 1e-8":28s}{"aligned":>9s}{"rotated 45":>12s}')
for name, step in tuned.items():
    hits = [run(step, *problem) for problem in valleys.values()]
    print(f'{name:28s}{hits[0] or "never":>9}{hits[1] or "never":>12}')
etas = np.geomspace(0.002, 2.0, 7)              # step sizes for Adam
print(f'\n{"Adam, eta =":28s}' + ''.join(f'{eta:>7.2g}' for eta in etas))
for beta1 in (0.0, 0.9):
    for name, problem in valleys.items():
        hits = [run(adam(eta, beta1=beta1), *problem) for eta in etas]
        print(f'{f"beta1 = {beta1}, {name}":28s}'
              + ''.join(f'{k or "never":>7}' for k in hits))
```

```
steps to f < 1e-8             aligned  rotated 45
GD, eta = 2/(mu + L)             6160        6160
heavy ball, tuned                 296         296

Adam, eta =                   0.002 0.0063   0.02  0.063    0.2   0.63      2
beta1 = 0.0, aligned           2122    760    249     78     22  never  never
beta1 = 0.0, rotated 45       never  never  never  never  never  never  never
beta1 = 0.9, aligned           2033    648    195    165    205    170    196
beta1 = 0.9, rotated 45       12862   9292   5898   3171   1581    799    559
```

Take the aligned valley first. Scaling alone needs 22 steps at its best step size, against 6,160 for gradient descent and 296 for heavy ball. That is thirteen times fewer than a heavy ball that was told both curvatures, from an optimizer that was told neither. The row also shows

what tuning Adam means. The count falls roughly in proportion to $1/\eta$, because the early steps have length about η and the count is mostly the distance to the minimizer measured in such steps. At the two largest step sizes the run never arrives: the iterate overshoots and ends in a cycle in which each coordinate jumps between $+\eta/2$ and $-\eta/2$. Adam has traded one unknown, the curvature, for another, the distance to the minimizer in units of η.

Adam with its default momentum needs between 165 and 205 steps at every step size from 0.02 to 2. Two things should be said about this row. The first is its robustness. A hundredfold range of step sizes changes the count by a quarter, and the two step sizes at which scaling alone cycles forever now converge, because the momentum average cancels the alternating steps. Heavy ball has no such latitude: its tuned step lies within 0.1% of its stability limit $(2 + 2\beta)/\lambda_{\max}$ (Exercise 9a of Section 35.1), so its 296 steps presuppose that the largest curvature is known almost exactly. The second is what the robustness costs. Momentum has made Adam seven times slower than scaling alone. The reason is the $\sqrt{\beta}$ proposition of Section 35.1.3: an iteration with momentum β contracts every mode by $\sqrt{\beta}$ per step as long as its roots are complex, and no faster. After the scaling, both coordinates run the same iteration on a round bowl, a problem of condition number one, for which the best momentum is none at all. With $\beta_1 = 0.9$ the factor is $\sqrt{0.9} \approx 0.95$ per step, which comes to about 230 steps for this tolerance, and the runs need somewhat fewer. Momentum that would help in an ill-conditioned valley is a brake on a round bowl, and Adam's default applies it either way.

Now the rotated valley. Gradient descent and heavy ball take the same number of steps as before, because their iterates rotate with the problem. Adam's do not. Scaling alone never reaches the tolerance, at any of the seven step sizes. Adam with momentum does, but only at the large end of the grid: its best count is 559, at $\eta = 2$, nearly twice the count of heavy ball, and at the smallest step size it needs 12,862 steps, twice as many as plain gradient descent. The robustness is gone as well, since the count now changes by a factor of more than twenty across the grid.

The fair comparison therefore reads as follows. Where the coordinate axes are the curvature directions, per-coordinate scaling beats a perfectly tuned heavy ball by an order of magnitude, and asks for less prior knowledge. Where they are not, nothing of the scaling survives. What still converges is the momentum, and a heavy ball with the right step does that job twice as fast. The rest of this subsection looks more closely at both halves of the statement.

On the aligned valley, the proposition predicts that the two coordinates carry the same iterate and that the stiff one receives a step a thousand times smaller. We print the iterate and the effective steps $\eta/\sqrt{\hat{\mathbf{v}}_t}$ of scaling-only Adam with $\epsilon = 0$ at $\eta = 0.1$. The last two lines test what the step count depends on: they repeat the run at three curvature ratios, and from a start a hundred times farther away.

```
A, x = rotated(0.0)
step, state, eta, gap = adam(0.1, beta1=0.0, eps=0.0), {}, 0.1, 0.0
for t in range(1, 49):
    x = step(x, A @ x, t, state)
    gap = max(gap, abs(x[0] - x[1]))
    if t in (1, 10, 48):
        eff = eta / np.sqrt(state['v'] / (1 - 0.999**t))
        print(f't = {t:2d}: x = {x[0]:.3g}, {x[1]:.3g};  '
```

```
                f'eta/sqrt(v_hat) = {eff[0]:.3g}, {eff[1]:.3g}')
    print(f'largest |x_1 - x_2| over the run: {gap:.1e}')
    counts = [run(adam(0.1, beta1=0.0), np.diag([1.0, r]), np.ones(2))
              for r in (1.0, 1e3, 1e5)]
    print('steps to f < 1e-8 at curvature ratios 1, 1e3, 1e5:', *counts)
    print('steps from (100, 100) at ratio 1e3:',
          run(adam(0.1, beta1=0.0), A, 100 * np.ones(2)))
```

```
t =  1: x = 0.9, 0.9;  eta/sqrt(v_hat) = 0.1, 0.0001
t = 10: x = 0.262, 0.262;  eta/sqrt(v_hat) = 0.152, 0.000152
t = 48: x = 3.91e-06, 3.91e-06;  eta/sqrt(v_hat) = 0.329, 0.000329
largest |x_1 - x_2| over the run: 5.6e-17
steps to f < 1e-8 at curvature ratios 1, 1e3, 1e5: 40 48 54
steps from (100, 100) at ratio 1e3: 4145
```

From the first step the effective steps stand in the ratio $1000{:}1$, the curvature ratio recovered from gradient sizes alone, with no Hessian. The two coordinates agree to rounding error at every step, and the effective steps grow from 0.1 to 0.33 on the flat coordinate as $\hat{v}_t$ follows the shrinking gradients. Raising the curvature ratio from 1 to 10^3 to 10^5 moves the count from 40 to 48 to 54: a hundred thousand times the conditioning costs fourteen steps. Moving the start a hundred times farther away raises it to 4,145. The count measures the starting distance in units of η, and hardly the curvature at all.

Why is the rotation fatal to all this? The rotated matrix has entries of size 500, and its stiff direction $(-1, 1)/\sqrt{2}$ loads equally on both coordinates. The gradient of each coordinate is therefore dominated by the stiff mode, and the two coordinates receive almost the same scale (following the runs, their $\hat{v}$ agree to within one percent). A per-coordinate scale is one number per coordinate, so it cannot divide the stiff and the flat component of the same coordinate by different amounts. With equal scales on both coordinates the preconditioner is a multiple of the identity. Adam with $\beta_1 = 0$ then becomes gradient descent whose step adapts to the stiff mode, and the flat mode crawls as it does under gradient descent. Adam with $\beta_1 = 0.9$ becomes a heavy ball. Its momentum is close to the tuned 0.88, but its step is $\eta/\sqrt{\hat{v}_t}$, set by the history of gradient sizes and not by the curvatures, and the best step size on the grid leaves it a factor of two behind.

Without momentum the run ends badly. As the stiff component dies out, $\hat{\mathbf{v}}$ shrinks, and the effective step grows until it reaches the stability limit $2/h_{\max}$ of the stiff mode. At $\eta = 0.2$, the best step size of the aligned valley, the loss falls to about 10^{-3} near $t = 1{,}700$, and then the stiff mode starts to oscillate. The oscillation holds the gradient magnitude constant, so $\hat{\mathbf{v}}_t \to \mathbf{g}_t^2$ and the update becomes the sign step $\eta \operatorname{sign}(\mathbf{g}_t)$, of length $\eta\sqrt{2}$ along the stiff direction. The run ends in a two-cycle of amplitude $\eta/\sqrt{2}$ across the valley, with $f = 250\,\eta^2 = 10$, however long it runs (Exercise 8 prints this value at every step size of the grid). Momentum averages the alternating steps, which is why the $\beta_1 = 0.9$ runs eventually reach the tolerance. Figure 35.17 shows the three behaviors on a valley shallow enough to draw.

Coordinate scaling removes ill-conditioning exactly when the coordinate axes are the eigenvectors of the curvature. It is a change of metric by a diagonal matrix, and no diagonal matrix represents curvature along a direction that mixes coordinates. A full matrix built from the outer products $\mathbf{g}_\tau \mathbf{g}_\tau^\top$ rotates with the problem and has no such defect, but it is too large for a network. The structured preconditioners at the end of the section address the limit for weight matrices, with factors that mix coordinates.

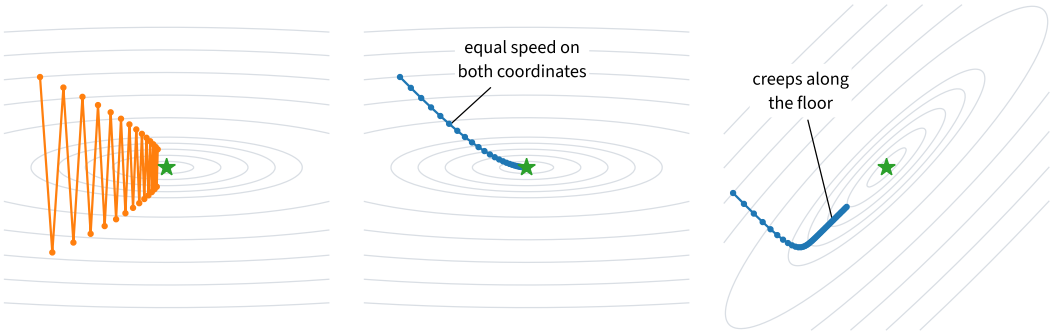


Figure 35.17: The same valley ($\kappa = 20$; the text's runs use $\kappa = 10^3$, whose level sets are too thin to draw) from the same start. (a) Gradient descent with one step size near the stability limit oscillates across the stiff direction and crawls along the flat one. (b) Per-coordinate normalization (Adam with $\beta_1 = 0$) moves both coordinates at equal speed and curves into the minimum. (c) The same update on the valley rotated by 45°, with the start rotated too: the coordinate scales no longer match the curvature directions, and the path creeps along the valley floor ($f = 0.19$ after 40 steps, against 4×10^{-7} in (b)).

35.5.2.2 A Convex Counterexample to Adam

RMSProp and Adam forget old gradients; AdaGrad does not. Can forgetting make a convex problem unsolvable? Reddi et al. (2018) construct a one-dimensional sequence of convex losses on which it does.

Proposition (Adam need not converge). *Let $C > 2$ and consider online optimization over $x \in [-1, 1]$ with the periodic losses*

$$f_t(x) = \begin{cases} C\,x, & t \bmod 3 = 1, \\ -x, & \text{otherwise.} \end{cases} \tag{35.54}$$

Run Adam without bias correction, with $\beta_1 = 0$, $\beta_2 = 1/(1 + C^2)$, steps $\eta_t = \eta/\sqrt{t}$ with $\eta < \sqrt{1 - \beta_2}$, and projection of each iterate back onto $[-1, 1]$, from $x_1 = 1$. Then $x_t > 0$ for every t and $x_{3k+1} = 1$ for every $k \geq 0$. Every period adds at least $2C - 4 > 0$ to the regret, so the average regret does not tend to zero. (Reddi et al. 2018)

Projection onto an interval is clipping (Section 35.6 treats the general case). The gradients of one period sum to $C - 2 > 0$, so the best fixed point is $x^\star = -1$, while Adam returns to $x = +1$, the worst point of the interval, at the start of every period. A period costs Adam at least $C - 2$, because $x = 1$ when the loss is Cx and the other two losses are at least -1 each. It costs $x^\star$ exactly $-(C - 2)$, hence the $2C - 4$. The proof is an induction over periods. Exercise 4 works through its key step, where the hypothesis $\eta < \sqrt{1 - \beta_2}$ keeps the downward step of each period from leaving $(0, 1]$. The failure is not special to $\beta_1 = 0$: the same paper constructs, for every fixed $\beta_1 < \sqrt{\beta_2}$, an online convex sequence and a stochastic convex problem on which Adam does not converge.

The mechanism is visible in one period at $C = 4$, $\beta_2 = 1/17$, once v has settled into its periodic pattern. After the gradients $+4, -1, -1$, v takes the values 15.12, 1.83 and 1.05, so $\sqrt{v}$ is 3.89, 1.35 and 1.02. The three displacements $-\eta g_t/\sqrt{v_t}$ are

$$-1.03\,\eta, \qquad +0.74\,\eta, \qquad +0.98\,\eta,$$

a net $+0.69\,\eta$ per period toward $+1$. Normalization gives the rare gradient C a displacement of about η, barely more than a gradient of size 1 receives, and C^2 leaves v within two steps. Together the two -1 steps therefore move x farther up than the one $+C$ step moves it down. Plain SGD with step $\eta g_t/C$ moves by $-\eta$, $+0.25\,\eta$, $+0.25\,\eta$, a net $-0.5\,\eta$ toward the optimum. Unlike the noise floor of constant-step SGD (Section 35.4), Adam's drift does not shrink away with the step: it points toward $+1$ at every step size.

The repair of Reddi et al. (2018), **AMSGrad**, divides by the running maximum $\tilde{\mathbf{v}}_t = \max(\tilde{\mathbf{v}}_{t-1}, \mathbf{v}_t)$ of the uncorrected average instead of by $\mathbf{v}_t$. The per-coordinate step $\eta_t/\sqrt{\tilde{v}_t}$ can then only fall, and the large gradient is never forgotten. In the periodic regime $\tilde{v} = 15.12$ throughout, and the displacements become $-1.03\,\eta$, $+0.26\,\eta$, $+0.26\,\eta$, a net $-0.51\,\eta$. So Adam should end at $+1$ and the other two at -1. We run all three with the same $1/\sqrt{t}$ decay from $x = 0$, the middle of the interval, so that Adam is not handed its bad endpoint.

```
def online(update, C=4.0, T=15000, alpha=0.5):
    """Projected iterates on the periodic losses; beta_1 = 0, no bias correction."""
    beta2 = 1 / (1 + C * C)
    x, v, vmax, xs = 0.0, 0.0, 0.0, []
    for t in range(1, T + 1):
        g = C if t % 3 == 1 else -1.0            # rare +C, frequent -1
        v = beta2 * v + (1 - beta2) * g * g
        vmax = max(vmax, v)                      # running maximum (AMSGrad)
        x = min(1.0, max(-1.0, x - alpha / np.sqrt(t) * update(g, v, vmax, C)))
        xs.append(x)
    return xs

updates = {'adam': lambda g, v, vmax, C: g / (np.sqrt(v) + 1e-8),
           'amsgrad': lambda g, v, vmax, C: g / (np.sqrt(vmax) + 1e-8),
           'sgd': lambda g, v, vmax, C: g / C}
print('x_t at t =        30      300     3000    15000   (x* = -1)')
for name, update in updates.items():
    xs = online(update)
    print(f'{name:9s}' + ''.join(f'{xs[t - 1]:9.3f}' for t in (30, 300, 3000, 15000)))
```

```
x_t at t =        30      300     3000    15000   (x* = -1)
adam          0.745    1.000    1.000    1.000
amsgrad      -0.953   -0.985   -0.995   -0.998
sgd          -0.954   -0.986   -0.995   -0.998
```

The step constant `alpha` of the cell is the η of the proposition, and $\eta = 0.5 < \sqrt{16/17}$, so the step hypothesis holds. Adam reaches $+1$ from the middle of the interval all the same. From $t \approx 300$ on it returns to $+1$ in every period and never drops below 0.97, while AMSGrad and SGD approach the optimum -1 and agree to two decimals. The construction shows that Adam has no general convergence guarantee. It does not predict failure on ordinary training problems.

35.5.3 Decoupled Weight Decay

35.5.3.1 Coupled and Decoupled Decay

Nearly every training recipe adds weight decay, so an adaptive method has to decide whether the decay term passes through its division. Under SGD the question does not arise: adding the penalty $\frac{\lambda}{2}\|\mathbf{w}\|^2$ to the loss adds $\lambda\mathbf{w}$ to the gradient, and the step $\mathbf{w}_t - \eta\,(\mathbf{g}_t + \lambda\mathbf{w}_t) = (1-\eta\lambda)\,\mathbf{w}_t - \eta\,\mathbf{g}_t$ is a gradient step on the loss followed by a shrinkage of the weights (Equation 9.30 in Section 9.5.4, which writes $\mathbf{x}$ for the weights). Through Adam the two are different updates. Feed the penalized gradient into Equation 35.52 and look at the decay term alone (take $\beta_1 = 0$ for clarity, so that the update is $-\eta\,(\mathbf{g}_t + \lambda\mathbf{w}_t)/(\sqrt{\hat{\mathbf{v}}_t} + \epsilon)$). The shrinkage of coordinate i in one step is

$$\text{coupled } (\ell_2 \text{ through Adam}): \quad \frac{\eta\,\lambda}{\sqrt{\hat{v}_{t,i}} + \epsilon}\, w_{t,i} \qquad \text{versus} \qquad \text{decoupled:} \quad \eta\,\lambda\, w_{t,i}. \tag{35.55}$$

The preconditioner divides the penalty gradient as well as the loss gradient, so the coupled shrinkage of a coordinate is inversely proportional to that coordinate's gradient scale. **AdamW** (Loshchilov and Hutter 2019) decouples the two: the loss gradient goes through the preconditioner, the decay does not,

$$\mathbf{w}_{t+1} = (1-\eta\lambda)\,\mathbf{w}_t - \eta\,\frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon}. \tag{35.56}$$

Every coordinate is shrunk by the same factor $1-\eta\lambda$ per step, as under SGD, and the moment estimates see only the loss gradient.

`torch.optim.Adam(weight_decay=...)` adds $\lambda\mathbf{w}$ to the gradient: it is the coupled rule. `torch.optim.AdamW` is the decoupled rule Equation 35.56.

The cleanest test of Equation 35.55 is a weight whose loss gradient is pure noise, so that decay is the only systematic force on it. If the noise has scale σ, then $\sqrt{\hat{v}} \approx \sigma$, and we expect the size $e^{-\eta\lambda T/\sigma}$ after T coupled steps, against $(1-\eta\lambda)^T$ under AdamW. The cell gives 10^4 weights the noise scale $\sigma = 10$ and 10^4 others the scale $\sigma = 0.1$, starts all of them at $w = 1$, takes 4,000 steps of each method with $\eta = 10^{-3}$ and $\lambda = 0.1$, and prints the mean of w over each group.

```
sigma, eta, lam, T, n = np.array([[10.0], [0.1]]), 1e-3, 0.1, 4000, 10_000
rng = np.random.default_rng()

def decay(decoupled, beta1=0.9, beta2=0.999, eps=1e-8):
    """Mean of w over n weights per noise scale after T Adam steps from w = 1."""
    w, m, v = np.ones((2, n)), np.zeros((2, n)), np.zeros((2, n))
    for t in range(1, T + 1):
        g = sigma * rng.standard_normal((2, n))     # loss gradient: pure noise
        if decoupled:
            w = (1 - eta * lam) * w                 # decay outside the preconditioner
        else:
            g = g + lam * w                         # l2 penalty inside the gradient
```

```
        m = beta1 * m + (1 - beta1) * g
        v = beta2 * v + (1 - beta2) * g * g
        w = w - eta * (m / (1 - beta1**t)) / (np.sqrt(v / (1 - beta2**t)) + eps)
    return w.mean(axis=1)

print('per-step decay rates: coupled', eta * lam / sigma.ravel(), ' AdamW', eta * lam)
print('mean w, coupled:', decay(False).round(3),
      '  predictions exp(-eta lam T / sigma):', np.exp(-eta * lam * T /
      ↪  sigma.ravel()).round(3))
print('mean w, AdamW:  ', decay(True).round(3),
      '  prediction (1 - eta lam)^T:', round((1 - eta * lam)**T, 3))
```

```
per-step decay rates: coupled [1.e-05 1.e-03]  AdamW 0.0001
mean w, coupled: [0.961 0.027]   predictions exp(-eta lam T / sigma): [0.961 0.018]
mean w, AdamW:   [0.67 0.67]   prediction (1 - eta lam)^T: 0.67
```

The same λ gives the two groups coupled decay rates of 10^{-5} and 10^{-3} per step, a factor of 100 apart. The noisy weights keep 0.96 of their size, as predicted, while the quiet weights fall to about 0.03, against a prediction of 0.018. The miss has a cause: at $w = 1$ the penalty gradient $\lambda w = 0.1$ of a quiet weight is as large as its noise and inflates its own $\hat{v}$, so these weights decay a little more slowly at first. Under AdamW both groups end at $0.67 = (1 - \eta\lambda)^{4000}$. For AdamW the prediction is exact in expectation, because the normalized noise steps have mean zero and the decay is a fixed factor per step.

35.5.3.2 Fixed Points

The experiment above had no fixed point, because its loss gradient was pure noise. With exact gradients an update can come to rest, and λ then means something else.

Proposition (fixed points of coupled and decoupled decay). *Let the gradient $\mathbf{g}(\mathbf{w}) = \nabla\ell(\mathbf{w})$ of a loss ℓ be exact (no noise), let $\epsilon > 0$, and call $\mathbf{w}$ a fixed point of an update if the update vanishes there once the moment estimates have settled. (a) Adam on $\ell + \frac{\lambda}{2}\|\mathbf{w}\|^2$ has a fixed point at $\mathbf{w}$ if and only if $\nabla\ell(\mathbf{w}) + \lambda\mathbf{w} = \mathbf{0}$, the stationary point of SGD on the same penalized loss. (b) AdamW has a fixed point at $\mathbf{w}$ if and only if, for every coordinate i,*

$$\lambda\, w_i(|g_i| + \epsilon) = -g_i, \qquad g_i = \partial_i\ell(\mathbf{w}). \tag{35.57}$$

Every fixed point of AdamW therefore lies strictly inside the box $|w_i| < 1/\lambda$, and as $\epsilon \to 0$ the condition becomes: either $g_i = 0$, or $|w_i| = 1/\lambda$ with w_i opposite in sign to g_i.

Proof. At a fixed point the vector $\mathbf{u}$ that enters the moment averages is constant, so $\hat{\mathbf{m}}_t \to \mathbf{u}$ and $\hat{\mathbf{v}}_t \to \mathbf{u}^2$. (a) For coupled Adam $\mathbf{u} = \mathbf{g} + \lambda\mathbf{w}$, and the update tends to $-\eta\, u_i/(|u_i| + \epsilon)$ in coordinate i, which vanishes exactly when $u_i = 0$. (b) For AdamW $\mathbf{u} = \mathbf{g}$, and the update of coordinate i tends to $-\eta\lambda w_i - \eta\, g_i/(|g_i| + \epsilon)$; setting it to zero and multiplying by $(|g_i| + \epsilon)/\eta$ gives Equation 35.57. Taking absolute values, $|w_i| = |g_i|/(\lambda(|g_i| + \epsilon)) < 1/\lambda$. As $\epsilon \to 0$ the condition reads $\lambda w_i |g_i| = -g_i$, which holds if $g_i = 0$ or if $\lambda w_i = -\text{sign}(g_i)$. ■

The proposition describes fixed points. It does not claim that one exists or that the iteration converges to it. Nor is the dichotomy only a limit: since $1 - \lambda|w_i| = \epsilon/(|g_i| + \epsilon)$, at $\epsilon = 10^{-8}$ a coordinate either sits within the fraction $\epsilon/|g_i|$ of the edge of the box or has a gradient of order ϵ.

Take the one-dimensional loss $\ell(w) = \frac{1}{2}(w-3)^2$, whose minimizer is $w = 3$. With $\lambda = \frac{1}{4}$, the fixed point of coupled Adam is the penalized minimizer $3/(1+\lambda) = 2.4$, while that of AdamW is 3.0: the box $|w| \le 4$ does not bind, and the decay leaves no trace. With $\lambda = 1$, the fixed point of coupled Adam is 1.5 and that of AdamW is 1.0, on the edge of the box.

The two readings of λ are now separate. In the noisy regime of the cell, AdamW applies the same per-step shrinkage to every coordinate and coupled Adam does not. At a fixed point the roles reverse: coupled Adam keeps the maximum a posteriori (MAP) point of a Gaussian prior on the weights (Section 36.6), while AdamW turns λ into the radius $1/\lambda$ of an ℓ_∞ box (Xie and Li 2024). Exercises 9 and 10 test the proposition on a quadratic and derive the box from the conditions of constrained optimization.

35.5.4 Warmup

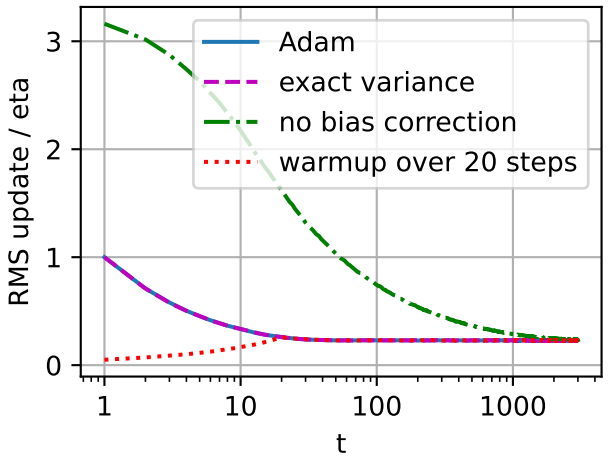
Adam's first update has size η in every coordinate: at $t = 1$, $\hat{\mathbf{m}}_1 = \mathbf{g}_1$ and $\hat{\mathbf{v}}_1 = \mathbf{g}_1^2$, so the step is $\eta \operatorname{sign}(\mathbf{g}_1)$. A coordinate whose gradient is 10^{-6} moves by η, just like one whose gradient is 10^3, on the evidence of a single sample. **Warmup** ramps the step size from about zero over the first steps of training (Goyal et al. 2017), which keeps these early updates small. It is tempting to blame the preconditioner, which at $t = 1$ rests on one gradient. The experiment below does not support that reading.

Is a step of η large? Not for a coordinate with a consistent gradient, whose stationary update is also η. For a coordinate whose gradient is noise, it is wasted motion. So we draw 2×10^4 independent gradient coordinates from $\mathcal{N}(0,1)$ at every step, run Adam with the default parameters, and record the root-mean-square update in units of η. A second curve replaces $\hat{\mathbf{v}}_t$ by the exact variance $\sigma^2 = 1$, which removes the estimation of the preconditioner altogether. A third drops both bias corrections, and a fourth multiplies Adam's step by a linear warmup over the first 20 steps.

```
n, T, beta1, beta2, eps, T_w = 20_000, 3000, 0.9, 0.999, 1e-8, 20
m, v = np.zeros(n), np.zeros(n)
rms = {'Adam': [], 'exact variance': [], 'no bias correction': [],
       f'warmup over {T_w} steps': []}
for t in range(1, T + 1):
    g = rng.standard_normal(n)                      # pure noise, sigma = 1
    m = beta1 * m + (1 - beta1) * g
    v = beta2 * v + (1 - beta2) * g * g
    m_hat, v_hat = m / (1 - beta1**t), v / (1 - beta2**t)
    step = m_hat / (np.sqrt(v_hat) + eps)          # Adam's update / eta
    for r, u in zip(rms.values(), (step, m_hat, m / (np.sqrt(v) + eps),
                                   min(1.0, t / T_w) * step)):
        r.append(np.sqrt(np.mean(u**2)))
d2l.plot(np.arange(1, T + 1), list(rms.values()), 't', 'RMS update / eta',
         legend=list(rms), xscale='log')
ts = (1, 2, 3, 10, 20, 30, 100, 1000, 3000)
print(f'{"t:":24s}' + ''.join(f'{t:>6d}' for t in ts))
for name, r in rms.items():
    print(f'{name + ":":24s}' + ''.join(f'{r[t - 1]:6.2f}' for t in ts))
```

```
t:                          1     2     3    10    20    30   100  1000  3000
Adam:                    1.00  0.71  0.58  0.33  0.26  0.24  0.23  0.23  0.23
exact variance:          1.00  0.71  0.58  0.33  0.26  0.24  0.23  0.23  0.23
```

```
no bias correction:      3.16  3.02  2.87  2.17  1.61  1.32  0.74  0.29  0.24
warmup over 20 steps:    0.05  0.07  0.09  0.17  0.26  0.24  0.23  0.23  0.23
```



The update starts at $1.00\,\eta$, is down to $0.33\,\eta$ at $t = 10$, and settles within about fifty steps at $\sqrt{(1-\beta_1)/(1+\beta_1)}\,\eta = 0.23\,\eta$, the stationary size of an exponential average of independent unit noise. The first steps of a noise coordinate are thus about four times too large, for about ten steps.

Is the unestimated preconditioner to blame? The second curve says no. With the exact variance in place of $\hat{\mathbf{v}}_t$ the update is the same to about 0.01 at every t, the sampling noise of 2×10^4 coordinates. The inflation comes from the momentum average, which needs about $1/(1-\beta_1) = 10$ steps to average the noise away. What the preconditioner does at $t = 1$ is the equalization above, and with equal scales on all coordinates this experiment cannot see it (Exercise 6 uses unequal ones).

The bias corrections matter far more. Without them the first update is $3.16\,\eta$, the factor $(1-\beta_1)/\sqrt{1-\beta_2}$ of Figure 35.16 times Adam's, and the transient lasts hundreds of steps where Adam's lasted ten: the update is still about $0.75\,\eta$ at $t = 100$ and $0.29\,\eta$ at $t = 1000$, because the uncorrected $\mathbf{v}_t$ fills only on the timescale $1/(1-\beta_2) = 1000$. An optimizer without bias correction would need a warmup of that length merely to undo its own initialization. With the corrections, a linear ramp over the first 20 steps removes what is left: the warmup curve never exceeds about $0.26\,\eta$.

Tens of steps, however, not the thousands used in practice. Long warmups rest on a second mechanism that this noise model does not contain, namely curvature. A step is stable only below the ceiling $2/\lambda_{\max}$ set by the largest curvature (Section 35.1.2). Right after initialization that curvature can be large and change quickly, so a step that is stable later in training may be unstable at first. Section 9.6 measures this in a real model.

35.5.5 Structured Preconditioners

35.5.5.1 Kronecker Factors

The rotated valley defeated every diagonal preconditioner. Every preconditioner of this section has the form $\mathbf{x}_{t+1} = \mathbf{x}_t - \eta\,\mathbf{P}_t^{-1}\mathbf{g}_t$ for a positive definite $\mathbf{P}_t$: gradient descent takes $\mathbf{P} = \mathbf{I}$, Newton's method the Hessian $\nabla^2 f$, and Adam a diagonal estimated from gradients.

A full matrix captures the curvature in every direction, but it is too large to store: one layer with a 4096×4096 weight matrix has 1.7×10^7 weights, and its curvature matrix has 2.8×10^{14} entries, about 2 petabytes in double precision.

What helps is that parameters come in matrices. Take a linear layer $\mathbf{y} = \mathbf{W}\mathbf{a}$ with $\mathbf{W} \in \mathbb{R}^{m \times n}$, input $\mathbf{a} \in \mathbb{R}^n$ and output gradient $\delta = \partial \ell / \partial \mathbf{y} \in \mathbb{R}^m$. Its gradient is the outer product $\delta \mathbf{a}^\top$. Write $\mathrm{vec}(\mathbf{V})$ for the vector that stacks the columns of a matrix $\mathbf{V}$; then $\mathrm{vec}(\delta \mathbf{a}^\top) = \mathbf{a} \otimes \delta$, where the **Kronecker product** $\mathbf{B} \otimes \mathbf{D}$ is the block matrix whose (i, j) block is $b_{ij}\mathbf{D}$.

The outer-product curvature of the layer, the block of the Fisher information matrix $\mathbb{E}[\nabla \log p \, \nabla \log p^\top]$ of Section 36.6 that belongs to $\mathbf{W}$, is $\mathbb{E}\big[(\mathbf{a}\mathbf{a}^\top) \otimes (\delta\delta^\top)\big]$. **K-FAC** (Martens and Grosse 2015) approximates it by $\mathbf{S}_{\text{in}} \otimes \mathbf{S}_{\text{out}}$ with $\mathbf{S}_{\text{in}} = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ $(n \times n)$ and $\mathbf{S}_{\text{out}} = \mathbb{E}[\delta\delta^\top]$ $(m \times m)$. The approximation is exact when inputs and output gradients are independent, which nobody claims of a trained network. The factorization is adopted because it can be inverted.

The factored form is useful because its inverse acts on the gradient matrix directly. For symmetric invertible factors,

$$\big(\mathbf{S}_{\text{in}} \otimes \mathbf{S}_{\text{out}}\big)^{-1} \mathrm{vec}(\mathbf{V}) = \mathrm{vec}\big(\mathbf{S}_{\text{out}}^{-1} \mathbf{V} \mathbf{S}_{\text{in}}^{-1}\big). \tag{35.58}$$

To see it, write column j of a product $\mathbf{DVB}$ as $\sum_k b_{kj} \mathbf{D}\mathbf{v}_k$, where $\mathbf{v}_k$ is column k of $\mathbf{V}$; stacked, these columns are the blocks of $(\mathbf{B}^\top \otimes \mathbf{D})\,\mathrm{vec}(\mathbf{V})$, so $\mathrm{vec}(\mathbf{DVB}) = (\mathbf{B}^\top \otimes \mathbf{D})\,\mathrm{vec}(\mathbf{V})$. Since $(\mathbf{B} \otimes \mathbf{D})^{-1} = \mathbf{B}^{-1} \otimes \mathbf{D}^{-1}$ and the factors are symmetric, taking $\mathbf{B} = \mathbf{S}_{\text{in}}^{-1}$ and $\mathbf{D} = \mathbf{S}_{\text{out}}^{-1}$ gives Equation 35.58. A curvature with $(mn)^2$ entries is applied through two inverses of sizes n and m. The cell counts what this saves for the layer above, with $m = n = 4096$.

```
m = n = 4096
print(f'stored entries: full {(m * n)**2:.1e}, two factors {m**2 + n**2:.1e}')
print(f'flops to invert: full {(m * n)**3:.1e}, two factors {m**3 + n**3:.1e}')
```

```
stored entries: full 2.8e+14, two factors 3.4e+07
flops to invert: full 4.7e+21, two factors 1.4e+11
```

The factors need 3.4×10^7 stored numbers instead of 2.8×10^{14}, and inverting them costs about 1.4×10^{11} operations instead of 4.7×10^{21}.

35.5.5.2 K-FAC, Shampoo and Muon

The Kronecker identity leaves open which matrix to factor and which power of it to apply. The methods in current use all have the two-sided form $\mathbf{G} \mapsto \mathbf{P}_{\text{l}}^{-a} \mathbf{G} \, \mathbf{P}_{\text{r}}^{-a}$ on a layer's gradient matrix $\mathbf{G}$ (bold, unlike the gradient bound G of the regret analysis). This is the ladder from Adam's diagonal to Muon.

K-FAC takes the factors from the second moments of the inputs and output gradients, with $a = 1$. It approximates preconditioning by the Fisher matrix, the **natural gradient** (Amari 1998), which is the metric view of AdaGrad with the Fisher matrix as the metric. For softmax cross-entropy and squared loss, the Fisher matrix (with labels drawn from the model) equals the Gauss–Newton matrix, a positive semidefinite approximation of the Hessian that drops the term containing the second derivatives of the network. For these losses K-FAC therefore approximates a curvature. **Shampoo** (Gupta et al. 2018) builds the factors AdaGrad-style

from the accumulated gradients, $\mathbf{P}_{\mathrm{l}} = \sum_t \mathbf{G}_t \mathbf{G}_t^\top$ and $\mathbf{P}_{\mathrm{r}} = \sum_t \mathbf{G}_t^\top \mathbf{G}_t$, with $a = \frac{1}{4}$. **Muon** (Jordan et al. 2024) is Shampoo without memory: with the current matrix alone, and $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$ square and invertible,

$$\left(\mathbf{G}\mathbf{G}^\top\right)^{-1/4} \mathbf{G} \left(\mathbf{G}^\top\mathbf{G}\right)^{-1/4} = \mathbf{U}\Sigma^{-1/2}\,\Sigma\,\Sigma^{-1/2}\mathbf{V}^\top = \mathbf{U}\mathbf{V}^\top,$$

the polar factor of $\mathbf{G}$. It keeps the singular vectors of $\mathbf{G}$ and sets every singular value to one. Where Adam equalizes the coordinates of the update, Muon equalizes its singular directions, and unlike a per-coordinate normalization this map commutes with rotations of the rows and columns (Exercise 8d). Muon as an optimizer, which applies the map to the momentum matrix, belongs to the main text (Section 9.7). The Newton–Schulz iteration, which reaches the polar factor with matrix products alone, is analyzed in Section 33.5.4.2.

None of the three reaches Newton's method, which would use the full curvature matrix. Muon keeps no second-moment state and needs a few matrix products per step; K-FAC and Shampoo invert or take roots of the two factors, usually every few steps. And the positive results of this section all assume curvature aligned with the coordinates, or with the Kronecker structure of a layer. The curvature of a deep network is neither diagonal nor fixed, so why these methods work as well as they do on deep networks is an open question.

35.5.6 Summary

Dividing each coordinate's step by the root mean square of its own gradients, as AdaGrad, RMSProp and Adam do, removes the condition number when the curvature is aligned with the coordinates: with $\epsilon = 0$ Adam's iterates do not depend on the scale of any coordinate. On the aligned valley this beats a tuned heavy ball by an order of magnitude. On the valley rotated by 45° scaling alone never converges, and Adam with momentum needs about twice the steps of heavy ball. A periodic convex sequence drives Adam to the worst point of an interval, because it forgets a rare large gradient. Through Adam, an ℓ_2 penalty decays coordinates at rates set by their gradient scales but keeps the MAP fixed point. AdamW's decay is uniform per step and turns λ into the radius of an ℓ_∞ box at its fixed points. Adam's first steps on a noise coordinate are about four times too large because of the momentum average, not the preconditioner, and without bias correction they are larger still, for hundreds of steps; long warmups answer to curvature. Kronecker factors apply a layer's curvature through two small inverses, and Muon equalizes the singular directions of a layer's update.

35.5.7 Exercises

1. **AdaGrad from the metric view.** Exercise 1b of Section 35.1 showed that the steepest descent direction under the norm $\|\mathbf{d}\|_{\mathbf{P}} = \sqrt{\mathbf{d}^\top \mathbf{P}\mathbf{d}}$ is proportional to $-\mathbf{P}^{-1}\nabla f$.
   1. Set $\mathbf{P}_t = \operatorname{diag}\left(\sqrt{\mathbf{s}_t}\right) + \epsilon\mathbf{I}$ with $\mathbf{s}_t = \sum_{\tau \le t} \mathbf{g}_\tau^2$ and recover Equation 35.50.
   2. Replace the objective f by cf for a constant $c > 0$ and set $\epsilon = 0$. How does the AdaGrad update change? How would it change if $\mathbf{P}_t$ were built from $\mathbf{s}_t$ instead of $\sqrt{\mathbf{s}_t}$? Compare both with gradient descent and with Newton's method.
   3. Let the gradients of one coordinate be i.i.d. with $\mathbb{E}[g_{t,i}^2] = \sigma^2$. How fast does the effective step of that coordinate decay in t under each of the two choices, and how does it depend on σ?
2. **Concentrated gradients and AdaGrad's bound.** Up to constant factors, projected online gradient descent with a tuned global step has regret $D_2 G\sqrt{T}$ on a convex set of

Euclidean diameter D_2 when $\|\mathbf{g}_t\| \le G$ (Exercise 7). Diagonal AdaGrad has regret $D_\infty \sum_{i=1}^{d} \sqrt{\sum_{t=1}^{T} g_{t,i}^2}$, where D_∞ is the diameter of the set in the ℓ_∞ norm (Duchi et al. 2011). Take the box $[-1,1]^d$ and one-hot gradients: at each step one coordinate i, drawn independently with probability p_i, receives the gradient $+1$ or -1, and all other coordinates receive 0.

1. Compute both bounds. For AdaGrad, bound the expectation of each square root with Jensen's inequality, and show that the ratio of the AdaGrad bound to the gradient-descent bound is at most $\sum_i \sqrt{p_i}/\sqrt{d}$.
2. Evaluate the ratio for uniform p_i and for Zipf frequencies $p_i \propto i^{-\alpha}$ with $d = 10^4$ and $\alpha \in \{1, 2, 3\}$.
3. Word frequencies in natural text follow Zipf's law with $\alpha \approx 1$, and your ratio there is about 0.63. How much of the factor $\sqrt{d}$ of Section 35.5.1 does a bag-of-words model obtain? Which distribution of gradient mass would obtain all of it?

3. **Bias correction and step inflation.** Let $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$ with $\mathbf{m}_0 = \mathbf{0}$.
   1. Repeat the computation of Equation 35.53 under $\mathbb{E}[\mathbf{g}_\tau] = \bar{\mathbf{g}}$ and show $\mathbb{E}[\mathbf{m}_t] = (1-\beta_1^t)\,\bar{\mathbf{g}}$.
   2. Without both corrections, Adam scales step t by the factor $(1-\beta_1^t)/\sqrt{1-\beta_2^t}$ of Figure 35.16. For $t \ll 1/(1-\beta_2)$, replace $1-\beta_2^t$ by $t\,(1-\beta_2)$. Show that the maximizer of the approximate factor satisfies $e^u = 1 + 2u$ with $u = -t \log \beta_1$, and solve this equation numerically.
   3. Express the location and the height of the peak through $1-\beta_1$ and $1-\beta_2$. How does each of the two depend on β_1 and on β_2? What does the result imply for running uncorrected Adam with a longer second-moment memory?
   4. Compare your approximation with the exact factor for $\beta_1 \in \{0.9, 0.99\}$ and $\beta_2 \in \{0.99, 0.999, 0.9999\}$. For which pair does the approximation fail, and why?
4. **The Reddi example by hand.** Take $C = 4$, $\beta_1 = 0$, the proposition's $\beta_2 = 1/(1+C^2) = 1/17$, and no bias correction, as in `online`.
   1. Assume a fixed step η, an iterate in the interior of $[-1,1]$, and v_t settled into its periodic pattern. Write v_a for the value of v after the $+C$ step, solve the three-step recursion for v_a in closed form, and derive the periodic values 15.12, 1.83, 1.05 and the three displacements of the text. Compute SGD's three steps $\eta g_t/C$. Which steps does Adam enlarge relative to SGD, and why?
   2. AMSGrad replaces v_t by $\tilde{v}_t = \max(\tilde{v}_{t-1}, v_t)$. Show that its effective step $\eta_t/\sqrt{\tilde{v}_t}$ is nonincreasing in t whenever η_t is. Compute its three displacements in the periodic regime and show that the net displacement is $-0.51\,\eta$.
   3. The proposition needs $\eta < \sqrt{1-\beta_2}$ for the steps $\eta_t = \eta/\sqrt{t}$. Show where this is used: starting a period at $x = 1$, use $v \ge (1-\beta_2)C^2$ after the $+C$ gradient to show that the first step of the period lands in $(0,1)$, so that the projection does not act on it.
5. code **Where ϵ matters.** Use `run` and `adam` of the `#adaptive-stochastic-methods-adam-vs-gd` cell with $\beta_1 = 0$ and $\eta = 0.1$ on the aligned valley.
   1. Sweep $\epsilon \in \{10^{-12}, 10^{-8}, 10^{-4}, 10^{-2}, 10^{-1}, 1\}$, record the step at which $f < 10^{-8}$, and print $\sqrt{\hat{\mathbf{v}}_t}$ at that step (let `run` also return the state). Explain the pattern of step counts with the printed values.
   2. Multiply both curvatures `h` and the tolerance by a common scale $s \in \{1, 10^{-4}, 10^{-8}, 10^{-10}\}$, raise the cap `K` to 10^5, and compare $\epsilon = 0$ with $\epsilon = 10^{-8}$. Which statement of the scale-invariance proposition does ϵ break, and below which gradient scale?

3. In a deep network, some layers receive gradients several orders of magnitude smaller than others. What does the gradient scale you found imply for the updates of those layers under the default ϵ?

6. code **Adam's first steps on noise.** Let every coordinate of the gradient be i.i.d. zero-mean noise with variance σ^2, as for a parameter that the loss does not yet constrain.
   1. Show that the bias-corrected first moment has variance
   $$\mathrm{Var}(\hat{m}_t) = \sigma^2 \, \frac{(1-\beta_1)(1+\beta_1^t)}{(1+\beta_1)(1-\beta_1^t)}. \tag{35.59}$$
   If $\hat{v}_t$ were replaced by the exact σ^2, what would the root-mean-square update be, in units of η, at $t = 1$ and as $t \to \infty$ at the default β_1? Compare with the warmup cell.
   2. The warmup cell gives every coordinate the same scale, which hides what the preconditioner does. Draw the scales σ_i log-uniformly from $[10^{-3}, 1]$ instead, and compare, across coordinates, the root-mean-square displacement $\eta\,\hat{m}_{t,i}/\sqrt{\hat{v}_{t,i}}$ at $t = 1$ and $t = 1000$ with that of SGD, $\eta\, g_{t,i}$. Which optimizer moves the small-scale coordinates as far as the large-scale ones, from which step on, and is that what you want for a coordinate that the loss does not constrain?
   3. Which timescale, $1/(1-\beta_1)$ or $1/(1-\beta_2)$, governs the transient of the warmup cell? Relate the answer to the two reasons for warmup given in Section 35.5.4.

7. **Regret of online gradient descent.** ⊠ Projected online gradient descent plays $\mathbf{x}_{t+1} = \Pi_{\mathcal{C}}(\mathbf{x}_t - \eta_t \mathbf{g}_t)$ with $\mathbf{g}_t = \nabla f_t(\mathbf{x}_t)$, where $\Pi_{\mathcal{C}}$ is the Euclidean projection onto a closed convex set $\mathcal{C}$ of diameter D, $\|\mathbf{g}_t\| \le G$, and the steps η_t are nonincreasing (Zinkevich 2003).
   1. Prove that the regret satisfies
   $$\sum_{t=1}^{T} f_t(\mathbf{x}_t) - \min_{\mathbf{x}\in\mathcal{C}} \sum_{t=1}^{T} f_t(\mathbf{x}) \;\le\; \frac{D^2}{2\eta_T} + \frac{G^2}{2}\sum_{t=1}^{T}\eta_t. \tag{35.60}$$
   Start from $f_t(\mathbf{x}_t) - f_t(\mathbf{x}) \le \mathbf{g}_t^\top(\mathbf{x}_t - \mathbf{x})$ and use that the projection does not increase distances to points of $\mathcal{C}$.
   2. Show that $\eta_t = D/(G\sqrt{t})$ gives regret at most $\frac{3}{2}DG\sqrt{T}$, and that for $\eta_t = c/t$ the bound grows linearly in T for every $c > 0$.
   3. Combine these two bounds with the decay rates of Exercise 1c to explain why AdaGrad divides by $\sqrt{\mathbf{s}_t}$ rather than by $\mathbf{s}_t$.

8. code **Rotation sensitivity.** ⊠ Use `rotated`, `run` and the update factories of the `#adaptive-stochastic-methods-adam-vs-gd` cell.
   1. Show that the iterates of gradient descent and of heavy ball on $\mathbf{A}_\theta$ from $\mathbf{R}_\theta(1,1)^\top$ are $\mathbf{R}_\theta$ times their iterates on the aligned valley, so that their step counts do not depend on θ.
   2. For $\beta_1 = 0$ at $\theta = \pi/4$, print f after $40{,}000$ steps at each η of the table's grid and compare it with the value $250\,\eta^2$ of the text. Which component of Adam supplies what survives of its advantage over gradient descent?
   3. Print Adam's effective per-coordinate steps at $\theta = \pi/16$. Why does even this small rotation remove most of the step-size ratio that Adam builds on the aligned valley?
   4. Exercise 5 of Section 35.1 identifies $-\mathrm{sign}(\mathbf{g})$ as the steepest direction under the ℓ_∞ norm and $-\mathbf{U}\mathbf{V}^\top$ as the steepest direction under the spectral norm, where $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$. Show that Adam with $\beta_1 = \beta_2 = 0$ and $\epsilon = 0$ steps along $-\mathrm{sign}(\mathbf{g}_t)$.

Then let $\mathbf{Q}_1$ and $\mathbf{Q}_2$ be orthogonal, show that the map $\mathbf{G} \mapsto \mathbf{U}\mathbf{V}^\top$ commutes with $\mathbf{G} \mapsto \mathbf{Q}_1\mathbf{G}\mathbf{Q}_2$, and give a 2×2 example in which the sign map does not. What does this suggest about the rotation sensitivity of Muon compared with Adam?

9. code **Coupled versus decoupled fixed points.** ▣ Let $\ell(\mathbf{w}) = \frac{1}{2}(\mathbf{w} - \mathbf{w}_0)^\top \mathrm{diag}(h_1, h_2)(\mathbf{w} - \mathbf{w}_0)$ with $h = (100, 1)$, $\mathbf{w}_0 = (1, 3)$, and exact gradients.
   1. For $\lambda = 0.1$, compute the fixed point of coupled Adam from the proposition of Section 35.5.3.
   2. For $\lambda = 0.1$ and $\epsilon = 10^{-8}$, solve the AdamW condition Equation 35.57 and show that the solution differs from $\mathbf{w}_0$ by less than 10^{-8}. What does weight decay contribute at this fixed point, and why?
   3. Choose λ so that the box $|w_i| \le 1/\lambda$ binds for coordinate 2 but not for coordinate 1. Predict both fixed points, and confirm all four predictions of parts (a)–(c) by running each method for 2×10^5 steps, with η decaying from 0.01 to 0 by a cosine schedule.
   4. Reconcile these fixed points with the `decay` cell. In the noise-dominated regime, which of the two updates applies the same per-step shrinkage to every coordinate, and why is that the property that matters during training?
10. [extended] **AdamW's implicit ℓ_∞ constraint.** ▣▣ The proposition of Section 35.5.3 found that the fixed points of AdamW lie in the box $|w_i| < 1/\lambda$ and, as $\epsilon \to 0$, on its boundary wherever the gradient does not vanish. Xie and Li (2024) prove that if full-batch AdamW with $\beta_1 \le \beta_2$ and a nonincreasing step schedule with divergent sum converges, its limit is a KKT point of $\min_{\mathbf{w}} \ell(\mathbf{w})$ subject to $\|\mathbf{w}\|_\infty \le 1/\lambda$.
    1. A feasible $\mathbf{w}$ is a KKT point of this problem (Section 35.6 derives the conditions in general) when every coordinate satisfies $\partial_i \ell(\mathbf{w}) = 0$ if $|w_i| < 1/\lambda$ and $w_i\, \partial_i \ell(\mathbf{w}) \le 0$ if $|w_i| = 1/\lambda$. Show that at such a point no feasible direction decreases ℓ to first order.
    2. Set $\epsilon = 0$, the setting of the theorem, in the AdamW condition Equation 35.57. Show that a coordinate with $\partial_i \ell \neq 0$ satisfies the KKT conditions. Why does the condition alone not show $|w_i| \le 1/\lambda$ for a coordinate whose gradient vanishes?
    3. Generate linearly separable logistic-regression data with $n = 100$, $d = 5$, standard normal features, and labels $y_i = \mathrm{sign}(\mathbf{x}_i^\top \mathbf{w}_{\mathrm{true}})$, so that the unregularized loss has no finite minimizer. Run full-batch AdamW with $\epsilon = 10^{-16}$ for 5×10^4 steps, with η decaying from 10^{-2} to 0 by a cosine schedule, for $\lambda \in \{0.05, 0.1, 0.2\}$. Compare the result with the solution of the box-constrained problem computed by projected gradient descent, which here clips every coordinate to $[-1/\lambda, 1/\lambda]$ after each gradient step, and with Adam on $\ell + \frac{\lambda}{2}\|\mathbf{w}\|^2$.
    4. Repeat the AdamW runs with minibatch gradients of size 10 and 50. Where does the iterate end up relative to the box, and why?
    5. What does λ control in AdamW according to these results, and how does that differ from the Gaussian-prior reading of λ in Section 36.6?
11. **The Kronecker identity and its cost.** Use column stacking for vec.
    1. Prove $\mathrm{vec}(\mathbf{DVB}) = (\mathbf{B}^\top \otimes \mathbf{D})\,\mathrm{vec}(\mathbf{V})$ by writing out column j of the product, and prove $(\mathbf{B} \otimes \mathbf{D})(\mathbf{B}' \otimes \mathbf{D}') = (\mathbf{B}\mathbf{B}') \otimes (\mathbf{D}\mathbf{D}')$.
    2. Deduce Equation 35.58 for symmetric invertible factors. Which formula replaces it for factors that are not symmetric, and which for row stacking?
    3. For an $m \times n$ layer, count the operations needed to invert the full $mn \times mn$ matrix and apply it, against inverting the two factors and applying them by Equation 35.58 ($O(m^3 + n^3 + mn(m+n))$). Evaluate both counts at $m = n = 4096$ and at $m = 4096$, $n = 16384$.
    4. For a single gradient $\mathbf{G} = \mathbf{U}\Sigma\mathbf{V}^\top$, set $\mathbf{P}_{\mathrm{l}} = \mathbf{G}\mathbf{G}^\top$ and $\mathbf{P}_{\mathrm{r}} = \mathbf{G}^\top\mathbf{G}$. Show that

the Shampoo step $\mathbf{P}_{\mathrm{l}}^{-1/4}\mathbf{G}\,\mathbf{P}_{\mathrm{r}}^{-1/4}$ equals $\mathbf{U}\mathbf{V}^{\top}$. What must the inverse fourth root mean when $\mathbf{G}$ is not square, and which singular vectors does the result keep?

35.6 Constrained Optimization

So far every variable could take any value. Many quantities in deep learning cannot. Attention weights must be nonnegative and sum to one, gradient clipping keeps the norm of a gradient at or below a threshold, a norm budget limits the weights, and the margins of a support vector machine must be at least one. At the best point of such a problem the gradient usually does not vanish, so the zero-gradient test of the earlier sections fails. How do we recognize the best point when the variables must stay in a set, and how do we compute it?

Gradient clipping gives the smallest instance. Clipping the gradient $\mathbf{y} = (3, 4)$ to norm at most 1 returns $(0.6, 0.8)$, the point of the unit disk nearest to $\mathbf{y}$. This point minimizes the squared distance $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ over the disk, yet it is not a point where any gradient vanishes: f has gradient $(-2.4, -3.2)$ there, and $f = 8$. Instead, no *allowed* direction lowers f: every move that keeps $\|\mathbf{x}\| \le 1$ makes f larger or, to first order, leaves it unchanged. We need a replacement for the condition "the gradient is zero", and a replacement for gradient descent, which on its own walks out of the feasible set.

The replacement is one number per constraint, its *multiplier*, which records how hard the optimum presses against that constraint. At $(0.6, 0.8)$ the push is 4 times the outward normal, and $(-2.4, -3.2) + 4\,(0.6, 0.8) = \mathbf{0}$. Multipliers give optimality conditions: Lagrange's for equality constraints, and those of Karush, Kuhn, and Tucker for inequalities. The projection that produced $(0.6, 0.8)$ gives an algorithm: take a gradient step, then return to the nearest feasible point. When a penalty with a kink, such as the absolute value that the lasso charges for every weight, replaces the hard wall, the same step becomes a proximal step, which sets weights to exactly zero. Convexity (Section 35.3) makes the conditions of this section sufficient and its algorithms convergent.

Three results carry the section: the KKT conditions with their sufficiency for convex problems, the obtuse-angle inequality that characterizes a projection, and soft-thresholding.

35.6.1 Equality Constraints and Lagrange Multipliers

35.6.1.1 Tangency and the Lagrange Condition

Equalities are simpler than inequalities, so we first replace the disk of the clipping example by its boundary: minimize $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ with $\mathbf{y} = (3, 4)$ over the unit circle, written as $h(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\|^2 - 1) = 0$. In general we minimize a differentiable $f : \mathbb{R}^n \to \mathbb{R}$ subject to one equality constraint,

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad h(\mathbf{x}) = 0.$$

The feasible set is a surface (in the example a curve), not a region, so every feasible move runs along it.

Figure 35.18 shows what distinguishes the optimum. At a feasible point where a level set of f *crosses* the circle, ∇f has a component along the circle, and sliding along the circle

against that component lowers f while staying feasible. Such a point cannot be optimal. Only where the level set is *tangent* to the circle is every feasible direction neutral to first order. Tangent curves have parallel normals, so at the optimum ∇f is a multiple of ∇h.

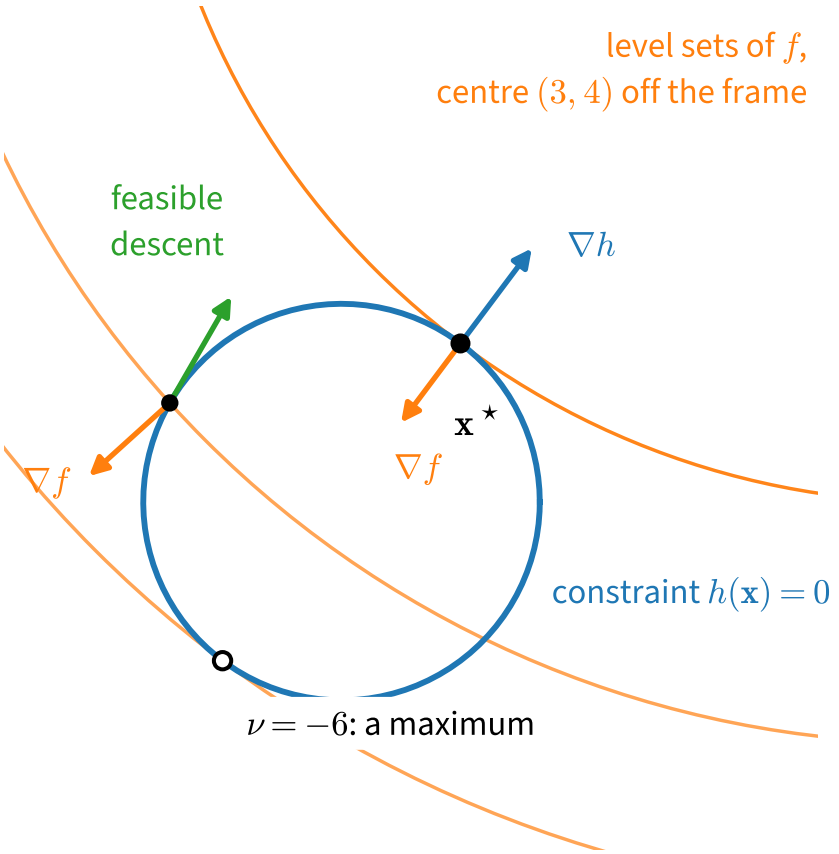


Figure 35.18: Minimizing $f = \frac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$, $\mathbf{y} = (3, 4)$, on the circle $h(\mathbf{x}) = 0$. At a feasible point where a level set crosses the circle, ∇f has a component along the constraint and sliding against it is a feasible descent. At $\mathbf{x}^\star = (0.6, 0.8)$ the level set is tangent and $\nabla f = -4\,\nabla h$. The antipode $(-0.6, -0.8)$ also satisfies the condition, with $\nu = -6$: it is the maximum.

Proposition (Lagrange condition). *Let $\mathbf{x}^\star$ be a local minimum of f on $\{\mathbf{x} : h(\mathbf{x}) = 0\}$. Assume that f is differentiable at $\mathbf{x}^\star$, that h is continuously differentiable near $\mathbf{x}^\star$, and that $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$. Then there is a unique scalar $\nu^\star$, the* **Lagrange multiplier***, with*

$$\nabla f(\mathbf{x}^\star) + \nu^\star \nabla h(\mathbf{x}^\star) = \mathbf{0}. \tag{35.61}$$

The objective's gradient is balanced by a multiple of the constraint's normal. This is the Lagrange condition of Section 34.2.4, where the tangency argument is made into a proof. That section writes the constraint as $g(\mathbf{x}) = c$ and the condition as $\nabla f = \mu \nabla g$; with $h = g - c$ and $\nu = -\mu$ the two statements agree. We use the plus sign because it is the sign of the Lagrangian below.

At the tangent point $(0.6, 0.8)$ of the figure, $\nabla f = \mathbf{x} - \mathbf{y} = (-2.4, -3.2)$ and $\nabla h = \mathbf{x} = (0.6, 0.8)$. The two vectors are parallel, with $\nabla f = -4\,\nabla h$, so $\nu^\star = 4$.

The hypothesis $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$ is a **constraint qualification**, and it is not a formality. Minimize $f(\mathbf{x}) = x_1$ subject to $h(\mathbf{x}) = x_1^2 + x_2^2 = 0$. The feasible set is the single point $\mathbf{0}$, which is therefore the minimum, yet $\nabla f(\mathbf{0}) = (1, 0)$ while $\nabla h(\mathbf{0}) = \mathbf{0}$, so no multiplier exists. The constraint set has collapsed to a point with no tangent directions, and the tangency argument has nothing to work with.

35.6.1.2 The Lagrangian

Two requirements, parallel gradients and feasibility, determine two unknowns, $\mathbf{x}$ and ν. Both requirements say that a single function of $(\mathbf{x}, \nu)$ is stationary. This function, the **Lagrangian**, adds to the objective the constraint function weighted by the multiplier:

$$\mathcal{L}(\mathbf{x}, \nu) = f(\mathbf{x}) + \nu\, h(\mathbf{x}). \tag{35.62}$$

Setting its derivatives to zero gives

$$\nabla_{\mathbf{x}} \mathcal{L} = \nabla f + \nu\, \nabla h = \mathbf{0} \qquad \text{and} \qquad \frac{\partial \mathcal{L}}{\partial \nu} = h(\mathbf{x}) = 0,$$

which are Equation 35.61 and the constraint. The constrained problem in $\mathbf{x}$ has become an unconstrained stationarity problem in $(\mathbf{x}, \nu)$.

On the circle, $\nabla_{\mathbf{x}} \mathcal{L} = (\mathbf{x} - \mathbf{y}) + \nu \mathbf{x} = \mathbf{0}$ gives $\mathbf{x} = \mathbf{y}/(1 + \nu)$, a point on the line through the origin and $\mathbf{y}$. Feasibility, $\|\mathbf{x}\| = 1$, then requires $|1 + \nu| = \|\mathbf{y}\| = 5$, which has two solutions:

- $\nu = 4$: $\mathbf{x} = (0.6, 0.8)$, at distance 4 from $\mathbf{y}$, with $f = 8$;
- $\nu = -6$: $\mathbf{x} = (-0.6, -0.8)$, at distance 6 from $\mathbf{y}$, with $f = 18$.

The second point is the point of the circle farthest from $\mathbf{y}$. The Lagrange condition is necessary, not sufficient: it holds wherever f restricted to the surface is stationary, maxima included. In this example the sign of the multiplier tells the two points apart. For inequality constraints the KKT conditions below turn that sign into a rule.

With several equality constraints $h_j(\mathbf{x}) = 0$, $j = 1, \dots, p$, each constraint gets its own multiplier, $\mathcal{L} = f + \sum_j \nu_j h_j$. The same argument, now assuming that the $\nabla h_j(\mathbf{x}^\star)$ are linearly independent, puts $\nabla f(\mathbf{x}^\star)$ in the *span* of the constraint gradients: the objective's gradient is balanced by a combination of the constraint normals.

35.6.1.3 The Closest Point on a Hyperplane

The smallest problem with a linear constraint asks for the point of a hyperplane nearest the origin: minimize $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|^2$ subject to $h(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} - b = 0$. Stationarity of $\mathcal{L} = \frac{1}{2}\|\mathbf{x}\|^2 + \nu(\mathbf{a}^\top \mathbf{x} - b)$ gives $\mathbf{x} + \nu \mathbf{a} = \mathbf{0}$, so the solution lies on the normal $\mathbf{a}$. Feasibility, $-\nu \|\mathbf{a}\|^2 = b$, fixes $\nu^\star = -b/\|\mathbf{a}\|^2$, and the solution and its value are

$$\mathbf{x}^\star = \frac{b}{\|\mathbf{a}\|^2}\,\mathbf{a}, \qquad p^\star = \frac{b^2}{2\,\|\mathbf{a}\|^2}.$$

The nearest point lies a distance $b/\|\mathbf{a}\|$ along the unit normal $\mathbf{a}/\|\mathbf{a}\|$, which is the projection formula of Section 33.1.

With $\mathbf{a} = (3, 4)$ and $b = 5$ the nearest point is $(0.6, 0.8)$ once more, with $\nu^\star = -0.2$ and $p^\star = \frac{1}{2}$: the line $3x_1 + 4x_2 = 5$ touches the unit circle at $(0.6, 0.8)$. The multiplier is more than a by-product of the calculation. Raising b from 5 to 5.1 moves the line away from the origin and raises the optimal value by about $0.02 = -\nu^\star \cdot 0.1$. Section 35.7.3 shows that this is what a multiplier always measures: the rate at which the optimum changes when its constraint is moved.

35.6.2 Inequality Constraints and the KKT Conditions

35.6.2.1 Active and Inactive Constraints

Most constraints in practice are one-sided: a norm at most r, a probability at least 0, a budget not exceeded. We write them as

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0, \; i = 1, \dots, m,$$

so that each g_i is negative inside its region and zero on its boundary. At a solution each inequality is in one of two regimes. Either $g_i(\mathbf{x}^\star) < 0$: the constraint is **inactive**, the solution lies strictly inside its region, and near $\mathbf{x}^\star$ the constraint has no effect. Or $g_i(\mathbf{x}^\star) = 0$: the constraint is **active** and behaves like an equality.

Clipping shows both regimes. Project $\mathbf{y}$ onto the ball of radius r: minimize $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ subject to $g(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\|^2 - r^2) \le 0$. With a multiplier λ, stationarity of $f + \lambda g$ gives $(\mathbf{x} - \mathbf{y}) + \lambda\mathbf{x} = \mathbf{0}$, so $\mathbf{x} = \mathbf{y}/(1 + \lambda)$: the solution is a radial rescaling of $\mathbf{y}$.

- If $\|\mathbf{y}\| \le r$, then $\mathbf{y}$ is feasible and minimizes f outright. The constraint is inactive, $\lambda^\star = 0$, and $\mathbf{x}^\star = \mathbf{y}$; for instance $\mathbf{y} = (0.3, 0.4)$ stays where it is.
- If $\|\mathbf{y}\| > r$, a solution inside the ball would be an unconstrained minimum of f, and the only one is the infeasible $\mathbf{y}$. So the constraint is active, and $\|\mathbf{x}^\star\| = r$ forces $\lambda^\star = \|\mathbf{y}\|/r - 1 > 0$. For $\mathbf{y} = (3, 4)$ and $r = 1$ this gives $\lambda^\star = 4$, $\mathbf{x}^\star = (0.6, 0.8)$, and $p^\star = \frac{1}{2} \cdot 4^2 = 8$.

Both cases combine into one formula,

$$\mathbf{x}^\star = \frac{\mathbf{y}}{\max\left(1, \; \|\mathbf{y}\|/r\right)},$$

which leaves short vectors alone and rescales long ones to length r. This is *gradient clipping* (Pascanu et al. 2013), applied to the update vector. In code the case split is a single `max`, and the balance of forces can be read off: at the clipped point the gradient of f is cancelled by $\lambda^\star$ times the gradient of the constraint, $\nabla g(\mathbf{x}) = \mathbf{x}$.

```
y, r = np.array([3.0, 4.0]), 1.0
lam = max(np.linalg.norm(y) / r - 1, 0.0)    # multiplier: 0 if y is inside the ball
x = y / (1 + lam)                            # x* = y / max(1, |y|/r)
print('x* =', x, ' lambda* =', lam, ' f(x*) =', 0.5 * ((x - y)**2).sum())
print('grad f(x*) =', x - y, ' lambda* grad g(x*) =', lam * x,
      ' sum =', (x - y) + lam * x)
```

```
x* = [0.6 0.8]  lambda* = 4.0  f(x*) = 8.0
grad f(x*) = [-2.4 -3.2]  lambda* grad g(x*) = [2.4 3.2]  sum = [0. 0.]
```

Stationarity holds although ∇f is far from zero: the gradient $(-2.4, -3.2)$ and the constraint's push $(2.4, 3.2)$ cancel, and the multiplier is the ratio of their lengths, 4.

The sign of the multiplier has a geometric meaning. The constraint's gradient $\nabla g = \mathbf{x}$ points out of the ball, and stationarity $-\nabla f = \lambda \nabla g$ with $\lambda > 0$ says that the objective pulls outward and the wall pushes back. If the objective pulled inward, moving inward would be feasible and would lower f, so the point would not be optimal. A negative multiplier is therefore never needed: the multiplier -6 would give the far point $(-0.6, -0.8)$ of the circle, which is not a minimum over the disk. An inequality constraint pushes the solution in one direction only.

35.6.2.2 The Karush–Kuhn–Tucker Conditions

With both kinds of constraints, the general problem is

$$\min_{\mathbf{x}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0, \; i = 1, \dots, m, \qquad h_j(\mathbf{x}) = 0, \; j = 1, \dots, p. \tag{35.63}$$

Its **Lagrangian** has one multiplier per constraint, and the inequality multipliers are nonnegative:

$$\mathcal{L}(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \, g_i(\mathbf{x}) + \sum_{j=1}^{p} \nu_j \, h_j(\mathbf{x}), \qquad \lambda_i \ge 0. \tag{35.64}$$

At a minimum $\mathbf{x}^\star$ where a constraint qualification holds (the next subsection says which), there are multipliers that satisfy the four **Karush–Kuhn–Tucker (KKT) conditions** (Karush 1939; Kuhn and Tucker 1951):

$$\begin{aligned}
&\text{stationarity:} && \nabla f(\mathbf{x}^\star) + \textstyle\sum_i \lambda_i^\star \nabla g_i(\mathbf{x}^\star) + \sum_j \nu_j^\star \nabla h_j(\mathbf{x}^\star) = \mathbf{0}, \\
&\text{primal feasibility:} && g_i(\mathbf{x}^\star) \le 0, \qquad h_j(\mathbf{x}^\star) = 0, \\
&\text{dual feasibility:} && \lambda_i^\star \ge 0, \\
&\text{complementary slackness:} && \lambda_i^\star \, g_i(\mathbf{x}^\star) = 0 \quad \text{for every } i.
\end{aligned} \tag{35.65}$$

The first three we have met in the examples: a balance of gradients, the constraints themselves, and a push in one direction only. **Complementary slackness** says that for each i at least one of $\lambda_i^\star$ and $g_i(\mathbf{x}^\star)$ is zero. An inactive constraint therefore has a zero multiplier and drops out of stationarity, and the multipliers tell which constraints contribute to stationarity. They do not always identify the active set, because a constraint can be active with $\lambda_i^\star = 0$ (it touches $\mathbf{x}^\star$ but does no work). In the ball projection complementary slackness is the case split we did by hand, and at the solution $\lambda^\star g(\mathbf{x}^\star) = 4 \cdot 0 = 0$.

Figure 35.19 draws stationarity and dual feasibility together: at the optimum, $-\nabla f$ is a nonnegative combination of the outward normals of the active constraints. Equivalently, $-\nabla f$ lies in the cone those normals span, the *normal cone*; inactive constraints add nothing to it.

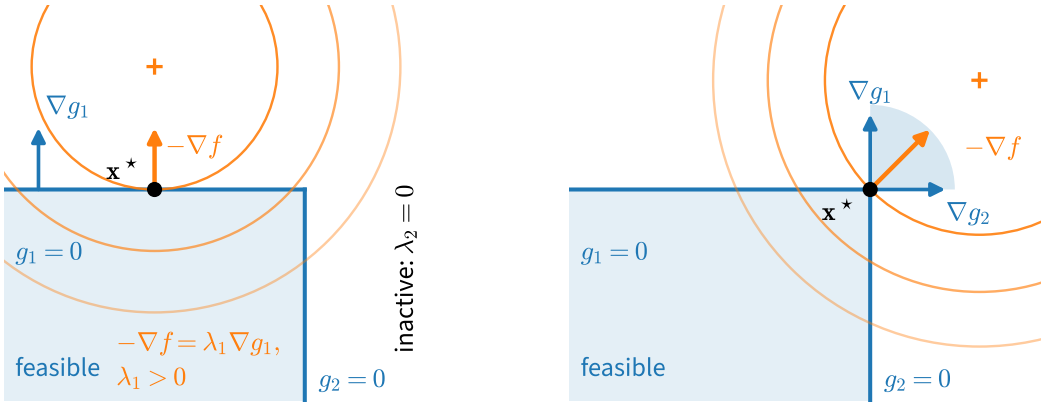


Figure 35.19: Geometry of the KKT conditions on a feasible region cut out by two inequality constraints. At the optimum $\mathbf{x}^\star$ in (a) one constraint is active and one is inactive: $-\nabla f$ points along the active constraint's outward normal with $\lambda_1 > 0$, while the inactive constraint has $\lambda_2 = 0$. At the corner in (b), where both constraints are active, $-\nabla f$ must lie in the cone spanned by both normals (shaded: the normal cone). The orange arcs are level sets of f around its unconstrained minimizer (+).

The KKT conditions are first-order conditions, and like the Lagrange condition they are not sufficient. Minimize $f(x) = -x^2$ on $[-1, 1]$, written as $x - 1 \le 0$ and $-1 - x \le 0$. At $x = 0$ the gradient is zero, both constraints are inactive, and both multipliers are zero, so $x = 0$ is a KKT point. It is the *maximum*. The minima $x = \pm 1$ are KKT points as well, each with multiplier 2 on its active constraint. Exercise 2 works out a two-dimensional version.

35.6.2.3 Constraint Qualifications and Sufficiency

Two questions remain. When must a minimum satisfy the KKT conditions, and when is a KKT point a minimum?

The first question has the same answer as for one equality constraint: when a constraint qualification holds. The standard one is the *linear independence constraint qualification* (LICQ). Let $\mathbf{x}^\star$ be a local minimum of Equation 35.63 with f, g_i, h_j continuously differentiable. If the gradients of the active inequality constraints together with the gradients of all equality constraints, $\{\nabla g_i(\mathbf{x}^\star) : g_i(\mathbf{x}^\star) = 0\} \cup \{\nabla h_j(\mathbf{x}^\star) : j = 1, \dots, p\}$, are linearly independent, then multipliers satisfying Equation 35.65 exist (Nocedal and Wright (2006), chapter 12, proves this). LICQ generalizes the condition $\nabla h \neq \mathbf{0}$, and the problems of Exercise 1 show how it fails.

For convex problems a condition that is far easier to check suffices: it asks for one point, not for gradients at the unknown solution. Call Equation 35.63 *convex* when f and every g_i are convex and every h_j is affine. A convex problem satisfies **Slater's condition** (Slater 1950) if there is a point $\bar{\mathbf{x}}$ with

$$g_i(\bar{\mathbf{x}}) < 0 \ \text{ for every } g_i \text{ that is not affine}, \qquad g_i(\bar{\mathbf{x}}) \le 0 \ \text{ for every affine } g_i, \qquad h_j(\bar{\mathbf{x}}) = 0 \ \text{ for every } j. \tag{35.66}$$

The point must be feasible, and strictly inside every curved constraint. A problem whose

constraints are all affine satisfies the condition as soon as it has a feasible point. (If f or some g_i is defined only on a subset of $\mathbb{R}^n$, which does not happen in this section, $\bar{\mathbf{x}}$ must also lie in the relative interior of that domain.) For the ball, $\bar{\mathbf{x}} = \mathbf{0}$ gives $g(\mathbf{0}) = -r^2/2 < 0$. Under Slater's condition every minimum of a convex problem with differentiable f and g_i is a KKT point (Boyd and Vandenberghe 2004); the reason is strong duality, which Section 35.7.2 derives from the same condition.

Convexity answers the second question, and the answer is the result of this section that we will use most.

Proposition (KKT sufficiency under convexity). *In problem Equation 35.63, let f and every g_i be convex and differentiable and every h_j affine. If $(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ satisfies the KKT conditions Equation 35.65, then $\mathbf{x}^\star$ is a global minimum.*

Proof. The function $\varphi(\mathbf{x}) = \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star)$ is convex: it is a nonnegative combination of convex functions plus affine terms. Stationarity says $\nabla\varphi(\mathbf{x}^\star) = \mathbf{0}$, so $\mathbf{x}^\star$ is a *global* minimizer of φ. Now take any feasible $\mathbf{x}$. Since $\lambda_i^\star \geq 0$, $g_i(\mathbf{x}) \leq 0$, and $h_j(\mathbf{x}) = 0$,

$$f(\mathbf{x}) \;\geq\; f(\mathbf{x}) + \sum_i \lambda_i^\star g_i(\mathbf{x}) + \sum_j \nu_j^\star h_j(\mathbf{x}) = \varphi(\mathbf{x}) \;\geq\; \varphi(\mathbf{x}^\star) = f(\mathbf{x}^\star),$$

where the last equality uses complementary slackness ($\sum_i \lambda_i^\star g_i(\mathbf{x}^\star) = 0$) and feasibility ($h_j(\mathbf{x}^\star) = 0$). ■

The proof contains an idea larger than the proposition. For *any* fixed multipliers with $\lambda \geq 0$, the first inequality of the chain holds, so the unconstrained minimum of the Lagrangian is a lower bound on f over the feasible set; at the KKT multipliers the bound is attained. Section 35.7 starts from this bound. For convex problems the KKT conditions are sufficient, and under Slater's condition also necessary, so they identify the minima completely.

35.6.3 Projections and Projected Gradient Descent

The KKT conditions recognize an optimum. To compute one we combine the nearest feasible point with gradient descent: take the ordinary gradient step, and if it leaves the feasible set, project back.

35.6.3.1 Projection onto a Convex Set

For a nonempty closed convex set $C \subseteq \mathbb{R}^n$, the **projection** of $\mathbf{y}$ onto C is the nearest point of C,

$$\Pi_C(\mathbf{y}) = \operatorname*{argmin}_{\mathbf{x}\in C} \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2.$$

A minimizer exists: we may restrict the search to the intersection of C with a large closed ball around $\mathbf{y}$, which is compact because C is closed, and the continuous objective attains its minimum there. It is unique because the objective is strictly convex. So Π_C is a well-defined map.

Both examples computed so far are projections: onto a hyperplane (the closest-point problem) and onto a ball (the clipping formula). The projection onto the half-space $\{\mathbf{x} : \mathbf{a}^\top\mathbf{x} \geq$

$b\}$ follows from the hyperplane case. A point that violates the constraint moves along $\mathbf{a}$ onto the boundary hyperplane, and a point that satisfies it stays, so

$$\Pi(\mathbf{y}) = \mathbf{y} + \frac{\max(0,\ b - \mathbf{a}^\top \mathbf{y})}{\|\mathbf{a}\|^2}\,\mathbf{a}.$$

Projections onto closed convex sets have a first-order characterization. The point $\hat{\mathbf{x}} = \Pi_C(\mathbf{y})$ minimizes the convex function $\tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$ over C. For any $\mathbf{x} \in C$ the whole segment from $\hat{\mathbf{x}}$ to $\mathbf{x}$ lies in C, so the directional derivative of the objective along it must be nonnegative: $(\hat{\mathbf{x}} - \mathbf{y})^\top (\mathbf{x} - \hat{\mathbf{x}}) \geq 0$. Rearranged,

$$\left(\mathbf{y} - \Pi_C(\mathbf{y})\right)^\top \left(\mathbf{x} - \Pi_C(\mathbf{y})\right) \ \leq\ 0 \qquad \text{for all } \mathbf{x} \in C. \tag{35.67}$$

The *residual* $\mathbf{y} - \Pi_C(\mathbf{y})$ makes an angle of at least 90° with the direction to every point of C (Figure 35.20, panel (a)). Equivalently, C lies entirely on the far side of the hyperplane through $\Pi_C(\mathbf{y})$ with normal $\mathbf{y} - \Pi_C(\mathbf{y})$, a *supporting hyperplane* of C.

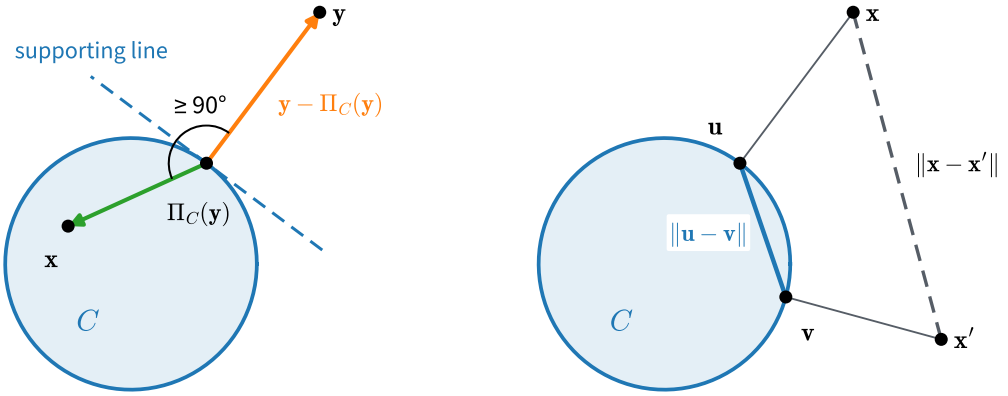


Figure 35.20: (a) The residual $\mathbf{y} - \Pi_C(\mathbf{y})$ makes an angle of at least 90° with every direction into C: the set lies on the far side of the supporting line through $\Pi_C(\mathbf{y})$. (b) Projection never increases distances: $\|\mathbf{u} - \mathbf{v}\| \leq \|\mathbf{x} - \mathbf{x}'\|$.

For the running example the inequality can be checked by hand. The residual of $\mathbf{y} = (3, 4)$ is $(2.4, 3.2) = 4\,(0.6, 0.8)$, the multiplier times the constraint normal once more. For any $\mathbf{x}$ in the unit disk,

$$(2.4, 3.2)^\top (\mathbf{x} - (0.6, 0.8)) = 4\,(0.6x_1 + 0.8x_2) - 4 \ \leq\ 4\,\|\mathbf{x}\| - 4 \ \leq\ 0,$$

by the Cauchy–Schwarz inequality $\mathbf{u}^\top \mathbf{v} \leq \|\mathbf{u}\|\,\|\mathbf{v}\|$. The supporting line is $3x_1 + 4x_2 = 5$, the hyperplane of the closest-point example.

The inequality is also *sufficient*. Suppose $\hat{\mathbf{x}} \in C$ satisfies it, and let $\varphi(\mathbf{x}) = \tfrac{1}{2}\|\mathbf{x} - \mathbf{y}\|^2$, whose gradient at $\hat{\mathbf{x}}$ is $\hat{\mathbf{x}} - \mathbf{y}$. A convex differentiable function lies above its tangent planes, $\varphi(\mathbf{x}) \geq \varphi(\hat{\mathbf{x}}) + \nabla\varphi(\hat{\mathbf{x}})^\top (\mathbf{x} - \hat{\mathbf{x}})$ (Section 35.3). For every $\mathbf{x} \in C$ the last term is nonnegative by

assumption, so $\varphi(\mathbf{x}) \geq \varphi(\hat{\mathbf{x}})$ and $\hat{\mathbf{x}} = \Pi_C(\mathbf{y})$. So Equation 35.67 *characterizes* the projection. Everything else we need about projections follows from it, first of all the property that makes projected algorithms converge.

Proposition (projections are nonexpansive). *For a nonempty closed convex set C and all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$,*

$$\|\Pi_C(\mathbf{x}) - \Pi_C(\mathbf{y})\| \;\leq\; \|\mathbf{x} - \mathbf{y}\|.$$

Proof. Write $\mathbf{u} = \Pi_C(\mathbf{x})$ and $\mathbf{v} = \Pi_C(\mathbf{y})$. Apply Equation 35.67 twice, at $\mathbf{x}$ with the test point $\mathbf{v}$ and at $\mathbf{y}$ with the test point $\mathbf{u}$:

$$(\mathbf{x} - \mathbf{u})^\top (\mathbf{v} - \mathbf{u}) \leq 0, \qquad (\mathbf{y} - \mathbf{v})^\top (\mathbf{u} - \mathbf{v}) \leq 0.$$

Adding them gives $(\mathbf{x} - \mathbf{y} - (\mathbf{u} - \mathbf{v}))^\top (\mathbf{v} - \mathbf{u}) \leq 0$, which rearranges to

$$\|\mathbf{u} - \mathbf{v}\|^2 \;\leq\; (\mathbf{x} - \mathbf{y})^\top (\mathbf{u} - \mathbf{v}) \;\leq\; \|\mathbf{x} - \mathbf{y}\|\,\|\mathbf{u} - \mathbf{v}\|$$

by Cauchy–Schwarz. Divide by $\|\mathbf{u} - \mathbf{v}\|$ (the claim is trivial when it is zero). ■

Panel (b) of Figure 35.20 shows the statement: two points outside the disk are farther apart than their projections. The proof needs convexity of C. Let $C = \{-1, 1\}$, two points. Then $y = 0.01$ projects to 1 and $y = -0.01$ projects to -1: inputs 0.02 apart give outputs 2 apart, and at $y = 0$ the projection is not even unique. Projection onto a nonconvex set can be discontinuous and multivalued.

35.6.3.2 Projected Gradient Descent

Projected gradient descent (PGD) follows each gradient step by a projection. With k counting iterations,

$$\mathbf{x}_{k+1} = \Pi_C\left(\mathbf{x}_k - \eta\, \nabla f(\mathbf{x}_k)\right). \tag{35.68}$$

The iterates are feasible by construction. Their fixed points are the constrained stationary points: for differentiable f, closed convex C, and any $\eta > 0$, the equation $\mathbf{x}^\star = \Pi_C(\mathbf{x}^\star - \eta \nabla f(\mathbf{x}^\star))$ holds if and only if $\mathbf{x}^\star \in C$ satisfies Equation 35.67 with $\mathbf{y} = \mathbf{x}^\star - \eta \nabla f(\mathbf{x}^\star)$. After the $\mathbf{x}^\star$ terms cancel and we divide by η, that inequality reads

$$\nabla f(\mathbf{x}^\star)^\top (\mathbf{x} - \mathbf{x}^\star) \;\geq\; 0 \qquad \text{for all } \mathbf{x} \in C.$$

No feasible direction is a descent direction. This is the constrained first-order condition, and it reduces to $\nabla f = \mathbf{0}$ when $C = \mathbb{R}^n$. For convex f it is also sufficient for a global minimum, by the tangent-plane inequality used above. For nonconvex f it certifies only stationarity. When $C = \{\mathbf{x} : g_i(\mathbf{x}) \leq 0\}$ with differentiable convex g_i and Slater's condition holds, the condition is equivalent to the KKT conditions.

The projection does not slow gradient descent down, and nonexpansiveness gives a three-line proof.

Proposition (PGD contracts at least as fast as GD). *Let C be nonempty, closed, and convex, let f be twice continuously differentiable with $\mu\mathbf{I} \preceq \nabla^2 f \preceq L\mathbf{I}$ for some $0 < \mu \le L$, let $\mathbf{x}^\star$ be the minimizer of f over C, and let $\eta > 0$. Then the iterates of Equation 35.68 satisfy*

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \;\le\; \rho\,\|\mathbf{x}_k - \mathbf{x}^\star\|, \qquad \rho = \max\left(|1-\eta\mu|,\; |1-\eta L|\right). \tag{35.69}$$

Proof. (i) By the fixed-point property, $\mathbf{x}^\star = \Pi_C(\mathbf{b})$ with $\mathbf{b} = \mathbf{x}^\star - \eta\nabla f(\mathbf{x}^\star)$. (ii) Choosing this $\mathbf{b}$ is the one non-routine step: it lets nonexpansiveness, applied to $\mathbf{x}_k - \eta\nabla f(\mathbf{x}_k)$ and $\mathbf{b}$, remove the projection,

$$\|\mathbf{x}_{k+1} - \mathbf{x}^\star\| \;\le\; \|(\mathbf{x}_k - \mathbf{x}^\star) - \eta\left(\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star)\right)\|.$$

(iii) By the fundamental theorem of calculus along the segment from $\mathbf{x}^\star$ to $\mathbf{x}_k$, $\nabla f(\mathbf{x}_k) - \nabla f(\mathbf{x}^\star) = \bar{\mathbf{H}}(\mathbf{x}_k - \mathbf{x}^\star)$ with $\bar{\mathbf{H}} = \int_0^1 \nabla^2 f(\mathbf{x}^\star + t(\mathbf{x}_k - \mathbf{x}^\star))\,dt$. This average of Hessians is symmetric with eigenvalues in $[\mu, L]$, so $\mathbf{I} - \eta\bar{\mathbf{H}}$ has eigenvalues $1 - \eta h$ with $h \in [\mu, L]$, all of magnitude at most ρ. Hence the right side is at most $\rho\,\|\mathbf{x}_k - \mathbf{x}^\star\|$. ■

Once step (ii) has removed the projection, what remains is plain gradient descent's contraction. For a quadratic $f = \frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ we have $\bar{\mathbf{H}} = \mathbf{A}$, and step (iii) is the per-mode factor $|1 - \eta\lambda_i(\mathbf{A})|$ of gradient descent from Section 35.1; the factor is below 1 if and only if $0 < \eta < 2/L$. For a convex f that is L-smooth but not strongly convex, PGD with $\eta = 1/L$ still has the $O(1/k)$ rate of gradient descent in the objective value (Nesterov 2018).

How tight is the factor ρ? Take the quadratic $f(\mathbf{x}) = \frac{1}{2}(x_1^2 + 1000\,x_2^2)$ with condition number $\kappa = 10^3$, restricted to the half-space $x_1 + x_2 \ge 1$, that is $g(\mathbf{x}) = 1 - x_1 - x_2 \le 0$. The KKT conditions can be solved by hand. The unconstrained minimizer $\mathbf{0}$ is infeasible, so the constraint is active. Stationarity, $(x_1, 1000\,x_2) - \lambda(1,1) = \mathbf{0}$, gives $x_1 = \lambda$ and $x_2 = \lambda/1000$, and the active constraint $x_1 + x_2 = 1$ gives $\lambda^\star = 1000/1001 \approx 0.999$, $\mathbf{x}^\star = (1000/1001,\; 1/1001) \approx (0.999, 0.001)$, and $p^\star = 500/1001 \approx 0.4995$.

Take the step $\eta = 0.0015$, an unremarkable value below the ceiling $2/L = 0.002$. The factor of the proposition is $\rho = \max(1-\eta,\; |1 - 1000\,\eta|) = 0.9985$. From the starting point $(3, 4)$, at distance 4.47 from $\mathbf{x}^\star$, this factor guarantees a distance of 10^{-6} after about $\log(4.47\times10^6)/\log(1/\rho) \approx 10{,}200$ steps. The run below records the largest ratio of successive distances, the number of steps it needs, the step from which every update is projected, the ratio per step from then on, and the gradient where it stops.

```
A = np.diag([1.0, 1000.0])                      # f(x) = 1/2 x^T A x, kappa = 1000
x_star = np.array([1000 / 1001, 1 / 1001])      # KKT point of x1 + x2 >= 1
project = lambda y: y + max(0.0, 1 - y.sum()) / 2  # onto the half-space x1 + x2 >= 1
eta = 1.5e-3
rho = max(abs(1 - eta), abs(1 - 1000 * eta))    # the GD contraction factor
x, dist, projected = np.array([3.0, 4.0]), [], []
for k in range(1000):
    y = x - eta * A @ x
    projected.append(y.sum() < 1)               # did the step leave C?
    x = project(y)
    dist.append(np.linalg.norm(x - x_star))
dist, projected = np.array(dist), np.array(projected)
ratios = dist[1:] / dist[:-1]
steps = np.argmax(dist < 1e-6) + 1
```

```
on_line = np.flatnonzero(~projected).max() + 2     # every step from here on is projected
print(f'bound rho = {rho:.4f}   largest ratio in a step = '
      f'{ratios[dist[:-1] > 1e-12].max():.4f}')
print(f'steps to |x - x*| < 1e-6: {steps}   every step projected from step {on_line} on')
print(f'ratio per step on the line: {np.median(ratios[on_line:steps - 1]):.4f}'
      f'   1 - 500.5 eta = {1 - 500.5 * eta:.4f}')
print(f'grad f at the end = {A @ x}')
```

```
bound rho = 0.9985   largest ratio in a step = 0.9977
steps to |x - x*| < 1e-6: 738   every step projected from step 733 on
ratio per step on the line: 0.2492   1 - 500.5 eta = 0.2492
grad f at the end = [0.999001 0.999001]
```

The largest ratio in a step, 0.9977, comes close to ρ, so the per-step bound is nearly tight for this problem. The step count is another matter: 738, not $10{,}200$. The printed numbers explain the difference, because the run has two phases. In the first, the constraint is inactive after the first step. The stiff coordinate x_2 dies out within a few dozen steps, and the flat coordinate x_1 shrinks by the factor $1-\eta$ per step, as in unconstrained gradient descent. Reaching the line from $x_1 = 3$ takes about $\log 3/\eta \approx 730$ steps, and from step 733 on every step is projected. In the second phase the iterate moves along the line $x_1 + x_2 = 1$, where the quadratic has curvature $(1+1000)/2 = 500.5$, and the distance shrinks by $1-500.5\,\eta = 0.249$ per step, the factor of gradient descent on that one-dimensional problem. A handful of further steps finish the run. The proposition's ρ bounds the worst step anywhere in the plane. Once the constraint is active only the curvature along the constraint counts, and here that curvature is far better matched to the step. The bound is thus tight for a single step and pessimistic for the run. What does carry over unchanged is the last number printed. The final gradient $(0.999, 0.999)$ equals $\lambda^\star(1,1) = -\lambda^\star \nabla g$: it does not vanish, and it hands us the multiplier $1000/1001$ computed by hand.

PGD is practical whenever Π_C is cheap. Nonnegativity constraints project onto the orthant by setting negative entries to zero, and max-norm constraints on the rows of a weight matrix rescale each row onto a ball after the update. The use most readers will have met is the search for adversarial examples: projected gradient ascent on an input perturbation δ with $\|\delta\|_\infty \le \epsilon$, where the projection clips each coordinate to $[-\epsilon, \epsilon]$ (Madry et al. 2018). Gradient clipping also projects, but it projects the *gradient vector* onto a ball, and it is not PGD for any constraint on the parameters (Exercise 3 shows the difference).

35.6.3.3 Projection onto the Simplex

Attention weights and class probabilities live on the **probability simplex** $\Delta = \{\mathbf{x} : \mathbf{x} \ge 0,\ \sum_i x_i = 1\}$. What is the point of Δ nearest a vector of scores $\mathbf{y}$? The problem

$$\min_{\mathbf{x}} \ \tfrac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2 \quad \text{subject to} \quad \textstyle\sum_i x_i = 1, \ \ -x_i \le 0,$$

is a small **quadratic program** (QP), a convex quadratic objective with affine constraints. The KKT conditions reduce it to finding one number. With a multiplier τ for the sum and $\lambda_i \ge 0$ for each sign constraint, stationarity reads $x_i - y_i + \tau - \lambda_i = 0$. Complementary slackness splits the coordinates. Where $x_i > 0$ we get $\lambda_i = 0$ and $x_i = y_i - \tau$. Where $x_i = 0$, stationarity gives $\lambda_i = \tau - y_i$, and dual feasibility $\lambda_i \ge 0$ says $y_i \le \tau$. Both cases combine as

$$x_i^\star = \max(y_i - \tau,\, 0), \qquad \text{with } \tau \text{ set by } \sum_i \max(y_i - \tau,\, 0) = 1. \tag{35.70}$$

Every score is shifted down by the same τ and clipped at zero. The threshold form satisfies stationarity, complementary slackness, dual feasibility, and the sign constraints for *every* τ. The budget $\sum_i x_i = 1$ alone fixes τ. The left side of the budget equation, $\varphi(\tau) = \sum_i \max(y_i - \tau, 0)$, is continuous, piecewise linear, and strictly decreasing where it is positive, so exactly one τ solves it.

A hand example shows how to find it. Take $\mathbf{y} = (0.9, 0.5, -0.2)$, already sorted, and guess that the k largest scores are positive. The budget then reads $\sum_{j \le k}(y_j - \tau) = 1$, so the guess predicts $\tau_k = (\sum_{j \le k} y_j - 1)/k$, and it is consistent only if the k-th score stays above the threshold:

- $k = 1$: $\tau_1 = -0.1$, and $y_1 = 0.9 > \tau_1$;
- $k = 2$: $\tau_2 = 0.2$, and $y_2 = 0.5 > \tau_2$;
- $k = 3$: $\tau_3 \approx 0.067$, but $y_3 = -0.2 < \tau_3$.

So $k = 2$, $\tau = 0.2$, and $\mathbf{x}^\star = (0.7, 0.3, 0)$ (Figure 35.21). Softmax of the same scores is $(0.50, 0.33, 0.17)$, positive everywhere.

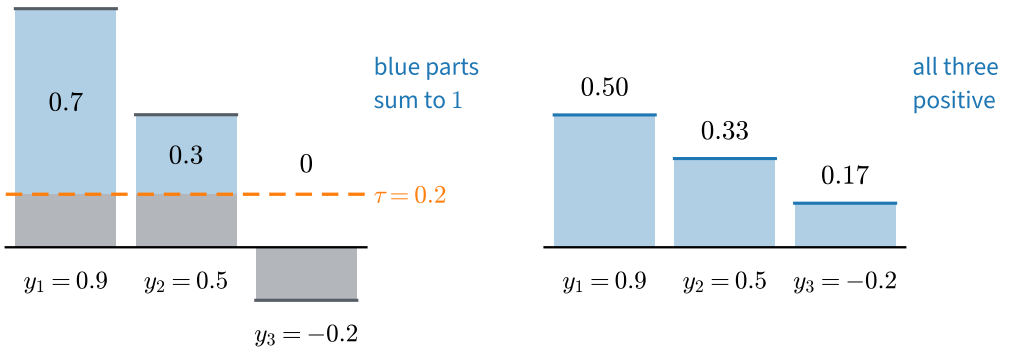


Figure 35.21: Projection of $\mathbf{y} = (0.9, 0.5, -0.2)$ onto the simplex (a): every score is shifted down by the threshold $\tau = 0.2$ and clipped at zero, and the budget $\sum_i x_i = 1$ fixes τ. Softmax of the same scores (b) keeps every coordinate positive.

Two facts make the hand procedure an algorithm. First, the *support* $\{i : x_i^\star > 0\}$ consists of the largest scores: if $y_i > y_j$ and $x_j^\star > 0$, then $x_i^\star \ge y_i - \tau > y_j - \tau = x_j^\star > 0$. (The *active* sign constraints are the complement, the zero coordinates.) Second, with the scores sorted as $u_1 \ge \cdots \ge u_n$, the test $u_k > \tau_k$ passes for k up to the support size and fails beyond it. Indeed $u_k > \tau_k$ rearranges to $\sum_{j<k}(u_j - u_k) < 1$, that is $\varphi(u_k) < 1 = \varphi(\tau)$, and since φ decreases this holds exactly when $u_k > \tau$. So we sort, take the largest k with $u_k > \tau_k$, and set $\tau = \tau_k$, which costs $O(n \log n)$ (Held et al. 1974; Duchi et al. 2008).

Applied to attention scores, this projection is *sparsemax* (Martins and Astudillo 2016), which puts exactly zero weight on low-scoring positions. The two maps differ in one term. Expand-

ing the squared distance shows that sparsemax maximizes $\mathbf{y}^\top \mathbf{x} - \frac{1}{2}\|\mathbf{x}\|^2$ over Δ, while softmax maximizes $\mathbf{y}^\top \mathbf{x} + H(\mathbf{x})$ with the entropy H in place of the quadratic (Equation 35.87).

A computed projection should be tested, and the KKT residuals are the obvious candidates. The cell evaluates them for six random scores, at the computed τ and at a wrong one, and then evaluates the certificate that Equation 35.67 provides. The left side of that inequality is affine in $\mathbf{x}$, so its maximum over Δ is attained at a vertex, and checking the n vertices $\mathbf{e}_i$ suffices.

```
def project_simplex(y):
    """Sort-and-threshold projection onto {x >= 0, sum x = 1}; returns x* and tau."""
    u = np.sort(y)[::-1]
    tau_k = (np.cumsum(u) - 1) / np.arange(1, len(y) + 1)   # tau_k for k = 1..n
    k = np.nonzero(u > tau_k)[0][-1]                          # largest k with u_k > tau_k
    return np.maximum(y - tau_k[k], 0.0), tau_k[k]

y = np.random.default_rng().normal(size=6)
x, tau = project_simplex(y)
print('x* =', x.round(3), ' zeros:', int((x == 0).sum()), ' tau =', round(tau, 3))
for t in (tau, tau + 0.3):                    # three conditions hold for every tau ...
    xt, lam = np.maximum(y - t, 0), np.maximum(0, t - y)
    print(f'tau = {t:6.3f}: stationarity {abs(xt - y + t - lam).max():.0e}, '
          f'comp. slack {abs(lam * xt).max():.0e}, dual feas. {lam.min() >= 0}, '
          f'sum x = {xt.sum():.3f}')           # ... only the budget differs
cert = ((y - x) @ (np.eye(6) - x).T).max()     # max_i (y - x*)^T (e_i - x*)
print(f'certificate max_i (y - x*)^T(e_i - x*) = {cert:.0e}  '
      f'(<= 0 up to rounding: {cert <= 1e-12})')
```

```
x* = [0.666 0.334 0.    0.    0.    0.   ]  zeros: 4  tau = 1.662
tau =  1.662: stationarity 0e+00, comp. slack 0e+00, dual feas. True, sum x = 1.000
tau =  1.962: stationarity 0e+00, comp. slack 0e+00, dual feas. True, sum x = 0.400
certificate max_i (y - x*)^T(e_i - x*) = -3e-16  (<= 0 up to rounding: True)
```

Typically three to five of the six coordinates are exactly zero. These are the active sign constraints, each with the positive multiplier $\tau - y_i$. The stationarity and complementary-slackness residuals are zero up to rounding for the correct τ *and* for the wrong one, and dual feasibility holds for both. Only the sum differs, so these residuals cannot confirm the solution. The certificate does: the largest value of the variational inequality over the vertices is at most rounding error, which proves that no point of the simplex is closer to $\mathbf{y}$.

35.6.4 Proximal Operators and Coordinate Descent

35.6.4.1 Soft-Thresholding and the Proximal Operator

A hard wall is not the only way to restrict weights. The **lasso** (Tibshirani 1996) adds a kinked penalty to a least-squares loss,

$$F(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2 + \lambda\|\mathbf{w}\|_1, \qquad \mathbf{A} \in \mathbb{R}^{n\times d},$$

and its minimizer typically has many coordinates exactly zero. The penalty weight is written λ because it plays the role of the multiplier of a constraint $\|\mathbf{w}\|_1 \le c$: every minimizer of F also solves the least-squares problem under the budget c that it uses up, and every budget

$c > 0$ has its λ. Section 35.7.3.2 proves both directions, for this penalty and for weight decay.

We plant a sparse $\mathbf{w}$ with five of its eight coordinates zero, generate $n = 100$ noisy observations, and set $\lambda = 3$. The step size is $\eta = 1/L$ with $L = \lambda_{\max}(\mathbf{A}^\top \mathbf{A})$, the largest eigenvalue of $\mathbf{A}^\top \mathbf{A}$ and the smoothness constant of the least-squares term.

```
rng = np.random.default_rng()
w_planted = np.array([2.0, 0, 0, -1.5, 0, 0, 0, 1.0])
A = rng.normal(size=(100, 8))
b = A @ w_planted + 0.05 * rng.normal(size=100)
lam = 3.0
L = np.linalg.eigvalsh(A.T @ A).max()
eta = 1 / L
F = lambda w: 0.5 * ((A @ w - b)**2).sum() + lam * np.abs(w).sum()
soft = lambda z, t: np.sign(z) * np.maximum(np.abs(z) - t, 0.0)  # derived below
print(f'L = lambda_max(A^T A) = {L:.1f};  |a_j|^2 range from '
      f'{(A**2).sum(0).min():.0f} to {(A**2).sum(0).max():.0f}')
```

```
L = lambda_max(A^T A) = 145.6;  |a_j|^2 range from 85 to 109
```

The output gives L between about 130 and 190 and squared column norms $\|\mathbf{a}_j\|^2$ around 100, where $\mathbf{a}_j$ is the j-th column of $\mathbf{A}$.

A first method to try is subgradient descent. The subgradients of $|w|$ (Section 35.3) are $\operatorname{sign}(w)$ away from zero and every number in $[-1, 1]$ at the kink. Subgradient descent picks one, $\mathbf{w} \leftarrow \mathbf{w} - \eta\,(\mathbf{A}^\top(\mathbf{A}\mathbf{w} - \mathbf{b}) + \lambda \operatorname{sign}(\mathbf{w}))$. Does it find the five zeros?

```
w_sub, signs = np.zeros(8), []
for k in range(50):                                  # subgradient descent, step 1/L
    w_sub = w_sub - eta * (A.T @ (A @ w_sub - b) + lam * np.sign(w_sub))
    signs.append(np.sign(w_sub))
null = w_planted == 0
S = np.array(signs)[-20:, null]
print(f'exact zeros: {int((w_sub == 0).sum())} of 8;  sign flips of the null '
      f'coordinates, last 20 steps: {(S[1:] != S[:-1]).sum()} of 95')
print(f'largest null coordinate: {np.abs(w_sub[null]).max():.3f};  '
      f'eta * lam = {eta * lam:.3f}')
```

```
exact zeros: 0 of 8;  sign flips of the null coordinates, last 20 steps: 95 of 95
largest null coordinate: 0.033;  eta * lam = 0.021
```

It does not. No coordinate is exactly zero, and the five *null* coordinates, those planted at zero, change sign at most steps, usually at every one, at a size of about $\eta\lambda$. The cause is the kink. At the lasso solution the smooth part of the gradient of a null coordinate is smaller than λ in magnitude. Near zero the step is therefore dominated by $-\eta\lambda \operatorname{sign}(w_j)$, which carries the coordinate across zero by about $\eta\lambda$ every time. The method uses one element of the interval $[-\lambda, \lambda]$ of subgradients of $\lambda|w_j|$ at zero, and with a fixed step no single element balances the smooth part.

The remedy is to solve the kinked part exactly. Start with one coordinate: minimize $\lambda|x| + \frac{1}{2}(x - z)^2$ over $x \in \mathbb{R}$. The objective is convex, so a point minimizes it if and only if 0 is one of its subgradients there.

- If the minimizer is $x \neq 0$, the objective is differentiable there and $\lambda \operatorname{sign}(x) + x - z = 0$. For $x > 0$ this gives $x = z - \lambda$, which is consistent only if $z > \lambda$; for $x < 0$ it gives $x = z + \lambda$, consistent only if $z < -\lambda$.
- If the minimizer is $x = 0$, the subgradients there are $[-\lambda, \lambda] - z$, and 0 lies in this interval if and only if $|z| \leq \lambda$.

The two cases cover every z, and together they give **soft-thresholding**,

$$\operatorname{prox}_{\lambda|\cdot|}(z) \;=\; \operatorname{sign}(z)\, \max\left(|z| - \lambda,\, 0\right). \tag{35.71}$$

Inputs of size at most λ are set to exactly zero, and larger ones are shrunk toward zero by λ. With $\lambda = 0.7$: $z = 0.5$ maps to 0, $z = 2$ to 1.3, and $z = -2$ to -1.3. This threshold is how ℓ_1 regularization produces exact zeros, where ℓ_2 regularization only shrinks.

The operation generalizes. The **proximal operator** of a function ψ finds the point that balances a lower value of ψ against the cost of moving away from $\mathbf{z}$:

$$\operatorname{prox}_{\psi}(\mathbf{z}) \;=\; \operatorname*{argmin}_{\mathbf{x}} \; \left(\psi(\mathbf{x}) + \tfrac{1}{2}\,\|\mathbf{x} - \mathbf{z}\|^2\right). \tag{35.72}$$

The prox is well defined for every *proper closed convex* ψ: convex, with values in $\mathbb{R} \cup \{+\infty\}$, finite somewhere, and with a closed epigraph. The objective in Equation 35.72 is then 1-strongly convex and has exactly one minimizer (Parikh and Boyd 2014).

Three choices of ψ show what the definition does:

- $\psi = 0$ gives the identity.
- ψ the *indicator* of a closed convex set C (zero on C, $+\infty$ outside) forces $\mathbf{x} \in C$, and the prox is the nearest point of C: $\operatorname{prox}_{\psi} = \Pi_C$. Proximal operators are projections, generalized from sets to functions.
- $\psi = \lambda\|\cdot\|_1$ separates into a sum over coordinates, so its prox is soft-thresholding Equation 35.71 applied to each coordinate.

Exercise 7 derives the prox of the Euclidean norm, which zeroes a whole block of weights at once. Exercise 8 shows that the prox of a norm and the projection onto a related ball add up to the identity.

35.6.4.2 Proximal Gradient Descent

The lasso has the form $F = s + \psi$: a smooth part s and a kinked part ψ whose prox is cheap. Assume that s is convex with an L-Lipschitz gradient, that ψ is proper closed convex, and that $0 < \eta \leq 1/L$. An L-smooth s lies below the quadratic $s(\mathbf{x}_k) + \nabla s(\mathbf{x}_k)^\top(\mathbf{x} - \mathbf{x}_k) + \frac{1}{2\eta}\|\mathbf{x} - \mathbf{x}_k\|^2$ (the descent lemma of Section 35.1, with $1/\eta \geq L$). Instead of linearizing ψ as well, we minimize this upper bound of s plus the exact ψ. Completing the square shows that the minimizer is

$$\operatorname*{argmin}_{\mathbf{x}} \; \psi(\mathbf{x}) + \frac{1}{2\eta}\left\|\mathbf{x} - (\mathbf{x}_k - \eta\nabla s(\mathbf{x}_k))\right\|^2 \;=\; \operatorname{prox}_{\eta\psi}\left(\mathbf{x}_k - \eta\,\nabla s(\mathbf{x}_k)\right),$$

a gradient step on the smooth part followed by a prox on the kinked part. The resulting iteration,

$$\mathbf{x}_{k+1} = \operatorname{prox}_{\eta\psi}\left(\mathbf{x}_k - \eta\,\nabla s(\mathbf{x}_k)\right), \tag{35.73}$$

is **proximal gradient descent**, called **ISTA** (iterative soft-thresholding) when $\psi = \lambda\|\cdot\|_1$. It contains projected gradient descent (take ψ the indicator of C) and gradient descent ($\psi = 0$).

Under the assumptions above, with $\eta = 1/L$, proximal gradient descent satisfies $F(\mathbf{x}_k) - F^\star \le L\|\mathbf{x}_0 - \mathbf{x}^\star\|^2/(2k)$ (Beck and Teboulle (2009), Theorem 3.1). This is the $O(1/k)$ rate of gradient descent, although F is not differentiable. If s is also twice continuously differentiable with $\mu\mathbf{I} \preceq \nabla^2 s$ for some $\mu > 0$, the proof of the PGD contraction goes through word for word. It needs only that $\mathbf{x}^\star$ is a fixed point of the update and that the prox is nonexpansive like the projection (Exercise 9 proves this).

On the lasso the prox is soft-thresholding with threshold $\eta\lambda$, so every coordinate that the gradient step leaves within $\eta\lambda$ of zero is set to exactly zero. That is the size at which the null coordinates of the subgradient run kept changing sign, so ISTA should hold them at zero.

```
w = np.zeros(8)
for k in range(50):                                  # ISTA: gradient step, then prox
    w = soft(w - eta * A.T @ (A @ w - b), eta * lam)
print(f'exact zeros: {int((w == 0).sum())} of 8;  support recovered: '
      f'{np.array_equal(w != 0, w_planted != 0)};  '
      f'max deviation on the support: {np.abs(w - w_planted)[~null].max():.3f}')
```

```
exact zeros: 5 of 8;  support recovered: True;  max deviation on the support: 0.029
```

After the same fifty steps, at the same cost per step as the subgradient method, ISTA sets the five null coordinates to zero and no others. The three nonzero coordinates sit within about 0.05 of the planted values. They cannot sit much closer, because the lasso solution itself is shrunk: a coordinate in the support is pulled toward zero by about $\lambda/\|\mathbf{a}_j\|^2 \approx 0.03$.

35.6.4.3 Coordinate Descent

This subsection can be skipped on a first reading: coordinate descent is the standard solver for the lasso, but it is not how networks are trained.

On the lasso the exact minimizer along a single coordinate is cheap, because it is the scalar problem just solved. Fix every coordinate but w_j, and let $\mathbf{r}_{-j} = \mathbf{b} - \sum_{i\neq j}\mathbf{a}_i w_i$ be the residual without coordinate j. The j-th subproblem, $\frac{1}{2}\|\mathbf{a}_j w_j - \mathbf{r}_{-j}\|^2 + \lambda|w_j|$, equals $\|\mathbf{a}_j\|^2\left(\frac{1}{2}(w_j - c_j/\|\mathbf{a}_j\|^2)^2 + (\lambda/\|\mathbf{a}_j\|^2)|w_j|\right)$ plus a constant, and soft-thresholding with threshold $\lambda/\|\mathbf{a}_j\|^2$ solves it:

$$w_j \leftarrow \frac{\operatorname{sign}(c_j)\,\max(|c_j| - \lambda,\,0)}{\|\mathbf{a}_j\|^2}, \qquad c_j = \mathbf{a}_j^\top \mathbf{r}_{-j}. \tag{35.74}$$

A coordinate whose correlation c_j with the remaining residual is at most λ is set to zero. Cycling through the coordinates with this update is the algorithm of the glmnet package (Friedman et al. 2010). With the residual kept up to date, a sweep costs as much as one gradient.

The same idea applies to any differentiable f with coordinate-wise smoothness constants L_j,

$$f(\mathbf{x} + t\mathbf{e}_j) \le f(\mathbf{x}) + t\,\partial_j f(\mathbf{x}) + \tfrac{1}{2}L_j t^2, \tag{35.75}$$

where minimizing the bound over t gives the **coordinate-descent** step

$$x_j \leftarrow x_j - \frac{1}{L_j}\partial_j f(\mathbf{x}). \tag{35.76}$$

Each L_j is the curvature along one axis and can be much smaller than the global L: for the lasso's smooth part $L_j = \|\mathbf{a}_j\|^2$, about 100, against an L of 130 to 190. On a quadratic $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x} - \mathbf{b}^\top\mathbf{x}$ with $\mathbf{A}$ positive definite the bound is an equality with $L_j = A_{jj}$, and the step

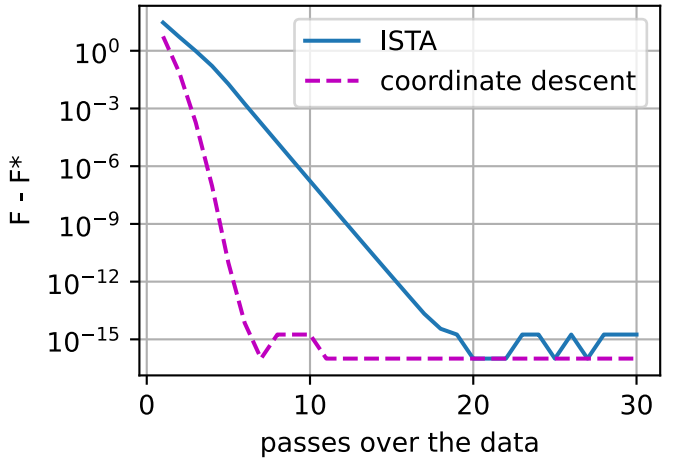
$$x_j \leftarrow x_j - \frac{(\mathbf{A}\mathbf{x} - \mathbf{b})_j}{A_{jj}} \tag{35.77}$$

minimizes f along the coordinate exactly. A cyclic sweep of these steps is the Gauss–Seidel iteration for $\mathbf{A}\mathbf{x} = \mathbf{b}$.

What is a sweep worth, compared with the ISTA step it costs? We count the passes over the data that each method needs to reach a gap of 10^{-8} on our lasso.

```
w_ref = np.zeros(8)
for k in range(5000):                               # reference optimum by ISTA
    w_ref = soft(w_ref - eta * A.T @ (A @ w_ref - b), eta * lam)
F_star, col2 = F(w_ref), (A**2).sum(0)              # col2[j] = |a_j|^2
w_i, w_c, r = np.zeros(8), np.zeros(8), b.copy()    # r = b - A w_c, kept up to date
gap_i, gap_c = [], []
for sweep in range(30):
    w_i = soft(w_i - eta * A.T @ (A @ w_i - b), eta * lam)   # one ISTA step
    for j in range(8):                                       # one coordinate sweep
        c_j = A[:, j] @ r + col2[j] * w_c[j]                 # a_j^T r_{-j}
        w_new = soft(c_j, lam) / col2[j]
        r -= A[:, j] * (w_new - w_c[j])
        w_c[j] = w_new
    gap_i.append(F(w_i) - F_star); gap_c.append(F(w_c) - F_star)
passes = lambda gap: next((k + 1 for k, g in enumerate(gap) if g < 1e-8),
                          '> 30')                        # not reached in 30 passes
print(f'passes to gap < 1e-8: ISTA {passes(gap_i)}, coordinate descent {passes(gap_c)}')
d2l.plot(np.arange(1, 31), [np.maximum(gap_i, 1e-16), np.maximum(gap_c, 1e-16)],
         'passes over the data', 'F - F*', legend=['ISTA', 'coordinate descent'],
         yscale='log')                                   # gaps floored at rounding level
```

```
passes to gap < 1e-8: ISTA 12, coordinate descent 5
```



Both methods converge linearly, as straight lines on the logarithmic axis, and coordinate descent is steeper: it reaches the gap in four to six sweeps, typically, where ISTA needs about fifteen to twenty-five steps. Nothing proved in this section predicts that factor. The known rates for coordinate descent are for coordinates drawn at random (Nesterov 2012), not for cyclic sweeps.

Coordinate descent has a limit. On a nonsmooth function whose kink is not a sum over coordinates it can stop short: for $f(x, y) = \max(|x|, |y|)$ at $(1, 1)$, no change of x alone or of y alone lowers f, although the minimum 0 lies at the origin. It works on the lasso because the kinked term $\lambda \sum_j |w_j|$ separates over coordinates (Tseng 2001) (Exercise 10). It remains the method of choice for sparse linear models and for alternating minimization over blocks of variables. It plays no part in training dense networks, because accelerators are built for large matrix products, not for sequential scalar updates.

Every multiplier in this section was found by solving for $\mathbf{x}$ first. Section 35.7 asks the reverse question: how to find the multipliers directly, and how a choice of multipliers certifies that no feasible point lies below the value reached. The clipping problem, with $\lambda^\star = 4$ and $p^\star = 8$, is its first example.

35.6.5 Summary

A constrained optimum is a point with no feasible descent direction. With one equality constraint and $\nabla h(\mathbf{x}^\star) \neq \mathbf{0}$, this means $\nabla f + \nu \nabla h = \mathbf{0}$ for a multiplier ν that measures how hard the optimum presses on the constraint. With inequalities the condition becomes the four KKT conditions, and complementary slackness $\lambda_i g_i = 0$ gives every inactive constraint a zero multiplier. For convex problems every KKT point is a global minimum, and under a constraint qualification (LICQ, or Slater's condition for convex problems) every minimum is a KKT point. Projection onto a closed convex set is unique, nonexpansive, and characterized by the obtuse-angle inequality, so projected gradient descent contracts the distance to the optimum at least as fast as gradient descent. The projections onto a ball, a half-space, and the simplex each reduce to finding one scalar, a multiplier. The proximal operator generalizes the projection from sets to functions. Soft-thresholding is the prox of $\lambda|\cdot|$, and ISTA and coordinate descent built on it solve the lasso with exact zeros, which subgradient steps do not produce.

35.6.6 Exercises

1. **Degenerate constraints.** The Lagrange condition Equation 35.61 and the KKT conditions Equation 35.65 hold at a minimum only under a constraint qualification. Each of the following problems violates one.
    1. Minimize $f(\mathbf{x}) = x_1 + x_2^2$ over $\mathbb{R}^3$ subject to $h_1(\mathbf{x}) = x_3 = 0$ and $h_2(\mathbf{x}) = x_3 - x_1^2 = 0$. Show that the feasible set is a line, find the minimizer, and show that both constraint gradients are nonzero everywhere. Show that no multipliers ν_1, ν_2 satisfy $\nabla f + \nu_1 \nabla h_1 + \nu_2 \nabla h_2 = \mathbf{0}$ at the minimizer. Describe the same line by two other equality constraints for which multipliers exist. Is a constraint qualification a property of the feasible set or of the functions that describe it?
    2. Minimize $f(x) = x$ over $\mathbb{R}$ subject to $x^2 \le 0$. Find $p^\star$ and show that no $\lambda \ge 0$ satisfies the KKT stationarity condition at the minimizer. Which of LICQ and Slater's condition fails? Explain why for each.
    3. Minimize $f(x) = x$ over $\mathbb{R}$ subject to the constraint $-x \le 0$ written twice, $g_1(x) = g_2(x) = -x$. Show that LICQ fails and Slater's condition holds. Find all pairs $(\lambda_1^\star, \lambda_2^\star)$ that satisfy the KKT conditions at the minimizer. Under the relaxed constraints $-x \le u_1$ and $-x \le u_2$, compute the optimal value $p^\star(u_1, u_2)$, and verify that every KKT pair satisfies $p^\star(\mathbf{u}) \ge p^\star(\mathbf{0}) - {\lambda^\star}^\top \mathbf{u}$ for all $\mathbf{u}$. Is $p^\star$ differentiable at $\mathbf{0}$, and what takes the place of the relation $\lambda^\star = -\nabla p^\star(\mathbf{0})$?
2. **KKT points that are not minima.** The KKT conditions Equation 35.65 are necessary at a local minimum under a constraint qualification, but a nonconvex problem can satisfy them at other points.
    1. Minimize $f(\mathbf{x}) = x_1^2 - x_2^2$ over the box $-1 \le x_1 \le 1$, $-1 \le x_2 \le 1$, written as four inequality constraints. Find all KKT points and their multipliers. Classify each KKT point as a local minimum, a local maximum, or neither.
    2. The maximum of f on the box is attained at $(\pm 1, 0)$. Which KKT condition fails at these points, and what value would the multiplier of the active constraint need?
    3. Give a problem in one variable with a single inequality constraint $g(x) \le 0$ that has a KKT point $x^\star$ which is a strict local maximum. Can the multiplier at such a point be positive when $g'(x^\star) \neq 0$? Compare with $\min -x^2$ on $[-1, 1]$ from the text.
3. **Projected fixed points.** For a closed convex set C, the fixed points of projected gradient descent Equation 35.68 are the constrained stationary points, whatever the step size $\eta > 0$.
    1. Let $C = \{\mathbf{x} : \mathbf{x} \ge 0\}$. Show that $\mathbf{x}^\star$ is a fixed point if and only if, for every coordinate i, either $\partial f / \partial x_i(\mathbf{x}^\star) = 0$ and $x_i^\star \ge 0$, or $\partial f / \partial x_i(\mathbf{x}^\star) > 0$ and $x_i^\star = 0$. Show that these conditions are the KKT conditions for the constraints $-x_i \le 0$, and express the multipliers through $\nabla f(\mathbf{x}^\star)$.
    2. Gradient clipping updates $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta \, \Pi_{B_c}(\nabla f(\mathbf{x}_k))$, where $B_c = \{\mathbf{z} : \|\mathbf{z}\| \le c\}$. Find the fixed points of this update and compare them with those of projected gradient descent onto the parameter ball $\{\mathbf{x} : \|\mathbf{x}\| \le r\}$. Which of the two updates solves a constrained problem?
    3. Binary weights restrict a parameter to the nonconvex set $C = \{-1, 1\}$, and projecting onto C rounds to the nearer element (to 1 at a tie). Let $f(x) = \frac{1}{2}(x - a)^2$ with $0 < a < 1$. For which step sizes is $x = 1$ a fixed point of Equation 35.68, and for which is $x = -1$ one? What do the iterates do when η exceeds both ranges?

Which point minimizes f over C? Does either point satisfy $f'(x^\star)(x - x^\star) \geq 0$ for all $x \in C$? Which step of the fixed-point argument fails for this set?

4. Run the PGD cell on the half-space with $\eta = 2.2/1000$ instead of 0.0015. The projection is nonexpansive; why does the run still diverge? Which factor of the contraction proposition exceeds 1?

4. code **Sparsemax and the ℓ_1 ball.** The function `project_simplex` computes the simplex projection Equation 35.70, which is sparsemax; its first output is the projected vector.

   1. Show that adding the same constant to every score leaves the projection unchanged. Divide the scores by a temperature $T > 0$, and let $u_1 > u_2$ be the two largest entries of $\mathbf{y}$. Show that the projection of $\mathbf{y}/T$ is one-hot exactly when $T \leq u_1 - u_2$, and find its limit as $T \to \infty$. Plot the number of nonzero entries of the projection of $\mathbf{y}/T$ against T for ten standard normal scores.
   2. Let S be the support of $\mathbf{x}^\star$, and suppose that no y_i equals the threshold τ. Derive the Jacobian $\partial \mathbf{x}^\star / \partial \mathbf{y}$ from Equation 35.70 and check it against central finite differences. How many arithmetic operations does the backward pass through sparsemax need for an input of length n?
   3. Show that the projection onto the ℓ_1 ball $\{\mathbf{x} : \|\mathbf{x}\|_1 \leq z\}$ keeps the signs of $\mathbf{y}$ and, when $\|\mathbf{y}\|_1 > z$, equals $\operatorname{sign}(\mathbf{y}) \odot z\, \Pi_\Delta(|\mathbf{y}|/z)$ (Duchi et al. 2008). Implement it with `project_simplex`. Certify the result with the variational inequality $(\mathbf{y} - \hat{\mathbf{x}})^\top (\mathbf{x} - \hat{\mathbf{x}}) \leq 0$, checked at the $2n$ points $\pm z\, \mathbf{e}_i$. Why do these points suffice?

5. **Second-order conditions on the sphere.** ⊠ The Lagrange condition is a first-order condition and cannot tell minima from saddles. A second-order test must account for the curvature of the constraint.

   1. Let f and h be twice continuously differentiable, and let $\mathbf{x}^\star$ satisfy Equation 35.61 with multiplier $\nu^\star$. For every curve $\mathbf{x}(t)$ on $\{h = 0\}$ with $\mathbf{x}(0) = \mathbf{x}^\star$ and $\dot{\mathbf{x}}(0) = \mathbf{v}$, show that

   $$\left. \frac{d^2}{dt^2} f(\mathbf{x}(t)) \right|_{t=0} = \mathbf{v}^\top \nabla_{\mathbf{x}}^2 \mathcal{L}(\mathbf{x}^\star, \nu^\star)\, \mathbf{v},$$

   with the Lagrangian Equation 35.62. Conclude that $\mathbf{v}^\top \nabla_{\mathbf{x}}^2 \mathcal{L}\, \mathbf{v} \geq 0$ for every tangent direction $\mathbf{v}$ at a local minimum. Why is the Hessian of f alone the wrong matrix for this test? Conversely, positive definiteness on the tangent space implies a strict local minimum (Nocedal and Wright (2006), chapter 12).

   2. Let $\mathbf{A}$ be symmetric with distinct eigenvalues $a_1 < a_2 < \cdots < a_n$ and unit eigenvectors $\mathbf{q}_1, \dots, \mathbf{q}_n$, and minimize $f(\mathbf{x}) = \mathbf{x}^\top \mathbf{A} \mathbf{x}$ subject to $\mathbf{x}^\top \mathbf{x} = 1$. Show that the Lagrange points are $\pm \mathbf{q}_i$ with $\nu^\star = -a_i$, and find the eigenvalues of $\nabla_{\mathbf{x}}^2 \mathcal{L}$ on the tangent space at $\mathbf{q}_i$. Which Lagrange points are local minima, which are local maxima, and which are saddles? How many independent descent directions does $\mathbf{q}_i$ have?

   3. Projected gradient descent onto the sphere, $\mathbf{x} \leftarrow (\mathbf{x} - \eta \nabla f(\mathbf{x}))/\|\mathbf{x} - \eta \nabla f(\mathbf{x})\|$, projects onto a nonconvex set. Show that for this f it is the power iteration for $\mathbf{I} - 2\eta \mathbf{A}$. Assume $a_1 + a_n > 0$. From a random start, what do the iterates $\mathbf{x}_k$ and the values $f(\mathbf{x}_k)$ do? How does the answer depend on $\eta > 0$?

6. code **Projection onto an ellipse.** Clip $\mathbf{y} = (3, 4)$ to the ellipse $\{\mathbf{x} : x_1^2/4 + x_2^2 \leq 1\}$, with semi-axes $a_1 = 2$ and $a_2 = 1$, written as $g(\mathbf{x}) = \frac{1}{2}(x_1^2/4 + x_2^2 - 1) \leq 0$.

1. Show from stationarity that $x_i^\star = y_i/(1+\lambda/a_i^2)$. Each coordinate is rescaled by its own factor, so the projection is not on the ray through $\mathbf{y}$.
2. Show that $\lambda \mapsto g(\mathbf{x}(\lambda))$ is strictly decreasing on $\lambda > 0$, tends to $-\frac{1}{2}$ as $\lambda \to \infty$, and is positive at $\lambda = 0$. Solve $g(\mathbf{x}(\lambda)) = 0$ by bisection; you should find $\lambda^\star \approx 4.59$ and $\mathbf{x}^\star \approx (1.40, 0.72)$. Compare with the point of the ellipse on the ray through $\mathbf{y}$, about $(0.70, 0.94)$, and check that it is farther from $\mathbf{y}$.
3. Explain why the ball, the simplex, and the ellipse projections all reduce to one scalar equation in a multiplier, and why the projection onto a general polytope does not.

7. **Block soft-thresholding (the group lasso).** The penalty $\lambda\|\mathbf{w}\|_2$ on a whole vector is kinked only at $\mathbf{w} = \mathbf{0}$. Its subgradients are $\mathbf{w}/\|\mathbf{w}\|$ for $\mathbf{w} \neq \mathbf{0}$ and every vector of the unit ball at $\mathbf{0}$ (times λ).
   1. Show that $\operatorname{prox}_{\lambda\|\cdot\|_2}(\mathbf{z}) = \max(0,\ 1-\lambda/\|\mathbf{z}\|)\,\mathbf{z}$.
   2. Compare the zero pattern with that of coordinatewise soft-thresholding Equation 35.71. When does the block prox return exactly $\mathbf{0}$, and can it zero a single coordinate of a block?
   3. What does the penalty $\lambda\sum_u \|\mathbf{w}_u\|_2$, summed over the vectors $\mathbf{w}_u$ of outgoing weights of the hidden units u of a network, do to the network?
   4. code Verify (a) against a brute-force minimization on a 2-D grid for $\mathbf{z} = (0.3, 0.4)$ and $\mathbf{z} = (3, 4)$ with $\lambda = 1$. The answers are $\mathbf{0}$ and $(2.4, 3.2)$; where has $(2.4, 3.2)$ appeared in this section?

8. **Moreau decomposition.** ▯ The *convex conjugate* of ψ is $\psi^*(\mathbf{y}) = \sup_{\mathbf{x}} (\mathbf{y}^\top\mathbf{x} - \psi(\mathbf{x}))$. For a proper closed convex ψ, two facts connect it with the proximal operator Equation 35.72: $\mathbf{x} = \operatorname{prox}_{\psi}(\mathbf{z})$ exactly when $\mathbf{z}-\mathbf{x} \in \partial\psi(\mathbf{x})$ (the set of subgradients), and $\mathbf{y} \in \partial\psi(\mathbf{x})$ exactly when $\mathbf{x} \in \partial\psi^*(\mathbf{y})$ (Parikh and Boyd 2014).
   1. Derive the Moreau decomposition $\mathbf{z} = \operatorname{prox}_{\psi}(\mathbf{z}) + \operatorname{prox}_{\psi^*}(\mathbf{z})$.
   2. The conjugate of $\psi = \lambda\|\cdot\|_1$ is the indicator of the box $\{\mathbf{y} : \|\mathbf{y}\|_\infty \le \lambda\}$. Show that the decomposition writes soft-thresholding Equation 35.71 as $\mathbf{z}$ minus a clipped copy of $\mathbf{z}$.
   3. The conjugate of $\psi = \lambda\|\cdot\|_2$ is the indicator of the ℓ_2 ball of radius λ, whose prox is the projection onto the ball, the clip-by-norm map of gradient clipping. Use the decomposition to recover the block soft-thresholding formula of the group-lasso exercise from the clipping formula. For $\mathbf{z} = (3, 4)$ and $\lambda = 1$, which two vectors of this section add up to $\mathbf{z}$?

9. code **Accelerated proximal gradient.** ▯ FISTA (Beck and Teboulle 2009) adds momentum to the ISTA update Equation 35.73 for $F = s + \psi$. With $\mathbf{v}_1 = \mathbf{x}_0$ and $t_1 = 1$, it iterates for $k \ge 1$

$$\mathbf{x}_k = \operatorname{prox}_{\eta\psi}(\mathbf{v}_k - \eta\nabla s(\mathbf{v}_k)), \qquad t_{k+1} = \tfrac{1}{2}\Big(1+\sqrt{1+4t_k^2}\Big), \qquad \mathbf{v}_{k+1} = \mathbf{x}_k + \frac{t_k-1}{t_{k+1}}(\mathbf{x}_k - \mathbf{x} \tag{35.78}$$

   With $\eta = 1/L$ its objective gap satisfies $F(\mathbf{x}_k) - F^\star \le 2L\|\mathbf{x}_0 - \mathbf{x}^\star\|^2/(k+1)^2$, against $O(1/k)$ for ISTA.
   1. Show that the prox is nonexpansive, $\|\operatorname{prox}_{\eta\psi}(\mathbf{u}) - \operatorname{prox}_{\eta\psi}(\mathbf{v})\| \le \|\mathbf{u}-\mathbf{v}\|$,

by adding the two subgradient inequalities that characterize $\operatorname{prox}_{\eta\psi}(\mathbf{u})$ and $\operatorname{prox}_{\eta\psi}(\mathbf{v})$, as in the proof for projections.

2. Add FISTA to the ISTA cell, on the same lasso, and plot $F(\mathbf{x}_k) - F^\star$ of both methods over 100 steps on a logarithmic axis, with $F^\star$ taken from 10^4 ISTA steps. Which method is ahead after 10 steps, and which after 30? Explain the straight line of ISTA with the PGD contraction proposition and part (a).
3. Is $F(\mathbf{x}_k)$ monotone along the FISTA iterates?
4. Replace $\mathbf{A}$ by a 30×100 Gaussian matrix, plant the same three nonzero coefficients, and run both methods for 1000 steps. Over which range of k is FISTA ahead now? Why does ISTA still end with a fast linear phase, although $\mathbf{A}^\top\mathbf{A}$ is singular?

10. code **Stalls of coordinate descent and separable penalties.** ☒

1. Let $f(x, y) = \max(|x|, |y|)$. Show that at $(1, 1)$ no change of x alone and no change of y alone lowers f, although the minimum 0 lies at the origin. Conclude that exact coordinate descent started at $(1, 1)$ never moves.
2. Let $F(\mathbf{w}) = s(\mathbf{w}) + \sum_j \psi_j(w_j)$ with s convex and differentiable and each ψ_j convex. Show that a point at which F is minimal along every coordinate separately is a global minimizer. Hint: minimality along coordinate j means $0 \in \partial_j s(\mathbf{w}) + \partial\psi_j(w_j)$, and the set of subgradients of a separable sum is the product of the one-dimensional sets. Why does (a) not contradict this?
3. On the quadratic $\frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x} - \mathbf{1}^\top\mathbf{x}$ with the 40×40 tridiagonal matrix $\mathbf{A} = 2\mathbf{I} - 0.95(\mathbf{E}_{+1} + \mathbf{E}_{-1})$, where $\mathbf{E}_{\pm 1}$ have ones on the first super- and subdiagonal, a cyclic sweep of exact coordinate steps Equation 35.77 is the Gauss–Seidel iteration. Theory predicts that its error shrinks per sweep by the square of the Jacobi factor $0.95\cos(\pi/41) \approx 0.947$, that is by about 0.897, while gradient descent with $\eta = 1/L$ shrinks it by $1 - \mu/L \approx 0.973$ per step, with $\mu \approx 0.106$ and $L \approx 3.89$ the extreme eigenvalues of $\mathbf{A}$. Run both for 100 sweeps or steps and measure the late factors.

35.7 Duality

An iterative solver stops when its progress becomes small and reports that it has converged. Whether the point it returns is optimal is a separate question. Without constraints, a zero gradient answers it for a convex function (Section 35.3). A constrained optimum usually has a nonzero gradient (Section 35.6), so that test is gone. Solvers need a lower bound on the optimal value to decide when to stop, and proofs need one to show that an answer is optimal. This section asks how to find such a bound, and what the multipliers of the KKT conditions measure.

The clipping problem of Section 35.6.2 shows the difficulty. It minimized the squared distance $\frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2$ to the point $(3, 4)$ over all $\mathbf{x}$ with $\|\mathbf{x}\| \le 1$ and found $\mathbf{x}^\star = (0.6, 0.8)$ with value 8. (That section called the point $\mathbf{y}$. Here it is $\mathbf{x}_0$, because $\mathbf{y}$ will be needed for labels and for the argument of a conjugate.) Suppose instead that an iterative solver stops at $\mathbf{x} = (0.55, 0.835)$. This point is feasible ($\|\mathbf{x}\|^2 = 0.9997$) and has value 8.01. Is that good enough, or is there a feasible point with value 7? The gradient at the true optimum is $(-2.4, -3.2)$, not zero, and the disk has infinitely many points, so we cannot check them all.

The bound comes from moving the constraint into the objective. Add λ times the constraint function $\frac{1}{2}(\|\mathbf{x}\|^2 - 1)$, with $\lambda \geq 0$. At every feasible point the added term is at most zero, so the minimum of the new objective over *all* of $\mathbb{R}^2$ lies below the constrained optimum. Each λ gives one lower bound, computed by an unconstrained minimization, and $\lambda = 4$ gives exactly 8: the solver's 8.01 is within 0.01 of optimal. Choosing the best λ is a second optimization problem, the dual problem. The dual problem is always convex. For a convex problem with a strictly feasible point, its value equals the constrained optimum. Its solution, the vector of multipliers, measures how fast the optimum improves when each constraint is relaxed.

We prove that no feasible point of the clipping problem beats $\mathbf{x}^\star$, compute duals by hand for a half-plane, the support vector machine, and a power allocation problem, and show a problem whose best bound stays strictly below its optimum. Three results carry the section: weak duality, which costs two lines; strong duality for convex problems, whose proof holds the one hard step; and the reading of the multipliers as prices. The conjugate, a tool for computing duals, comes after the prices and can wait for a second reading. All computations are small enough for NumPy, and `d21` supplies the plots.

35.7.1 The Dual Problem

We want to prove that no feasible point of the clipping problem has a value below 8, without visiting feasible points. Every bound in this section comes from one fact: at a feasible point, every inequality constraint function is at most zero.

35.7.1.1 A Lower Bound from One Multiplier

The clipping problem minimizes $f(\mathbf{x}) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2$ with $\mathbf{x}_0 = (3, 4)$, subject to $g(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\|^2 - 1) \leq 0$. The factor $\frac{1}{2}$ makes the constraint's gradient equal to $\mathbf{x}$. The KKT conditions gave the answer $\mathbf{x}^\star = \mathbf{x}_0/\|\mathbf{x}_0\| = (0.6, 0.8)$ with value $p^\star = \frac{1}{2}(5-1)^2 = 8$, and the multiplier $\lambda^\star = 4$ from the stationarity condition $\nabla f(\mathbf{x}^\star) + \lambda^\star \mathbf{x}^\star = (-2.4, -3.2) + 4\,(0.6, 0.8) = \mathbf{0}$.

Take any $\lambda \geq 0$ and any feasible $\mathbf{x}$. Since $g(\mathbf{x}) \leq 0$, the product $\lambda g(\mathbf{x})$ is at most zero, and $f(\mathbf{x}) + \lambda g(\mathbf{x}) \leq f(\mathbf{x})$. The minimum of the left side over all of $\mathbb{R}^2$ is at most its value at this feasible $\mathbf{x}$, and hence at most $f(\mathbf{x})$. Taking the best feasible $\mathbf{x}$ on the right gives

$$\min_{\mathbf{x}\in\mathbb{R}^2}\big[f(\mathbf{x}) + \lambda g(\mathbf{x})\big] \;\leq\; \min_{\|\mathbf{x}\|\leq 1} f(\mathbf{x}) \;=\; p^\star \qquad \text{for every } \lambda \geq 0.$$

We compute the left side. The function being minimized is the Lagrangian $\mathcal{L}(\mathbf{x}, \lambda) = \frac{1}{2}\|\mathbf{x} - \mathbf{x}_0\|^2 + \frac{\lambda}{2}(\|\mathbf{x}\|^2 - 1)$. For $\lambda \geq 0$ it is a convex quadratic in $\mathbf{x}$ with Hessian $(1+\lambda)\mathbf{I}$, so its minimizer is the point where the gradient $(\mathbf{x} - \mathbf{x}_0) + \lambda\mathbf{x}$ vanishes: $\mathbf{x}(\lambda) = \mathbf{x}_0/(1+\lambda)$. The minimizer shrinks $\mathbf{x}_0$ toward the origin, more strongly for larger λ. To substitute it, we use $\mathbf{x}(\lambda) - \mathbf{x}_0 = -\lambda\mathbf{x}_0/(1+\lambda)$ and $\|\mathbf{x}_0\|^2 = 25$, so $\frac{1}{2}\|\mathbf{x}(\lambda) - \mathbf{x}_0\|^2 = \frac{25}{2}\lambda^2/(1+\lambda)^2$ and $\|\mathbf{x}(\lambda)\|^2 = 25/(1+\lambda)^2$. The two terms with the factor $25/(1+\lambda)^2$ add up to $\frac{25}{2}(\lambda^2 + \lambda)/(1+\lambda)^2$, and the minimum value is

$$q(\lambda) \;=\; \frac{25}{2}\,\frac{\lambda}{1+\lambda} \;-\; \frac{\lambda}{2}. \tag{35.79}$$

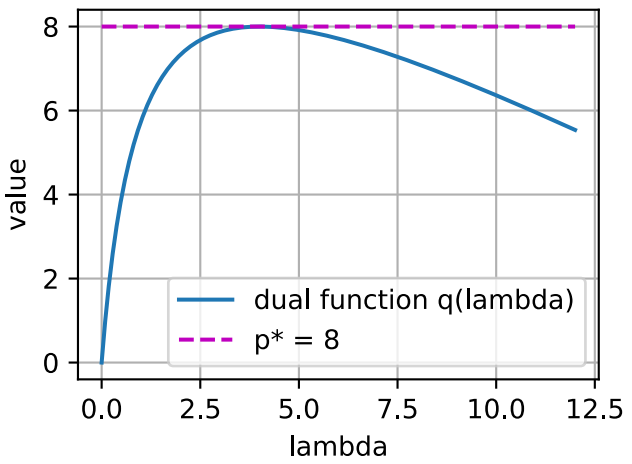
The first term rises toward 12.5 as λ grows. The second term, λ times the constant $-\frac{1}{2}$ in the constraint function, falls without bound. The best bound balances the two.

A few values make this concrete. The function gives $q(0) = 0$, $q(1) = 5.75$, $q(2) = 7.33$, $q(3) = 7.875$, $q(4) = 8$, $q(5) = 7.92$ and $q(8) = 7.11$. Every one of them is a valid lower bound on $p^\star$. Its derivative $q'(\lambda) = \frac{25}{2(1+\lambda)^2} - \frac{1}{2}$ vanishes where $(1+\lambda)^2 = 25$, at $\lambda = 4$, and $q''(\lambda) = -25/(1+\lambda)^3 < 0$, so $\lambda = 4$ is the maximum, with value $q(4) = 10 - 2 = 8 = p^\star$. The best bound equals the optimum, and the λ that attains it is the KKT multiplier. The solver's 8.01 is therefore within 0.01 of optimal. Even $\lambda = 3$, picked without solving anything, proves that it is within 0.135.

Why is $\lambda = 4$ the right value? The constraint value at the shrunken point is $g(\mathbf{x}(\lambda)) = \frac{1}{2}(25/(1+\lambda)^2 - 1)$. For $\lambda < 4$ it is positive: the multiplier is too small to keep the minimizer in the disk ($\mathbf{x}(1) = (1.5, 2)$ has norm 2.5), and the bound is the value of an infeasible point. For $\lambda > 4$ it is negative: the minimizer is pushed deep inside ($\mathbf{x}(8)$ has norm $\frac{5}{9}$), and the term $\lambda g(\mathbf{x}(\lambda)) < 0$ lowers the bound. At $\lambda = 4$ the minimizer $\mathbf{x}(4) = (0.6, 0.8)$ lies on the circle, the added term vanishes, and the bound is the value of a feasible point.

The cell evaluates $f + \lambda g$ at $\mathbf{x}(\lambda)$ on a grid of λ and plots the bounds against $p^\star$; the shape of the curve is what to look at.

```
x0 = np.array([3.0, 4.0])                          # the point to clip
f = lambda x: 0.5 * np.sum((x - x0) ** 2)          # objective
g = lambda x: 0.5 * (np.sum(x ** 2) - 1.0)         # constraint g(x) <= 0
x_of = lambda lam: x0 / (1 + lam)                  # minimizer of L(., lam)
q = lambda lam: f(x_of(lam)) + lam * g(x_of(lam))  # dual function
p_star = f(x0 / np.linalg.norm(x0))                # value at the clipped point
lams = np.linspace(0.0, 12.0, 1201)
qs = np.array([q(l) for l in lams])
d2l.plot(lams, [qs, np.full_like(lams, p_star)], 'lambda', 'value',
         legend=['dual function q(lambda)', 'p* = 8'])
```



The curve is concave and stays below the line $p^\star = 8$, touching it once, at $\lambda = 4$. Concavity is not a property of the disk: the next subsection proves it for every problem.

35.7.1.2 The Lagrange Dual Function

Nothing in this argument used the shape of the disk. The general problem of Section 35.6.2 minimizes f subject to inequality and equality constraints,

$$\min_{\mathbf{x}\in\mathcal{D}} \; f(\mathbf{x}) \quad \text{subject to} \quad g_i(\mathbf{x}) \le 0,\; i = 1, \dots, m, \qquad h_j(\mathbf{x}) = 0,\; j = 1, \dots, p,$$

with optimal value $p^\star$, the infimum of f over the feasible points ($p^\star = +\infty$ if there are none). The set $\mathcal{D}$ is the domain of the problem, where f and all constraint functions are defined: it is $\mathbb{R}^n$ unless a problem keeps a set as a hard constraint, as the gap example at the end of this section does with $[0, 1]$. The Lagrangian adds one multiplier per constraint,

$$\mathcal{L}(\mathbf{x}, \lambda, \nu) = f(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i \, g_i(\mathbf{x}) + \sum_{j=1}^{p} \nu_j \, h_j(\mathbf{x}),$$

with $\lambda_i \ge 0$ and ν_j of either sign. We write $\lambda \succeq 0$ for "every $\lambda_i \ge 0$", the elementwise sense of $\succeq$ on vectors.

The bound of the clipping problem becomes a function of the multipliers. The **Lagrange dual function** is the minimum of the Lagrangian over the domain, for fixed multipliers,

$$q(\lambda, \nu) = \inf_{\mathbf{x}\in\mathcal{D}} \; \mathcal{L}(\mathbf{x}, \lambda, \nu). \tag{35.80}$$

In words, q asks how low f can go once the constraints are replaced by penalties with the given weights. For the clipping problem it is Equation 35.79. The infimum need not be attained or even finite. Minimize x subject to $-x \le 0$: the Lagrangian $(1-\lambda)x$ is unbounded below unless $\lambda = 1$, so $q(1) = 0$ and $q(\lambda) = -\infty$ for every other λ. A value $-\infty$ is a true but useless bound. The set where $q > -\infty$ is the *effective domain* of q; in the SVM dual below, this restriction turns into an equality constraint.

The dual function has two properties that need no assumption on f, g_i, or h_j.

Proposition (the dual function is concave). *For any primal problem, $q(\lambda, \nu)$ in Equation 35.80 is concave.*

Proof. For each fixed $\mathbf{x}$, the map $(\lambda, \nu) \mapsto \mathcal{L}(\mathbf{x}, \lambda, \nu)$ is affine, because the multipliers enter linearly. The dual function is the pointwise infimum of this family of affine functions. Its **hypograph**, the set of points on or below its graph, is the intersection of the half-spaces below the affine functions, and an intersection of convex sets is convex. So q is concave. ■

Proposition (weak duality). *If $\mathbf{x}$ is feasible and $\lambda \succeq 0$, then $q(\lambda, \nu) \le f(\mathbf{x})$. Consequently*

$$d^\star \;=\; \sup_{\lambda\succeq 0,\,\nu} q(\lambda, \nu) \;\le\; \inf_{\mathbf{x}\text{ feasible}} f(\mathbf{x}) \;=\; p^\star. \tag{35.81}$$

Proof. For feasible $\mathbf{x}$, each $\lambda_i g_i(\mathbf{x}) \le 0$ (nonnegative times nonpositive) and each $\nu_j h_j(\mathbf{x}) = 0$, so

$$q(\lambda, \nu) \;\le\; \mathcal{L}(\mathbf{x}, \lambda, \nu) \;=\; f(\mathbf{x}) + \sum_i \lambda_i g_i(\mathbf{x}) + \sum_j \nu_j h_j(\mathbf{x}) \;\le\; f(\mathbf{x}).$$

Take the supremum over (λ, ν) on the left and the infimum over feasible $\mathbf{x}$ on the right. ■

This is the inequality that made every value $q(\lambda)$ of the clipping problem a bound. It was also the proof of KKT sufficiency in Section 35.6.2, which ran the same chain at the KKT multipliers, where complementary slackness makes the bound tight. The **dual problem** maximizes q over $\lambda \succeq 0$ and ν. It is always a convex problem, a concave function maximized over a convex set, even when the primal problem is not convex. Its optimal value $d^\star$ is a lower bound on $p^\star$, and every dual feasible point gives one. The difference $p^\star - d^\star \geq 0$ is the **duality gap**. It was zero for the clipping problem, and the last worked example of this section shows a problem where it is not. When must it be zero? That is the next question.

35.7.2 Strong Duality

For the clipping problem the best bound equalled the optimum. For a nonconvex problem it can fall short, and then the dual certifies only part of the answer. Convexity and one strictly feasible point guarantee equality, and a picture of the attainable pairs of constraint and objective values shows why.

35.7.2.1 Slater's Condition

Proposition (Slater's condition implies strong duality). *Suppose the problem is convex (f and every g_i convex, every h_j affine) and satisfies Slater's condition Equation 35.66: some feasible point $\bar{\mathbf{x}}$ in the relative interior of $\mathcal{D}$ has $g_i(\bar{\mathbf{x}}) < 0$ for every g_i that is not affine. Then $d^\star = p^\star$, and if $p^\star$ is finite the dual optimum is attained.*

This is the constraint qualification that Section 35.6.2 introduced for convex problems: one point strictly inside all the curved constraints. For the general proof see Boyd and Vandenberghe (2004), section 5.3.2. The clipping problem satisfies it: f and g are convex, and $\mathbf{0}$ has $g(\mathbf{0}) = -\frac{1}{2} < 0$. So $q(4) = p^\star$ was guaranteed before we computed anything.

Now the picture. Take a single inequality constraint g and map each $\mathbf{x} \in \mathcal{D}$ to the pair $(g(\mathbf{x}), f(\mathbf{x}))$ of constraint value and objective value. Collect these pairs and every point above and to the right of them,

$$\mathcal{A} = \big\{(u,t) : u \geq g(\mathbf{x}),\ t \geq f(\mathbf{x}) \text{ for some } \mathbf{x} \in \mathcal{D}\big\}.$$

The optimum $p^\star$ is the lowest t that $\mathcal{A}$ reaches on the line $u = 0$. For $\lambda \geq 0$, the smallest value of $t + \lambda u$ over the points that one $\mathbf{x}$ contributes is $f(\mathbf{x}) + \lambda g(\mathbf{x})$, so $q(\lambda) = \inf_{(u,t)\in\mathcal{A}}(t + \lambda u)$. In other words, the line $t + \lambda u = q(\lambda)$, of slope $-\lambda$, is lowered until it touches $\mathcal{A}$ from below, and $q(\lambda)$ is its height at $u = 0$. Since the line stays below $\mathcal{A}$, its height at $u = 0$ is at most $p^\star$. This is weak duality drawn as a picture (Figure 35.22).

Proof (one inequality constraint with $g(\bar{\mathbf{x}}) < 0$, $p^\star$ finite). First, $\mathcal{A}$ is convex. Take (u_1, t_1) and (u_2, t_2) in $\mathcal{A}$, contributed by $\mathbf{x}_1$ and $\mathbf{x}_2$, and $\theta \in [0,1]$. By convexity of g, the point $\mathbf{x}_\theta = \theta\mathbf{x}_1 + (1-\theta)\mathbf{x}_2$ has $g(\mathbf{x}_\theta) \leq \theta g(\mathbf{x}_1) + (1-\theta)g(\mathbf{x}_2) \leq \theta u_1 + (1-\theta)u_2$, and the same holds for f, so the combination of the two points lies in $\mathcal{A}$. Second, $(0, p^\star)$ is a boundary point of $\mathcal{A}$: it is a limit of points of $\mathcal{A}$, while $(0, p^\star - \varepsilon)$ is not in $\mathcal{A}$ for any $\varepsilon > 0$. A convex set has a supporting line at each boundary point: some $(\lambda, \gamma) \neq (0,0)$ satisfies $\gamma t + \lambda u \geq \gamma p^\star$ for all $(u,t) \in \mathcal{A}$ (Boyd and Vandenberghe (2004), section 2.5.2; it is the fact behind the subgradients of Section 35.3). Because $\mathcal{A}$ extends upward and to the right without limit, $\gamma \geq 0$ and $\lambda \geq 0$. If $\gamma = 0$, the line is vertical, $\lambda > 0$, and $\lambda u \geq 0$ forces $u \geq 0$ on all of $\mathcal{A}$;

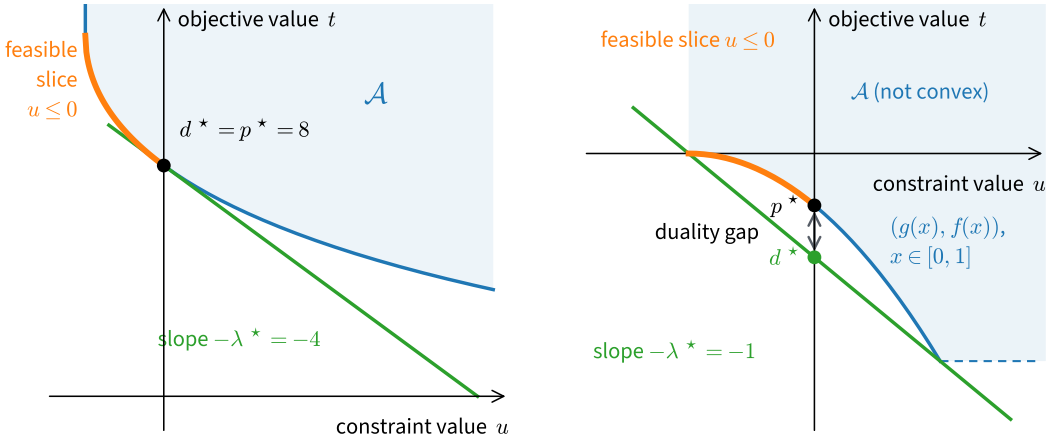


Figure 35.22: The supporting-line geometry of duality. Every $\mathbf{x}$ maps to $(g(\mathbf{x}), f(\mathbf{x}))$; the shaded set $\mathcal{A}$ holds these pairs and everything above and to the right of them. Evaluating the dual at λ lowers a line of slope $-\lambda$ until it supports $\mathcal{A}$ from below, and its height at $u = 0$ is $q(\lambda)$. (a) The clipping problem: $\mathcal{A}$ is convex and $\mathbf{0}$ is strictly feasible, so the best line touches at $(0, p^\star)$ with slope -4: $d^\star = p^\star = 8$. (b) Minimizing $-x^2$ on $[0, 1]$ subject to $x \le \frac{1}{2}$: $\mathcal{A}$ is not convex, the best line passes under the point $(0, p^\star)$, and $d^\star = -\frac{1}{2} < -\frac{1}{4} = p^\star$.

the strictly feasible point contributes $u = g(\bar{\mathbf{x}}) < 0$, a contradiction. So $\gamma > 0$, and dividing by γ gives $t + (\lambda/\gamma)\, u \ge p^\star$ on $\mathcal{A}$, that is, $q(\lambda/\gamma) \ge p^\star$. With weak duality, $q(\lambda/\gamma) = p^\star$: the dual optimum is attained and equals $p^\star$. ■

The proof used convexity once, to make $\mathcal{A}$ convex, and the strictly feasible point once, to rule out a vertical line, which no finite multiplier can represent. Panel (a) of the figure is the clipping problem: at $\|\mathbf{x}\| = r$ the smallest objective is $\frac{1}{2}(5 - r)^2$ and the constraint value is $\frac{1}{2}(r^2 - 1)$, so the lower edge of $\mathcal{A}$ is $t = \frac{1}{2}\left(5 - \sqrt{1 + 2u}\right)^2$. Its slope at $u = 0$ is -4, which is why the best line has slope $-\lambda^\star = -4$. Panel (b) is the nonconvex example worked at the end of this section: its set $\mathcal{A}$ is not convex, and every line below it passes under $(0, p^\star)$.

35.7.2.2 Saddle Points

Strong duality has an equivalent form in terms of the Lagrangian alone. The primal problem can be written as a min–max problem: for any $\mathbf{x}$,

$$\sup_{\lambda \succeq 0,\, \nu} \mathcal{L}(\mathbf{x}, \lambda, \nu) = \begin{cases} f(\mathbf{x}) & \text{if } \mathbf{x} \text{ is feasible,} \\ +\infty & \text{otherwise.} \end{cases}$$

A violated constraint lets its multiplier drive the supremum to $+\infty$, while for feasible $\mathbf{x}$ the best the multipliers can do is vanish on every slack constraint. So $p^\star = \inf_{\mathbf{x}} \sup_{\lambda \succeq 0, \nu} \mathcal{L}$, and by definition $d^\star = \sup_{\lambda \succeq 0, \nu} \inf_{\mathbf{x}} \mathcal{L}$. Weak duality is the inequality $\sup \inf \le \inf \sup$, which holds for any function of two variables, and strong duality says that the order of the two optimizations does not matter. When both optima are attained, this equality of the two orders is the same as a saddle point of the Lagrangian.

Proposition (strong duality is a saddle point). *Suppose strong duality holds with both optima attained, at a primal optimum* $\mathbf{x}^\star$ *and a dual optimum* $(\lambda^\star, \nu^\star)$*. Then* $(\mathbf{x}^\star, \lambda^\star, \nu^\star)$ *is a* **saddle point** *of the Lagrangian: for all* $\mathbf{x} \in \mathcal{D}$ *and all* $\lambda \succeq 0$*,* ν*,*

$$\mathcal{L}(\mathbf{x}^\star, \lambda, \nu) \;\le\; \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) \;\le\; \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star). \tag{35.82}$$

Conversely, any saddle point of $\mathcal{L}$ *over* $\mathbf{x} \in \mathcal{D}$ *and* $(\lambda \succeq 0, \nu)$ *closes the duality gap, with saddle value* $p^\star = d^\star$*.*

Proof. Using the definitions and weak duality,

$$d^\star = q(\lambda^\star, \nu^\star) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star) \;\le\; \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) \;\le\; f(\mathbf{x}^\star) = p^\star,$$

where the last inequality holds because $\lambda_i^\star \ge 0$ and $g_i(\mathbf{x}^\star) \le 0$ make every added term nonpositive. Strong duality turns every inequality in this chain into an equality. Equality in the first says that $\mathbf{x}^\star$ minimizes $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$, the right half of Equation 35.82. Equality in the second forces $\sum_i \lambda_i^\star g_i(\mathbf{x}^\star) = 0$, which is complementary slackness. Then for any $\lambda \succeq 0$ and ν, $\mathcal{L}(\mathbf{x}^\star, \lambda, \nu) = f(\mathbf{x}^\star) + \sum_i \lambda_i g_i(\mathbf{x}^\star) \le f(\mathbf{x}^\star) = \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star)$, the left half. For the converse, a saddle point gives

$$\inf_{\mathbf{x}} \sup_{\lambda \succeq 0,\, \nu} \mathcal{L} \;\le\; \sup_{\lambda \succeq 0,\, \nu} \mathcal{L}(\mathbf{x}^\star, \lambda, \nu) = \mathcal{L}(\mathbf{x}^\star, \lambda^\star, \nu^\star) = \inf_{\mathbf{x}} \mathcal{L}(\mathbf{x}, \lambda^\star, \nu^\star) \;\le\; \sup_{\lambda \succeq 0,\, \nu} \inf_{\mathbf{x}} \mathcal{L},$$

that is, $p^\star \le d^\star$; weak duality supplies the reverse inequality. ■

In the clipping problem the saddle point is $(\mathbf{x}^\star, \lambda^\star) = ((0.6, 0.8), 4)$: moving $\mathbf{x}$ away from it raises $\mathcal{L}(\cdot, 4)$, and changing λ leaves $\mathcal{L}(\mathbf{x}^\star, \lambda) = 8$ unchanged, because $g(\mathbf{x}^\star) = 0$.

Strong duality has three practical uses.

- *The dual can be the smaller problem.* The SVM dual has one variable per data point, sign constraints, and at most one equality; the ridge dual of Exercise 9 is a linear system of the size of the data set rather than of the feature space.
- *Every dual point is a stopping test.* For any feasible $\mathbf{x}$ and any $\lambda \succeq 0$, ν, the difference $f(\mathbf{x}) - q(\lambda, \nu)$ bounds how far $f(\mathbf{x})$ is above $p^\star$.
- *The dual solution locates the primal solution.* The optimum $\mathbf{x}^\star$ is *among* the minimizers of $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$. When that minimizer is unique, for example when f is strictly convex, it is $\mathbf{x}^\star$: in the clipping problem, $\mathbf{x}(4) = \mathbf{x}_0/5 = \mathbf{x}^\star$. Otherwise the minimizers can include infeasible points, and feasibility and complementary slackness select $\mathbf{x}^\star$ among them. In a linear program, $\mathcal{L}(\cdot, \lambda^\star)$ is constant in $\mathbf{x}$, so every point minimizes it; in the SVM with an offset below, the Lagrangian does not determine the offset, and complementary slackness does.

The proposition also proves the necessity of the KKT conditions, which Section 35.6.2 showed sufficient for convex problems, leaving necessity to a constraint qualification. Suppose a convex problem with differentiable f, g_i, h_j on $\mathcal{D} = \mathbb{R}^n$ satisfies Slater's condition and attains its optimum at $\mathbf{x}^\star$. Then strong duality holds with an attained dual optimum, $\mathbf{x}^\star$ minimizes $\mathcal{L}(\cdot, \lambda^\star, \nu^\star)$, so $\nabla_{\mathbf{x}} \mathcal{L} = \mathbf{0}$ there, and complementary slackness holds. Together with primal and dual feasibility these are the four KKT conditions: under Slater's condition they are necessary as well as sufficient.

The same min–max form describes training objectives that are not constrained problems at all. Generative adversarial networks (Section 16.1) and adversarial training are $\min_\theta \max_\phi$ problems (Goodfellow et al. 2014; Madry et al. 2018), and without a saddle point a gap can separate their min max and max min values. Even when a saddle point exists, taking gradient descent steps in the minimizing variable and ascent steps in the maximizing one at the same time need not reach it: for the bilinear objective xy, whose saddle point is the origin, the iterates spiral outward for every step size (Exercise 6).

35.7.3 Multipliers as Prices

What is a constraint worth? In the clipping problem, if the disk were a little larger, how much closer to $\mathbf{x}_0$ could we get? The optimal multiplier answers this question. The same answer shows that, for a convex loss, weight decay and a norm constraint on the weights produce the same solutions.

35.7.3.1 Shadow Prices

Relax each constraint by a given amount and record the optimal value as a function of the constraint levels: for $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{v} \in \mathbb{R}^p$ let

$$p^\star(\mathbf{u}, \mathbf{v}) = \inf\{f(\mathbf{x}) : g_i(\mathbf{x}) \le u_i,\ h_j(\mathbf{x}) = v_j\},$$

so that $p^\star(\mathbf{0}, \mathbf{0})$ is the original optimum and $u_i > 0$ loosens inequality i. We have drawn this function already. For one inequality constraint, the lowest t that the set $\mathcal{A}$ of Figure 35.22 reaches at level u is $p^\star(u)$: the lower edge of $\mathcal{A}$ is its graph, and the supporting line of slope $-\lambda^\star$ lies below it. The proposition says this for any number of constraints.

Proposition (multipliers are shadow prices). *Suppose strong duality holds with dual optimum $(\lambda^\star, \nu^\star)$. Then for all* $\mathbf{u}$ *and* $\mathbf{v}$*,*

$$p^\star(\mathbf{u}, \mathbf{v}) \ \ge\ p^\star(\mathbf{0}, \mathbf{0}) - \lambda^{\star\top}\mathbf{u} - \nu^{\star\top}\mathbf{v}, \tag{35.83}$$

and if $p^\star$ is differentiable at $(\mathbf{0}, \mathbf{0})$, then $\nabla_{\mathbf{u}} p^\star = -\lambda^\star$ and $\nabla_{\mathbf{v}} p^\star = -\nu^\star$ there.

Proof. Let $\mathbf{x}$ be feasible for the perturbed problem: $g_i(\mathbf{x}) \le u_i$ and $h_j(\mathbf{x}) = v_j$. Then

$$p^\star(\mathbf{0}, \mathbf{0}) = q(\lambda^\star, \nu^\star) \ \le\ f(\mathbf{x}) + \sum_i \lambda_i^\star g_i(\mathbf{x}) + \sum_j \nu_j^\star h_j(\mathbf{x}) \ \le\ f(\mathbf{x}) + \lambda^{\star\top}\mathbf{u} + \nu^{\star\top}\mathbf{v},$$

using strong duality, the definition of q as an infimum, $\lambda_i^\star g_i(\mathbf{x}) \le \lambda_i^\star u_i$, and $h_j(\mathbf{x}) = v_j$. Taking the infimum over all such $\mathbf{x}$ gives Equation 35.83. If $p^\star$ is differentiable at the origin, the function $p^\star(\mathbf{u}, \mathbf{v}) - p^\star(\mathbf{0}, \mathbf{0}) + \lambda^{\star\top}\mathbf{u} + \nu^{\star\top}\mathbf{v}$ is nonnegative and vanishes at the origin, so the origin is its minimum and its gradient is zero there. ■

The multiplier $\lambda_i^\star$ is the **shadow price** of constraint i: the rate at which the optimal value falls as the constraint is relaxed. The inequality holds for perturbations of any size. Relaxing constraint i by $u_i > 0$ improves the optimum by at most $\lambda_i^\star u_i$, and tightening it by the same amount makes the optimum worse by at least $\lambda_i^\star u_i$; to first order, when $p^\star$ is differentiable,

the change is exactly $\lambda_i^\star u_i$. A slack constraint has $\lambda_i^\star = 0$ by complementary slackness: relaxing a constraint that does not bind does not help.

For the clipping problem, relaxing $g(\mathbf{x}) \le 0$ to $g(\mathbf{x}) \le u$ enlarges the disk to radius $r = \sqrt{1+2u}$, and $p^\star(u) = \frac{1}{2}(5 - \sqrt{1+2u})^2$ is the edge computed after the proof of strong duality, with slope $-4 = -\lambda^\star$ at $u = 0$. Relaxing by $u = \frac{1}{2}$, a disk of radius $\sqrt{2}$, gives $p^\star = \frac{1}{2}(5-\sqrt{2})^2 = 6.43$, above the bound $8 - 4 \cdot \frac{1}{2} = 6$ that the proposition guarantees.

The multiplier depends on how the constraint is written. Here $\lambda^\star$ prices u, half the squared radius, not the radius r. Per unit of radius, $\mathrm{d}p^\star/\mathrm{d}r = -(5-r)$, which is r times the price per unit of u by the chain rule and agrees with $-\lambda^\star$ only because $r = 1$. Rescaling a constraint rescales its multiplier: written as $\|\mathbf{x}\|^2 - 1 \le 0$, the same constraint would have multiplier 2.

The closest-point example of Section 35.6.1.3 is the equality case. There the minimum of $\frac{1}{2}\|\mathbf{x}\|^2$ over the hyperplane $\mathbf{a}^\top \mathbf{x} = b$ is $p^\star(b) = b^2/(2\|\mathbf{a}\|^2)$, and $\partial p^\star/\partial b = -\nu^\star$ is the proposition for the perturbation $h(\mathbf{x}) = \mathbf{a}^\top \mathbf{x} - b = v$, which moves the hyperplane to $b + v$. This is the rise of $0.02 = 0.2 \cdot 0.1$ that we observed there without explaining it.

35.7.3.2 Weight Decay and Norm Constraints

Weight decay (Section 2.7) adds a penalty to a training loss $\ell(\mathbf{w})$. A norm constraint limits the weights instead. With the main text's convention the two problems are

$$\min_{\mathbf{w}} \ \ell(\mathbf{w}) + \frac{\lambda}{2}\|\mathbf{w}\|^2 \qquad \text{and} \qquad \min_{\mathbf{w}} \ \ell(\mathbf{w}) \quad \text{subject to} \quad \tfrac{1}{2}\|\mathbf{w}\|^2 \le \tfrac{1}{2}r^2.$$

The penalized problem is stationary where $\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$. The constrained problem, with multiplier $\lambda \ge 0$ on $g(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 - \frac{1}{2}r^2$, has KKT stationarity $\nabla \ell(\mathbf{w}) + \lambda \mathbf{w} = \mathbf{0}$: the same equation, with the weight-decay coefficient in the role of the multiplier.

For a *convex* loss the correspondence holds in both directions. From penalty to constraint, let $\mathbf{w}_\lambda$ minimize the penalized objective and set the budget to $r = \|\mathbf{w}_\lambda\|$. Then $\mathbf{w}_\lambda$ is feasible, the constraint is active so complementary slackness holds, and stationarity is the penalized optimality condition. The KKT conditions are sufficient for convex problems (Section 35.6.2), so $\mathbf{w}_\lambda$ solves the constrained problem.

From constraint to penalty, take $r > 0$. The point $\mathbf{w} = \mathbf{0}$ is then strictly feasible, so Slater's condition holds. If the constrained minimum is attained at $\mathbf{w}^\star$, KKT necessity gives a multiplier $\lambda^\star(r) \ge 0$, and $\mathbf{w}^\star$ is among the minimizers of the convex function $\ell + \frac{\lambda^\star}{2}\|\mathbf{w}\|^2$. It is the only one when $\lambda^\star > 0$, which makes the penalized objective strictly convex. When the budget is slack, $\lambda^\star = 0$ and the penalized problem is the unregularized one. The shadow-price proposition reads $\lambda^\star(r) = -\partial p^\star/\partial(\frac{1}{2}r^2)$: the weight-decay coefficient is the rate at which the best loss falls per unit of half squared norm.

Neither direction used the shape of the squared norm. Let Ω be any convex penalty, for instance $\|\mathbf{w}\|_1$ for the lasso of Section 35.6.4.1. A minimizer $\mathbf{w}_\lambda$ of $\ell + \lambda\Omega$ solves the problem with budget $\Omega(\mathbf{w}) \le \Omega(\mathbf{w}_\lambda)$, because every $\mathbf{w}$ within that budget has $\ell(\mathbf{w}) \ge \ell(\mathbf{w}_\lambda) + \lambda\,(\Omega(\mathbf{w}_\lambda) - \Omega(\mathbf{w})) \ge \ell(\mathbf{w}_\lambda)$. Conversely, for convex ℓ, a budget above the minimum of Ω has a Slater point, and the saddle-point proposition makes a constrained minimizer a minimizer of $\ell + \lambda^\star\Omega$.

The equivalence needs the convexity of ℓ. For a nonconvex loss a penalized stationary point is still a KKT point of the matching constrained problem, but the two solution sets can differ, and some budgets are reached by no penalty at all (Exercise 7).

The same term has a third reading. For a squared loss with noise variance σ^2 and a Gaussian prior $\mathcal{N}(\mathbf{0}, \tau^2\mathbf{I})$ on the weights, $\frac{\lambda}{2}\|\mathbf{w}\|^2$ with $\lambda = \sigma^2/\tau^2$ is the negative log of the prior, up to a constant and the overall scale (Section 36.6). The one coefficient λ is thus a decay rate, a multiplier, and a ratio of variances.

Does the constrained problem with budget $r = \|\mathbf{w}_\lambda\|$ return the penalized solution, and does the multiplier recovered from its solution equal λ? The cell below takes the convex loss $\ell(\mathbf{w}) = \frac{1}{2}\|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$ with random data, solves the penalized problem in closed form from $(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})\mathbf{w} = \mathbf{A}^\top\mathbf{b}$, and solves the constrained problem by projected gradient descent with step $1/\lambda_{\max}(\mathbf{A}^\top\mathbf{A})$, where the projection onto the ball is clipping. On the active constraint, the inner product of stationarity with $\mathbf{w}$ gives $\lambda = -\mathbf{w}^\top\nabla\ell(\mathbf{w})/r^2$.

```
rng = np.random.default_rng()
A, b = rng.normal(size=(30, 5)), rng.normal(size=30)
grad = lambda w: A.T @ (A @ w - b)                    # gradient of 1/2 |Aw - b|^2
step = 1.0 / np.linalg.eigvalsh(A.T @ A).max()        # 1 / lambda_max(A^T A)
print(' lambda   r = |w_pen|   |w_con - w_pen|   recovered multiplier')
for lam in [0.1, 1.0, 10.0]:
    w_pen = np.linalg.solve(A.T @ A + lam * np.eye(5), A.T @ b)  # penalty form
    r = np.linalg.norm(w_pen)                         # its norm becomes the budget
    w = np.zeros(5)                                   # constraint form: PGD
    for _ in range(5000):
        w -= step * grad(w)
        w *= min(1.0, r / np.linalg.norm(w))          # project onto the r-ball
    lam_rec = -(w @ grad(w)) / r ** 2                 # KKT: grad l + lam w = 0
    print(f'{lam:7.2f}  {r:10.4f}  {np.linalg.norm(w - w_pen):15.2e}  {lam_rec:20.6f}')
```

```
 lambda   r = |w_pen|   |w_con - w_pen|   recovered multiplier
   0.10       0.4138         1.28e-16              0.100000
   1.00       0.3860         1.58e-16              1.000000
  10.00       0.2501         5.72e-17             10.000000
```

In every row the constrained solution matches the penalized one to machine precision, and the multiplier recovered from the constrained solution reproduces λ to the six printed decimals; the budgets r change with the random data. Penalty and constraint are two parameterizations of the same family of solutions, and the multiplier converts one into the other.

35.7.4 The Convex Conjugate

Computing q needs an unconstrained minimization for every value of the multipliers. For linear constraints this minimization has a structure that lets us do it once and for all. With constraints $\mathbf{A}\mathbf{x} \preceq \mathbf{b}$ the Lagrangian is $f(\mathbf{x}) + \lambda^\top(\mathbf{A}\mathbf{x} - \mathbf{b})$, so

$$q(\lambda) = \inf_{\mathbf{x}}\big[f(\mathbf{x}) + (\mathbf{A}^\top\lambda)^\top\mathbf{x}\big] - \mathbf{b}^\top\lambda.$$

The infimum depends on f and on one vector, $\mathbf{A}^\top\lambda$, and not otherwise on $\mathbf{A}$ or $\mathbf{b}$. If we tabulate, once for each f, the value of $\inf_{\mathbf{x}}[f(\mathbf{x}) - \mathbf{y}^\top\mathbf{x}]$ as a function of $\mathbf{y}$, then every lin-

early constrained dual with objective f can be read off the table. That table is the convex conjugate.

35.7.4.1 Supporting Lines

Start in one dimension with $f(x) = \frac{1}{2}x^2$ and the slope $y = 2$. Which lines $t = 2x + c$ of slope 2 stay below the graph? We need $2x + c \le \frac{1}{2}x^2$ for every x, that is, $c \le \min_x(\frac{1}{2}x^2 - 2x) = -2$, attained at $x = 2$. The highest such line, $t = 2x - 2$, touches the parabola at $(2, 2)$, where its slope matches $f'(2) = 2$. Lowering a line of slope 2 until it touches the graph thus puts its intercept at -2, and the number $\max_x(2x - \frac{1}{2}x^2) = 2$ is minus that intercept. Figure 35.23(a) draws this.

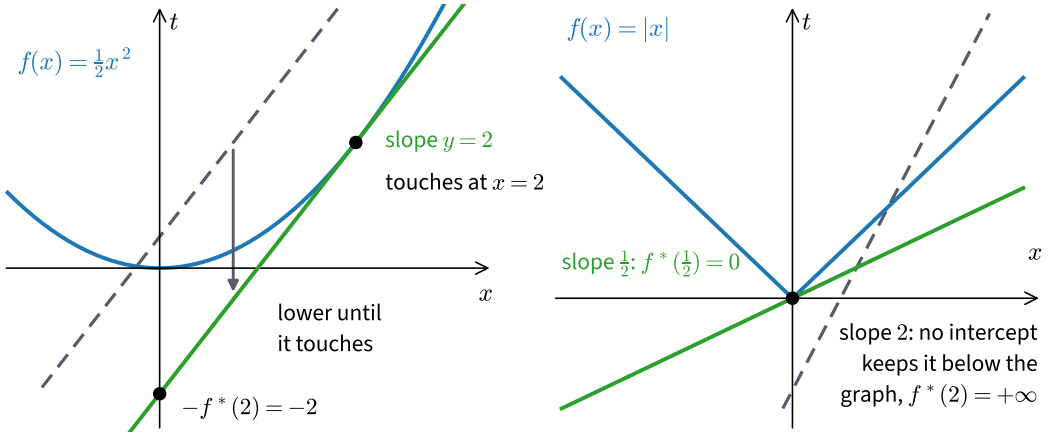


Figure 35.23: The convex conjugate as a supporting line. (a) For $f(x) = \frac{1}{2}x^2$ and slope 2, the dashed line of slope 2 is lowered until it touches the graph, at $x = 2$; its intercept is -2, so $f^*(2) = 2$. (b) For $f(x) = |x|$ a line of slope $\frac{1}{2}$ touches at the origin with intercept 0, while a line of slope 2 eventually runs above the graph for every intercept: $f^*(\frac{1}{2}) = 0$ and $f^*(2) = +\infty$. The conjugate is finite exactly for the slopes that admit a line below the graph.

The **convex conjugate** (or Fenchel–Legendre transform) of a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ records this number for every slope (an asterisk $*$ marks a conjugate, a star $\star$ an optimum):

$$f^*(\mathbf{y}) \;=\; \sup_{\mathbf{x}} \; \left(\mathbf{y}^\top \mathbf{x} - f(\mathbf{x})\right). \tag{35.84}$$

That is, $f^*(\mathbf{y})$ is minus the intercept of the highest affine function with slope $\mathbf{y}$ that lies below the graph of f. The conjugate describes f by its supporting lines instead of its values. For $f(x) = \frac{1}{2}x^2$ and any y the supremum is attained at $x = y$, so $f^*(y) = \frac{1}{2}y^2$.

We allow f to take the value $+\infty$, which encodes a domain. Points where $f = +\infty$ never attain the supremum, and convexity of such a function means convexity of its epigraph.

The supremum can be $+\infty$. For $f(x) = |x|$ and slope 2, $2x - |x| = x$ for $x > 0$, which grows without bound: no line of slope 2 stays below the V, whatever its intercept. For $|y| \le 1$ instead, $yx - |x| \le 0$ with equality at $x = 0$, so the line of slope y through the origin is a

supporting line. Hence $f^*(y) = 0$ for $|y| \le 1$ and $f^*(y) = +\infty$ otherwise (Figure 35.23(b)). The conjugate of the absolute value is the *indicator* of the interval $[-1, 1]$, the function that is 0 on the set and $+\infty$ off it.

Two properties follow directly from the definition. First, f^* is convex for every f, convex or not. For fixed $\mathbf{x}$ the expression $\mathbf{y}^\top\mathbf{x} - f(\mathbf{x})$ is affine in $\mathbf{y}$, and a pointwise supremum of affine functions is convex, because its epigraph is the intersection of the half-spaces above them. The same argument with an infimum made q concave.

Second, the definition gives $f^*(\mathbf{y}) \ge \mathbf{y}^\top\mathbf{x} - f(\mathbf{x})$ for every $\mathbf{x}$, which rearranges into the **Fenchel–Young inequality**,

$$\mathbf{y}^\top\mathbf{x} \;\le\; f(\mathbf{x}) + f^*(\mathbf{y}) \qquad \text{for all } \mathbf{x}, \mathbf{y}. \tag{35.85}$$

Equality holds exactly when $\mathbf{x}$ attains the supremum in Equation 35.84. For a differentiable *convex* f this happens exactly when $\mathbf{y} = \nabla f(\mathbf{x})$, since $\mathbf{x} \mapsto \mathbf{y}^\top\mathbf{x} - f(\mathbf{x})$ is then concave and maximized where its gradient vanishes. Convexity is needed. The double well $\phi_0(x) = (x^2 - 1)^2$ has $\phi_0'(0) = 0$, but $\phi_0^*(0) = -\min \phi_0 = 0$ and $\phi_0(0) + \phi_0^*(0) = 1 > 0$, because $x = 0$ is a local maximum of ϕ_0, not a point where the line of slope 0 touches from below. For the self-conjugate pair $\frac{1}{2}\|\mathbf{x}\|^2$, Fenchel–Young becomes the familiar $\mathbf{y}^\top\mathbf{x} \le \frac{1}{2}\|\mathbf{x}\|^2 + \frac{1}{2}\|\mathbf{y}\|^2$.

Does the conjugate lose information? For a convex f that is *proper* (never $-\infty$ and not $+\infty$ everywhere) and *closed* (its epigraph is a closed set), the supporting lines determine the graph, and applying the conjugate twice recovers it: $f^{**} = f$ (Rockafellar 1970). For a nonconvex f that has at least one affine function below it, f^{**} is the largest closed convex function below f, its convex envelope (Exercise 1d).

The pairs used in this section and its exercises are:

- $\frac{1}{2}\|\mathbf{x}\|^2$ and $\frac{1}{2}\|\mathbf{y}\|^2$, the self-conjugate pair;
- $\frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ and $\frac{1}{2}\mathbf{y}^\top\mathbf{A}^{-1}\mathbf{y}$ for $\mathbf{A} \succ 0$ (the maximizer is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$);
- e^x and $y\log y - y$ for $y > 0$, with value 0 at $y = 0$ and $+\infty$ for $y < 0$ (the maximizer is $x = \log y$; for instance $f^*(1) = -1$ and $f^*(e) = 0$);
- $\lambda\|\mathbf{x}\|_1$ and the indicator of the box $\{\mathbf{y} : \|\mathbf{y}\|_\infty \le \lambda\}$, the vector version of the absolute value above.

35.7.4.2 Duals of Linearly Constrained Problems

We now return to the problem that motivated the conjugate, $\min f(\mathbf{x})$ subject to $\mathbf{A}\mathbf{x} \preceq \mathbf{b}$. The one step that is not routine is pulling a minus sign out of the infimum, which turns it into the supremum of the conjugate's definition:

$$\begin{aligned} q(\lambda) &= \inf_{\mathbf{x}} \left[f(\mathbf{x}) + (\mathbf{A}^\top\lambda)^\top\mathbf{x} \right] - \mathbf{b}^\top\lambda \\ &= -\sup_{\mathbf{x}} \left[(-\mathbf{A}^\top\lambda)^\top\mathbf{x} - f(\mathbf{x}) \right] - \mathbf{b}^\top\lambda \\ &= -f^*(-\mathbf{A}^\top\lambda) - \mathbf{b}^\top\lambda. \end{aligned} \tag{35.86}$$

For equality constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$ the same three lines give $q(\nu) = -f^*(-\mathbf{A}^\top\nu) - \mathbf{b}^\top\nu$, with ν of either sign. The dual function is concave for a second reason now: f^* is convex and its argument is linear in λ.

The smallest case is the closest point of a half-plane to the origin: minimize $\frac{1}{2}\|\mathbf{x}\|^2$ subject to $\mathbf{a}^\top\mathbf{x} \le b$ with $\mathbf{a} = (1, 2)$ and $b = -1$. The origin violates the constraint ($\mathbf{a}^\top\mathbf{0} = 0 > -1$), so the constraint will be active. With $f^*(\mathbf{y}) = \frac{1}{2}\|\mathbf{y}\|^2$ and $\|\mathbf{a}\|^2 = 5$, Equation 35.86 gives at once

$$q(\lambda) = -\tfrac{1}{2}\lambda^2\|\mathbf{a}\|^2 - b\lambda = \lambda - \tfrac{5}{2}\lambda^2.$$

Setting $q'(\lambda) = 1 - 5\lambda$ to zero gives $\lambda^\star = \frac{1}{5}$ and $d^\star = \frac{1}{5} - \frac{1}{10} = \frac{1}{10}$. The minimizer of the Lagrangian at $\lambda^\star$ is the maximizer in the conjugate, $\mathbf{x} = -\lambda^\star\mathbf{a} = (-0.2, -0.4)$. It satisfies $\mathbf{a}^\top\mathbf{x} = -1 = b$ and has value $\frac{1}{2}(0.04 + 0.16) = \frac{1}{10}$, so it is feasible with value $d^\star$, and weak duality makes it optimal. It is the projection $(b/\|\mathbf{a}\|^2)\,\mathbf{a}$ of Section 35.6.1.3, here for a different line. The dual had one variable where the primal had two, and solving it took one line.

35.7.4.3 Log-Sum-Exp and Entropy

The softmax turns scores into probabilities by exponentiating and normalizing. Why exponentials? The conjugate of log-sum-exp, $\operatorname{lse}(\mathbf{x}) = \log\sum_i e^{x_i}$, answers this. Let $\Delta = \{\mathbf{p} : \mathbf{p} \succeq 0,\ \mathbf{1}^\top\mathbf{p} = 1\}$ be the probability simplex and $H(\mathbf{p}) = -\sum_i p_i \log p_i$ the entropy.

We compute $\operatorname{lse}^*(\mathbf{p}) = \sup_{\mathbf{x}} \mathbf{p}^\top\mathbf{x} - \operatorname{lse}(\mathbf{x})$. If $\mathbf{p}$ has a negative coordinate, sending that $x_i \to -\infty$ makes the objective unbounded. If $\mathbf{1}^\top\mathbf{p} \ne 1$, moving $\mathbf{x} = c\mathbf{1}$ toward $\pm\infty$ does the same, because $\operatorname{lse}(c\mathbf{1}) = c + \log n$. So the supremum is finite only on the simplex. There, stationarity demands $\operatorname{softmax}(\mathbf{x}) = \mathbf{p}$, solved by $\mathbf{x} = \log\mathbf{p}$, and the value is $\sum_i p_i \log p_i$.

The stationarity argument covers $\mathbf{p}$ with all coordinates positive, since softmax outputs are strictly positive. For $\mathbf{p}$ on the boundary of the simplex, with some $p_i = 0$, no maximizer exists. Sending those $x_i \to -\infty$ still approaches the same value, so the formula holds on the whole simplex with the convention $0 \log 0 = 0$. Because lse is convex and finite, $\operatorname{lse}^{**} = \operatorname{lse}$, and

$$\operatorname{lse}^*(\mathbf{p}) = \begin{cases} \sum_i p_i \log p_i = -H(\mathbf{p}), & \mathbf{p} \in \Delta, \\ +\infty, & \text{otherwise}, \end{cases} \qquad \text{and dually} \quad \operatorname{lse}(\mathbf{x}) = \max_{\mathbf{p}\in\Delta} \left(\mathbf{p}^\top\mathbf{x} + H(\mathbf{p})\right). \tag{35.87}$$

Log-sum-exp is the largest value of a linear score plus an entropy bonus over all distributions on n outcomes. The maximizer is the softmax. By the equality case of Fenchel–Young it is the $\mathbf{p}$ with $\mathbf{p} = \nabla\operatorname{lse}(\mathbf{x})$, and differentiating gives $\partial \operatorname{lse}/\partial x_i = e^{x_i}/\sum_j e^{x_j}$, the softmax (the Hessian is computed in Section 35.3). Equivalently, among all distributions with the same expected score $\mathbf{p}^\top\mathbf{x}$, it has the largest entropy, the maximum-entropy property proved in Section 36.4.2. Exercise 2 derives the softmax from this constraint side. Without the entropy term the maximizer of $\mathbf{p}^\top\mathbf{x}$ is a one-hot vector at $\arg\max_i x_i$. The entropy replaces this hard selection by a distribution. The same construction, applied to the convex function that defines an f-divergence, gives the variational representations of divergences used to train adversarial models (Section 37.3.4).

35.7.5 Worked Examples

Three examples show what the dual exposes. In the support vector machine, complementary slackness makes the solution depend on a few data points, and the dual sees the data only through inner products. Water-filling is the standard example of a multiplier with units an engineer cares about, rate per unit of power. The last example shows the gap that convexity prevents.

35.7.5.1 The Support Vector Machine Dual

Given linearly separable data $(\mathbf{x}_i, y_i)$ with labels $y_i \in \{\pm 1\}$, the support vector machine (Cortes and Vapnik 1995) looks for the separating hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$ with the widest margin. It solves

$$\min_{\mathbf{w},b} \ \tfrac{1}{2}\|\mathbf{w}\|^2 \quad \text{subject to} \quad y_i\,(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1, \quad i = 1, \dots, n. \tag{35.88}$$

A point $\mathbf{x}$ lies at distance $|\mathbf{w}^\top \mathbf{x} + b|/\|\mathbf{w}\|$ from the hyperplane (the signed-distance formula of Section 33.1), so the two margin hyperplanes $\mathbf{w}^\top \mathbf{x} + b = \pm 1$ sit at distance $1/\|\mathbf{w}\|$ on either side of the boundary, $2/\|\mathbf{w}\|$ apart. Minimizing $\|\mathbf{w}\|$ maximizes the margin. This is a convex quadratic program with one inequality per data point. The constraints are affine, so Slater's condition needs only a feasible point, and scaling any separating $(\mathbf{w}, b)$ up until every margin is at least 1 gives one. So strong duality is guaranteed. Form the Lagrangian with multipliers $\alpha_i \geq 0$,

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \tfrac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i \left(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right),$$

and compute the dual function $q(\alpha) = \inf_{\mathbf{w},b} \mathcal{L}$. The Lagrangian is affine in b, with coefficient $-\sum_i \alpha_i y_i$. Unless that coefficient vanishes, sending b to $\pm\infty$ drives $\mathcal{L}$ to $-\infty$, so $q(\alpha) = -\infty$ whenever $\sum_i \alpha_i y_i \neq 0$. This is the effective domain of Equation 35.80 at work: the condition $\sum_i \alpha_i y_i = 0$ reappears as an equality constraint of the dual. On that domain the b-term drops out, leaving a strictly convex quadratic in $\mathbf{w}$ whose infimum is attained at its stationary point,

$$\mathbf{w} = \sum_i \alpha_i y_i \, \mathbf{x}_i, \qquad \text{on the domain} \qquad \sum_i \alpha_i y_i = 0.$$

The weight vector is a combination of the data points, with the multipliers as coefficients. Substituting back yields the **SVM dual**:

$$\max_{\alpha \geq 0,\ \sum_i \alpha_i y_i = 0} \ \sum_i \alpha_i \ - \ \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j \, y_i y_j \, \mathbf{x}_i^\top \mathbf{x}_j. \tag{35.89}$$

Two consequences hold before any algorithm is chosen. First, complementary slackness, $\alpha_i(1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)) = 0$, says that a positive multiplier forces the margin of point i to equal 1. Such points are the **support vectors**, and they lie on the margin hyperplanes. In the terms of Figure 35.19, their constraints are active, and only their constraint normals enter the stationarity balance. Points with $\alpha_i = 0$ do not contribute to the weight vector. Second,

the data enter Equation 35.89 only through the inner products $\mathbf{x}_i^\top \mathbf{x}_j$. Replacing them by a kernel evaluation $k(\mathbf{x}_i, \mathbf{x}_j)$ gives a nonlinear classifier, an option the primal formulation does not expose directly.

The equality constraint $\sum_i \alpha_i y_i = 0$ entered only because the unpenalized offset b made the Lagrangian affine in one direction. To keep the dual feasible set an orthant, on which projection clips each coordinate at zero, the cell below solves a variant with the offset folded into the weights: with a constant feature $\tilde{\mathbf{x}}_i = (\mathbf{x}_i, 1)$ and $\tilde{\mathbf{w}} = (\mathbf{w}, b)$, minimize $\frac{1}{2}\|\tilde{\mathbf{w}}\|^2$ subject to $y_i \tilde{\mathbf{w}}^\top \tilde{\mathbf{x}}_i \geq 1$. This is a different problem, since it also penalizes the offset, and its boundary can differ slightly from that of Equation 35.88; but its Lagrangian is strictly convex in every primal variable, so no equality constraint appears. Writing $\mathbf{Q}_{ij} = y_i y_j \tilde{\mathbf{x}}_i^\top \tilde{\mathbf{x}}_j$, the dual is

$$\max_{\alpha \succeq 0} \quad \mathbf{1}^\top \alpha - \tfrac{1}{2} \alpha^\top \mathbf{Q}\, \alpha, \tag{35.90}$$

a concave quadratic over the orthant. Projected gradient ascent, the update of Section 35.6.3 with the sign of the step reversed, iterates $\alpha \leftarrow \max(\mathbf{0},\ \alpha + \eta\,(\mathbf{1} - \mathbf{Q}\alpha))$ with $\eta = 1/\lambda_{\max}(\mathbf{Q})$, the descent-lemma step for a gradient that is $\lambda_{\max}(\mathbf{Q})$-Lipschitz. At its limit three things should hold: the hyperplane separates the data, the positive multipliers sit on points of margin 1, and the dual value meets the primal value. The cell runs the ascent on eight points in the plane.

```
X = np.array([[1.0, 1.5], [2.0, 0.5], [2.5, 2.0], [1.5, 3.0],
              [-1.0, -0.5], [-2.0, 1.0], [-1.5, -2.0], [0.5, -1.5]])
y = np.array([1.0, 1.0, 1.0, 1.0, -1.0, -1.0, -1.0, -1.0])
Xa = np.hstack([X, np.ones((len(X), 1))])     # fold the offset b into w
G = y[:, None] * Xa                           # rows y_i x~_i
Q = G @ G.T                                   # Q_ij = y_i y_j x~_i^T x~_j
alpha = np.zeros(len(X))
eta = 1.0 / np.linalg.eigvalsh(Q).max()       # 1 / lambda_max(Q)
for _ in range(5000):                         # projected gradient ascent
    alpha = np.maximum(0.0, alpha + eta * (1.0 - Q @ alpha))
w = G.T @ alpha                               # w~* = sum_i alpha_i y_i x~_i
margins = y * (Xa @ w)
dual, primal = alpha.sum() - 0.5 * alpha @ Q @ alpha, 0.5 * w @ w
print('alpha* =', alpha.round(4))
print('rank of Q =', np.linalg.matrix_rank(Q), '  multipliers > 0:', int((alpha >
 ↪ 1e-8).sum()))
print('w* =', w[:2].round(4), ' b* =', w[2].round(4),
      ' separates data:', bool((np.sign(Xa @ w) == y).all()))
print('margins y_i (w^T x_i + b) =', margins.round(4))
print(f'primal = {primal:.6f}, dual = {dual:.6f}, gap = {abs(primal - dual):.1e}, '
      f'comp. slack = {np.abs(alpha * (margins - 1.0)).max():.1e}')
```

```
alpha* = [0.1309 0.0732 0.     0.     0.     0.2442 0.     0.3885]
rank of Q = 3   multipliers > 0: 4
w* = [0.5714 0.5714]  b* = -0.4286  separates data: True
margins y_i (w^T x_i + b) = [1.     1.     2.1429 2.1429 1.2857 1.     2.4286 1.    ]
primal = 0.418367, dual = 0.418367, gap = 4.4e-16, comp. slack = 3.0e-16
```

All three hold. Four multipliers are positive, and those four points have margin 1; the other four sit at margins 1.29 to 2.43 with $\alpha_i = 0$. The printed digits identify the exact solution,

$\tilde{\mathbf{w}}^\star = (\frac{4}{7}, \frac{4}{7}, -\frac{3}{7})$, that is, $\mathbf{w}^\star = (\frac{4}{7}, \frac{4}{7})$ and $b^\star = -\frac{3}{7}$, with $p^\star = d^\star = \frac{41}{98} \approx 0.418367$ and a gap near 10^{-16}. The equal values also follow from an identity, $\sum_i \alpha_i^\star = \|\tilde{\mathbf{w}}^\star\|^2$ at the optimum (Exercise 4a).

The printout also shows a limit of reading the dual. The primal solution $\tilde{\mathbf{w}}^\star$ is unique, because the primal objective is strictly convex. The dual solution is not: $\mathbf{Q} = \mathbf{G}\mathbf{G}^\top$, with $\mathbf{G}$ the 8×3 matrix of rows $y_i \tilde{\mathbf{x}}_i^\top$, has rank 3. The four rows of the support vectors are four vectors in $\mathbb{R}^3$, so some nonzero $\mathbf{d}$ supported on them has $\mathbf{G}^\top \mathbf{d} = \mathbf{0}$. Along $\mathbf{d}$ the quadratic term of the dual objective does not change, and neither does the linear term: $\mathbf{1}^\top \mathbf{d} = \sum_i d_i\, y_i \tilde{\mathbf{w}}^{\star\top} \tilde{\mathbf{x}}_i = \tilde{\mathbf{w}}^{\star\top} \mathbf{G}^\top \mathbf{d} = 0$, because every support vector has margin 1. Moving $\alpha^\star$ along $\mathbf{d}$ therefore keeps $\tilde{\mathbf{w}}^\star$, the dual value, and feasibility until a coordinate reaches zero. The optimal multipliers form a segment, and projected ascent from $\mathbf{0}$ returns one point of it. At either end of the segment, one of the four margin-1 points has $\alpha_i = 0$ (Exercise 4c). Complementary slackness works in one direction: $\alpha_i > 0$ implies margin 1, but margin 1 does not imply $\alpha_i > 0$.

35.7.5.2 Water-Filling

The second example has a closed-form solution with a geometric picture. A transmitter splits a power budget P across n independent channels. Channel i has noise power $N_i > 0$, and with power P_i it carries $\log(1 + P_i/N_i)$ nats per use (Cover and Thomas 1999). (Cover and Thomas write $\frac{1}{2}\log$ for a real Gaussian channel; the factor rescales the multiplier below but not the allocation.) The allocation problem is

$$\max_{\mathbf{P}} \; U(\mathbf{P}) = \sum_{i=1}^{n} \log\left(1 + \frac{P_i}{N_i}\right) \quad \text{subject to} \quad P_i \geq 0, \;\; \sum_i P_i = P, \tag{35.91}$$

a concave objective maximized under affine constraints, so minimizing $-U$ is a convex problem. Slater's condition holds because the constraints are affine and a positive allocation exists, so the KKT conditions characterize the optimum. With multiplier ν for the budget and $\lambda_i \geq 0$ for $P_i \geq 0$, stationarity of the Lagrangian of $-U$ reads

$$\frac{1}{N_i + P_i} = \nu - \lambda_i.$$

On a channel with $P_i > 0$, complementary slackness sets $\lambda_i = 0$ and forces $N_i + P_i = 1/\nu$, the same value for every channel that receives power. On a channel with $P_i = 0$, dual feasibility requires $\lambda_i = \nu - 1/N_i \geq 0$, that is, $N_i \geq 1/\nu$. Writing $\theta = 1/\nu$ for the common value,

$$P_i^\star = \max(\theta - N_i,\, 0), \qquad \text{with } \theta \text{ set by } \sum_i \max(\theta - N_i,\, 0) = P.$$

This is the **water-filling** solution. Draw each channel as a basin with floor at height N_i and pour in P units of water. As Figure 35.24 shows, the water settles at a common level θ, fills the quiet channels most, and never reaches basins whose floor is above the level. The left side of the level equation is continuous, piecewise linear, zero for $\theta \leq \min_i N_i$, and strictly increasing for $\theta > \min_i N_i$, so for $P > 0$ it has exactly one root, and bisection finds it. The

simplex projection of Section 35.6.3 has the same threshold structure upside down: there a common level is subtracted from the coordinates and the result is clipped at zero.

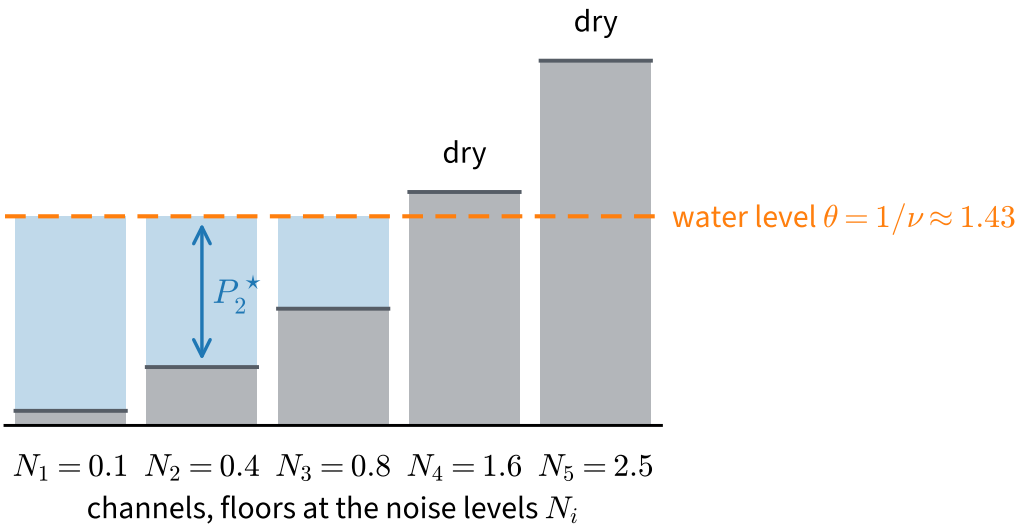


Figure 35.24: Water-filling with the cell's noise floors and budget: pouring $P = 3$ units of power into basins with floors at N_i fills the three quiet channels to the common level $\theta = 1/\nu^\star \approx 1.43$, allocating $P_i^\star = \theta - N_i$ to each; the two basins whose floors exceed the level receive nothing, as complementary slackness requires.

The shadow-price proposition gives the multiplier its engineering meaning. Maximizing U is minimizing $-U$, and relaxing the budget to $P + v$ changes $-U^\star$ at rate $-\nu^\star$, so $\mathrm{d}U^\star/\mathrm{d}P = \nu^\star = 1/\theta$: the extra rate from one more unit of power. The cell below computes the allocation for five channels and checks the price by re-solving at $P \pm 10^{-4}$.

```
N = np.array([0.1, 0.4, 0.8, 1.6, 2.5])       # channel noise levels
P = 3.0                                       # total power budget

def water_fill(P, N, steps=60):
    lo, hi = N.min(), N.min() + P             # bracket for the water level
    for _ in range(steps):                    # bisection on the level theta
        mid = 0.5 * (lo + hi)
        lo, hi = (mid, hi) if np.maximum(0, mid - N).sum() < P else (lo, mid)
    theta = 0.5 * (lo + hi)
    return theta, np.maximum(0.0, theta - N)

theta, power = water_fill(P, N)
print('water level theta =', f'{theta:.6f}', ' P* =', power.round(4),
      ' total =', f'{power.sum():.6f}')
print('wet channels at the common level:', (N + power)[power > 0].round(6))
print('dry channels above the level:    ', bool((N[power == 0] >= theta).all()))
rate = lambda power: np.log(1.0 + power / N).sum()
dP = 1e-4                                     # shadow price, two ways
nu = 1.0 / theta
```

```
    sens = (rate(water_fill(P + dP, N)[1]) - rate(water_fill(P - dP, N)[1])) / (2 * dP)
    print(f'nu = 1/theta = {nu:.6f} vs finite-difference dU*/dP = {sens:.6f}')
```

```
water level theta = 1.433333  P* = [1.3333 1.0333 0.6333 0.     0.    ]  total = 3.000000
wet channels at the common level: [1.433333 1.433333 1.433333]
dry channels above the level:     True
nu = 1/theta = 0.697674 vs finite-difference dU*/dP = 0.697674
```

The budget is met to the printed precision, the three wet channels sit at the same level $\theta = 1.4333$, and the channels with floors 1.6 and 2.5 stay dry, as dual feasibility demands. Here the root is exact: $\theta = (3 + 0.1 + 0.4 + 0.8)/3$. The multiplier $\nu^\star = 1/\theta = 0.697674$ matches the finite-difference sensitivity of the optimal rate to the budget in all six printed digits.

35.7.5.3 An Explicit Duality Gap

Weak duality always holds; strong duality need not. A one-dimensional problem shows a gap. On the domain $\mathcal{D} = [0, 1]$,

$$\min_{x \in [0,1]} \; f(x) = -x^2 \quad \text{subject to} \quad g(x) = x - \tfrac{1}{2} \le 0.$$

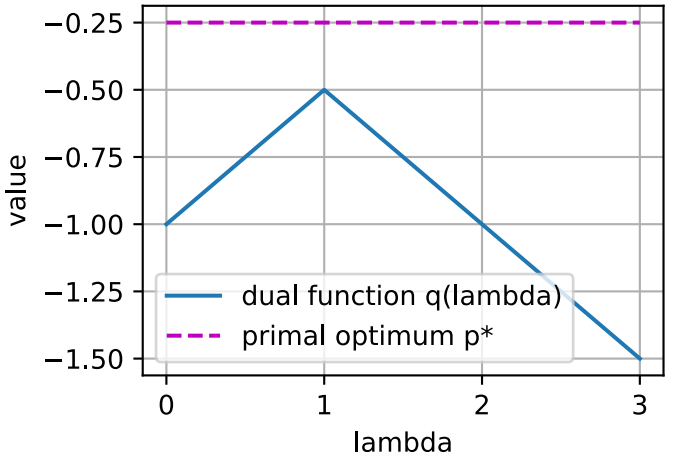
The objective is concave, so the problem is not convex. The feasible set is $[0, \frac{1}{2}]$, and the optimum is at its right end, $p^\star = f(\frac{1}{2}) = -\frac{1}{4}$. Only the constraint $g(x) \le 0$ is dualized: the domain $[0, 1]$ stays a hard constraint, so the infimum in Equation 35.80 runs over $[0, 1]$, and the proof of weak duality goes through unchanged. The Lagrangian $\mathcal{L}(x, \lambda) = -x^2 + \lambda(x - \frac{1}{2})$ is concave in x, so its minimum over $[0, 1]$ is at an endpoint: $\mathcal{L}(0, \lambda) = -\lambda/2$ and $\mathcal{L}(1, \lambda) = \lambda/2 - 1$. Hence

$$q(\lambda) = \min\left(-\frac{\lambda}{2},\; \frac{\lambda}{2} - 1\right),$$

piecewise linear and concave, as it must be. The two pieces cross at $\lambda^\star = 1$, where $d^\star = -\frac{1}{2} < -\frac{1}{4} = p^\star$: a duality gap of $\frac{1}{4}$. The cell evaluates $p^\star$ and q on grids and plots the dual function against $p^\star$, the counterpart of the first plot of this section.

```
    xs = np.linspace(0.0, 1.0, 4001)              # the domain D = [0, 1]
    fx, gx = -xs**2, xs - 0.5                     # objective and constraint on D
    p_star = fx[gx <= 0.0].min()                  # primal optimum: -1/4
    lams = np.linspace(0.0, 3.0, 601)
    q = np.array([(fx + lam * gx).min() for lam in lams])  # dual function on D
    d_star, lam_star = q.max(), lams[q.argmax()]
    print(f'p* = {p_star:.4f},  d* = {d_star:.4f} at lambda = {lam_star:.2f},'
          f'  gap = {p_star - d_star:.4f}')
    d2l.plot(lams, [q, np.full_like(lams, p_star)], 'lambda', 'value',
             legend=['dual function q(lambda)', 'primal optimum p*'])
```

```
p* = -0.2500,  d* = -0.5000 at lambda = 1.00,  gap = 0.2500
```



Where the curve of the clipping problem touched its line, the tent peaks at $-\frac{1}{2}$, a quarter below the primal optimum. A strictly feasible point exists ($x = \frac{1}{4}$ lies inside $\mathcal{D}$ and has $g(\frac{1}{4}) = -\frac{1}{4} < 0$), so the gap is caused by the missing convexity, not by a missing Slater point. In panel (b) of Figure 35.22, the image curve $\{(u, -(u + \frac{1}{2})^2) : u \in [-\frac{1}{2}, \frac{1}{2}]\}$ bends the wrong way, and no line below $\mathcal{A}$ passes through its point at $u = 0$; the best one, of slope -1, certifies only $-\frac{1}{2}$. The gap also depends on how the problem is written (Exercise 5).

Two questions remain open. A nonconvex problem can keep a gap, so its dual bounds but does not solve it. And iterating on $\mathbf{x}$ and λ together need not converge even when a saddle point exists, which is the difficulty behind minimax training. Training a deep network is not a convex problem, but many of its subproblems are: projections, clipped updates, last-layer fits, and trust-region steps on a convex model, and everything in this section applies to them. All of it also assumed exact arithmetic; the next section asks when finite precision, rather than the algorithm, decides the answer.

35.7.6 Summary

The dual function $q(\lambda, \nu) = \inf_{\mathbf{x}} \mathcal{L}$ is concave for every problem and never exceeds $p^\star$, so each choice of multipliers $\lambda \succeq 0$ certifies a lower bound. For a convex problem with a strictly feasible point (Slater's condition) the best bound is exact, and attained primal and dual optima form a saddle point of the Lagrangian. For linear constraints the dual is a conjugate, $q(\lambda) = -f^*(-\mathbf{A}^\top \lambda) - \mathbf{b}^\top \lambda$, which turns a table of conjugates into a table of duals. Under strong duality the optimal multipliers are shadow prices, with $\lambda_i^\star = -\partial p^\star / \partial u_i$ and $\nu_j^\star = -\partial p^\star / \partial v_j$ when $p^\star$ is differentiable. The SVM multipliers are positive only on support vectors, the multiplier $1/\theta$ of the power budget in water-filling is the rate gained per unit of power, and for a convex loss the weight-decay coefficient is the multiplier of a norm budget. A nonconvex problem can keep a gap, as the concave objective with $p^\star - d^\star = \frac{1}{4}$ shows.

35.7.7 Exercises

1. **Conjugate pairs.** Compute each conjugate from Equation 35.84.
    1. $f(\mathbf{x}) = \lambda \|\mathbf{x}\|_1$ with $\lambda > 0$.

2. $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^\top \mathbf{A}\mathbf{x}$ with $\mathbf{A} \succ 0$. What does the Fenchel–Young inequality Equation 35.85 state for this pair?
3. The softplus $f(t) = \log(1+e^t)$ on $\mathbb{R}$. The binary cross-entropy of a logit t against a soft label $p \in [0,1]$ is $\log(1+e^t) - pt$. Use Fenchel–Young to show that it is at least the binary entropy $H(p) = -p\log p - (1-p)\log(1-p)$, with equality exactly when $p = 1/(1+e^{-t})$, the sigmoid of t. Write the difference as a KL divergence.
4. The double well $\phi_0(x) = (x^2-1)^2$ of Section 35.3.4 (the function ϕ there without its tilt $x/2$) is not convex. Find its convex envelope ϕ_0^{**} and the set on which $\phi_0^{**} < \phi_0$.

Adapted from Boyd and Vandenberghe, Convex Optimization, Examples 3.22 and 3.26.

2. **Maximum entropy and temperature.** The entropy of a probability vector $\mathbf{p} \in \mathbb{R}^n$ is $H(\mathbf{p}) = -\sum_i p_i \log p_i$. This exercise derives the softmax from the constraint side.
   1. Maximize H subject to $\sum_i p_i = 1$ with one Lagrange multiplier, ignoring the sign constraints $p_i \geq 0$. Show that the only stationary point is the uniform distribution. Why does ignoring the sign constraints not change the answer?
   2. Fix scores $s_1, \dots, s_n$, not all equal, and a target $\bar{s}$ strictly between $\min_i s_i$ and $\max_i s_i$, and add the constraint $\sum_i p_i s_i = \bar{s}$. Show that the maximizer is $\mathbf{p} = \mathrm{softmax}(\beta \mathbf{s})$ for a scalar β determined by the multiplier of the score constraint. Show that the mean score under $\mathrm{softmax}(\beta\mathbf{s})$ increases strictly with β, so that exactly one β meets the target.
   3. Let $H^\star(\bar{s})$ be the maximal entropy. Show that $\mathrm{d}H^\star/\mathrm{d}\bar{s} = -\beta$, so that the multiplier prices the score constraint as the multiplier of Section 35.6.1.3 prices the offset b. For which targets $\bar{s}$ is β negative? What happens to the softmax temperature $1/\beta$ as $\bar{s}$ approaches $\max_i s_i$?

3. **A dual certificate for ℓ_1 minimization.** Basis pursuit,

$$\min_{\mathbf{x}} \ \|\mathbf{x}\|_1 \quad \text{subject to} \quad \mathbf{A}\mathbf{x} = \mathbf{b}, \tag{35.92}$$

with $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $m < n$, selects a sparse solution of an underdetermined linear system. Its constraints are affine, so Slater's condition needs only a feasible point and gives $d^\star = p^\star$ whenever the system has a solution. The conjugate of $\|\cdot\|_1$ is the indicator of the box $\{\mathbf{y} : \|\mathbf{y}\|_\infty \leq 1\}$.
   1. Use the equality form of Equation 35.86 to show that, after replacing ν by $-\nu$, the dual problem is $\max_\nu \mathbf{b}^\top \nu$ subject to $\|\mathbf{A}^\top \nu\|_\infty \leq 1$.
   2. Solve both problems by hand for $\mathbf{A} = (1 \ \ 2)$ and $b = 2$, and check $d^\star = p^\star$.
   3. Let $\mathbf{x}^\star$ and $\nu^\star$ be optimal. Show that $x_i^\star = 0$ wherever $|(\mathbf{A}^\top \nu^\star)_i| < 1$, and that $(\mathbf{A}^\top \nu^\star)_i = \mathrm{sign}(x_i^\star)$ wherever $x_i^\star \neq 0$. A candidate $\mathbf{x}$ with $\mathbf{A}\mathbf{x} = \mathbf{b}$ is given. Which conditions must a vector ν satisfy to certify that $\mathbf{x}$ is optimal without solving the problem?

4. **Support vectors and the dual value.** For the offset-folded SVM dual Equation 35.90, let $\alpha^\star$ be optimal and $\tilde{\mathbf{w}}^\star = \sum_i \alpha_i^\star y_i \tilde{\mathbf{x}}_i$.
   1. Show that $\sum_i \alpha_i^\star = \|\tilde{\mathbf{w}}^\star\|^2$ by writing the KKT conditions of the dual over the orthant. Conclude that the dual optimal value equals the primal value $\frac{1}{2}\|\tilde{\mathbf{w}}^\star\|^2$.

2. Show that deleting a training point with $\alpha_j^\star = 0$ leaves the remaining multipliers optimal for the smaller problem and leaves $\tilde{\mathbf{w}}^\star$ unchanged. In leave-one-out validation, each of the n points is classified by the SVM trained on the other $n-1$ points. Conclude that the leave-one-out error rate is at most the fraction of points with $\alpha_i^\star > 0$, for every optimal $\alpha^\star$. Which optimal $\alpha^\star$ gives the best bound?
3. code The matrix $\mathbf{Q}$ of the SVM cell has rank 3. Let S be the four indices with $\alpha_i^\star > 0$ and $\mathbf{G}_S$ the corresponding rows of `G`. Find a nonzero $\mathbf{d} \in \mathbb{R}^4$ with $\mathbf{G}_S^\top \mathbf{d} = \mathbf{0}$ (use `np.linalg.svd` of $\mathbf{G}_S^\top$), and move $\alpha_S^\star + t\mathbf{d}$ in each direction until a coordinate reaches zero. Verify that both endpoints are dual feasible, have dual value $\frac{41}{98}$, and give the same $\tilde{\mathbf{w}}^\star$. Which margin-1 point has $\alpha_i = 0$ at each endpoint? What does this say about reading "support vector" off the multipliers? (You should find the endpoints $(0, 0.204, 0, 0, 0, 0.192, 0, 0.441)$ and $(0.204, 0, 0, 0, 0, 0.273, 0, 0.359)$.)

5. code **Duality gaps and formulations.** The gap cell of Section 35.7.5.3 minimizes the Lagrangian over the domain $[0, 1]$ and dualizes only the constraint $x \le \frac{1}{2}$.

   1. Replace the objective by $f(x) = (x-1)^2$. Compute $p^\star$, the dual function, $d^\star$, and $\lambda^\star$ by hand, and confirm them with the modified cell. Why is the gap now zero?
   2. Keep $f(x) = -x^2$, but write the constraint as $x^2 \le \frac{1}{4}$, which defines the same feasible set on $[0, 1]$. Compute the new dual function and $d^\star$ by hand and with the cell.
   3. Keep $f(x) = -x^2$ and the constraint $x \le \frac{1}{2}$, but treat $0 \le x$ and $x \le 1$ as constraints with their own multipliers and minimize the Lagrangian over all of $\mathbb{R}$. What is $d^\star$ now? Rank the dual bounds of the three formulations of the same problem (the one in the text and those of the two previous parts). Is a duality gap a property of the problem or of its formulation?

6. code **Gradient descent–ascent.** ▯ Primal–dual methods and GAN training update the minimizing variable by gradient descent and the maximizing variable by gradient ascent at the same time.

   1. For the bilinear game $\min_x \max_y xy$, whose saddle point is the origin, write simultaneous descent–ascent, $x \leftarrow x - \eta y$ and $y \leftarrow y + \eta x$, as a linear map. Show that every nonzero start spirals away from the saddle for every $\eta > 0$. Show that the alternating version, in which the y-update uses the new x, is a linear map with determinant 1. What do its iterates do for $\eta < 2$?
   2. Run simultaneous descent–ascent on the Lagrangian $\mathcal{L}(\mathbf{x}, \nu) = \frac{1}{2}\mathbf{x}^\top\mathbf{x} + \nu(\mathbf{a}^\top\mathbf{x} - b)$ of the closest-point problem of Section 35.6.1.3, here with $\mathbf{a} = (1, 2)$, $b = 1$, and the start $\mathbf{x} = \mathbf{0}$, $\nu = 0$. Find the step sizes for which the iterates converge to $(\mathbf{x}^\star, \nu^\star)$. Derive them from the eigenvalues of the update acting on the pair $(\mathbf{a}^\top\mathbf{x}/\|\mathbf{a}\|, \nu)$ and on the components of $\mathbf{x}$ orthogonal to $\mathbf{a}$.
   3. Add $\frac{\epsilon}{2}x^2$ with $0 < \epsilon \le 2$ to the objective of the bilinear game. For which step sizes does simultaneous descent–ascent converge now? Which term of the Lagrangian of the previous part plays the role of ϵ?

7. code **Penalty versus constraint without convexity.** ▯ For a convex loss, Section 35.7.3.2 shows that weight decay and a norm constraint trace the same solutions. Take instead the nonconvex one-dimensional loss $\ell(w) = (w-1)^2(w-3)^2 - w$, whose local minima lie near $w = 1.16$ and $w = 3.11$.

1. On a fine grid of w, compute the penalized minimizers $w_\lambda = \operatorname{argmin}_w \ell(w) + \frac{\lambda}{2}w^2$ for λ on a logarithmic grid, and the constrained optimum $p^\star(r) = \min\{\ell(w) : w^2 \le r^2\}$ for $r \in [0, 3.2]$. Plot the points $(w_\lambda^2, \ell(w_\lambda))$ on top of the curve $(r^2, p^\star(r))$. Which budgets r are never the norm $|w_\lambda|$ of a penalized minimizer?
2. Show that w_λ minimizes $\ell(w) + \frac{\lambda}{2}w^2$ if and only if the line of slope $-\frac{\lambda}{2}$ through $(w_\lambda^2, \ell(w_\lambda))$ lies below every point of the set $\{(w^2, \ell(w)) : w \in \mathbb{R}\}$. Use this and the supporting-line picture of Figure 35.22 to explain the missing budgets.
3. Pick a budget r in the missing range at which the constraint is active. Compute the KKT multiplier λ of the constrained solution, with the constraint written $\frac{1}{2}w^2 \le \frac{1}{2}r^2$ as in the text, and find a point where the penalized objective with this λ is lower. Compute the duality gap $p^\star(r) - d^\star(r)$ of the constrained problem.

8. [extended] **Entropic optimal transport.** ⊠⊠ Given probability vectors $\mathbf{p} \in \mathbb{R}^n$ and $\mathbf{r} \in \mathbb{R}^m$ and a cost matrix $\mathbf{C} \in \mathbb{R}^{n \times m}$, optimal transport finds the cheapest plan Γ that moves the mass $\mathbf{p}$ to the mass $\mathbf{r}$,

$$\mathrm{OT} = \min_{\Gamma \ge 0} \sum_{i,j} C_{ij}\Gamma_{ij} \quad \text{subject to} \quad \Gamma\mathbf{1} = \mathbf{p}, \;\; \Gamma^\top\mathbf{1} = \mathbf{r}, \tag{35.93}$$

a linear program with nm variables. Entropic regularization (Cuturi 2013) adds $\varepsilon \sum_{i,j} \Gamma_{ij}(\log \Gamma_{ij} - 1)$ to the objective, with $\varepsilon > 0$; let Γ_ε be the regularized optimum. Its solution by Sinkhorn iterations is used to compare distributions (Peyré and Cuturi 2019).

1. Form the Lagrangian with the terms $\phi^\top(\mathbf{p} - \Gamma\mathbf{1}) + \psi^\top(\mathbf{r} - \Gamma^\top\mathbf{1})$ for the marginal constraints. Show that $\Gamma_\varepsilon = \operatorname{diag}(\mathbf{u})\,\mathbf{K}\,\operatorname{diag}(\mathbf{v})$ with $K_{ij} = e^{-C_{ij}/\varepsilon}$, $u_i = e^{\phi_i/\varepsilon}$, and $v_j = e^{\psi_j/\varepsilon}$. Why can the sign constraints $\Gamma_{ij} \ge 0$ be left out of the Lagrangian?
2. Show that the dual function is

$$q(\phi, \psi) = \mathbf{p}^\top\phi + \mathbf{r}^\top\psi - \varepsilon \sum_{i,j} e^{(\phi_i + \psi_j - C_{ij})/\varepsilon}, \tag{35.94}$$

and that maximizing it exactly over ϕ with ψ fixed gives the Sinkhorn update $\mathbf{u} = \mathbf{p}/(\mathbf{K}\mathbf{v})$, elementwise. Conclude that Sinkhorn's algorithm is block-coordinate ascent on a concave function and that its dual value never decreases. Why is every dual value a lower bound on OT?
3. Let $H(\Gamma) = -\sum_{i,j} \Gamma_{ij} \log \Gamma_{ij}$, and grant that every feasible plan satisfies $\max(H(\mathbf{p}), H(\mathbf{r})) \le H(\Gamma) \le H(\mathbf{p}) + H(\mathbf{r})$. Show that $0 \le \sum_{i,j} C_{ij}(\Gamma_\varepsilon)_{ij} - \mathrm{OT} \le \varepsilon \min(H(\mathbf{p}), H(\mathbf{r}))$.
4. Let the costs lie in $[0, 1]$. In float64, e^{-x} rounds to zero for x above about 745. For which ε can entries of $\mathbf{K}$ vanish, and what then happens to the multiplicative updates $\mathbf{u} = \mathbf{p}/(\mathbf{K}\mathbf{v})$? Explain why updating ϕ and ψ directly, with a log-sum-exp in place of the products with $\mathbf{K}$, avoids the problem.

9. code **The dual of ridge regression.** Write $\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{A}\mathbf{w}-\mathbf{b}\|^2+\frac{\lambda}{2}\|\mathbf{w}\|^2$, with $\mathbf{A}\in\mathbb{R}^{n\times d}$, as $\min_{\mathbf{w},\mathbf{z}} \frac{1}{2}\|\mathbf{z}\|^2+\frac{\lambda}{2}\|\mathbf{w}\|^2$ subject to $\mathbf{A}\mathbf{w}-\mathbf{b}-\mathbf{z}=\mathbf{0}$, with multiplier $\alpha\in\mathbb{R}^n$.
   1. Minimize the Lagrangian over $\mathbf{z}$ and $\mathbf{w}$ to get $\mathbf{z}=\alpha$, $\mathbf{w}=-\mathbf{A}^\top\alpha/\lambda$, and the dual function $q(\alpha)=-\frac{1}{2}\|\alpha\|^2-\frac{1}{2\lambda}\alpha^\top\mathbf{A}\mathbf{A}^\top\alpha-\mathbf{b}^\top\alpha$. Why does strong duality hold?
   2. Maximize q to get $\alpha^\star=-\lambda(\mathbf{A}\mathbf{A}^\top+\lambda\mathbf{I})^{-1}\mathbf{b}$ and $\mathbf{w}^\star=\mathbf{A}^\top(\mathbf{A}\mathbf{A}^\top+\lambda\mathbf{I})^{-1}\mathbf{b}$. Explain why the solution is a combination of the rows of $\mathbf{A}$, as the SVM weight vector is a combination of the data points, and why the data enter only through $\mathbf{A}\mathbf{A}^\top$.
   3. For $n=20$ examples and $d=50$ features, verify numerically that the primal solve $(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})\mathbf{w}=\mathbf{A}^\top\mathbf{b}$ and the dual formula agree to about 10^{-14}, and that the primal and dual values coincide. Which solve is cheaper when $n\ll d$? What does replacing $\mathbf{A}\mathbf{A}^\top$ by a kernel matrix give?

10. code **Dual ascent and the gradient of the dual.** For the clipping problem, $q(\lambda)=\frac{25}{2}\frac{\lambda}{1+\lambda}-\frac{\lambda}{2}$ and the Lagrangian's minimizer is $\mathbf{x}(\lambda)=\mathbf{x}_0/(1+\lambda)$.
   1. Show that $q'(\lambda)=g(\mathbf{x}(\lambda))$. Explain why the same holds for any problem whose Lagrangian has a unique minimizer $\mathbf{x}(\lambda)$ that depends smoothly on λ: differentiate $q(\lambda)=\mathcal{L}(\mathbf{x}(\lambda),\lambda)$ and use stationarity to drop the term through $\mathbf{x}$. The gradient of the dual is the vector of constraint values at the Lagrangian's minimizer.
   2. Run projected dual ascent, $\lambda\leftarrow\max(0,\lambda+\eta\,g(\mathbf{x}(\lambda)))$, from $\lambda=0$ with $\eta\in\{0.5,1,2\}$, and report the iterate after 10, 50, and 100 steps. (For $\eta=0.5$ you should see about 5.07, 4.02, and 4.0001.) The first step jumps to 12η, far past $\lambda^\star=4$. Explain this from $q''(\lambda)=-25/(1+\lambda)^3$, which is -25 at $\lambda=0$ and -0.2 at $\lambda=4$. Why does a step size that is safe near $\lambda^\star$ jump so far at the start?
   3. For $\eta=0.5$, are the points $\mathbf{x}(\lambda_k)$ feasible? Show that every feasible $\mathbf{x}(\lambda_k)$ gives an upper bound $f(\mathbf{x}(\lambda_k))\ge p^\star$ while $q(\lambda_k)$ gives a lower bound, and print the certified gap $f(\mathbf{x}(\lambda_k))-q(\lambda_k)$ along the run. Relate dual ascent to the Sinkhorn updates of Exercise 8, which maximize the same kind of dual exactly in blocks.

11. **The chapter's quadratic against a half-plane.** Minimize $\frac{1}{2}(x_1^2+1000\,x_2^2)$ subject to $x_1+x_2\ge 1$.
   1. Write the constraint as $-x_1-x_2\le -1$ and use Equation 35.86 with the conjugate of $\frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ from Exercise 1b to get $q(\lambda)=\lambda-\frac{1}{2}\lambda^2(1+\frac{1}{1000})$. Find $\lambda^\star=\frac{1000}{1001}$, $p^\star=d^\star=\frac{500}{1001}\approx 0.4995$, and $\mathbf{x}^\star=(0.999,0.000999)$.
   2. Why does $x_2^\star$ carry almost none of the constraint? By how much, to first order, does the optimum rise if the constraint becomes $x_1+x_2\ge 1+\delta$?
   3. Replace 1000 by a curvature ratio κ and let $\kappa\to\infty$. What do $\lambda^\star$, $p^\star$, and $\mathbf{x}^\star$ tend to? Fixed-step gradient descent on the unconstrained quadratic slows down by a factor of about κ (Section 35.1). Does the ill-conditioning make this constrained problem hard to solve through its dual?

35.8 Numerical Stability and Conditioning

Every computation in this chapter so far ran in floating point, and we read its results as if they were exact real numbers. Usually that is harmless. Occasionally a training loss turns into `NaN`, or a solver returns an answer that looks reasonable and is wrong. This section asks when finite-precision arithmetic, rather than the algorithm, decides the answer, and what

we can do about it.

Two small computations show the two ways this happens. The three logits $(1000, 1001, 1002)$ define an ordinary softmax, with probabilities $(0.09, 0.24, 0.67)$, yet the formula evaluated as written returns `NaN` three times, even in fp64. And `np.linalg.solve`, applied to a 12×12 linear system, returns an answer that is off by 30%, although it is the exact solution of a system that differs from the stated one in the seventeenth digit. The first failure stops training with a `NaN` loss; the second goes unnoticed.

The two failures have different causes. The softmax formula passes through e^{1000}, which no floating-point format can hold, and an equivalent formula that never forms it is accurate to a few units in the last place. The linear system is so sensitive that a rounding error in the sixteenth digit of its data moves its answer in the first, and no rewriting of the solver can undo that.

We first show what a computer stores and how much error each rounding introduces. We then compute softmax and cross-entropy without passing through values the format cannot hold, and the variance without subtracting nearly equal numbers. A condition number for a function decides whether the algorithm or the problem is to blame. The condition number of a linear system sets how many digits a least-squares solve keeps and, as in Section 35.1, how many steps gradient descent needs. Section 5.5 shows how training uses the low-precision formats; here we derive why computations in them fail.

Three results matter more than the individual formulas: the rounding model with its unit roundoff u, the principle that an unstable algorithm routes a well-conditioned problem through an ill-conditioned step, and the bound of the forward error by the condition number times the backward error. Summation order and ridge regularization can be skipped on a first reading.

```
from fractions import Fraction
```

35.8.1 Floating-Point Arithmetic

35.8.1.1 Representation and Spacing

What does a computer store when it stores 1.1? The answer is easiest to see in a toy format with only two bits after the binary point. Between 1 and 2 it can hold $(1.00)_2 = 1$, $(1.01)_2 = 1.25$, $(1.10)_2 = 1.5$, and $(1.11)_2 = 1.75$. Multiplying by powers of two gives the other intervals: $2, 2.5, 3, 3.5$ between 2 and 4, and $0.5, 0.625, 0.75, 0.875$ between 0.5 and 1. Each *octave*, an interval from 2^e to 2^{e+1}, holds four values, so the gap between neighbors doubles each time the magnitude doubles.

A number that is not on the grid is rounded to the nearest one: $1.1 \mapsto 1$, a relative error of -0.091; $1.3 \mapsto 1.25$, an error of -0.038; $3.3 \mapsto 3.5$, an error of $+0.061$. The midpoint 1.125 is a tie, and ties go to the neighbor whose last bit is 0, here 1 (an error of -0.111). No relative error exceeds $1/8$: rounding moves a number by at most half a gap, and in every octave half a gap is $1/8$ of the octave's smallest number. Figure 35.25 draws this format on the real line.

Real formats work the same way with more bits. A floating-point number is scientific notation in base two with a fixed budget of p binary digits after the point,

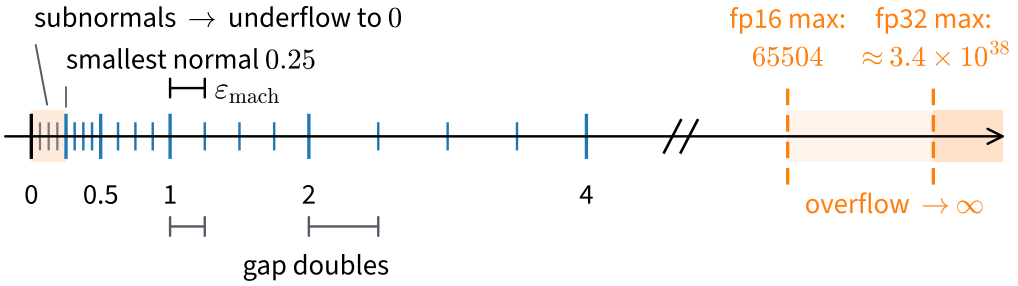


Figure 35.25: Floating-point numbers on the real line, drawn for a toy format with two mantissa bits (four values per octave, $\varepsilon_{\text{mach}} = 1/4$, smallest normal 0.25). The absolute gap between neighbors doubles at every power of two while the relative gap stays near $\varepsilon_{\text{mach}}$; below the smallest normal the subnormals continue with the same gap down to zero; after the axis break, each real format ends at its overflow threshold (fp16 at 65504, fp32 at about 3.4×10^{38}).

$$x = (-1)^s \cdot (1.b_1 b_2 \ldots b_p)_2 \cdot 2^e, \tag{35.95}$$

with a sign bit s, *mantissa* bits $b_1, \ldots, b_p$, and an integer exponent e from a fixed range. The exponent sets the range of magnitudes, and the mantissa sets the relative precision. In the octave $[2^e, 2^{e+1})$ the values are spaced 2^{e-p} apart. The gap from 1 to the next number, $\varepsilon_{\text{mach}} = 2^{-p}$, is called *machine epsilon*; the toy format has $p = 2$ and $\varepsilon_{\text{mach}} = 1/4$.

Rounding to the nearest number moves x by at most half a gap, 2^{e-p-1}, and $|x| \geq 2^e$, so the relative error is at most 2^{-p-1}. A *normal* number is one of the form Equation 35.95 with its exponent in range. For every real x whose rounded value is normal,

$$\text{fl}(x) = x\,(1+\delta), \qquad |\delta| \leq u = \tfrac{1}{2}\,\varepsilon_{\text{mach}} = 2^{-p-1}. \tag{35.96}$$

Each rounding thus perturbs a number by at most one part in 2^{p+1}: one part in 8 for the toy format, about one part in 17 million for fp32 ($p = 23$), and one part in 256 for bf16 ($p = 7$). The bound u is called the *unit roundoff*. IEEE arithmetic rounds the result of each basic operation correctly, so the computed $x \oplus y$ equals $(x + y)(1 + \delta)$ with $|\delta| \leq u$, and the same holds for $-$, $\times$, and $/$ (IEEE 2019; Goldberg 1991).

The model has three exceptions. Results too large to store overflow. Results smaller than the smallest normal number underflow, passing through the *subnormal* numbers that fill the gap to zero. Undefined results, such as $0/0$, become NaN. All three are absolute effects rather than relative ones, and we return to them once the formats are on the table.

35.8.1.2 Formats

Deep learning uses five formats below double precision, fp64. The library reports each one's $\varepsilon_{\text{mach}}$, smallest normal number, and largest finite number:

```
print(f'{"dtype":>14} {"eps":>10} {"smallest normal":>16} {"max":>10}')
for dt in [torch.float16, torch.float32, torch.bfloat16,
           torch.float8_e4m3fn, torch.float8_e5m2]:
    fi = torch.finfo(dt)
    print(f'{str(dt).removeprefix("torch."):>14} {fi.eps:10.3e} '
          f'{fi.smallest_normal:16.3e} {fi.max:10.3e}')
```

```
         dtype        eps  smallest normal        max
       float16  9.766e-04        6.104e-05  6.550e+04
       float32  1.192e-07        1.175e-38  3.403e+38
      bfloat16  7.812e-03        1.175e-38  3.390e+38
  float8_e4m3fn  1.250e-01        1.562e-02  4.480e+02
    float8_e5m2  2.500e-01        6.104e-05  5.734e+04
```

fp32 keeps $p = 23$ bits, about seven decimal digits, and reaches 3.4×10^{38}. **fp16** keeps $p = 10$ bits but has a small exponent range: it overflows above 65504, and below $2^{-14} \approx 6.1 \times 10^{-5}$ its numbers are subnormal. **bf16** (`bfloat16` in the libraries) makes the opposite choice. It keeps fp32's eight exponent bits, and hence its range, and only $p = 7$ mantissa bits, $\varepsilon_{\text{mach}} = 2^{-7} \approx 0.0078$, two to three decimal digits. The value 2^{-8} that is often quoted for bf16 is its unit roundoff $u = \varepsilon_{\text{mach}}/2$. The two fp8 formats continue the progression: E4M3 keeps three bits (about one decimal digit) and stops at 448, and E5M2 keeps two bits with fp16's range. With so little range, fp8 is usable only with a scale factor per tensor or per block of values.

bf16 is the top half of an fp32 number: the same sign bit, the same eight exponent bits, and the first seven mantissa bits. Rounding an fp32 number to bf16 therefore means rounding away its low 16 bits, which we can write as integer arithmetic on the bit pattern. We add `0x7FFF`, plus one more when the lowest kept bit is 1, and then clear the low 16 bits. The addition carries into the kept bits when the discarded part is above the midpoint, or at the midpoint with an odd kept part, and only then. This is round to nearest, ties to even, written as bit truncation; Exercise 7 builds stochastic rounding on it. The cell checks one input above a midpoint and two ties:

```
def to_bf16(x):
    """Round float32 to the nearest bfloat16, ties to even."""
    bits = np.atleast_1d(np.asarray(x, np.float32)).view(np.uint32)
    bits = (bits + 0x7FFF + ((bits >> 16) & 1)) & 0xFFFF0000
    return bits.astype(np.uint32).view(np.float32)

for x, expected in [(1 + 2**-8 + 2**-10, 1 + 2**-7),   # above the midpoint
                    (1 + 2**-8, 1.0),                  # tie, 1 is even
                    (1 + 3 * 2**-8, 1 + 2**-6)]:       # tie, 1 + 2^-6 is even
    print(f'{x:.8f} -> {to_bf16(x).item():.8f}   (expected {expected:.8f})')
```

```
1.00488281 -> 1.00781250   (expected 1.00781250)
1.00390625 -> 1.00000000   (expected 1.00000000)
1.01171875 -> 1.01562500   (expected 1.01562500)
```

The first input lies above the midpoint between 1 and $1 + 2^{-7}$ and rounds up. The second is that midpoint and rounds to the even neighbor 1; the third is the midpoint between $1 + 2^{-7}$ and $1 + 2^{-6}$ and rounds up to the even neighbor. The next cell shows the same tie rule in fp32, the growth of the gap with the magnitude, and the largest logit each format can

exponentiate:

```
eps = np.finfo(np.float32).eps
one = np.float32(1.0)
print('1 + eps   != 1 :', one + eps != one)
print('1 + eps/2 == 1 :', one + np.float32(eps / 2) == one)
print('gap between float32 neighbors at 1    :', np.spacing(one))
print('gap between float32 neighbors at 2^20 :', np.spacing(np.float32(2.0**20)))
for dt in [np.float16, np.float32, np.float64]:
    print(f'{np.dtype(dt).name}: exp(x) overflows for x >',
          f'{np.log(np.finfo(dt).max):.2f}')
```

```
1 + eps   != 1 : True
1 + eps/2 == 1 : True
gap between float32 neighbors at 1    : 1.1920929e-07
gap between float32 neighbors at 2^20 : 0.125
float16: exp(x) overflows for x > 11.09
float32: exp(x) overflows for x > 88.72
float64: exp(x) overflows for x > 709.78
```

The value $1 + \varepsilon_{\text{mach}}/2$ is the midpoint between 1 and its upper neighbor and rounds to the even one, 1. Half a gap, the worst case, does occur; as a relative error it is $u/(1+u)$, just inside the bound $|\delta| \le u$. The gap at 2^{20} is 2^{20} times the gap at 1, about a million times. The last three lines locate the logits at which e^x overflows: 11.09 in fp16, 88.72 in fp32, and 709.78 in fp64.

35.8.1.3 Overflow, Underflow, and Small Updates

The exponential turns additive scale into multiplicative scale, so a modest logit already overflows: fp16's threshold $\log 65504 \approx 11.09$ is well inside the range of ordinary unnormalized scores.

At the other end, e^{-x} underflows. Below the smallest normal number the format continues through subnormal numbers, which keep the last gap but have fewer and fewer significant bits, and then reaches 0. A later log of that 0 returns $-\infty$, and the backward pass returns `inf` or `NaN`. In fp16 the subnormals start below $2^{-14} \approx 6.1 \times 10^{-5}$, the smallest of them is $2^{-24} \approx 6 \times 10^{-8}$, and everything below $2^{-25} \approx 3 \times 10^{-8}$ rounds to zero.

Addition has a second absolute threshold. A training step replaces a weight w by $w - \eta g$. If the step ηg is small compared with the gap next to w, the sum rounds back to w, and the step is lost.

Proposition (small updates vanish). *Let $w > 0$ be a floating-point number with $2^e \le w < 2^{e+1}$ in a format with p mantissa bits, and let Δ be real. If $|\Delta| < 2^{e-p-1}$, then $\mathrm{fl}(w + \Delta) = w$, except when $w = 2^e$ and $\Delta < 0$; in that case the same holds for $|\Delta| < 2^{e-p-2}$.*

Proof. In $[2^e, 2^{e+1}]$ the representable numbers are 2^{e-p} apart, so the neighbors of w are $w \pm 2^{e-p}$, except that below $w = 2^e$ the octave $[2^{e-1}, 2^e)$ begins and the gap halves to 2^{e-p-1}. Rounding to the nearest number returns w whenever $w + \Delta$ is closer to w than to the neighbor on its side, that is, whenever $|\Delta|$ is less than half of that gap. ■

For a weight update, $\Delta = -\eta g$, and for $2^e < w < 2^{e+1}$ the proposition says

$$\mathrm{fl}(w - \eta g) = w \qquad \text{whenever } |\eta g| < 2^{e-p-1} = u\,2^{e}. \tag{35.97}$$

An update smaller than about $u|w|$ is lost. The relative model Equation 35.96 is accurate for products and quotients, but for a sum of two numbers of very different size it hides this absolute threshold.

A weight $w = 1$ sits on a power of two, the exceptional case of the proposition: below it the gap halves, so a decrease is lost below $u/2$ (an increase, which Section 5.5 tries, below u). In bf16 a weight equal to 1 thus ignores any decrease below $2^{-9} \approx 0.002$, and a learning rate of 10^{-3} with a gradient of order 1 leaves it unchanged. The same threshold is $2^{-12} \approx 2.4\times10^{-4}$ in fp16 and $2^{-25} \approx 3\times10^{-8}$ in fp32, which is why mixed-precision training keeps a master copy of the weights in fp32. The cell tries steps on either side of the thresholds; the bf16 lines compute the step in fp32 and round the result with `to_bf16`:

```
one32, one16 = np.float32(1.0), np.float16(1.0)
print('bfloat16: 1 - 1e-3 ->', to_bf16(one32 - np.float32(1e-3)).item())
print('bfloat16: 1 - 2e-3 ->', to_bf16(one32 - np.float32(2e-3)).item())
print('fp16    : 1 - 1e-4 ->', one16 - np.float16(1e-4))
print('fp16    : 1 - 3e-4 ->', one16 - np.float16(3e-4))
print('fp32    : 1 - 1e-3 ->', one32 - np.float32(1e-3))
```

```
bfloat16: 1 - 1e-3 -> 1.0
bfloat16: 1 - 2e-3 -> 0.99609375
fp16    : 1 - 1e-4 -> 1.0
fp16    : 1 - 3e-4 -> 0.9995
fp32    : 1 - 1e-3 -> 0.999
```

A step of 10^{-3} leaves the bf16 weight at 1, while a step of 2×10^{-3}, above the threshold 0.00195, moves it to the neighbor $1 - 2^{-8} = 0.99609375$. fp16 loses 10^{-4} and keeps 3×10^{-4}, which lands on its neighbor $1 - 2^{-11} \approx 0.9995$; fp32 keeps the full step.

Gradients meet the underflow threshold instead. fp16 gradients below 6.1×10^{-5} lose bits and those below 3×10^{-8} become zero. Multiplying the loss by a factor S before the backward pass multiplies every gradient by S and moves them back into range; dividing by S in fp32 afterward undoes the scaling (Section 5.5.3.3 shows the practice, and Exercise 1 finds the admissible S).

Stochastic rounding (Gupta et al. 2015) removes the lost-update threshold in a different way. It rounds up or down at random, with probabilities chosen so that the expected stored value equals the exact result, so updates too small for round-to-nearest still accumulate in expectation (Exercise 7).

35.8.2 Stable Softmax and Cross-Entropy

Softmax, log-sum-exp, and cross-entropy all exponentiate logits, so all three meet the overflow and underflow thresholds above. Our running example for them is a classifier with two classes and logits $\mathbf{z} = (0, t)$, whose label is the unlikely class 0. Its probabilities are $\mathrm{softmax}(\mathbf{z}) = \big(1/(1+e^{t}),\, e^{t}/(1+e^{t})\big)$, its log-sum-exp is $\log(1+e^{t})$, and the cross-entropy of class 0 is $\log(1 + e^{t}) \approx t$ for large t. None of these numbers is extreme: at $t = 100$ the probabilities are 3.7×10^{-44} and 1, and the loss is 100.

35.8.2.1 The Max Shift

The softmax of logits $\mathbf{z} \in \mathbb{R}^n$ is

$$\operatorname{softmax}(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}}.$$

At $t = 100$ the direct formula forms e^{100}, above fp32's threshold 88.72. The second entry becomes `inf/inf`, which is `NaN`, and the first becomes $1/\infty = 0$. The probabilities themselves are representable; only the intermediate exponential is not. Multiplying numerator and denominator by the positive constant e^{-c} changes nothing, so softmax is *shift-invariant*,

$$\operatorname{softmax}(\mathbf{z} - c\mathbf{1}) = \operatorname{softmax}(\mathbf{z}) \qquad \text{for every } c \in \mathbb{R}. \tag{35.98}$$

The same factoring under the logarithm rewrites the **log-sum-exp** $\operatorname{lse}(\mathbf{z}) = \log \sum_j e^{z_j}$, the logarithm of the softmax denominator:

$$\operatorname{lse}(\mathbf{z}) = c + \log \sum_{j=1}^{n} e^{z_j - c}. \tag{35.99}$$

Both identities hold for every c, and the choice $c = \max_j z_j$ makes the computation safe. Every exponent $z_j - c$ is then at most 0, so no term overflows, and the largest term equals 1, so the sum lies in $[1, n]$ and its logarithm cannot be $-\infty$. At $t = 100$ the direct formula should fail and the shifted one should not. At $t = 10$ both are finite; do they return the same bits?

```
def softmax_naive(z):
    e = np.exp(z)
    return e / e.sum()

def softmax_stable(z):
    e = np.exp(z - z.max())        # shift by the max: largest exponent is 0
    return e / e.sum()

z = np.array([0.0, 100.0], dtype=np.float32)
with np.errstate(over='ignore', invalid='ignore'):
    print('naive  softmax(0, 100):', softmax_naive(z))
print('stable softmax(0, 100):', softmax_stable(z))
z = np.array([1000.0, 1001.0, 1002.0])        # the opening's logits, fp64
with np.errstate(over='ignore', invalid='ignore'):
    print('naive  softmax(1000, 1001, 1002):', softmax_naive(z))
print('stable softmax(1000, 1001, 1002):', softmax_stable(z).round(4))
z = np.array([0.0, 10.0], dtype=np.float32)
a, b = softmax_naive(z), softmax_stable(z)
print('softmax(0, 10), naive minus stable, in ulps:',
      (a - b) / np.spacing(b))
```

```
naive  softmax(0, 100): [ 0. nan]
stable softmax(0, 100): [3.8e-44 1.0e+00]
```

```
naive  softmax(1000, 1001, 1002): [nan nan nan]
stable softmax(1000, 1001, 1002): [0.09   0.2447 0.6652]
softmax(0, 10), naive minus stable, in ulps: [2. 0.]
```

The shifted softmax returns $(3.8\times10^{-44}, 1)$, the exact 3.7×10^{-44} rounded to the subnormal grid with five significant bits left; the direct formula returns 0 and `NaN`. More precision is no cure. The logits of the opening fail even in fp64, because e^{1000} overflows there too: the direct formula returns three `NaN` where the shifted one returns $(0.09, 0.24, 0.67)$. At $t = 10$ the two agree to the last bit or differ by one or two units in the last place (*ulps*): the shift identity holds for real numbers, and the two orders of evaluation round differently. The shift does not handle non-finite logits: a row of all $-\infty$ has $c = -\infty$, and $z_j - c$ is undefined.

35.8.2.2 Bounds for Log-Sum-Exp

Log-sum-exp is the normalizer of every exponential-family model (Section 36.4), and it is the cross-entropy loss up to one subtraction. The shift Equation 35.99 also says how far it is from the largest logit:

Proposition (log-sum-exp sandwich). *For every* $\mathbf{z} \in \mathbb{R}^n$,

$$\max_j z_j \;\le\; \operatorname{lse}(\mathbf{z}) \;\le\; \max_j z_j + \log n.$$

Proof. Put $c = \max_j z_j$ in Equation 35.99. Every term $e^{z_j - c} \le 1$ and the maximizing term equals 1, so the sum lies in $[1, n]$ and its logarithm in $[0, \log n]$. Adding c gives both inequalities. ■

Log-sum-exp is therefore a *soft maximum*. On the running example, $\operatorname{lse}(0, t) = \log(1 + e^t)$ stays within $\log 2 \approx 0.69$ of $\max(0, t)$, with the full gap at $t = 0$, where the two logits tie (Figure 35.26).

Its gradient is the softmax, $\partial \operatorname{lse}(\mathbf{z})/\partial z_i = e^{z_i}/\sum_j e^{z_j} = \operatorname{softmax}(\mathbf{z})_i$, which is why the gradient of the cross-entropy below, for label y, is $\operatorname{softmax}(\mathbf{z}) - \mathbf{e}_y$; Section 35.3 proves that log-sum-exp is convex.

Subtracting the shifted log-sum-exp from the shifted logit gives the **log-softmax**, the logarithm of a probability computed without forming the probability:

$$\log \operatorname{softmax}(\mathbf{z})_i = (z_i - c) - \log \sum_{j=1}^{n} e^{z_j - c}, \qquad c = \max_j z_j. \tag{35.100}$$

A tiny probability then never underflows to zero before its logarithm is taken. The equivalent form $z_i - \operatorname{lse}(\mathbf{z})$ is worse in floating point: it stores the large number $\operatorname{lse} = c + \log \sum_j e^{z_j - c}$ and then subtracts it from the large number z_i. How much worse? The cell computes both forms in fp32 for the logits $(1000, 1001, 1002)$ of the introduction:

```python
z = np.array([1000.0, 1001.0, 1002.0], dtype=np.float32)
c = z.max()
s = np.log(np.exp(z - c).sum())          # log sum exp(z - c), in [0, log 3]
with np.errstate(over='ignore'):
    print('naive log(sum(exp(z))):', np.log(np.exp(z).sum()))
```

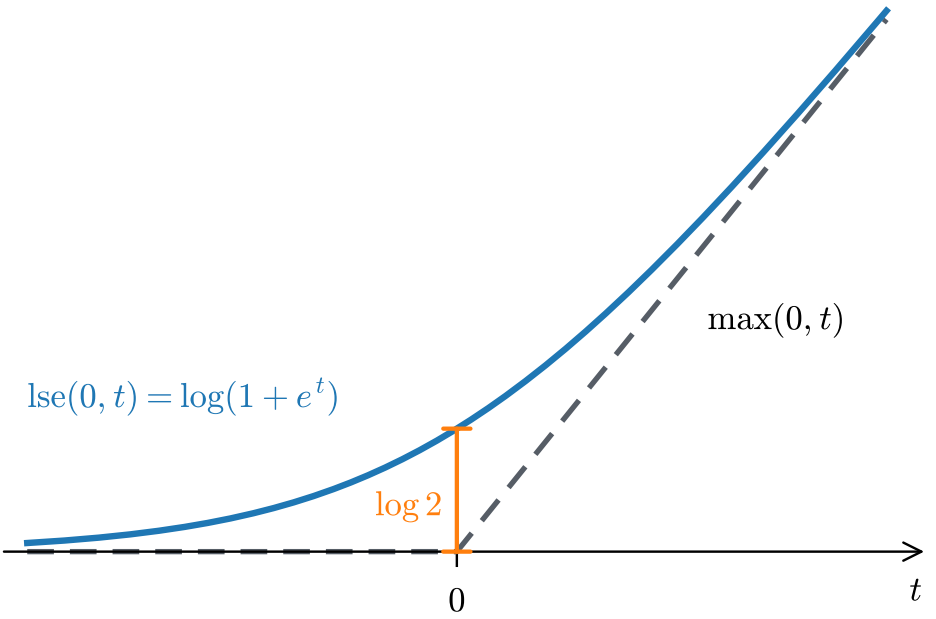


Figure 35.26: Log-sum-exp is a soft maximum: $\mathrm{lse}(0, t) = \log(1 + e^t)$ stays within $\log 2$ of $\max(0, t)$, the sandwich bound with $n = 2$. The gap is largest at $t = 0$, where the two logits tie, and vanishes as $|t|$ grows.

```
lse = c + s
print('lse = c + s           :', lse, '  float32 gap at 1002:', np.spacing(c))
log_p_sub = z - lse                      # forms lse, then subtracts it
log_p = (z - c) - s                      # never forms c + s
print(f'sum of exp(z - lse)    : {np.exp(log_p_sub).sum():.8f}')
print(f'sum of exp((z - c) - s): {np.exp(log_p).sum():.8f}')
```

```
naive log(sum(exp(z))): inf
lse = c + s           : 1002.4076   float32 gap at 1002: 6.1035156e-05
sum of exp(z - lse)    : 1.00001323
sum of exp((z - c) - s): 1.00000000
```

The direct computation overflows to `inf`, and it would in fp64 too, because $1000 > 709.78$; dividing `inf` by `inf` is what gave the three `NaN` of the introduction. The stable value $\mathrm{lse} = 1002.4076$ lies in the sandwich $[1002, 1002 + \log 3]$. The two log-softmax forms differ: the probabilities from $z_i - \mathrm{lse}$ sum to 1.0000132, those from Equation 35.100 to 1.0000000.

The subtraction $z_i - \mathrm{lse}$ is exact, because two floating-point numbers within a factor of two of each other have an exactly representable difference: both are multiples of the smaller one's gap, and their difference is too small to need more bits (Sterbenz's lemma, Exercise 8). The error is in lse itself. Near 1002, fp32 numbers are $2^{-14} \approx 6.1 \times 10^{-5}$ apart, and the true value 1002.4076060 is stored 1.32×10^{-5} too low. Every log-probability is then 1.32×10^{-5} too high, every probability is too large by the factor $e^{1.32 \times 10^{-5}}$, and the sum exceeds 1 by 1.32×10^{-5}. An absolute error in an exponent becomes a relative error of the

same size in its exponential, and the subtraction exposed an error that was already in its input, the theme of Section 35.8.3.

35.8.2.3 Cross-Entropy from Logits

The cross-entropy loss of label y is the negative log-softmax, so Equation 35.100 gives it directly from the logits,

$$-\log \operatorname{softmax}(\mathbf{z})_y = \operatorname{lse}(\mathbf{z}) - z_y. \tag{35.101}$$

The loss needs one stable log-sum-exp and one subtraction, and no probability. On the running example, the loss of class 0 is $\operatorname{lse}(0, t) - 0 = \log(1 + e^t)$. If the probabilities are formed first, the true-class probability $e^{-t}/(1+e^{-t}) \approx e^{-t}$ leaves fp32's normal range once $t > 87.3$ and reaches zero near $t = 104$, and its negative logarithm becomes infinite. The cell computes the loss both ways with the library's functions; its last two values of t, 103 and 104, straddle that threshold:

```
print('    t  CE from logits    CE via probabilities')
for t in [20.0, 60.0, 103.0, 104.0]:
    logits = torch.tensor([[0.0, t]])
    y = torch.tensor([0])                       # label = the unlikely class
    from_logits = F.cross_entropy(logits, y)
    probs = F.softmax(logits, dim=1)            # stable softmax, then...
    from_probs = -torch.log(probs[0, y])        # ...take the log yourself
    print(f'{t:5.0f}  {from_logits.item():15.4f}  {from_probs.item():15.4f}')
```

```
    t  CE from logits    CE via probabilities
   20          20.0000          20.0000
   60          60.0000          60.0000
  103         103.0000         103.2789
  104         104.0000              inf
```

Computed from the logits, the loss is t in every row, and the probability route agrees at $t = 20$ and 60. At $t = 103$ the probability e^{-103} is a subnormal number, and what the second column shows depends on how the library treats subnormals.

PyTorch keeps them: $e^{-103} \approx 1.32 \times 2^{-149}$ rounds to the smallest fp32 subnormal, 2^{-149}. One significant bit survives, and the loss reads $149 \log 2 = 103.2789$, wrong in the first decimal place and with no warning. At $t = 104$, $e^{-104} < 2^{-150}$ rounds to 0, and the loss is `inf`.

Losses, likelihoods, and posteriors should therefore stay in log space until a probability is needed. The fused loss of Section 3.4.2 computes Equation 35.101, and Section 37.1 explains what the cross-entropy measures. Section 36.11 adds hundreds of per-pixel log probabilities for the same reason.

35.8.3 Catastrophic Cancellation

35.8.3.1 The Condition Number of a Function

In fp32, $\log(1 + x)$ at $x = 10^{-8}$ evaluates to 0, a relative error of 100%, while the library function `log1p` returns 10^{-8}. Is $\log(1 + x)$ near $x = 0$ a hard function to compute, or is the

formula to blame? We need a number that says how much a function amplifies an error in its input, independently of how the function is computed.

The **relative condition number** of a differentiable function f at a point x with $f(x) \neq 0$ is the factor by which a small relative change in x becomes a relative change in $f(x)$. A relative change δ of the input gives, to first order, $f(x(1+\delta)) \approx f(x) + x f'(x)\,\delta$; dividing by $f(x)$ gives the factor

$$\kappa_f(x) = \left| \frac{x\, f'(x)}{f(x)} \right|. \tag{35.102}$$

The exponential has $\kappa_f = |x|$: at $x = 88$ it turns an input error of one unit roundoff into 88 of them in the output. This is the mechanism seen in the log-softmax cell, where an absolute error in an exponent became the same relative error in a probability. The condition number is a first-order statement about the exact function. It says nothing about how f is computed, and it fails where $f(x) = 0$, where no relative error is defined.

Subtraction has two inputs, and the same reasoning applies to both. Let $\hat{a} = a(1+\delta_a)$ and $\hat{b} = b(1+\delta_b)$ carry relative errors $|\delta_a|, |\delta_b| \leq u$, and let the subtraction itself be exact. Then $\hat{a} - \hat{b} = (a-b) + (a\delta_a - b\delta_b)$, and the triangle inequality bounds the relative error of the difference:

$$\frac{|(\hat{a}-\hat{b}) - (a-b)|}{|a-b|} = \frac{|a\delta_a - b\delta_b|}{|a-b|} \leq \frac{|a|+|b|}{|a-b|}\, u. \tag{35.103}$$

The factor $(|a|+|b|)/|a-b|$ is subtraction's condition number, and it is large when $a \approx b$. Subtracting nearly equal numbers that already carry errors is called **catastrophic cancellation**: the leading digits cancel, and the errors in the trailing digits become the leading digits of the result. For reference, here are the condition numbers of a few common problems:

problem	κ_f	regime
$a - b$	$(\lvert a\rvert + \lvert b\rvert)/\lvert a-b\rvert$	large when $a \approx b$
$\log(1+x)$	$x/((1+x)\log(1+x))$	1.000 at $x = 10^{-8}$, 0.82 at $x = 0.5$
e^x	$\lvert x\rvert$	large for large $\lvert x\rvert$
$1 - \cos x$	$x \sin x/(1-\cos x)$	tends to 2 as $x \to 0$
$\sqrt{x}$	$1/2$	always
variance of data with mean m, spread s	at most $2(1 + \lvert m\rvert/s)$	derived below

The second row answers the opening question: $\log(1+x)$ near $x = 0$ is a well-conditioned problem. The algorithm `log(1 + x)` is not, because it passes through an ill-conditioned step. It first forms $y = 1 + x$ and then evaluates $\log y$ at $y \approx 1$, where the logarithm has $\kappa_{\log}(y) = 1/|\log y| \approx 1/x = 10^8$. Rounding $1 + x$ introduces a relative error of u in y, which becomes a relative error of up to $10^8 u$, more than 100%, in the result. `log1p` is written so that the rounding error of $1 + x$ never reaches the result; for tiny x it returns x itself.

This is the principle on which this section is built: *an unstable algorithm routes a well-conditioned problem through an ill-conditioned intermediate step.* When κ_f is small and the computed result is bad, change the algorithm. When κ_f itself is large, no algorithm helps and only changing the problem does, the subject of Section 35.8.4.

35.8.3.2 Subtraction of Nearly Equal Numbers

The cell repeats the $\log(1+x)$ experiment and then subtracts two fp32 numbers that agree in their first seven digits, 1.0002344 and 1.0002341, whose true difference is 3.0×10^{-7}:

```
x = np.float32(1e-8)
print('float32 rounds 1 + x to:', np.float32(1.0) + x)
print('log(1 + x) =', np.log(np.float32(1.0) + x), '   log1p(x) =', np.log1p(x))
a, b = np.float32(1.0002344), np.float32(1.0002341)
print(f'stored a = {float(a):.10f}, b = {float(b):.10f}')
print('a - b =', a - b, '=', (a - b) / np.spacing(np.float32(1.0)), 'ulps of 1')
print('float64: log(1 + 1e-20) =', np.log(1 + 1e-20),
      '  log1p(1e-20) =', np.log1p(1e-20))
```

```
float32 rounds 1 + x to: 1.0
log(1 + x) = 0.0    log1p(x) = 1e-08
stored a = 1.0002343655, b = 1.0002341270
a - b = 2.3841858e-07 = 2.0 ulps of 1
float64: log(1 + 1e-20) = 0.0   log1p(1e-20) = 1e-20
```

Between 1 and 2, fp32 differences can only be whole multiples of $\varepsilon_{\text{mach}} = 1.19 \times 10^{-7}$, and the true difference 3.0×10^{-7} is not one. The subtraction returned two ulps, 2.38×10^{-7}, 20% too small, and it made no error of its own: the inputs were stored as 1.0002343655 and 1.0002341270, and their difference is what the cell printed. The error was in the stored inputs. The factor Equation 35.103 is $2.0005/(3 \times 10^{-7}) \approx 7 \times 10^{6}$, which times $u \approx 6 \times 10^{-8}$ allows a relative error of up to 0.4; the observed 0.2 is inside the bound.

Cancellation is common in $\log(1+x)$ and $e^x - 1$ near 0 (use `log1p` and `expm1`), in $1 - \cos x$ near 0 (use $2\sin^2(x/2)$), and in the quadratic formula near a double root (Exercise 2). In each of these a reformulation removes the subtraction.

Finite differences are the one case where the cure is a compromise. A gradient check compares automatic differentiation with the forward difference $(f(x+h) - f(x))/h$. How small should h be? The truncation error is about $\frac{h}{2}|f''(x)|$, by Taylor's theorem. Each of the two function values carries a rounding error of about $u|f(x)|$, and the division by h turns them into an error of about $2u|f(x)|/h$. The sum of the two errors is smallest where they are equal, at

$$h^\star = 2\sqrt{\frac{u\,|f(x)|}{|f''(x)|}}, \qquad \text{with error about } 2\sqrt{u\,|f(x)f''(x)|}.$$

The best forward difference keeps only about half of the format's digits. For $f = \sin$ at $x = 1$ in fp64, $h^\star \approx 2 \times 10^{-8}$, with predicted error 1.8×10^{-8}; in fp32, $h^\star \approx 5 \times 10^{-4}$. The cell measures the error at steps from 10^{-2} to 10^{-12} in both formats.

```
print('      h   error in fp64   error in fp32')
for h in 10.0 ** -np.arange(2, 13):
    err = [abs(float((np.sin(t(1) + t(h)) - np.sin(t(1))) / t(h)) - np.cos(1.0))
           for t in (np.float64, np.float32)]  # forward difference of sin at 1
    print(f'{h:7.0e}  {err[0]:14.1e}  {err[1]:14.1e}')
```

```
      h   error in fp64   error in fp32
1e-02          4.2e-03          4.2e-03
1e-03          4.2e-04          4.0e-04
1e-04          4.2e-05          2.8e-04
1e-05          4.2e-06          3.9e-03
1e-06          4.2e-07          6.3e-02
1e-07          4.2e-08          5.6e-02
1e-08          3.0e-09          5.4e-01
1e-09          5.3e-08          5.4e-01
1e-10          5.8e-08          5.4e-01
1e-11          1.2e-06          5.4e-01
1e-12          4.3e-05          5.4e-01
```

In fp64 the error falls in proportion to h down to about $h = 10^{-8}$, where it is of order 10^{-8} or below (the exact value at a single h is rounding noise), and rises again to 4×10^{-5} at $h = 10^{-12}$ (too much cancellation); at $h = 10^{-2}$ it is 4×10^{-3} (too much truncation). In fp32 the best steps, 10^{-3} and 10^{-4}, leave about three digits, and from $h = 10^{-8}$ down the sum $1 + h$ rounds to 1, so the difference quotient is 0 and the error is the whole derivative, $\cos 1 = 0.54$. This is why gradient checks are run in fp64.

Higher precision does not remove a cancellation; it moves the input at which the failure starts. The last line of the $\log(1+x)$ cell shows $\log(1+x)$ returning 0 in fp64 at $x = 10^{-20}$, just as fp32 did at 10^{-8}, while `log1p` is accurate in both. Reformulation removes the ill-conditioned step itself (Higham 2002).

35.8.3.3 One-Pass Variance

Cancellation also hides inside formulas that look harmless. The variance is the standard example. Normalization layers compute it at every step, and so does any code that standardizes data.

The variance of data $x_1, \dots, x_N$ with mean m and standard deviation s can be computed in one pass from the identity

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - m)^2 = \frac{1}{N}\sum_{i=1}^{N} x_i^2 - m^2,$$

which needs only the running sums of x_i and x_i^2. The identity is exact in real arithmetic and unstable in floating point. When $s \ll |m|$, both terms on the right are close to m^2 while their difference is s^2, so the cancellation factor Equation 35.103 is about $2m^2/s^2$. With $m = 10^9$ and $s = 1$ that is 2×10^{18}, more than the 10^{16} that fp64 can absorb.

How much of this does the problem itself force? Perturb each x_i by a relative error δ_i with $|\delta_i| \le u$, so that $x_i \mapsto x_i + x_i\delta_i$. The derivative of the variance with respect to x_i is

$\frac{2}{N}(x_i - m)$, because the change of m multiplies $\sum_i (x_i - m) = 0$. To first order the variance therefore changes by

$$\frac{2}{N}\sum_{i=1}^{N}(x_i - m)\,x_i\delta_i = \frac{2}{N}\sum_{i=1}^{N}(x_i - m)^2\delta_i + \frac{2m}{N}\sum_{i=1}^{N}(x_i - m)\,\delta_i,$$

where we wrote $x_i = (x_i - m) + m$. The first sum is at most $2s^2u$ in magnitude. The second is at most $2|m|u \cdot \frac{1}{N}\sum_i |x_i - m| \le 2|m|s\,u$, by the Cauchy–Schwarz inequality. Relative to s^2, the condition number of the variance is at most $2(1 + |m|/s) \approx 2|m|/s$, about 2×10^9 here (perturbing every x_i by $\delta_i = u\,\mathrm{sign}(x_i - m)$ gives $1.6\,|m|/s$ for Gaussian data). The problem allows an error of about $2 \times 10^9 u \approx 2 \times 10^{-7}$, seven digits. The one-pass formula behaves like $(m/s)^2 u$: it *squares* the condition number, as the normal equations will square the condition number of a matrix in Section 35.8.4.

A better one-pass algorithm keeps the deviations at the scale s. Welford (1962) updates a running mean m_k and the running sum of *centered* squares $M_k = \sum_{i \le k}(x_i - m_k)^2$ with the deviation $d_k = x_k - m_{k-1}$ of each new value from the old mean:

$$m_k = m_{k-1} + \frac{d_k}{k}, \qquad M_k = M_{k-1} + d_k\,(x_k - m_k). \tag{35.104}$$

The update of M_k multiplies two different deviations, from the old mean and from the new mean. That asymmetry makes the recursion correct:

Proposition (Welford's recursion). *With $m_0 = M_0 = 0$, the recursions Equation 35.104 satisfy, for every $k \ge 1$ and in exact arithmetic,*

$$m_k = \frac{1}{k}\sum_{i=1}^{k} x_i, \qquad M_k = \sum_{i=1}^{k}(x_i - m_k)^2.$$

Proof. The mean claim is the identity $k\,m_k = (k-1)\,m_{k-1} + x_k$, immediate from the first recursion. For the second claim, induct on k. Since $m_k - m_{k-1} = d_k/k$ and $x_k - m_k = d_k\,(k-1)/k$, splitting the new sum of squares at its last term and centering the first $k-1$ terms at m_{k-1} gives

$$\sum_{i=1}^{k}(x_i - m_k)^2 = \sum_{i=1}^{k-1}\Big((x_i - m_{k-1}) - \tfrac{d_k}{k}\Big)^2 + (x_k - m_k)^2 = M_{k-1} + (k-1)\tfrac{d_k^2}{k^2} + \tfrac{(k-1)^2}{k^2}d_k^2.$$

The cross term vanished because $\sum_{i \le k-1}(x_i - m_{k-1}) = 0$, and the inductive hypothesis named the first sum M_{k-1}. The two correction terms combine to $\frac{k-1}{k}\,d_k^2 = d_k \cdot d_k\frac{k-1}{k} = d_k(x_k - m_k)$, which is what the recursion adds. ∎

Why the recursion is also accurate in floating point is the hard step, and we argue it only informally; Higham (2002) gives the analysis. Welford's recursion does subtract large numbers: x_k and m_{k-1} are both near 10^9. But that subtraction is exact, by Sterbenz's lemma again, and x_k is exact data, so the only error in d_k is that of the running mean, some

tens of gaps at $|m|$ in a run of 10^5 steps. Every quantity that enters M_k is therefore a deviation of size s, and the rounding errors of the updates enter relative to s^2. Only the error of the center is inherited from the scale m, and a constant error ε of the center changes a centered sum of squares only by $N\varepsilon^2$, because the first-order term multiplies $\sum_i (x_i - m) = 0$. In practice the result is accurate to about the problem's own amplification, $u|m|/s \approx 1.1 \times 10^{-7}$, where the one-pass formula pays $(m/s)^2 u \approx 100$.

The cell runs the three methods on 10^5 fp64 samples with mean 10^9 and variance 1:

```
rng = np.random.default_rng()
x = 1e9 + rng.normal(0.0, 1.0, size=100_000)    # mean 1e9, variance 1

naive = (x**2).mean() - x.mean()**2             # one pass, cancels
two_pass = ((x - x.mean())**2).mean()           # subtract the mean first

m, M = 0.0, 0.0                                 # Welford: one pass
for k, xk in enumerate(x, start=1):
    d = xk - m
    m += d / k                                  # m_k
    M += d * (xk - m)                           # M_k
welford = M / len(x)

print(f'naive E[x^2] - E[x]^2 : {naive:12.6f}   (= {naive / 128:.0f} x 128)')
print(f'Welford, one pass     : {welford:12.6f}')
print(f'two-pass reference    : {two_pass:12.6f}')
print(f'Welford vs two-pass   : {abs(welford - two_pass) / two_pass:.1e} relative')
print('float64 gap near 1e18 :', np.spacing(1e18))
```

```
naive E[x^2] - E[x]^2 :  -128.000000   (= -1 x 128)
Welford, one pass     :     0.999287
two-pass reference    :     0.999287
Welford vs two-pass   : 6.8e-09 relative
float64 gap near 1e18 : 128.0
```

The naive result is a whole multiple of 128: typically 0, ± 128, or ± 256, never near the true value 1, and sometimes negative. Both terms of the formula lie in $[2^{59}, 2^{60})$, where fp64 numbers are $2^{59-52} = 128$ apart, so their difference can only be a multiple of 128: the predicted error of about 100, made discrete. Welford's estimate agrees with the two-pass reference to seven digits or more, and often to eight or nine, because the rounding errors of the updates have both signs and partly cancel in the sum.

The same care applies wherever a variance is accumulated in one pass, for instance to streaming statistics over a data set. (The running moments that batch normalization uses at test time are exponential moving averages, Section 7.3, which do not subtract large sums.)

35.8.3.4 Summation Order

The naive variance and the two-pass reference both add up 10^5 terms, and the order of the additions changes their rounding errors. Each addition contributes one factor $(1+\delta_k)$, and in a left-to-right sum the first term passes through all $n-1$ of them. To first order in u, the sum $\mathrm{fl}(\sum_i x_i)$ computed left to right satisfies

$$\left|\mathrm{fl}\Big(\sum_{i=1}^{n} x_i\Big) - \sum_{i=1}^{n} x_i\right| \le (n-1)\,u \sum_{i=1}^{n} |x_i|.$$

The bound grows with the number of terms, because early terms collect the rounding errors of every later addition. *Pairwise* summation adds the two halves of the list recursively, so each term passes through only $\log_2 n$ additions and the bound becomes $u \log_2 n \sum_i |x_i|$; NumPy's `sum` uses a blocked version of it. Kahan's compensated summation carries the rounding error of each addition in a second variable and achieves an error bound independent of n (Exercise 10). The cell adds 10^7 copies of fp32's 0.1, which is 0.100000001490, in order and pairwise. The pairwise sum pads the list with zeros to 2^{24} terms and adds neighbors until one number is left:

```
x = np.full(10_000_000, 0.1, dtype=np.float32)
exact = len(x) * float(x[0])                  # x[0] = 0.100000001490...
s = np.concatenate([x, np.zeros(2**24 - len(x), np.float32)])  # pad to 2^24
while len(s) > 1:                             # pairwise: add neighbors, halve
    s = s[0::2] + s[1::2]
print(f'in order (cumsum): {np.cumsum(x)[-1]:16.4f}')
print(f'pairwise         : {s[0]:16.4f}')
print(f'exact            : {exact:16.4f}')
```

```
in order (cumsum):     1087937.0000
pairwise         :     1000000.0000
exact            :     1000000.0149
```

The in-order sum ends at $1{,}087{,}937$, 9% too high, inside its bound $(n-1)u\sum_i |x_i| \approx 6 \times 10^5$. Pairwise summation returns $1{,}000{,}000$, within 0.015 of the exact value and inside its bound $u \log_2 n \sum_i |x_i| \approx 1.4$.

The in-order error has the mechanism behind lost small updates: each addend is rounded to the gap of the running sum. Once the sum passes $2^{19} = 524{,}288$, fp32 numbers are $1/16$ apart, and 0.1 is 1.6 gaps, so each addition rounds to two gaps, 0.125, 25% too much. Above 2^{20} the gap is $1/8$ and the addend rounds to one gap, again 0.125; between 2^{17} and 2^{19} it rounded to 0.09375, too little. Because the result depends on the order, a sum reduced in parallel need not be the same on another device or in another run. Welford's recursion can be evaluated pairwise as well: the merge rule of Exercise 9 combines the statistics of two partial sums, and it is how running moments are combined across devices.

35.8.4 Conditioning

The condition number of a matrix and the filter factors of ridge regression are developed in Section 33.5. New here are forward and backward error, a formula that measures the backward error, and the price in digits of the normal equations.

35.8.4.1 Two Nearly Parallel Lines

So far the answer of a computation was a number. For a linear system it is a vector, and Section 33.5.3.2 showed on the system

$$x_1 + x_2 = 2, \qquad x_1 + 1.0001\,x_2 = 2.0001,$$

how fragile such an answer can be. The solution is $(1, 1)$, and raising the second right-hand side by 10^{-4}, a relative change of 3.5×10^{-5} in the data, moves it to $(0, 2)$. Each equation is a line in the plane, and the solution is where the lines cross. These two lines have slopes -1 and -0.9999 and meet at an angle of 0.003°, so shifting one of them slightly slides the crossing far along the other (Figure 35.27).

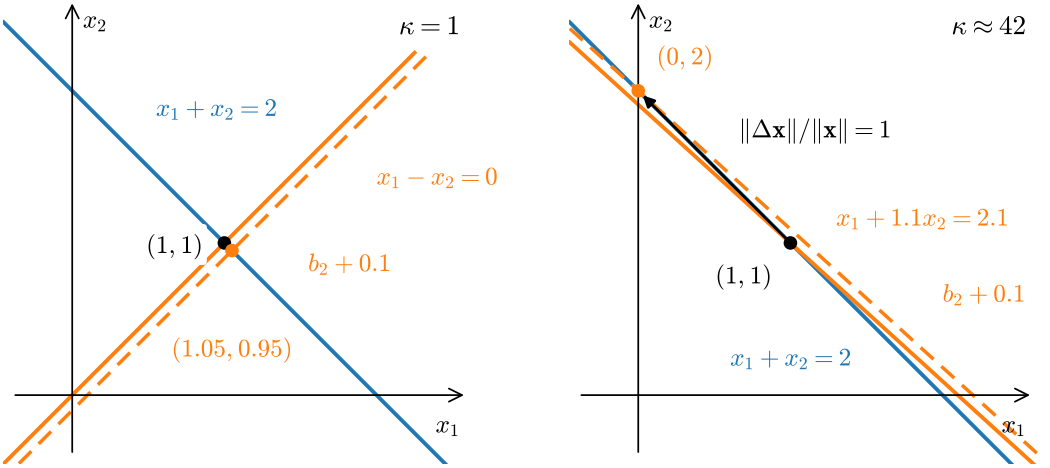


Figure 35.27: A linear system is sensitive when its lines are nearly parallel. (a) Two perpendicular lines meet at $(1, 1)$; shifting the second by 0.1 moves the crossing by 5%: amplification 1, $\kappa = 1$. (b) The lines $x_1 + x_2 = 2$ and $x_1 + 1.1x_2 = 2.1$ also meet at $(1, 1)$, but the same shift moves the crossing to $(0, 2)$, a 100% change from a 3.4% change in $\mathbf{b}$: amplification 29, bounded by $\kappa \approx 42$. The text's system, with 1.0001 in place of 1.1, has the same geometry at an angle of 0.003° and $\kappa \approx 4 \times 10^4$.

Let us read the same numbers in the other direction. Suppose a solver returned $\hat{\mathbf{x}} = (0, 2)$ for the original system. Its **forward error**, the distance to the true solution, is 100%. Yet $(0, 2)$ is the exact solution of a nearby problem, the one with b_2 changed by 10^{-4}. The smallest such change of the data is the **backward error**. A solver with a tiny backward error has solved a nearby problem exactly, and it can still be far from the answer.

How far is a property of the problem, not of the solver. It is the condition number of Section 35.8.3.1 once more, now for the function $\mathbf{b} \mapsto \mathbf{A}^{-1}\mathbf{b}$: the largest factor by which a relative change of the input can become a relative change of the output, taken over all right-hand sides and all directions of the change. For a matrix with singular values $\sigma_1 \geq \cdots \geq \sigma_d > 0$ that factor is $\kappa(\mathbf{A}) = \sigma_1/\sigma_d$, by Equation 33.58, and it is about 4×10^4 for the two lines. The shift $(0, 10^{-4})$ realizes 2.8×10^4 of it, because it is not along the most amplified direction (Exercise 11).

35.8.4.2 Backward and Forward Error

For a linear system $\mathbf{Ax} = \mathbf{b}$ and a computed solution $\hat{\mathbf{x}}$, the forward error is $\|\hat{\mathbf{x}} - \mathbf{x}\|/\|\mathbf{x}\|$ and the backward error is the size of the smallest change of the data that makes $\hat{\mathbf{x}}$ exact. An algorithm is **backward stable** if its backward error is of the order of u, so that it returns the exact answer of a problem that differs from the given one by about one rounding of the data. The SVD and the QR decomposition are backward stable, and so in practice is the LU

decomposition with pivoting that `np.linalg.solve` uses. The one-pass variance is not: the variance of any data set is nonnegative, so a negative output is the exact variance of no data at all (Exercise 3).

The definition asks for a minimization over all changes of the data. Luckily, for changes of the matrix it has a closed form, computable from the residual.

Proposition (backward error from the residual). *Let $\hat{\mathbf{x}} \neq \mathbf{0}$ and $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$. The smallest $\|\Delta\mathbf{A}\|_2$ with $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$ is $\|\mathbf{r}\|/\|\hat{\mathbf{x}}\|$, attained by $\Delta\mathbf{A} = \mathbf{r}\hat{\mathbf{x}}^{\top}/\|\hat{\mathbf{x}}\|^2$.*

Proof. Any such $\Delta\mathbf{A}$ satisfies $\Delta\mathbf{A}\hat{\mathbf{x}} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{r}$, so $\|\mathbf{r}\| \le \|\Delta\mathbf{A}\|\,\|\hat{\mathbf{x}}\|$. The rank-one matrix $\mathbf{r}\hat{\mathbf{x}}^{\top}/\|\hat{\mathbf{x}}\|^2$ maps $\hat{\mathbf{x}}$ to $\mathbf{r}$, and its norm is $\|\mathbf{r}\|\,\|\hat{\mathbf{x}}\|/\|\hat{\mathbf{x}}\|^2 = \|\mathbf{r}\|/\|\hat{\mathbf{x}}\|$. ∎

Relative to the size of $\mathbf{A}$, the backward error is

$$\omega = \frac{\|\mathbf{b} - \mathbf{A}\hat{\mathbf{x}}\|}{\|\mathbf{A}\|\,\|\hat{\mathbf{x}}\|}. \tag{35.105}$$

A small residual certifies a small backward error, whatever the condition of the problem. For $\hat{\mathbf{x}} = (0, 2)$ in the two-line system, $\mathbf{r} = (0, -10^{-4})$ and $\omega = 10^{-4}/(2.00005 \cdot 2) = 2.5 \times 10^{-5}$. The version that perturbs $\mathbf{b}$ as well is due to Rigal and Gaches (see Higham (2002)). In floating point the residual is itself computed with rounding errors, so printed values of ω below about u are noise.

The condition number converts backward error into forward error. This is the result to remember, and its proof needs only the two norms $\|\mathbf{A}\| = \sigma_1$ and $\|\mathbf{A}^{-1}\| = 1/\sigma_d$ (Section 33.5).

Proposition (forward error $\le$ condition number $\times$ backward error). *Let $\mathbf{A}$ be invertible, let $\mathbf{A}\mathbf{x} = \mathbf{b}$, and suppose the computed $\hat{\mathbf{x}}$ exactly solves a nearby system, $(\mathbf{A} + \Delta\mathbf{A})\,\hat{\mathbf{x}} = \mathbf{b}$ with $\|\Delta\mathbf{A}\| \le \omega\|\mathbf{A}\|$. Then*

$$\frac{\|\hat{\mathbf{x}} - \mathbf{x}\|}{\|\hat{\mathbf{x}}\|} \;\le\; \kappa(\mathbf{A})\,\omega. \tag{35.106}$$

Proof. Subtracting $\mathbf{A}\mathbf{x} = \mathbf{b}$ from $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$ gives $\mathbf{A}(\hat{\mathbf{x}} - \mathbf{x}) = -\Delta\mathbf{A}\,\hat{\mathbf{x}}$, hence $\hat{\mathbf{x}} - \mathbf{x} = -\mathbf{A}^{-1}\Delta\mathbf{A}\,\hat{\mathbf{x}}$ and

$$\|\hat{\mathbf{x}} - \mathbf{x}\| \le \|\mathbf{A}^{-1}\|\,\|\Delta\mathbf{A}\|\,\|\hat{\mathbf{x}}\| \le \frac{1}{\sigma_d}\,\sigma_1\,\omega\,\|\hat{\mathbf{x}}\| = \kappa(\mathbf{A})\,\omega\,\|\hat{\mathbf{x}}\|. \qquad \blacksquare$$

The error here is measured relative to $\hat{\mathbf{x}}$, the vector we have. Relative to the true $\mathbf{x}$, substituting $\|\hat{\mathbf{x}}\| \le \|\mathbf{x}\| + \|\hat{\mathbf{x}} - \mathbf{x}\|$ and solving for $\|\hat{\mathbf{x}} - \mathbf{x}\|$ gives the bound $\kappa\omega/(1 - \kappa\omega)$ when $\kappa\omega < 1$, the same to first order. On the two-line system, $\kappa\omega = 4 \times 10^4 \cdot 2.5 \times 10^{-5} = 1.0$, and the error relative to $\hat{\mathbf{x}}$ is $\|(-1, 1)\|/\|(0, 2)\| = 0.71$, inside the bound. For an error in the right-hand side instead of the matrix, the corresponding bound is Equation 33.58.

A backward-stable solver has $\omega \approx u$, and taking $\log_{10}$ of the bound gives the rule of thumb

$$\text{correct digits in } \hat{\mathbf{x}} \;\approx\; \text{digits carried by the format} \;-\; \log_{10}\kappa(\mathbf{A}).$$

The rule is a worst case. A matrix with $\kappa = 10^3$ can lose three of fp32's seven digits. The chapter's running example $\frac{1}{2}(x_1^2 + 1000x_2^2)$ has that κ and loses none: its matrix is diagonal, and each unknown takes one correctly rounded division.

35.8.4.3 Example: The Hilbert Matrix

The **Hilbert matrix** $\mathbf{H}_{ij} = 1/(i+j-1)$, $i, j = 1, \dots, n$, has a condition number that grows exponentially with n: its rows become nearly parallel in the same way as the two lines above. We solve $\mathbf{Hx} = \mathbf{b}$ with $\mathbf{b} = \mathbf{H1}$, whose exact solution is $\mathbf{1}$, and compare the forward error with the rule of thumb and with the backward error Equation 35.105. One caveat before we judge the solver. Computing $\mathbf{b} = \mathbf{H1}$ in fp64 rounds every entry of $\mathbf{b}$, a change of the data of relative size u before the solver starts. (Rounding the entries $1/3, 1/7, \dots$ does no harm here, because $\mathbf{b}$ is built from the matrix as stored.) How much does that alone cost? Every fp64 number is a rational number, so the stored system can also be solved without any rounding, by Gauss–Jordan elimination on `Fraction` entries. The table reports the error of that exact solve beside the error of `np.linalg.solve`.

```
def solve_exact(A, b):
    """Gauss-Jordan elimination in exact rational arithmetic."""
    M = [[Fraction(a) for a in row] + [Fraction(c)] for row, c in zip(A, b)]
    n = len(M)
    for j in range(n):              # no pivoting: A is positive definite here
        for i in range(n):
            if i != j:
                r = M[i][j] / M[j][j]
                M[i] = [a - r * c for a, c in zip(M[i], M[j])]
    return np.array([float(M[i][n] / M[i][i]) for i in range(n)])
```

```
print(' n     kappa  forward error  digits kept  16 - log10 kappa  backward error'
      '  exact solve error')
for n in [4, 6, 8, 10, 12]:
    i = np.arange(n)
    H = 1.0 / (1.0 + i[:, None] + i[None, :])   # Hilbert matrix
    x_true = np.ones(n)
    b = H @ x_true
    x_hat = np.linalg.solve(H, b)
    kappa = np.linalg.cond(H)
    fwd = np.linalg.norm(x_hat - x_true) / np.linalg.norm(x_true)
    omega = (np.linalg.norm(b - H @ x_hat)
             / (np.linalg.norm(H, 2) * np.linalg.norm(x_hat)))
    exact = np.linalg.norm(solve_exact(H, b) - x_true) / np.linalg.norm(x_true)
    print(f'{n:2d}  {kappa:8.1e}  {fwd:13.1e}  {-np.log10(fwd):11.1f}'
          f'  {16 - np.log10(kappa):16.1f}  {omega:14.1e}  {exact:17.1e}')
```

n	kappa	forward error	digits kept	16 - log10 kappa	backward error	exact solve error
4	1.6e+04	4.1e-14	13.4	11.8	0.0e+00	1.1e-13
6	1.5e+07	1.4e-10	9.8	8.8	1.3e-16	1.0e-10
8	1.5e+10	6.1e-08	7.2	5.8	5.7e-17	7.0e-08
10	1.6e+13	8.7e-05	4.1	2.8	1.0e-16	4.5e-05
12	1.8e+16	3.2e-01	0.5	-0.2	9.8e-17	1.4e-01

The digits kept fall from about 13 at $n = 4$ to well under one at $n = 12$, while the backward

error stays at or below about 10^{-16}: each computed vector solves a system that differs from the stored one in the sixteenth or seventeenth digit. Is the solver to blame for the lost digits, then? The last column acquits it. The exact solve of the stored system, which commits no rounding error at all, misses $\mathbf{1}$ by an error of the same order in every row, about 10^{-7}, 10^{-4} and 10^{-1} for $n = 8$, 10 and 12. The digits were lost when $\mathbf{b}$ was stored, before any solver ran. The rule of thumb is pessimistic by about one digit (from about half a digit to about one and a half, depending on the row), because the error is not aligned with the most amplified direction (Exercise 11). At $n = 12$, $\kappa u \approx 2$ and the bound no longer constrains anything. The last decimals vary with the LAPACK build; the trend does not.

35.8.4.4 Conditioning of the Normal Equations

The least-squares problem $\min_{\mathbf{w}} \|\mathbf{A}\mathbf{w} - \mathbf{b}\|^2$ with $\mathbf{A} \in \mathbb{R}^{n \times d}$ of full column rank is solved by the normal equations $\mathbf{A}^\top\mathbf{A}\mathbf{w} = \mathbf{A}^\top\mathbf{b}$. This is an algebraically equivalent route through a worse-conditioned matrix (Section 33.5.3.2):

$$\kappa(\mathbf{A}^\top\mathbf{A}) = \kappa(\mathbf{A})^2. \tag{35.107}$$

We have seen this before, in the one-pass variance: the convenient formula squares the condition number of the problem. By the rule of thumb, the normal equations can lose about $2\log_{10}\kappa$ digits. A solve by QR ($\mathbf{A} = \mathbf{Q}\mathbf{R}$ with orthonormal $\mathbf{Q}$ and triangular $\mathbf{R}$) or by the SVD works on $\mathbf{A}$ directly and loses about $\log_{10}\kappa$, *when the residual* $\mathbf{b} - \mathbf{A}\mathbf{w}$ *is small*. With a large residual the least-squares problem itself is more sensitive: its condition number is about $\kappa + \kappa^2 \tan\theta$, where θ is the angle between $\mathbf{b}$ and the column space of $\mathbf{A}$. QR and SVD then lose close to $2\log_{10}\kappa$ digits too, but never more than the problem forces; the normal equations lose that many whatever the residual.

The chapter's running example is too well conditioned to show the difference: its design has $\mathbf{A}^\top\mathbf{A} = \mathrm{diag}(1, 1000)$, so $\kappa(\mathbf{A}) = \sqrt{1000} \approx 32$. The cell therefore builds harder designs, with $\kappa(\mathbf{A}) = 10^3$ and 10^6, from random orthonormal factors and a right-hand side in the column space (zero residual). We expect gaps of three and of six digits:

```
rng = np.random.default_rng()
n, d = 100, 10
for kappa in [1e3, 1e6]:
    U, _ = np.linalg.qr(rng.normal(size=(n, d)))    # orthonormal columns
    V, _ = np.linalg.qr(rng.normal(size=(d, d)))
    sigma = np.logspace(0, -np.log10(kappa), d)     # kappa(A) as chosen
    A = U * sigma @ V.T                             # A = U diag(sigma) V^T
    w_true = rng.normal(size=d)
    b = A @ w_true                                  # zero residual
    w_ne = np.linalg.solve(A.T @ A, A.T @ b)        # normal equations
    w_ls = np.linalg.lstsq(A, b, rcond=None)[0]     # SVD-based solve
    digits = [-np.log10(np.linalg.norm(w - w_true) / np.linalg.norm(w_true))
              for w in (w_ne, w_ls)]
    print(f'kappa(A) = {kappa:.0e}: normal equations keep {digits[0]:4.1f} '
          f'digits, lstsq keeps {digits[1]:4.1f}')
```

```
kappa(A) = 1e+03: normal equations keep 10.3 digits, lstsq keeps 14.3
kappa(A) = 1e+06: normal equations keep  5.7 digits, lstsq keeps 11.7
```

At $\kappa(\mathbf{A}) = 10^3$ the normal equations keep about 10 to 12 digits against about 14 for `lstsq`, and at 10^6 about 4 to 6 against about 11 to 13. The exact digits vary from run to run by a digit or two; the gaps scatter around $\log_{10}\kappa$, mostly within a digit of it. This is why `lstsq` and the pseudoinverse of Section 33.5.3.1 work from the SVD and not from $(\mathbf{A}^\top\mathbf{A})^{-1}\mathbf{A}^\top$.

35.8.4.5 Ridge Regularization and Conditioning

When $\kappa(\mathbf{A})$ itself is large, because features are nearly collinear or the design is rank deficient, no choice of algorithm helps: the problem must change. One change keeps the answer. A **preconditioner** changes variables so that the new problem is better conditioned and its solution maps back to the original one. Dividing each column of $\mathbf{A}$ by its norm is an example (Exercise 6), and the per-coordinate step sizes of Section 35.5 do the same inside an optimizer.

Ridge regularization changes the answer as well. It minimizes $\frac{1}{2}\|\mathbf{A}\mathbf{w}-\mathbf{b}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$, whose minimizer is

$$\mathbf{w}_\lambda = (\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{A}^\top\mathbf{b}, \tag{35.108}$$

the ridge solution of Section 33.5.3.1. Let $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$ be the singular values of $\mathbf{A} \in \mathbb{R}^{n\times d}$, with $\sigma_i = 0$ beyond its rank. The matrix $\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I}$ has the eigenvectors of $\mathbf{A}^\top\mathbf{A}$ and the eigenvalues $\sigma_i^2 + \lambda \geq \lambda > 0$. It is therefore positive definite, hence invertible even when $\mathbf{A}$ is rank deficient, and

$$\kappa(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I}) = \frac{\sigma_1^2 + \lambda}{\sigma_d^2 + \lambda}. \tag{35.109}$$

Whenever $\sigma_1 > \sigma_d$ this ratio decreases strictly in λ (its derivative is $(\sigma_d^2 - \sigma_1^2)/(\sigma_d^2+\lambda)^2 < 0$) and tends to 1 as $\lambda \to \infty$. Adding λ lifts the floor of the spectrum. While $\lambda \ll \sigma_1^2$ it barely moves the ceiling, so κ falls about in proportion to $1/\lambda$ once $\lambda \gg \sigma_d^2$. The damping term of second-order optimizers and the jitter added to a kernel matrix in Gaussian process regression (Chapter 40) are this $\lambda\mathbf{I}$ under other names.

The price is a different minimizer. In the present notation, with $\mathbf{v}_i$ the right singular vectors of $\mathbf{A}$ and $\mathbf{w}_0 = \mathbf{A}^+\mathbf{b}$ the minimum-norm least-squares solution, the filter factors of Equation 33.56 read

$$\mathbf{w}_\lambda = \sum_{i=1}^{d} \frac{\sigma_i^2}{\sigma_i^2 + \lambda}\,(\mathbf{v}_i^\top\mathbf{w}_0)\,\mathbf{v}_i. \tag{35.110}$$

Ridge damps most the directions with small σ_i, and these are the very directions in which the unregularized solve amplifies data errors by $1/\sigma_i$. Figure 35.28 shows the effect on the level sets; each panel is centered on its own minimizer, so the shift of the minimizer toward the origin is not drawn.

The same κ counts the steps of gradient descent on the ridge objective. At the best fixed step, gradient descent on a quadratic with condition number $\kappa > 1$ contracts the error by $\rho = (\kappa - 1)/(\kappa + 1)$ per step, by Equation 35.9. Started at the origin, where the error is

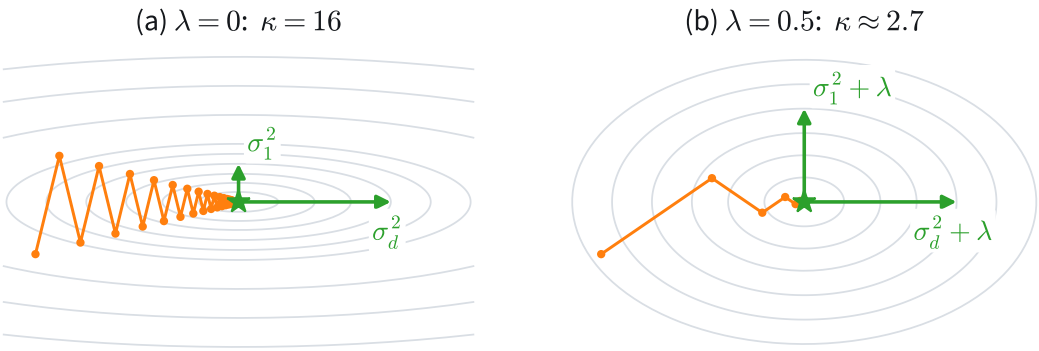


Figure 35.28: Level sets of the least-squares objective before and after adding the ridge term $\frac{\lambda}{2}\|\mathbf{w}\|^2$, drawn for $\kappa = 16$ so that both panels stay legible (the running example's $\kappa = 1000$ would draw as a line). The penalty lifts every eigenvalue of $\mathbf{A}^\top\mathbf{A}$ by λ; the semi-axes, proportional to $1/\sqrt{\sigma_i^2 + \lambda}$, are long where the curvature is small, and the valley rounds into a bowl. Orange: gradient-descent iterates from the same start at each panel's best fixed step; green arrows: the principal axes, labeled by their curvatures.

$-\mathbf{w}^\star$, it has relative error $\|\mathbf{w}_k - \mathbf{w}^\star\|/\|\mathbf{w}^\star\| \le \rho^k$ after k steps, so a tolerance $0 < \text{tol} < 1$ is met within

$$k \le \left\lceil \frac{\log(1/\text{tol})}{\log \frac{\kappa+1}{\kappa-1}} \right\rceil \le \left\lceil \frac{\kappa}{2} \log \frac{1}{\text{tol}} \right\rceil \quad \text{steps.} \tag{35.111}$$

The second inequality holds because $\log \frac{\kappa+1}{\kappa-1} = 2\,\text{artanh}(1/\kappa) \ge 2/\kappa$. For six digits it allows $\frac{1}{2}\log 10^6 = 6.9$ steps per unit of κ; a run in more than two dimensions can finish sooner (Exercise 6).

How many steps does a given λ save, and what does it cost? We measure both on the running example. Its objective $\frac{1}{2}(w_1^2 + 1000w_2^2) - w_1 - 1000w_2$ is least squares with $\mathbf{A}^\top\mathbf{A} = \text{diag}(1, 1000)$ and $\mathbf{A}^\top\mathbf{b} = (1, 1000)$. Its minimizer is $(1, 1)$, so the ridge minimizer displays the two filter factors directly. The function `gd_iterations` runs gradient descent at the best fixed step and returns the step count and the ridge minimizer:

```
def gd_iterations(AtA, Atb, lam, tol=1e-6):
    """GD steps at step 2/(L+mu) to reach relative error tol on ridge."""
    M = AtA + lam * np.eye(len(Atb))
    w_star = np.linalg.solve(M, Atb)
    mu, L = np.linalg.eigvalsh(M)[[0, -1]]
    eta, w, k = 2 / (L + mu), np.zeros(len(Atb)), 0
    while np.linalg.norm(w - w_star) > tol * np.linalg.norm(w_star):
        w, k = w - eta * (M @ w - Atb), k + 1
    return k, w_star

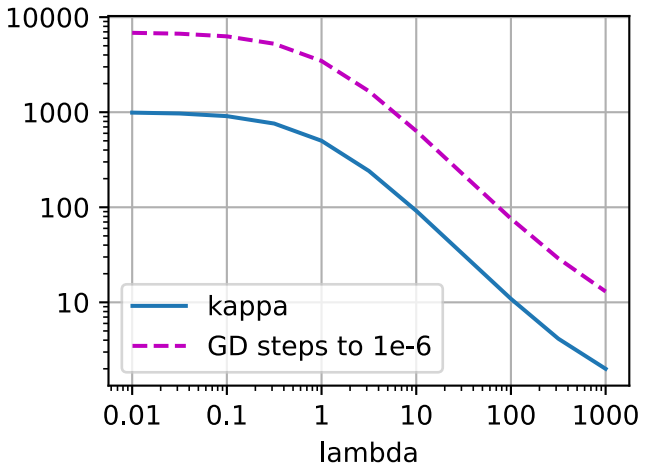
AtA, Atb = np.diag([1.0, 1000.0]), np.array([1.0, 1000.0])
print('lambda   kappa  steps  steps/kappa  ridge minimizer')
```

```
for lam in [0, 1, 10, 100, 1000]:
    kappa = (1000 + lam) / (1 + lam)
    k, w_star = gd_iterations(AtA, Atb, lam)
    print(f'{lam:6d} {kappa:7.1f} {k:6d} {k / kappa:12.2f}  {w_star.round(4)}')
```

```
lambda   kappa  steps  steps/kappa  ridge minimizer
     0  1000.0   6908         6.91  [1. 1.]
     1   500.5   3458         6.91  [0.5   0.999]
    10    91.8    635         6.92  [0.0909 0.9901]
   100    10.9     76         6.98  [0.0099 0.9091]
  1000     2.0     13         6.51  [0.001 0.5  ]
```

The condition number falls from 1000 to 2 as λ goes from 0 to 1000, and the step count falls with it, from $6{,}908$ to 13. The ratio of steps to κ stays near 6.9 (at small κ the approximation $2/\kappa$ is loose, and rounding up to a whole step matters). The last column shows what the saved steps cost. At $\lambda = 1$ the slow direction, with $\sigma^2 = 1$, is halved to 0.5, while the stiff direction, with $\sigma^2 = 1000$, moves by 0.1%. At $\lambda = 1000$ the stiff direction is halved too. A sweep over λ shows κ and the step count together:

```
lams = np.logspace(-2, 3, 11)
kappas = (1000 + lams) / (1 + lams)
steps = [gd_iterations(AtA, Atb, lam)[0] for lam in lams]
d2l.plot(lams, [kappas, steps], 'lambda', None,
         legend=['kappa', 'GD steps to 1e-6'], xscale='log', yscale='log')
```



On log–log axes the two curves are parallel, a factor of 6.9 apart, with slope close to -1 while λ lies well between the eigenvalues 1 and 1000. Ridge thus conditions the linear solve and gradient descent alike, and it pays for both by shrinking the solution. Section 35.7.3.2 shows that the penalty $\frac{\lambda}{2}\|\mathbf{w}\|^2$ corresponds to a constraint on $\|\mathbf{w}\|$, so a λ chosen for statistical reasons also sets the conditioning of the computation.

None of these tools covers error that grows over many steps of an iteration, such as a gradient through a hundred layers; Section 4.4 takes that up.

35.8.5 Summary

A floating-point format stores numbers with about p bits of relative precision and a finite exponent range. Every rounding introduces a relative error of at most $u = 2^{-p-1}$, while overflow, underflow, and a lost small update are absolute effects that the relative model does not show. Softmax and cross-entropy fail because they pass through values the format cannot hold, and the one-pass variance because it subtracts nearly equal large numbers; none of their answers is extreme. The max shift, cross-entropy computed from logits, and Welford's recursion avoid the detour. The relative condition number $\kappa_f = |xf'/f|$ separates a bad algorithm from a hard problem: when it is small and the computed answer is bad, the algorithm is to blame. For a linear system, $\kappa(\mathbf{A})$ bounds the forward error by κ times the backward error, which a residual certifies, so about $\log_{10}\kappa$ digits are lost, and twice that through the normal equations. Ridge adds λ to every squared singular value, which lowers κ for the solve and for gradient descent alike, at the price of shrinking the solution by the filter factors $\sigma_i^2/(\sigma_i^2+\lambda)$.

35.8.6 Exercises

1. **Loss-scale window.** Loss scaling multiplies the loss by a factor S, which multiplies every gradient by S, and divides the gradients by S after they have been converted to fp32. In fp16 the smallest normal number is $2^{-14} \approx 6.1 \times 10^{-5}$, the smallest subnormal number is $2^{-24} \approx 6.0 \times 10^{-8}$ (values below 2^{-25} round to zero), and the largest finite number is 65504. Suppose that the exact nonzero gradient entries of a network have magnitudes between 10^{-9} and 10^{-1}, and that the backward pass computes the scaled gradients in fp16.

    1. Find all powers of two S for which every scaled gradient is a normal fp16 number.
    2. The gradients of a deeper network have magnitudes between 10^{-12} and 10^{-1}. Show that no scale keeps all of them normal. Take the largest power of two S that avoids overflow. How many significant bits does a scaled gradient of magnitude 10^{-12} keep, and what is its relative rounding error?
    3. Dynamic loss scaling (Section 5.5.3.3) picks the largest scale that avoids overflow and thereby sacrifices the smallest gradients. Why is that the better end to sacrifice?
    4. How do the answers to the first two sub-items change when the gradients are computed in bf16?

2. code **Blame the problem or the algorithm.** Each expression below is evaluated in fp32. For each, compute the relative condition number Equation 35.102 of the *function* in the stated regime and the cancellation factor Equation 35.103 of the *formula as written*. Decide whether the problem or the algorithm is to blame, and rewrite the formula so that its error matches the problem's. Check both forms against fp64.

    1. The root of smaller magnitude of $ax^2 + bx + c = 0$ for $b > 0$ and $b^2 \gg 4|ac|$, as a function of c, for example $a = c = 1$ and $b = 10^4$. (Hint: multiply by the conjugate.)
    2. The function $\log(1-e^{-x})$ for $x > 0$, which computes $\log(1-q)$ from $\log q = -x$, at $x \in \{10^{-6}, 10^{-3}, 1, 20\}$. Find one formulation that is accurate for small x and one that is accurate for large x, and choose a switching point.
    3. The binary cross-entropy of a logit t with label 1, $\log(1+e^{-t})$, at $t \in \{-800, -50, 0, 50\}$. At which of these does the formula as written overflow, and at which does it return 0? Show that $\max(-t, 0) + \operatorname{log1p}(e^{-|t|})$ equals it, never

overflows, and loses no digits. Which part of the log-sum-exp sandwich is it?

4. $\sqrt{x+1}-\sqrt{x}$ at $x=10^8$.

Part 2 is adapted from Martin Mächler, Accurately Computing log(1 − exp(−|a|)), Section 2.

3. code **A negative variance.** The one-pass formula $\frac{1}{N}\sum_i x_i^2 - m^2$ of Section 35.8.3 cancels badly when the mean is large compared with the standard deviation.

   1. Construct three fp64 numbers for which the one-pass formula returns a strictly negative variance. Explain the magnitude of the result from the spacing of fp64 numbers near x_i^2.
   2. Run Welford's recursion Equation 35.104 on the same three numbers and compare with the exact variance.
   3. A backward-stable variance algorithm (Section 35.8.4) returns the exact variance of slightly perturbed data. Explain why a single negative output proves that the one-pass formula is not backward stable.

4. code **Normal equations in floating point.** The normal equations square the condition number Equation 35.107.

   1. For $\tau > 0$, the $(m+1)\times m$ matrix $\mathbf{A}$ of Läuchli (1961) has a first row of ones and $\tau\mathbf{I}$ below it. Compute $\mathbf{A}^\top\mathbf{A}$, the singular values of $\mathbf{A}$, and $\kappa(\mathbf{A})$. Show that for $\tau = 10^{-8}$ the matrix $\mathbf{A}^\top\mathbf{A}$ rounds to a singular matrix in fp64. How many correct digits does the rule of thumb of Section 35.8.4 predict for a QR-based least-squares solve with this $\mathbf{A}$ when $\mathbf{b}$ lies in the range of $\mathbf{A}$? What changes in the prediction when the residual is large?
   2. Modify the `#numerical-stability-conditioning-normal-equations` cell to sweep $\kappa(\mathbf{A}) = 10^k$ for $k = 2, \dots, 10$. Plot the correct digits of the normal equations and of `lstsq` against k, together with the two rule-of-thumb lines $16-2k$ and $16-k$. At which k does the rule predict that the normal equations keep no correct digit, and what do you observe there?

5. **Ridge as a conditioning dial.** Let $\mathbf{A}$ have singular values $\sigma_1 = 1$ and $\sigma_d = 10^{-3}$, and let $\mathbf{w}_0 = \mathbf{A}^+\mathbf{b}$ be the least-squares minimizer.

   1. For $\lambda \in \{0, 10^{-4}, 10^{-2}\}$, compute $\kappa(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})$ from Equation 35.109, the number of correct float64 digits that a direct solve of Equation 35.108 can keep, and the number of gradient-descent steps that Equation 35.111 predicts for a relative error of 10^{-6}.
   2. Using the filter factors Equation 35.110, choose λ so that 90% of the $\mathbf{v}_d$ component of $\mathbf{w}_0$ survives. What are $\kappa(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})$ and the predicted digits and steps then? Is the trade worth it?
   3. Show that $\|\mathbf{w}_\lambda\|$ is strictly decreasing in λ unless $\mathbf{A}^\top\mathbf{b} = \mathbf{0}$. Conclude that every norm budget r with $0 < r < \|\mathbf{w}_0\|$ in Section 35.7.3.2 corresponds to exactly one $\lambda > 0$.

6. code **Feature scaling as preconditioning.** Features measured in different units give the columns of a design matrix very different norms. Let $\mathbf{X}$ be a 200×5 matrix with standard normal entries, let $\mathbf{A} = \mathbf{X}\,\mathrm{diag}(1,3,10,30,100)$, and let $\mathbf{b} = \mathbf{A}\mathbf{w}^\star$ for a random $\mathbf{w}^\star$.

   1. Apply `gd_iterations` of the `#numerical-stability-conditioning-ridge` cell, which takes $\mathbf{A}^\top\mathbf{A}$ and $\mathbf{A}^\top\mathbf{b}$, to this design at $\lambda = 0$. Compute $\kappa(\mathbf{A}^\top\mathbf{A})$ and compare the step count with the bound Equation 35.111. Explain the shortfall through the

components of the start's error along the two extreme eigenvectors of $\mathbf{A}^\top\mathbf{A}$.
2. Divide every column of $\mathbf{A}$ by its norm and repeat the measurement. How does the minimizer of the rescaled problem map back to $\mathbf{w}^\star$?
3. Part 2 of Exercise 1 of Section 35.2 writes gradient descent in new coordinates $\mathbf{u}$ with $\mathbf{w} = \mathbf{T}\mathbf{u}$ as gradient descent on the original problem preconditioned by a positive definite matrix $\mathbf{P}$, with update $\mathbf{w} \leftarrow \mathbf{w} - \eta\,\mathbf{P}^{-1}\nabla f(\mathbf{w})$. Which $\mathbf{T}$ and which $\mathbf{P}$ correspond to column scaling? How is $\mathbf{P}$ related to the diagonal of $\mathbf{A}^\top\mathbf{A}$?

7. code **Stochastic rounding.** ◪ Stochastic rounding (Gupta et al. 2015) rounds a real number x between adjacent representable values $x_- < x_+$ to x_+ with probability $q = (x - x_-)/(x_+ - x_-)$ and to x_- otherwise.
   1. Show that the rounded value has expectation x and variance $q(1-q)\Delta^2 \le \Delta^2/4$, where $\Delta = x_+ - x_-$.
   2. A bf16 weight starts at $w = 1$ and receives $K = 1000$ updates of -10^{-4}, each followed by stochastic rounding. The weight stays in $[0.5, 1]$, where the bf16 spacing is $\Delta = 2^{-8}$. Show that the rounding errors of different steps are uncorrelated, and predict the mean and the standard deviation of the final weight. Which final weight does round-to-nearest produce?
   3. For a positive fp32 number, adding a uniformly random integer from $\{0, \dots, 2^{16} - 1\}$ to its bit pattern and then clearing the lower 16 bits rounds it stochastically to bf16. Explain why. Implement this rounding next to the `to_bf16` function of Section 35.8.1 and check the predictions of the previous sub-item with 1000 independent runs. Compare with an fp32 master copy that receives the same updates.

8. **Exact subtraction.** ◪ Subtraction can be exact and still return an inaccurate difference.
   1. Show that the bound Equation 35.103 is attained for suitable δ_a and δ_b with $|\delta_a|, |\delta_b| \le u$.
   2. Prove Sterbenz's lemma (Sterbenz 1974): if a and b are floating-point numbers in the format Equation 35.95 with $b/2 \le a \le 2b$, then $a - b$ is a floating-point number, so the computed difference is exact. Assume that the format includes subnormal numbers.
   3. In the `#numerical-stability-conditioning-log1p` cell, the fp32 difference $a - b$ misses the true value 3.0×10^{-7} by about 20 percent. Use Sterbenz's lemma and the printed number of ulps to verify that the subtraction was exact, compute the relative errors of the stored a and b, and check that the bound of the first sub-item covers the error.

9. code **Merging running moments.** ◪ Data-parallel training computes statistics on shards and combines them. Let shard A hold k_A values with mean m_A and centered sum of squares M_A, and let shard B hold k_B values with m_B and M_B.
   1. Derive exact formulas for the mean m and the centered sum of squares M of the union of the two shards in terms of k_A, m_A, M_A, k_B, m_B, and M_B.
   2. Show that the case $k_B = 1$ is Welford's recursion Equation 35.104.
   3. Split the 10^5 samples of the `#numerical-stability-conditioning-welford` cell into 2^j shards of nearly equal size (`np.array_split`) for $j \in \{1, 4, 8\}$. Compute each shard's statistics with Welford's recursion, merge them pairwise in a binary tree, and compare the variance with the two-pass reference. Merge the shards a second time by adding their sums $\sum_i x_i$ and $\sum_i x_i^2$ and applying the one-pass for-

mula. Explain the difference.

10. code **Summation error growth.** ▯ Let $S_n = \sum_{i=1}^{n} x_i$ and let $\hat{S}_n$ be its computed value. In-order summation has the worst-case bound $|\hat{S}_n - S_n| \le (n-1)\,u \sum_i |x_i| + O(u^2)$.

    1. For $n = 2^k$, recursive pairwise summation splits the terms into two halves, sums each half recursively, and adds the two results. Prove that it satisfies $|\hat{S}_n - S_n| \le u \log_2(n) \sum_i |x_i| + O(u^2)$.
    2. For $n \in \{2^4, 2^8, 2^{12}, 2^{16}, 2^{20}\}$ uniform random numbers in $[0, 1)$ stored as fp32, measure the relative errors of in-order summation (`np.cumsum(x)[-1]`), of `np.sum` (a blocked pairwise summation), and of Kahan's compensated summation (Kahan 1965) against a float64 reference. Kahan's method keeps a correction c: after each addition $t = s + y$ it sets $c = (t - s) - y$ and subtracts c from the next term. Plot the three errors against n together with the two bounds.
    3. How does the measured error of in-order summation grow with n compared with its worst-case bound, and why? Why is the text's sum of 10^7 copies of 0.1 so much closer to the worst case?
    4. What limits the error of the two accurate methods at large n?

    Adapted from Steven G. Johnson, MIT 18.335 Introduction to Numerical Methods, Problem Set 1, Problem 4.

11. code **Residual as a certificate.** ▯ The two-line system of Section 35.8.4.1 and the Hilbert systems of the `#numerical-stability-conditioning-hilbert` cell show how much of the condition number a perturbation uses.

    1. For the system $x_1 + x_2 = 2$, $x_1 + 1.0001x_2 = 2.0001$, find the direction of $\Delta\mathbf{b}$, with $\|\Delta\mathbf{b}\| = 10^{-4}$, that moves $\mathbf{x}$ the most, and the one that moves it the least. Show that the ratio of the two amplifications is $\kappa(\mathbf{A})$, and that the text's shift $(0, 10^{-4})$ reaches 2.8×10^4 of the possible 4×10^4.
    2. The computed solution $\hat{\mathbf{x}}$ of $\mathbf{Hx} = \mathbf{H1}$ solves exactly the system whose right-hand side is $\mathbf{H1}$ perturbed by $\Delta\mathbf{b} = \mathbf{H}(\hat{\mathbf{x}} - \mathbf{1})$, so the right-hand-side bound of the text gives $\|\hat{\mathbf{x}} - \mathbf{1}\| / \|\mathbf{1}\| \le \kappa(\mathbf{H})\, \|\Delta\mathbf{b}\| / \|\mathbf{H1}\|$. Evaluate $\Delta\mathbf{b}$ in fp64 as `H @ x_hat - b` for each n of the cell, and compare the right-hand side of the bound with the forward error. In which row does the bound appear to fail? Why can a residual computed in working precision do that?
    3. Every fp64 number is a rational number, so `fractions.Fraction` evaluates $\mathbf{H}(\hat{\mathbf{x}} - \mathbf{1})$ for the stored matrix $\mathbf{H}$ without any rounding. Recompute the bound this way and find, in each row, the ratio of the forward error to the bound.
    4. Write that ratio as a product of two factors, each at most 1, one depending only on the right-hand side $\mathbf{H1}$ and one depending only on the error vector $\hat{\mathbf{x}} - \mathbf{1}$. Evaluate both. What do their values say about the directions of $\mathbf{1}$ and of the error relative to the singular vectors of $\mathbf{H}$?

12. code **Mixed-precision iterative refinement.** ▯▯ Mixed-precision training does the bulk of its arithmetic in low precision and keeps the quantities that accumulate small corrections in high precision. Iterative refinement (Moler 1967; Carson and Higham 2018) applies the same split to $\mathbf{Ax} = \mathbf{b}$: the $O(n^3)$ factorization runs in low precision and only $O(n^2)$ residual work runs in high precision. Starting from an fp32 solution $\mathbf{x}_0$, it repeats

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k, \qquad (\mathbf{A} + \Delta\mathbf{A}_k)\,\mathbf{d}_k = \mathbf{r}_k, \qquad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k,$$

where the residual and the update are computed in fp64 and the correction $\mathbf{d}_k$ comes from an fp32 solve.

1. Model the fp32 solve as backward stable, $\|\Delta\mathbf{A}_k\| \le c\,u_{32}\,\|\mathbf{A}\|$ with $u_{32} = 2^{-24}$ and a moderate constant c, and neglect the fp64 rounding errors. Show that the error $\mathbf{e}_k = \mathbf{x}_k - \mathbf{x}$ satisfies $\mathbf{e}_{k+1} = (\mathbf{A} + \Delta\mathbf{A}_k)^{-1}\Delta\mathbf{A}_k\,\mathbf{e}_k$. Derive a contraction factor in terms of $c\,\kappa(\mathbf{A})\,u_{32}$, and state when it is below 1.
2. Implement the iteration for the Hilbert matrices of the `#numerical-stability-conditioning-hilbert` cell with $n = 3, \dots, 10$, using `np.linalg.solve` on fp32 copies of $\mathbf{H}$ and $\mathbf{r}_k$ for the correction. Record the forward error over ten iterations and compare its final value with that of an fp64 solve. For which n does refinement reach the accuracy of the fp64 solve? For which n does it still converge, but too slowly to reach that accuracy within ten iterations? For which n does the error grow?
3. Estimate the measured contraction per iteration for each n and compare it with $\kappa(\mathbf{H})\,u_{32}$ and with the spectral radius of $\mathbf{I} - \widehat{\mathbf{H}}^{-1}\mathbf{H}$, where $\widehat{\mathbf{H}}$ is the fp32 copy of $\mathbf{H}$ and the inverse is applied in fp64. Which perturbation dominates $\Delta\mathbf{A}_k$ in this experiment, and how pessimistic is the bound of the first sub-item?

35.9 Further Reading

This section points past the chapter, and its topics are not of equal weight. Five bear directly on training networks and come first under their headings: the edge of stability, iterate averaging, interpolation, the Fisher matrix, and saddle-point methods. The others are classical optimization and numerical analysis, for when a problem calls for them. Textbooks and courses for what the chapter does teach are listed in its introduction (Chapter 35), and Section 9.11 surveys the practice of training. Nothing else in the book depends on this section.

35.9.1 Gradient Methods

35.9.1.1 Self-Stabilization at the Edge of Stability

Under full-batch gradient descent the sharpness of a network's loss rises to $2/\eta$ and stays there (Cohen et al. 2021). The quadratic model of Section 35.1.2.4 explains why the top mode grows while the sharpness exceeds $2/\eta$, but not what brings the sharpness back down. Damian et al. (2023) keep the cubic term of the Taylor expansion along the top eigenvector. The growing oscillation along that direction then moves the iterate against the gradient of the sharpness, which lowers it, so the sharpness and the oscillation cycle around $2/\eta$. Without the cubic term the oscillation simply grows.

35.9.1.2 Optimal Polynomials and Conjugate Gradients

Take the quadratic $f = \frac{1}{2}\mathbf{x}^\top\mathbf{A}\mathbf{x}$ with eigenvalues in $[\mu, L]$ and minimizer $\mathbf{0}$. Every method whose k-th step lies in the span of the gradients seen so far arrives at $\mathbf{x}_k = p_k(\mathbf{A})\mathbf{x}_0$, for some polynomial p_k of degree at most k with $p_k(0) = 1$. The polynomial that is smallest on the whole interval is a scaled Chebyshev polynomial. Its maximum there is $2\rho^k/(1+\rho^{2k})$, about $2\rho^k$, with the $\rho = (\sqrt{\kappa}-1)/(\sqrt{\kappa}+1)$ of heavy ball (Section 35.1.3). When the dimension d is large compared with k, the eigenvalues can fill the interval, and no method of this kind then beats that rate in the worst case. Conjugate gradients finds at every step the polynomial that minimizes f for the actual $\mathbf{A}$ and $\mathbf{x}_0$, without knowing μ or L, and in

exact arithmetic stops after at most d steps; Nocedal and Wright (2006) (ch. 5) derive it and its preconditioned form.

35.9.1.3 Rates Certified by Semidefinite Programs

The hand proof of Section 35.3.3.3 gives the factor $1-\mu/L$ per step, and the section quotes the exact $(1-\mu/L)^2$, which halves the step count. The exact value comes from *performance estimation*. Drori and Teboulle (2014) write the worst case of a method over all L-smooth convex functions as an optimization problem over the Gram matrix (all inner products) of its iterates and gradients, and bound it by a small semidefinite program, whose dual solution is a proof. Taylor et al. (2018) obtain the squared factor in this way. Lessard et al. (2016) treat the gradient as an unknown function inside a linear recursion, constrained only by μ and L, and certify rates by a semidefinite program whose size does not grow with the number of steps. This is how they found the function on which heavy ball fails (Section 35.1.3).

35.9.2 Stochastic and Adaptive Methods

35.9.2.1 Averaging with a Constant Step

The noise floor of Section 35.4.1 is proportional to η. A decaying step removes it but makes late progress slow. Averaging the iterates removes it differently. Polyak and Juditsky (1992) show that the average of iterates with steps decaying more slowly than $1/k$ reaches the asymptotically optimal covariance. Bach and Moulines (2013) show that for least squares a *constant* step suffices: the averaged iterate reaches an error of order $1/n$ after n examples, without strong convexity.

35.9.2.2 Interpolation

When the model fits every example exactly, the per-example gradients vanish at the minimizer, the floor disappears, and on a strongly convex loss constant-step SGD converges linearly (Section 35.4.4). Ma et al. (2018) analyze this regime and find, for least squares, the batch size beyond which larger batches stop helping. It is set by the ratio of per-example to full-batch curvature rather than by the noise. Under a *strong growth condition*, in which the gradient noise shrinks with the full gradient, SGD can even be accelerated (Vaswani et al. 2019).

35.9.2.3 The Fisher and the Gauss–Newton Matrix

K-FAC (Section 35.5.5) approximates the Fisher matrix $\mathbf{F} = \mathbb{E}[\nabla \log p \, \nabla \log p^\top]$, with the labels drawn from the model (Amari 1998; Martens and Grosse 2015). For softmax cross-entropy and squared loss, $\mathbf{F}$ equals the Gauss–Newton matrix $\mathbf{J}^\top \mathbf{H}_{\text{out}} \mathbf{J}$, with $\mathbf{H}_{\text{out}}$ the Hessian of the loss in the network's output. In general this needs the output to be the natural parameter of an exponential-family likelihood (Section 36.4). The *empirical* Fisher, built from the labels of the data, is cheaper and is often what code calls the Fisher, but it equals neither matrix in general (Martens 2020).

35.9.2.4 Mirror Descent

AdaGrad's metric view (Section 35.5.1) measures step length in a quadratic norm. On the probability simplex the natural steps are multiplicative. Mirror descent replaces $\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2$ in the proximal form of the gradient step by a Bregman divergence $D_\phi(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x}) -$

$\phi(\mathbf{y}) - \nabla\phi(\mathbf{y})^\top(\mathbf{x}-\mathbf{y})$ (Nemirovski and Yudin 1983; Beck and Teboulle 2003). With ϕ the negative entropy, the update becomes the exponentiated gradient $x_i \leftarrow x_i e^{-\eta g_i} / \sum_j x_j e^{-\eta g_j}$. For gradients bounded coordinatewise, the regret defined in that subsection then grows on the simplex in d dimensions with $\sqrt{\log d}$ instead of the $\sqrt{d}$ of projected gradient steps.

35.9.3 Constraints and Duality

35.9.3.1 Saddle-Point Methods

The saddle points of Section 35.7.2 are also the solutions of two-player games such as GAN training (Section 16.1). Simultaneous gradient descent and ascent spirals away from the saddle of xy for every step size (exercise 6). The extragradient method takes a trial step, evaluates the gradient there, and uses that gradient for the actual step from the original point. On convex–concave problems with Lipschitz gradients its averaged iterates converge at the rate $O(1/k)$ (Nemirovski 2004). Optimistic methods reuse the previous gradient in place of the trial one, which saves a gradient evaluation per step, and Daskalakis et al. (2018) apply them to GANs.

35.9.3.2 Problem Classes and Interior-Point Methods

The worked duals of Section 35.7.5 are quadratic programs. Convex problems come in nested classes: linear programs, quadratic programs, second-order cone programs, and semidefinite programs. Interior-point methods solve each to high accuracy in polynomial time. They replace the constraints $g_i(\mathbf{x}) \le 0$ by the barrier $-\frac{1}{s}\sum_i \log(-g_i(\mathbf{x}))$ and follow the minimizers as s grows, along the *central path*. At each such minimizer the numbers $-1/(s\, g_i(\mathbf{x}))$ are dual feasible multipliers, so the duality gap is m/s for m constraints, a certificate of the kind Section 35.7 builds by hand (Boyd and Vandenberghe 2004; Nesterov 2018).

35.9.3.3 Operator Splitting and ADMM

The proximal steps of Section 35.6.4.1 handle a smooth loss plus one simple nonsmooth term. Problems of the form $\min_{\mathbf{x}} f(\mathbf{x}) + h(\mathbf{A}\mathbf{x})$, with both terms convex and possibly nonsmooth, have the Fenchel dual $\max_{\mathbf{y}} -f^*(-\mathbf{A}^\top\mathbf{y}) - h^*(\mathbf{y})$ (Rockafellar 1970). Introducing $\mathbf{z} = \mathbf{A}\mathbf{x}$ and a quadratic penalty $\frac{\tau}{2}\|\mathbf{A}\mathbf{x}-\mathbf{z}\|^2$ in the Lagrangian gives the alternating direction method of multipliers. It minimizes over $\mathbf{x}$, then over $\mathbf{z}$ (a proximal step on h), then takes a dual ascent step on $\mathbf{y}$, and it converges for every $\tau > 0$. Boyd et al. (2011) use it to split problems across machines; Parikh and Boyd (2014) list closed-form proximal operators for common functions.

35.9.3.4 The Frank–Wolfe Method

Projected gradient descent needs a projection, which onto a nuclear-norm ball (matrices whose singular values sum to at most a bound) costs a full SVD. The Frank–Wolfe method (Frank and Wolfe 1956) minimizes the linearization over the set instead, $\mathbf{s}_k = \operatorname{argmin}_{\mathbf{s}\in C} \nabla f(\mathbf{x}_k)^\top\mathbf{s}$, and moves to $\mathbf{x}_{k+1} = (1-\gamma_k)\mathbf{x}_k + \gamma_k\mathbf{s}_k$ with $\gamma_k = 2/(k+2)$. Over the nuclear-norm ball this needs only the top singular pair of the gradient, and over the simplex only its smallest coordinate. The rate is $O(LD^2/k)$ on a set of diameter D, and the number $\nabla f(\mathbf{x}_k)^\top(\mathbf{x}_k - \mathbf{s}_k)$, an upper bound on $f(\mathbf{x}_k) - f^\star$, comes free with every step (Jaggi 2013).

35.9.4 Floating Point

35.9.4.1 Probabilistic Rounding-Error Analysis

The summation bounds of Section 35.8.3 grow like nu for n terms added in order, while measured errors are usually far smaller. Higham and Mary (2019) model the rounding errors as independent random variables with mean zero and obtain bounds of order $\sqrt{n}\,u$ that hold with high probability. The model fails when the rounding errors are correlated, as in that subsection's sum of 10^7 copies of 0.1, where long runs of additions round the same way.

35.9.4.2 Growth Factors and Iterative Refinement

Section 35.8.4 calls LU factorization with pivoting backward stable in practice. Its backward error is bounded by a low-degree polynomial in n times $\rho_n u$, where the *growth factor* ρ_n is the largest entry produced during elimination divided by the largest entry of $\mathbf{A}$. Wilkinson's example reaches $\rho_n = 2^{n-1}$, yet ρ_n stays small for almost all matrices met in practice, and QR factorization removes the worst case at about twice the cost (Higham 2002; Trefethen and Bau 1997). Mixed-precision iterative refinement factors $\mathbf{A}$ in fp32, computes residuals in fp64, and solves for corrections with the cheap factors; the last exercise of Section 35.8 finds how ill conditioned $\mathbf{A}$ may be before it fails. Solving each correction with GMRES, an iterative solver for general linear systems, preconditioned by the low-precision factors, extends it to far worse conditioned matrices (Carson and Higham 2018).

Chapter 36

Probability and Statistical Learning

The main text uses probability constantly and seldom stops to argue about it. It adopts squared error and cross-entropy as losses (Section 2.1, Section 3.1), reports an accuracy measured on a few thousand test examples, and keeps the best of many configurations by their validation scores (Section 2.6). Each step is an inference from a finite sample. This chapter supplies the arguments.

One example runs through it: a coin that lands heads 9 times in 13 flips. We would estimate its bias as $9/13 \approx 0.69$, yet a fair coin does at least this well more than one time in eight. A test accuracy is the same coin, flipped once per test question, and the same doubt applies to it.

Resolving such doubts takes four objects: a distribution, which describes how outcomes vary; an estimator, which turns a sample into a number; a posterior, which describes what we believe about the parameters after seeing the data; and a generalization bound, which says when performance on a sample controls performance on new data. The middle two are easily confused: the scatter of an estimate over the samples we did not draw is not the spread of a posterior given the sample we have, although the two often produce similar error bars.

The chapter takes them in order. Distributions need a language first: densities for one quantity (Section 36.1), then for several at once, together with the rule for what a transformation does to a density (Section 36.2). With it we meet the distributions that have names (Section 36.3) and find that nearly all of them share one form, the exponential family (Section 36.4). That common form is why linear, logistic and softmax regression all train with the gradient "prediction minus observation".

Estimators occupy the middle of the chapter. Section 36.5 supplies error bars, and Section 36.6 derives the losses of the main text from one principle. The Fisher information connects the two: it is the expected curvature of the negative log-likelihood, and its inverse is the variance of the fitted parameters. Posteriors enter through models with hidden variables, whose likelihood can only be bounded from below (Section 36.7), and Section 36.8 computes with posteriors that have no closed form. Guarantees come last: Hoeffding's inequality (Section 36.9), valid at every sample size, and what remains of it once the model

is chosen after seeing the data (Section 36.10), the mathematics behind Chapter 3.6. One classifier, fitted by counting pixels, then puts the first three objects to work on real images (Section 36.11).

The opening sections repeat parts of Section 1.6, now with proofs and for densities, and a reader who knows them can skim. Bayesian computation and naive Bayes can be skipped at first; no other section needs them. Entropy and the KL divergence appear only as tools; their theory is in Chapter 37. Measure theory, time series and nonparametric Bayesian models are left out.

Resources and Further Reading

The following references accompany the material this chapter teaches; topics it does not teach are collected in Section 36.12.

Books

- Introduction to Probability — Blitzstein & Hwang: introduces random variables, expectation, conditioning and the named distributions of Section 36.1 and Section 36.3; a free PDF accompanies Harvard's Stat 110.
- Mathematics for Machine Learning — Deisenroth, Faisal & Ong: Chapter 6 gives a self-contained review of probability and distributions, including the exponential family of Section 36.4.
- All of Statistics — Wasserman: a compact treatment of estimation, testing, confidence intervals and the bootstrap for computer scientists, covering Section 36.5 and Section 36.6; the companion page includes errata, code, and datasets.
- Computer Age Statistical Inference — Efron & Hastie: discusses the bootstrap, large-scale testing, and the relation between frequentist and Bayesian inference, for Section 36.5.
- Pattern Recognition and Machine Learning — Bishop: covers the exponential family, mixture models and EM (Section 36.7), and the Laplace and variational approximations of Section 36.8; Microsoft Research provides a free PDF.
- Probabilistic Machine Learning: An Introduction — Murphy: develops machine learning through probability and decision theory, including MAP estimation (Section 36.6) and generative classifiers (Section 36.11), with a free draft and runnable notebooks.
- Information Theory, Inference, and Learning Algorithms — MacKay: connects inference with information theory, and its chapters on Laplace's method, Monte Carlo methods and variational methods accompany Section 36.8.
- Bayesian Data Analysis — Gelman et al.: treats priors, posteriors, Markov chain Monte Carlo and its diagnostics, and practical Bayesian workflow, for Section 36.8; the third edition is available as a free PDF.
- High-Dimensional Probability — Vershynin: develops concentration inequalities, sub-Gaussian variables and high-dimensional geometry, the material of Section 36.9.
- Understanding Machine Learning: From Theory to Algorithms — Shalev-Shwartz & Ben-David: derives uniform convergence, Rademacher complexity and margin bounds, as in Section 36.10; the authors provide a free PDF.

Courses and video lectures

- Statistics 110: Probability — Harvard (Blitzstein): provides lecture videos, problem sets, and more than 250 practice problems with solutions for Section 36.1 and Section 36.3.

- Stat 110 lecture videos — YouTube: contains the complete lecture series on probability, distributions, and conditioning.

Tutorials and notes

- Probability Theory Review — Stanford CS229 (Maleki & Do): reviews random variables, expectation, variance, Gaussian distributions, and multivariate probability for machine learning, a short companion to Section 36.1.

36.1 Random Variables

The probability of Section 1.6 was built by counting. A die has six faces, a label is one of ten classes, and the probability of an event is the sum of the probabilities of the outcomes in it. Most of the numbers a network works with are not like that. A pixel intensity, a weight drawn at initialization, the Gaussian noise that a diffusion model adds to an image: each can take any value in an interval, and there are too many values to list one by one. What is the probability of a single one of them?

Think of a dart thrown at a round board so carelessly that it is equally likely to land anywhere on the board, and suppose it comes to rest at half the radius from the center. Asked how likely that was, we would round the distance, say to between 0.45 and 0.55 radii, which happens about one time in ten. Asking for one more digit makes the answer ten times smaller, and another digit makes it ten times smaller again. With all the digits the probability is zero, and yet the dart landed somewhere.

The way out is to stop asking for the probability of a point and to ask how fast probability accumulates near it. Each time we narrow the interval tenfold, its probability falls tenfold, so the ratio of probability to width stays fixed. This probability per unit length is the *density*. With it, every tool of discrete probability carries over to quantities that vary continuously, with sums replaced by integrals.

Part of what follows is Section 1.6 again, now with proofs and with densities: the mean and the variance were motivated there, and we do not motivate them a second time. New are the density itself, a sampler built from the cumulative distribution function, the inequalities of Markov and Chebyshev, and a density whose tails are so heavy that both fail. The two results to remember are that the density is the derivative of the cumulative distribution function Equation 36.5 and Markov's inequality Equation 36.13, from which the other tail bounds of this chapter descend. We follow the dart throughout, one number at a time. Its two coordinates, and what happens to a density when a variable is transformed, wait for Section 36.2.

36.1.1 Densities

We build the density in three steps: at a point, as the limit of ever narrower bins; over an interval, as an area; and through the function that accumulates the area, which will also tell us how to draw samples.

36.1.1.1 From Bins to a Density

To make "probability per unit length" precise, we first need the probability that the dart lands within a given range of distances. Let R be the dart's distance from the center of a board of radius 1. The dart lands within distance r of the center when it lands in the disk of

radius r. Since every part of the board is equally likely, the probability is that disk's share of the board's area:

$$P(R \le r) = \frac{\pi r^2}{\pi} = r^2, \qquad 0 \le r \le 1.$$

Half the radius holds only a quarter of the probability, because it holds only a quarter of the area. A bin of width ϵ centered at 0.5 has probability $(0.5+\epsilon/2)^2 - (0.5-\epsilon/2)^2 = \epsilon$, which gives the numbers of the opening:

bin	width ϵ	probability	probability / ϵ
$[0.45, 0.55]$	0.1	0.1	1
$[0.495, 0.505]$	0.01	0.01	1
$[0.4995, 0.5005]$	0.001	0.001	1

Each tenfold narrowing divides the probability by ten, and the ratio does not move. The value at which the ratio settles is the quantity we name. The *probability density* of a random variable X at x is the probability per unit length near x,

$$p(x) = \lim_{\epsilon \downarrow 0} \frac{P(x \le X < x + \epsilon)}{\epsilon},$$

wherever this limit exists. Equivalently, the probability of a short interval is its length times the density,

$$P(x \le X < x + \epsilon) \approx \epsilon\, p(x), \tag{36.1}$$

with an error that shrinks faster than ϵ. This describes the density one point at a time, which is all we need for the moment. The definition proper speaks of intervals and follows in the next subsection. For the dart at a distance r in $(0, 1)$, the interval $[r, r+\epsilon)$ has probability $(r+\epsilon)^2 - r^2 = 2r\epsilon + \epsilon^2$. The ratio to the width is $2r + \epsilon$, and letting $\epsilon \to 0$,

$$p(r) = 2r, \qquad 0 \le r \le 1.$$

The density of the distance grows linearly from the center to the rim, as the circumference of the circle at radius r does. At $r = 0.5$ the density is 1, the ratio of the table.

A density is not a probability. The dart's density at $r = 0.9$ is 1.8, and a variable that is equally likely to land anywhere in $[0, 0.1]$ has density 10 there, since it puts probability 1 on an interval of length 0.1. A density can take any non-negative value, but it can exceed 10 only on a set of total length less than $1/10$, or the probability of that set would exceed one. A density is a rate, like a speed. A speed tells how fast distance accumulates, not how far the car has gone. In the same way, the dart's density of 1.8 at $r = 0.9$ tells how fast probability accumulates as r grows past 0.9, not how much probability sits at 0.9.

A point has no length, so it has no probability. For every $\epsilon > 0$ the point x lies in $[x, x+\epsilon)$, so $P(X = x)$ is at most about $\epsilon\, p(x)$, which goes to zero with ϵ. So, for a variable with a density,

$$P(X = x) = 0 \qquad \text{for every } x. \tag{36.2}$$

This resolves the dart's puzzle: the dart lands at distance 0.5 with probability zero, as it does at every other given distance, and yet it lands somewhere. Probability zero does not mean impossible.

A point that carries positive probability is called an *atom*. A variable with a density has no atoms, a discrete variable such as a die roll has only atoms, and some variables have both. The output of a ReLU unit whose input is standard normal is 0 with probability $\frac{1}{2}$, an atom, and spreads the other half over $(0, \infty)$ with a density (Exercise 2 works it out).

The table came from the area formula. A simulation needs no formula. A computer throws darts uniformly at the board by throwing them uniformly at the square $[-1, 1]^2$ around it and keeping those that land on the board, which leaves every part of the board equally likely. We count the fraction of darts in each of the three bins of the table and divide by the bin's width.

```
rng = np.random.default_rng()
xy = rng.uniform(-1, 1, size=(4 * 10**6, 2))          # darts on the square
on_board = (xy**2).sum(axis=1) <= 1                    # keep those on the board
r_dart = np.linalg.norm(xy[on_board], axis=1)          # distances R
for width in [0.1, 0.01, 0.001]:
    prob = np.mean(np.abs(r_dart - 0.5) <= width / 2)  # fraction in the bin
    print(f'width {width:<5}  probability {prob:.5f}  ratio {prob / width:.3f}')
```

```
width 0.1    probability 0.09983  ratio 0.998
width 0.01   probability 0.01002  ratio 1.002
width 0.001  probability 0.00099  ratio 0.990
```

The fraction falls tenfold with each tenfold narrowing, and the ratio stays near $1 = p(0.5)$. What the table could not show is how noisy a narrow bin is. About 3.1 million of the four million darts land on the board, and only about $3{,}000$ of those fall in a bin of width 0.001, so the last ratio wobbles by a few percent from run to run.

36.1.1.2 Probability as Area

The density was read off from the probabilities of short intervals. We now go back the other way: given a density, what is the probability of a long interval? Cut the real line into pieces $[\epsilon i, \epsilon(i+1))$ of width ϵ. By Equation 36.1, each piece has probability about $\epsilon\, p(\epsilon i)$, and the probabilities of disjoint pieces add. A sum of widths times heights is a Riemann sum, and as the width shrinks it tends to the area under the curve, the integral (Section 34.6). Since X takes some value, all pieces together have probability one:

$$1 = P(X \in \mathbb{R}) \approx \sum_i \epsilon\, p(\epsilon i) \;\xrightarrow[\epsilon \to 0]{}\; \int_{-\infty}^{\infty} p(x)\, dx.$$

Summing only the pieces between a and b suggests that the probability of an interval is the area under the density above it. The argument is a motivation and not a proof: there are about $1/\epsilon$ pieces, and their errors, each small compared with ϵ, add up to something small only if they are small uniformly in x. We therefore turn the conclusion into the definition.

Definition. A random variable X *has the density* p when $p \geq 0$ and, for all $a < b$,

$$P(X \in (a, b]) = \int_a^b p(x)\, dx. \tag{36.3}$$

Letting $a \to -\infty$ and $b \to \infty$ shows that the total area under a density is one. These two properties, $p \geq 0$ and total area 1, are all that a function needs in order to be a density. Figure 36.1 shows the definition for a density with two bumps. The dart passes the test: for $0 \leq a < b \leq 1$ the integral $\int_a^b 2r\, dr = b^2 - a^2$ is the probability $P(a < R \leq b)$ that the areas gave, so $2r$ is its density in this sense too. The probability of landing in the outer half of the radius is $\int_{0.5}^1 2r\, dr = 1 - 0.25 = 0.75$. Three quarters of the board lies more than half the radius from the center.

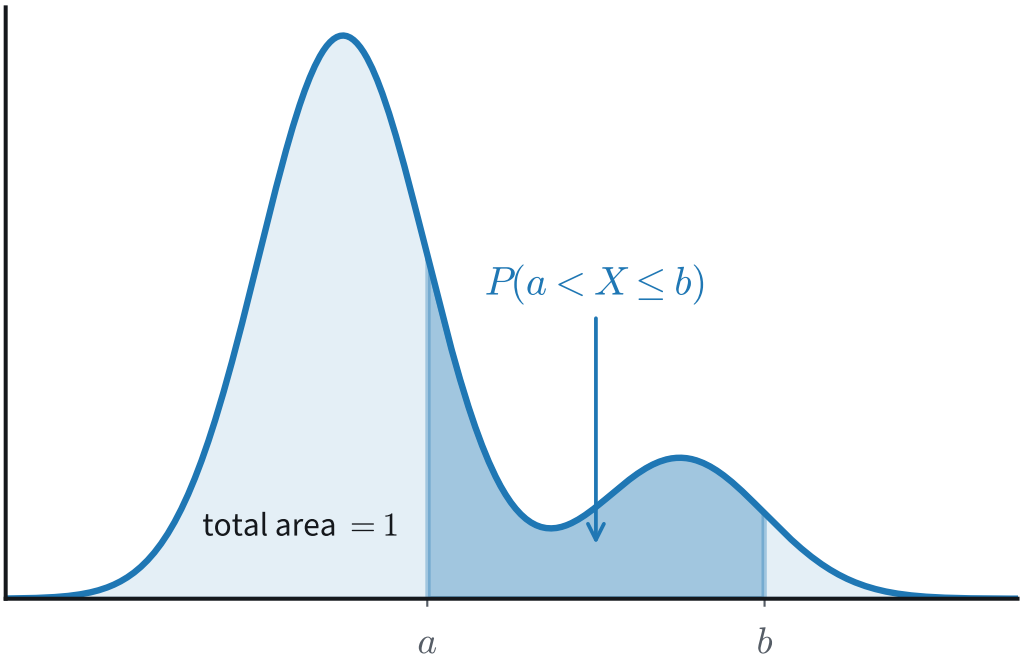


Figure 36.1: A probability density $p(x)$. The total area under the curve is 1; the probability that X lands in the interval from a to b is the area of the shaded slab above it, $\int_a^b p(x)\, dx$. The density itself is not a probability (it may exceed 1); only its integral over an interval is.

36.1.1.3 The Cumulative Distribution Function

Often we want a function whose values are probabilities, and one that also describes variables without a density, such as a die roll. The *cumulative distribution function* (c.d.f.) of X is the probability that X lands at or below x; when X has a density, it is the area under the density to the left of x:

$$F(x) = P(X \le x) = \int_{-\infty}^{x} p(t)\,dt. \tag{36.4}$$

So F accumulates probability from left to right. It rises from 0 as $x \to -\infty$ to 1 as $x \to \infty$. It never decreases, because the non-negative p only adds area, and when X has a density it is continuous. For the dart we have already computed it: $F(r) = 0$ for $r < 0$, $F(r) = r^2$ for $0 \le r \le 1$, and $F(r) = 1$ for $r > 1$. It is the function we started from, and the density $2r$ came from it.

The c.d.f. is defined for every random variable. A fair coin that reports 0 or 1 has

$$F(x) = \begin{cases} 0 & x < 0, \\ \frac{1}{2} & 0 \le x < 1, \\ 1 & x \ge 1, \end{cases}$$

a *staircase* that jumps by the probability of each atom. Since $F(x) = P(X \le x)$ includes the atom at x itself, F takes the upper value at each jump. In general every c.d.f. is *right-continuous*, $\lim_{y \downarrow x} F(y) = F(x)$. A variable with both kinds of probability has both features: the ReLU output above has a c.d.f. that is 0 for $x < 0$, jumps to $\frac{1}{2}$ at 0, and then rises smoothly to 1. One function describes all three kinds of variable.

When a density exists, the c.d.f. and the density carry the same information. Going from p to F is integration; going back is differentiation. This is where the limit of the bins returns, now as a theorem.

Proposition (density is the derivative of the c.d.f.). *If X has a density p that is continuous at x, then F is differentiable at x and*

$$F'(x) = p(x). \tag{36.5}$$

In particular, the probability per unit length $P(x \le X < x + \epsilon)/\epsilon$ tends to $p(x)$ as $\epsilon \downarrow 0$.

Proof. By Equation 36.4, F is the area-so-far function of p. The fundamental theorem of calculus says that the derivative of an area-so-far function is the integrand, at every point where the integrand is continuous (Section 34.6). For the second statement, the point x has probability at most $\int_{x-\delta}^{x} p(t)\,dt$ for every $\delta > 0$, and these integrals tend to zero with δ. Including or excluding an endpoint therefore changes nothing, $P(x \le X < x + \epsilon) = F(x + \epsilon) - F(x)$, and the ratio is a difference quotient of F. ■

For the dart, $F'(r) = 2r$ on $(0, 1)$ is the density we found from the bins. Figure 36.2 puts the two functions side by side: the shaded area $\int_a^b p$ on the left equals the rise $F(b) - F(a)$ on the right.

36.1.1.4 Quantiles and Inverse-Transform Sampling

A computer has a source of random numbers that are equally likely to fall anywhere between 0 and 1. To simulate the dart, or any other density, we need to turn these numbers into samples with the right distribution.

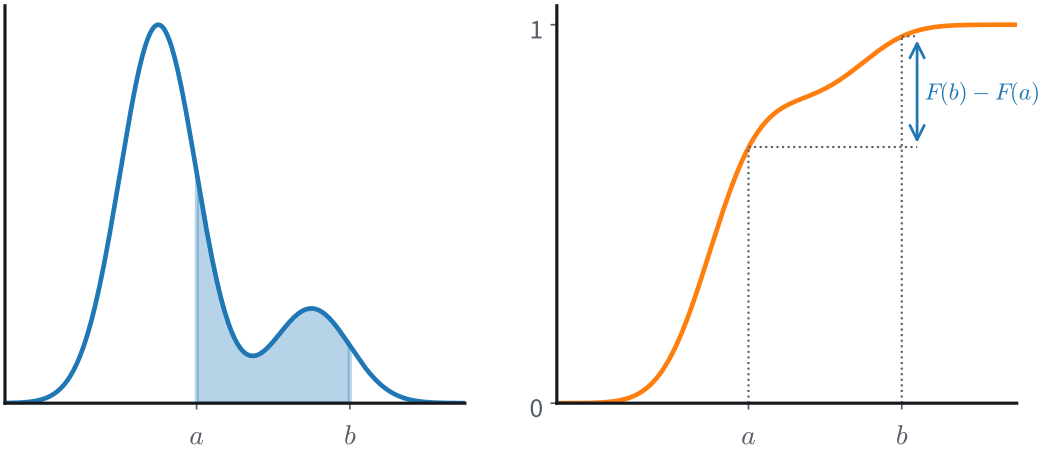


Figure 36.2: Left: a density p with the area over the interval from a to b shaded. Right: the matching c.d.f. $F(x) = \int_{-\infty}^{x} p$, an increasing curve from 0 to 1; the rise $F(b) - F(a)$ equals the shaded area on the left. The density is the slope of the c.d.f. $(F' = p)$ and the c.d.f. is the accumulated area under the density.

The source is the simplest density of all. A variable U is *uniform on* $[0, 1]$ when its density is 1 on $[0, 1]$ and 0 elsewhere. Every subinterval of $[0, 1]$ then has probability equal to its length, and the c.d.f. is $F_U(u) = u$ for $0 \le u \le 1$.

The c.d.f. maps a value to a probability. Going back, from a probability level u to the value below which that fraction of the probability lies, gives the *quantile function*. When F is continuous, and strictly increasing on the interval I where $0 < F < 1$, it maps I one-to-one onto $(0, 1)$, and its inverse $F^{-1} : (0, 1) \to I$ is the quantile function. Its values at $u = 0.25, 0.5, 0.75$ are the *quartiles*, and the middle one is the *median*. For the dart, $F(r) = r^2$ gives $F^{-1}(u) = \sqrt{u}$, with quartiles 0.5, 0.707 and 0.866. The median dart lands at 0.707, farther out than half the radius, because three quarters of the board lies beyond 0.5.

The quantile function is also a sampler.

Proposition (inverse-transform sampling). *Let F be a continuous c.d.f. that is strictly increasing on the interval I where $0 < F < 1$, and let U be uniform on $[0, 1]$. Then $X = F^{-1}(U)$ has c.d.f. F.*

Proof. With probability one, U lies in $(0, 1)$, so X is defined and lies in I. For x in I, the events $F^{-1}(U) \le x$ and $U \le F(x)$ coincide, because F is increasing on I. The uniform gives the second event probability $F(x)$, the length of $[0, F(x)]$:

$$P(F^{-1}(U) \le x) = P(U \le F(x)) = F(x).$$

For x to the left of I, $F(x) = 0$ and $X > x$ always; for x to the right of I, $F(x) = 1$ and $X \le x$ always. ■

The hypothesis allows a c.d.f. that is flat outside an interval, as the dart's is ($F = 0$ for $r < 0$ and $F = 1$ for $r > 1$). The *generalized inverse* $F^{-}(u) = \min\{x : F(x) \ge u\}$ needs

no hypothesis at all and samples staircases too (Exercise 4).

For the dart, $R = \sqrt{U}$. The tempting shortcut $R = U$ is wrong, and we can say by how much before running anything: it puts half of the darts within $r < 0.5$, where only a quarter belong. Figure 36.3 shows the construction for the two-bump density. Where the density is high, the c.d.f. is steep, so an even spread of levels lands in a narrow range of values.

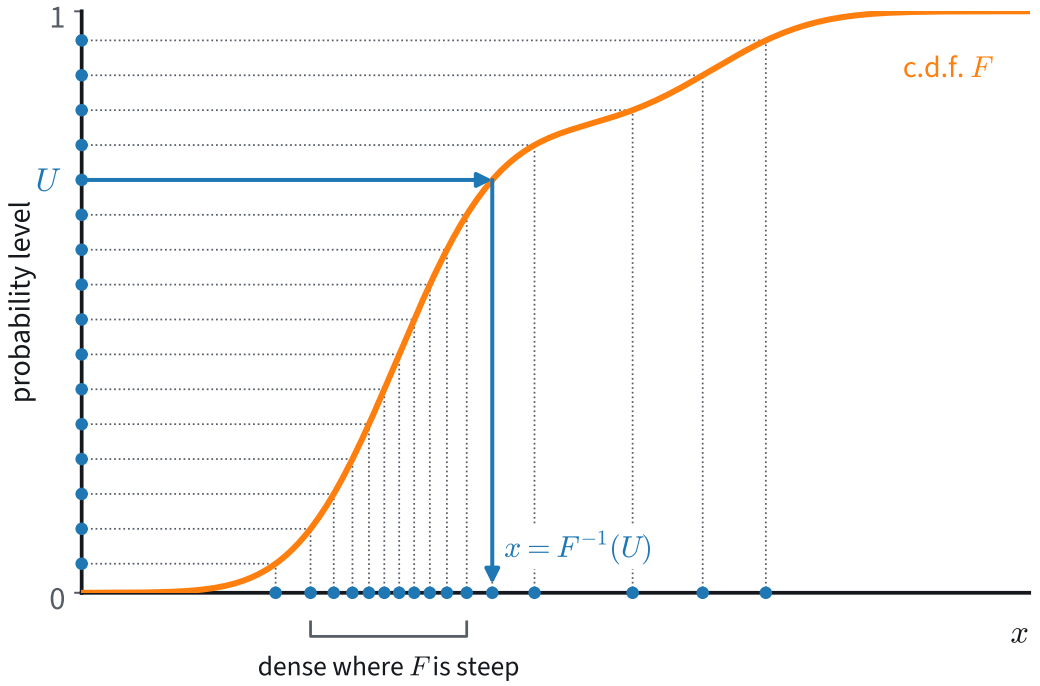


Figure 36.3: Inverse-transform sampling. Uniform levels U on the vertical axis (left) reflect through the c.d.f. F down to values $x = F^{-1}(U)$ on the horizontal axis. Where F is steep, that is, where the density $p = F'$ is large, a uniform spread of levels is compressed into a dense cluster of samples, so the histogram of reflected points reproduces p.

We now have two ways to produce the dart's distance, the darts thrown at the board above and the samples $\sqrt{U}$. The cell compares them, with the shortcut alongside.

```
u = rng.uniform(size=10**6)
r = np.sqrt(u)                                          # R = F^{-1}(U)
print(f'P(R < 0.5): darts {np.mean(r_dart < 0.5):.3f}, sqrt(U) '
      f'{np.mean(r < 0.5):.3f}, naive U {np.mean(u < 0.5):.3f}')
print(f'mean distance: darts {r_dart.mean():.3f}, sqrt(U) {r.mean():.3f}')
```

```
P(R < 0.5): darts 0.250, sqrt(U) 0.250, naive U 0.500
mean distance: darts 0.667, sqrt(U) 0.667
```

The two agree. Both put a quarter of their distances below 0.5, and both have a mean distance close to 0.667, a value the next subsection derives. The shortcut puts half of its distances there: it spreads distances evenly, whereas the board has more area far from the

center.

36.1.2 Mean and Variance

A density describes the whole distribution, but two numbers often say enough: where the distribution is centered, and how far it spreads. Section 1.6 introduced both for discrete variables and said why we want them. Here we extend them to densities and prove their algebra.

36.1.2.1 The Mean

The mean is the average of the values of X, each weighted by its probability. For a variable with a density, slice the line into bins of width ϵ as before: the bin at ϵi contributes its value ϵi times its probability, about $\epsilon\, p(\epsilon i)$, and the sum tends to an integral. The *mean*, or *expectation*, of X is

$$\mu_X = \mathbb{E}[X] = \sum_i x_i\, p_i \quad \text{(discrete)}, \qquad \mu_X = \mathbb{E}[X] = \int_{-\infty}^{\infty} x\, p(x)\, dx \quad \text{(density)}. \tag{36.6}$$

As in the discrete case, the mean is a balance point: a rod with mass density p along its length balances at μ_X. The definition requires $\sum_i |x_i|\, p_i < \infty$ in the discrete case and $\int |x|\, p(x)\, dx < \infty$ for a density. Otherwise the mean does not exist, and below we meet a density for which it fails. The same weighting averages any function of X,

$$\mathbb{E}[g(X)] = \sum_i g(x_i)\, p_i, \qquad \mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(x)\, p(x)\, dx,$$

again provided the sum or integral of $|g|$ is finite. We weight the values of g by the probabilities of X, without first working out the distribution of the new variable $g(X)$. The rule is called the *law of the unconscious statistician*, because it is easy to use without noticing that it is a theorem rather than the definition of $\mathbb{E}[g(X)]$. In general $\mathbb{E}[g(X)]$ differs from $g(\mathbb{E}[X])$; for convex g, Jensen's inequality (Section 35.3.3.1) says which of the two is larger.

For the dart,

$$\mathbb{E}[R] = \int_0^1 r \cdot 2r\, dr = \tfrac{2}{3}.$$

The mean distance is 0.667, not 0.5: the board has more area far from the center than near it. The mean lies a little below the median 0.707, because the few darts near the center pull the average down more than they move the median.

The rule we use most often is that the mean is linear. Section 1.6 used it; here we prove it.

Proposition (linearity of expectation). *For random variables X, Y whose means exist and constants c, d,*

$$\mathbb{E}[cX + d] = c\,\mathbb{E}[X] + d, \qquad \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]. \tag{36.7}$$

Proof. Both follow from the linearity of the sum (or integral) that defines the mean. For the first, $\mathbb{E}[cX+d] = \sum_i (cx_i + d)p_i = c\sum_i x_i p_i + d\sum_i p_i = c\,\mathbb{E}[X] + d$, using $\sum_i p_i = 1$. For the second, let p_{ij} be the probability that $(X, Y) = (x_i, y_j)$, the joint distribution of the pair as in Section 1.6. Split the sum and recognize the marginals $\sum_j p_{ij} = P(X = x_i)$ and $\sum_i p_{ij} = P(Y = y_j)$:

$$\mathbb{E}[X+Y] = \sum_{i,j} (x_i + y_j)\, p_{ij} = \sum_{i,j} x_i\, p_{ij} + \sum_{i,j} y_j\, p_{ij} = \mathbb{E}[X] + \mathbb{E}[Y].$$

For densities the same computation runs with integrals over a joint density, the continuous counterpart of p_{ij}, which Section 36.2 defines. ■

The second identity needs no independence: the mean of a sum is the sum of the means however strongly X and Y depend on each other. The average loss over a dataset is a sum of this kind.

36.1.2.2 Variance and Standard Deviation

The mean says nothing about spread. As in Section 1.6, we measure spread by the average squared distance from the mean, the *variance*

$$\sigma_X^2 = \operatorname{Var}(X) = \mathbb{E}\big[(X - \mu_X)^2\big], \tag{36.8}$$

which is defined when $\mathbb{E}[X^2] < \infty$. Expanding the square and applying linearity Equation 36.7, with μ_X treated as the constant it is, gives the form that is usually easier to compute, the mean square minus the square of the mean:

$$\operatorname{Var}(X) = \mathbb{E}\big[X^2 - 2\mu_X X + \mu_X^2\big] = \mathbb{E}[X^2] - 2\mu_X\,\mathbb{E}[X] + \mu_X^2 = \mathbb{E}[X^2] - \mu_X^2. \tag{36.9}$$

For a variable with a density, the mean and the variance are therefore the two integrals

$$\mu_X = \int_{-\infty}^{\infty} x\, p(x)\, dx, \qquad \sigma_X^2 = \int_{-\infty}^{\infty} x^2 p(x)\, dx - \mu_X^2. \tag{36.10}$$

For the dart, $\mathbb{E}[R^2] = \int_0^1 r^2 \cdot 2r\, dr = \frac{1}{2}$, so $\operatorname{Var}(R) = \frac{1}{2} - \frac{4}{9} = \frac{1}{18} \approx 0.056$. For the uniform U on $[0,1]$, $\mu_U = \int_0^1 u\, du = \frac{1}{2}$ and $\operatorname{Var}(U) = \frac{1}{3} - \frac{1}{4} = \frac{1}{12}$.

Rescaling units and centering data are affine maps. Under them the shift drops out of the variance and the scale comes out squared. By linearity the mean maps as $\mu_{cX+d} = c\mu_X + d$, so the deviation is $(cX + d) - \mu_{cX+d} = c(X - \mu_X)$, and squaring and taking the mean gives

$$\operatorname{Var}(cX + d) = c^2 \operatorname{Var}(X). \tag{36.11}$$

Two further facts complete the list. A variance is never negative, and it is zero if and only if X is constant with probability one: the mean of the non-negative variable $(X - \mu_X)^2$ is

zero only if that variable is zero with probability one. And variances of independent variables add (Section 1.6); Section 36.2.2.1 proves this together with the term that replaces independence.

The variance carries the square of the units of X, dollars squared for a profit in dollars. Its square root, the *standard deviation*

$$\sigma_X = \sqrt{\mathrm{Var}(X)}, \tag{36.12}$$

has the units of X again, and $\sigma_{cX+d} = |c|\,\sigma_X$. For the dart, $\sigma_R = \sqrt{1/18} \approx 0.236$ in units of the board's radius.

36.1.3 Tails

The standard deviation suggests a scale for typical deviations from the mean. How much probability can lie far out, if all we know is the mean, or the mean and the variance? Two inequalities answer this for every distribution at once. They are crude, and they are also the best possible, as one example on three points will show. A bell-shaped density with neither a mean nor a variance then shows what the two assumptions were protecting us from.

36.1.3.1 Markov's and Chebyshev's Inequalities

The first inequality uses only the mean, and applies to variables that are never negative. Its proof is one line, and nearly every tail bound in this chapter is obtained by applying it to a well-chosen function of the variable.

Proposition (Markov's inequality). *If $X \geq 0$ has a mean and $t > 0$, then*

$$P(X \geq t) \leq \frac{\mathbb{E}[X]}{t}. \tag{36.13}$$

Proof. Let $\mathbf{1}_{X \geq t}$ be the indicator variable that is 1 when $X \geq t$ and 0 otherwise. Then $X \geq t\,\mathbf{1}_{X \geq t}$ for every outcome: when $X \geq t$ the right-hand side is $t \leq X$, and otherwise it is $0 \leq X$. Taking means of both sides and using $\mathbb{E}[\mathbf{1}_{X \geq t}] = P(X \geq t)$ gives $\mathbb{E}[X] \geq t\,P(X \geq t)$. ■

Geometrically, on $[0, \infty)$ the step function that is 0 below t and t from t on lies under the line $y = x$, so its mean, $t\,P(X \geq t)$, lies under the mean of X. With t ten times the mean, at most a tenth of any non-negative population exceeds ten times its average, whether it is incomes, file sizes or gradient norms.

Markov's inequality uses only the mean, so it is crude. Applied to the squared deviation $(X - \mu_X)^2$, whose mean is the variance, it bounds both tails at once. The result is due to Bienaymé and to Chebyshev (Chebyshev 1867).

Proposition (Chebyshev's inequality). *For any X with finite, nonzero variance and any $\alpha > 0$,*

$$P(|X - \mu_X| \geq \alpha\sigma_X) \leq \frac{1}{\alpha^2}. \tag{36.14}$$

Proof. The variable $Z = (X - \mu_X)^2$ is non-negative with mean $\mathrm{Var}(X) = \sigma_X^2$, and the event $|X - \mu_X| \geq \alpha\sigma_X$ is the event $Z \geq \alpha^2\sigma_X^2$. Since $\sigma_X > 0$, the threshold $t = \alpha^2\sigma_X^2$ is positive,

and Markov's inequality Equation 36.13 gives $P(|X-\mu_X| \geq \alpha\sigma_X) \leq \sigma_X^2/(\alpha^2\sigma_X^2) = 1/\alpha^2$. ■

So at most 1% of any distribution lies ten or more standard deviations from its mean. The same argument with a threshold $\epsilon > 0$ in place of $\alpha\sigma_X$ gives a form that needs only a finite variance and also covers $\sigma_X = 0$:

$$P(|X - \mu_X| \geq \epsilon) \leq \frac{\text{Var}(X)}{\epsilon^2}.$$

For the dart, $\mu_R = 0.667$ and $\sigma_R = 0.236$. Two standard deviations below the mean is 0.195, and two above is beyond the board, so $P(|R - \mu_R| \geq 2\sigma_R) = P(R \leq 0.195) = 0.195^2 \approx 0.038$, far below the bound $\frac{1}{4}$.

Can a bound that holds for every distribution be improved? Not without further assumptions, and a distribution on three points shows it. Let X take the values $a-2$ and $a+2$ with probability q each, and a with probability $1-2q$, where $0 < q \leq \frac{1}{2}$. We call it the *three-atom example*. By symmetry its mean is a, and $\text{Var}(X) = \mathbb{E}[(X-a)^2] = 4q+4q = 8q$, so $\sigma_X = 2\sqrt{2q}$. Take $\alpha = 2$. The threshold $2\sigma_X = 4\sqrt{2q}$ shrinks as q does, while the outer atoms stay at distance 2. Figure 36.4 shows the three regimes. For $q > \frac{1}{8}$ the threshold exceeds 2, no atom lies that far out, and the probability of the event is 0. At $q = \frac{1}{8}$ the threshold equals 2, so $\sigma_X = 1$ and $a \pm 2$ lie two standard deviations from the mean. Both outer atoms count, the probability is $2q = \frac{1}{4}$, and the bound holds with equality. For $q < \frac{1}{8}$ both atoms lie beyond the threshold, and their total mass $2q$ is below $\frac{1}{4}$.

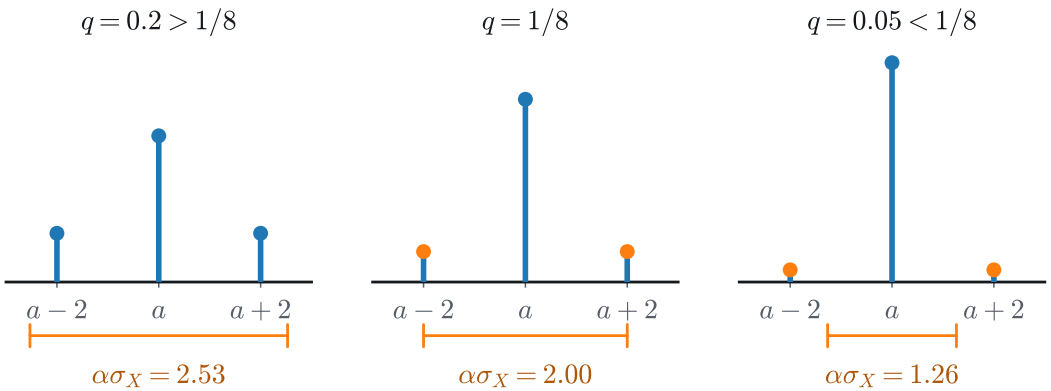


Figure 36.4: Chebyshev's bound on the three-atom example: masses q, $1-2q$, q drawn as vertical bars at $a-2$, a, $a+2$, with the threshold distance $\alpha\sigma_X = 4\sqrt{2q}$ for $\alpha = 2$ marked as a horizontal bracket from $a - \alpha\sigma_X$ to $a + \alpha\sigma_X$. The three panels show the regimes $q > 1/8$, $q = 1/8$ (where the bound is attained with equality), and $q < 1/8$. Orange atoms lie at or beyond the threshold and count toward the event $|X - a| \geq \alpha\sigma_X$.

The same example settles every $\alpha \geq 1$. Choose $q = 1/(2\alpha^2)$, which is at most $\frac{1}{2}$. Then $\sigma_X = 2\sqrt{2q} = 2/\alpha$, the threshold $\alpha\sigma_X = 2$ lands on the outer atoms, and $P(|X - a| \geq \alpha\sigma_X) = 2q = 1/\alpha^2$. Equality holds, so no constant smaller than $1/\alpha^2$ works for every distribution. For $\alpha < 1$ the bound exceeds 1 and says nothing.

Chebyshev's bound is the worst case over all distributions with a given variance. For the distributions we usually meet it is loose. A Gaussian with mean μ and standard deviation σ, the bell curve with density $e^{-(x-\mu)^2/(2\sigma^2)}/(\sigma\sqrt{2\pi})$ that Section 36.3.2.3 treats in full, puts much less probability beyond two and three standard deviations than the bound allows:

distance from the mean	Chebyshev's bound	Gaussian
2σ or more	25%	4.6%
3σ or more	11%	0.27%

The worst case puts all of its tail probability at a single distance, as the three-atom example does. Typical distributions spread their tails out and thin them quickly.

36.1.3.2 Heavy Tails

Chebyshev's inequality needs a finite variance, and the mean needs $\int |x|\,p(x)\,dx < \infty$. Both can fail for a density that looks harmless. The *Cauchy distribution* has the bell-shaped density

$$p(x) = \frac{1}{\pi(1+x^2)}.$$

It is a density, since $\int_{-\infty}^{\infty} \frac{dx}{1+x^2} = [\arctan x]_{-\infty}^{\infty} = \pi$. Unlike a Gaussian, whose tails fall off like $e^{-x^2/2}$, its tails fall off only like $1/(\pi x^2)$. The second moment fails first. Over the range $[-L, L]$,

$$\int_{-L}^{L} x^2 p(x)\,dx = \frac{2}{\pi}(L - \arctan L),$$

which is 5.43, 62.7 and 636 at $L = 10$, 100 and 1000 and grows without bound, because the integrand $x^2/(\pi(1+x^2))$ tends to $1/\pi$. So $\mathbb{E}[X^2] = \infty$, and the Cauchy distribution has no finite variance.

The mean fails in a subtler way. The integrand $x\,p(x)$ is odd, which tempts us to declare the mean 0 by symmetry, and indeed $\int_{-L}^{L} x\,p(x)\,dx = 0$ for every L. But the definition requires absolute convergence, and

$$\int_{-L}^{L} |x|\,p(x)\,dx = \frac{1}{\pi}\log(1+L^2)$$

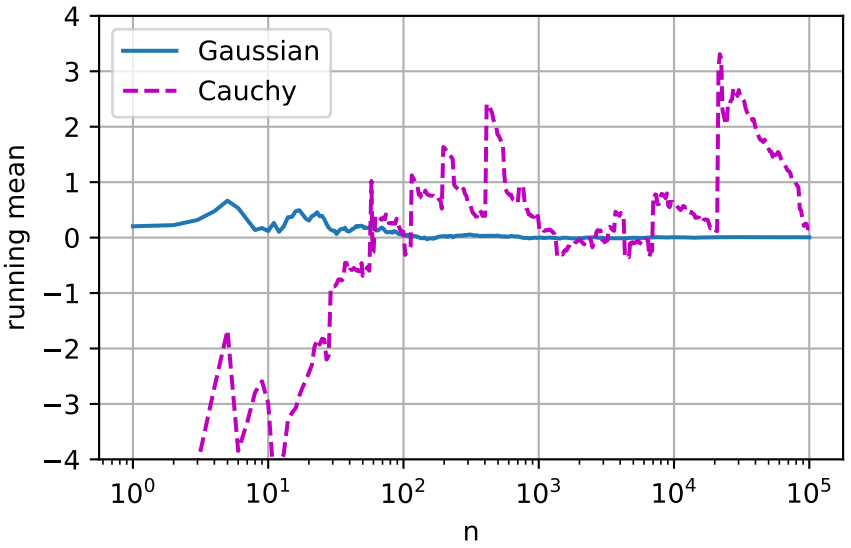
grows without bound. So the mean does not exist. Without absolute convergence the answer depends on how the limits are taken: over the lopsided range $[-L, 2L]$,

$$\int_{-L}^{2L} x\,p(x)\,dx = \frac{1}{2\pi}\log\frac{1+4L^2}{1+L^2} \xrightarrow[L\to\infty]{} \frac{\log 2}{\pi} \approx 0.221.$$

With no mean to converge to, what does an average of Cauchy draws do? One might still hope that an average of 1000 draws lies closer to 0 than a single draw, which lies farther than 1 from 0 with probability $P(|X| > 1) = 1 - \frac{2}{\pi} \arctan 1 = \frac{1}{2}$. The cell plots the running mean of 10^5 Cauchy draws next to that of 10^5 standard Gaussian draws, and then counts how many of 1000 such averages lie farther than 1 from 0.

```
n = 10**5
idx = np.unique(np.logspace(0, 5, 500).astype(int)) - 1  # log-spaced n
running_mean = lambda x: (np.cumsum(x) / np.arange(1, n + 1))[idx]
d2l.plot(idx + 1, [running_mean(rng.standard_normal(n)),
                   running_mean(rng.standard_cauchy(n))],
         'n', 'running mean', legend=['Gaussian', 'Cauchy'],
         xscale='log', ylim=[-4, 4], figsize=(5, 3))
averages = rng.standard_cauchy((1000, 1000)).mean(axis=1)
print(f'averages of 1000 Cauchy draws with |average| > 1: '
      f'{np.mean(np.abs(averages) > 1):.2f}')
```

```
averages of 1000 Cauchy draws with |average| > 1: 0.49
```



The Gaussian running mean settles near 0. The Cauchy running mean does not settle: it moves in jumps, each caused by a single huge draw, and a jump of order one can come at any n. Averaging did not help at all. About half of the averages of 1000 draws lie farther than 1 from 0, the same fraction as for single draws. In fact the average of n independent Cauchy draws is again a standard Cauchy variable, a fact we state without proof.

A distribution is called *heavy-tailed* when its tails decay more slowly than every exponential $e^{-\lambda t}$ with $\lambda > 0$. Power-law tails, with $P(|X| > t)$ decaying like $t^{-\alpha}$, are the standard example: they make every moment $\mathbb{E}[|X|^k]$ with $k \geq \alpha$ infinite, and the Cauchy has $\alpha = 1$. A heavy tail does not require infinite moments, though. The log-normal distribution of Section 36.2.3.1 is heavy-tailed, yet all of its moments are finite. Heavy tails matter in deep learning because averages over them converge slowly or not at all. The noise in minibatch gradients, for example, has been argued to be heavy-tailed (Simsekli et al. 2019), a claim

that Exercise 6 tests.

All of this concerned one number at a time, and the dart has two. Its coordinates turn out to be uncorrelated and yet far from independent, and to say what that means we need densities over the plane, which are the subject of Section 36.2.

36.1.4 Summary

A variable that varies continuously gives every single value probability zero, so its distribution is described by a density, the probability per unit length. The probability of an interval is the area under the density, and the c.d.f. accumulates that area. Markov's and Chebyshev's inequalities bound the tails of any distribution from its mean or variance; Chebyshev's bound is attained in the worst case but loose for typical distributions. The Cauchy distribution has neither a mean nor a variance, and averaging its draws does nothing. Markov's inequality is the result to carry forward: the tail bounds of Section 36.9 are all obtained from it.

36.1.5 Exercises

1. **Power-law tail.** Let X have the density $p(x) = 1/x^2$ for $x \geq 1$ and $p(x) = 0$ otherwise.
    1. Verify that p is a density, and compute $P(X > 2)$ and the c.d.f. $F(x)$ for every real x.
    2. Find the quantile function $F^{-1}(u)$ for $u \in (0, 1)$, the median, and the other two quartiles. Write down the inverse-transform sampler $X = F^{-1}(U)$ of Figure 36.3 for this distribution.
    3. Show that $\mathbb{E}[X] = \infty$. Clipping X at a level $c > 1$, as gradient clipping does for a gradient norm, gives $\min(X, c)$. Compute $\mathbb{E}[\min(X, c)]$, and describe how it grows with c.
    4. For the Cauchy density $p_C(x) = 1/(\pi(1 + x^2))$, derive the values of $\int_{-L}^{L} x\, p_C(x)\, dx$ and $\int_{-L}^{2L} x\, p_C(x)\, dx$ quoted in the text, and their limits as $L \to \infty$. What do the two limits say about the mean of the Cauchy distribution? Why can no such ambiguity arise for the mean of the non-negative variable X of part (a)?

2. **Rectified Gaussian.** Let $X \sim \mathcal{N}(0, 1)$ have the c.d.f. Φ, and let $Y = \max(0, X)$. This is the output of a ReLU unit with a standard normal pre-activation, a common model of a unit at initialization.
    1. Express the c.d.f. F_Y through Φ and sketch it. Where does Y have an atom, and what is its mass? Is Y discrete, does it have a density, or is it mixed?
    2. Explain why Y has no density. Which hypothesis of the change-of-variables proposition Equation 36.29 fails for the ReLU?
    3. Clipping to $[-c, c]$ for a threshold $c > 0$, as the hard-tanh activation does for $c = 1$, gives $C = \min(\max(X, -c), c)$. Write down the c.d.f. of C, and find the threshold c at which the atoms of C together hold 10% of the probability. The equation has no closed-form solution; use tabulated values of Φ, or `math.erf` with $\Phi(z) = \frac{1}{2}(1 + \mathrm{erf}(z/\sqrt{2}))$.
    4. Show that $\mathbb{E}[Y^2] = \frac{1}{2}$ and $\mathbb{E}[Y] = 1/\sqrt{2\pi}$. Hint: write $X^2 = Y^2 + \min(0, X)^2$ and use the symmetry of X. The He initialization of Section 4.4.2.2 multiplies the

weight variance by 2; which of these two moments does the factor compensate for?

3. code **Logistic distribution and calibration.** The logistic distribution has the c.d.f. $F(x) = \text{sigmoid}(x) = 1/(1+e^{-x})$.
   1. Find its density with Equation 36.5 and its quantile function. What are its quartiles?
   2. Let X have a continuous, strictly increasing c.d.f. F. Show that $F(X)$ is uniform on $[0,1]$. This is the converse of inverse-transform sampling.
   3. A probabilistic regression model predicts for each input i a logistic distribution for the target y_i, centered at the correct location μ_i but with a scale s, that is, the c.d.f. $\text{sigmoid}\big((y-\mu_i)/s\big)$. The actual errors $y_i - \mu_i$ follow the standard logistic distribution. Draw 10^4 errors with the quantile function from (a), compute $u_i = \text{sigmoid}\big((y_i - \mu_i)/s\big)$ for $s \in \{0.5, 1, 2\}$, and plot the three histograms of the u_i. Which shape reveals predicted distributions that are too narrow, and which reveals distributions that are too wide? Such histograms are a standard check of the calibration of probabilistic forecasts (Gneiting et al. 2007).

4. code **Sampling from a staircase.** ⊠ The inverse-transform sampler of Figure 36.3 assumes a c.d.f. that is continuous and strictly increasing where it lies strictly between 0 and 1. A categorical distribution, such as the softmax over the vocabulary from which a language model samples its next token, has a staircase c.d.f. instead. For any c.d.f. F, define the generalized inverse $F^{-}(u) = \min\{x : F(x) \geq u\}$ for $u \in (0,1)$. The set is nonempty and bounded below, because $0 < u < 1$ and F tends to 0 and 1 at $-\infty$ and ∞. Its infimum belongs to it, because every c.d.f. is right-continuous, so the minimum exists.
   1. Show that $F^{-}(u) \leq x$ holds exactly when $u \leq F(x)$. Conclude that $F^{-}(U)$ has the c.d.f. F when U is uniform on $[0,1]$.
   2. Let X take the value $k \in \{1, \ldots, K\}$ with probability p_k, and let $c_k = p_1 + \cdots + p_k$. Show that $F^{-}(u)$ is the smallest k with $c_k \geq u$. Implement the sampler with NumPy's `cumsum` and `searchsorted`, and check the frequencies of 10^5 draws for $\mathbf{p} = (0.1, 0.2, 0.3, 0.25, 0.15)$. What does it cost to draw m samples from one distribution over K classes? What does each draw cost when the distribution changes at every draw, as in autoregressive decoding?
   3. Apply the softmax to 50,000 logits, each 3 times a standard normal draw, store the probabilities in float32, and compute $c_K - 1$ from a float32 `cumsum`. How often does a uniform draw exceed c_K, and what does `searchsorted` return then? Give two ways to make the sampler safe.

5. **One-sided Chebyshev.** ⊠ Chebyshev's inequality Equation 36.14 bounds both tails at once. Cantelli's inequality bounds one tail: for X with mean μ and variance σ^2, and for every $t > 0$,

$$P(X - \mu \geq t) \leq \frac{\sigma^2}{\sigma^2 + t^2}. \tag{36.15}$$

   1. For every $u \geq 0$, apply Markov's inequality Equation 36.13 to $(X - \mu + u)^2$ to show that $P(X - \mu \geq t) \leq (\sigma^2 + u^2)/(t+u)^2$. Choose u to prove Cantelli's inequality.

2. Write the bound for $t = \alpha\sigma$, and compare it with Equation 36.14 at $\alpha = 1$ and $\alpha = 2$.
3. For given μ, σ and t, find a distribution on two points with mean μ and variance σ^2 that attains the bound.

6. [extended] **Averages of heavy tails.** ⊠⊠ Simsekli et al. (2019) argued that the gradient noise of SGD in deep networks is heavy-tailed and better described by an α-stable distribution, whose variance is infinite for $\alpha < 2$, than by a Gaussian. This project measures how a Monte Carlo average behaves in that regime and then tests the premise on a small model. The Pareto density $p(x) = \alpha x^{-\alpha-1}$ on $x \geq 1$, with $\alpha > 0$, contains the density of problem 1 as the case $\alpha = 1$.

   1. Generalize problem 1b and c to this family: show that $\mathbb{E}[X^k] < \infty$ exactly when $k < \alpha$, compute the mean μ for $\alpha > 1$, and write down an inverse-transform sampler.

   2. Let $1 < \alpha < 2$, so that the mean is finite but the variance is not. Show that the largest of n independent draws exceeds t with probability close to $nt^{-\alpha}$ when $nt^{-\alpha}$ is small, and find the scale t_n that the largest draw typically reaches. Argue heuristically that the error $|\bar{X}_n - \mu|$ of the sample mean is of order $n^{1/\alpha-1}$ rather than $n^{-1/2}$. The generalized central limit theorem that Simsekli et al. invoke makes this argument precise.

   3. For $\alpha \in \{1.5, 2.5, 3\}$ and $n \in \{10, 10^2, \dots, 10^5\}$, estimate the median of $|\bar{X}_n - \mu|$ over 400 repetitions, and fit the slope of the logarithm of the median error against $\log n$. Compare the slope at $\alpha = 1.5$ with the rate $n^{1/\alpha-1}$ from (b). Compare the slopes at $\alpha = 2.5$ and $\alpha = 3$ with the rate $n^{-1/2}$ that holds when the variance is finite.

   4. In each repetition, form the interval $\bar{X}_n \pm 1.96\,\hat{\sigma}_n/\sqrt{n}$, where $\hat{\sigma}_n$ is the sample standard deviation; for Gaussian averages it covers μ with probability about 95%. Record how often it covers μ. Explain the coverage at $\alpha = 1.5$ and $n = 10^5$, given that $\hat{\sigma}_n$ is finite in every run. What does this imply for error bars computed from heavy-tailed samples, such as minibatch gradients?

   5. The Hill estimator (Hill 1975) estimates the tail index from the k largest of m positive draws $X_{(1)} \geq X_{(2)} \geq \cdots \geq X_{(m)}$ as

$$\hat{\alpha}_k = \Big(\frac{1}{k} \sum_{i=1}^{k} \log \frac{X_{(i)}}{X_{(k+1)}} \Big)^{-1}. \tag{36.16}$$

      Check it on 10^5 Pareto draws for $\alpha \in \{1.5, 3\}$ and $k \in \{10^2, 10^3, 10^4\}$. Then train the softmax regression of Section 3.4. At the trained parameters, collect the coordinates of the difference between the minibatch gradient and the full-batch gradient over 1000 minibatches, for batch sizes 32 and 256. Plot $\hat{\alpha}_k$ of their absolute values against k for both batch sizes. Is the estimate below 2? How much does the conclusion depend on k?

36.2 Joint Distributions and Transformations

The dart of Section 36.1 was reduced to one number, its distance R from the center of a board of radius 1, with density $2r$. But a dart lands at a point, and a point has two coordinates, a horizontal X and a vertical Y. Most of what we model comes in such groups. An image is thousands of pixels whose values depend on each other, and every supervised problem is a pair, an input and a target. One density per variable cannot describe a group, because it is silent on how the variables move together.

Two questions occupy this section. The first is what one variable tells us about another. If the dart lands near the top of the board, at $Y = 0.9$, then X must lie within 0.44 of the center line, so Y certainly carries information about X. Yet we will find that the best guess for X is 0 whatever Y is, and that the covariance of the two is zero. Dependence, predictability and correlation are three different things, and it takes the density of the pair to tell them apart. The second question is what a transformation does to a density, as when a layer acts on its input or a sampler on its noise. The dart's distance can be drawn as $R = \sqrt{U}$ from a uniform U, and the square root has turned a flat density into $2r$. Evidently a map does more to a density than relabel its axis.

Three results carry most of the weight later in the book: the law of total variance Equation 36.21, the conditional mean as the best squared-error predictor Equation 36.22, and the change of variables Equation 36.30. The covariance matrix and the remark that a correlation is a cosine can be skipped on a first reading.

```
rng = np.random.default_rng()
```

36.2.1 Joint and Conditional Densities

Everything that can be asked about a pair is answered by one function of two arguments. We define it, and then extract three things from it: the distribution of one coordinate alone, the distribution of one coordinate once the other is known, and the average of one given the other, which is what a regression model estimates.

36.2.1.1 Joint and Marginal Densities

The density extends to two variables with small squares in place of short intervals. The *joint density* $p_{X,Y}$ of a pair (X, Y) is probability per unit area: a small square of side ϵ at (x, y) has probability about its area times the density,

$$P\big(X \in [x, x+\epsilon],\ Y \in [y, y+\epsilon]\big) \approx \epsilon^2\, p_{X,Y}(x, y).$$

As for one variable, slicing a region into such squares leads to the property that we take as the definition: $p_{X,Y} \geq 0$, and $P((X, Y) \in \mathcal{D}) = \iint_{\mathcal{D}} p_{X,Y}(x, y)\, dx\, dy$ for a region $\mathcal{D}$ of the plane, so that the total volume under the joint density is one. For n variables the same holds with n-dimensional boxes.

The dart's position has the joint density $p_{X,Y}(x, y) = 1/\pi$ on the unit disk and 0 outside, since every part of the board is equally likely and the board has area π. Although it is constant on the disk, it is not a function of x times a function of y: the edge of the board ties the two coordinates together.

Often we hold a joint density but want the distribution of one coordinate alone, its *marginal distribution*. A thin vertical strip $X \in [x, x+\epsilon]$ has probability about $\epsilon\, p_X(x)$. Slicing the strip into squares in y and adding their probabilities,

$$\epsilon\, p_X(x) \approx \sum_i P(X \in [x, x+\epsilon],\ Y \in [\epsilon i, \epsilon(i+1)]) \approx \sum_i \epsilon^2\, p_{X,Y}(x, \epsilon i).$$

Cancelling one ϵ and recognizing the remaining sum as a Riemann sum in y,

$$p_X(x) = \int_{-\infty}^{\infty} p_{X,Y}(x, y)\, dy. \tag{36.17}$$

The marginal density at x is the joint density integrated up the vertical strip at x, as in Figure 36.5. We *integrate out* the variable we do not care about. For the dart, the strip at x crosses the board from $-\sqrt{1-x^2}$ to $\sqrt{1-x^2}$, so

$$p_X(x) = \int_{-\sqrt{1-x^2}}^{\sqrt{1-x^2}} \frac{1}{\pi}\, dy = \frac{2}{\pi}\sqrt{1-x^2}, \qquad -1 \le x \le 1,$$

a semicircle. As a check, $\int_{-1}^{1} \sqrt{1-x^2}\, dx$ is the area of half the unit disk, $\pi/2$, so the marginal integrates to 1. The joint density is flat, but the marginal is not: a strip through the middle of the board is longer than one near the edge.

36.2.1.2 Conditional Densities and Independence

Once we learn that $Y = y$, how should the density of X change? For discrete variables the answer is $P(X = x \mid Y = y) = P(X = x, Y = y)/P(Y = y)$. With densities this formula divides zero by zero, because $P(Y = y) = 0$. The way out is the one that defined the density. We condition on a thin horizontal strip $Y \in [y, y+\epsilon]$, which has positive probability, and let the strip shrink. By the square approximation and the one-variable rule,

$$P(X \in [x, x+\epsilon] \,\big|\, Y \in [y, y+\epsilon]) \approx \frac{\epsilon^2\, p_{X,Y}(x, y)}{\epsilon\, p_Y(y)} = \epsilon\, \frac{p_{X,Y}(x, y)}{p_Y(y)}.$$

Inside the strip, the probability of a short x-interval is its length times $p_{X,Y}(x, y)/p_Y(y)$. This ratio is the probability per unit length of X given $Y = y$, and we define the *conditional density*

$$p_{X|Y}(x \mid y) = \frac{p_{X,Y}(x, y)}{p_Y(y)}, \qquad p_Y(y) > 0. \tag{36.18}$$

For each y it is a density in x: it is non-negative, and by the marginal rule Equation 36.17 it integrates to $p_Y(y)/p_Y(y) = 1$. Geometrically, we cut the joint density along the line $Y = y$ and rescale the profile to unit area. For the dart, the line $Y = y$ crosses the board on $[-\sqrt{1-y^2}, \sqrt{1-y^2}]$, and

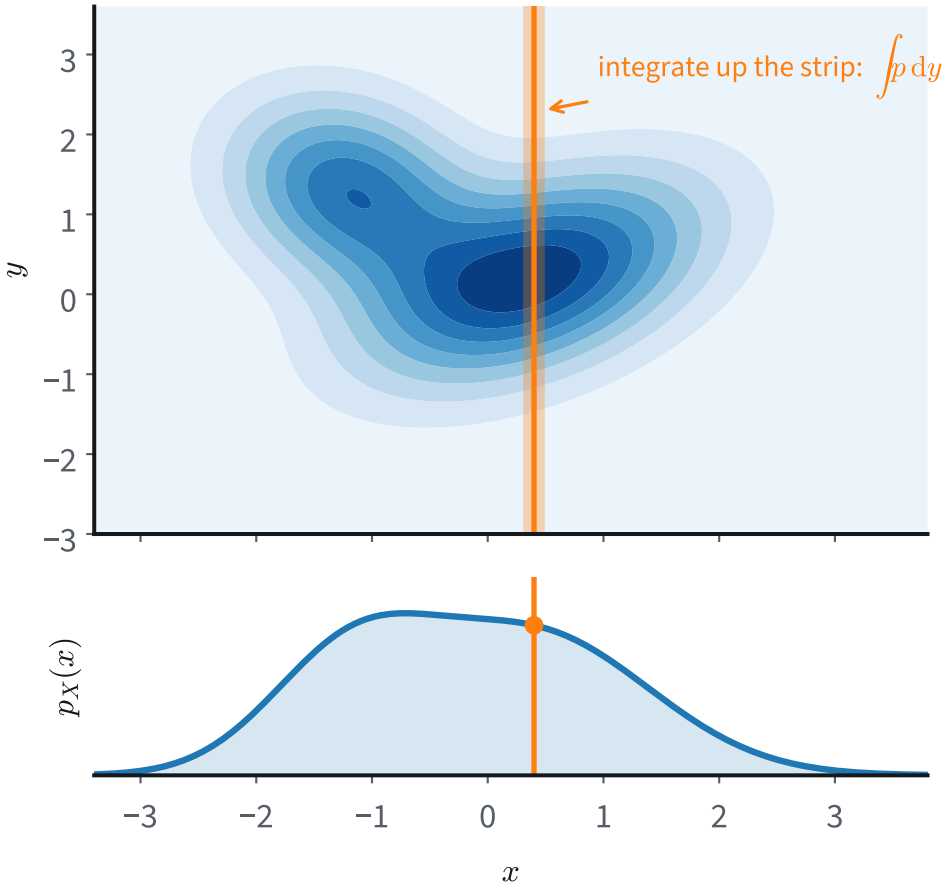


Figure 36.5: A continuous joint density $p_{X,Y}(x, y)$ shown by its contours, with one vertical strip at x highlighted. Integrating the joint density up that strip, over all y, gives the height of the marginal density $p_X(x)$ plotted along the x-axis below. Sweeping the strip across x traces out the whole marginal.

$$p_{X|Y}(x \mid y) = \frac{1/\pi}{(2/\pi)\sqrt{1-y^2}} = \frac{1}{2\sqrt{1-y^2}} \qquad \text{for } |x| \le \sqrt{1-y^2}.$$

Given $Y = y$, the horizontal position is uniform on the chord at height y. The chord shrinks as $|y|$ grows, so learning Y changes the distribution of X.

Rearranging Equation 36.18 gives the *chain rule* $p_{X,Y}(x, y) = p_{X|Y}(x \mid y)\, p_Y(y)$. Writing it both ways, $p_{X|Y}\, p_Y = p_{Y|X}\, p_X$, and solving for $p_{X|Y}$ gives *Bayes' rule for densities*,

$$p_{X|Y}(x \mid y) = \frac{p_{Y|X}(y \mid x)\, p_X(x)}{p_Y(y)}. \tag{36.19}$$

It turns a model of how Y arises from X into a statement about X after seeing Y, as Bayes' theorem did for discrete variables in Section 1.6.

In the special case where learning Y tells us nothing about X, that is, $p_{X|Y}(x \mid y) = p_X(x)$ whenever $p_Y(y) > 0$, the two variables are called *independent*, written $X \perp Y$. By the chain rule this is the same as saying that the joint density factorizes, $p_{X,Y}(x, y) = p_X(x)\, p_Y(y)$ (up to sets of probability zero): multiply by $p_Y(y)$ to go one way and divide by it to go the other.

Independence means that the conditional density of X is the same function of x, with the same location and the same shape, at every y. Take the joint density $p_{X,Y}(x, y) = 4xy$ on the unit square $[0, 1]^2$. Its marginal is $p_Y(y) = \int_0^1 4xy\, dx = 2y$, so $p_{X|Y}(x \mid y) = 4xy/(2y) = 2x$, the same for every y, and $X \perp Y$. The dart is the opposite case: its conditional density is uniform at every y, but on a chord whose length depends on y.

Figure 36.6 shows that a common shape is not enough. For a correlated pair, the conditional densities at three heights have the same shape but different centers, so the pair is dependent; only when the joint factorizes do the three coincide.

Independence also gives the density of a sum. The pair $(X, X + Y)$ has the joint density $p_X(x)\, p_Y(s - x)$ at (x, s), and integrating out x as in Equation 36.17 leaves the *convolution* $p_{X+Y}(s) = \int p_X(x)\, p_Y(s - x)\, dx$.

36.2.1.3 Conditional Expectation

A regression model predicts a target from an input. We keep the letters of the previous subsection, so here the target is X and the input, the variable we observe, is Y; the main text writes the pair the other way round. Training with the squared loss looks for the function h that makes $\mathbb{E}[(X - h(Y))^2]$ small. Which function is best? Within a thin strip $Y \approx y$ the model must predict one number for a whole spread of targets. The number c closest on average, in squared distance, to a variable X is its mean, since expanding the square around μ_X gives $\mathbb{E}[(X - c)^2] = \mathrm{Var}(X) + (\mu_X - c)^2$. This suggests predicting the average of X within the strip. The *conditional expectation* of X given $Y = y$ is the mean of the conditional density,

$$\mathbb{E}[X \mid Y{=}y] = \int x\, p_{X|Y}(x \mid y)\, dx. \tag{36.20}$$

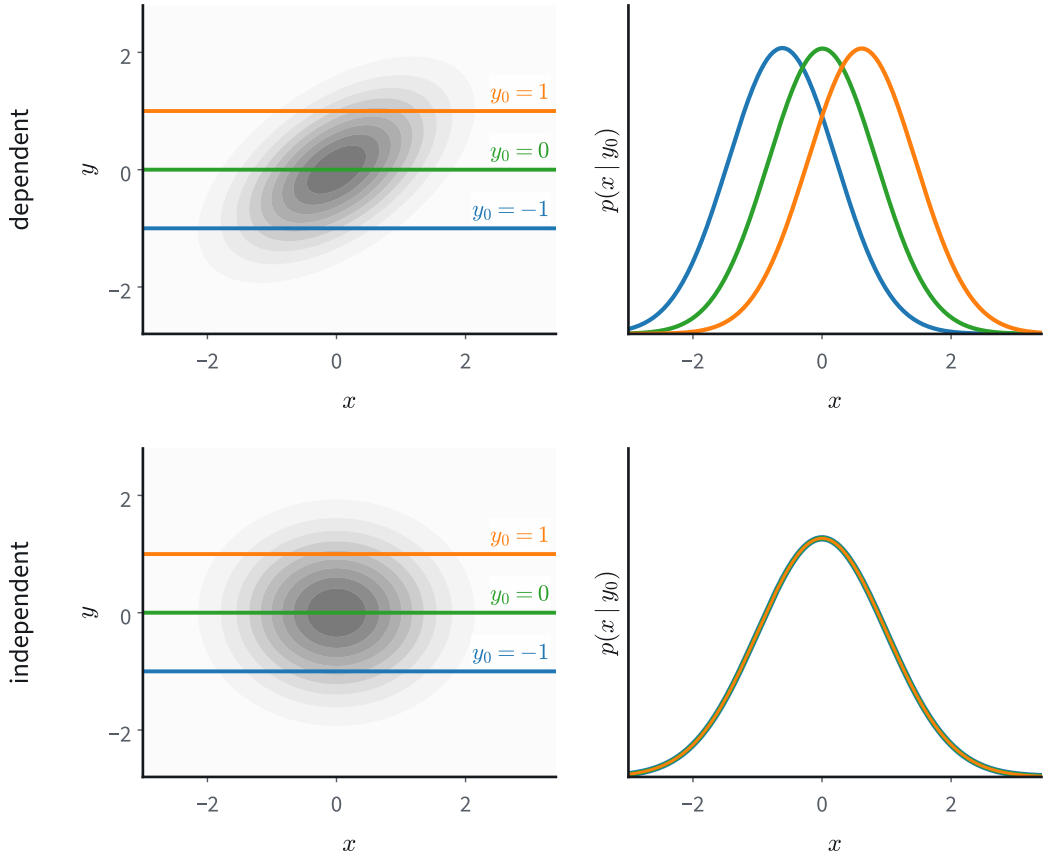


Figure 36.6: Conditioning as slicing. Left: a joint density with three horizontal slices at $y_0 = -1, 0, 1$. Right: each slice renormalized to unit area is the conditional density $p(x \mid y_0)$. Top: a correlated pair; the three conditionals have the same shape but shifted centers, so the conditional depends on y_0 and the pair is dependent. Bottom: a pair whose joint factorizes; the three conditionals coincide, which is independence.

It is the average of X among the outcomes with $Y = y$. As y varies it traces a function of y; evaluated at the random Y, it becomes a random variable $\mathbb{E}[X \mid Y]$ that depends on Y alone. The same strip average defines $\mathbb{E}[g(X,Y) \mid Y{=}y] = \int g(x,y)\, p_{X|Y}(x \mid y)\, dx$ for any function g. A factor $h(Y)$ is constant within each strip, so it comes out of the average:

$$\mathbb{E}[h(Y)\, g(X,Y) \mid Y] = h(Y)\, \mathbb{E}[g(X,Y) \mid Y].$$

The spread within a strip is the *conditional variance* $\mathrm{Var}(X \mid Y) = \mathbb{E}[X^2 \mid Y] - (\mathbb{E}[X \mid Y])^2$, the computational form Equation 36.9 applied within each strip.

Averaging the strip averages should give back the overall average, and the overall spread should split into a spread within strips and a spread between them. Both are true, and the second is the more useful.

Proposition (laws of total expectation and variance). *Let X, Y have a joint density, and let X have finite variance (for discrete pairs, the same identities hold with integrals replaced by sums). Then*

$$\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}[X \mid Y]\big], \qquad \mathrm{Var}(X) = \mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] + \mathrm{Var}\big(\mathbb{E}[X \mid Y]\big). \tag{36.21}$$

Proof. For the first identity, the *tower property*, write out the inner mean and use the chain rule $p_{X|Y}(x \mid y)\, p_Y(y) = p_{X,Y}(x,y)$ to put the strips back together into the joint density:

$$\mathbb{E}\big[\mathbb{E}[X \mid Y]\big] = \int \Big(\int x\, p_{X|Y}(x \mid y)\, dx \Big) p_Y(y)\, dy = \iint x\, p_{X,Y}(x,y)\, dx\, dy = \mathbb{E}[X].$$

The values of y with $p_Y(y) = 0$, where the conditional density is undefined, contribute to neither side: Y lands there with probability zero. The same computation with $g(x,y)$ in place of x gives $\mathbb{E}\big[\mathbb{E}[g(X,Y) \mid Y]\big] = \mathbb{E}[g(X,Y)]$. For the second identity, expand both terms on the right. By the tower property applied to X^2,

$$\mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] = \mathbb{E}\big[\mathbb{E}[X^2 \mid Y]\big] - \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big] = \mathbb{E}[X^2] - \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big].$$

By the computational form Equation 36.9 applied to the variable $\mathbb{E}[X \mid Y]$, whose mean is $\mathbb{E}[X]$ by the tower property,

$$\mathrm{Var}\big(\mathbb{E}[X \mid Y]\big) = \mathbb{E}\big[(\mathbb{E}[X \mid Y])^2\big] - \mathbb{E}[X]^2.$$

Adding the two lines, the term $\mathbb{E}[(\mathbb{E}[X \mid Y])^2]$ cancels and leaves $\mathbb{E}[X^2] - \mathbb{E}[X]^2 = \mathrm{Var}(X)$. ∎

The second identity is sometimes called *Eve's law*, after the pattern E, V, V, E of its right side. It splits the spread of X into the average spread within each strip, the part that knowing Y cannot remove, and the spread of the strip averages, the part that Y explains. Statisticians use this split to attribute the variation in data to groups, under the name analysis of variance. The split also answers the regression question.

Proposition (the conditional mean is the best squared-error predictor). *Let X have finite variance, and let h be any function with $\mathbb{E}[h(Y)^2] < \infty$. Then*

$$\mathbb{E}\big[(X - h(Y))^2\big] = \mathbb{E}\big[\mathrm{Var}(X \mid Y)\big] + \mathbb{E}\big[(\mathbb{E}[X \mid Y] - h(Y))^2\big]. \tag{36.22}$$

Proof. Write $m(Y) = \mathbb{E}[X \mid Y]$ and split the error into two parts, $X - h(Y) = (X - m(Y)) + (m(Y) - h(Y))$. Squaring gives two squares and a cross term, and the cross term is the only step that needs thought. The mean of the first square is $\mathbb{E}[\mathrm{Var}(X \mid Y)]$ by the tower property. The cross term $2(X - m(Y))(m(Y) - h(Y))$ has conditional mean zero, because the factor $m(Y) - h(Y)$ depends on Y alone and comes out of the strip average, leaving $\mathbb{E}[X - m(Y) \mid Y] = m(Y) - m(Y) = 0$. By the tower property its mean is zero too. ■

The first term does not depend on h, and the second is never negative and vanishes for $h(Y) = \mathbb{E}[X \mid Y]$. So the conditional mean is the best predictor in squared error, and no predictor can do better than the average spread within the strips. A network trained with the squared loss is estimating $\mathbb{E}[\text{target} \mid \text{input}]$, which is how Section 2.1 reads its model, there written $\mathbb{E}[y \mid \mathbf{x}]$.

Two examples show the split with numbers. For the dart, predict X from Y. Given $Y = y$, X is uniform on a chord symmetric about 0, so $\mathbb{E}[X \mid Y = y] = 0$. A variable uniform on an interval of half-width w has variance $w^2/3$, so $\mathrm{Var}(X \mid Y = y) = (1 - y^2)/3$. By symmetry $\mathbb{E}[X^2] = \mathbb{E}[Y^2] = \frac{1}{2}\mathbb{E}[R^2] = \frac{1}{4}$, and Eve's law gives

$$\mathrm{Var}(X) = \mathbb{E}\Big[\frac{1 - Y^2}{3}\Big] + \mathrm{Var}(0) = \frac{1 - 1/4}{3} + 0 = \frac{1}{4}.$$

All of the variance is unexplained. Knowing Y narrows the range of X but never shifts its center, so the best predictor of X from Y is the constant 0, although X and Y are dependent.

For the second example, draw a label Z that is 1 with probability 0.2 and 0 with probability 0.8. Given $Z = 1$, draw X from the normal distribution $\mathcal{N}(3, 1)$ with mean 3 and variance 1, and given $Z = 0$ from $\mathcal{N}(-1, 1)$. This mixture has the two-bump density of Figure 36.1. Its mean is $0.2 \cdot 3 + 0.8 \cdot (-1) = -0.2$. Within each component the variance is 1, so $\mathbb{E}[\mathrm{Var}(X \mid Z)] = 1$. Between the components, $\mathbb{E}[X \mid Z]$ takes the values 3 and -1, whose variance is $0.2 \cdot 0.8 \cdot (3 - (-1))^2 = 2.56$. The total variance is 3.56, and the label explains $2.56/3.56 = 72\%$ of it. In a sample of a million draws the two parts will come out near 1 and 2.56. Should their sum match the variance of the sample only as roughly as the parts match their targets, or exactly?

```
n = 10**6
z = rng.uniform(size=n) < 0.2                          # component label Z
x = np.where(z, rng.normal(3, 1, n), rng.normal(-1, 1, n))
w = np.array([np.mean(~z), np.mean(z)])                # P(Z=0), P(Z=1)
cond_mean = np.array([x[~z].mean(), x[z].mean()])      # E[X | Z]
within = w @ np.array([x[~z].var(), x[z].var()])       # E[Var(X | Z)]
between = w @ (cond_mean - x.mean())**2                # Var(E[X | Z])
print(f'Var(X) {x.var():.3f} = within {within:.3f} + between {between:.3f}')
```

```
Var(X) 3.562 = within 1.000 + between 2.562
```

Exactly. The within and between parts are close to 1 and 2.56, with sampling noise in the third decimal, but their sum equals the variance of the sample in every run, up to rounding in the last printed digit. Eve's law holds for the sample itself, with the group fractions and group variances in place of the probabilities and conditional variances.

36.2.2 Covariance and Correlation

The conditional mean describes how the average of X moves with Y, and it is a whole function. Often a single number is enough: do X and Y tend to be large together, or does one tend to be large when the other is small? The covariance is that number. It has two defects, which the rest of this part repairs or at least exposes: it carries units, which the correlation removes, and it sees only straight-line trends, which nothing removes.

36.2.2.1 Covariance

Section 1.6 defined the *covariance* as the average product of the two deviations. For discrete X, Y taking (x_i, y_j) with probability p_{ij} it is

$$\sigma_{XY} = \operatorname{Cov}(X,Y) = \sum_{i,j} (x_i - \mu_X)(y_j - \mu_Y)\, p_{ij} = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y], \tag{36.23}$$

where the second form follows by expanding the product, as for the computational form of the variance. In particular $\operatorname{Cov}(X,X) = \operatorname{Var}(X)$. We assume throughout that both variances are finite; since $|xy| \le (x^2+y^2)/2$, the mean of XY then exists.

As an example, let $X \in \{1,3\}$ and $Y \in \{-1,3\}$ with

$$\begin{aligned} P(X{=}1, Y{=}{-1}) &= \tfrac{\theta}{2}, & P(X{=}1, Y{=}3) &= \tfrac{1-\theta}{2}, \\ P(X{=}3, Y{=}{-1}) &= \tfrac{1-\theta}{2}, & P(X{=}3, Y{=}3) &= \tfrac{\theta}{2}, \end{aligned}$$

for a parameter $\theta \in [0,1]$. We call this the *four-cell example*. Each variable takes its two values with probability $\tfrac12$, so $\mu_X = 2$ and $\mu_Y = 1$. The mean of the product is

$$\mathbb{E}[XY] = \tfrac12(-\theta + 3(1-\theta) - 3(1-\theta) + 9\theta) = 4\theta,$$

so $\operatorname{Cov}(X,Y) = 4\theta - 2$: it is $+2$ at $\theta = 1$, -2 at $\theta = 0$, and 0 at $\theta = \tfrac12$.

For variables with a joint density, the sum becomes an integral,

$$\sigma_{XY} = \iint (x - \mu_X)(y - \mu_Y)\, p_{X,Y}(x,y)\, dx\, dy, \tag{36.24}$$

the deviation product averaged over the joint density. For the dart, both coordinates have mean 0, and $\operatorname{Cov}(X,Y) = \mathbb{E}[XY] = \frac{1}{\pi}\iint_{\text{disk}} xy\, dx\, dy = 0$, because the integrand is odd in x and the disk is symmetric. Yet X and Y are dependent, as the shrinking chords showed. Covariance detects a linear trend. The dart's dependence is symmetric and has none, so zero covariance does not mean independence (Section 1.6 has a discrete example, $Y = X^2$).

Covariance carries the units of both variables. In Figure 36.7 the bottom row repeats the draws of the top row with Y in cents instead of dollars. Every sample covariance grows by a factor of 100, while no cloud changes shape. Only the sign of a covariance means something without the units; the correlation coefficient below removes them.

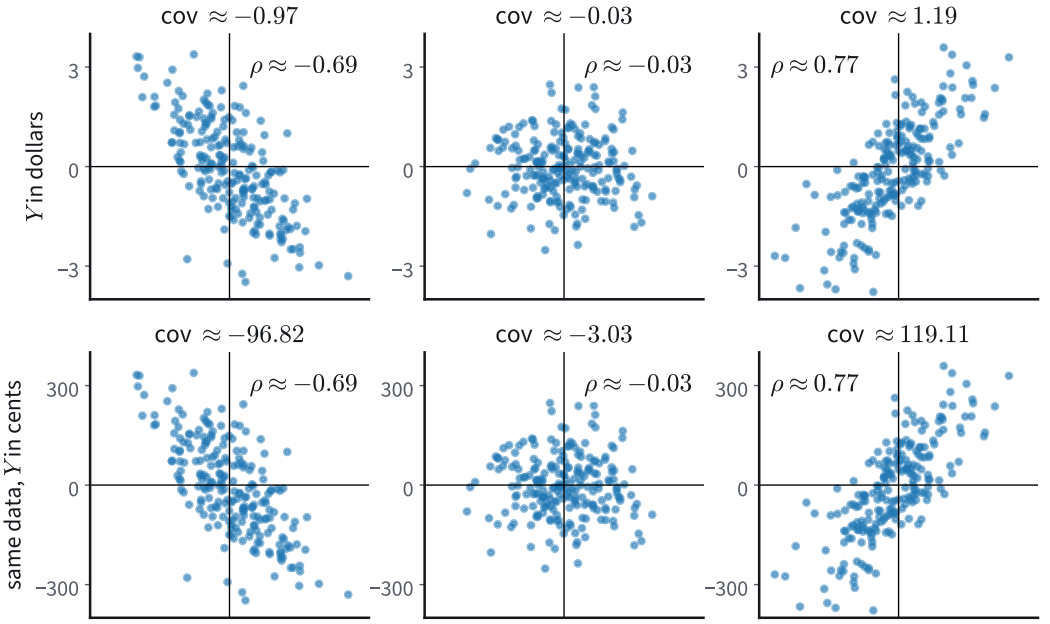


Figure 36.7: Top row: scatter clouds of (X, Y) pairs with covariance tuned negative, zero, and positive (left to right), Y in dollars: negative covariance tilts the cloud down-right, zero leaves it round, positive tilts it up-right. Bottom row: the identical draws with Y restated in cents. Every sample covariance is multiplied by 100 while each panel's correlation ρ (in its top corner) is unchanged: the magnitude of covariance carries units, only its sign is scale-free.

Covariance is linear in each argument. Shifts drop out of the deviations and scales pull through, so $\mathrm{Cov}(cX + d, Y) = c\,\mathrm{Cov}(X, Y)$. The deviation of a sum is the sum of the deviations, so $\mathrm{Cov}(X + Z, Y) = \mathrm{Cov}(X, Y) + \mathrm{Cov}(Z, Y)$ by linearity of the mean. The same holds in the second slot. Independent variables have zero covariance: when the joint density factorizes, the double integral for $\mathbb{E}[XY]$ splits into $\mathbb{E}[X]\,\mathbb{E}[Y]$.

Covariance also completes the rule for the variance of a sum. Write $\tilde{X} = X - \mu_X$ and $\tilde{Y} = Y - \mu_Y$. By linearity $X + Y$ has mean $\mu_X + \mu_Y$, so its deviation is $\tilde{X} + \tilde{Y}$. Expanding $\mathbb{E}[(\tilde{X} + \tilde{Y})^2] = \mathbb{E}[\tilde{X}^2] + 2\,\mathbb{E}[\tilde{X}\tilde{Y}] + \mathbb{E}[\tilde{Y}^2]$ gives

$$\mathrm{Var}(X + Y) = \mathrm{Var}(X) + \mathrm{Var}(Y) + 2\,\mathrm{Cov}(X, Y). \tag{36.25}$$

For independent X and Y the covariance vanishes and the variances add.

The most important use of this rule is the average of many draws. Expanding the square of a sum of n deviations in the same way gives $\mathrm{Var}(\sum_i X_i) = \sum_i \sum_j \mathrm{Cov}(X_i, X_j)$. Let $X_1, \ldots, X_n$ be independent with common variance σ^2. Every cross term vanishes, so the

sum has variance $n\sigma^2$, and the affine rule with $c = 1/n$ divides it by n^2. The average $\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i$ therefore has

$$\operatorname{Var}(\bar{X}_n) = \frac{1}{n^2} \cdot n\sigma^2 = \frac{\sigma^2}{n}. \tag{36.26}$$

Averaging n independent draws divides the variance by n and the standard deviation by $\sqrt{n}$, so four times as many draws halve the error. This is the rule behind every *Monte Carlo* estimate, an average of computed samples that stands in for an expectation (Section 36.5.1.7). For the dart, the average distance of n darts has variance $1/(18n)$, and 556 darts bring its standard deviation down to 0.01. The $\sigma/\sqrt{n}$ rule needs a finite variance. The *law of large numbers* itself, that $\bar{X}_n$ converges to the mean, needs only a finite mean (Section 36.5). The Cauchy distribution has neither, and averages of its draws do not settle however large n is.

36.2.2.2 Correlation

Whether the covariance $4\theta - 2$ of the four-cell example is large depends on the units of X and Y. To remove the units, divide by the two standard deviations, which carry the same units. The *correlation coefficient* is

$$\rho(X, Y) = \frac{\operatorname{Cov}(X, Y)}{\sigma_X \sigma_Y}. \tag{36.27}$$

Section 1.6 stated that ρ lies between -1 and 1. Here is the proof, and with it the cases of equality.

Proposition (correlation bound). *For any X, Y with finite, nonzero variances, $-1 \le \rho(X, Y) \le 1$. Equality holds if and only if $Y = cX + d$ with probability one for constants $c \neq 0$ and d, with $\rho = 1$ when $c > 0$ and $\rho = -1$ when $c < 0$.*

Proof. A variance is never negative, so $\operatorname{Var}(tX + Y) \ge 0$ for every real t. By the variance-of-a-sum rule and the affine rule this is a quadratic in t,

$$\operatorname{Var}(tX + Y) = \operatorname{Var}(X)\, t^2 + 2\operatorname{Cov}(X, Y)\, t + \operatorname{Var}(Y) \;\ge\; 0 \quad \text{for every } t.$$

A quadratic with positive leading coefficient that is never negative has at most one real root, so its discriminant is not positive:

$$\big(2\operatorname{Cov}(X, Y)\big)^2 - 4\operatorname{Var}(X)\operatorname{Var}(Y) \le 0, \qquad \text{that is} \qquad \operatorname{Cov}(X, Y)^2 \le \operatorname{Var}(X)\operatorname{Var}(Y).$$

Dividing by $\sigma_X^2 \sigma_Y^2$ gives $\rho^2 \le 1$. If $\rho^2 = 1$, the discriminant vanishes and the quadratic has a double root $t^\star$ with $\operatorname{Var}(t^\star X + Y) = 0$. A variable of zero variance is constant with probability one, so $Y = -t^\star X + d$ with probability one, and $t^\star \neq 0$ because $\operatorname{Var}(Y) > 0$. Conversely, if $Y = cX + d$ with $c \neq 0$, then $\operatorname{Cov}(X, Y) = c\operatorname{Var}(X)$ and $\sigma_Y = |c|\sigma_X$, so $\rho = c/|c| = \pm 1$. ∎

The correlation measures how tightly X and Y follow a straight line, and the sign of the line's slope, but not the slope itself: $Y = 0.5X$ and $Y = 2X$ both have $\rho = 1$. For the

four-cell example $\sigma_X = 1$ and $\sigma_Y = 2$, so $\rho = (4\theta - 2)/2 = 2\theta - 1$, running from -1 to 1 as θ runs from 0 to 1. Figure 36.8 shows two blind spots of ρ: two clouds with the same correlation and different slopes, and a cloud on a parabola whose correlation is zero.

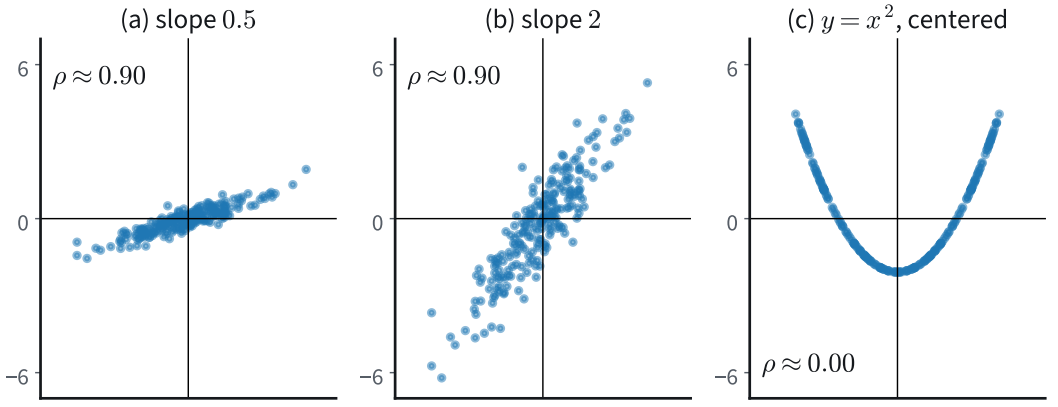


Figure 36.8: Correlation measures the tightness and sign of a linear relation, not its slope: (a) and (b) have the same $\rho \approx 0.9$ at slopes 0.5 and 2. It also misses a nonlinear relation: in (c) Y is a function of X and $\rho \approx 0$.

The proof is the standard proof of the Cauchy–Schwarz inequality $|\mathbf{u}^\top \mathbf{v}| \le \|\mathbf{u}\|\,\|\mathbf{v}\|$, and the correlation is a cosine. On random variables with mean zero and finite variance, the mean of a product is an inner product (counting two variables as the same when they agree with probability one). For the centered variables $\tilde{X}$ and $\tilde{Y}$ the inner product is $\langle \tilde{X}, \tilde{Y} \rangle = \mathbb{E}[\tilde{X}\tilde{Y}] = \mathrm{Cov}(X, Y)$, the length it defines is the standard deviation, $\|\tilde{X}\| = \sqrt{\mathbb{E}[\tilde{X}^2]} = \sigma_X$, and the correlation is the cosine $\langle \tilde{X}, \tilde{Y} \rangle / (\|\tilde{X}\|\,\|\tilde{Y}\|)$. Uncorrelated variables are orthogonal, and $\rho = \pm 1$ means parallel.

36.2.2.3 The Covariance Matrix

With n variables there is a covariance for every pair, and the natural way to keep them is a matrix. For a random vector $\mathbf{X} = (X_1, \dots, X_n)^\top$ with mean μ, the *covariance matrix* is the mean of the outer product of the deviation with itself,

$$\Sigma = \mathbb{E}\big[(\mathbf{X} - \mu)(\mathbf{X} - \mu)^\top\big], \qquad \Sigma_{ij} = \mathrm{Cov}(X_i, X_j). \tag{36.28}$$

The diagonal holds the variances and the off-diagonal entries the covariances. The matrix is symmetric, because covariance is. It is also positive semidefinite. For any fixed vector $\mathbf{a}$, the variable $\mathbf{a}^\top \mathbf{X} = \sum_i a_i X_i$ has deviation $\mathbf{a}^\top (\mathbf{X} - \mu)$, so

$$\mathrm{Var}(\mathbf{a}^\top \mathbf{X}) = \mathbb{E}\big[\mathbf{a}^\top (\mathbf{X} - \mu)(\mathbf{X} - \mu)^\top \mathbf{a}\big] = \mathbf{a}^\top \Sigma\, \mathbf{a} \;\ge\; 0.$$

The quadratic form of Σ in the direction $\mathbf{a}$ is the variance of the data projected onto $\mathbf{a}$, and a variance is never negative. For the dart, $\Sigma = \mathrm{diag}(\frac{1}{4}, \frac{1}{4})$: the coordinates are uncorrelated and spread equally.

The eigenvectors of Σ are the directions of principal component analysis (Section 33.5) and the axes of the multivariate Gaussian's elliptical contours (Section 36.3).

36.2.3 Transformations of Random Variables

Machine learning creates new random variables from old ones all the time: an activation function applied to a pre-activation, a sampler applied to uniform noise, a normalizing flow applied to a Gaussian. If X has a known density and $Y = g(X)$, what is the density of Y?

36.2.3.1 The One-Dimensional Rule

The obvious guess, $p_Y(y) = p_X(g^{-1}(y))$, is wrong, and the dart already shows why. Its distance is sampled as $R = \sqrt{U}$ with U uniform on $[0, 1]$, and this map turns the flat density of U into the density $2r$, which is not flat. The probability in a short interval must survive the map. An interval of width Δx at x carries probability about $p_X(x)\,\Delta x$, and g sends it to an interval of width about $\Delta y = |g'(x)|\,\Delta x$ at $y = g(x)$, which must carry the same probability:

$$p_Y(y)\,|g'(x)|\,\Delta x \approx p_X(x)\,\Delta x.$$

Where g stretches the line, the density must thin by the stretch factor $|g'(x)|$; where g compresses, the density must rise. Figure 36.9 shows the two equal areas for $Y = e^X$.

Proposition (change of variables in one dimension). *Let X have a density p_X that is continuous on an open interval I with $P(X \in I) = 1$, and let $g : I \to \mathbb{R}$ be continuously differentiable with $g' \neq 0$ on I, so that g is strictly monotone and one-to-one. Then $Y = g(X)$ has the density*

$$p_Y(y) = p_X(g^{-1}(y)) \left| \frac{dg^{-1}}{dy}(y) \right| \tag{36.29}$$

on the interval $g(I)$, and 0 outside it.

Proof. Let g be increasing. Then $g(X) \le y$ exactly when $X \le g^{-1}(y)$, so the c.d.f.s satisfy $F_Y(y) = F_X(g^{-1}(y))$ for y in $g(I)$. Differentiating with the chain rule and Equation 36.5 gives $p_Y(y) = p_X(g^{-1}(y))\,\frac{dg^{-1}}{dy}(y)$. If g is decreasing, then $g(X) \le y$ exactly when $X \ge g^{-1}(y)$, so $F_Y(y) = 1 - F_X(g^{-1}(y))$. Its derivative carries a minus sign, which cancels the negative sign of dg^{-1}/dy; the absolute value covers both cases. In either case F_Y is continuously differentiable on $g(I)$, which carries all the probability, so by the fundamental theorem of calculus its derivative is a density of Y. ■

The new density is the old one, read at the point that maps to y, times the local compression of the inverse map. Since $dg^{-1}/dy = 1/g'(x)$ at $x = g^{-1}(y)$, this is the factor $1/|g'(x)|$ of the stretching argument.

Inverse-transform sampling is a change of variables, and the rule recovers the dart's density from its sampler. Take $X = U$, uniform on $I = (0, 1)$, and $g(u) = \sqrt{u}$, so $g^{-1}(r) = r^2$ and $|dg^{-1}/dr| = 2r$. The rule gives $p_R(r) = 1 \cdot 2r$, the density of the dart's distance.

When g is not one-to-one, several points map to the same y, and their contributions add:

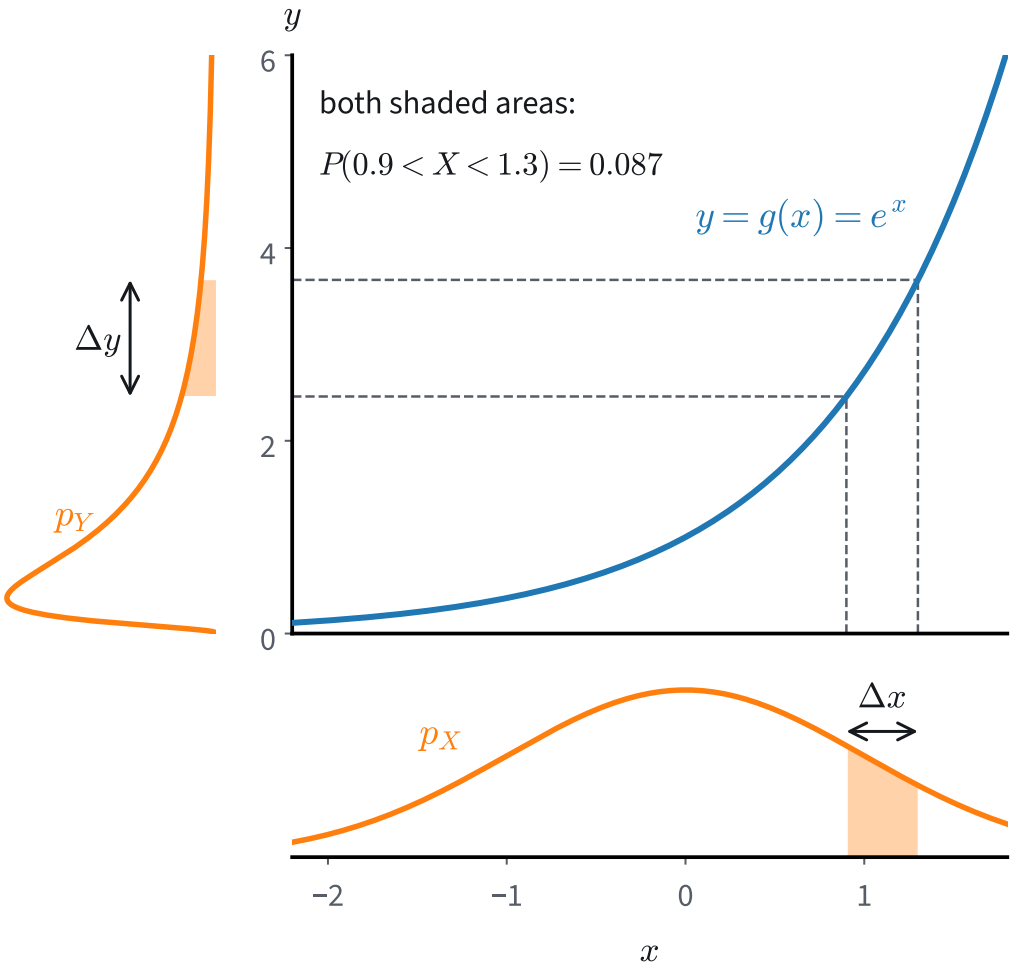


Figure 36.9: Change of variables. An interval of width Δx under p_X (bottom) is mapped by g to an interval of width $\Delta y = g'(x)\,\Delta x$ under p_Y (left). The two shaded areas are the same probability, so where g stretches ($\Delta y > \Delta x$) the density p_Y must be lower by the factor $|g'(x)|$. Here $X \sim \mathcal{N}(0, 1)$ and $Y = e^X$.

$$p_Y(y) = \sum_{x:\, g(x)=y} \frac{p_X(x)}{|g'(x)|},$$

for values y with finitely many preimages, each with $g'(x) \neq 0$. For $Y = X^2$ and $y > 0$, the two preimages $\pm\sqrt{y}$ have $|g'| = 2\sqrt{y}$, so $p_Y(y) = [p_X(\sqrt{y}) + p_X(-\sqrt{y})]/(2\sqrt{y})$. The hypothesis $g' \neq 0$ matters. The ReLU has $g' = 0$ on the whole half-line $(-\infty, 0)$, which it collapses to the single point 0, and the result is an atom rather than a density (Exercise 2).

As a worked example, let $X \sim \mathcal{N}(0,1)$ have the standard normal density $p_X(x) = e^{-x^2/2}/\sqrt{2\pi}$, and let $Y = e^X$. Then $g^{-1}(y) = \log y$ and $|dg^{-1}/dy| = 1/y$ for $y > 0$, and the rule gives the *log-normal* density

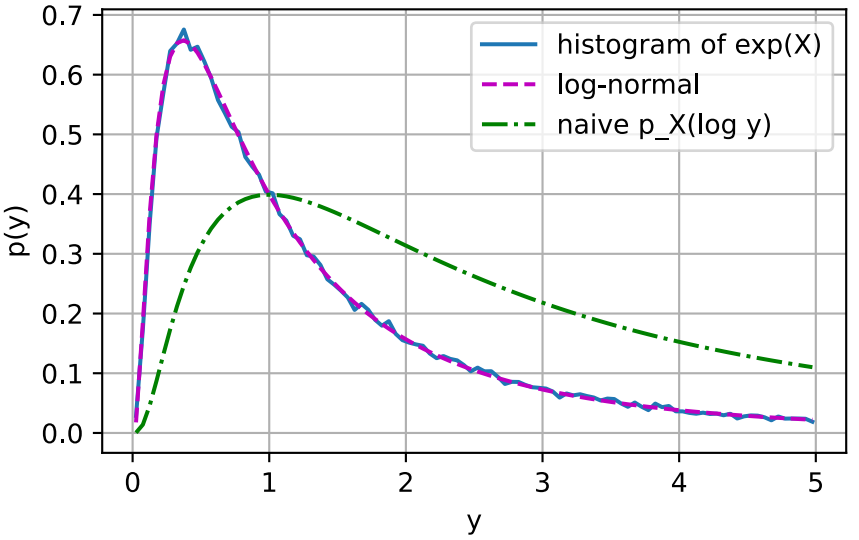
$$p_Y(y) = \frac{1}{y\sqrt{2\pi}} \exp\Big(-\tfrac{1}{2}(\log y)^2 \Big), \qquad y > 0.$$

The factor $1/y$ is the change-of-variables correction. Setting the derivative of $\log p_Y(y) = -\log y - \tfrac{1}{2}(\log y)^2 + \text{const}$ to zero gives $\log y = -1$, so the density peaks at $y = e^{-1} \approx 0.368$. The naive guess $p_X(\log y)$ peaks at $y = 1$, and it is not even a density. Substituting $y = e^x$, its total area is $\int p_X(x)\, e^x\, dx = \mathbb{E}[e^X] = e^{1/2} \approx 1.65$ (complete the square, $x - x^2/2 = \tfrac{1}{2} - \tfrac{1}{2}(x-1)^2$).

The same computation gives the moments $\mathbb{E}[Y^k] = \mathbb{E}[e^{kX}] = e^{k^2/2}$, all finite. Yet the tail $P(Y > t) = P(X > \log t)$ decays more slowly than any exponential in t. The log-normal is therefore heavy-tailed, although, unlike the Cauchy distribution, it has every moment.

The cell draws a histogram of 10^5 samples of e^X over both candidates. The naive curve must enclose 0.65 too much area; it is worth guessing where the excess sits before looking.

```
y = np.exp(rng.standard_normal(10**5))                # Y = e^X
edges = np.linspace(0, 5, 101)
mids = (edges[:-1] + edges[1:]) / 2
hist = np.histogram(y, edges)[0] / (len(y) * (edges[1] - edges[0]))
p_X = lambda x: np.exp(-x**2 / 2) / np.sqrt(2 * np.pi)
d2l.plot(mids, [hist, p_X(np.log(mids)) / mids, p_X(np.log(mids))],
         'y', 'p(y)', figsize=(5, 3),
         legend=['histogram of exp(X)', 'log-normal', 'naive p_X(log y)'])
```



The histogram follows the log-normal curve up to sampling noise, with its peak well left of $y = 1$, near e^{-1}. The naive curve peaks in the wrong place. It is the true density times y, so it lies below the data for $y < 1$ and above it for $y > 1$, by a factor that grows into the tail.

36.2.3.2 The Multivariate Rule

In several dimensions, a map stretches areas and volumes rather than lengths, and the stretch factor becomes a determinant. The smallest case is a linear map. Let $\mathbf{Y} = \mathbf{A}\mathbf{X}$ with $\mathbf{A} = \operatorname{diag}(2, 3)$. A unit square becomes a 2×3 rectangle, which must carry the same probability as the square, spread over area $6 = \det \mathbf{A}$. So the density of $\mathbf{Y}$ is one sixth of the density of $\mathbf{X}$ at the corresponding point, $p_{\mathbf{Y}}(\mathbf{y}) = p_{\mathbf{X}}(\mathbf{A}^{-1}\mathbf{y})/6$.

For a nonlinear map, the stretch varies from point to point, and the change of variables theorem for integrals accounts for it (Section 34.6). Let g be a continuously differentiable bijection between two regions of $\mathbb{R}^n$, with continuously differentiable inverse, and write $J_{g^{-1}}$ for the Jacobian matrix of partial derivatives of the inverse. Locally, g^{-1} scales volume by $|\det J_{g^{-1}}|$, and substituting $\mathbf{x} = g^{-1}(\mathbf{y})$ in an integral over the preimage of a region $\mathcal{D}$ gives

$$\int_{g^{-1}(\mathcal{D})} f(\mathbf{x})\, d\mathbf{x} = \int_{\mathcal{D}} f(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right|\, d\mathbf{y}.$$

Now let $\mathbf{Y} = g(\mathbf{X})$. The event $\mathbf{Y} \in \mathcal{D}$ is the event $\mathbf{X} \in g^{-1}(\mathcal{D})$, so the theorem with $f = p_{\mathbf{X}}$ reads

$$P(\mathbf{Y} \in \mathcal{D}) = \int_{g^{-1}(\mathcal{D})} p_{\mathbf{X}}(\mathbf{x})\, d\mathbf{x} = \int_{\mathcal{D}} p_{\mathbf{X}}(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right|\, d\mathbf{y}.$$

The integrand on the right is therefore the density of $\mathbf{Y}$:

$$\begin{aligned} p_{\mathbf{Y}}(\mathbf{y}) &= p_{\mathbf{X}}\big(g^{-1}(\mathbf{y})\big)\,\big|\det J_{g^{-1}}(\mathbf{y})\big|, \\ \text{equivalently} \qquad \log p_{\mathbf{Y}}(\mathbf{y}) &= \log p_{\mathbf{X}}(\mathbf{x}) - \log\big|\det J_g(\mathbf{x})\big| \quad \text{at } \mathbf{x} = g^{-1}(\mathbf{y}). \end{aligned} \tag{36.30}$$

The second form uses $J_{g^{-1}}(\mathbf{y}) = J_g(\mathbf{x})^{-1}$. Passing a sample through g subtracts the log of the local volume stretch from its log-density. Because $\log|\det(J_2 J_1)| = \log|\det J_1| + \log|\det J_2|$, a composition of maps subtracts the sum of their log-determinants. This is how *normalizing flows* compute exact likelihoods: they stack invertible layers whose log-determinants are cheap (Rezende and Mohamed 2015), and Section 38.2.3 builds one whose layers are the steps of a differential equation. Exercise 6 builds a coupling layer.

The dart gives a last example. Describe its position in polar coordinates, the distance R and the angle Θ, with $(x, y) = (r\cos\theta, r\sin\theta)$. The Jacobian determinant of this map from (r, θ) to (x, y) is $\cos\theta \cdot r\cos\theta - (-r\sin\theta)\sin\theta = r$, so the rule gives

$$p_{R,\Theta}(r, \theta) = \frac{1}{\pi} \cdot r = 2r \cdot \frac{1}{2\pi}, \qquad 0 \le r \le 1,\ 0 \le \theta < 2\pi.$$

The joint density factorizes. The angle is uniform on $[0, 2\pi)$, the distance has the density $2r$ once more, now from a Jacobian, and the two are independent. In Cartesian coordinates the dart's X and Y were dependent; in the right coordinates the same dart is described by two independent variables.

The section leaves a question about averages. We know the variance of an average of n independent draws, σ^2/n Equation 36.26, but not its distribution. Chebyshev's inequality (Section 36.1) turns the variance into a bound, $\sigma^2/(n\epsilon^2)$, on the chance that the average errs by ϵ or more, and for bounded or Gaussian draws this is far too pessimistic: the chance falls exponentially fast in n. Which distribution an average approaches is settled in Section 36.3.2.3, and the exponential rate in Section 36.9. Before either, we need densities other than the dart's.

36.2.4 Summary

Marginals come from integrating out, and conditionals from slicing the joint density along a thin strip. The conditional mean is the best squared-error predictor, and the law of total variance splits a variance into the part that the conditioning variable explains and the part it cannot. Covariance and correlation measure a linear trend, not dependence in general, and the average of n independent draws with finite variance has variance σ^2/n. A smooth invertible map changes a density by the factor of its local stretch, the absolute Jacobian determinant. The dart shows all of it: its coordinates are dependent and uncorrelated, and in polar coordinates they become two independent variables.

36.2.5 Exercises

1. code **Classes missing from a minibatch.** A minibatch holds B labels drawn independently and uniformly from K classes. Let M be the number of classes that do not appear in it, and write $M = \sum_{k=1}^{K} I_k$, where $I_k = 1$ if class k is missing and $I_k = 0$ otherwise.
    1. Compute $\mathbb{E}[M]$ with linearity Equation 36.7, and evaluate it for $K = 1000$ and $B = 256$.

   2. Compute $\mathrm{Cov}(I_j, I_k)$ for $j \neq k$. Are the indicators independent?
   3. Extend the variance-of-a-sum rule Equation 36.25 to K terms, and compute the standard deviation of M for $K = 1000$ and $B = 256$. Compare it with the value that independent indicators would give, and check both against 10^4 simulated minibatches. How can covariances as small as those in (b) change the answer this much?

2. **Uncorrelated but dependent.** Whitening, for example with the principal components of Section 33.5, makes coordinates uncorrelated. The following pair has Gaussian coordinates and zero covariance. Let $X \sim \mathcal{N}(0,1)$, let S be a random sign independent of X with $P(S=1) = P(S=-1) = \frac{1}{2}$, and let $Y = SX$.

   1. Show that $Y \sim \mathcal{N}(0,1)$ by computing its c.d.f., and that $\mathrm{Cov}(X,Y) = 0$.
   2. Compare $P(|X| \le 1, |Y| \le 1)$ with $P(|X| \le 1)\,P(|Y| \le 1)$. Are X and Y independent?
   3. Show that $X+Y$ has an atom at 0, so that $X+Y$ is not Gaussian. With probability one the point (X,Y) lies on one of the two lines $y = \pm x$, and a double integral over a line is zero, because a line has no area. Conclude that the pair (X,Y) has no joint density, although each coordinate has one.

3. **Averaging correlated estimates.** The text derives $\mathrm{Var}(\bar{X}_n) = \sigma^2/n$ for independent draws. Averages of dependent estimates behave differently.

   1. Let $Y_1, \dots, Y_n$ each have variance σ^2, with correlation ρ between every pair, as the predictions of n ensemble members trained on the same data might. Show that

$$\mathrm{Var}\Big(\frac{1}{n}\sum_{i=1}^{n} Y_i\Big) = \sigma^2\Big(\rho + \frac{1-\rho}{n}\Big). \tag{36.31}$$

      What does averaging achieve as $n \to \infty$?

   2. Use $\mathbf{a}^\top \Sigma\, \mathbf{a} \ge 0$ for the covariance matrix Σ of $Y_1, \dots, Y_n$ to show that $\rho \ge -1/(n-1)$. Give variables that attain this bound.

4. code **Sigmoid of a Gaussian.** Let $X \sim \mathcal{N}(0, s^2)$ be the pre-activation of a sigmoid unit, and let $Y = \mathrm{sigmoid}(X)$, whose inverse is $\mathrm{logit}(y) = \log(y/(1-y))$.

   1. Use Equation 36.29 to find the density p_Y on $(0,1)$.
   2. For which s is $y = \frac{1}{2}$ a local minimum of p_Y? Writing $\log p_Y$ as a function of $z = \mathrm{logit}(y)$ simplifies the computation.
   3. Modify the `#random-variables-change-of-variables` cell to compare a histogram of Y with p_Y for $s \in \{1, \sqrt{2}, 3\}$. What does the histogram at $s = 3$ imply for the derivative $\mathrm{sigmoid}'(X) = Y(1-Y)$, through which backpropagation passes the gradient of these units?

5. **Explained variance.** ⊠ A regression model h predicts a target X from an input Y. The fraction of the variance that it explains is $R^2 = 1 - \mathbb{E}\big[(X - h(Y))^2\big]/\mathrm{Var}(X)$.

   1. Use Equation 36.22 and the law of total variance Equation 36.21 to show that every predictor has $R^2 \le \mathrm{Var}(\mathbb{E}[X \mid Y])/\mathrm{Var}(X)$. Evaluate this ceiling for the two-component mixture of the text, predicting X from the label Z, and for the dart, predicting X from Y.

2. Given the input, the target is -1 or $+1$ with probability $\frac{1}{2}$ each. Which prediction minimizes the expected squared error, and how often does the target take that value? Which predictions minimize the expected absolute error $\mathbb{E}\big[|X - h(Y)|\big]$?

6. code **Affine coupling layer and Box–Muller.** ⍰ Normalizing flows stack invertible maps whose log-determinants in Equation 36.30 are cheap to compute. The affine coupling layer of RealNVP (Dinh, Sohl-Dickstein, et al. 2017) maps $\mathbf{x} \in \mathbb{R}^2$ to

$$y_1 = x_1, \qquad y_2 = x_2\, e^{s(x_1)} + t(x_1), \tag{36.32}$$

where s and t are arbitrary functions, for example small neural networks.

1. Write down the inverse map and the Jacobian $J_g(\mathbf{x})$. Show that $\log|\det J_g(\mathbf{x})| = s(x_1)$, however complicated s and t are.
2. Let X_1 and X_2 be independent standard normal variables, and take $s(x_1) = \tanh(x_1)$ and $t(x_1) = x_1^2/2$. Use Equation 36.30 to write $\log p_{\mathbf{Y}}(\mathbf{y})$ in closed form. With NumPy, evaluate $p_{\mathbf{Y}}$ on a grid over $[-6, 6] \times [-12, 24]$ with spacing 0.02 and check that its Riemann sum is close to 1. Check the log-determinant against the determinant of a finite-difference Jacobian at a few random points.
3. The first coordinate passes through the layer unchanged. What is the distribution of Y_1, whatever s and t are? How must a second coupling layer be arranged so that the composition can change both marginals, and what is the log-determinant of the composition?
4. Flows can also run from uniform noise to Gaussians. Let U_1, U_2 be independent and uniform on $(0, 1)$, and set $R = \sqrt{-2\log U_1}$, $\Theta = 2\pi U_2$, and $(Z_1, Z_2) = (R\cos\Theta, R\sin\Theta)$. Find the density of R with the one-dimensional rule, then the joint density of (R, Θ), and change to Cartesian coordinates with the polar Jacobian of the text to show that Z_1 and Z_2 are independent standard normal variables. This is the Box–Muller method. Check with 10^5 samples that the sample covariance matrix of (Z_1, Z_2) is close to the identity.

36.3 Distributions

Section 36.1 and Section 36.2 gave us a language for any random quantity: a mass function or a density, a mean, a variance. A model that predicts a label uses this language whether or not it says so. A binary classifier outputs one number, the probability that the label is 1. A language model outputs a probability for every token in its vocabulary, and the numbers sum to one. A regression network outputs a predicted value and is trained with squared error. Each of these models has chosen a particular distribution for its labels, and that choice fixes the loss it minimizes. Which distributions are these, where do they come from, and what are their means and variances?

A coin will carry us through most of the answer. Suppose it lands heads 9 times in 13 flips. If the coin is fair, how likely is that? To answer, we need the distribution of the number of heads, and the same distribution describes how many test examples a classifier gets right. We will find that the answer is about 9%.

The idea of this section is that a handful of distributions grow out of a single coin flip. Counting heads gives the binomial. Many flips that each rarely come up heads give the Poisson, and the time until the next rare event gives the exponential. Many ordinary flips, added up and rescaled, give the Gaussian. Because each distribution is built from simpler pieces, its

mean and variance follow from the construction, by a short argument that is different each time.

The choice of distribution has consequences: a count model whose variance is too small makes every error bar computed from it too narrow. Three results matter well beyond this section: the Poisson limit together with its waiting times, the central limit theorem, and the conditioning formula for Gaussian vectors. The discrete uniform, the Gamma and the Laplace distributions can be skimmed on a first reading.

Figure 36.10 shows how the distributions of this section are built from one another.

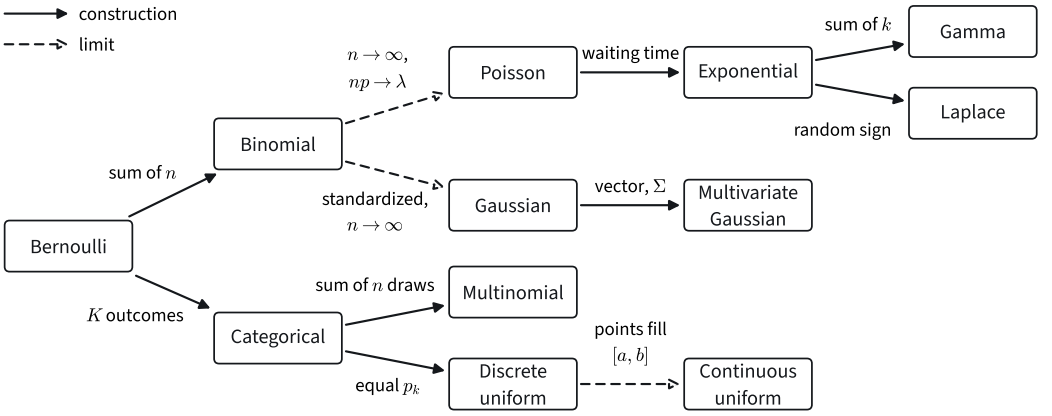


Figure 36.10: Every distribution of this section grows out of one coin flip. Arrows with filled heads construct a new distribution (a sum, a waiting time, a random sign, a vector of coordinates, equal probabilities); dashed arrows with open heads take a limit as the number of flips or points grows. The Poisson and the exponential describe one stream of events, counted and timed.

36.3.1 Discrete Distributions

A discrete random variable takes values in a countable set such as $\{0, 1\}$ or $\{0, 1, 2, \dots\}$. Its distribution is a probability mass function $P(X = k) \geq 0$ with $\sum_k P(X = k) = 1$, its mean is $\mathbb{E}[X] = \sum_k k\, P(X = k)$, and its variance is $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$. Figure 36.11 shows the distributions of this part in the order of the text.

36.3.1.1 Bernoulli

The simplest random quantity has two outcomes: a flip comes up heads or tails, a user clicks or does not, a prediction is right or wrong. One number describes it completely, the probability p of the outcome we call 1.

A *Bernoulli* random variable $X \sim \mathrm{Bernoulli}(p)$ takes the value 1 with probability p and 0 with probability $1 - p$, where $0 \leq p \leq 1$. Both cases fit in one formula,

$$P(X = x) = p^x (1 - p)^{1 - x}, \qquad x \in \{0, 1\}. \tag{36.33}$$

The exponent selects the factor: for $x = 1$ the formula reads p, and for $x = 0$ it reads $1 - p$.

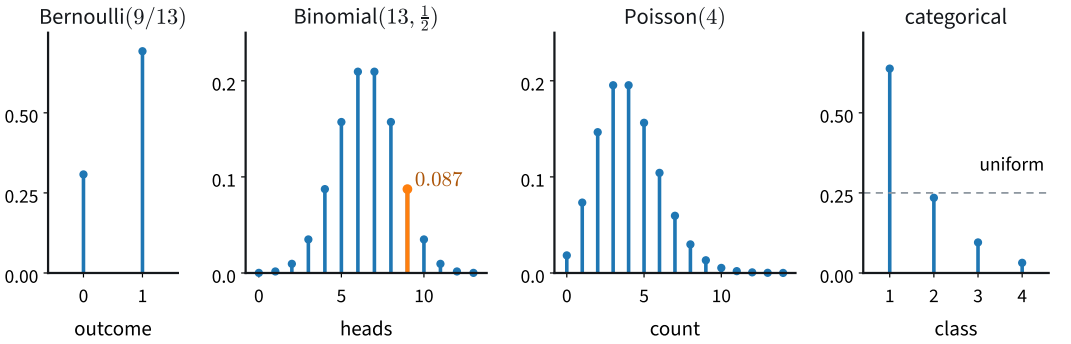


Figure 36.11: Four probability mass functions on the running examples. Bernoulli$(9/13)$: one flip of a coin with the observed frequency of heads. Binomial$(13, \frac{1}{2})$: the number of heads in 13 fair flips; the orange stem is $P(X = 9) = 715/8192 \approx 0.087$. Poisson$(4)$: requests per minute at an average of four. A categorical over four classes from the logits $(2, 1, 0.1, -1)$, against the uniform $1/4$. The binomial and the Poisson peak near their means, 6.5 and 4, and share a vertical scale; the two panels with few outcomes share another.

The moments are those of Section 1.6: the mean is $\mathbb{E}[X] = p$, and because $X^2 = X$ the variance is $\mathrm{Var}(X) = p - p^2 = p(1-p)$, the product of the two probabilities, which is largest, $\frac{1}{4}$, for the fair coin.

A binary classifier outputs an estimate $\hat{p}$ of the probability that the label y is 1. By Equation 36.33, the negative log-probability it assigns to the observed label is $-\log P(y) = -y \log \hat{p} - (1-y)\log(1-\hat{p})$, the *binary cross-entropy*. A classifier trained with this loss is fitting a Bernoulli distribution to each label.

Two coins appear in what follows, and they should be kept apart. The question of the opening is about a fair coin, $p = \frac{1}{2}$. The coin we would guess from the data has $p = 9/13 \approx 0.69$, and one flip of it has variance $\frac{9}{13} \cdot \frac{4}{13} \approx 0.21$.

36.3.1.2 Binomial

The question of the opening is about a count: how many heads in 13 fair flips, and how likely are exactly 9?

Start with two flips. The outcomes are TT, HT, TH and HH, so

$$P(X = 0) = (1-p)^2, \qquad P(X = 1) = 2p(1-p), \qquad P(X = 2) = p^2.$$

The factor 2 counts the two orderings of one head and one tail, and the three probabilities sum to $(p + 1 - p)^2 = 1$.

In general, the number of heads in n flips is a sum of n indicator variables, one per flip. If the flips $X_1, \dots, X_n$ are independent and each is Bernoulli(p) with the same p, their sum has the *binomial* distribution,

$$X = \sum_{i=1}^{n} X_i \sim \text{Binomial}(n, p). \tag{36.34}$$

Exactly k heads can occur in $\binom{n}{k}$ orderings, one for each choice of which k flips are heads, and by independence each ordering has probability $p^k(1-p)^{n-k}$. Hence

$$P(X = k) = \binom{n}{k} p^k (1-p)^{n-k}, \qquad k = 0, 1, \dots, n. \tag{36.35}$$

The probability of a count is the number of orderings times the probability of one ordering. For $n = 2$ it gives the three numbers above. For the fair coin, 9 heads in 13 flips can occur in $\binom{13}{9} = 715$ orderings, each of probability $2^{-13} = 1/8192$, so

$$P(X = 9) = \frac{715}{8192} \approx 0.087.$$

Nine heads *or more* has probability $(715 + 286 + 78 + 13 + 1)/8192 = 1093/8192 \approx 0.133$. Whether a tail probability of 0.133 is reason to doubt the coin is a question for statistics, which this section does not take up.

Mean and variance. Summing k times the pmf would be laborious. The construction Equation 36.34 gives both moments in one line each. Expectation is linear for any random variables, dependent or not, so

$$\mathbb{E}[X] = \sum_{i=1}^{n} \mathbb{E}[X_i] = np.$$

Variance adds over *independent* summands, because the covariance of two independent variables is zero (Section 36.2), so

$$\text{Var}(X) = \sum_{i=1}^{n} \text{Var}(X_i) = np(1-p).$$

So n flips have n times the mean and n times the variance of one flip. For the fair coin the mean is 6.5, the variance 3.25 and the standard deviation $\sqrt{3.25} \approx 1.80$, so 9 heads lie $2.5/1.80 \approx 1.4$ standard deviations above the mean.

The two formulas need different hypotheses, and the difference is easy to forget. The mean np holds even if the flips are dependent, and with different probabilities p_i it becomes $\sum_i p_i$. The variance needs independence. If the 13 "flips" were one flip copied 13 times, X would be 0 or 13, and its variance would be $13^2 \cdot \frac{1}{4} = 42.25$, thirteen times $np(1-p)$.

The successive ratio. The pmf can be computed without any factorial. When we divide consecutive binomial coefficients, the factorials cancel:

$$\frac{\binom{n}{k+1}}{\binom{n}{k}} = \frac{k!\,(n-k)!}{(k+1)!\,(n-k-1)!} = \frac{n-k}{k+1}, \qquad \text{so} \qquad \frac{P(X=k+1)}{P(X=k)} = \frac{n-k}{k+1} \cdot \frac{p}{1-p}.$$

The ratio counts orderings: a set of $k+1$ heads arises from $k+1$ different sets of k heads by adding one of the $n-k$ remaining flips, and the factor $p/(1-p)$ trades the probability of a tail for that of a head. Starting from $P(X=0)=(1-p)^n$ and multiplying by the ratio builds the whole pmf. The ratio exceeds 1 while $k+1<(n+1)p$ and falls below 1 afterwards, so the pmf rises to a single peak near the mean and then falls.

The recursion takes three lines of code. We keep it as a function, because the Poisson limit and the Gaussian bell below are both measured against it.

```
def binomial_pmf(n, p, kmax):
    """P(X=k) for k = 0..kmax, built from P(X=0) by the successive ratio."""
    pmf = [(1 - p) ** n]
    for k in range(kmax):
        pmf.append(pmf[-1] * max(n - k, 0) / (k + 1) * p / (1 - p))
    return np.array(pmf)

n, p = 13, 0.5
pmf = binomial_pmf(n, p, n)
print(f'P(X=9): by the ratio {pmf[9]:.4f}, by counting {math.comb(13, 9) / 2**13:.4f}')
print(f'P(X>=9) = {pmf[9:].sum():.4f}')
print(f'mean np = {n * p}, variance np(1-p) = {n * p * (1 - p)}')
```

```
P(X=9): by the ratio 0.0873, by counting 0.0873
P(X>=9) = 0.1334
mean np = 6.5, variance np(1-p) = 3.25
```

The ratio reproduces $715/8192 \approx 0.0873$ and the tail probability 0.1334.

36.3.1.3 Poisson

Some counts have no natural number of trials. A web server receives requests at an average rate of $\lambda = 4$ per minute, and nobody fixes in advance how many requests may arrive. What is the chance of a minute with no request at all, or of a burst of 10 or more?

Slice the minute into n short pieces. The model makes two assumptions about them: a piece is so short that it never holds two requests, and the pieces are independent trials with the same probability $4/n$ of a request. The count is then Binomial$(n, 4/n)$. Its mean is $n \cdot \frac{4}{n} = 4$ for every n, and its variance $4(1-\frac{4}{n})$ approaches 4. The chance of a silent minute is $(1-\frac{4}{n})^n$, which is 0.0115 at $n=20$, 0.0169 at $n=100$ and 0.0182 at $n=1000$, approaching $e^{-4} \approx 0.0183$. The numbers settle as the slices shrink, which suggests that the binomial has a limit.

Derivation of the limit. Take Binomial$(n, \lambda/n)$ and fix k. The pmf splits into four factors, three of which have simple limits:

$$\binom{n}{k}\left(\frac{\lambda}{n}\right)^k\left(1-\frac{\lambda}{n}\right)^{n-k} = \underbrace{\frac{n(n-1)\cdots(n-k+1)}{n^k}}_{\to 1}\cdot\frac{\lambda^k}{k!}\cdot\underbrace{\left(1-\frac{\lambda}{n}\right)^n}_{\to e^{-\lambda}}\cdot\underbrace{\left(1-\frac{\lambda}{n}\right)^{-k}}_{\to 1}.$$

The first factor is a product of k ratios that each tend to 1, the third is the limit $(1-\lambda/n)^n \to e^{-\lambda}$ that defines the exponential function, and the last tends to 1 because k is fixed. The

same computation works for any success probabilities p_n with $np_n \to \lambda$, since then $(1 - p_n)^n \to e^{-\lambda}$ as well. This limit is the *law of rare events*. The surviving factors are the *Poisson* pmf with rate $\lambda > 0$,

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \qquad k = 0, 1, 2, \ldots \tag{36.36}$$

and we write $X \sim \text{Poisson}(\lambda)$. The Poisson describes a count made of many independent chances, each small, with a fixed expected total λ. Its probabilities sum to one because $\sum_{k \geq 0} \lambda^k / k! = e^{\lambda}$.

Mean and variance. We compute the moments from the pmf directly. The factor k cancels against the $k!$, and the index shift $j = k - 1$ turns the remaining sum into the total probability of a Poisson:

$$\mathbb{E}[X] = \sum_{k \geq 1} k \, \frac{e^{-\lambda} \lambda^k}{k!} = \lambda \sum_{k \geq 1} \frac{e^{-\lambda} \lambda^{k-1}}{(k-1)!} = \lambda \sum_{j \geq 0} \frac{e^{-\lambda} \lambda^j}{j!} = \lambda.$$

The same shift applied twice gives $\mathbb{E}[X(X-1)] = \lambda^2$, so

$$\text{Var}(X) = \mathbb{E}[X(X-1)] + \mathbb{E}[X] - \mathbb{E}[X]^2 = \lambda^2 + \lambda - \lambda^2 = \lambda.$$

The Poisson's mean and variance are equal. For the server, $P(X = 0) = e^{-4} \approx 0.018$, $P(X = 4) = \frac{4^4}{4!} e^{-4} \approx 0.195$ and $P(X \geq 10) \approx 0.008$: a silent minute occurs about once an hour, and a burst of ten or more about once in two hours. If request counts measured over many minutes have a variance several times their mean, one of the two assumptions has failed (the rate varies from minute to minute, or requests trigger further requests), and error bars computed from the Poisson are too narrow. The usual repair, a Poisson whose rate is itself random, is exercise 3.

The Poisson process. Run the slicing over the whole timeline, with rate λ per unit time. A window of length t holds nt slices, so its count is Binomial$(nt, \lambda/n)$, which tends to Poisson(λt). Windows that do not overlap use disjoint sets of slices, so their counts are independent. This stream of events is the *Poisson process* with rate λ, and the exponential distribution below describes the times between its events.

The Poisson pmf has a successive ratio of its own, $P(X = k+1)/P(X = k) = \lambda/(k+1)$. The cell builds the pmf from it, prints the three probabilities for the server, and simulates 100,000 minutes.

```
rng = np.random.default_rng()

def poisson_pmf(lam, kmax):
    """P(X=k) for k = 0..kmax, built from P(X=0) = e^-lam by the ratio lam/(k+1)."""
    pmf = [math.exp(-lam)]
    for k in range(kmax):
        pmf.append(pmf[-1] * lam / (k + 1))
    return np.array(pmf)
```

```
pmf = poisson_pmf(4.0, 60)
print(f'P(0) = {pmf[0]:.3f}, P(4) = {pmf[4]:.3f}, P(X>=10) = {pmf[10:].sum():.3f}')
counts = rng.poisson(4.0, 100000)
print(f'100,000 simulated minutes: mean {counts.mean():.2f}, '
      f'variance {counts.var(ddof=1):.2f}')
```

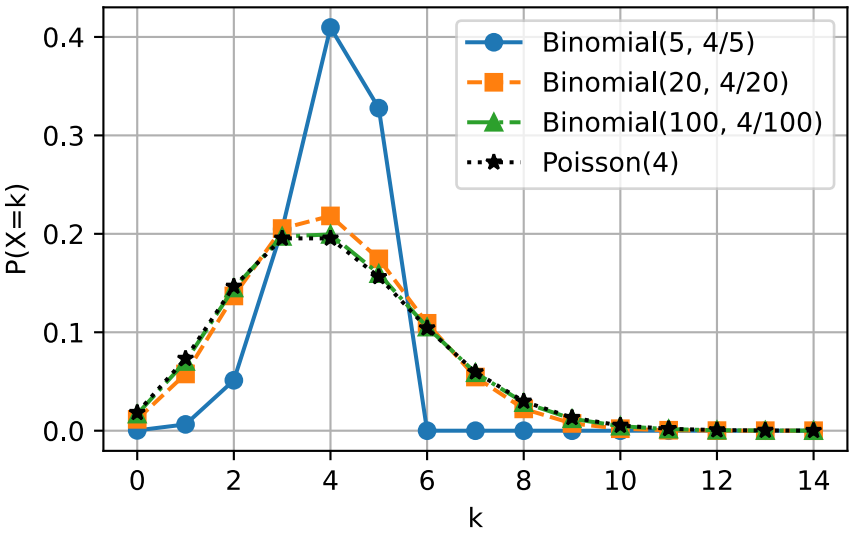
```
P(0) = 0.018, P(4) = 0.195, P(X>=10) = 0.008
100,000 simulated minutes: mean 4.00, variance 4.01
```

The simulated counts have mean and variance both close to 4.

The binomial-to-Poisson limit. How fast does $\text{Binomial}(n, 4/n)$ approach $\text{Poisson}(4)$? The plot overlays the two, and the cell measures the gap by the *total-variation distance* $\frac{1}{2}\sum_k |P_{\text{Bin}}(k) - P_{\text{Poi}}(k)|$, which is the largest difference between the probabilities that the two give to any one event.

```
ns, ks = (5, 20, 100), np.arange(15)
d2l.plot(ks, [binomial_pmf(n, 4 / n, 14) for n in ns] + [poisson_pmf(4.0, 14)],
         xlabel='k', ylabel='P(X=k)', fmts=('o-', 's--', '^-.', 'k*:'),
         legend=[f'Binomial({n}, 4/{n})' for n in ns] + ['Poisson(4)'],
         figsize=(5, 3))
for n in (20, 100, 1000):
    tv = 0.5 * np.abs(binomial_pmf(n, 4 / n, 60) - poisson_pmf(4.0, 60)).sum()
    print(f'n = {n:4d}: total-variation distance {tv:.3f}')
```

```
n =   20: total-variation distance 0.056
n =  100: total-variation distance 0.010
n = 1000: total-variation distance 0.001
```



At $n = 5$ the binomial cannot exceed five requests, and its peak is too high. By $n = 100$ the two pmfs coincide to plotting accuracy. The distance is 0.056 at $n = 20$, 0.010 at $n = 100$ and 0.001 at $n = 1000$, so on these three values it falls in proportion to $1/n$.

36.3.1.4 Categorical and Multinomial

A classifier with K classes, or a language model choosing the next token, has more than two outcomes. The categorical distribution is the coin with K faces.

A *categorical* random variable takes values in $\{1, \dots, K\}$ with probabilities given by a vector $\mathbf{p} = (p_1, \dots, p_K)$, where $p_k \geq 0$ and $\sum_k p_k = 1$:

$$P(X = k) = p_k. \tag{36.37}$$

For $K = 2$ it is the Bernoulli. A network produces the p_k from real-valued scores $z_1, \dots, z_K$, the *logits*, through the *softmax*

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}. \tag{36.38}$$

Exponentiating makes every score positive, and dividing by the sum makes the scores add to one. The logits $\mathbf{z} = (2, 1, 0.1, -1)$, for example, give $\mathbf{p} \approx (0.638, 0.235, 0.095, 0.032)$. With the label written as a one-hot vector $\mathbf{y}$, the negative log-probability of the observed class is $-\log p_y = -\sum_k y_k \log p_k$, the *softmax cross-entropy* of Section 3.1.2. Training a classifier with it fits a categorical distribution to each label.

The discrete uniform. When all probabilities are equal, the categorical on $\{1, \dots, n\}$ with $p_k = 1/n$ is the *discrete uniform* $\text{Unif}\{1, \dots, n\}$: a fair die, or an index chosen uniformly at random. Its moments follow from the sums $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ and $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$:

$$\mathbb{E}[X] = \frac{1}{n}\sum_{i=1}^{n} i = \frac{n+1}{2}, \qquad \text{Var}(X) = \frac{(n+1)(2n+1)}{6} - \frac{(n+1)^2}{4} = \frac{(n+1)(2(2n+1) - 3(n+$$

The mean is the midpoint, and the variance grows like n^2 because the values spread over a range of length $n - 1$. A die has mean 3.5 and variance $35/12 \approx 2.92$.

The multinomial. Draw n independent categorical samples and count how often each class occurs. The counts $\mathbf{x} = (x_1, \dots, x_K)$, with $\sum_k x_k = n$, have the *multinomial* pmf

$$P(\mathbf{x}) = \frac{n!}{x_1! \cdots x_K!} \prod_{k=1}^{K} p_k^{x_k}. \tag{36.39}$$

As for the binomial, the coefficient counts the orderings of the n draws that produce these counts, and the product is the probability of one ordering. For $K = 2$ the coefficient is $\binom{n}{x_1}$, and the multinomial is the binomial again: $13!/(9!\,4!) = 715$. Each count on its own is $\text{Binomial}(n, p_k)$, since a draw either lands in class k or not, so $\mathbb{E}[x_k] = np_k$ and $\text{Var}(x_k) = np_k(1-p_k)$. The whole count vector has mean $n\mathbf{p}$ and covariance $n(\text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top)$, which exercise 10 derives; its off-diagonal entries are negative because the classes compete for the same n draws. With $n = 1$ the count vector is the one-hot vector of a single categorical draw.

The cell computes the softmax of the example logits and draws $10{,}000$ classes from the resulting categorical distribution.

```
z = np.array([2.0, 1.0, 0.1, -1.0])          # logits for K = 4 classes
p_cat = np.exp(z) / np.exp(z).sum()           # softmax
draws = rng.choice(4, size=10000, p=p_cat)
print('softmax p   =', p_cat.round(3))
print('frequencies =', (np.bincount(draws, minlength=4) / 10000).round(3))
```

```
softmax p   = [0.638 0.235 0.095 0.032]
frequencies = [0.639 0.234 0.095 0.033]
```

The frequencies agree with **p** to within about a percentage point.

36.3.2 Continuous Distributions

A continuous random variable has a density $p(x) \geq 0$ with $\int p(x)\,dx = 1$. Probabilities are areas under the density, and means and variances are integrals, as in Section 36.1. We start with the exponential, because the Poisson process has already built it for us.

36.3.2.1 Exponential

Return to the server. The Poisson tells us how many requests arrive in a minute. An engineer choosing a timeout asks a different question about the same stream of requests: how long until the next one?

The two questions share an answer. The wait for the next request exceeds one minute exactly when the minute contains no request, so $P(\text{wait} > 1) = P(N = 0) = e^{-4} \approx 0.018$, the same number as the Poisson's $P(0)$. Figure 36.12 shows the one stream of events that both distributions describe.

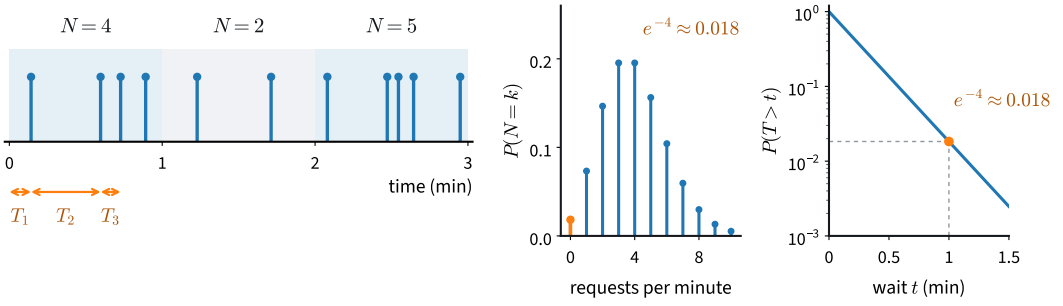


Figure 36.12: One stream of requests at an average rate of four per minute, seen two ways. (a) Event times over three minutes; the unit windows hold $N = 4$, 2 and 5 requests, and T_1, T_2, T_3 are the first three waits. (b) The number of requests in a minute is Poisson(4); a silent minute has probability $e^{-4} \approx 0.018$. (c) The probability that a wait exceeds t minutes is e^{-4t}, a straight line on the logarithmic axis; a wait longer than a minute and a silent minute are the same event.

The same argument works for a wait of any length $t \geq 0$. In a Poisson process with rate λ, the wait T from time 0 to the first event exceeds t exactly when the window $[0, t]$ is empty, and the count in that window is Poisson(λt). Hence

$$P(T > t) = P(N(t) = 0) = e^{-\lambda t}, \qquad t \geq 0.$$

The cumulative distribution function is $F(t) = 1 - e^{-\lambda t}$, and its derivative is the density of the *exponential* distribution with rate $\lambda > 0$,

$$p(t) = \lambda e^{-\lambda t} \quad (t \geq 0), \qquad p(t) = 0 \quad (t < 0), \qquad F(t) = 1 - e^{-\lambda t}, \qquad T \sim \text{Exp}(\lambda). \tag{36.40}$$

The density is the rate at which the probability of an empty window decays, so the process determines it. After each event the future slices of the process are fresh independent trials, so the wait after any event has the same distribution: the gaps between consecutive events are independent $\text{Exp}(\lambda)$ variables.

Mean and variance. Two integrals by parts do the work: $\int_0^\infty u\, e^{-u}\, du = [-ue^{-u}]_0^\infty + \int_0^\infty e^{-u}\, du = 1$ and $\int_0^\infty u^2 e^{-u}\, du = [-u^2 e^{-u}]_0^\infty + 2\int_0^\infty u\, e^{-u}\, du = 2$. Substituting $u = \lambda t$,

$$\mathbb{E}[T] = \int_0^\infty t\, \lambda e^{-\lambda t}\, dt = \frac{1}{\lambda}, \qquad \mathbb{E}[T^2] = \frac{2}{\lambda^2}, \qquad \text{Var}(T) = \frac{2}{\lambda^2} - \frac{1}{\lambda^2} = \frac{1}{\lambda^2}.$$

More frequent events mean shorter waits. At four requests per minute the mean wait is $\frac{1}{4}$ minute, or 15 seconds. The median m solves $F(m) = \frac{1}{2}$, so $m = \log 2/\lambda \approx 10.4$ seconds: half of all waits are shorter than about 10 seconds even though the mean is 15. The density is largest at $t = 0$ and has a long right tail, which pulls the mean above the median.

Memorylessness. Having already waited s minutes without a request, the chance of waiting t minutes more is the same as at the start:

$$P(T > s + t \mid T > s) = \frac{P(T > s + t)}{P(T > s)} = \frac{e^{-\lambda(s+t)}}{e^{-\lambda s}} = e^{-\lambda t} = P(T > t).$$

Time already spent waiting tells us nothing about the time still to wait. Among continuous distributions on $(0, \infty)$, only the exponential has this property; exercise 4 proves it. The property also marks the limit of the model. A light bulb whose filament thins with use is more likely to fail in the next hour the older it is, so its lifetime is not exponential.

The Gamma distribution. The time T_k of the k-th request is the sum of k independent waits. By the binomial's argument, linearity and independence, its mean is k/λ and its variance k/λ^2. Its density also follows from the process. The k-th event comes after t exactly when at most $k - 1$ events fall in $[0, t]$:

$$P(T_k > t) = P(N(t) \leq k - 1) = \sum_{j=0}^{k-1} e^{-\lambda t} \frac{(\lambda t)^j}{j!}.$$

Differentiating the j-th term gives $-\lambda e^{-\lambda t}\frac{(\lambda t)^j}{j!} + \lambda e^{-\lambda t}\frac{(\lambda t)^{j-1}}{(j-1)!}$, whose second part cancels the first part of the term before it. The sum telescopes, and only the first part of the last term survives:

$$p(t) = -\frac{d}{dt}P(T_k > t) = \frac{\lambda^k\, t^{k-1} e^{-\lambda t}}{(k-1)!}, \qquad t > 0.$$

For $k = 1$ this is the exponential density. To allow values of k that are not integers, replace $(k-1)!$ by the *Gamma function* $\Gamma(z) = \int_0^\infty u^{z-1} e^{-u}\, du$ for $z > 0$. Integration by parts gives $\Gamma(z+1) = z\,\Gamma(z)$, and $\Gamma(1) = 1$, so $\Gamma(n+1) = n!$. The *Gamma distribution* with shape $\alpha > 0$ and rate $\beta > 0$ has the density

$$p(t) = \frac{\beta^\alpha\, t^{\alpha-1} e^{-\beta t}}{\Gamma(\alpha)}, \qquad t > 0, \qquad T \sim \text{Gamma}(\alpha, \beta). \tag{36.41}$$

The substitution $u = \beta t$ turns its integral into $\Gamma(\alpha)$, so it integrates to one. The same substitution gives $\mathbb{E}[T] = \Gamma(\alpha+1)/(\beta\,\Gamma(\alpha)) = \alpha/\beta$ and $\text{Var}(T) = \alpha/\beta^2$, which for $\alpha = k$, $\beta = \lambda$ are the moments of the k-th waiting time. For the server, the 4th request arrives after a mean of 1 minute with standard deviation 0.5 minutes, and $P(T_4 > 1) = P(N(1) \le 3) \approx 0.43$: in 43% of minutes, fewer than four requests arrive, although four is the average.

Sampling. The inverse-transform method of Section 36.1 turns a uniform draw U into a draw from any continuous distribution with an invertible cumulative distribution function F, by setting $X = F^{-1}(U)$. For the exponential, $F^{-1}(q) = -\log(1-q)/\lambda$, so $T = -\log(1-U)/\lambda$. The variable $1 - U$ has the same distribution as U. We use it because generators return values in $[0, 1)$, so $U = 0$ can occur, and $-\log U$ would then be infinite.

If the construction is right, it can be run backwards: draw exponential gaps, and Poisson counts must come out. The cell draws $60{,}000$ gaps at rate 4, adds them up into event times, and counts the events in each of the first $10{,}000$ minutes.

```
lam = 4.0                                       # requests per minute
gaps = -np.log(1 - rng.random(60000)) / lam     # inverse transform: Exp(lam)
times = np.cumsum(gaps)                         # event times, about 15,000 min
counts = np.bincount(times[times < 10000].astype(int), minlength=10000)
print(f'requests per minute: mean {counts.mean():.2f}, variance {counts.var(ddof=1):.2f}')
print(f'silent minutes {np.mean(counts == 0):.4f},   waits over 1 min '
      f'{np.mean(gaps > 1):.4f},   e^-4 = {math.exp(-4):.4f}')
print(f'mean wait {gaps.mean():.3f} min,   minutes with at most 3 requests '
      f'{np.mean(counts <= 3):.3f}')
```

```
requests per minute: mean 3.99, variance 4.00
silent minutes 0.0204,   waits over 1 min 0.0191,   e^-4 = 0.0183
mean wait 0.252 min,   minutes with at most 3 requests 0.434
```

The counts have mean and variance close to 4. The fraction of silent minutes and the fraction of waits longer than a minute are both close to e^{-4}, and the fraction of minutes with at most three requests is close to the Gamma value $P(T_4 > 1) \approx 0.43$. Gaps drawn as exponentials produce Poisson counts.

36.3.2.2 Uniform

Every sampler in this section starts from the same raw material, a number drawn uniformly from $[0, 1)$. Its distribution, and that of a number drawn uniformly from any other interval,

is the continuous uniform.

The *uniform* distribution $U(a,b)$ on an interval $[a,b]$ with $a<b$ has constant density, fixed by $\int p = 1$:

$$p(x) = \begin{cases} \frac{1}{b-a} & x \in [a,b], \\ 0 & \text{otherwise.} \end{cases} \tag{36.42}$$

Every subinterval of the same length has the same probability, its length divided by $b-a$. The moments follow by direct integration; for the variance, center the interval at its midpoint, so that it runs from $-c$ to c with $c = \frac{b-a}{2}$:

$$\mathbb{E}[X] = \int_a^b \frac{x}{b-a}\,dx = \frac{b^2-a^2}{2(b-a)} = \frac{a+b}{2}, \qquad \mathrm{Var}(X) = \int_{-c}^{c} \frac{y^2}{2c}\,dy = \frac{c^2}{3} = \frac{(b-a)^2}{12}.$$

$U(0,1)$ has mean $\frac{1}{2}$ and variance $\frac{1}{12}$, and $U(-\sqrt{3},\sqrt{3})$ has variance $(2\sqrt{3})^2/12 = 1$. The formula continues the discrete uniform: n equally likely points spaced h apart have variance $h^2(n^2-1)/12$, which tends to $(b-a)^2/12$ as the points fill $[a,b]$ with $nh = b-a$. Through the inverse transform, $U(0,1)$ becomes any continuous distribution; the exponential sampler above is the example. The uniform has hard edges and no tails, so it looks nothing like a Gaussian. For that reason we use it below as the summand that the central limit theorem turns into one.

36.3.2.3 Gaussian

The successive ratio computes any binomial probability, but it does not tell us what the distribution looks like. Is there a formula for its shape near the peak? There is, and it turns out that only the mean and the variance of the coin enter it. This is the most important distribution of the section.

The log-ratio near the mean. Write $q = 1-p$ and measure the count from the mean, $k = np + x$. The successive ratio of the binomial becomes

$$\frac{P(X=k+1)}{P(X=k)} = \frac{(n-k)\,p}{(k+1)\,q} = \frac{(nq-x)\,p}{(np+x+1)\,q} = \frac{1-x/(nq)}{1+(x+1)/(np)}.$$

For x of the order of the standard deviation $\sqrt{npq}$, both $x/(nq)$ and $(x+1)/(np)$ are small, and $\log(1+\epsilon) \approx \epsilon$ gives

$$\log\frac{P(X=k+1)}{P(X=k)} \approx -\frac{x}{nq} - \frac{x+1}{np} = -\frac{x}{npq} - \frac{1}{np},$$

where we used $\frac{1}{nq} + \frac{1}{np} = \frac{p+q}{npq} = \frac{1}{npq}$. The step from $\log P(X=k)$ to $\log P(X=k+1)$ is, to leading order, linear in the distance x from the mean, with slope $-1/(npq)$. A function whose steps grow linearly is a parabola. Indeed, the parabola $-x^2/(2npq)$ changes by

$$-\frac{(x+1)^2 - x^2}{2npq} = -\frac{x}{npq} - \frac{1}{2npq}$$

between x and $x+1$. This differs from the step of $\log P$ by the constant $(p-q)/(2npq)$, which is zero for a fair coin and in general adds up to a quantity of order $1/\sqrt{n}$ over the $\sqrt{npq}$ steps that matter. Therefore

$$\log P(X = np + x) \approx C - \frac{x^2}{2npq}$$

with a constant C, at every count $k = np + x$ within a few standard deviations of the mean, whether or not np is an integer (for the coin, $np = 6.5$). Near its peak the binomial is a bell, $e^C e^{-x^2/(2npq)}$, centered at the mean and as wide as its standard deviation $\sqrt{npq}$. We have been informal about the remainders, so this is a sketch and not a proof. It is the idea behind the *de Moivre–Laplace theorem*, the first central limit theorem, which makes the approximation precise for x of order $\sqrt{npq}$.

The normalizer. The constant C follows from the fact that probabilities sum to one. Counts are spaced one apart, so the sum of the bell over all counts is close to the integral $e^C \int e^{-x^2/(2npq)}\,dx$, and we need $I = \int_{-\infty}^{\infty} e^{-x^2/2}\,dx$. The function $e^{-x^2/2}$ has no elementary antiderivative, and the integral looks hopeless. The trick is to compute I^2 instead. It is an integral over the plane, and polar coordinates $x = r\cos\theta$, $y = r\sin\theta$, with area element $dx\,dy = r\,dr\,d\theta$, make it elementary:

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)/2}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} e^{-r^2/2}\,r\,dr\,d\theta = 2\pi\left[-e^{-r^2/2}\right]_0^{\infty} = 2\pi.$$

So $I = \sqrt{2\pi}$. The factor r of the polar area element gives $r\,e^{-r^2/2}$ the simple antiderivative $-e^{-r^2/2}$, which $e^{-x^2/2}$ alone does not have. Substituting $x = \sqrt{npq}\,u$ gives $\int e^{-x^2/(2npq)}\,dx = \sqrt{2\pi npq}$, so $e^C = 1/\sqrt{2\pi npq}$ and

$$P(X = np + x) \approx \frac{1}{\sqrt{2\pi npq}}\,e^{-x^2/(2npq)}.$$

For the coin, $npq = 3.25$ and $x = 2.5$ give $e^{-2.5^2/6.5}/\sqrt{2\pi \cdot 3.25} \approx 0.0846$, against the exact 0.0873. Thirteen flips are few. The cell measures the bell against the exact pmf of fair coins as n grows.

```
for n in (13, 100, 1000):
    k = np.arange(n + 1)
    bell = np.exp(-(k - n / 2)**2 / (n / 2)) / np.sqrt(np.pi * n / 2)
    exact = binomial_pmf(n, 0.5, n)
    peak = n // 2
    print(f'n = {n:4d}: relative error at the peak {abs(bell[peak] / exact[peak] - 1):.2%}')
print(f'n = 13, k = 9: bell {np.exp(-2.5**2 / 6.5) / np.sqrt(np.pi * 6.5):.4f},'
      f' exact {binomial_pmf(13, 0.5, 13)[9]:.4f}')
```

```
n =   13: relative error at the peak 1.66%
n =  100: relative error at the peak 0.25%
n = 1000: relative error at the peak 0.03%
n = 13, k = 9: bell 0.0846, exact 0.0873
```

The bell is 3% low at $k = 9$ for 13 flips, and its relative error at the peak is 1.7% at $n = 13$, 0.25% at $n = 100$ and 0.03% at $n = 1000$, falling like $1/n$.

The standard Gaussian. The parabola suggests rescaling. The standardized count $Y^{(n)} = (X - np)/\sqrt{npq}$ has mean 0 and variance 1 for every n, and its values are spaced $1/\sqrt{npq}$ apart. The bell says that $P(X = k) \approx \phi(y)/\sqrt{npq}$ at $y = (k - np)/\sqrt{npq}$, a density $\phi(y)$ times the spacing, with

$$\phi(y) = \frac{1}{\sqrt{2\pi}}\, e^{-y^2/2}.$$

This is the density of the *standard Gaussian* $\mathcal{N}(0, 1)$, the limit of the standardized binomial. It integrates to one because $I = \sqrt{2\pi}$, and its mean is 0 by symmetry. For the variance, integrate by parts with $u = x$ and $dv = x\, e^{-x^2/2}\, dx$:

$$\int_{-\infty}^{\infty} x^2\, \frac{e^{-x^2/2}}{\sqrt{2\pi}}\, dx = \frac{1}{\sqrt{2\pi}} \Big(\big[-x\, e^{-x^2/2} \big]_{-\infty}^{\infty} + \int_{-\infty}^{\infty} e^{-x^2/2}\, dx \Big) = 0 + 1 = 1.$$

The standard Gaussian has variance 1. Shifting by μ and scaling by $\sigma > 0$ gives the variable $X = \mu + \sigma Y$, whose density by the change-of-variables formula of Section 36.2 is $\phi((x - \mu)/\sigma)/\sigma$, the *Gaussian* (or normal) distribution

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, \exp\!\Big(-\frac{(x - \mu)^2}{2\sigma^2} \Big), \qquad X \sim \mathcal{N}(\mu, \sigma^2). \tag{36.43}$$

Since $X = \mu + \sigma Y$, its mean is μ and its variance σ^2: the two parameters are the first two moments, and σ is the width of the bell.

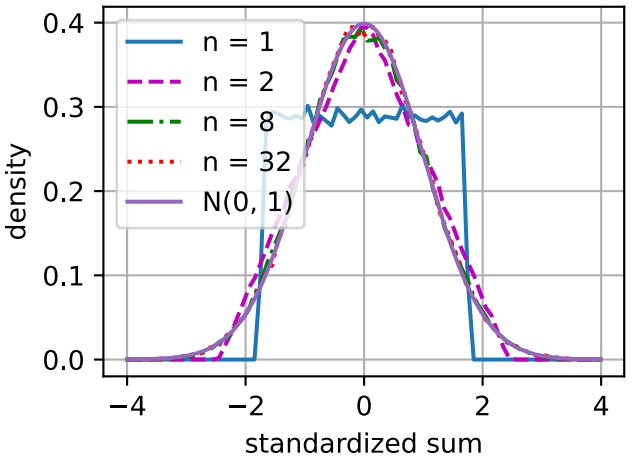
The central limit theorem. The derivation used little about coin flips beyond their mean and variance, and the general theorem needs only these two moments. Let $X_1, X_2, \ldots$ be independent and identically distributed with mean μ and variance $0 < \sigma^2 < \infty$, and let $S_n = \sum_{i=1}^{n} X_i$. Then for every y,

$$\lim_{n \to \infty} P\Big(\frac{S_n - n\mu}{\sigma\sqrt{n}} \le y \Big) = \Phi(y) = \int_{-\infty}^{y} \phi(u)\, du. \tag{36.44}$$

The standardized sum of many independent copies of any distribution with finite variance has, in the limit, the standard Gaussian distribution. For Bernoulli summands this is the de Moivre–Laplace theorem; we take the general proof as given (Wasserman 2013). Only the mean and the variance of the summands enter the limit, which is why sums of many small independent effects, such as measurement noise, so often look Gaussian.

Sums of uniforms. The theorem is easy to watch. The cell starts from $U(0, 1)$, flat with hard edges and no tails, forms sums of n draws, standardizes them by their mean $n/2$ and standard deviation $\sqrt{n/12}$, and compares their histograms with ϕ.

```
ns, x = (1, 2, 8, 32), np.arange(-4, 4, 0.01)
curves = []
for n in ns:                                    # standardized sums of n uniforms
    s = (rng.random((100000, n)).sum(1) - n / 2) / np.sqrt(n / 12)
    hist, edges = np.histogram(s, bins=80, range=(-4, 4), density=True)
    curves.append(np.interp(x, (edges[:-1] + edges[1:]) / 2, hist))
curves.append(np.exp(-x**2 / 2) / np.sqrt(2 * np.pi))     # N(0, 1) density
d2l.plot(x, curves, xlabel='standardized sum', ylabel='density',
         legend=[f'n = {n}' for n in ns] + ['N(0, 1)'])
```



At $n = 1$ we see the uniform's plateau and at $n = 2$ a triangle. By $n = 8$ the histogram is close to the Gaussian, and at $n = 32$ the two cannot be told apart at the resolution of the plot.

The hypothesis of a finite variance cannot be dropped. The averages of Cauchy draws in Section 36.1 never settle, however many are averaged, whereas an average of n standard Gaussians is $\mathcal{N}(0, 1/n)$.

Gaussian noise and squared error. A regression network that predicts $\hat{y}$ and models the label as $y \sim \mathcal{N}(\hat{y}, \sigma^2)$ assigns the observed label the negative log-probability $-\log p(y) = (y - \hat{y})^2/(2\sigma^2) + \frac{1}{2}\log(2\pi\sigma^2)$, which is the squared error up to a scale and a constant (Section 2.1.3).

36.3.2.4 Laplace

Squared error punishes a residual of 10 a hundred times more than a residual of 1, so a few outliers can dominate a fit. A loss that grows only linearly, the absolute error, corresponds to a different noise distribution (Section 2.1.3.3).

Let S be a fair random sign, $P(S = 1) = P(S = -1) = \frac{1}{2}$, and let E be an exponential variable with mean $b > 0$ (rate $1/b$), independent of S. The variable $X = \mu + SE$ lies above μ exactly when $S = 1$, and then $X - \mu = E$, so for $x > \mu$ its density is $\frac{1}{2} \cdot \frac{1}{b} e^{-(x-\mu)/b}$; by symmetry the same holds below μ with $|x - \mu|$. This is the *Laplace* distribution,

$$p(x) = \frac{1}{2b}\, e^{-|x-\mu|/b}, \qquad X \sim \text{Laplace}(\mu, b). \tag{36.45}$$

Each side carries half the mass of an exponential. The density is symmetric, has a sharp peak at μ, and decays exponentially on both sides.

Mean and variance. The construction gives both. The mean is μ because $\mathbb{E}[SE] = \mathbb{E}[S]\,\mathbb{E}[E] = 0$. Because $S^2 = 1$, the variance is the exponential's second moment, $2/\lambda^2$ with $\lambda = 1/b$:

$$\text{Var}(X) = \mathbb{E}\big[S^2 E^2\big] = \mathbb{E}[E^2] = 2b^2.$$

Laplace noise and absolute error. A regression network that models the label as $y \sim \text{Laplace}(\hat{y}, b)$ assigns it the negative log-probability $-\log p(y) = |y - \hat{y}|/b + \log 2b$, the absolute error divided by b plus a constant. As a prior on weights, the Laplace density gives the ℓ_1 penalty $\|\mathbf{w}\|_1$ used by the LASSO (Tibshirani 1996). For a location μ fitted to data $x_1, \dots, x_m$, the probability that the Laplace model assigns to the data is largest where $\sum_i |x_i - \mu|$ is smallest, which is at the sample median. A single outlier can move the sample mean arbitrarily far, but it moves the median by at most one data point.

Tails. Figure 36.13 compares the uniform, the Gaussian and the Laplace at the same variance. At matched variance $\sigma^2 = 2b^2$, the Laplace probability of landing more than three standard deviations from the center is $P(|X - \mu| > 3\sigma) = e^{-3\sigma/b} = e^{-3\sqrt{2}} \approx 0.014$, against 0.0027 for the Gaussian: about five times more.

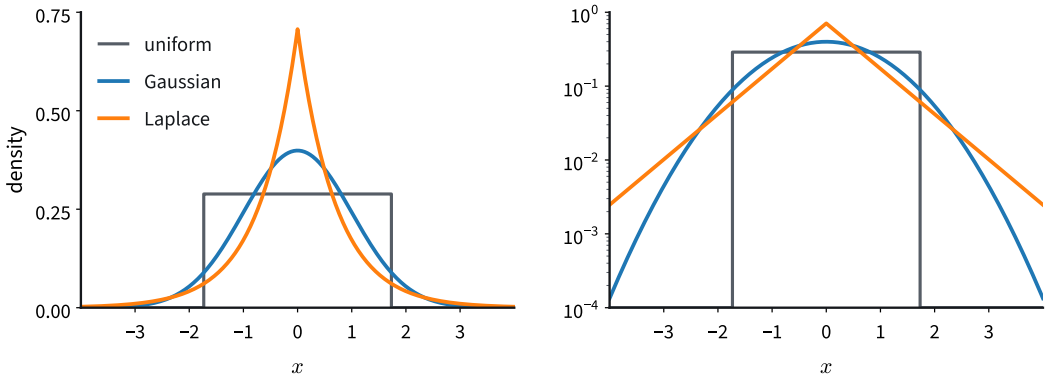


Figure 36.13: Three densities with variance one: the uniform on $[-\sqrt{3}, \sqrt{3}]$, the standard Gaussian, and the Laplace with $b = 1/\sqrt{2}$. (a) The densities. (b) The same on a logarithmic axis, where the Gaussian is a parabola, the Laplace two straight lines, and the uniform a plateau that ends at $\pm\sqrt{3}$. At three standard deviations the Laplace carries about five times the Gaussian's tail probability.

The cell samples the Laplace by its construction, a random sign times an exponential, and counts the draws beyond three standard deviations.

```
b, m = 1.0, 200000
lap = rng.choice([-1.0, 1.0], m) * rng.exponential(b, m)   # S * E
sigma = math.sqrt(2) * b                                    # matched sd
gau = rng.normal(0.0, sigma, m)
print(f'sample variances: Laplace {lap.var(ddof=1):.2f}, '
      f'Gaussian {gau.var(ddof=1):.2f}   (2b^2 = 2)')
print(f'P(|X| > 3 sd): Laplace {np.mean(np.abs(lap) > 3 * sigma):.4f}, '
      f'Gaussian {np.mean(np.abs(gau) > 3 * sigma):.4f}')
```

```
sample variances: Laplace 1.98, Gaussian 2.00   (2b^2 = 2)
P(|X| > 3 sd): Laplace 0.0140, Gaussian 0.0027
```

Both sample variances are close to 2, and the tail fractions are about 0.014 and 0.003.

36.3.3 The Multivariate Gaussian

The noise in an image, the latent code of a generative model, and the predictions of a model at many inputs are vectors, and their coordinates are correlated. The multivariate Gaussian is the distribution of such vectors that we can compute with. Our running example has two coordinates with variance 2 each and covariance 1,

$$\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}.$$

36.3.3.1 Contours and Eigenvectors

The *multivariate Gaussian* $\mathcal{N}(\mu, \Sigma)$ on $\mathbb{R}^d$ has a mean vector μ and a covariance matrix Σ that is symmetric and positive definite. Its density is

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\!\Big(-\tfrac{1}{2}(\mathbf{x}-\mu)^{\top}\Sigma^{-1}(\mathbf{x}-\mu)\Big), \tag{36.46}$$

where $|\Sigma|$ is the determinant. For $d = 1$ it is Equation 36.43. The density depends on $\mathbf{x}$ only through the quadratic form in the exponent, so it is constant on the sets where that form is constant. What do these sets look like?

The running example by hand. Turn the axes by 45° and use the sum and difference coordinates $u = (x_1 + x_2)/\sqrt{2}$ and $w = (x_1 - x_2)/\sqrt{2}$, with $\mu = \mathbf{0}$. Their variances and covariance follow from the entries of Σ:

$$\operatorname{Var}(u) = \frac{2 + 2 + 2\cdot 1}{2} = 3, \qquad \operatorname{Var}(w) = \frac{2 + 2 - 2\cdot 1}{2} = 1, \qquad \operatorname{Cov}(u, w) = \frac{2 - 2}{2} = 0.$$

In the turned coordinates the covariance is diagonal. The new axes are the eigenvectors $\mathbf{v}_1 = (1,1)/\sqrt{2}$ and $\mathbf{v}_2 = (1,-1)/\sqrt{2}$ of Σ, with eigenvalues $\gamma_1 = 3$ and $\gamma_2 = 1$; indeed $\Sigma\mathbf{v}_1 = (3,3)/\sqrt{2} = 3\mathbf{v}_1$.

The general case. The spectral theorem says that a symmetric matrix has an orthonormal basis of eigenvectors, $\Sigma = \mathbf{V}\,\mathrm{diag}(\gamma_1, \dots, \gamma_d)\mathbf{V}^{\top}$ with $\mathbf{V}^{\top}\mathbf{V} = \mathbf{I}$, and positive definiteness

means that every $\gamma_i > 0$. In the coordinates $\mathbf{u} = \mathbf{V}^\top(\mathbf{x} - \mu)$ the quadratic form becomes a weighted sum of squares,

$$(\mathbf{x} - \mu)^\top \Sigma^{-1} (\mathbf{x} - \mu) = \mathbf{u}^\top \operatorname{diag}(\gamma_1^{-1}, \dots, \gamma_d^{-1})\, \mathbf{u} = \sum_{i=1}^{d} \frac{u_i^2}{\gamma_i}.$$

The set where this sum equals r^2 is an ellipsoid with axes along the eigenvectors $\mathbf{v}_i$ and half-lengths $r\sqrt{\gamma_i}$. The contours of a Gaussian are therefore ellipsoids whose axes are the eigenvectors of the covariance, stretched by the standard deviations $\sqrt{\gamma_i}$ along them. For the running example they are ellipses at 45° with half-lengths in the ratio $\sqrt{3} : 1$ (Figure 36.14, left). Because $|\Sigma| = \prod_i \gamma_i$, the density also factorizes in $\mathbf{u}$ into d one-dimensional Gaussians: the coordinates along the eigenvectors are independent with variances γ_i. When $\Sigma = \sigma^2 \mathbf{I}$, the contours are spheres and the original coordinates are already independent.

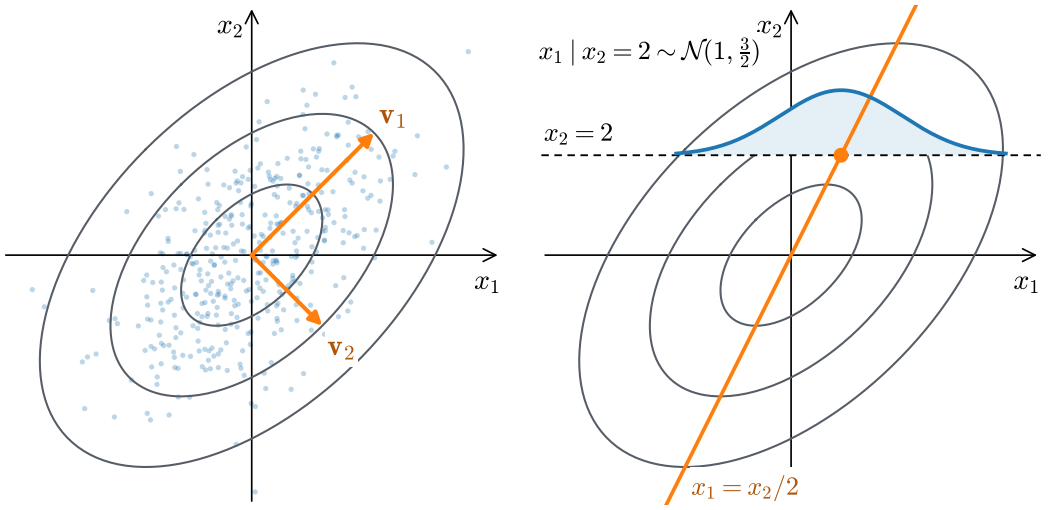


Figure 36.14: The running Gaussian with covariance Σ with rows $(2, 1)$ and $(1, 2)$. (a) 400 samples and three contours of the density, on which the quadratic form in the exponent equals 1, 4 and 9. The axes of the ellipses lie along the eigenvectors of Σ, with half-lengths proportional to $\sqrt{\gamma_1} = \sqrt{3}$ and $\sqrt{\gamma_2} = 1$; the arrows $\mathbf{v}_1$ and $\mathbf{v}_2$ reach the middle contour. (b) Observing $x_2 = 2$ slices the density along the dashed line; the slice is the conditional density $\mathcal{N}(1, \frac{3}{2})$ of x_1. The conditional means of all slices lie on the line $x_1 = x_2/2$, and the conditional variance is $3/2$ for every slice.

In high dimension the picture of a bell around the mean misleads: a standard Gaussian vector has norm close to $\sqrt{d}$, and two independent draws are nearly orthogonal. Section 36.9 proves both.

36.3.3.2 Linear Maps and Sampling

In practice we build Gaussian vectors from other Gaussian vectors: we mix coordinates, or project onto a few directions. These are linear maps, and a linear map keeps a Gaussian

Gaussian. We prove this first for invertible maps, which is all that sampling needs. Projections follow in the next subsection, once we know that a block of a Gaussian vector is Gaussian.

Proposition (invertible linear maps). Let $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$ in $\mathbb{R}^d$, let $\mathbf{A}$ be an invertible $d \times d$ matrix and $\mathbf{b} \in \mathbb{R}^d$. Then

$$\mathbf{A}\mathbf{x} + \mathbf{b} \sim \mathcal{N}\big(\mathbf{A}\mu + \mathbf{b},\ \mathbf{A}\Sigma\mathbf{A}^{\top}\big).$$

Proof. The mean and covariance hold for any distribution with these moments, by linearity: $\mathbb{E}[(\mathbf{A}\mathbf{x} - \mathbf{A}\mu)(\mathbf{A}\mathbf{x} - \mathbf{A}\mu)^{\top}] = \mathbf{A}\,\mathbb{E}[(\mathbf{x}-\mu)(\mathbf{x}-\mu)^{\top}]\,\mathbf{A}^{\top} = \mathbf{A}\Sigma\mathbf{A}^{\top}$. It remains to show that the distribution is Gaussian. The change-of-variables formula of Section 36.2 gives the density of $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$ as $p_{\mathbf{y}}(\mathbf{y}) = p_{\mathbf{x}}(\mathbf{A}^{-1}(\mathbf{y} - \mathbf{b}))/|\det \mathbf{A}|$. With $\mathbf{c} = \mathbf{y} - \mathbf{A}\mu - \mathbf{b}$, the exponent becomes $-\tfrac{1}{2}\mathbf{c}^{\top}\mathbf{A}^{-\top}\Sigma^{-1}\mathbf{A}^{-1}\mathbf{c} = -\tfrac{1}{2}\mathbf{c}^{\top}(\mathbf{A}\Sigma\mathbf{A}^{\top})^{-1}\mathbf{c}$, and $|\det \mathbf{A}|\,|\Sigma|^{1/2} = |\mathbf{A}\Sigma\mathbf{A}^{\top}|^{1/2}$, so $p_{\mathbf{y}}$ is the Gaussian density with the stated parameters. ■

Some condition on $\mathbf{A}$ is needed. With $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right)$ the two outputs are always equal, and $\mathbf{A}\Sigma\mathbf{A}^{\top} = \left(\begin{smallmatrix} 6 & 6 \\ 6 & 6 \end{smallmatrix}\right)$ is singular. The image is a Gaussian spread along the line $y_1 = y_2$, and it has no density in the plane.

Sampling by a matrix factor. How do we draw from $\mathcal{N}(\mu, \Sigma)$ when the generator produces only independent standard normals? Stack d of them into $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$, whose density is the product of d copies of ϕ. If a square matrix $\mathbf{L}$ satisfies $\mathbf{L}\mathbf{L}^{\top} = \Sigma$, it is invertible, and by the proposition $\mathbf{x} = \mu + \mathbf{L}\mathbf{z}$ is Gaussian with mean μ and covariance $\mathbf{L}\,\mathbf{I}\,\mathbf{L}^{\top} = \Sigma$.

A matrix factor turns independent noise into correlated noise. Because $\mathbf{x}$ is a differentiable function of μ and $\mathbf{L}$ for fixed $\mathbf{z}$, this is also the *reparameterization* through which variational autoencoders backpropagate (Section 34.7.3). The cheapest factor is the *Cholesky factor*: every symmetric positive definite matrix factors as $\Sigma = \mathbf{L}\mathbf{L}^{\top}$ with $\mathbf{L}$ lower triangular with a positive diagonal. For the running example, matching entries of $\mathbf{L}\mathbf{L}^{\top}$ to Σ one at a time gives $L_{11} = \sqrt{2}$, then $L_{21} = \Sigma_{21}/L_{11} = 1/\sqrt{2}$, then $L_{22} = \sqrt{\Sigma_{22} - L_{21}^2} = \sqrt{3/2}$, and indeed $L_{21}^2 + L_{22}^2 = \tfrac{1}{2} + \tfrac{3}{2} = 2$.

Any other factor works as well, for instance the eigen-factor $\mathbf{V}\,\mathrm{diag}(\sqrt{\gamma_1}, \dots, \sqrt{\gamma_d})$; two factors of the same Σ differ by an orthogonal matrix, which exercise 8 shows. PyTorch's `MultivariateNormal` samples with the Cholesky factor, and NumPy's `multivariate_normal` uses a factor from the singular value decomposition by default.

The cell samples by the recipe itself and not by a library call, so that the mechanism is visible, and prints the covariance of the samples and its eigenvalues.

```
Sigma = np.array([[2.0, 1.0], [1.0, 2.0]])
L = np.linalg.cholesky(Sigma)                  # Sigma = L L^T
x = rng.standard_normal((400000, 2)) @ L.T     # each row is x = L z
C = np.cov(x.T)
print('L =', L.round(3).tolist())
print('sample covariance =', C.round(2).tolist())
print('its eigenvalues   =', np.linalg.eigvalsh(C).round(2))
```

```
L = [[1.414, 0.0], [0.707, 1.225]]
sample covariance = [[1.99, 0.99], [0.99, 1.99]]
its eigenvalues   = [1.   2.99]
```

The factor is the one computed by hand, the sample covariance agrees with Σ to about 0.01, and its eigenvalues are close to 1 and 3.

36.3.3.3 Conditioning

Often we observe some coordinates of a Gaussian vector and want to predict the others. In the running example, if we observe $x_2 = 2$, where is x_1 likely to be? Split the vector into blocks $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$, and split its parameters to match:

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix}, \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}.$$

The idea is to remove from $\mathbf{x}_1$ the part that $\mathbf{x}_2$ predicts linearly. What is left over carries no information about $\mathbf{x}_2$.

Proposition (residual and marginal). Let $\mathbf{B} = \Sigma_{12}\Sigma_{22}^{-1}$. The residual $\mathbf{r} = \mathbf{x}_1 - \mathbf{B}\mathbf{x}_2$ is independent of $\mathbf{x}_2$, and $\mathbf{x}_2 \sim \mathcal{N}(\mu_2, \Sigma_{22})$.

Proof. The block Σ_{22} is positive definite, as a diagonal block of a positive definite matrix, so it is invertible. The residual is uncorrelated with $\mathbf{x}_2$: $\operatorname{Cov}(\mathbf{r}, \mathbf{x}_2) = \Sigma_{12} - \mathbf{B}\Sigma_{22} = \mathbf{0}$. The map $\mathbf{x} \mapsto (\mathbf{r}, \mathbf{x}_2)$ is the square matrix $\left(\begin{smallmatrix} \mathbf{I} & -\mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{smallmatrix}\right)$, which has determinant 1, so $(\mathbf{r}, \mathbf{x}_2)$ is Gaussian by the proposition on invertible linear maps. Its covariance is block diagonal, because the off-diagonal blocks are $\operatorname{Cov}(\mathbf{r}, \mathbf{x}_2) = \mathbf{0}$. Then the inverse covariance is block diagonal too, the quadratic form in the exponent splits into a term in $\mathbf{r}$ plus a term in $\mathbf{x}_2$, and the determinant is the product of the two blocks' determinants. The density is therefore a Gaussian density in $\mathbf{r}$ times a Gaussian density in $\mathbf{x}_2$: the two are independent, and $\mathbf{x}_2$ has mean μ_2 and covariance Σ_{22}. ■

The one idea of the proof is the choice of $\mathbf{B}$, which makes the residual uncorrelated with $\mathbf{x}_2$. The second claim says that a block of a Gaussian vector is Gaussian (for the first block, reorder the coordinates, which is an invertible map). It settles the linear maps we postponed.

Corollary (wide matrices). If $\mathbf{A}$ is an $m \times d$ matrix of full row rank $m \le d$ and $\mathbf{b} \in \mathbb{R}^m$, then $\mathbf{A}\mathbf{x} + \mathbf{b} \sim \mathcal{N}(\mathbf{A}\mu + \mathbf{b}, \mathbf{A}\Sigma\mathbf{A}^\top)$. To see this, complete $\mathbf{A}$ by $d - m$ further rows to an invertible square matrix. Its image of $\mathbf{x}$ is Gaussian, and $\mathbf{A}\mathbf{x} + \mathbf{b}$ is a block of it.

Now condition on $\mathbf{x}_2$. Since $\mathbf{x}_1 = \mathbf{r} + \mathbf{B}\mathbf{x}_2$ and $\mathbf{r}$ is independent of $\mathbf{x}_2$, given $\mathbf{x}_2$ the block $\mathbf{x}_1$ is the Gaussian $\mathbf{r}$ shifted by $\mathbf{B}\mathbf{x}_2$. The residual has mean $\mu_1 - \mathbf{B}\mu_2$ and covariance $\Sigma_{11} - \mathbf{B}\Sigma_{21} - \Sigma_{12}\mathbf{B}^\top + \mathbf{B}\Sigma_{22}\mathbf{B}^\top = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$, so $\mathbf{x}_1 \mid \mathbf{x}_2$ is Gaussian with

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(\mathbf{x}_2 - \mu_2), \qquad \Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}. \tag{36.47}$$

The conditional mean is a linear function of the observed block, and the conditional covariance does not depend on the observed value. For the running example, $\mathbf{B} = \frac{1}{2}$, so given $x_2 = 2$ the conditional mean is $\frac{1}{2} \cdot 2 = 1$ and the conditional variance is $2 - 1 \cdot \frac{1}{2} \cdot 1 = \frac{3}{2}$: $x_1 \mid x_2 = 2 \sim \mathcal{N}(1, \frac{3}{2})$, the slice drawn in Figure 36.14 (right).

The matrix $\mathbf{B}$ is the regression coefficient of $\mathbf{x}_1$ on $\mathbf{x}_2$: the multiple of $\mathbf{x}_2$ whose removal leaves no correlation. The conditional mean is linear regression read off the covariance. The conditional covariance, called the *Schur complement* of Σ_{22}, is never larger than Σ_{11}, because the subtracted term $\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ is positive semidefinite: observing x_2 shrank the

variance of x_1 from 2 to $\frac{3}{2}$. With $\mathbf{x}_2$ the observed values of a function and $\mathbf{x}_1$ its values at new inputs, this computation is Gaussian process regression (Chapter 40).

A reader may object that uncorrelated variables need not be independent. That is right, and the proof used more than zero correlation: it used that $(\mathbf{r}, \mathbf{x}_2)$ is *jointly* Gaussian. Without that, the conclusion fails. If $Z \sim \mathcal{N}(0,1)$ and S is an independent fair sign, then SZ is also $\mathcal{N}(0,1)$, and $\mathrm{Cov}(SZ, Z) = \mathbb{E}[S]\,\mathbb{E}[Z^2] = 0$, yet $|SZ| = |Z|$, so the two are far from independent.

Conditioning can also be done on samples, with no formula at all: keep only the samples whose x_2 is within 0.05 of 2 and look at their x_1. The cell does this with the Cholesky samples, after forming the residual $r = x_1 - \frac{1}{2}x_2$.

```
r = x[:, 0] - 0.5 * x[:, 1]                    # residual x1 - B x2, B = 1/2
print(f'Cov(r, x2) = {np.cov(r, x[:, 1])[0, 1]:.3f},  Var(r) = {r.var(ddof=1):.3f}')
near = x[np.abs(x[:, 1] - 2) < 0.05, 0]        # x1 of samples with x2 near 2
print(f'{near.size} samples with x2 near 2: mean of x1 {near.mean():.2f}, '
      f'variance {near.var(ddof=1):.2f}')
```

```
Cov(r, x2) = -0.003,  Var(r) = 1.499
4091 samples with x2 near 2: mean of x1 0.97, variance 1.52
```

The residual's covariance with x_2 is close to 0 and its variance close to $\frac{3}{2}$. About four thousand samples fall in the slice, and their x_1 has mean near 1 and variance near $\frac{3}{2}$, as $\mathcal{N}(1, \frac{3}{2})$ requires.

Two questions are left open. Nearly all of the distributions of this section share one algebraic form, which Section 36.4 uses to treat their losses and gradients together. And the probability $715/8192$ assumed that the coin is fair; how well 13 flips determine the bias of a coin is the question of Section 36.5.

36.3.4 Summary

The table collects the distributions of this section, in the order of the text. For the categorical and multinomial rows, a draw is coded as a one-hot vector, as the cross-entropy loss codes it.

Distribution	pmf or pdf	Mean	Variance	Model assumption
Bernoulli(p)	$p^x(1-p)^{1-x}$	p	$p(1-p)$	one yes/no outcome
Binomial(n,p)	$\binom{n}{k}p^k(1-p)^{n-k}$	np	$np(1-p)$	n fixed, independent trials, one p
Poisson(λ)	$\lambda^k e^{-\lambda}/k!$	λ	λ	many rare independent chances; mean equals variance
Categorical$(\mathbf{p})$	p_k	$\mathbf{p}$	$\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$	one of K outcomes

Distribution	pmf or pdf	Mean	Variance	Model assumption
Multinomial$(n, \mathbf{p})$	Equation 36.39	$n\mathbf{p}$	$n(\operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top)$	n fixed, independent draws
Unif$\{1, \dots, n\}$	$1/n$	$(n+1)/2$	$(n^2-1)/12$	equally likely values
Exp(λ)	$\lambda e^{-\lambda t}$, $t \geq 0$	$1/\lambda$	$1/\lambda^2$	constant event rate (no ageing)
Gamma(α, β)	$\beta^\alpha t^{\alpha-1} e^{-\beta t}/\Gamma(\alpha)$	α/β	α/β^2	time of the α-th event (integer α)
$U(a, b)$	$1/(b-a)$ on $[a, b]$	$(a+b)/2$	$(b-a)^2/12$	equal-length intervals equally likely
$\mathcal{N}(\mu, \sigma^2)$	$\frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/(2\sigma^2)}$	μ	σ^2	sum of many small independent effects, finite variance
Laplace(μ, b)	$\frac{1}{2b} e^{-\lvert x-\mu\rvert/b}$	μ	$2b^2$	symmetric, exponential tails
$\mathcal{N}(\mu, \Sigma)$	Equation 36.46	μ	Σ	jointly Gaussian coordinates

Other distributions live elsewhere: the Cauchy in Section 36.1, the log-normal in Section 36.2, the Beta and the Dirichlet in Section 36.4, and the negative binomial, the geometric and Student's t in exercises 3, 4 and 7.

A small set of distributions describes the labels and noise of most models, and the negative log-probability of each is a familiar loss. All of them grow out of one coin flip by a sum, a limit, a waiting time, a random sign, or a vector of coordinates. The construction of each distribution gives its mean and variance in a few lines, and it gives a sampler as well. The central limit theorem explains why the Gaussian appears whenever many small effects with finite variance add up. A Gaussian vector stays Gaussian under linear maps and conditioning, and conditioning on one block is linear regression on it.

36.3.5 Exercises

1. **Error counts on a test set.** A classifier is scored on $n = 10{,}000$ independent test examples, each of which it gets wrong with the same probability q, its true error rate. Its number of errors X then has the distribution Binomial(n, q).
   1. Let $q = 0.1$. Use the Gaussian approximation of the binomial (the de Moivre–Laplace theorem) to find the interval around nq that contains X with probability about 95%. Two classifiers with this error rate are scored on two independent test sets of this size. Find the standard deviation of the difference of their observed accuracies.

   2. Let $q = 5 \cdot 10^{-4}$. Compute $P(X = 0)$ and $P(X \geq 10)$ exactly and under the Poisson approximation $\text{Poisson}(nq)$. Show that the logarithm of $P(X = 0)$ under the binomial differs from its Poisson value by about $-nq^2/2$.
   3. The classifier makes no errors on the n test examples. Under the Poisson approximation, find the largest error rate q for which this outcome still has probability at least 0.05, and express it as a multiple of $1/n$. Use (b) to explain why the Poisson approximation is accurate for every error rate this small.
2. **Conditioning on a noisy observation.** The conditional formulas Equation 36.47 turn a joint Gaussian into a prediction of one block from another.
   1. Take the bivariate Gaussian with $\mu = \mathbf{0}$ and $\Sigma = \left(\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right)$, where $|\rho| < 1$. Identify $\mathbf{B}$ and the residual r, and check $\text{Cov}(r, x_2) = 0$ directly. Show that $x_1 \mid x_2 \sim \mathcal{N}(\rho x_2, 1 - \rho^2)$. Why does observing x_2 never increase the uncertainty about x_1?
   2. A signal $s \sim \mathcal{N}(0, \tau^2)$ is observed with additive noise, $y = s + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is independent of s. Write (s, y) as a linear map of (s, ϵ) to obtain its joint Gaussian distribution, and compute the conditional distribution of s given y. Show that the conditional precision $1/\text{Var}(s \mid y)$ is the sum of the prior precision $1/\tau^2$ and the noise precision $1/\sigma^2$. How does $\mathbb{E}[s \mid y]$ behave as $\sigma^2 \to 0$ and as $\sigma^2 \to \infty$? This computation is Gaussian process regression (Chapter 40) with a single observation.
3. **Gamma–Poisson mixture.** Let the Poisson rate itself be random, $\lambda \sim \text{Gamma}(\alpha, \beta)$ with the density Equation 36.41, and let $X \mid \lambda \sim \text{Poisson}(\lambda)$.
   1. Use the law of total expectation $\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}[X \mid \lambda]\big]$ and the law of total variance $\text{Var}(X) = \mathbb{E}\big[\text{Var}(X \mid \lambda)\big] + \text{Var}\big(\mathbb{E}[X \mid \lambda]\big)$ to compute $\mathbb{E}[X]$ and $\text{Var}(X)$. Write the variance in terms of the mean $m = \alpha/\beta$ and the shape α. In which limit of α and β does the Poisson reappear?
   2. Show that X has the *negative binomial* pmf
$$P(X = k) = \frac{\Gamma(k + \alpha)}{k!\,\Gamma(\alpha)} \Big(\frac{\beta}{1 + \beta}\Big)^{\alpha} \Big(\frac{1}{1 + \beta}\Big)^{k}, \qquad k = 0, 1, 2, \dots,$$
   by integrating out λ with the help of the Gamma normalizer.
   3. A count dataset has sample mean 2 and sample variance 6. Choose α and β so that the model matches both. Which named distribution results?
4. **Converse of memorylessness.** ▯ The exponential distribution is memoryless. This problem proves the converse.
   1. Let $X > 0$ be a continuous random variable whose survival function $G(t) = P(X > t)$ satisfies $G(s+t) = G(s)\,G(t)$ for all $s, t \geq 0$. Show that $G(t) = e^{-\lambda t}$ for some $\lambda > 0$. Start from $G(1)$, extend the formula to rational t, and then use that G is nonincreasing. Why can $G(1)$ be neither 0 nor 1?
   2. Let X take values in $\{1, 2, \dots\}$ and satisfy $P(X > s + t \mid X > s) = P(X > t)$ for all integers $s, t \geq 0$. Show that X is geometric, $P(X > t) = (1 - p)^t$ for some $p \in (0, 1]$.
5. code **Convergence rate of the CLT.** ▯▯ Let F_n be the c.d.f. of the standardized sum $(S_n - n\mu)/(\sigma\sqrt{n})$ of n independent summands with mean μ and standard deviation σ. Measure the error of the Gaussian approximation by the Kolmogorov distance $D_n = \sup_x |F_n(x) - \Phi(x)|$, where Φ and ϕ denote the standard Gaussian c.d.f. and density. Section 36.3.2.3 judges the approximation by eye; this problem measures its rate.
   1. Compute D_n for $n \in \{1, 2, 4, \dots, 64\}$ and three summand distributions, $U(0, 1)$, $\text{Exp}(1)$ and $\text{Bernoulli}(1/2)$. For uniform summands, compute the density of

S_n by repeated numerical convolution of the $U(0,1)$ density on a fine grid (`np.convolve`, rescaled by the grid spacing) and integrate it to F_n. For exponential summands, S_n is the time of the n-th event of a Poisson process with rate 1, so $P(S_n \le s) = P(N \ge n)$ for $N \sim \text{Poisson}(s)$. For the two continuous summands, take the supremum over a fine grid of $x \in [-4, 4]$. For Bernoulli summands, use the exact binomial pmf and compare Φ with F_n on both sides of every jump. Plot D_n against n on log-log axes and fit the slopes.

2. For summands that are not confined to a lattice and have skewness $\gamma = \mathbb{E}[(X-\mu)^3]/\sigma^3$, the Edgeworth expansion (Hall 1992) refines the central limit theorem to

$$F_n(x) = \Phi(x) - \frac{\gamma}{6\sqrt{n}}\,(x^2-1)\,\phi(x) + o(n^{-1/2}). \tag{36.48}$$

Predict the limit of $\sqrt{n}\,D_n$ for exponential summands and compare it with your measurements. What does the expansion predict for uniform summands, and which rate did you measure for them?

3. Bernoulli$(1/2)$ summands also have $\gamma = 0$. Explain the slope you measured for them by the size of the largest jump of F_n, and predict the limit of $\sqrt{n}\,D_n$. Then compare F_n at the lattice points k with the continuity-corrected value $\Phi\big((k + \tfrac{1}{2} - n\mu)/(\sigma\sqrt{n})\big)$. Which rate does the corrected error have?

4. The correctness indicators of a classifier with accuracy 0.99 are Bernoulli(0.99) variables. Compute their skewness, and assume that the skewness term of (b) also governs the continuity-corrected error. Predict the number of test examples needed before the corrected error falls below 0.01, and check the prediction by computing the corrected error for $n \in \{10^2, 10^3, 10^4\}$. Build the pmf of S_n from the number of errors $n - S_n \sim \text{Binomial}(n, 0.01)$: the successive-ratio recursion started from $P(S_n = 0) = 0.01^n$ underflows to zero in floating point once n exceeds about 160, whereas the recursion for the errors starts from 0.99^n. What does the result imply for Gaussian error bars on the test accuracy of a highly accurate model?

6. [extended] **Burstiness of word counts.** ⊠⊠ Church and Gale (1995) found that word counts in text vary across documents more than a Poisson model allows, and that Poisson mixtures such as the negative binomial fit them better. Reproduce the finding at small scale on *The Time Machine* (Section 8.2). Download the text with `d2l.download(d2l.DATA_URL + 'timemachine.txt')`, keep only letters as `d2l.TimeMachine._preprocess` does, split it into words, and cut the word sequence into consecutive chunks of $L = 500$ words, which play the role of documents.

   1. For each of the words "the", "and", "of", "time", "traveller", "weena" and "morlocks", count its occurrences in every chunk, and compute the ratio of the sample variance to the sample mean of the counts. Which ratio does a Poisson model predict? Which words deviate most from it?
   2. Fit a Poisson and a negative binomial model (problem 3b) to the counts of each word by maximum likelihood, and compare the maximized log-likelihoods. Parameterize the negative binomial by its mean m and its shape α. The maximum-likelihood m is the sample mean, so only α requires a one-dimensional search, for example on a logarithmic grid.
   3. For each word, compare the empirical fraction of chunks that do not contain the word, and the empirical *adaptation* $P(X \ge 2 \mid X \ge 1)$, with the predictions of both fitted models. What does the adaptation measure, and why does a Poisson model underestimate it for a name such as "traveller"?

4. Repeat (a) for chunk lengths $L \in \{100, 250, 500, 2000\}$. Assume that the rate of a word in a chunk is L times a per-word rate drawn from a Gamma distribution with a fixed shape. Use problem 3a to predict how the variance-to-mean ratio grows with L, and compare the prediction with your measurements.

7. **The Student-t as a Gamma scale mixture.** ▣ A Gaussian whose precision (inverse variance) is itself random has heavier tails than any Gaussian. The result is Student's t distribution, which Section 36.5 uses for the mean of a small sample. Let $\tau \sim \text{Gamma}(\nu/2, \nu/2)$ with the density Equation 36.41 (shape and rate both $\nu/2$, for some $\nu > 0$), and let $X \mid \tau \sim \mathcal{N}(\mu, 1/\tau)$.
   1. Integrate τ out to show that X has the density
   $$p(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\pi\nu}\,\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{(x-\mu)^2}{\nu}\right)^{-(\nu+1)/2}, \tag{36.49}$$
   Student's t distribution with ν degrees of freedom. *Hint:* $\int_0^\infty \tau^{a-1} e^{-c\tau}\, d\tau = \Gamma(a)/c^a$ for $a, c > 0$.
   2. Show that $p(x)$ decays like $|x|^{-(\nu+1)}$, so that $\mathbb{E}|X-\mu|^k$ is finite only for $k < \nu$. Which distribution of Section 36.1 is the case $\nu = 1$?
   3. Show that $\text{Var}(X) = \nu/(\nu-2)$ for $\nu > 2$ by the law of total variance, $\text{Var}(X) = \mathbb{E}\big[\text{Var}(X \mid \tau)\big] + \text{Var}\big(\mathbb{E}[X \mid \tau]\big)$.
   4. code Draw 10^5 samples for $\nu = 3$ and $\mu = 0$ by the mixture recipe (first τ, then X), and compare the fraction with $|X| > 3$ with the Gaussian's 0.0027. Then draw 10^4 sample means of 20 draws each and compare their standard deviation with $\sqrt{3/20}$.

8. **Two factors of one covariance.** ▣ Let $\Sigma = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$.
   1. Compute the Cholesky factor $\mathbf{L}$ by hand, and the eigen-factor $\mathbf{M} = \mathbf{V}\,\text{diag}(\sqrt{\gamma_1}, \sqrt{\gamma_2})$ from the eigenvectors and eigenvalues found in the text. Verify $\mathbf{L}\mathbf{L}^\top = \mathbf{M}\mathbf{M}^\top = \Sigma$.
   2. Show that any two invertible factors with $\mathbf{M}_1\mathbf{M}_1^\top = \mathbf{M}_2\mathbf{M}_2^\top$ satisfy $\mathbf{M}_2 = \mathbf{M}_1\mathbf{Q}$ with $\mathbf{Q}$ orthogonal (*hint:* set $\mathbf{Q} = \mathbf{M}_1^{-1}\mathbf{M}_2$). What is the distribution of $\mathbf{Q}\mathbf{z}$ for $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$? Conclude that $\mu + \mathbf{M}_1\mathbf{z}$ and $\mu + \mathbf{M}_2\mathbf{z}$ have the same distribution.
   3. *Whitening.* Show that $\mathbf{z} = \mathbf{L}^{-1}(\mathbf{x} - \mu)$ has covariance $\mathbf{I}$ when $\mathbf{x} \sim \mathcal{N}(\mu, \Sigma)$. Explain why computing $\mathbf{z}$ for one $\mathbf{x}$ takes about d^2 operations once $\mathbf{L}$ is known, against about $d^3/3$ for computing $\mathbf{L}$.
   4. code Sample $10{,}000$ points by each factor, print both empirical covariances, whiten the Cholesky samples with a triangular solve, and print the covariance of the result.

9. **Superposition, thinning, and the order of arrivals.** ▣ Requests reach a server from two independent sources, which are Poisson processes with rates λ_1 and λ_2.
   1. Show that the merged stream is a Poisson process with rate $\lambda_1 + \lambda_2$. Use the slicing picture, or show by the binomial theorem that the sum of independent $\text{Poisson}(\lambda_1)$ and $\text{Poisson}(\lambda_2)$ counts is $\text{Poisson}(\lambda_1 + \lambda_2)$.
   2. Each request of a rate-λ process is logged independently with probability q. Show that the logged requests form a Poisson process with rate $q\lambda$: condition on the number of requests and sum a binomial over a Poisson. With $\lambda = 4$ per minute and $q = \frac{1}{4}$, what is the probability that a minute logs nothing?
   3. Given that exactly n events fell in $[0, t]$, show that their times have the distribution of n independent $U(0, t)$ draws, sorted. Use the slicing picture: every slice is equally likely to hold an event.

4. code Simulate a rate-4 process for $1{,}000$ minutes by exponential gaps, thin it with $q = \frac{1}{4}$, and check that the thinned counts per minute have mean and variance close to 1. Collect the positions within the minute of the events in minutes with exactly three events, and check that their histogram is flat.

10. **Covariance of the multinomial.** ▯ Write the multinomial count vector as $\mathbf{x} = \sum_{i=1}^{n} \mathbf{e}_{Y_i}$, a sum of n independent one-hot vectors with $P(Y_i = k) = p_k$.
    1. Show that $\mathbb{E}[\mathbf{e}_Y] = \mathbf{p}$ and $\mathrm{Cov}(\mathbf{e}_Y) = \mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$, and hence that $\mathbb{E}[\mathbf{x}] = n\mathbf{p}$ and $\mathrm{Cov}(\mathbf{x}) = n(\mathrm{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top)$. Read off $\mathrm{Var}(x_k) = np_k(1-p_k)$ and $\mathrm{Cov}(x_j, x_k) = -np_jp_k$ for $j \neq k$. Why is every covariance negative?
    2. Show that $\mathrm{Cov}(\mathbf{x})\mathbf{1} = \mathbf{0}$, and explain this by the constraint $\sum_k x_k = n$.
    3. For 12 rolls of a fair die, compute $\mathrm{Var}(x_k)$, $\mathrm{Cov}(x_j, x_k)$, and the correlation between two counts.
    4. Show that x_k alone is $\mathrm{Binomial}(n, p_k)$ by grouping the other $K-1$ classes into one.

36.4 The Exponential Family

Linear regression, logistic regression and softmax regression each come with a loss of their own. Yet when we differentiate these losses, all three give the same training rule: the input times the difference between the prediction and the observation. Agreement of this kind usually has a single cause. The catalogue of Section 36.3, in which each mean and variance needed its own trick, raises the same suspicion.

Take the coin that runs through this chapter, which came up heads 9 times in 13 flips. The obvious estimate of its bias is $9/13 \approx 0.69$. A logistic-regression model with only a bias term, fitted to the same flips, reaches this number by a detour: its last layer computes the log-odds $\log(9/4) \approx 0.81$, and a sigmoid turns that into $9/13$. Why does a classifier work with the log-odds rather than with the probability itself?

Both observations have one explanation. Nearly every distribution of the catalogue can be written in a common form, in which the parameters act through the exponential of a linear function of a few statistics of the data. Distributions of this form make up the *exponential family*, and for the coin the parameter in the exponent is the log-odds. Four facts follow from the form alone. The mean and the variance are the first two derivatives of one function. Fitting by maximum likelihood is a convex problem whose gradient is prediction minus observation. The softmax is what the form gives for class labels. And a prior of the same form is updated by adding counts.

The second fact is the one a deep learning reader came for, and the first is what proves it (Section 36.4.3.2 and Section 36.4.3.3). Together they say which loss to use for a new kind of label, such as a count, and when the fit has no solution at all, as for a coin that came up heads in every flip. Two parts can be skipped on a first reading: maximum entropy (Section 36.4.2), which explains where the form comes from, and the derivations of the softmax from noisy choices and from physics.

36.4.1 One Form for Many Distributions

We begin with the smallest distribution of the catalogue, a single coin flip, and rewrite it until the log-odds appears. The rewriting shows what the general form has to look like.

36.4.1.1 The Bernoulli in Exponential Form

A coin with bias p shows heads ($x = 1$) with probability p and tails ($x = 0$) with probability $1-p$, which the previous section wrote compactly as $p^x(1-p)^{1-x}$. To see how the outcome x and the parameter p interact, we take the logarithm and collect the terms that contain x:

$$\log P(X = x) = x \log p + (1 - x) \log(1 - p) = x \log \frac{p}{1 - p} + \log(1 - p).$$

The outcome enters only through x itself, multiplied by the log-odds $\log \frac{p}{1-p}$, and the remaining term depends on p alone. We name the log-odds $\eta = \log \frac{p}{1-p}$ and express the second term through it. Solving for p gives $p = e^\eta/(1 + e^\eta)$, so $1 - p = 1/(1 + e^\eta)$ and $\log(1 - p) = -\log(1 + e^\eta)$. Exponentiating,

$$P(X = x) = \exp\big(x\eta - \log(1 + e^\eta)\big), \qquad x \in \{0, 1\}.$$

The coin's probability is the exponential of a linear function of x minus a term that does not involve x. That term is the normalizer: the two unnormalized probabilities are $e^0 = 1$ and e^η, and they sum to $1 + e^\eta$.

Three pieces of this formula will recur. The *statistic* $T(x) = x$ is what the outcome contributes. The *natural parameter* η multiplies it; for the coin it is the log-odds, also called the *logit*. The *log-partition function* $A(\eta) = \log(1 + e^\eta)$ is the logarithm of the normalizer, a function known as the *softplus*. Reading the substitution backward,

$$p = \frac{e^\eta}{1 + e^\eta} = \text{sigmoid}(\eta), \qquad \text{sigmoid}(\eta) = \frac{1}{1 + e^{-\eta}}.$$

The sigmoid of logistic regression is thus nothing more than the map from the coin's natural parameter back to its mean. Nobody had to choose it.

For the chapter's coin, the estimate $9/13$ corresponds to the natural parameter

$$\hat{\eta} = \log \frac{9/13}{4/13} = \log \frac{9}{4} \approx 0.811,$$

and the sigmoid maps it back, $\text{sigmoid}(\log \frac{9}{4}) = \frac{9/4}{1+9/4} = \frac{9}{13}$. We show below that $\hat{\eta}$ is the maximum-likelihood value of the log-odds, and we ask what the curvature of the loss at $\hat{\eta}$ tells us.

36.4.1.2 Definition and Members

The coin's rewriting has three parts: a statistic of the data, a parameter that multiplies it, and a normalizer. Allowing several statistics at once, and a fixed weight on each outcome, gives a form that fits most of the catalogue.

A family of distributions is an *exponential family* if its probability mass or density function can be written as

$$p(\mathbf{x} \mid \eta) = h(\mathbf{x})\, \exp\big(\eta^\top T(\mathbf{x}) - A(\eta)\big). \tag{36.50}$$

The data and the parameters meet only in the inner product $\eta^\top T(\mathbf{x})$. The *base measure* $h(\mathbf{x}) \geq 0$ is a fixed weight on the outcomes that does not depend on the parameters; the set where $h > 0$ is the support. The *natural parameters* $\eta \in \mathbb{R}^d$ are the adjustable parameters, and the *sufficient statistics* $T(\mathbf{x}) \in \mathbb{R}^d$ are the features of the data that they multiply. The *log-partition function* is the logarithm of the normalizer,

$$A(\eta) = \log \int h(\mathbf{x})\ \exp(\eta^\top T(\mathbf{x}))\, d\mathbf{x}, \tag{36.51}$$

with a sum in place of the integral for discrete $\mathbf{x}$. In other words, A is whatever makes Equation 36.50 sum or integrate to one, and the natural parameters range over the set where $A(\eta)$ is finite. The coin is the case $h(x) = 1$, $T(x) = x$ and $A(\eta) = \log(1 + e^\eta)$.

The name *sufficient* says what T does for estimation. For n independent observations the likelihood is

$$\prod_{i=1}^{n} p(\mathbf{x}_i \mid \eta) = \Big(\prod_{i=1}^{n} h(\mathbf{x}_i)\Big) \exp\Big(\eta^\top \sum_{i=1}^{n} T(\mathbf{x}_i) - nA(\eta)\Big),$$

so the parameters touch the data only through the sum $\sum_i T(\mathbf{x}_i)$. Two data sets with the same sum have likelihoods that differ by a factor that does not depend on η, and they lead to the same estimates. For the coin the sum is the number of heads. Any reordering of the 13 flips has the same probability under every bias, so once we know that there were 9 heads, the order tells us nothing more about the bias.

Matching the pieces identifies the other members. The Poisson probability of k events at rate λ is $\frac{\lambda^k e^{-\lambda}}{k!} = \frac{1}{k!}\exp(k \log \lambda - \lambda)$, so $h(k) = 1/k!$, $T(k) = k$, $\eta = \log \lambda$ and $A(\eta) = e^\eta = \lambda$. The Gaussian with unknown mean and variance needs two statistics. Expanding the square in its exponent and collecting the terms in x and in x^2 gives

$$\frac{1}{\sqrt{2\pi}} \exp\Big(\underbrace{\tfrac{\mu}{\sigma^2}}_{\eta_1}\, x + \underbrace{\big(-\tfrac{1}{2\sigma^2}\big)}_{\eta_2}\, x^2 - \underbrace{\big(\tfrac{\mu^2}{2\sigma^2} + \log\sigma\big)}_{A(\eta)}\Big),$$

so $T(x) = (x, x^2)^\top$ and $h(x) = 1/\sqrt{2\pi}$. The second natural parameter must be negative: with $\eta_2 < 0$ the factor $e^{\eta_2 x^2}$ produces the Gaussian's decay, and with $\eta_2 \geq 0$ the integral that defines A diverges. Table 36.1 lists these and other members.

Table 36.1: Members of the exponential family. In each row the probability or density is $h(\mathbf{x}) \exp(\eta^\top T(\mathbf{x}) - A(\eta))$; the binomial needs a fixed number of trials n and the Laplace a fixed location μ.

distribution	$h(\mathbf{x})$	$T(\mathbf{x})$	η	$A(\eta)$
Bernoulli(p)	1	x	$\log \frac{p}{1-p}$	$\log(1 + e^\eta)$
Binomial(n, p)	$\binom{n}{x}$	x	$\log \frac{p}{1-p}$	$n \log(1 + e^\eta)$
Poisson(λ)	$1/k!$	k	$\log \lambda$	e^η
Categorical, one-hot $\mathbf{y}$	1	$\mathbf{y}$	logits $\mathbf{z}$	$\log \sum_k e^{z_k}$

distribution	$h(\mathbf{x})$	$T(\mathbf{x})$	η	$A(\eta)$
Exponential(λ)	1 for $x \geq 0$	x	$-\lambda$	$-\log(-\eta)$
$\mathcal{N}(\mu, \sigma^2)$	$1/\sqrt{2\pi}$	(x, x^2)	$(\frac{\mu}{\sigma^2}, -\frac{1}{2\sigma^2})$	$-\frac{\eta_1^2}{4\eta_2} - \frac{1}{2}\log(-2\eta_2)$
Laplace(μ, b)	1	$\lvert x - \mu \rvert$	$-1/b$	$\log(-2/\eta)$

The categorical row writes the class probabilities through the logits. They form the softmax, whose origins Section 36.4.4 takes up. Two more members need matrices: the multivariate Gaussian, with the statistics $\mathbf{x}$ and $\mathbf{x}\mathbf{x}^\top$ and the natural parameters $\Sigma^{-1}\mu$ and $-\frac{1}{2}\Sigma^{-1}$, and the multinomial with a fixed number of draws. The split between h, η and T is not unique. The one redundancy that matters later is in the categorical row: its K one-hot statistics sum to one, so one logit is superfluous (Section 36.4.3.2).

36.4.1.3 Distributions Outside the Family

Not every distribution has this form, and the two main exceptions fail for different reasons.

The uniform distribution $U(0, \theta)$ has the density $1/\theta$ on $[0, \theta]$ and zero elsewhere, so its support moves with the parameter. Every member of Equation 36.50 has the same support for all parameters, the set where $h > 0$, because the exponential factor is never zero. A moving support changes how estimation works. Take three observations, 0.1, 0.2 and 0.9. The likelihood is θ^{-3} when $\theta \geq 0.9$ and 0 when $\theta < 0.9$, because a smaller θ cannot produce the observation 0.9, so it is largest at $\hat{\theta} = \max_i x_i = 0.9$.

Inside the family, maximum likelihood matches the mean of the sufficient statistic to its sample average, as we show below. The same recipe here sets the mean $\theta/2$ equal to $\bar{x} = 0.4$ and gives $\theta = 0.8$, a value under which the observation 0.9 is impossible. The uniform's likelihood depends on the data through their maximum, not through a sum, and the tools of this section do not apply to it. Exercise 8 compares the two estimates as the sample grows.

Heavy-tailed distributions fall outside for another reason. The *Student-t* distribution is a Gaussian whose precision $1/\sigma^2$ is itself drawn at random; averaging over the precision turns the e^{-x^2} tails into polynomial ones (Exercise 7 derives its density). With unit scale and ν degrees of freedom, its likelihood for n observations is a product of the factors $\left(1 + (x_i - \mu)^2/\nu\right)^{-(\nu+1)/2}$, which no fixed number of sums of the data can summarize. We do not prove this; it is a theorem of Pitman, Koopman and Darmois that, among regular families with a fixed support, only exponential families have a sufficient statistic of fixed dimension (Darmois 1935; Koopman 1936; Pitman 1936).

Convexity of the negative log-likelihood, which we prove below for the family, can fail outside it. For the Student-t, the negative log-likelihood of the location μ has a separate local minimum near each cluster when the data form two well-separated clusters.

36.4.2 Maximum Entropy

So far the exponential form came from rewriting distributions we already had. A different question explains where it comes from. Suppose that all we know about a random quantity is the average of a few of its statistics: which distribution should we assume? The answer that assumes nothing beyond those averages is a member of the exponential family.

36.4.2.1 Entropy and a Loaded Die

To make "assumes nothing beyond the averages" precise, we need a measure of how spread out a distribution is. For a distribution $p = (p_1, \dots, p_K)$ on a finite set, the *entropy* is

$$H[p] = -\sum_{k=1}^{K} p_k \log p_k,$$

measured in nats, with the convention $0 \log 0 = 0$. Each term weighs the surprise $-\log p_k$ of outcome k by its probability, so the entropy is the average surprise. A fair die has six outcomes of surprise $\log 6$ each and the entropy $\log 6 \approx 1.79$; a die that always shows a six is never surprising and has entropy 0. Among distributions on K outcomes the uniform one has the largest entropy, $\log K$, which we prove below.

E. T. Jaynes posed the question in this form (Jaynes 1957). Suppose a die is known to average 4.5 rather than 3.5, and nothing else is known about it. Many distributions on $\{1, \dots, 6\}$ have this mean. Half threes and half sixes is one, but it asserts that 1, 2, 4 and 5 never occur, which nothing we know supports; its entropy is $\log 2 \approx 0.69$.

Jaynes' principle picks, among all distributions with the required mean, the one of largest entropy, the least committal choice. We state the answer before proving it. The maximizer is an *exponential tilt* of the uniform distribution,

$$p_k = \frac{e^{\eta k}}{\sum_{j=1}^{6} e^{\eta j}}, \qquad k = 1, \dots, 6,$$

with η chosen so that the mean is 4.5. The die of threes and sixes gave up $\log 6 - \log 2 \approx 1.10$ of the fair die's 1.79 nats to reach this mean. Before running the cell, guess how much the tilt has to give up. The mean of the tilt increases with η, so bisection finds the right η.

```
k = np.arange(1, 7)
def tilt(eta):                                  # p_k proportional to exp(eta k)
    w = np.exp(eta * k)
    return w / w.sum()

lo, hi = -5.0, 5.0
for _ in range(50):                             # bisection: the mean increases with eta
    eta = (lo + hi) / 2
    lo, hi = (eta, hi) if (tilt(eta) * k).sum() < 4.5 else (lo, eta)
p = tilt(eta)
print('eta =', round(eta, 3), ' p =', p.round(3))
print('H =', round(-(p * np.log(p)).sum(), 3), ' log 6 =', round(np.log(6), 3))
```

```
eta = 0.371  p = [0.054 0.079 0.114 0.165 0.24  0.347]
H = 1.614  log 6 = 1.792
```

The tilt is geometric: each face is $e^{0.371} \approx 1.45$ times as likely as the face below it. Its entropy is 1.61 nats, only 0.18 less than that of the fair die, and $\eta = 0$ gives the fair die back. Figure 36.15 shows the tilts for three required means.

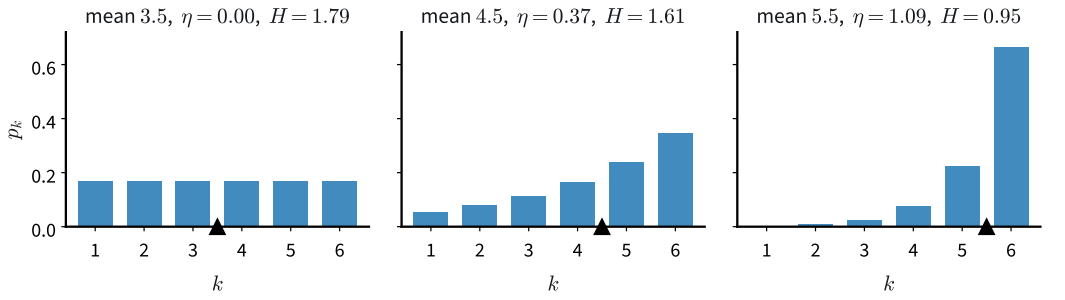


Figure 36.15: Maximum-entropy distributions of a die whose average roll is fixed at 3.5, 4.5 and 5.5 (black triangles). Each is an exponential tilt $p_k \propto e^{\eta k}$ of the uniform distribution; the tilt η grows with the required mean and the entropy H falls.

The tilt has the form Equation 36.50 with $h \equiv 1$ on the six faces, the statistic $T(k) = k$ and the natural parameter η. Fixing the average of a statistic and maximizing entropy has produced the exponential of a multiple of that statistic. This is no accident of the die.

36.4.2.2 The Maximum-Entropy Proposition

The general statement needs one extension of entropy, because some sets of outcomes come with natural weights. For a distribution q and a base measure h, the *entropy relative to* h is

$$H_h[q] = -\sum_{\mathbf{x}} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{h(\mathbf{x})}$$

for discrete outcomes, with an integral for densities. With $h \equiv 1$ it is the entropy defined above.

The proof compares q with a member p of the family. For distributions p and q on the same set, with $p(\mathbf{x}) > 0$ wherever $q(\mathbf{x}) > 0$, the *Kullback–Leibler divergence* of q from p is the sum $D_{\mathrm{KL}}(q\|p) = \sum q \log(q/p)$ over the outcomes with $q > 0$, or the corresponding integral for densities (Cover and Thomas 1999). Section 37.1 develops entropy and this divergence in full. Here we need a single property, and a tangent line proves it. The logarithm is concave, so $\log u \le u - 1$ for all $u > 0$, with equality only at $u = 1$.

Lemma (Gibbs' inequality). *The Kullback–Leibler divergence is never negative,*

$$D_{\mathrm{KL}}(q\|p) = \sum_{\mathbf{x}:\, q(\mathbf{x})>0} q(\mathbf{x}) \log \frac{q(\mathbf{x})}{p(\mathbf{x})} \ge 0, \tag{36.52}$$

with equality only if $q = p$.

Proof. Apply the tangent-line bound with $u = p(\mathbf{x})/q(\mathbf{x})$ and sum over the outcomes with $q(\mathbf{x}) > 0$:

$$-D_{\mathrm{KL}}(q\|p) = \sum_{q>0} q \log \frac{p}{q} \le \sum_{q>0} q\Big(\frac{p}{q} - 1\Big) = \sum_{q>0} p - 1 \le 0.$$

Equality in the first step needs $p = q$ wherever $q > 0$; then $\sum_{q>0} p = 1$, so p puts no mass outside the support of q, and $p = q$. ■

Proposition (maximum entropy). *Let $p(\mathbf{x}) = h(\mathbf{x}) \exp(\eta^\top T(\mathbf{x}) - A(\eta))$ be a member of an exponential family, and let q be any distribution (with a density, in the continuous case) that is zero wherever h is zero, for which $\mathbb{E}_q[T(\mathbf{x})]$ is finite, and which has the same mean statistic, $\mathbb{E}_q[T(\mathbf{x})] = \mathbb{E}_p[T(\mathbf{x})]$. Then $H_h[q] \le H_h[p]$, with equality only if $q = p$.*

Proof. We split the logarithm in $H_h[q]$ at p, $\log \frac{q}{h} = \log \frac{q}{p} + \log \frac{p}{h}$. The first part contributes $-D_{\mathrm{KL}}(q\|p) \le 0$ by the lemma, which applies because $p > 0$ wherever $h > 0$. The second part is linear in the statistic, $\log \frac{p}{h} = \eta^\top T - A(\eta)$, so only the mean of T under q enters:

$$-\sum_{\mathbf{x}} q \log \frac{p}{h} = -\eta^\top \mathbb{E}_q[T] + A(\eta) = -\eta^\top \mathbb{E}_p[T] + A(\eta) = H_h[p].$$

The middle step uses the constraint, and the last step is the same computation with p in place of q. Adding the two parts gives $H_h[q] = H_h[p] - D_{\mathrm{KL}}(q\|p) \le H_h[p]$, with equality only if $q = p$. ■

The step that matters is the second: because the exponent is linear in T, the cross term depends on q only through $\mathbb{E}_q[T]$, which the constraint fixes. In words, *if some member of the family meets the constraints, it is the unique distribution of largest entropy under them*. No Lagrange multipliers are needed, and the proposition shows that the stationary point a Lagrangian calculation finds is the global maximum.

For the die, the tilt with $\eta \approx 0.371$ is a member with $h \equiv 1$ and $T(k) = k$ whose mean is 4.5, so it is the unique maximizer. Three familiar distributions are corollaries.

- *No constraint* ($T \equiv 0$): every q qualifies, and the maximizer is $p \propto h$, provided h has a finite sum. On a finite set with $h \equiv 1$ this is the uniform distribution, with entropy $\log K$; the fair die is the case $K = 6$.
- *Mean and variance on* $\mathbb{R}$: with $T(x) = (x, x^2)$ and a constant h, the maximizer is the Gaussian. For a constant h, H_h differs from the entropy of a density by a constant, so the Gaussian has the largest entropy among all densities with a given mean and variance.
- *Mean on the counts* $\{0, 1, 2, \dots\}$, $T(k) = k$: with $h(k) = 1/k!$ the maximizer of H_h with mean m is the Poisson distribution with $\lambda = m$; with $h \equiv 1$ it is the geometric distribution $p_k = (1 - r)r^k$ with $r = m/(1 + m)$. The base measure decides which, as Exercise 2 develops.

The hypothesis that some member meets the constraints cannot be dropped. Fix only the mean m on all of $\mathbb{R}$, with $h \equiv 1$. The candidate $e^{\eta x}$ is not integrable for any η, so no member exists. Nor does any maximizer: the Gaussian $\mathcal{N}(m, s^2)$ has mean m and the entropy $\frac{1}{2}\log(2\pi e s^2)$, which grows without bound as s grows.

36.4.3 The Log-Partition Function

In the catalogue each mean needed its own trick: $X^2 = X$ for the coin, a sum of independent flips for the binomial, an index shift in the series for the Poisson, and integration by parts for the Gaussian. For a member of the exponential family all of these are one computation, the derivative of A. The second derivative gives the variance, and the same function controls maximum likelihood.

36.4.3.1 The Moment Property

The function A is the logarithm of a sum of exponentials of $\eta^\top T(\mathbf{x})$. Differentiating such a logarithm brings down $T(\mathbf{x})$, weighted by the probability of $\mathbf{x}$, which is the mean of T.

Proposition (mean of the sufficient statistics). *For an exponential-family distribution Equation 36.50,*

$$\nabla_\eta A(\eta) = \mathbb{E}[T(\mathbf{x})]. \tag{36.53}$$

Proof. Differentiate Equation 36.51 under the integral sign. The interchange of derivative and integral is valid in the interior of the set of natural parameters with $A(\eta) < \infty$ (Folland 1999); this set is convex (a consequence of Hölder's inequality), and we take η in its interior from now on. Writing $Z(\eta) = e^{A(\eta)} = \int h(\mathbf{x}) e^{\eta^\top T(\mathbf{x})} d\mathbf{x}$, the derivative of the logarithm is $\nabla A = (\nabla Z)/Z$, and

$$\nabla_\eta Z = \int h(\mathbf{x})\, T(\mathbf{x})\, e^{\eta^\top T(\mathbf{x})}\, d\mathbf{x}.$$

Dividing by $Z = e^A$ moves the factor e^{-A} inside the integral, where it completes the density Equation 36.50:

$$\nabla_\eta A = \frac{1}{Z}\int h(\mathbf{x})\, T(\mathbf{x})\, e^{\eta^\top T(\mathbf{x})}\, d\mathbf{x} = \int T(\mathbf{x})\, \underbrace{h(\mathbf{x})\, e^{\eta^\top T(\mathbf{x}) - A(\eta)}}_{p(\mathbf{x}\mid\eta)}\, d\mathbf{x} = \mathbb{E}[T(\mathbf{x})]. \quad \blacksquare$$

Differentiating $\nabla A = (\nabla Z)/Z$ once more, with the same interchange, gives the covariance of the statistics:

$$\nabla^2 A = \frac{\nabla^2 Z}{Z} - \frac{(\nabla Z)(\nabla Z)^\top}{Z^2} = \mathbb{E}[T\,T^\top] - \mathbb{E}[T]\,\mathbb{E}[T]^\top = \mathrm{Cov}(T).$$

The slope of A is the mean of the statistic and its curvature is the covariance. A covariance matrix is positive semidefinite, since $\mathbf{v}^\top \mathrm{Cov}(T)\mathbf{v} = \mathrm{Var}(\mathbf{v}^\top T) \geq 0$ for every $\mathbf{v}$ (Section 33.3.2.2), so A is convex.

For the coin, $A(\eta) = \log(1 + e^\eta)$ gives $A'(\eta) = \mathrm{sigmoid}(\eta) = p$, the Bernoulli mean, and $A''(\eta) = \mathrm{sigmoid}(\eta)(1 - \mathrm{sigmoid}(\eta)) = p(1-p)$, its variance. At $\hat{\eta} = \log\frac{9}{4}$ these are $9/13 \approx 0.692$ and $\frac{9}{13}\cdot\frac{4}{13} = \frac{36}{169} \approx 0.213$. For the Poisson, $A(\eta) = e^\eta$ is its own derivative, so at $\lambda = 4$, that is $\eta = \log 4$, both derivatives equal 4. Since the moments are derivatives, each framework's automatic differentiation computes them for any A that we can write as code. We try it on these two cases.

```
def derivatives(A, eta):
    """First and second derivative of the scalar function A at eta."""
    eta = torch.tensor(eta, dtype=torch.float64, requires_grad=True)
    (dA,) = torch.autograd.grad(A(eta), eta, create_graph=True)
    (d2A,) = torch.autograd.grad(dA, eta)
    return dA.item(), d2A.item()
```

```
cases = [('Bernoulli', torch.nn.functional.softplus, math.log(9 / 4), 9 / 13, 36 / 169),
         ('Poisson', torch.exp, math.log(4), 4, 4)]
for name, A, eta, mean, var in cases:
    dA, d2A = derivatives(A, eta)
    print(f"{name:9s} A' = {dA:.4f} (mean {mean:.4f})  A'' = {d2A:.4f} (variance {var:.4f})")
```

```
Bernoulli A' = 0.6923 (mean 0.6923)  A'' = 0.2130 (variance 0.2130)
Poisson   A' = 4.0000 (mean 4.0000)  A'' = 4.0000 (variance 4.0000)
```

Both derivatives agree with the mean and the variance to the four digits printed. The Poisson's equal mean and variance, which the catalogue derived from the series, is the statement $A = A' = A''$ for $A(\eta) = e^{\eta}$.

36.4.3.2 Maximum Likelihood as Moment Matching

The moment property turns maximum likelihood into a condition on means: the fitted model must reproduce the sample average of the statistic. For a sample $\mathbf{x}_1, \dots, \mathbf{x}_n$, taking the negative logarithm of the product likelihood from the definition gives

$$-\sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \eta) = n\,A(\eta) - \eta^{\top} \sum_{i=1}^{n} T(\mathbf{x}_i) - \sum_{i=1}^{n} \log h(\mathbf{x}_i). \tag{36.54}$$

The data enter only through the sum of the statistics, and the negative log-likelihood is convex in η because A is convex and the second term is linear. Setting its gradient to zero and using the moment property gives the condition

$$\nabla A(\hat{\eta}) = \bar{T}, \qquad \bar{T} = \frac{1}{n} \sum_{i=1}^{n} T(\mathbf{x}_i).$$

Maximum likelihood sets the model's mean of the statistic equal to its sample mean, which is called *moment matching*. Because the negative log-likelihood is convex, any solution of this equation is a global minimizer.

For the coin with 9 heads in 13 flips, the negative log-likelihood is $13A(\eta) - 9\eta = 13\log(1 + e^{\eta}) - 9\eta$. Its derivative $13\,\mathrm{sigmoid}(\eta) - 9$ vanishes where $\mathrm{sigmoid}(\hat{\eta}) = 9/13$, at $\hat{\eta} = \log \frac{9}{4} \approx 0.811$. The minimum value is

$$13 \log \frac{13}{4} - 9 \log \frac{9}{4} \approx 8.02 \text{ nats},$$

which is 13 times the entropy of a coin with bias $9/13$, $-\frac{9}{13}\log\frac{9}{13} - \frac{4}{13}\log\frac{4}{13} \approx 0.617$. At the optimum the loss per flip equals the entropy of the fitted coin.

This holds for every family with $h \equiv 1$: the loss per observation at $\hat{\eta}$ is $A(\hat{\eta}) - \hat{\eta}^{\top}\bar{T}$, and the entropy of the fitted member is $A(\hat{\eta}) - \hat{\eta}^{\top}\nabla A(\hat{\eta})$, the same number because $\nabla A(\hat{\eta}) = \bar{T}$.

The second derivative of the loss at the minimum is $13A''(\hat{\eta}) = 13 \cdot \frac{36}{169} = \frac{36}{13} \approx 2.77$. A large curvature means that the loss rises steeply away from $\hat{\eta}$, so the data pin the log-odds

down sharply. Thirteen flips do not: the loss is only about half a nat above its minimum at a distance of $1/\sqrt{2.77} \approx 0.60$ from $\hat{\eta} \approx 0.81$. Section 36.6 calls this curvature the Fisher information and turns it into an error bar. Figure 36.16 shows both the tangent of A at $\hat{\eta}$ and the loss with its minimum.

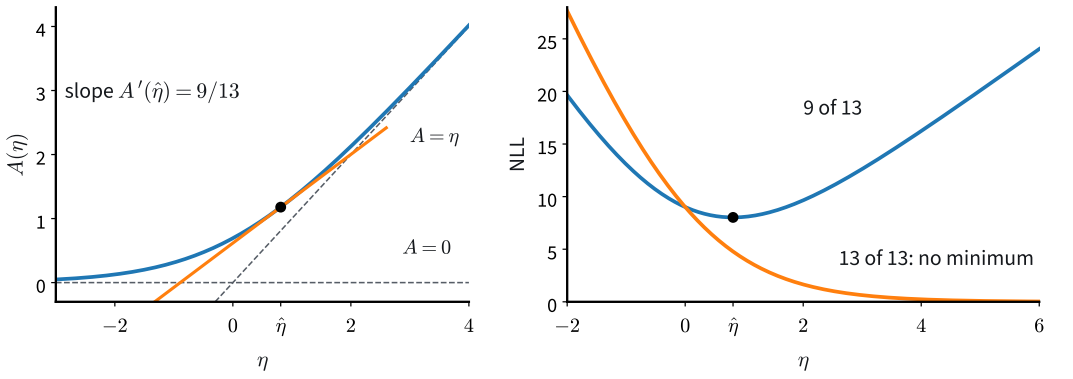


Figure 36.16: Left: the Bernoulli log-partition function $A(\eta) = \log(1 + e^{\eta})$ with its asymptotes and its tangent at $\hat{\eta} = \log(9/4)$, whose slope is the mean $9/13$. Right: the negative log-likelihood $13A(\eta) - x\eta$ for $x = 9$ heads in 13 flips, with its minimum at $\hat{\eta}$, and for $x = 13$ heads, which decreases for every η and has no minimizer.

The equation $\nabla A(\hat{\eta}) = \bar{T}$ has a solution only if $\bar{T}$ is a mean that some member of the family attains. For the coin the attainable means are $\text{sigmoid}(\eta)$ for real η, the open interval $(0, 1)$. If all 13 flips are heads, $\bar{T} = 1$ is not attainable. The negative log-likelihood is then $13A(\eta) - 13\eta = 13\log(1 + e^{-\eta})$, which decreases for every η: it is 9.01 at $\eta = 0$, 0.087 at $\eta = 5$ and 0.0006 at $\eta = 10$. It approaches 0 without reaching it, so there is no minimizer; the fitted bias runs to 1 and $\hat{\eta}$ to infinity.

Logistic regression fails in the same way when a hyperplane separates the two classes of the training data: the loss keeps falling as the weights grow, and the weights grow without bound. A prior on the parameters, which we meet at the end of this section, or weight decay restores a finite solution; Exercise 9 computes it.

When a solution exists, it is unique if the negative log-likelihood is strictly convex, which holds when $\text{Cov}(T)$ is positive definite, that is, when no nontrivial linear combination of the statistics is constant on the support. Such a representation is called *minimal*. The categorical distribution with all K logits as natural parameters is not minimal: the one-hot statistics sum to one, and adding the same constant to every logit leaves every probability unchanged. Its negative log-likelihood is flat along the direction $\mathbf{1}$, and its minimizers form a line. Implementations keep all K logits and use this invariance to subtract the largest logit before exponentiating (Section 3.4).

36.4.3.3 Generalized Linear Models

Regression and classification models predict a distribution of a label y for each input $\mathbf{x}$. A *generalized linear model* takes that distribution from an exponential family Equation 36.50 and lets the input act on it through a linear function $\mathbf{W}^{\top}\mathbf{x}$ (a bias is a constant feature). We make the simplest choice and let this function be the natural parameter itself, $\eta = \mathbf{W}^{\top}\mathbf{x}$, which is called the *canonical link*. Write $\mu(\mathbf{x}) = \nabla A(\eta) = \mathbb{E}[T(y) \mid \mathbf{x}]$ for the model's mean

of the sufficient statistic. The negative log-likelihood of one example is $A(\eta) - \eta^\top T(y) - \log h(y)$, so by the moment property Equation 36.53 its gradient with respect to η is $\mu(\mathbf{x}) - T(y)$, and the chain rule through $\eta = \mathbf{W}^\top \mathbf{x}$ gives

$$\nabla_{\mathbf{W}}(-\log p(y \mid \mathbf{x})) = \mathbf{x}\,(\mu(\mathbf{x}) - T(y))^\top. \tag{36.55}$$

The gradient is the input times the error of the predicted mean: prediction minus observation, whatever the family. This is the common rule that the opening asked about, and with the moment property in hand it took two lines. The loss is convex in $\mathbf{W}$, because it is the convex function A composed with a linear function of $\mathbf{W}$, plus a linear term. The simple gradient belongs to the canonical link: the probit model of Section 36.4.4 passes the same linear function through a Gaussian CDF, and its gradient carries an extra weight that depends on the prediction. Four families give four familiar models:

label y	family	$T(y)$	mean $\mu(\mathbf{x})$	model
real number	Gaussian, variance 1	y	η	linear regression
yes or no	Bernoulli	$y \in \{0, 1\}$	sigmoid(η)	logistic regression
one of K classes	categorical	one-hot $\mathbf{y}$	softmax(η)	softmax regression
count	Poisson	y	e^η	Poisson regression

A count model shows the rule at work. Suppose a Poisson model predicts the mean $e^\eta = 3$ for some input and the observed count is $y = 5$. The gradient of the loss with respect to η is $3 - 5 = -2$, so a gradient step raises η and with it the predicted mean.

The Gaussian row is the family with the variance fixed at 1, so that $h(y) \propto e^{-y^2/2}$ and the loss is the squared loss of Section 2.1 plus a constant. A neural network keeps the last step, a linear map into the natural parameters followed by the family's negative log-likelihood, and replaces the fixed input $\mathbf{x}$ by learned features. The loss is then no longer convex in the weights, but its gradient with respect to the natural parameters is still prediction minus observation (Bishop 2006).

36.4.4 Origins of the Softmax

The classifiers of the main text turn their K logits $\mathbf{z}$ into class probabilities with the *softmax*,

$$\text{softmax}(\mathbf{z})_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}, \qquad k = 1, \dots, K. \tag{36.56}$$

The categorical rows of both tables above lead to the same map. Each probability grows with its own logit, and the probabilities sum to one. Other maps share these properties: normalizing $\log(1 + e^{z_k})$ instead of e^{z_k} also gives positive numbers that sum to one and keep the order of the logits. Under the softmax, however, $z_i - z_j = \log(p_i/p_j)$ is a log-ratio of probabilities. Three arguments from different fields lead to this choice: a model of noisy

choices, the Boltzmann distribution of physics, and the exponential family. The third is the shortest, and the only one that the rest of the section uses.

36.4.4.1 Noisy Choices and Gumbel Noise

The first argument starts from how labels come about. Suppose that a decision maker, such as a shopper or whoever labels our data, gives each option k a score z_k but does not always pick the best one. A simple model of this behavior perturbs every score by independent noise ϵ_k and picks the largest perturbed score,

$$y = \operatorname*{argmax}_k \, (z_k + \epsilon_k).$$

The probability of class k is the chance that option k wins this noisy contest, so the noise distribution decides its form. Gaussian noise is the obvious first choice. With two options and $\epsilon_k \sim \mathcal{N}(0, s^2)$, option 1 wins when $\epsilon_2 - \epsilon_1 < z_1 - z_2$. The difference $\epsilon_2 - \epsilon_1$ is Gaussian with variance $2s^2$, so

$$P(y = 1) = \Phi\Big(\frac{z_1 - z_2}{\sqrt{2}\, s}\Big),$$

where Φ is the standard Gaussian CDF (we write ϕ for its density). This is the *probit* model: the probability depends only on the score difference, measured in units of the noise.

Unfortunately the probit model has no closed form for more than two options. The chance that option k wins is an integral, over its own noise, of a product of $K - 1$ Gaussian CDFs, and this integral and its gradient must be computed numerically for every example at every training step.

Gumbel noise removes the integral. The standard *Gumbel* distribution has the CDF $G(x) = \exp(-e^{-x})$. It is the limiting distribution of the suitably shifted and scaled maximum of many independent draws from a distribution with an exponentially decaying tail, such as the Gaussian or the exponential (Gumbel 1954), and McFadden (1974) used it as the noise in models of economic choices.

Proposition (Gumbel noise gives the softmax). *If $\epsilon_1, \dots, \epsilon_K$ are independent standard Gumbel variables, the noisy argmax follows the softmax probabilities,*

$$P\Big(\operatorname*{argmax}_j \, (z_j + \epsilon_j) = k\Big) = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}.$$

Proof. Write $S = \sum_j e^{z_j}$. The perturbed score $z_j + \epsilon_j$ has the CDF $G(t - z_j) = \exp(-e^{z_j} e^{-t})$. If option k scores $t = z_k + \epsilon_k$, it wins with probability $\prod_{j \neq k} \exp(-e^{z_j} e^{-t}) = \exp(-(S - e^{z_k}) e^{-t})$, by independence. We multiply by the density $e^{z_k} e^{-t} \exp(-e^{z_k} e^{-t})$ of t, integrate, and substitute $v = e^{-t}$:

$$P(y = k) = \int_{-\infty}^{\infty} e^{z_k} e^{-t} \exp(-S e^{-t}) \, dt = e^{z_k} \int_0^{\infty} e^{-Sv} \, dv = \frac{e^{z_k}}{S}. \quad \blacksquare$$

The proof works because a product of Gumbel CDFs is again a Gumbel CDF. Over all K options, $P(\max_j(z_j + \epsilon_j) \le t) = \exp(-Se^{-t}) = G(t - \log S)$, so the winning score is a standard Gumbel variable shifted by the log-sum-exp $\log \sum_j e^{z_j}$. The log-sum-exp is thus a smoothed maximum of the logits, and its gradient, the softmax, gives the chance that each option attains the maximum. Exercise 1 proves the proposition a second way, through races between exponential waiting times.

Read from right to left, the proposition is a sampler: to draw a class with the softmax probabilities, add independent Gumbel noise to the logits and take the argmax. This is the *Gumbel-max trick*.

Until the end of Section 36.4.4, T denotes a temperature, as in the main text, and not the statistic. Because $\operatorname{argmax}_k(z_k + T\epsilon_k) = \operatorname{argmax}_k(z_k/T + \epsilon_k)$ for $T > 0$, Gumbel noise of scale T samples from $\operatorname{softmax}(\mathbf{z}/T)$: the amount of noise in the choice acts as a temperature. The cell uses the trick as a sampler. It draws 10^5 choices among four logits at $T = 1$ and at $T = 2$ and counts how often each option wins.

```
z = np.array([2.0, 1.0, 0.1, -1.0])                    # four logits
softmax = lambda v: np.exp(v) / np.exp(v).sum()
rng = np.random.default_rng()
n = 100_000
for T in (1.0, 2.0):                                    # noise scale = temperature
    winners = np.argmax(z + T * rng.gumbel(size=(n, 4)), axis=1)
    freq = np.bincount(winners, minlength=4) / n
    print(f'T = {T:.0f}: argmax frequencies {freq.round(3)}')
    print(f'      softmax(z / T)        {softmax(z / T).round(3)}'
          f'  max diff {np.abs(freq - softmax(z / T)).max():.3f}')
```

```
T = 1: argmax frequencies [0.638 0.234 0.095 0.033]
      softmax(z / T)        [0.638 0.235 0.095 0.032]  max diff 0.001
T = 2: argmax frequencies [0.451 0.274 0.173 0.102]
      softmax(z / T)        [0.451 0.274 0.174 0.101]  max diff 0.002
```

The frequencies differ from $\operatorname{softmax}(\mathbf{z}/T)$ by a few thousandths at most, the size of the Monte Carlo error for 10^5 draws. Doubling the noise flattens the frequencies in the same way as halving the logits.

With two options, the difference of two independent standard Gumbel variables has the logistic distribution, whose CDF is $\operatorname{sigmoid}(x) = 1/(1 + e^{-x})$, so this *logit* model gives $P(y = 1) = \operatorname{sigmoid}(z_1 - z_2)$. The sigmoid is close to a rescaled Gaussian CDF, $|\operatorname{sigmoid}(1.7x) - \Phi(x)| < 0.01$ for all x, so fitted probit and logit models give nearly the same probabilities. They part company in the tails: on a confidently mislabeled example the logit model's loss grows linearly in the score difference and the probit model's quadratically, so such examples pull a probit fit harder (Exercise 4).

36.4.4.2 Temperature and the Boltzmann Distribution

The second argument comes from physics, where the same exponential appears for a different reason. A physical system in thermal equilibrium at temperature T, such as a molecule in a gas, occupies a state k of energy E_k with probability

$$p_k = \frac{\exp(-E_k/(k_{\mathrm{B}}T))}{\sum_j \exp(-E_j/(k_{\mathrm{B}}T))},$$

where k_{B} is Boltzmann's constant. Low-energy states are the most probable, and the temperature sets how strongly they are preferred. Boltzmann found this law for the molecules of a gas, and Gibbs (1902) made it the basis of statistical mechanics. If we measure temperature in units of energy, so that $k_{\mathrm{B}} = 1$, and set $E_k = -z_k$, the law becomes $\mathrm{softmax}(\mathbf{z}/T)$. A logit is a negative energy, and the most probable class is the state of lowest energy.

The maximum-entropy proposition explains this exponential without the physics. A physicist fixes the mean energy and assumes nothing else; here the same constraint fixes the average logit $\sum_k p_k z_k$. Take the K classes as the outcomes, with $h \equiv 1$, and the logit z_k of class k as the statistic. Among all distributions on the classes with a given average logit, strictly between the smallest and the largest logit, the one of largest entropy is then $p_k \propto e^{\eta z_k}$. For averages above the mean of the logits, $\eta > 0$, and this is $\mathrm{softmax}(\mathbf{z}/T)$ with $T = 1/\eta$. As T falls from ∞ to 0, the average logit rises from the mean of the logits to their maximum. For the logits $\mathbf{z} = (2, 1, 0.1, -1)$ of the simulation, the plain softmax ($T = 1$) has the average logit 1.49, between the mean 0.525 and the maximum 2.

36.4.4.3 The Categorical as an Exponential Family

The third argument is the one this section has already made. With the class written as a one-hot vector $\mathbf{y}$, the categorical distribution is $P(\mathbf{y}) = \exp(\mathbf{z}^\top \mathbf{y} - A(\mathbf{z}))$ with $A(\mathbf{z}) = \log \sum_k e^{z_k}$, an exponential family with the logits as natural parameters. By the moment property its mean is $\nabla A(\mathbf{z}) = \mathrm{softmax}(\mathbf{z}) = \mathbb{E}[\mathbf{y}]$. Its negative log-likelihood is convex in the logits, and by Equation 36.55 its gradient is $\mathrm{softmax}(\mathbf{z}) - \mathbf{y}$.

The three arguments agree on the function and explain different things about it: noisy choices give an exact sampler, the Boltzmann distribution gives the temperature, and the exponential family gives the convex loss and its gradient.

36.4.5 Conjugate Priors

Maximum likelihood turns 9 heads in 13 flips into one number, $9/13$. One number says nothing about how far to trust it: 9 heads in 13 flips and 900 in 1300 give the same estimate. A distribution over the bias would carry this information, and it should absorb the next flip by a small update rather than a new calculation.

From here on the bias is an unknown quantity with a distribution of its own, and we write it θ, as the following sections do. A distribution over θ before the data is a *prior*. Bayes' rule multiplies it by the likelihood and normalizes the product, $p(\theta \mid x) \propto p(\theta)\, P(x \mid \theta)$, which gives the *posterior*. A prior is *conjugate* to a likelihood if the posterior has the same functional form as the prior, so that observing data changes only its parameters.

36.4.5.1 The Beta Prior on a Coin

A prior on a coin's bias is a density on $[0, 1]$. As a function of θ, the likelihood of x heads in n flips is $\theta^x(1-\theta)^{n-x}$, so a prior with the same shape combines with it by adding exponents. Its normalizer uses the Gamma function $\Gamma(z) = \int_0^\infty t^{z-1}e^{-t}\,dt$ of Section 36.3,

which extends the factorial: $\Gamma(z+1) = z\,\Gamma(z)$ and $\Gamma(n+1) = n!$. The *Beta* distribution with parameters $a, b > 0$ has the density

$$\text{Beta}(\theta \mid a, b) = \frac{\theta^{a-1}(1-\theta)^{b-1}}{B(a,b)}, \qquad 0 \le \theta \le 1, \tag{36.57}$$

where the normalizer $B(a,b) = \int_0^1 \theta^{a-1}(1-\theta)^{b-1}d\theta$ equals $\Gamma(a)\Gamma(b)/\Gamma(a+b)$. The mean follows from the normalizer, because multiplying the density by θ raises a by one:

$$\mathbb{E}[\theta] = \frac{1}{B(a,b)}\int_0^1 \theta^{a}(1-\theta)^{b-1}\,d\theta = \frac{B(a+1,b)}{B(a,b)} = \frac{a}{a+b},$$

using $\Gamma(a+1) = a\,\Gamma(a)$ and $\Gamma(a+b+1) = (a+b)\Gamma(a+b)$ to simplify the ratio. The same step applied twice gives the variance. With $m = a/(a+b)$ the prior mean,

$$\text{Var}(\theta) = \frac{ab}{(a+b)^2(a+b+1)} = \frac{m(1-m)}{a+b+1}.$$

The larger $a+b$, the more concentrated the prior. The case $a = b = 1$ is the uniform density on $[0,1]$.

Now we observe x heads in n flips. Multiplying the prior by the likelihood adds the exponents,

$$\underbrace{\theta^{a-1}(1-\theta)^{b-1}}_{\text{prior}} \cdot \underbrace{\theta^{x}(1-\theta)^{n-x}}_{\text{likelihood}} = \theta^{(a+x)-1}(1-\theta)^{(b+n-x)-1},$$

which, up to normalization, is again a Beta density, with parameters shifted by the data:

$$\theta \mid x \;\sim\; \text{Beta}\big(a+x,\; b+n-x\big). \tag{36.58}$$

This update has a *pseudo-count* interpretation. The parameters a and b act as prior counts of heads and tails, and observing x heads and $n-x$ tails adds these to the counts. The posterior mean

$$\mathbb{E}[\theta \mid x] = \frac{a+x}{a+b+n}$$

is the frequency of heads after including the pseudo-counts. It lies between the prior mean $a/(a+b)$ and the maximum-likelihood estimate x/n, and approaches the latter as n grows.

How many flips the prior is worth depends on the summary. The posterior mode is $(a+x-1)/(a+b+n-2)$ when both posterior parameters exceed 1, so there the prior contributes $a-1$ heads and $b-1$ tails. The convention, once and for all: *a and b are the pseudo-counts of the posterior mean, $a-1$ and $b-1$ those of the posterior mode*. The uniform prior $\text{Beta}(1,1)$ adds one imagined flip of each kind to the mean and none to the mode.

For the chapter's coin with the uniform prior, 9 heads in 13 flips give the posterior $\mathrm{Beta}(10, 5)$. Its mean is $10/15 = 2/3$, which is *Laplace's rule of succession* $(x + 1)/(n + 2)$ (Laplace 1814). Its mode $9/13$ is the maximum-likelihood estimate, and its standard deviation is $\sqrt{(2/3)(1/3)/16} \approx 0.118$.

After 90 heads in 130 flips the posterior is $\mathrm{Beta}(91, 41)$, with mean 0.689 and standard deviation 0.040. Ten times as many flips shrink the standard deviation by a factor of $2.9 \approx \sqrt{133/16}$: the posterior width shrinks like $1/\sqrt{n}$, as the standard error of Section 36.5 will.

What is the probability that the next flip is heads? Given θ it is θ, and averaging over the posterior gives $P(\text{heads} \mid x) = \int_0^1 \theta\, p(\theta \mid x)\, d\theta = \mathbb{E}[\theta \mid x]$. For the coin this is $2/3$: Laplace's rule is the predicted probability of the next head.

Conjugacy spared us the integral in Bayes' rule. Without it we multiply the prior by the likelihood on a fine grid of θ and normalize the sum. Take a Gaussian prior on the natural parameter, $\eta \sim \mathcal{N}(0, 1)$, the kind of prior that Section 36.8 puts on the weights of a logistic regression. It says that the coin is probably close to fair, and it is not conjugate to the Bernoulli: by the change of variables its density in θ is $\phi(\eta)/(\theta(1-\theta))$, with ϕ the standard Gaussian density, and that is no Beta density. The cell computes the posterior under both priors on the same grid.

```
x, n = 9, 13                                  # 9 heads in 13 flips
theta = np.linspace(0, 1, 100_001)[1:-1]      # interior grid points
eta = np.log(theta) - np.log1p(-theta)        # natural parameter
priors = {'Beta(1, 1)': np.ones_like(theta),  # densities in theta
          'eta ~ N(0, 1)': np.exp(-eta**2 / 2) / (theta * (1 - theta))}
for name, prior in priors.items():
    post = prior * theta**x * (1 - theta)**(n - x)
    post /= post.sum()                        # grid weights sum to one
    mean = post @ theta
    sd = math.sqrt(post @ (theta - mean)**2)
    print(f'{name:13s}: mean {mean:.3f}, sd {sd:.3f}, '
          f'mode {theta[post.argmax()]:.3f}')
```

```
Beta(1, 1)   : mean 0.667, sd 0.118, mode 0.692
eta ~ N(0, 1): mean 0.644, sd 0.111, mode 0.665
```

Under the uniform prior the grid returns the moments of $\mathrm{Beta}(10, 5)$, mean 0.667 and standard deviation 0.118, and its mode $9/13 \approx 0.692$. The Gaussian prior on the log-odds pulls the posterior toward a fair coin: mean 0.644, mode 0.665, and a slightly smaller standard deviation, 0.111. The grid handles the non-conjugate prior as easily as the conjugate one, but only because θ is a single number; Section 36.8 takes up the general case. Figure 36.17 follows the update further.

36.4.5.2 Conjugacy for Every Exponential Family

The Beta update looks like a lucky match of exponents. It is an instance of a general fact: every exponential family has a conjugate prior, and the update always adds the sufficient statistics.

Proposition (conjugate prior). *Consider a likelihood Equation 36.50 and the prior on the natural parameters*

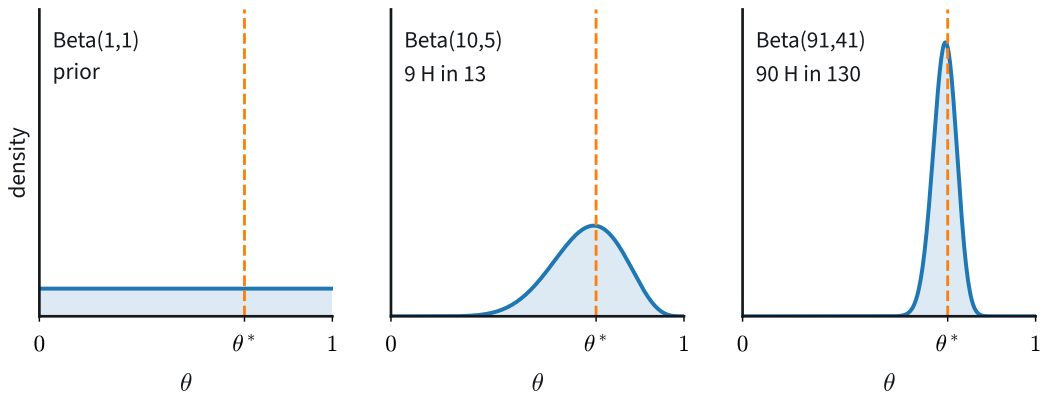


Figure 36.17: Bayesian updating. The flat Beta$(1,1)$ prior on the bias θ (left); the Beta$(10,5)$ posterior after 9 heads in 13 flips of a coin with true bias $\theta^* = 0.7$ (dashed), with mean $2/3$ and standard deviation 0.12 (middle); the Beta$(91,41)$ posterior after 130 flips, with standard deviation 0.04 (right). Each update adds the heads and tails to the pseudo-counts, and the width shrinks like $1/\sqrt{n}$.

$$p(\eta) \propto \exp\big(\nu^\top \eta - \kappa\, A(\eta)\big), \tag{36.59}$$

with $\nu \in \mathbb{R}^d$ and $\kappa \in \mathbb{R}$ chosen such that this density is normalizable. After n independent observations $\mathbf{x}_1, \dots, \mathbf{x}_n$, the posterior has the same form, with ν replaced by $\nu + \sum_i T(\mathbf{x}_i)$ and κ by $\kappa + n$.

Proof. The likelihood contributes $\exp(\eta^\top \sum_i T(\mathbf{x}_i) - nA(\eta))$ times a factor that does not depend on η, so

$$e^{\nu^\top \eta - \kappa A(\eta)} \cdot e^{\eta^\top \sum_i T(\mathbf{x}_i) - nA(\eta)} = e^{(\nu + \sum_i T(\mathbf{x}_i))^\top \eta - (\kappa + n) A(\eta)}.$$

Normalizing the product gives the posterior, so the posterior has the prior's form with the updated ν and κ. ■

The prior's own parameters ν and κ, called *hyperparameters*, act as *pseudo-data*: ν is a prior total of the sufficient statistic and κ a prior sample size (Bishop 2006). For the coin the prior lives on the logit $\eta = \log \frac{\theta}{1-\theta}$, and a change of variables from η to θ turns it into Beta$(\nu, \kappa - \nu)$ on θ; the update $\nu \to \nu + x$, $\kappa \to \kappa + n$ is the Beta update $a \to a + x$, $b \to b + n - x$. Exercise 7 carries out the change of variables, including its Jacobian.

36.4.5.3 Gamma–Poisson and Dirichlet–Multinomial

Two more pairs follow the same recipe: one for rates of events, one for the probabilities of words. As a function of the rate, n Poisson counts $x_1, \dots, x_n$ have a likelihood proportional to $\lambda^{\sum_i x_i} e^{-n\lambda}$. The matching prior is the *Gamma* distribution with shape $a > 0$ and rate $b > 0$, the density Gamma$(\lambda \mid a, b) = \frac{b^a}{\Gamma(a)} \lambda^{a-1} e^{-b\lambda}$ on $\lambda > 0$ with mean a/b, and the posterior is Gamma$\big(a + \sum_i x_i,\ b + n\big)$: the prior acts as a imagined events in b imagined periods. Exercise 7 derives this pair from the proposition.

The *Dirichlet* distribution is a density on probability vectors $\mathbf{p}$ with K entries, $p_k \geq 0$ and $\sum_k p_k = 1$. With parameters $a_k > 0$ it is

$$\mathrm{Dir}(\mathbf{p} \mid \mathbf{a}) = \frac{\Gamma(\sum_k a_k)}{\prod_k \Gamma(a_k)} \prod_{k=1}^{K} p_k^{a_k - 1},$$

with mean $\mathbb{E}[p_k] = a_k / \sum_j a_j$; for $K = 2$ it is the Beta distribution. Counts $x_1, \dots, x_K$ of the K categories have the likelihood $\prod_k p_k^{x_k}$, so the posterior is $\mathrm{Dir}(\mathbf{a} + \mathbf{x})$: one pseudo-count per category.

Suppose a three-word vocabulary shows the counts $(3, 0, 1)$. The maximum-likelihood estimate $(0.75, 0, 0.25)$ gives the unseen second word probability zero, and any text containing it becomes impossible. Under the prior $\mathrm{Dir}(1, 1, 1)$ the posterior is $\mathrm{Dir}(4, 1, 2)$, with mean $(4/7, 1/7, 2/7) \approx (0.571, 0.143, 0.286)$. Adding one to every count before normalizing is called *add-one smoothing*, and it is the posterior mean under this prior; the classifier of Section 36.11 uses the case $K = 2$.

A closed-form posterior is the exception: outside the conjugate pairs it must be computed numerically (Section 36.8). Something more basic is missing first. An estimate from 13 flips is a number without an error bar: how far can $9/13$ be from the true bias, and do two coins with 9 and 11 heads in 13 flips differ? Section 36.5 takes up these questions.

36.4.6 Summary

Nearly every distribution of the catalogue has the form $h(\mathbf{x}) \exp(\eta^\top T(\mathbf{x}) - A(\eta))$, in which the data and the parameters meet only in the inner product $\eta^\top T(\mathbf{x})$, and maximum entropy produces this form when only the averages of the statistics are fixed. The log-partition function generates moments: ∇A is the mean of T and $\nabla^2 A$ its covariance, so A is convex. Maximum likelihood is therefore a convex problem that matches the mean of T to its sample average, and it has a solution only if that average is attainable. A generalized linear model with the canonical link trains with the gradient input times prediction minus observation, which is why linear, logistic, softmax and Poisson regression share one rule; other links, such as the probit, do not. The softmax is the mean of the categorical family. Every exponential family has a conjugate prior whose update adds the sufficient statistics; for the coin, $\mathrm{Beta}(a, b)$ adds a and b pseudo-counts to the posterior mean and $a - 1$ and $b - 1$ to the mode.

36.4.7 Exercises

1. **Exponential races.** Let $X_1, \dots, X_m$ be independent with $X_i \sim \mathrm{Exp}(\lambda_i)$, for example the waiting times for the next event in m independent event streams.
   1. Show that $\min_i X_i \sim \mathrm{Exp}(\sum_i \lambda_i)$.
   2. Show that the j-th stream fires first with probability $P(X_j = \min_i X_i) = \lambda_j / \sum_i \lambda_i$.
   3. Let $V_1, \dots, V_K$ be independent $\mathrm{Exp}(1)$ variables and let $\mathbf{z} \in \mathbb{R}^K$ be a vector of logits. Use (b) to show that $\operatorname{argmax}_k (z_k - \log V_k)$ is distributed according to the softmax probabilities Equation 36.56. Because $-\log V_k$ has the standard Gumbel distribution, this is a second proof that Gumbel noise gives the softmax.

2. **Maximum entropy on the integers.** Consider distributions $(p_0, p_1, p_2, \dots)$ on $\{0, 1, 2, \dots\}$ with a fixed mean $m > 0$.
   1. Maximize the Shannon entropy $H[p] = -\sum_k p_k \log p_k$ subject to $\sum_k p_k = 1$ and $\sum_k k\, p_k = m$, using Lagrange multipliers (Section 35.6.1). Show that the stationary point is a geometric distribution $p_k = (1-r)\, r^k$, and express r through m.
   2. Repeat the derivation for the entropy relative to $h(k) = 1/k!$, $H_h[p] = -\sum_k p_k \log(p_k/h(k))$, and show that the stationary point is Poisson(m).
   3. At $m = 4$, compute the Shannon entropy of both distributions numerically. Which one is larger, and why must it be? Compare $P(X \geq 15)$ under the two distributions. Which assumption about large counts does the choice of base measure encode?
   4. A stationary point of a Lagrangian need not be a maximum. Use the maximum-entropy proposition to prove that the two distributions of
      (a) and (b) are the global maximizers, by exhibiting each as a member of an exponential family whose mean equals m.
3. **Moments from the log-partition.** The moment property Equation 36.53 and its second-order form $\nabla^2 A = \mathrm{Cov}(T)$ turn moment calculations into differentiation.
   1. For the Poisson, $A(\eta) = e^\eta$. Derive the mean and the variance from A. Which property of the function A makes them equal?
   2. For the Gaussian, the natural parameters are $\eta_1 = \mu/\sigma^2$ and $\eta_2 = -1/(2\sigma^2)$. Show that $A = -\eta_1^2/(4\eta_2) - \frac{1}{2}\log(-2\eta_2)$, and differentiate it to recover $\mathbb{E}[x]$, $\mathbb{E}[x^2]$, $\mathrm{Var}(x)$ and $\mathrm{Cov}(x, x^2)$.
   3. The Laplace distribution with a fixed location μ and scale b has the sufficient statistic $T(x) = |x - \mu|$. Write it in the form Equation 36.50, and use A to compute $\mathbb{E}\,|x-\mu|$ and $\mathrm{Var}|x-\mu|$. Check the result against the Laplace variance $2b^2$.
   4. For n Gaussian observations, write the two moment-matching equations $\nabla A(\hat\eta) = \bar T$ and solve them for $\hat\mu$ and $\hat\sigma^2$. What happens for $n = 1$? Relate your answer to the condition that $\bar T$ be an attainable mean.
4. code **Probit and logit.** For two classes with the score difference $m = z_1 - z_2$, the logit model gives $P(y=1) = \mathrm{sigmoid}(m)$ and the probit model with noise scale s gives $P(y=1) = \Phi(m/(\sqrt{2}\, s))$.
   1. Show that the difference of two independent standard Gumbel variables has the CDF sigmoid(x). The standard Gumbel distribution has variance $\pi^2/6$. Find the scale c for which the distribution with CDF sigmoid(cx) has variance 1, like the standard Gaussian.
   2. Find numerically the scale c that minimizes $\max_x |\,\mathrm{sigmoid}(cx) - \Phi(x)|$, and the value of this maximum. Why is this scale smaller than the one of (a)?
   3. An example of class 2 lies on the wrong side of the decision boundary, with $m > 0$. Show that its negative log-likelihood $-\log \mathrm{sigmoid}(-m)$ under the logit model lies between m and $m + e^{-m}$, and that its derivative with respect to m is less than 1. Use the bounds $\frac{x}{1+x^2}\,\phi(x) \leq 1 - \Phi(x) \leq \frac{\phi(x)}{x}$ for $x > 0$ to show that its negative log-likelihood $-\log \Phi(-m/(\sqrt{2}\, s))$ under the probit model grows like $m^2/(4s^2)$, and that its derivative grows linearly in m.
   4. Draw 100 inputs $x \sim \mathcal{N}(1, 1)$ of class 1 and 100 inputs $x \sim \mathcal{N}(-1, 1)$ of class 2, and let $m = wx + b$. Fit (w, b) by maximum likelihood under both models. Then add three examples at $x = 6$ labeled as class 2, and fit again. For each model,

compute the ratio of the slope w with these mislabeled examples to the slope without them, and explain the difference between the models with (c). Compute $\log \Phi$ with a numerically stable routine such as `scipy.special.log_ndtr`.

5. **Minimal categorical family.** ▣ Code a categorical outcome as a one-hot vector $\mathbf{x} \in \{0,1\}^K$ with class probabilities $\mathbf{p} = \text{softmax}(\mathbf{z})$, written as an exponential family with $T(\mathbf{x}) = \mathbf{x}$, natural parameters $\mathbf{z}$ and $A(\mathbf{z}) = \log \sum_k e^{z_k}$.
   1. Compute $\nabla^2 A(\mathbf{z})$ and show that it equals the covariance $\text{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top$ of the one-hot vector $\mathbf{x}$. Every one-hot vector satisfies $\mathbf{x}\mathbf{x}^\top = \text{diag}(\mathbf{x})$. Show that $\nabla^2 A(\mathbf{z})\,\mathbf{1} = \mathbf{0}$, and that the gradient of the negative log-likelihood of a dataset is orthogonal to $\mathbf{1}$ at every $\mathbf{z}$. Gradient descent with a small step size runs on the logits from a starting point $\mathbf{z}_0$, and every class occurs in the data. Which function of $\mathbf{z}$ does gradient descent never change? Which of the minimizers does it therefore approach?
   2. Fix $z_K = 0$. This gives a minimal representation with $\eta = (z_1, \dots, z_{K-1})$ and $T(\mathbf{x}) = (x_1, \dots, x_{K-1})$. Show that $\nabla^2 A(\eta)$ is positive definite whenever every $p_k > 0$. Given class counts $n_1, \dots, n_K$, solve the moment-matching condition $\nabla A(\hat{\eta}) = \bar{T}$ for the maximum-likelihood $\hat{\eta}$. What happens when some class has $n_k = 0$?
6. code **Poisson regression.** ▣ A generalized linear model for counts makes the natural parameter of a Poisson an affine function of the features: $y_i \mid \mathbf{x}_i \sim \text{Poisson}(e^{\eta_i})$ with $\eta_i = \mathbf{w}^\top \mathbf{x}_i + b$.
   1. Use the negative log-likelihood Equation 36.54 to write the loss as a function of $(\mathbf{w}, b)$, and compute its gradient and Hessian. Show that the loss is convex. Which equations does a zero gradient impose on the fitted means $\hat{m}_i = e^{\hat{\eta}_i}$?
   2. Generate $n = 1000$ examples with $\mathbf{x}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_2)$, $\mathbf{w} = (-0.3, 1.0)$ and $b = 0.5$. Starting from zero, fit $\mathbf{u} = (\mathbf{w}, b)$ with Newton's method, which repeats the step $\mathbf{u} \leftarrow \mathbf{u} - \left(\nabla^2 L(\mathbf{u})\right)^{-1} \nabla L(\mathbf{u})$ for the loss L, and check the equations of (a) at the solution.
   3. Multiply each rate by an independent $g_i \sim \text{Gamma}(1, 1)$, which has mean 1, so that the counts become overdispersed while their conditional means stay e^{η_i}. Fit the Poisson model to 200 such datasets and to 200 datasets without the multiplier. Compare the mean and the standard deviation of the estimates across datasets. Explain the behavior of the mean with the equations of (a).
7. **Conjugate priors from natural parameters.** ▣ For a scalar natural parameter η, the conjugate prior Equation 36.59 reads $p(\eta) \propto \exp\big(\nu\,\eta - \kappa\,A(\eta)\big)$, and n observations update it by $\nu \to \nu + \sum_i T(x_i)$ and $\kappa \to \kappa + n$.
   1. For the Poisson, $\eta = \log \lambda$ and $A(\eta) = e^\eta$. Change variables from η to λ, including the Jacobian, and show that the prior on λ is a Gamma distribution. Identify its shape and rate, and check that the update reproduces the Gamma posterior of Section 36.4.5.3.
   2. For the Bernoulli, $\eta = \log \frac{\theta}{1-\theta}$ and $A(\eta) = \log(1 + e^\eta)$. Show that the prior on θ is $\text{Beta}(\nu, \kappa - \nu)$, and find the values of (ν, κ) for which it is normalizable.
   3. Take the flat prior on the logit, $\nu = \kappa = 0$. Which density on θ does it induce, and is that density normalizable? After x heads in n flips, when is the posterior normalizable, and what is its mean? Compare it with the mean of the posterior Equation 36.58 under the uniform prior $\text{Beta}(1, 1)$ on θ. What does the comparison say about the phrase "a flat prior"?
8. code **Moment matching outside the family.** Let $x_1, \dots, x_n$ be independent draws from $U(0, \theta)$.

   1. Write the likelihood and show that the maximum-likelihood estimate is $\hat{\theta}_{\text{ML}} = \max_i x_i$. Using the CDF $(t/\theta)^n$ of the maximum, show that $\mathbb{E}[\hat{\theta}_{\text{ML}}] = n\theta/(n+1)$ and $\text{Var}(\hat{\theta}_{\text{ML}}) = n\theta^2/((n+1)^2(n+2))$.
   2. Show that the moment estimate $\hat{\theta}_{\text{MM}} = 2\bar{x}$ is unbiased with variance $\theta^2/(3n)$.
   3. With $\theta = 1$, simulate $20{,}000$ data sets for each of $n = 10, 100, 1000$ and report the root-mean-square error of both estimates. Which one falls like $1/n$ and which like $1/\sqrt{n}$? Which property of the uniform distribution, absent in every exponential family, allows the faster rate?
   4. Show that $n(\theta - \hat{\theta}_{\text{ML}})$ converges in distribution to an exponential distribution with mean θ, and check the mean in your simulation at $n = 1000$.

9. code **A likelihood with no maximizer.** This exercise follows the coin with 13 heads in 13 flips.
   1. Show that the negative log-likelihood in the logit is $13\log(1 + e^{-\eta})$, that it decreases for every η with infimum 0, and that it has no minimizer. What is $\bar{T}$, and why is it not an attainable mean?
   2. Under the prior $\text{Beta}(2, 2)$ the posterior is $\text{Beta}(15, 2)$. Compute its mode and its mean, and explain why its one imagined head and one imagined tail in the mode restore a solution strictly inside $(0, 1)$.
   3. Logistic regression on the two examples $(x, y) = (1, 1)$ and $(-1, 0)$, with $P(y = 1 \mid x) = \text{sigmoid}(wx)$, has the negative log-likelihood $2\log(1 + e^{-w})$, which has no minimizer. Add the penalty $\frac{\lambda}{2}w^2$ with $\lambda = 0.1$, show that the minimizer solves $2\,\text{sigmoid}(-w) = \lambda w$, solve this equation by bisection, and report w^* and the fitted probability $\text{sigmoid}(w^*)$. To which prior on w does the penalty correspond?

10. **Dirichlet smoothing and the next word.** A three-word vocabulary shows the counts $\mathbf{c} = (3, 0, 1)$, and the prior is $\text{Dir}(1, 1, 1)$.
   1. Write the posterior and its mean, and compare the mean with the maximum-likelihood estimate.
   2. Show that the probability that the next word is word k, $\int p_k \,\text{Dir}(\mathbf{p} \mid \mathbf{a} + \mathbf{c})\, d\mathbf{p}$, equals the posterior mean $(a_k + c_k)/\sum_j (a_j + c_j)$. Use $\Gamma(z+1) = z\,\Gamma(z)$ on the normalizer.
   3. For the prior $\text{Dir}(s, \dots, s)$ on K words, show that the posterior mean tends to the maximum-likelihood estimate as $s \to 0$ and to the uniform distribution as $s \to \infty$.
   4. A corpus has 10^6 words and a vocabulary of $50{,}000$, of which $20{,}000$ never occur. With add-one smoothing ($s = 1$), by what factor does the estimate for a word seen 5 times change, and what total probability do the unseen words receive?

36.5 Statistics

Every model in this book has been judged by a number computed on held-out data: a test accuracy, a validation loss, a score on a benchmark. We read such a number as a property of the model. It is also a property of the particular examples that happened to land in the test set. Suppose a classifier answers 912 of 1,000 test questions correctly. On another 1,000 questions from the same source it would score a little higher or a little lower, so its accuracy of 91.2% is one draw of a random number. A second model answers 921 of the same questions. Is it the better model, or did these questions happen to suit it?

Statistics answers such questions by asking how an estimate would vary over the samples we did not draw. That variation has a distribution, the *sampling distribution* of the estimate, and every tool in this section describes it or reasons with it. Its center and its width say whether an estimate is systematically off and how much it scatters. A hypothesis test asks whether an observed difference fits inside the scatter that chance alone produces. A confidence interval reports the parameter values that the data cannot rule out. When no formula describes the sampling distribution, the bootstrap simulates it from the data themselves.

These tools matter because most decisions in deep learning are made by comparing such numbers: which model ships, which hyperparameter wins, whether an idea helped. A decision based on a gap smaller than its own noise is a decision made by chance. The previous sections described how data vary when the distribution is known; this one turns data into estimates with error bars. The data need not come from the world. An expectation that we estimate by averaging samples drawn by the computer, a Monte Carlo estimate, is a sample mean like any other, and the same theory gives its error bar (Section 36.5.1.7). A test accuracy is a coin with one flip per question, so the coin of the earlier sections, 9 heads in 13 flips, returns along the way as a check.

Three results are worth remembering: the mean squared error is the squared bias plus the variance; a p-value is uniform under the null hypothesis; and a confidence interval is the set of values that a test keeps. Consistency and the delta method can be skipped on a first reading. Every guarantee below holds for an estimator or a test chosen before the data are seen; the end of the section says what goes wrong otherwise.

```
from scipy.stats import norm
```

```
rng = np.random.default_rng()
```

36.5.1 Estimators and Their Sampling Distribution

Before we can ask whether 91.2% is trustworthy, we need a name for the rule that produced it and a model of how the test questions arrived.

36.5.1.1 Estimators and the Sample Proportion

An *estimator* is a rule that turns data $x_1, \dots, x_n$ into a number $\hat{\theta}_n$ meant to approximate an unknown quantity θ. The sample mean, the sample median and the fraction of correct answers are estimators. Because the data are random, so is $\hat{\theta}_n$. Its distribution over repeated samples of size n is its sampling distribution.

For the test accuracy, let $x_i = 1$ if the classifier answers question i correctly and $x_i = 0$ otherwise. If the questions are drawn independently from a large pool on which the classifier's accuracy is θ, the x_i are independent Bernoulli variables with mean θ. The accuracy is unknown, so we write θ for it and $\hat{\theta}$ for its estimate, the fraction of correct answers among the n. By linearity of expectation $\mathbb{E}[\hat{\theta}] = \theta$, and the variance of a frequency is $\mathrm{Var}(\hat{\theta}) = \theta(1-\theta)/n$ by Equation 1.49. The estimate is centered on the true accuracy, and its scatter shrinks like $1/\sqrt{n}$. At $\theta = 0.912$ and $n = 1{,}000$ the standard deviation is $\sqrt{0.912 \cdot 0.088/1000} \approx 0.0090$, or 0.9 percentage points. By the central limit theorem the

sampling distribution is close to Gaussian, so about 95% of test sets give an accuracy within ± 1.8 points of θ.

We can watch the sampling distribution directly: draw 10,000 test sets of 1,000 questions for a classifier whose accuracy is 0.912, and compute the accuracy on each.

```
theta, n = 0.912, 1000
acc = rng.binomial(n, theta, 10000) / n      # accuracy on 10,000 test sets
print(f'mean {acc.mean():.4f}, std {acc.std():.4f}, '
      f'formula {np.sqrt(theta * (1 - theta) / n):.4f}')
```

```
mean 0.9119, std 0.0089, formula 0.0090
```

As predicted, they center on 0.912 with a standard deviation of 0.0090. These 10,000 accuracies, on test sets we did not draw, are the object that every later tool in this section reasons about.

Now the second model. If its 921 correct answers had come from an independent test set, the gap of 0.009 between the two accuracies would have a standard deviation of $\sqrt{0.0090^2 + 0.0085^2} \approx 0.012$, where 0.0085 is the same formula at 0.921. The gap is 0.7 of its own standard error, so a single comparison of this kind cannot tell the models apart. Both models in fact answered the same questions, and Section 36.5.2.6 shows how much sharper the comparison becomes when we use this fact.

36.5.1.2 Bias and Variance

An estimator can go wrong in two ways: it can be off on average, or it can scatter widely around its own average. The *bias* of $\hat{\theta}_n$ is the offset of its center from the truth,

$$\mathrm{Bias}(\hat{\theta}_n) = \mathbb{E}[\hat{\theta}_n] - \theta, \tag{36.60}$$

where the expectation runs over the random sample. An estimator whose bias is zero for every value of θ is *unbiased*. The *variance* measures the scatter around the estimator's own center, and its square root is the *standard error*,

$$\mathrm{Var}(\hat{\theta}_n) = \mathbb{E}\big[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^2\big], \qquad \mathrm{se}(\hat{\theta}_n) = \sqrt{\mathrm{Var}(\hat{\theta}_n)}. \tag{36.61}$$

The bias says where the estimates land on average; the standard error says how far a single estimate typically lands from there. The sample proportion has bias 0 and standard error 0.009 in the example above. Variance is measured from the estimator's center, not from θ: the rule that ignores the data and always answers 0.5 has variance zero, and all of its error is bias.

36.5.1.3 The Sample Variance

The sample proportion is unbiased. A biased estimator turns up as soon as a standard error has to be computed from data: the standard error of a mean contains the variance σ^2 of the data, which is rarely known and must be estimated from the same sample. The natural estimate is the *plug-in variance*, the average squared deviation from the sample mean,

$$s_0^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$

It is biased low. The deviations are measured from $\bar{x}$, the point that makes the sum of squared deviations smallest for this sample, while the true mean μ is a different point at which the same sum is larger.

The smallest case shows by how much. For $n = 2$ the sample mean is the midpoint, so both deviations are $\pm(x_1 - x_2)/2$, and

$$\sum_{i=1}^{2}(x_i - \bar{x})^2 = \frac{(x_1 - x_2)^2}{2}.$$

The difference $x_1 - x_2$ has mean 0 and variance $2\sigma^2$, so the right side has expectation σ^2. Dividing by $n = 2$ gives an estimator whose average is $\sigma^2/2$, half the truth; dividing by $n - 1 = 1$ gives σ^2 on average. The same correction works for every n.

Proposition (unbiased sample variance). *Let $x_1, \dots, x_n$ be i.i.d. with mean μ and finite variance σ^2, and let $n \geq 2$. Then*

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \qquad \text{satisfies} \qquad \mathbb{E}[s^2] = \sigma^2. \tag{36.62}$$

Proof. We compare the deviations from $\bar{x}$ with those from μ. Write $x_i - \bar{x} = (x_i - \mu) - (\bar{x} - \mu)$, expand the square, and use $\sum_i (x_i - \mu) = n(\bar{x} - \mu)$ in the cross term:

$$\sum_{i=1}^{n}(x_i - \bar{x})^2 = \sum_{i=1}^{n}(x_i - \mu)^2 - 2(\bar{x} - \mu)\cdot n(\bar{x} - \mu) + n(\bar{x} - \mu)^2 = \sum_{i=1}^{n}(x_i - \mu)^2 - n(\bar{x} - \mu)^2.$$

The cross term is twice the last term with the opposite sign, so measuring from $\bar{x}$ removes $n(\bar{x} - \mu)^2$ from the sum. Now take expectations. Each $\mathbb{E}[(x_i - \mu)^2]$ equals σ^2, so the first sum has expectation $n\sigma^2$. The sample mean has variance σ^2/n, so $\mathbb{E}[n(\bar{x} - \mu)^2] = \sigma^2$. Hence the sum of squared deviations has expectation $(n - 1)\sigma^2$, and dividing by $n - 1$ gives $\mathbb{E}[s^2] = \sigma^2$. $\blacksquare$

The usual intuition counts *degrees of freedom*. The n deviations $x_i - \bar{x}$ sum to zero, so only $n - 1$ of them can vary freely: estimating $\bar{x}$ from the data used up one. The plug-in variance has bias $\frac{n-1}{n}\sigma^2 - \sigma^2 = -\sigma^2/n$, which vanishes as n grows.

Unbiasedness does not survive a nonlinear map. Jensen's inequality for the concave square root gives $\mathbb{E}[s] \leq \sqrt{\mathbb{E}[s^2]} = \sigma$, with strict inequality unless s is constant. For Gaussian data at $n = 2$, the sample standard deviation is $s = |x_1 - x_2|/\sqrt{2}$, and $\mathbb{E}[s] = \sigma\sqrt{2/\pi} \approx 0.80\,\sigma$: the square root of an unbiased variance underestimates the standard deviation by 20% on average.

Numerical libraries select the denominator with a "delta degrees of freedom" argument: `np.var(x, ddof=0)` divides by n, and `ddof=1` divides by $n - 1$.

36.5.1.4 The Bias–Variance Decomposition

To see the two kinds of error side by side we need an estimator that trades one for the other. Take $n = 30$ measurements $x_i \sim \mathcal{N}(\theta, \sigma^2)$ with $\theta = 1$ and $\sigma = 4$, a small sample with a lot of noise. The sample mean $\bar{x}$ is unbiased with variance $\sigma^2/n = 16/30 \approx 0.533$. The *shrunk mean* $c\,\bar{x}$, with a factor $0 < c < 1$, pulls the sample mean toward 0. Its bias is $c\theta - \theta = (c-1)\theta$ and its variance is $c^2\sigma^2/n$. At $c = 0.65$ the bias is -0.35, and the standard error drops from 0.73 to 0.47. Figure 36.18 draws both sampling distributions.

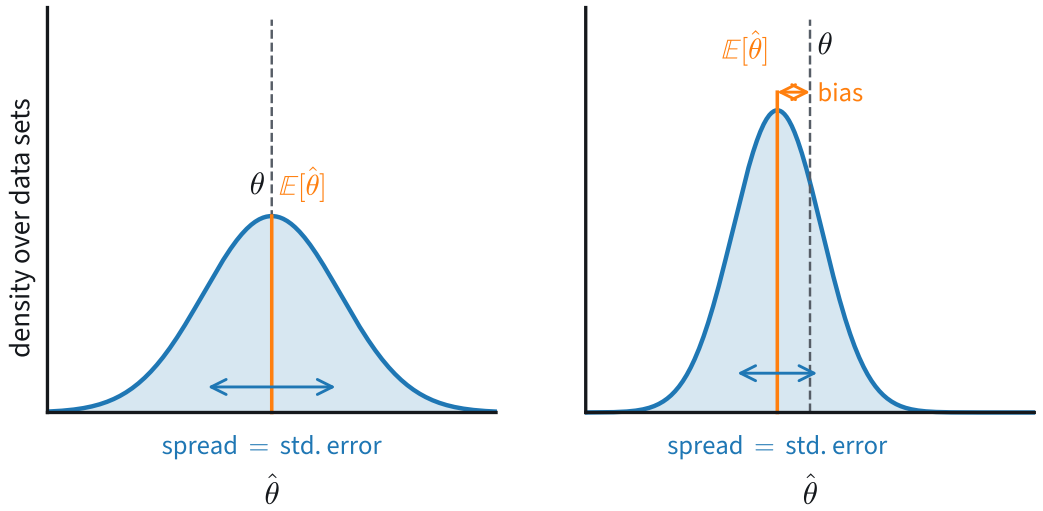


Figure 36.18: The sampling distributions of two estimators of $\theta = 1$ from 30 draws of $\mathcal{N}(1, 16)$, over many data sets. Left: the sample mean $\bar{x}$, with bias 0 and standard error 0.73. Right: the shrunk mean $0.65\,\bar{x}$, with bias -0.35 and standard error 0.47. The bias is the offset of the center from θ; the standard error is the width.

Which of the two is better? The shrunk mean is wrong on average, yet a single estimate from it often lands closer to θ. Settling the question needs one number that counts both kinds of error. The *mean squared error* is that number, the average squared distance of the estimator from the truth,

$$\mathrm{MSE}(\hat{\theta}_n) = \mathbb{E}\big[(\hat{\theta}_n - \theta)^2\big]. \tag{36.63}$$

It splits into the two errors we already have.

Proposition (bias–variance decomposition). *If $\hat{\theta}_n$ has finite variance, then*

$$\mathrm{MSE}(\hat{\theta}_n) = \mathrm{Bias}(\hat{\theta}_n)^2 + \mathrm{Var}(\hat{\theta}_n). \tag{36.64}$$

Proof. Let $m = \mathbb{E}[\hat{\theta}_n]$ be the center of the estimator, and add and subtract it inside the square:

$$(\hat{\theta}_n - \theta)^2 = (\hat{\theta}_n - m)^2 + 2(\hat{\theta}_n - m)(m - \theta) + (m - \theta)^2.$$

Take expectations term by term. The first term gives $\mathrm{Var}(\hat{\theta}_n)$, and the last is the constant $\mathrm{Bias}(\hat{\theta}_n)^2$. The middle term vanishes, because $m - \theta$ is a constant and $\mathbb{E}[\hat{\theta}_n - m] = 0$ by the definition of m. ■

The decomposition is an instance of Pythagoras' theorem. Treat random variables as vectors with the inner product $\langle U, V \rangle = \mathbb{E}[UV]$. The fluctuation $\hat{\theta}_n - m$ has mean zero, which says that it is orthogonal to every constant a: $\langle \hat{\theta}_n - m,\, a \rangle = a\,\mathbb{E}[\hat{\theta}_n - m] = 0$. The error $\hat{\theta}_n - \theta$ is the sum of this fluctuation and the constant $m - \theta$, so its squared length, the MSE, is the sum of the squared lengths of two orthogonal legs, se^2 and Bias^2. In the same geometry, $\mathbb{E}[X]$ is the orthogonal projection of X onto the constants, the projection of Section 33.1 with an inner product built from expectations. Figure 36.19 draws the triangle.

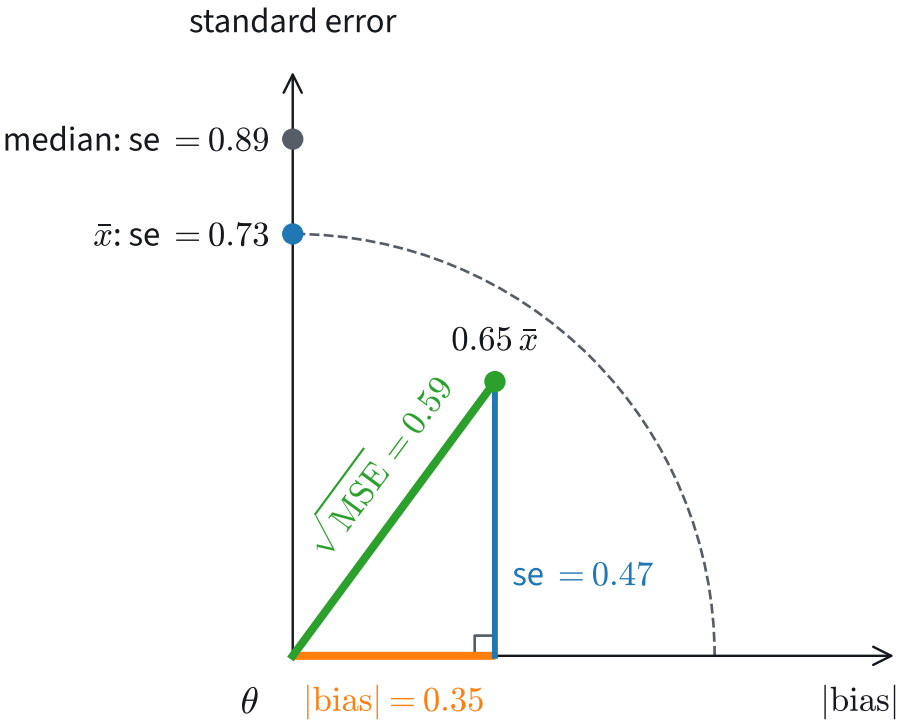


Figure 36.19: The bias–variance decomposition as Pythagoras' theorem, for estimators of $\theta = 1$ from 30 draws of $\mathcal{N}(1, 16)$. Each estimator is a point at horizontal distance $|\mathrm{bias}|$ and height se, and its distance from the origin is $\sqrt{\mathrm{MSE}}$. Shrinking $\bar{x}$ to $0.65\,\bar{x}$ adds a horizontal leg of 0.35 but shortens the vertical one from 0.73 to 0.47, so the hypotenuse falls to 0.59 and the point lies inside the dashed circle of estimators as good as $\bar{x}$. The sample median is unbiased but lies outside it.

The decomposition lets us rank estimators. For data sets of 30 draws it predicts an MSE of 0.533 for the sample mean, all of it variance, and $0.123 + 0.225 = 0.348$ for the shrunk mean $0.65\,\bar{x}$. The sample median is unbiased here by symmetry, but we have no formula for its variance yet. The cells below measure all three over 10,000 data sets.

```
theta, sigma, n = 1.0, 4.0, 30
data = rng.normal(theta, sigma, (10000, n))   # 10,000 data sets of size 30
estimates = {'mean': data.mean(axis=1),
             '0.65 * mean': 0.65 * data.mean(axis=1),
             'median': np.median(data, axis=1)}
```

```
predicted = {'mean': sigma**2 / n,            # bias^2 + variance, by formula
             '0.65 * mean': (0.35 * theta)**2 + 0.65**2 * sigma**2 / n,
             'median': None}                  # no formula yet
print(f'{"estimator":>12} {"bias^2":>7} {"var":>6} {"MSE":>6} {"predicted":>9}')
for name, est in estimates.items():
    bias2 = (est.mean() - theta) ** 2
    var = est.var()                           # a plain average: divides by the count
    mse = ((est - theta) ** 2).mean()
    pred = '-' if predicted[name] is None else f'{predicted[name]:.3f}'
    print(f'{name:>12} {bias2:7.3f} {var:6.3f} {mse:6.3f} {pred:>9}')
```

```
   estimator  bias^2    var    MSE predicted
        mean   0.000  0.532  0.532     0.533
0.65 * mean   0.125  0.225  0.350     0.348
      median   0.000  0.805  0.805         -
```

Every run gives an MSE of about 0.53 for the sample mean and about $0.12 + 0.23 = 0.35$ for the shrunk mean, as computed. The median has about 0.80: it is unbiased, but its variance is half again that of $\bar{x}$. A biased estimator wins, and an unbiased estimator can still be a poor one.

36.5.1.5 The Shrunk Mean and the Trade-off

The table raises a question: how much bias should we accept in exchange for less variance? For the shrunk mean the answer is a short calculation. By Equation 36.64, with the bias and variance found above,

$$\mathrm{MSE}(c) = (1-c)^2\theta^2 + c^2\frac{\sigma^2}{n}.$$

The first term grows as we shrink harder, and the second falls. Setting the derivative $-2(1-c)\theta^2 + 2c\,\sigma^2/n$ to zero gives the best factor and its error,

$$c^\star = \frac{\theta^2}{\theta^2 + \sigma^2/n}, \qquad \mathrm{MSE}(c^\star) = \frac{\theta^2\,\sigma^2/n}{\theta^2 + \sigma^2/n} < \frac{\sigma^2}{n}.$$

The best shrinkage always beats the sample mean, and it helps most when the noise σ^2/n is large compared with the signal θ^2. At $\theta = 1$, $\sigma = 4$ and $n = 30$ we get $c^\star = 1/1.533 \approx 0.65$, the factor used above, and the MSE falls from 0.533 to 0.348. Figure 36.20 plots the two terms and their sum against c. Here the U shape is exact: it is the sum of a falling parabola and a rising one.

Unfortunately, the optimal factor $c^\star$ contains the unknown θ, so it is not an estimator we can compute. A factor fixed without knowing θ can do harm. At $\theta = 10$ the same $c = 0.65$ has

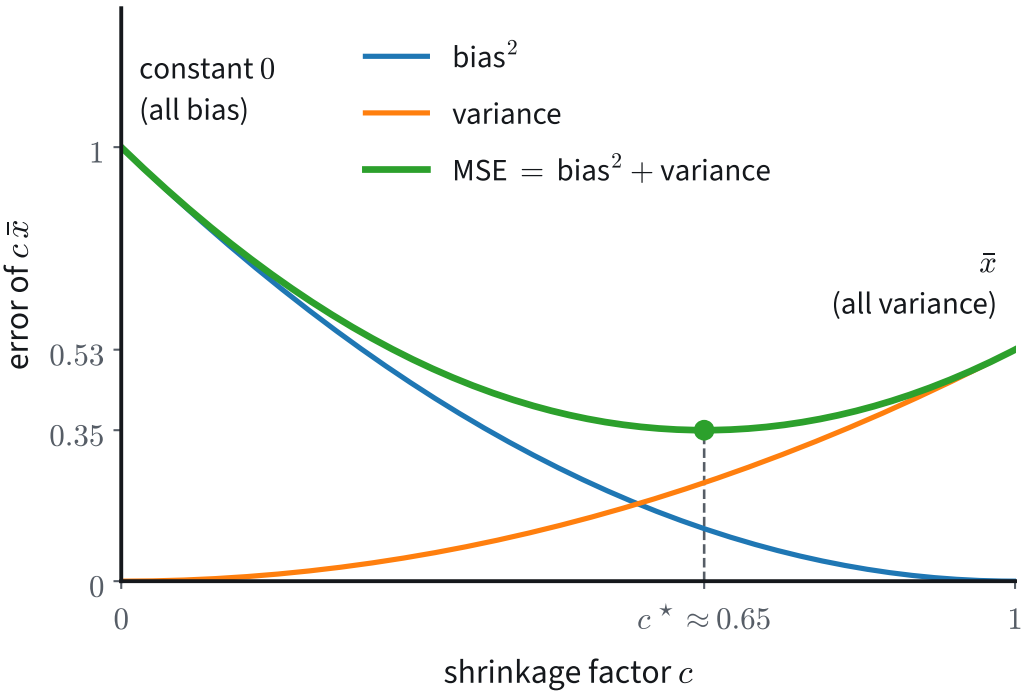


Figure 36.20: Squared bias and variance of the shrunk mean $c\,\bar{x}$ as functions of the factor c, for $\theta = 1$, $\sigma = 4$ and $n = 30$. Their sum, the MSE, is smallest at $c^\star \approx 0.65$, where it is 0.35 against 0.53 for the sample mean at $c = 1$. The curves are exact for this estimator.

bias -3.5 and MSE $12.25 + 0.23 \approx 12.5$, more than twenty times the 0.533 of $\bar{x}$. Shrinking toward 0 helps only when the truth is near 0. Exercise 1 replaces θ in $c^\star$ by its estimate, and exercise 13 shows that in three or more dimensions a similar substitution beats the sample mean for every θ.

The same trade-off governs prediction. At a fixed input x, the prediction $\hat{f}(x)$ of a model fitted to a random training set is an estimator of the unknown number $f^*(x)$. Its expected squared error on a fresh label is the MSE of this estimator plus the variance of the label noise, which no model can remove (Equation 2.28 in Section 2.6). In many families of models, more flexibility lowers the bias and raises the variance, and regularization plays the role of the factor c (Geman et al. 1992). Unlike the MSE of the shrunk mean, the error along such a family need not trace a U: past the point where a model fits its training data exactly, the test error can fall again (Section 36.10.3.2).

36.5.1.6 Consistency and Efficiency

Bias and variance describe an estimator at one sample size. A different question is whether it gets things right eventually: with enough data, does the estimate settle on the truth? An estimator is *consistent* if it *converges in probability* to θ,

$$P(|\hat{\theta}_n - \theta| > \varepsilon) \to 0 \quad \text{as } n \to \infty, \text{ for every } \varepsilon > 0,$$

written $\hat{\theta}_n \xrightarrow{P} \theta$: the chance of missing θ by more than any fixed margin vanishes as the sample grows.

The decomposition gives a simple sufficient condition. Markov's inequality Equation 36.13, applied to the nonnegative variable $(\hat{\theta}_n - \theta)^2$ with the threshold ε^2, bounds the chance of a large miss by the MSE,

$$P(|\hat{\theta}_n - \theta| > \varepsilon) \le \frac{\mathrm{MSE}(\hat{\theta}_n)}{\varepsilon^2} = \frac{\mathrm{Bias}(\hat{\theta}_n)^2 + \mathrm{Var}(\hat{\theta}_n)}{\varepsilon^2}.$$

If both the bias and the variance tend to zero, the estimator is therefore consistent. Neither condition suffices alone. The first observation by itself, $\hat{\theta}_n = x_1$, is unbiased at every n, but its variance stays σ^2 and it never settles. The estimator $\bar{x} + 1$ has variance $\sigma^2/n \to 0$, but it settles on the wrong value. The plug-in variance s_0^2 shows the remaining case: it is biased at every n, but its bias $-\sigma^2/n$ and, for data with a finite fourth moment, its variance both vanish, so it is consistent.

For the sample mean, the condition gives the law that makes averaging work.

Proposition (weak law of large numbers). *If $x_1, \dots, x_n$ are i.i.d. with mean μ and finite variance σ^2, then the sample mean $\bar{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$ satisfies*

$$P(|\bar{x}_n - \mu| > \varepsilon) \le \frac{\sigma^2}{n\,\varepsilon^2} \quad \text{for every } \varepsilon > 0, \qquad \text{so} \qquad \bar{x}_n \xrightarrow{P} \mu. \tag{36.65}$$

Proof. By linearity $\mathbb{E}[\bar{x}_n] = \mu$, so the bias is zero. The variance of a sum of independent variables is the sum of their variances, so $\mathrm{Var}(\bar{x}_n) = n\sigma^2/n^2 = \sigma^2/n$. The bound above with $\mathrm{MSE} = \sigma^2/n$ is the displayed inequality, and its right side tends to zero. ■

Finite variance is more than the law needs: a finite mean $\mathbb{E}|x_1| < \infty$ suffices, with a longer proof (Wasserman 2013). The law is why a Monte Carlo estimate of an integral converges, why the accuracy on a large test set estimates the true accuracy, and why the average training loss tracks the expected loss it stands in for. The bound is crude. For our classifier it says that an accuracy measured on 1,000 questions misses by more than 3 points with probability at most $0.0803/(1000 \cdot 0.03^2) \approx 0.09$, while the Gaussian approximation puts the probability near 0.001. Its strength is that it needs nothing beyond a finite variance; Section 36.9 derives the sharper bounds that more assumptions allow.

Consistency does not rank two estimators that are both consistent. Among unbiased estimators, the one with the smaller variance is called more *efficient*: it reaches the same precision with less data. For Gaussian data the sample median is unbiased for the mean, like $\bar{x}$, but for large n its variance is about $\frac{\pi}{2}\,\sigma^2/n$ (exercise 9). The median therefore needs about $\pi/2 \approx 1.57$ times as many observations for the same standard error, an efficiency of $2/\pi \approx 64\%$; at $n = 30$ the table above shows 0.80 against 0.53. The sample proportion, by contrast, has the smallest variance that any unbiased estimator of a Bernoulli parameter can have, a floor known as the Cramér–Rao bound (Section 36.6.3.4).

36.5.1.7 Monte Carlo Estimation

So far the sample came from the world: test questions, measurements, random seeds. The law of large numbers does not ask where a sample comes from, and this gives the sample mean a second use, as a way to compute. Most expectations in deep learning (an expected loss, a normalizing constant, the gradient of either) are integrals with no closed form, and many run over thousands of dimensions, where no grid is affordable. Samples we can still draw. The idea dates from the first electronic computers (Metropolis and Ulam 1949): to compute $\mathbb{E}[g(X)]$ for X with density p, draw independent samples $x_1, \dots, x_n$ from p and average,

$$\mathbb{E}[g(X)] \approx \frac{1}{n} \sum_{i=1}^{n} g(x_i). \tag{36.66}$$

The **Monte Carlo** estimate is a Riemann sum whose slices are placed by chance, with equal weight $1/n$: regions of high density receive more samples. By the law of large numbers Equation 36.65 the average converges to $\mathbb{E}[g(X)]$ as $n \to \infty$, and it is unbiased. How fast it converges is its variance. If $X_1, \dots, X_n$ are independent with density p and $\sigma^2 = \mathrm{Var}(g(X))$ is finite, then

$$\mathrm{Var}\Big(\frac{1}{n} \sum_{i=1}^{n} g(X_i)\Big) = \frac{\sigma^2}{n}. \tag{36.67}$$

There is nothing new in this formula. It is the variance σ^2/n of a sample mean from the proof of the law of large numbers, with $g(X_i)$ in the place of x_i (Equation 36.26 in Section 36.2). What is new is the use we make of it. The estimate's root-mean-square error is $\sigma/\sqrt{n}$: a hundred times more samples give one more decimal digit. That is slow. But the rate does not mention the dimension of x, and this is the whole case for sampling. It does need a finite variance, and Section 36.1 shows what heavy tails do to an average (see also exercise 11).

A grid behaves differently. To resolution ϵ in d dimensions it needs $N = \epsilon^{-d}$ points. The left rule of Section 34.6 has error proportional to $\epsilon = N^{-1/d}$, and a second-order rule, such as the *midpoint rule* (tag each slice at its center), has error proportional to $\epsilon^2 = N^{-2/d}$. Monte Carlo has $N^{-1/2}$ in every dimension. In $d = 1$ the second-order grid wins; at $d = 4$ the exponents tie; at $d = 10$ the grid falls to $N^{-1/5}$. The constant σ can grow with d, but the exponent cannot (exercise 10).

A single run shows what the rate means in practice. We estimate $\int_0^1 e^{-x^2} dx = \mathbb{E}[e^{-U^2}]$ for U uniform on $[0, 1]$, whose exact value $\frac{\sqrt{\pi}}{2}\,\mathrm{erf}(1) = 0.746824$ follows from the definition of erf, and print the running average after n samples, its error, and the predicted scale $\sigma/\sqrt{n}$, with $\sigma = \mathrm{sd}(e^{-U^2}) \approx 0.2$ estimated from the samples.

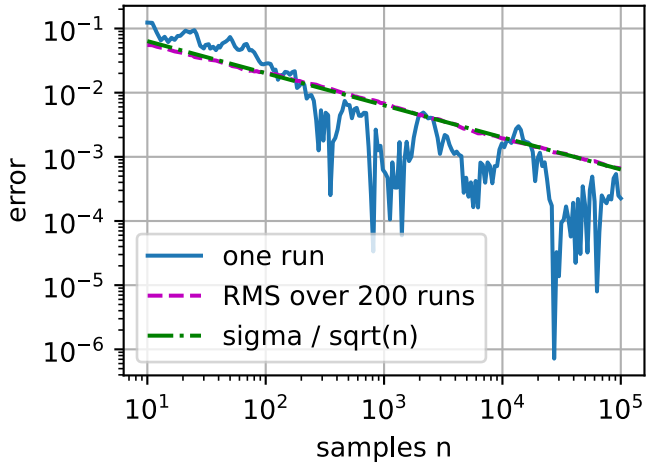
```
exact = math.sqrt(math.pi) / 2 * math.erf(1)
g = np.exp(-rng.uniform(size=10**5)**2)             # g(U), U uniform on [0, 1]
running = np.cumsum(g) / np.arange(1, 10**5 + 1)    # average of the first n
sigma = g.std()
for n in [10, 100, 1000, 10000, 100000]:
    print(f'n={n:<6}  estimate={running[n - 1]:.5f}  '
          f'error={abs(running[n - 1] - exact):.1e}  sigma/sqrt(n)={sigma / n**0.5:.1e}')
print(f'exact (sqrt(pi)/2) erf(1) = {exact:.6f}')
```

```
n=10      estimate=0.87045  error=1.2e-01  sigma/sqrt(n)=6.4e-02
n=100     estimate=0.77742  error=3.1e-02  sigma/sqrt(n)=2.0e-02
n=1000    estimate=0.74604  error=7.8e-04  sigma/sqrt(n)=6.4e-03
n=10000   estimate=0.74536  error=1.5e-03  sigma/sqrt(n)=2.0e-03
n=100000  estimate=0.74705  error=2.3e-04  sigma/sqrt(n)=6.4e-04
exact (sqrt(pi)/2) erf(1) = 0.746824
```

The errors have the predicted size, but each row is one draw of a random error: it rarely exceeds $\sigma/\sqrt{n}$ by more than a factor of two or three, and it can fall far below it. What does the error do between the rows? We plot it at 200 values of n on logarithmic axes, against the prediction, together with the root-mean-square error over 200 independent runs, which is what Equation 36.67 predicts.

```
n = np.unique(np.logspace(1, 5, 200).astype(int))
sq = np.zeros(len(n))
for _ in range(200):                                # 200 independent runs
    g_run = np.exp(-rng.uniform(size=10**5)**2)
    sq += (np.cumsum(g_run)[n - 1] / n - exact)**2
rms = np.sqrt(sq / 200)                             # root-mean-square error
ratio = rms * np.sqrt(n) / sigma
print(f'RMS error / (sigma/sqrt(n)) over all n: {ratio.min():.2f} to {ratio.max():.2f}')
d2l.plot(n, [np.abs(running[n - 1] - exact), rms, sigma / np.sqrt(n)],
         'samples n', 'error', xscale='log', yscale='log',
         legend=['one run', 'RMS over 200 runs', 'sigma / sqrt(n)'])
```

```
RMS error / (sigma/sqrt(n)) over all n: 0.87 to 1.08
```



A single running average wanders around the prediction, because its errors at nearby n share most of their samples: it can stay on one side of the line for a long stretch, and it dips sharply wherever the error changes sign. The root-mean-square error over the 200 runs stays within about a fifth of $\sigma/\sqrt{n}$ at every n, so it falls with slope $-\frac{1}{2}$ across all four decades.

36.5.2 Hypothesis Testing

An error bar describes one estimate. Most practical questions compare two: is a new model better than the old one? This part builds a test for that question by simulation, then derives the same test in closed form, and asks how often it catches a real improvement.

36.5.2.1 Comparison of Two Models

Training is random: the initialization, the order of the data and the dropout masks all depend on a seed, so two runs of the same model reach different accuracies. Suppose we train model A and model B with 20 seeds each and record their test accuracies. Seed noise moves a single run by about one percentage point, and model B is in truth better by 0.8 points. Looking only at the 40 numbers, can we tell? The natural summary is the gap between the two mean accuracies, which serves as the *test statistic* of the comparison.

A test starts from the claim that it tries to refute. The *null hypothesis* H_0 states that the two lists of accuracies come from the same distribution, so that in particular the two models are equally good on average. The *alternative* H_A states that the distributions differ. We ask whether the observed gap would be surprising if H_0 were true. If it would, we *reject* H_0; if not, we *fail to reject* it. Failing to reject is weaker than accepting, much as a court finds a defendant "not guilty" rather than "innocent."

36.5.2.2 The Permutation Test

To judge whether a gap is surprising under H_0 we need its distribution under H_0, the *null distribution*. A *permutation test* builds it from the data (Fisher 1935). If the two lists come from the same distribution, the labels "A" and "B" carry no information: every assignment of the 40 numbers to two groups of 20 was equally likely to produce what we saw. So we shuffle the labels, recompute the gap, and repeat R times. The shuffled gaps show how

large a gap chance alone produces. The function below does this for all R shuffles at once and counts how many shuffled gaps are at least as large in magnitude as the observed one.

```
def perm_p(acc_a, acc_b, rng, R=2000):
    """Two-sided permutation p-value and the R shuffled gaps."""
    k = len(acc_a)
    observed = acc_b.mean() - acc_a.mean()
    pooled = np.tile(np.concatenate([acc_a, acc_b]), (R, 1))
    shuffled = rng.permuted(pooled, axis=1)    # R relabelings
    gaps = shuffled[:, k:].mean(1) - shuffled[:, :k].mean(1)
    p = (1 + (np.abs(gaps) >= abs(observed)).sum()) / (R + 1)
    return p, gaps
```

Counting the observed labeling as one of the shuffles gives the *p-value*,

$$p = \frac{1 + \#\{r : |\text{gap}_r| \geq |\text{gap}_{\text{obs}}|\}}{R + 1},$$

the probability, computed under H_0, of a gap at least as extreme as the observed one. A small p says that the observed gap would be rare if the labels did not matter. This p-value is *two-sided*: a gap in either direction counts. A one-sided test that asks only whether B is better counts only the shuffled gaps at least as large as the observed one.

Before seeing the data we fix a *significance level* α, conventionally $\alpha = 0.05$, and we reject H_0 when $p \leq \alpha$. Here is one simulated experiment with the numbers above.

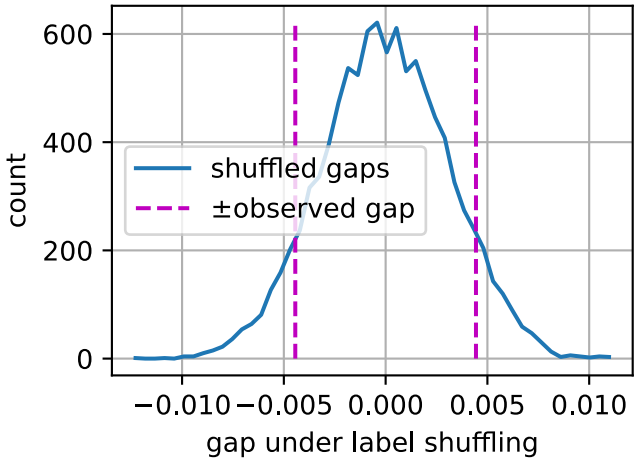
```
k = 20                                      # training seeds per model
acc_a = rng.normal(0.850, 0.010, k)         # per-seed accuracy, model A
acc_b = rng.normal(0.858, 0.010, k)         # model B: a true gap of 0.008
p, gaps = perm_p(acc_a, acc_b, rng, R=10000)
observed = acc_b.mean() - acc_a.mean()
print(f'observed gap {observed:.4f}, permutation p-value {p:.3f}')
```

```
observed gap 0.0044, permutation p-value 0.158
```

The printed gap scatters around the true 0.008 from run to run. In most runs p falls below 0.05 and the test rejects H_0, but in a sizable minority it fails to reject, as it may have done above: with this much seed noise, 20 seeds are not always enough. The subsections below measure how often this happens, and show that it is a property of the experiment's design rather than a flaw of the test.

A histogram of the shuffled gaps shows the null distribution, with the observed gap and its mirror image.

```
counts, edges = np.histogram(gaps, bins=50)
centers = (edges[:-1] + edges[1:]) / 2
top = counts.max()
d2l.plot([centers, [observed] * 2, [-observed] * 2],
         [counts, [0, top], [0, top]], 'gap under label shuffling', 'count',
         legend=['shuffled gaps', '±observed gap'],
         fmts=('-', 'm--', 'm--'))
```



The shuffled gaps form a bell around zero, although the test assumed no shape for the data, and the observed gap usually lies in a tail. The same recipe works for any statistic that compares two groups: a difference of medians, of error rates or of losses. (For a difference of means, the classical counterpart in closed form is Welch's t-test.) The recipe is the one every hypothesis test follows: state H_0 and H_A, fix α and a sample size large enough to detect the effect that matters, and only then collect the data, compute the p-value, and reject H_0 if $p \le \alpha$.

36.5.2.3 The z-Test

Shuffling builds the null distribution by simulation. When the data are Gaussian with known variance, the null distribution has a closed form, and so do the p-value and the power. Let $x_1, \dots, x_n$ be i.i.d. $\mathcal{N}(\mu, \sigma^2)$ with σ known, and test $H_0 : \mu = 0$ with the standardized sample mean

$$T = \frac{\sqrt{n}\,\bar{x}}{\sigma}.$$

Under H_0, $\bar{x} \sim \mathcal{N}(0, \sigma^2/n)$, so T is exactly standard normal. Let Φ be the standard normal c.d.f. and z_q its quantile, $\Phi(z_q) = q$, so that $z_{0.975} \approx 1.96$. The two-sided p-value is $2(1 - \Phi(|T|))$, and $p \le \alpha$ holds exactly when $|T| \ge z_{1-\alpha/2}$. At $\alpha = 0.05$ the test rejects when the sample mean lies more than 1.96 standard errors from 0. Figure 36.21 shows the two rejection tails.

The model comparison has a two-sample version of this test. With k seeds per model and per-seed standard deviation σ, the gap $\bar{x}_B - \bar{x}_A$ has variance $2\sigma^2/k$, and $T = (\bar{x}_B - \bar{x}_A)/(\sigma\sqrt{2/k})$ is standard normal under H_0. With $\sigma = 0.01$ and $k = 20$ the gap has standard error 0.0032, so a gap of 0.008 lies 2.5 standard errors from zero: beyond 1.96, but not by much.

36.5.2.4 Two Kinds of Error and Power

A test can go wrong in two ways, and a good design controls both. A *type I error*, or false positive, rejects a true H_0: we declare an improvement that is not there. A *type II error*, or

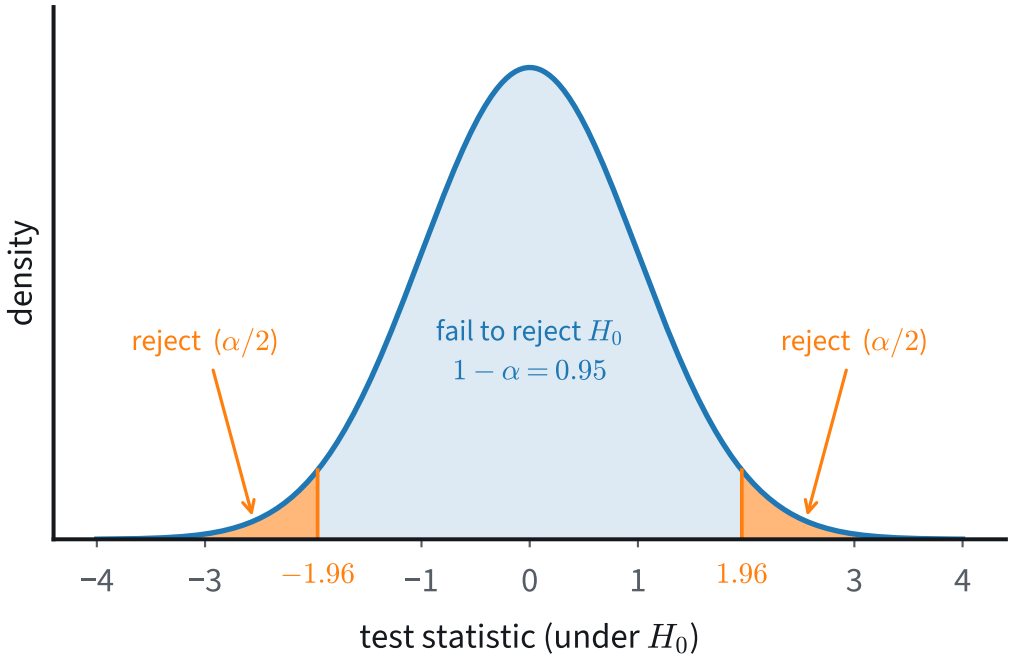


Figure 36.21: The rejection region of the two-sided z-test at $\alpha = 0.05$. Under H_0 the statistic is standard normal; the central region holds probability $1 - \alpha = 0.95$, and the two tails beyond ± 1.96 hold α together. A statistic in the tails would be rare under H_0, so we reject H_0.

false negative, fails to reject a false H_0: we miss a real improvement. Their rates are

$$\alpha = P(\text{reject } H_0 \mid H_0 \text{ true}), \qquad \beta = P(\text{fail to reject } H_0 \mid H_0 \text{ false}),$$

where β is evaluated at a given alternative, such as a true gap of 0.8 points. The *power* $1-\beta$ is the probability of detecting that effect. Figure 36.22 arranges the four outcomes.

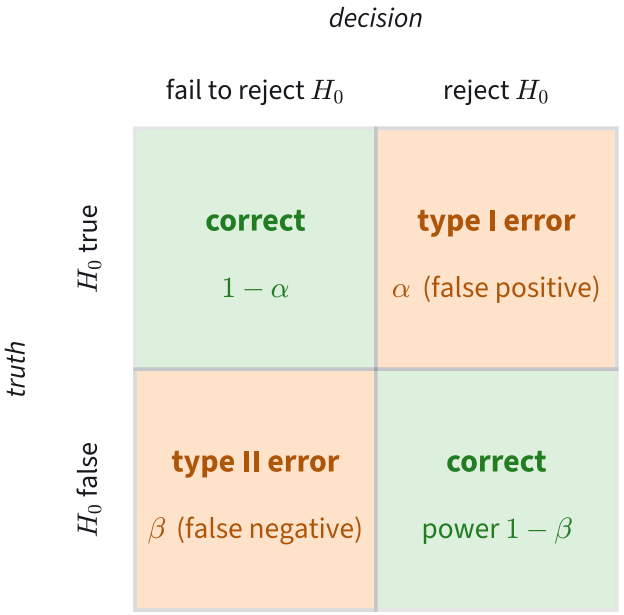


Figure 36.22: The 2×2 decision matrix of a hypothesis test: correct decisions on the diagonal, the type I error (rate α) and the type II error (rate β) off the diagonal, and the power $1-\beta$ in the correct-rejection cell.

The rule "reject when $p \le \alpha$" has a type I error rate of α because under H_0 the p-value is uniformly distributed. The corrections for running many tests, below, cannot be understood without this fact.

Proposition (p-values are uniform under the null). *Let the test statistic T have a continuous c.d.f. F under H_0, and let $p = 1 - F(T)$ (one-sided) or $p = 2\min\{F(T), 1-F(T)\}$ (two-sided). Then $P_{H_0}(p \le \alpha) = \alpha$ for every $\alpha \in [0,1]$.*

Proof. First, $U = F(T)$ is uniform on $[0,1]$. For $u \in (0,1)$, let t_u be the largest t with $F(t) = u$; it exists because F is continuous and tends to 0 and 1. Since F is nondecreasing, $F(T) \le u$ holds exactly when $T \le t_u$, so $P(U \le u) = F(t_u) = u$. For the one-sided p-value, $P(1-U \le \alpha) = P(U \ge 1-\alpha) = \alpha$. For the two-sided one, $2\min\{U, 1-U\} \le \alpha$ means $U \le \alpha/2$ or $U \ge 1-\alpha/2$, and these two disjoint events have probability $\alpha/2$ each. ■

For the permutation p-value the argument is a counting one. Under H_0 the observed labeling and the R shuffles are exchangeable, so the observed gap is equally likely to hold any of the $R+1$ ranks among the $R+1$ gaps. The p-value is at most α only when the observed

gap is among the $\lfloor \alpha(R+1) \rfloor$ largest, so $P_{H_0}(p \le \alpha) \le \alpha$ (ties only make p larger). The $+1$ in the numerator secures this guarantee. Uniformity is what a significance level means: a test at level α rejects a true null in a fraction α of experiments.

What replaces the uniform distribution when H_0 is false? We run the two-sample z-test of the model comparison on 2,000 simulated experiments with 20 seeds per model, once without a true gap and once with the gap of 0.008, sort the p-values into ten bins, and count those at or below 0.05.

```
se = 0.010 * np.sqrt(2 / k)                 # se of the gap, k seeds per model
for gap in (0.0, 0.008):
    acc_a = rng.normal(0.850, 0.010, (2000, k))
    acc_b = rng.normal(0.850 + gap, 0.010, (2000, k))
    T = (acc_b.mean(axis=1) - acc_a.mean(axis=1)) / se
    p = 2 * norm.sf(np.abs(T))               # two-sided z-test p-values
    hist = np.histogram(p, bins=10, range=(0, 1))[0] / 2000
    print(f'gap {gap:.3f}:', ' '.join(f'{h:.2f}' for h in hist),
          f'  p <= 0.05: {np.mean(p <= 0.05):.2f}')
```

```
gap 0.000: 0.10 0.10 0.11 0.10 0.09 0.10 0.10 0.10 0.10 0.10   p <= 0.05: 0.05
gap 0.008: 0.82 0.08 0.04 0.02 0.01 0.01 0.01 0.01 0.01 0.01   p <= 0.05: 0.72
```

Under H_0 each of the ten bins holds about a tenth of the p-values, and about 5% of them fall at or below 0.05. Under the gap of 0.008 about 80% fall below 0.1 and 70% to 74% at or below 0.05; this pile-up near zero is the power of the test, which the formula below puts at 0.72.

The p-value is easy to misread (Wasserstein and Lazar 2016). It is the probability of data at least this extreme, computed under H_0. It is not the probability that H_0 is true, which would require a prior and Bayes' rule. A large p-value does not confirm H_0 either; it means only that this experiment did not detect an effect, possibly because it could not.

How often does the model comparison detect its true gap? For the z-test the power has a closed form. If the true mean is $\mu \neq 0$, then T is normal with mean $\sqrt{n}\,\mu/\sigma$ and variance 1. Write $\delta = |\mu|/\sigma$ for the *standardized effect*, the effect measured in units of the noise. The test rejects when $|T| \ge z_{1-\alpha/2}$, which happens with probability

$$1 - \beta \approx \Phi(\delta\sqrt{n} - z_{1-\alpha/2}),$$

where we have dropped the far tail on the wrong side, of probability $\Phi(-\delta\sqrt{n} - z_{1-\alpha/2})$. The power grows with the effect and with $\sqrt{n}$. For $\delta = 1$ and $n = 8$ it is $\Phi(2.83 - 1.96) \approx 0.81$.

In the two-sample comparison the statistic has mean $\delta\sqrt{k/2}$ with $\delta = 0.008/0.010 = 0.8$, so the power with k seeds per model is $\Phi(0.8\sqrt{k/2} - 1.96)$: 0.72 with 20 seeds and 0.24 with 5. The formula belongs to the z-test, which is told σ. The permutation test is told nothing, so we measure its power on 400 fresh experiments for each design.

```
def power(k, gap, rng, num_experiments=400):
    """Fraction of fresh experiments in which the permutation test rejects."""
    rejections = 0
```

```
        for _ in range(num_experiments):
            acc_a = rng.normal(0.850, 0.010, k)
            acc_b = rng.normal(0.850 + gap, 0.010, k)
            rejections += perm_p(acc_a, acc_b, rng)[0] <= 0.05
        return rejections / num_experiments

    for k in (20, 5):
        print(f'{k:2d} seeds: rejects in {power(k, 0.008, rng):.2f} of runs, '
              f'formula {norm.cdf(0.8 * np.sqrt(k / 2) - 1.96):.2f}')
```

```
20 seeds: rejects in 0.70 of runs, formula 0.72
 5 seeds: rejects in 0.16 of runs, formula 0.24
```

It rejects in about 70% of the experiments with 20 seeds and about 20% with 5; with 400 experiments, each printed rate scatters by about two points from run to run. Both rates are a little below the formula on average: the permutation test in effect estimates the noise from the data, and that costs it a little power. The design we ran therefore misses a real 0.8-point improvement about three times in ten, and with five seeds about four times in five. When a test with power 0.2 fails to reject, we have learned almost nothing. A common target is power 0.8.

36.5.2.5 Sample Size

Power can be planned before any data are collected. Setting the power equal to $1-\beta$ gives $\delta\sqrt{n} - z_{1-\alpha/2} = z_{1-\beta}$, and solving for n gives the sample size needed to detect a standardized effect δ at level α with power $1-\beta$:

$$n = \frac{\left(z_{1-\alpha/2} + z_{1-\beta}\right)^2}{\delta^2}. \tag{36.68}$$

The sample size grows like $1/\delta^2$: halving the effect we want to detect takes four times the data. At $\alpha = 0.05$ and power 0.8, the quantiles are $z_{0.975} \approx 1.960$ and $z_{0.8} \approx 0.842$, so $n \approx 7.85/\delta^2$. Figure 36.23 plots the power curves.

Comparing two groups of k observations each, with common standard deviation σ, doubles the variance of the statistic, $\mathrm{Var}(\bar{x}_B - \bar{x}_A) = 2\sigma^2/k$, and so doubles the sample size needed per group:

$$k = \frac{2\left(z_{1-\alpha/2} + z_{1-\beta}\right)^2}{\delta^2}, \qquad \delta = \frac{|\mu_B - \mu_A|}{\sigma}. \tag{36.69}$$

For the model comparison, $\delta = 0.8$ gives $k = 2 \cdot 7.85/0.64 \approx 25$ seeds per model for power 0.8. Twenty seeds fall a little short.

36.5.2.6 Paired Comparisons

We return to the two classifiers of the introduction, with 912 and 921 correct answers. They answered the *same* 1,000 questions, and some questions are hard for both. Treating the two accuracies as independent samples leaves this shared difficulty in the noise. A *paired* comparison works with the differences per question instead. Let $d_i \in \{-1, 0, 1\}$ be model

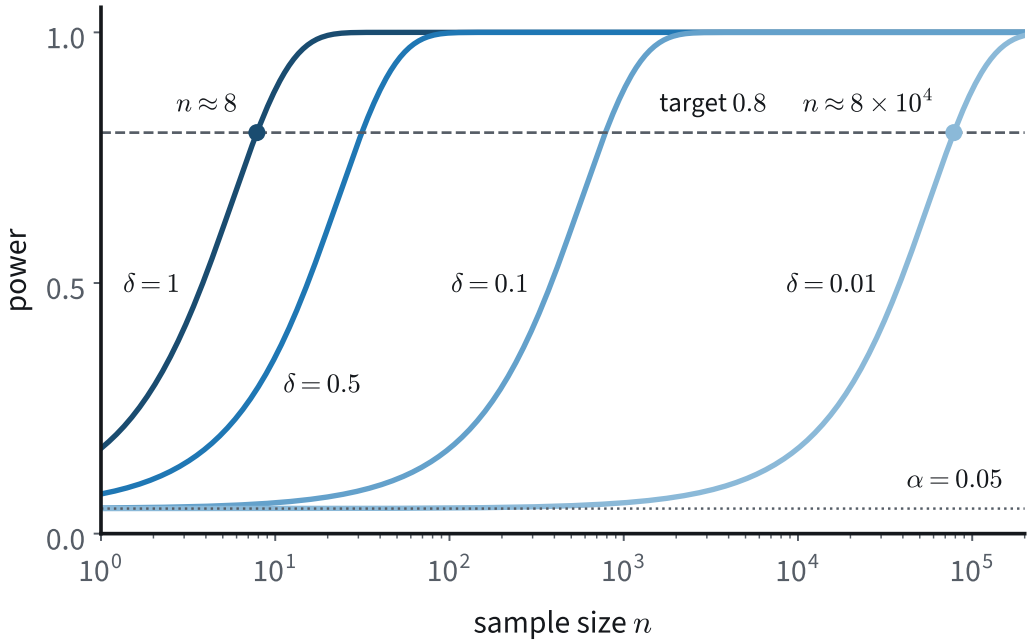


Figure 36.23: Power of the two-sided z-test at $\alpha = 0.05$ as a function of the sample size n, one curve per standardized effect δ. Small-effect curves start near α (dotted line), where the test rejects only by chance, and every curve climbs toward 1. The dashed line marks the target 0.8, reached at a sample size proportional to $1/\delta^2$: about 8 observations for $\delta = 1$ and about 78,500 for $\delta = 0.01$.

B's correctness minus model A's on question i. On most questions the models agree, and $d_i = 0$. Suppose they disagree on 41 questions: B alone is right on 25 and A alone on 16, a net of $+9$, which is the gap of 0.9 points. Under the null hypothesis that the two models are interchangeable, each disagreement is equally likely to go either way, so the right shuffle flips the sign of each d_i at random. This is the *sign-flip permutation test*.

```
d = np.r_[np.ones(25), -np.ones(16), np.zeros(959)]   # B only, A only, agree
nonzero = d[d != 0]                                    # agreements never change
signs = rng.choice([-1, 1], size=(100000, len(nonzero)))
flipped = (signs * nonzero).sum(axis=1) / len(d)       # gaps under H_0
p = (1 + (np.abs(flipped) >= abs(d.mean())).sum()) / (len(flipped) + 1)
print(f'gap {d.mean():.3f}, sign-flip p-value {p:.2f}')
```

```
gap 0.009, sign-flip p-value 0.21
```

Every run gives $p \approx 0.21$. (With the exact binomial count in place of random sign flips, this is McNemar's test.) Treated as two independent samples, the same counts give a gap of 0.7 standard errors and $p \approx 0.47$. Pairing halves the standard error of the gap, from 0.012 to $\sqrt{41}/1000 \approx 0.006$, because the 959 questions on which the models agree no longer contribute noise. The evidence is the split of 25 against 16 among the 41 disagreements, and it is still not enough: a nine-question lead on a thousand questions does not show that B is better.

Pairing matters most when an evaluation is planned. Consider two models with accuracies of 90% and 91% that disagree on 4% of the questions, with B alone right on 2.5% and A alone on 1.5%. Then d_i has mean 0.01 and variance $\mathbb{E}[d_i^2] - 0.01^2 = 0.04 - 0.0001 = 0.0399$, a standardized effect of $\delta = 0.01/\sqrt{0.0399} \approx 0.050$. By Equation 36.68, a paired test detects the gap with power 0.8 on about $7.85 \cdot 0.0399/0.01^2 \approx 3{,}100$ questions. Two independent test sets, one per model, need about $7.85 \cdot 0.1719/0.01^2 \approx 13{,}500$ questions each, because the unpaired gap has variance $0.9 \cdot 0.1 + 0.91 \cdot 0.09 = 0.1719$ per pair of questions.

Pairing cuts the size of each test set by a factor of about four here, and the number of questions that must be labeled by a factor of about nine; the saving grows when the models agree more often. Benchmark comparisons should therefore use one test set and be analyzed per question. The same holds for seeds: when both models are trained with the same seeds, the per-seed differences are the units whose signs we flip (exercise 8).

36.5.2.7 Multiple Testing

The guarantee $P(\text{reject} \mid H_0) = \alpha$ holds for one test fixed in advance, and a hyperparameter sweep runs many. Compare 20 configurations with a baseline, none of them actually better, each at $\alpha = 0.05$. If the tests are independent, the chance that at least one of them "wins" is $1 - 0.95^{20} \approx 0.64$. Reporting only the winners, or trying analyses until one clears $p \le \alpha$ (*p-hacking*), turns noise into apparent findings. The simplest remedy is the *Bonferroni correction* (Bonferroni 1936): test each of the m hypotheses at level α/m. If A_j is the event that test j rejects its true null, the union bound gives

$$P\Big(\bigcup_{j=1}^{m} A_j\Big) \le \sum_{j=1}^{m} P(A_j) = m \cdot \frac{\alpha}{m} = \alpha,$$

so the chance of any false positive stays at most α, with no assumption about how the tests depend on each other. For 20 independent tests at $0.05/20$, the chance is $1-(1-0.0025)^{20} \approx 0.049$.

When m runs into the hundreds, the Bonferroni threshold is so strict that real effects are missed. Large-scale screening then controls the *false discovery rate* instead, the expected fraction of false positives among the rejected hypotheses. The procedure of Benjamini and Hochberg (Benjamini and Hochberg 1995) controls it at level α when the p-values are independent or positively dependent; exercise 3 states the rule and tests it.

36.5.3 Confidence Intervals

A p-value answers a yes-or-no question about one value of the parameter. Often we want the complementary view: the whole range of values that the data allow, reported as an error bar around the estimate.

36.5.3.1 Definition and Interpretation

A *confidence interval* is built by a procedure that captures the fixed unknown parameter in a stated fraction of repeated experiments (Neyman 1937). Formally, a $1-\alpha$ confidence interval is an interval C_n computed from the data such that

$$P_\theta(C_n \ni \theta) \geq 1-\alpha \quad \text{for every } \theta. \tag{36.70}$$

We write $C_n \ni \theta$ to stress where the randomness lives: θ is fixed, and the interval changes from data set to data set. The guarantee describes the procedure. Over many data sets, at least 95% of the 95% intervals contain θ; a given interval either contains it or does not, and nothing in the interval tells which. Figure 36.24 shows 100 intervals from 100 independent data sets. Many intervals used in practice meet the guarantee only in the limit: an *asymptotic* $1-\alpha$ interval satisfies $\liminf_{n\to\infty} P_\theta(C_n \ni \theta) \geq 1-\alpha$ for every θ.

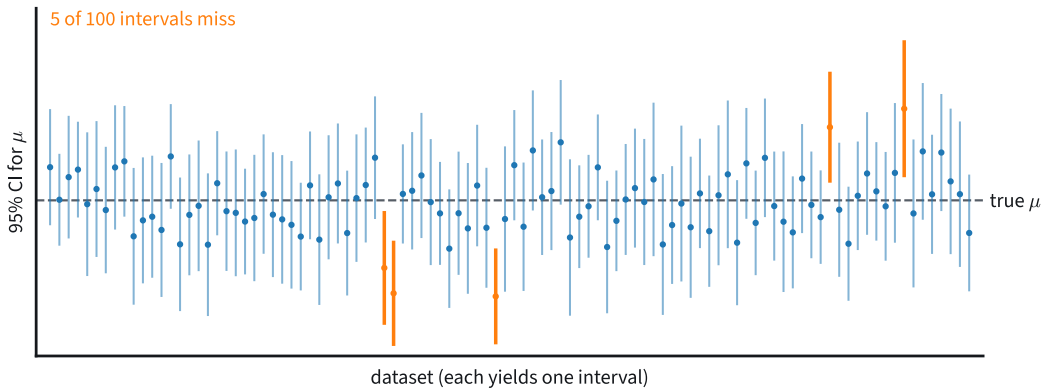


Figure 36.24: One hundred 95% confidence intervals, each computed from its own independent data set, against the fixed true mean (dashed line). Most intervals contain it; the orange ones miss. The 95% describes the long-run hit rate of the procedure; a single interval carries no mark of whether it contains the truth.

36.5.3.2 The Gaussian Interval

We first build an interval in the simplest case, where every step is exact. Take $x_1, \dots, x_n$ i.i.d. $\mathcal{N}(\mu, \sigma^2)$ with σ known. Then $\bar{x} \sim \mathcal{N}(\mu, \sigma^2/n)$, so with $h = z_{1-\alpha/2}\,\sigma/\sqrt{n}$ the estimate lands within h of μ with probability exactly $1-\alpha$. Now read the same event as a statement about μ:

$$|\bar{x} - \mu| \le h \quad \Longleftrightarrow \quad \mu \in [\bar{x} - h,\ \bar{x} + h],$$

so $P_\mu(\mu \in [\bar{x} - h, \bar{x} + h]) = 1 - \alpha$ for every μ. This change of viewpoint is the whole construction. At $\alpha = 0.05$ the interval is $\bar{x} \pm 1.96\,\sigma/\sqrt{n}$.

When σ is unknown, we replace it by the sample standard deviation s of Equation 36.62. The studentized statistic $T = (\bar{x} - \mu)/(s/\sqrt{n})$ is then no longer normal. For Gaussian data it has Student's t distribution with $n-1$ degrees of freedom (Student (Gosset 1908), whose tails are heavier because s fluctuates too. We take this fact on faith; exercise 7 builds the same distribution as a Gaussian with a random precision. Inverting as before, with the quantile $t_{n-1,1-\alpha/2}$ of the t distribution in place of $z_{1-\alpha/2}$, gives the interval

$$\left[\bar{x} - t_{n-1,1-\alpha/2}\,\frac{s}{\sqrt{n}},\ \bar{x} + t_{n-1,1-\alpha/2}\,\frac{s}{\sqrt{n}}\right], \tag{36.71}$$

whose coverage is exactly $1-\alpha$ for Gaussian data at every $n \ge 2$. The quantile matters at small n: $t_{n-1,0.975}$ is 12.71 at $n = 2$, 2.776 at $n = 5$ and 1.984 at $n = 100$, and it approaches 1.96 as n grows. The interval used in practice replaces the quantile by its limit,

$$\left[\bar{x} - 1.96\,\frac{s}{\sqrt{n}},\ \bar{x} + 1.96\,\frac{s}{\sqrt{n}}\right], \tag{36.72}$$

an asymptotic 95% interval that is too short at every finite n: for Gaussian data its coverage is $P(|t_{n-1}| \le 1.96)$, which is 0.947 at $n = 100$ but only 0.878 at $n = 5$.

For data that are not Gaussian, the central limit theorem (Section 36.3.2.3) states that $\sqrt{n}(\bar{x} - \mu)/\sigma$ approaches a standard normal distribution whenever the variance is finite, and s approaches σ by the law of large numbers. So Equation 36.72 is an asymptotic 95% interval for the mean of any distribution with finite variance. How large n must be depends on the distribution. Skewed data need more observations (exercise 4), and an accuracy near 100% needs far more, because then many test sets report a perfect score (exercise 5). In every case the half-width shrinks like $1/\sqrt{n}$, so halving an error bar takes four times the data.

An accuracy is the mean of 0/1 indicators with $s^2 \approx \hat{\theta}(1-\hat{\theta})$, so the same interval applies to it as $\hat{\theta} \pm 1.96\sqrt{\hat{\theta}(1-\hat{\theta})/n}$, often called the *Wald interval*. We compute it for our classifier and for the chapter's coin, 9 heads in 13 flips.

```
for n_correct, n in ((912, 1000), (9, 13)):
    theta_hat = n_correct / n
    half = 1.96 * np.sqrt(theta_hat * (1 - theta_hat) / n)
    print(f'{n_correct:3d} of {n:4d}: '
```

```
            f'({theta_hat - half:.3f}, {theta_hat + half:.3f})')
```

```
912 of 1000: (0.894, 0.930)
  9 of   13: (0.441, 0.943)
```

The classifier's accuracy is $91.2\% \pm 1.8$ points. The coin's interval $(0.44, 0.94)$ contains $\frac{1}{2}$, so 13 flips cannot show that the coin is biased. Narrowing its interval to ± 0.05 would take about $1.96^2 \cdot 0.69 \cdot 0.31 / 0.05^2 \approx 330$ flips.

A Bayesian *credible interval* for the same coin answers a different question. It treats θ as random with a prior distribution, and it reports an interval that holds posterior probability $1-\alpha$ given the observed data. A uniform prior gives the posterior $\text{Beta}(10, 5)$ (Section 36.4), whose central 95% interval, between the 2.5% and 97.5% posterior quantiles, is $(0.42, 0.87)$. The two intervals are similar here, but they make different statements. The credible interval says that, given the prior, θ lies in it with probability 0.95. The confidence interval says that the procedure that produced it contains the true θ in at least 95% of experiments, whatever θ is. The Wald interval, moreover, meets that guarantee only approximately, for large n: at 13 flips its coverage is about 91% at $\theta = \frac{1}{2}$ and below 50% at $\theta = 0.05$.

Coverage is a statement about many data sets, so it can be measured. We draw 1,000 data sets of $n = 100$ standard normal observations, build the interval Equation 36.72 from each, and count how many contain the true mean 0.

```
mu, n, trials = 0.0, 100, 1000
data = rng.normal(mu, 1.0, (trials, n))          # one row = one data set
mu_hat = data.mean(axis=1)
se = data.std(axis=1, ddof=1) / np.sqrt(n)
hits = np.abs(mu_hat - mu) <= 1.96 * se          # does the interval contain mu?
print(f'first interval ({mu_hat[0] - 1.96 * se[0]:.3f}, '
      f'{mu_hat[0] + 1.96 * se[0]:.3f}) contains 0: {hits[0]}')
print(f'{hits.sum()} of {trials} intervals contain the true mean '
      f'(nominal 950, exact 947)')
```

```
first interval (-0.299, 0.098) contains 0: True
951 of 1000 intervals contain the true mean (nominal 950, exact 947)
```

The first interval contains 0 in about 95% of runs. The count lands near 947, typically within ± 14, rather than near the nominal 950. The shortfall is systematic. At $n = 100$ the exact quantile $t_{99,0.975} = 1.984$ is slightly larger than 1.96, and the coverage of the 1.96 interval is $P(|t_{99}| \le 1.96) = 0.947$. The spread of ± 14 is two binomial standard deviations of the count, $2\sqrt{1000 \cdot 0.947 \cdot 0.053} \approx 14$. The same construction gives the error bar on a mean accuracy over k training seeds: the seed-to-seed standard deviation divided by $\sqrt{k}$, times the appropriate quantile.

36.5.3.3 Tests and Intervals

The number 1.96 has now appeared twice, as the rejection threshold of the z-test and as the multiplier of the interval's half-width. The two uses are one fact seen from two sides.

For Gaussian data with known σ, the z-test of $H_0 : \mu = \mu_0$ keeps H_0 when $|T| \le z_{1-\alpha/2}$, with $T = \sqrt{n}(\bar{x} - \mu_0)/\sigma$. That is the condition $|\bar{x} - \mu_0| \le h$ with the half-width $h = z_{1-\alpha/2}\,\sigma/\sqrt{n}$ from above, and it says that μ_0 lies in $[\bar{x} - h, \bar{x} + h]$. So *a 95% confidence*

interval is the set of null values that a 5% test keeps. Coverage follows from the level of the test: for every μ, $P_\mu(\mu \in [\bar{x} - h, \bar{x} + h]) = 1 - P_\mu(\text{the test of } H_0 : \mu \text{ rejects}) = 1 - \alpha$.

The same construction works for any family of tests. Suppose that for each θ_0 we have a level-α test of $H_0 : \theta = \theta_0$. The set of values θ_0 that it does not reject contains the true θ with probability at least $1 - \alpha$, by the same one-line argument, so it is a $1 - \alpha$ confidence set. This is how intervals are built when no formula inverts the test. For 29 correct answers out of 30, the Wald interval reaches past an accuracy of 1, while inverting an exact binomial test gives an interval inside $[0, 1]$ (exercise 5).

The duality also says what to report for a comparison: an interval for the difference, such as $\bar{x}_B - \bar{x}_A \pm 1.96\,\sigma\sqrt{2/k}$ for two models over k seeds. It excludes 0 exactly when the test rejects, and it shows the size of the effect as well.

36.5.3.4 The Delta Method

We often report not $\hat{\theta}$ itself but a function of it: the log-odds of an accuracy, the ratio of two losses, a relative improvement. Its error bar follows from the error bar of $\hat{\theta}$. If g is differentiable at θ, the first-order Taylor expansion

$$g(\hat{\theta}) \approx g(\theta) + g'(\theta)\,(\hat{\theta} - \theta)$$

says that near θ the map g stretches deviations by the factor $|g'(\theta)|$. The standard error therefore propagates as

$$\operatorname{se}(g(\hat{\theta})) \approx \left|g'(\hat{\theta})\right| \operatorname{se}(\hat{\theta}), \tag{36.73}$$

where the unknown θ in g' has been replaced by its estimate. This rule is the *delta method* (Wasserman 2013). For an accuracy $\hat{\theta} = 0.90$ on $n = 1{,}000$ questions, $\operatorname{se}(\hat{\theta}) = \sqrt{0.9 \cdot 0.1/1000} \approx 0.0095$. Its log-odds $g(\hat{\theta}) = \log \frac{\hat{\theta}}{1-\hat{\theta}} \approx 2.20$ has slope $g'(\hat{\theta}) = 1/(\hat{\theta}(1-\hat{\theta})) \approx 11.1$, so the log-odds has a standard error of about $11.1 \cdot 0.0095 \approx 0.105$.

For a vector of estimates with covariance matrix Σ the same expansion gives $\operatorname{Var}(g(\hat{\theta})) \approx \nabla g^\top \Sigma\, \nabla g$, and the off-diagonal entries matter: the ratio of two losses measured on the same test set has a correlated numerator and denominator.

The rule needs $g'(\theta) \neq 0$. Where the slope vanishes, as for $g(\theta) = (\theta - \tfrac{1}{2})^2$ at $\theta = \tfrac{1}{2}$, the first-order term disappears and $g(\hat{\theta})$ fluctuates on the scale $1/n$ rather than $1/\sqrt{n}$. The rule also needs $\hat{\theta}$ to stay where the linear approximation holds, which fails for an accuracy near 100% (exercise 5). When no derivative is convenient at all, we can resample instead.

36.5.4 The Bootstrap

Every standard error so far came from a formula. A median, an AUC or a BLEU score varies from test set to test set just as an accuracy does, but it has no convenient formula. This part simulates the sampling distribution instead, first checks the simulation where the answer is known, and then shows where it breaks.

36.5.4.1 The Plug-in Principle

If we knew the distribution F that generated the data, we could simulate the sampling distribution of any statistic: draw many fresh data sets from F and recompute the statistic on each. The *bootstrap* (Efron 1979) makes one substitution. It replaces the unknown F by the *empirical distribution* $\hat{F}_n$, which puts mass $1/n$ on each observed point. Drawing n points from $\hat{F}_n$ is the same as resampling the data n times with replacement. The procedure has three steps:

1. Draw n points from the data with replacement to form a *bootstrap resample*; some points appear several times, others not at all.
2. Compute the statistic $\hat{\theta}^*$ on the resample.
3. Repeat R times to obtain $\hat{\theta}^*_1, \dots, \hat{\theta}^*_R$.

The standard deviation of the replicates estimates the standard error of $\hat{\theta}$, and their $\alpha/2$ and $1-\alpha/2$ quantiles form the *percentile interval*. Figure 36.25 shows the procedure. The bootstrap assumes that the observations are i.i.d. and that the statistic changes smoothly when the data change a little.

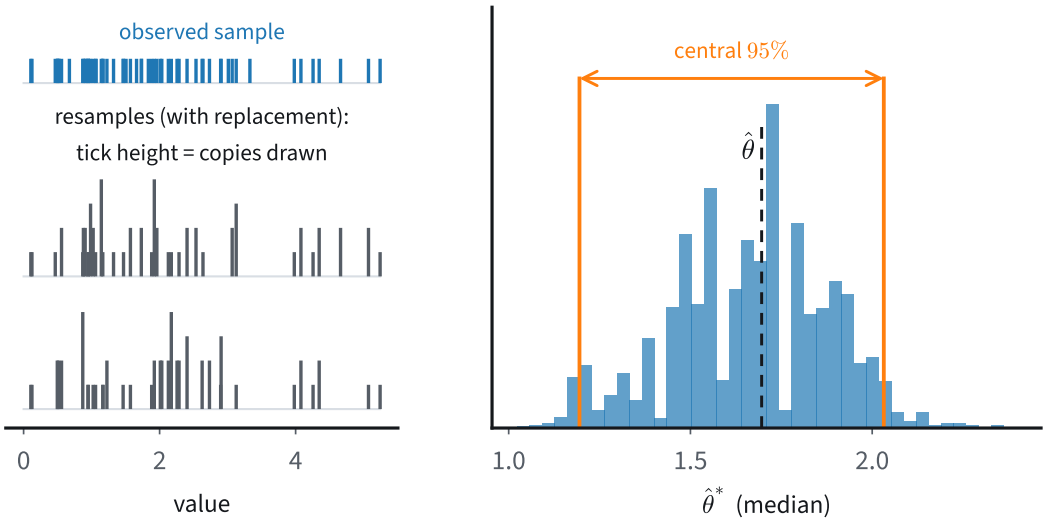


Figure 36.25: The ordinary bootstrap. From one observed sample (top left), draw many resamples of the same size with replacement (bottom left; a tick's height counts how many copies of that observation were drawn) and recompute the statistic, here the median. The replicate histogram (right) estimates the sampling distribution around the observed estimate $\hat{\theta}$ (dashed line); its standard deviation estimates the standard error, and its central quantiles give a percentile interval. The histogram is jagged because a resample's median can take only a few of the observed values; this mild non-smoothness becomes a failure for the maximum.

36.5.4.2 Bootstrap Standard Errors

Before trusting the bootstrap where no formula exists, we check it where one does. Our test set has 912 correct and 88 incorrect answers. Resampling its 1,000 indicators and recom-

puting the accuracy should reproduce the standard error 0.0090.

```
correct = np.r_[np.ones(912), np.zeros(88)]          # the opening's test set
idx = rng.integers(0, len(correct), (5000, len(correct)))  # 5,000 resamples
boot = correct[idx].mean(axis=1)                     # accuracy of each resample
print(f'bootstrap se {boot.std(ddof=1):.4f}, '
      f'formula {np.sqrt(0.912 * 0.088 / 1000):.4f}')
```

```
bootstrap se 0.0090, formula 0.0090
```

The two agree to within a few percent. Now a statistic without a convenient formula: the median of 200 draws from the exponential distribution with mean 1, a skewed distribution whose median $\log 2 \approx 0.69$ lies below its mean. A formula for the median's standard error does exist, $1/(2f(m)\sqrt{n})$ with $f(m)$ the density at the median m (exercise 9), but it asks for a density that in practice we do not know. Here we do know it, $f(m) = e^{-\log 2} = \frac{1}{2}$, so the bootstrap can be held to a prediction: $1/\sqrt{200} \approx 0.071$.

```
data = rng.exponential(1.0, 200)    # its median's se formula needs the unknown density
theta_hat = np.median(data)
idx = rng.integers(0, len(data), (10000, len(data)))  # resample with replacement
boot = np.median(data[idx], axis=1)                   # one median per resample
lo, hi = np.percentile(boot, [2.5, 97.5])
print(f'median {theta_hat:.3f}, bootstrap se {boot.std(ddof=1):.3f}, '
      f'95% percentile interval ({lo:.3f}, {hi:.3f})')
```

```
median 0.741, bootstrap se 0.090, 95% percentile interval (0.636, 0.993)
```

The bootstrap never asked for the density, and its standard error is typically near 0.07. The printed median scatters around $\log 2$, usually by less than 0.15, and the bootstrap standard error itself moves between about 0.05 and 0.11 from one data set to the next. A resample's median can only be one of the few observed values near the sample median, so the bootstrap distribution is lumpy and depends on the random spacing of those values. The percentile interval is correspondingly rough, and in some runs it only just contains $\log 2$. The median is still regular enough for the bootstrap, because the density at the median is positive.

36.5.4.3 Failures of the Bootstrap

The bootstrap fails when the statistic does not change smoothly with the data. The sample maximum is the standard example. Let $x_1, \dots, x_n$ be i.i.d. uniform on $[0, \theta]$ and $\hat{\theta} = \max_i x_i$. A resample has the same maximum as the data unless it misses the largest point in all n draws, so

$$P(\hat{\theta}^* = \hat{\theta}) = 1 - \left(1 - \frac{1}{n}\right)^n \;\to\; 1 - \frac{1}{e} \approx 0.63.$$

The bootstrap distribution therefore puts 63% of its mass on a single value, while the true sampling distribution of $\hat{\theta}$ is continuous. Since no resample's maximum exceeds $\hat{\theta} < \theta$, the percentile interval never contains θ. Exercise 6 measures whether reflecting the interval about the estimate repairs it.

The bootstrap also fails when it resamples units that are not independent, and the questions of an evaluation set often are not. A reading-comprehension benchmark may hold G passages with m questions each, and a model that misreads a passage tends to miss several of its questions. If the correctness indicators of two questions on the same passage have correlation ρ, and different passages are independent, the accuracy has variance

$$\operatorname{Var}(\hat{\theta}) = \frac{\theta(1-\theta)}{Gm}(1 + (m-1)\rho),$$

larger by the factor $1 + (m-1)\rho$ than for Gm independent questions. A bootstrap that resamples individual questions assumes independence, so it understates the standard error by the factor $\sqrt{1 + (m-1)\rho}$, which is $\sqrt{2.9} \approx 1.7$ at $m = 20$ and a modest $\rho = 0.1$ (Miller 2024). Resampling whole passages restores the right answer (exercise 12).

Two questions remain open. First, nothing here said which estimator to use, or how precisely the data can pin down a parameter at all; Section 36.6 answers both with the likelihood and its curvature. Second, every guarantee here holds for one estimator or one test fixed before the data were seen. A model chosen because it scored best on a validation set is not fixed in advance, since choosing the best of many configurations is multiple testing in another form. Section 36.10 asks how large a sample must be for a guarantee to hold for every model in a class at once.

36.5.5 Summary

An estimate computed from random data is itself random, and its sampling distribution over the data sets we did not draw decides how far it can be trusted. Its mean squared error is the squared bias plus the variance, so a biased estimator such as the shrunk mean can beat the unbiased sample mean. A Monte Carlo estimate of an expectation is a sample mean of computed samples, so its root-mean-square error is $\sigma/\sqrt{n}$ in every dimension, where the error of a grid worsens with each dimension added. A test rejects when the observed difference would be rare under the null hypothesis; because a p-value is uniform under the null, it errs at rate α, and its power is set by the standardized effect times $\sqrt{n}$, which pairing two models on the same questions raises. A confidence interval is the set of parameter values that a test keeps. The bootstrap simulates the sampling distribution by resampling the data; it works for smooth statistics of independent observations and fails for the maximum and for clustered test sets.

36.5.6 Exercises

1. code **Shrinking the sample mean.** The text found that the shrunk mean $c\,\bar{x}$ of $x_1, \dots, x_n$ i.i.d. $\mathcal{N}(\theta, \sigma^2)$ has the smallest MSE at $c^\star = \theta^2/(\theta^2 + \sigma^2/n)$.
    1. Extend the text's simulation of the sample mean, the shrunk mean and the median to a grid of factors $c \in [0, 1]$ at $\theta = 1$, $\sigma = 4$ and $n = 30$. Plot the squared bias, the variance and the MSE against c with `d2l.plot`, and compare the empirical minimizer with $c^\star$.
    2. Shrink toward a point a instead of 0, with the estimator $a + c(\bar{x} - a)$. Compute its MSE and its optimal factor. Which a would you choose, and what does the answer say about the choice of 0 in the text?
    3. $c^\star$ is not an estimator, because it contains θ. Plug in $\bar{x}$ for θ and s^2 for σ^2, giving $\hat{c} = \bar{x}^2/(\bar{x}^2 + s^2/n)$, and measure the MSE of $\hat{c}\,\bar{x}$ by simulation at

$\theta \in \{0, 1, 3, 10\}$. Where does it beat $\bar{x}$, and where does it lose? Exercise 13 shows that in three or more dimensions a similar substitution wins for every θ.

Hint for (c): at $\theta = 0$ the plug-in factor is itself noisy; look at its distribution.

2. **Reading a p-value.** A p-value is computed under H_0, so by itself it does not give the probability that H_0 is true.

   1. A two-sided z-test of $H_0 : \mu = 0$ on $n = 20$ observations reports $p = 0.5$. Compute the power of this test at $\alpha = 0.05$ against the standardized effect $\delta = 0.1$. Why is $p = 0.5$ then no evidence that H_0 is true?
   2. In a research program, 90% of the tested ideas have no effect. Each idea is tested once at $\alpha = 0.05$, with power 0.8 against its true effect. What fraction of the significant results are false positives? Recompute the fraction for power 0.2. This calculation is the core of the argument of Ioannidis (2005) . Which of its ingredients does a p-value not contain?

3. code **Multiple comparisons.** A hyperparameter sweep compares many configurations with one baseline, so it runs many tests at once.

   1. Twenty models, none of them better than the baseline, are each compared with it by an independent test at $\alpha = 0.05$. Compute the probability that at least one comparison is significant, and the same probability at the Bonferroni threshold $\alpha/20$. Show that the Bonferroni correction holds the probability of at least one false positive at or below α even when the tests are dependent.
   2. The Benjamini–Hochberg procedure (Benjamini and Hochberg 1995) sorts the m p-values, $p_{(1)} \le \cdots \le p_{(m)}$, finds the largest k with $p_{(k)} \le k\alpha/m$, and rejects the k hypotheses with the smallest p-values. Simulate a sweep of $m = 200$ configurations, of which $m_0 = 180$ have no effect. Draw the z-statistics from $\mathcal{N}(0, 1)$ for these and from $\mathcal{N}(3, 1)$ for the other 20, and compute the two-sided p-values `2 * norm.sf(np.abs(z))`. Over $1{,}000$ repetitions, report the average numbers of true and false discoveries and the average false-discovery proportion (the fraction of rejections that are false, or 0 without rejections) for the uncorrected threshold 0.05, for Bonferroni and for Benjamini–Hochberg. For independent tests, Benjamini–Hochberg guarantees an expected false-discovery proportion of at most $\alpha m_0/m$. Does the simulation agree?

4. code **Coverage at small n.** The interval Equation 36.72 uses the Gaussian quantile 1.96 in place of the t-quantile, and it assumes approximately Gaussian sample means.

   1. Run the coverage simulation of the text (1,000 intervals from standard normal data sets) with $n \in \{2, 5, 10, 30, 100\}$ and $100{,}000$ trials, and record the coverage of the interval. Then replace 1.96 by the 97.5% quantile of the t-distribution with $n-1$ degrees of freedom, which is 12.71, 2.776, 2.262, 2.045 and 1.984 for these n, and record the coverage again.
   2. Set $n = 2$ and replace 1.96 by 1.0, the 75% quantile of the t-distribution with one degree of freedom, so that the interval has 50% coverage. Compute the coverage separately for the intervals shorter and longer than the median width. Does the difference contradict Equation 36.70? Would you read a short interval as a sign of a precise estimate?
   3. Draw the data from the exponential distribution with mean 1 at $n = 10$, keeping the t-quantile 2.262. Measure the coverage, and count separately the intervals that lie entirely below and entirely above the true mean. Explain the asymmetry

from the shape of the distribution.

5. code **Delta method near a boundary.** By Equation 36.73, an accuracy $\hat{\theta} = 0.90$ measured on $n = 1{,}000$ test examples has a log-odds $\log \frac{\hat{\theta}}{1-\hat{\theta}}$ with standard error about 0.105.

   1. Simulate $100{,}000$ test sets by drawing the number of correct predictions from the binomial distribution with $n = 1{,}000$ and $\theta = 0.9$. Compare the standard deviation of the log-odds with the delta-method value.
   2. Repeat with $\theta = 0.999$. Estimate the fraction of test sets with $\hat{\theta} = 1$, compare it with its exact value, and state the log-odds of such a test set.
   3. Which assumption behind the first-order expansion fails at $\theta = 0.999$? Compute the expected number of incorrect answers, $n(1-\theta)$, for both values of θ, and explain why this count, rather than n alone, decides whether the delta method and the Wald interval can be trusted.
   4. A benchmark has 30 questions, and a model answers 29 of them correctly. Compute the Wald interval $\hat{\theta} \pm 1.96\sqrt{\hat{\theta}(1-\hat{\theta})/30}$ and note where it ends. Then invert a test: for each θ_0 on a fine grid of $[0.5, 1]$, compute the exact two-sided p-value of $H_0 : \theta = \theta_0$, $\min\{1,\, 2\min\{P_{\theta_0}(K \le 29),\, P_{\theta_0}(K \ge 29)\}\}$ with $K \sim \text{Binomial}(30, \theta_0)$, and collect the θ_0 that it keeps at $\alpha = 0.05$. Compare the two intervals, and their coverage at $\theta = 0.95$ over $100{,}000$ simulated benchmarks.

6. code **Bootstrap of the maximum.** Let $x_1, \dots, x_n$ be i.i.d. $U(0, \theta)$, let $\hat{\theta} = \max_i x_i$, and let $\hat{\theta}^*$ be the maximum of a bootstrap resample. The text showed that $P(\hat{\theta}^* = \hat{\theta}) \to 1 - 1/e$.

   1. Show that $n(\theta - \hat{\theta})/\theta$ converges in distribution to the exponential distribution with mean 1, whereas $n(\hat{\theta} - \hat{\theta}^*)/\hat{\theta}$, conditionally on the data, has an atom of probability about 0.63 at 0. Which part of the true sampling distribution does the bootstrap fail to reproduce?
   2. The basic bootstrap interval reflects the percentile quantiles $q_{\alpha/2}$ and $q_{1-\alpha/2}$ of the replicates about the estimate, $[2\hat{\theta} - q_{1-\alpha/2},\; 2\hat{\theta} - q_{\alpha/2}]$. Adapt the text's bootstrap of the median to the maximum, with $n = 100$ and $\theta = 1$, and estimate the coverage of the basic 95% interval over $1{,}000$ data sets with $R = 2{,}000$ resamples each.
   3. For one data set, plot the histogram of $n(\hat{\theta} - \hat{\theta}^*)$ next to the histogram of $n(\theta - \hat{\theta})$ over fresh data sets, and use the two pictures to explain the coverage you measured.

   Adapted from Larry Wasserman, All of Statistics, Chapter 8, Exercise 7.

7. code **Choosing the variance denominator.** ▣ Let $S = \sum_i (x_i - \bar{x})^2$ and consider the estimators S/c of σ^2. The plug-in variance s_0^2 and the unbiased estimator s^2 of Equation 36.62 are the cases $c = n$ and $c = n - 1$. For Gaussian data, S/σ^2 has the χ^2 distribution with $n - 1$ degrees of freedom, so $\text{Var}(S) = 2(n-1)\sigma^4$.

   1. Show that for Gaussian data the MSE of S/c is minimized at $c = n + 1$. Rank $c = n - 1$, n and $n + 1$ by their MSE at $n = 3$ and $\sigma^2 = 4$.

2. Estimate the three MSEs by simulation, over $100{,}000$ Gaussian data sets with $n = 3$ and $\sigma^2 = 4$, and compare them with your answer to (a).
3. For non-Gaussian i.i.d. data with excess kurtosis $\kappa = \mathbb{E}[(x-\mu)^4]/\sigma^4 - 3$, the variance of S is
$$\mathrm{Var}(S) = (n-1)^2\sigma^4\left(\frac{\kappa}{n} + \frac{2}{n-1}\right). \tag{36.74}$$
Find the MSE-optimal c as a function of n and κ. Evaluate it for Laplace data, which have $\kappa = 3$, at $n = 3$ and $n = 10$, and check the case $n = 3$ by simulation.
4. The optimal shrinkage factor $c^\star$ of the text depends on the unknown θ, whereas for Gaussian data the optimal denominator depends on n alone. Why? What does (c) imply for the use of an MSE-optimal denominator when the distribution of the data is unknown?

8. code **Paired comparisons.** ▯ Two models evaluated on the same test examples or the same seeds share part of their noise, and a paired test removes it.

1. In the text the two models disagree on 41 questions, 25 to 16. Under H_0 the number of questions that only B gets right is Binomial$(41, \frac{1}{2})$. Compute the exact two-sided p-value and compare it with the Monte Carlo value of the sign-flip cell in Section 36.5.2.6. Then fix the net gap at 9 and vary the number of disagreements: compute p for 9 to 0, 17 to 8, 25 to 16 and 55 to 46. What decides the verdict, the gap or the number of disagreements?
2. Model the per-seed accuracies with a seed effect that both models share: for seed s, the accuracies are $0.850 + u_s + e_{A,s}$ and $0.858 + u_s + e_{B,s}$, where $u_s \sim \mathcal{N}(0, 0.01^2)$ and $e_{A,s}, e_{B,s} \sim \mathcal{N}(0, 0.003^2)$ are independent. Implement the sign-flip test on the per-seed differences as an exact test: with k seeds, enumerate all 2^k sign patterns instead of drawing random ones. Over $1{,}000$ experiments with 10 seeds, compare its power with that of the unpaired permutation test `perm_p`.
3. Repeat with 5 seeds. What is the smallest two-sided p-value that the exact sign-flip test can return, and what does it imply at $\alpha = 0.05$? A Monte Carlo version that draws 999 random sign patterns, as `perm_p` draws random shuffles, rejects in a few percent of these experiments. Why?

9. **Standard error of the median.** ▯ Let $x_1, \dots, x_n$ be i.i.d. with c.d.f. F, median m, and density $f(m) > 0$ at the median. The sample median $\hat{m}$ is at most $m + t$ exactly when at least half of the observations are at most $m + t$.

1. The empirical c.d.f. $\hat{F}_n(m+t)$ has mean $F(m+t)$ and variance $F(m+t)(1 - F(m+t))/n$. Use the central limit theorem to show that $P(\hat{m} \le m + t) \approx \Phi(2\sqrt{n}\, f(m)\, t)$ for small t, and conclude that $\mathrm{se}(\hat{m}) \approx 1/(2f(m)\sqrt{n})$. Relate the factor $1/f(m)$ to the slope g' in Equation 36.73.
2. Evaluate the approximation for the exponential distribution with mean 1 at $n = 200$, and compare it with the bootstrap standard error that your run of the text's bootstrap of the median prints. Rerun the cell several times: the printed value moves between about 0.05 and 0.11. Why does the bootstrap estimate of a median's standard error vary this much from data set to data set, when the bootstrap estimate of a mean's standard error does not?
3. Compute the ratio $\mathrm{Var}(\hat{m})/\mathrm{Var}(\bar{x})$ for large n for Gaussian data and for Laplace data. Which estimator of the location would you choose for heavy-tailed data?

Hint for (b): a resample's median is one of a few observed values near $\hat{m}$, and the spacing of those values is random.

10. code **Grids against Monte Carlo.** ⊠ Section 36.5.1.7 compares exponents: with N evaluations in d dimensions, a second-order grid has error of order $N^{-2/d}$ and Monte Carlo of order $N^{-1/2}$. Test this comparison on $f(\mathbf{x}) = \exp(-\|\mathbf{x}\|^2)$ over $[0,1]^d$, whose integral is $\left(\frac{\sqrt{\pi}}{2}\operatorname{erf}(1)\right)^d$ by the product rule Equation 34.62.

    1. Implement the midpoint rule on the full grid of m^d cubes and the Monte Carlo average Equation 36.66 over $N = m^d$ uniform samples. For $d \in \{1, 2, 4, 5, 6\}$ and N up to about 10^6, plot the relative error against N on log-log axes, using for Monte Carlo the root mean squared error over 20 repetitions. Do the slopes match the predicted exponents?
    2. At $d = 4$ the exponents tie. Which method is more accurate there? What sets the constant in front of each rate, and how does each constant change with d?
    3. Both errors can be predicted. The midpoint sum of f is a product of one-dimensional midpoint sums. The root mean squared error of Monte Carlo is $\operatorname{sd}\big(f(\mathbf{U})\big)/\sqrt{N}$ by Equation 36.67, for $\mathbf{U}$ uniform on $[0,1]^d$, and the product rule factors $\mathbb{E}\big[f(\mathbf{U})^2\big]$ into one-dimensional integrals. Compute both predicted relative errors for $d \in \{4, 5, 6\}$ at $N \approx 10^5$ and compare them with your measurements. At which d do the two methods break even? How reliably do 20 repetitions decide the comparison at that d?

11. code **Monte Carlo under heavy tails.** ⊠ The Monte Carlo average Equation 36.66 converges by the law of large numbers when the mean is finite, and its $1/\sqrt{n}$ rate rests on the finite variance in Equation 36.67. Use the Pareto and Cauchy densities of exercise 3, and write $\bar{X}_n$ for the running mean of the first n samples.

    1. Show that $X = U^{-1/\alpha}$, with U uniform on $(0,1)$, has the Pareto density with tail index α, by computing $P(X > x)$. Find the analogous transformation of U for the Cauchy density.
    2. For $\alpha = 3$ and $\alpha = 1.5$, draw 200 independent sequences of 10^5 samples each. Plot the median over sequences of $|\bar{X}_n - \mathbb{E}[X]|$ against n on log-log axes, and fit the slopes. Why is the median a better summary here than the root mean squared error?
    3. Show that the largest of n Pareto samples exceeds $n^{1/\alpha}$ with probability approaching $1 - e^{-1}$ as $n \to \infty$. A single sample of that size shifts $\bar{X}_n$ by about $n^{1/\alpha - 1}$. Use this to predict the slope for $1 < \alpha < 2$. Why does the same argument not set the slope for $\alpha = 3$?
    4. The Pareto density with $\alpha = 1$ and the Cauchy density have no finite mean, so the error of (b) is undefined for them. Draw 200 sequences of 10^5 samples from each, and plot the 25th, 50th and 75th percentiles of $\bar{X}_n$ over sequences against n. What does the running mean do in each case?

12. code **Clustered test sets.** ⊠ Evaluation sets often consist of clusters of related items, such as several questions about one passage, and the i.i.d. bootstrap then understates the standard error (Miller 2024). A test set has G clusters of m items each, its expected accuracy is θ, the correctness indicators of two items in the same cluster have correlation ρ, and items in different clusters are independent.

    1. Show that the average of m variables with common variance v and pairwise cor-

relation ρ has variance $v(1+(m-1)\rho)/m$, and conclude that the accuracy $\hat{\theta}$ on the test set has variance

$$\operatorname{Var}(\hat{\theta}) = \frac{\theta(1-\theta)}{Gm}\bigl(1+(m-1)\rho\bigr). \tag{36.75}$$

2. Simulate $G = 100$ and $m = 20$, drawing a cluster accuracy $\theta_g \sim \operatorname{Beta}(6.3, 2.7)$ for each cluster and conditionally independent items given θ_g. These choices give $\theta = 0.7$ and $\rho = 1/(6.3+2.7+1) = 0.1$. Over 300 test sets, compare the standard deviation of $\hat{\theta}$ with the average i.i.d. bootstrap standard error, which resamples items as in the text's bootstrap of the accuracy, and with the average cluster bootstrap standard error, which resamples whole clusters, using $R = 1{,}000$ resamples for each. By which factor does the i.i.d. bootstrap understate the standard error? Compare the factor with the prediction of (a).

13. code **Stein's paradox.** ▯▯ Observe $\mathbf{x} \sim \mathcal{N}(\theta, \mathbf{I}_d)$ and measure an estimator by its risk $\mathbb{E}\|\hat{\theta} - \theta\|^2$, the MSE with squares replaced by squared norms. The estimator $\mathbf{x}$ has risk d for every θ. James and Stein (1961) showed that for $d \geq 3$ the estimator

$$\hat{\theta}_{\mathrm{JS}} = \left(1 - \frac{d-2}{\|\mathbf{x}\|^2}\right)\mathbf{x} \tag{36.76}$$

has smaller risk for every θ.

1. Prove Stein's lemma (Stein 1981): if $z \sim \mathcal{N}(\theta, 1)$ and h is differentiable with $\mathbb{E}|h'(z)| < \infty$, then $\mathbb{E}[(z-\theta)h(z)] = \mathbb{E}[h'(z)]$. Integrate by parts.
2. For an estimator $\mathbf{x} + \mathbf{g}(\mathbf{x})$ with $\mathbf{g}$ weakly differentiable (here: differentiable away from the origin, with $\mathbb{E}\|\mathbf{g}(\mathbf{x})\|^2 < \infty$), show that its risk is

$$d + \mathbb{E}\|\mathbf{g}(\mathbf{x})\|^2 + 2\sum_{i=1}^{d} \mathbb{E}\left[\frac{\partial g_i}{\partial x_i}(\mathbf{x})\right]. \tag{36.77}$$

   Use this identity to show that $\hat{\theta}_{\mathrm{JS}}$ has risk $d - (d-2)^2\,\mathbb{E}\bigl[1/\|\mathbf{x}\|^2\bigr]$. Where does the argument need $d \geq 3$?
3. Simulate the risks of $\mathbf{x}$, of $\hat{\theta}_{\mathrm{JS}}$, and of the positive-part estimator $\max\bigl(0, 1-(d-2)/\|\mathbf{x}\|^2\bigr)\mathbf{x}$ for $d = 10$ and $\|\theta\| \in \{0, 1, 2, 5, 10, 20\}$, and plot them against $\|\theta\|$.
4. Write the optimal factor $c^\star$ for $c\,\mathbf{x}$ in d dimensions, the analogue of the text's $c^\star$. Which function of the data replaces the unknown quantity in $\hat{\theta}_{\mathrm{JS}}$, and why does this substitute become accurate as d grows? Compare with exercise 1c.

14. [extended] **Peeking at a running test.** ▯▯ An A/B test, or an evaluation during training, is often watched while the data arrive and stopped as soon as the result looks significant (Johari et al. 2017). Let $x_1, x_2, \ldots$ be i.i.d. $\mathcal{N}(\mu, 1)$, let $H_0 : \mu = 0$, and let $S_n = \sum_{i \leq n} x_i$. The z-test at time n rejects when $|S_n|/\sqrt{n} \geq 1.96$.

1. Simulate $4{,}000$ runs of length $N = 10{,}000$ under H_0. Estimate the probability that at least one of K equally spaced looks rejects, for $K \in \{1, 2, 5, 10, 100, 1000, 10000\}$. The law of the iterated logarithm implies that this probability tends to one as monitoring continues indefinitely. How fast does it grow in your table?

2. Put the prior $\mathcal{N}(0, \tau^2)$ on μ and define the mixture likelihood ratio

$$\Lambda_n = \int \prod_{i=1}^{n} \frac{p(x_i \mid \mu)}{p(x_i \mid 0)} \, \mathcal{N}(\mu; 0, \tau^2) \, d\mu. \tag{36.78}$$

   Show that $\Lambda_n = (1 + n\tau^2)^{-1/2} \exp\!\big(\tau^2 S_n^2 / (2(1 + n\tau^2))\big)$, and that under H_0 it is a martingale: $\mathbb{E}[\Lambda_{n+1} \mid x_1, \dots, x_n] = \Lambda_n$.
3. Ville's inequality, which we take on faith, states that a nonnegative martingale with $\Lambda_0 = 1$ satisfies $P(\sup_n \Lambda_n \geq 1/\alpha) \leq \alpha$. Conclude that rejecting at the first n with $\Lambda_n \geq 1/\alpha$ has type I error at most α, however often the data are inspected. Derive the rejection boundary for $|S_n|$, and compare its growth in n with that of the fixed boundary $1.96\sqrt{n}$.
4. Check the type I error of the sequential test by simulation for $\tau^2 \in \{0.01, 0.1, 1\}$. Then set $\mu = 0.2$, and take the fixed-sample z-test whose n follows from Equation 36.68 for power 0.8 as the reference. For each of the three values of τ^2, report the power of the sequential test by that n, its median stopping time, and its power by N. How does the choice of τ^2 affect these numbers? What does a test that is valid at every look give up, and what does it gain?

36.6 Maximum Likelihood

Suppose we flip a coin thirteen times and see nine heads and four tails. Anyone would estimate the probability of heads as $9/13 \approx 0.69$. But why that number, and how far should we trust it? Could the coin still be fair?

One rule answers both questions: choose the parameters under which the observed data are most probable. This is the principle of *maximum likelihood*, and it stands behind nearly every training loss in the book. For the coin it returns $9/13$. For a network it turns a model of the noise in the targets into the loss we minimize: squared error and cross-entropy, which the earlier chapters adopted because they work, both arise this way. The same loss also says how precisely the data determine the parameters. The more sharply it curves around its minimum, the smaller the error bar, and for the coin the curvature will tell us whether thirteen flips can rule out a fair one.

Section 36.5 measured the error of estimators that were given in closed form, such as a sample mean or a sample proportion. Here the estimator is defined by an optimization, and its error bar comes out of the loss itself. We first derive the losses of the earlier chapters from the principle. We then ask how accurate the maximum-likelihood estimate is. That part is the heart of the section, and its main result, asymptotic normality, is the one to remember. Three heads in a row show why we sometimes need a belief held before seeing any data, and we turn such a belief into a regularizer. Models in which part of every observation is hidden follow in Section 36.7.

36.6.1 The Maximum Likelihood Principle

A probabilistic model assigns a probability to every data set it could have produced. Once the data are in, we can turn this around: hold the data fixed and ask how probable they are under each setting of the parameters.

Let θ be the parameters of a model and X the observed data set. The **likelihood** is the probability of the observed data, or for continuous data its density, viewed as a function of

the parameters, $P(X \mid \theta)$. The **maximum likelihood estimate** (MLE) is the parameter that makes it largest,

$$\hat{\theta} = \operatorname*{argmax}_{\theta} P(X \mid \theta). \tag{36.79}$$

Among all parameter settings, the MLE is the one under which what we observed was most probable. The rule treats θ as an unknown constant: it assigns probabilities to data sets, never to parameter values, a distinction that will matter when we add a prior (Section 36.6.4).

36.6.1.1 The Coin

The coin is the smallest model on which we can work the rule through to the end. Its one parameter is the probability θ of heads. Section 36.3 called this number p; now it is the unknown we estimate. The flips are independent, so their probabilities multiply. A sequence with n_H heads and n_T tails has probability

$$P(X \mid \theta) = \theta^{n_H}(1-\theta)^{n_T}. \tag{36.80}$$

Read as a function of θ, this is the likelihood. For our sequence HHHTHTTHHHHHT, with $n_H = 9$ heads and $n_T = 4$ tails, it is $\theta^9(1-\theta)^4$. A fair coin gives the sequence probability $2^{-13} \approx 1.2 \times 10^{-4}$, and a coin with $\theta = 9/13$ gives it 3.3×10^{-4}, about 2.7 times as much.

To find the maximum we take the logarithm first. The logarithm is strictly increasing, so it does not move the location of the maximum, and it turns the product into a sum that is easy to differentiate. Setting the derivative to zero,

$$\frac{d}{d\theta}\Big[9\log\theta + 4\log(1-\theta)\Big] = \frac{9}{\theta} - \frac{4}{1-\theta} = 0 \quad\Longrightarrow\quad 9(1-\theta) = 4\theta \quad\Longrightarrow\quad \hat{\theta} = \frac{9}{13}.$$

The endpoints $\theta = 0$ and $\theta = 1$ give probability zero to a sequence that contains both outcomes, so the maximum lies inside the interval and this root is it. The same steps on the general likelihood Equation 36.80 give

$$\frac{n_H}{\theta} - \frac{n_T}{1-\theta} = 0 \quad\Longrightarrow\quad \hat{\theta} = \frac{n_H}{n_H + n_T}.$$

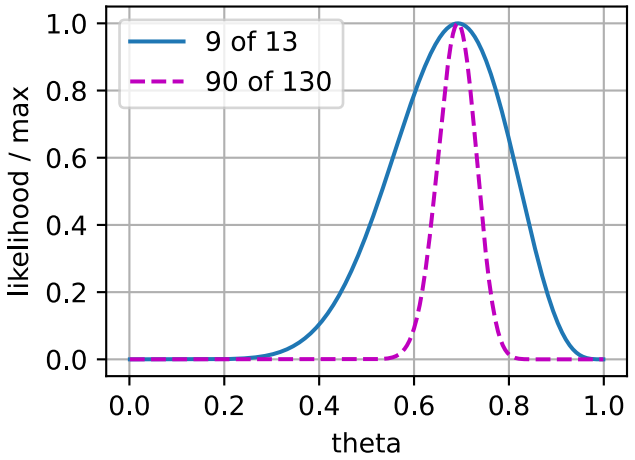
The MLE of a coin's bias is the observed frequency of heads, as anyone would have guessed. The derivative is a sum of one term per flip, $x/\theta - (1-x)/(1-\theta)$ for an outcome $x \in \{0, 1\}$. Each term is the gradient of one observation's log-probability, and it will reappear when we measure the accuracy of $\hat{\theta}$.

Suppose the same frequency had come from ten times as many flips, 90 heads in 130. The estimate cannot change, since only the ratio enters it. What changes is how firmly the data single it out. We plot both likelihoods, each divided by its maximum so that the two fit on one axis.

```
theta = np.linspace(0.001, 0.999, 999)

def likelihood(theta, n_H, n_T):  # divided by its maximum
    lik = theta**n_H * (1 - theta)**n_T
    return lik / lik.max()

d2l.plot(theta, [likelihood(theta, 9, 4), likelihood(theta, 90, 40)], 'theta',
         'likelihood / max', legend=['9 of 13', '90 of 130'])
```



Both curves peak at $9/13 \approx 0.69$, and the second is about three times narrower. We will turn the width of this peak into an error bar.

36.6.1.2 The Negative Log-Likelihood

The coin had a closed-form answer. A model with millions of parameters has none, so we must optimize numerically, and in floating point the likelihood itself is then unusable. With n independent data points it is a product of n probabilities,

$$P(X \mid \theta) = \prod_{i=1}^{n} p(x_i \mid \theta),$$

each in $[0, 1]$. A billion factors near $1/2$ produce roughly $(1/2)^{10^9}$, which underflows to zero in any floating-point format long before we can differentiate it.

The logarithm turns the product into a sum that stays in range:

$$\log\left((1/2)^{10^9}\right) = 10^9 \cdot \log(1/2) \approx -6.93 \times 10^8.$$

Maximizing the likelihood is the same as maximizing the *log-likelihood* $\log P(X \mid \theta)$. We prefer to minimize losses, so we flip the sign and define the **negative log-likelihood** (NLL):

$$\ell(\theta) = -\log P(X \mid \theta) = -\sum_{i=1}^{n} \log p(x_i \mid \theta). \tag{36.81}$$

The NLL is a sum of one term per observation, the surprise of the model at that observation. For the coin it reads $\ell(\theta) = -(n_H \log\theta + n_T \log(1-\theta))$, and nothing stops us from minimizing it by gradient descent as if no closed form existed. We do so for almost nine million flips and print the result next to $n_H/(n_H+n_T)$.

```
n_H = 8675309
n_T = 256245
theta = torch.tensor(0.5, requires_grad=True)
lr = 1e-9
for _ in range(100):
    loss = -(n_H * torch.log(theta) + n_T * torch.log(1 - theta))
    loss.backward()
    with torch.no_grad():
        theta -= lr * theta.grad
    theta.grad.zero_()
theta, n_H / (n_H + n_T)
```

```
(tensor(0.9713, requires_grad=True), 0.9713101437890875)
```

Gradient descent lands on 0.9713, the observed frequency to the digits printed. The learning rate 10^{-9} looks absurd, and the sum forces it on us: the gradient of a loss summed over nearly nine million flips carries that factor of n, and the largest stable step shrinks in proportion (Section 35.1.2; Exercise 1 finds the threshold). This is why the minibatch losses of this book average over examples instead of summing.

The logarithm also gives the gradient an additive form,

$$\nabla_\theta \ell(\theta) = -\sum_{i=1}^{n} \frac{\nabla_\theta p(x_i \mid \theta)}{p(x_i \mid \theta)},$$

one term per example, where the product rule on the likelihood itself would give n products of $n-1$ factors. A random subset of the data therefore gives an unbiased estimate of $\nabla\ell$.

36.6.2 Losses as Negative Log-Likelihoods

The coin's NLL is a loss we derived from a model. The losses of the earlier chapters were written down directly. Most of the common ones are NLLs too, so choosing a loss amounts to choosing a model of the data.

36.6.2.1 Cross-Entropy and the Entropy Floor

A classifier's training loss is an average NLL, and its training curve often levels off at some value above zero. To see what that value is and how low it can go, we first rewrite the average NLL as a sum over outcomes rather than over examples. Let $\hat{p}_{\text{data}}$ be the *empirical distribution* of the sample: an outcome x that occurs n_x times among the n observations gets probability $\hat{p}_{\text{data}}(x) = n_x/n$, its observed frequency. For the coin, $\hat{p}_{\text{data}}(\text{H}) = 9/13$ and $\hat{p}_{\text{data}}(\text{T}) = 4/13$.

Now regroup the sum over data points into a sum over *distinct outcomes*, pooling the observations with $x_i = x$. On the coin, the nine head-flips pool into nine copies of $\log p_\theta(\mathrm{H})$, so heads enters the average with weight $9/13$ and tails with weight $4/13$. In general,

$$\begin{aligned}\frac{1}{n}\,\ell(\theta) &= -\frac{1}{n}\sum_{i=1}^{n}\log p_\theta(x_i) = -\sum_x \frac{n_x}{n}\,\log p_\theta(x)\\ &= -\sum_x \widehat{p}_{\text{data}}(x)\,\log p_\theta(x) = \mathrm{CE}(\widehat{p}_{\text{data}},\,p_\theta)\,.\end{aligned}$$

The right side is the *cross-entropy* from the empirical distribution to the model, $\mathrm{CE}(p,q) = -\sum_x p(x)\log q(x)$. Scaling by the positive constant $1/n$ does not move the minimizer, so for discrete outcomes maximum likelihood is minimum cross-entropy. The identity is one line of regrouping, and it is the reason a classification loss carries that name.

How low can the cross-entropy go? It splits into a part fixed by the data and a part that depends on the model. Recall from Section 36.4.2 the entropy $H(p) = -\sum_x p(x)\log p(x)$ of a distribution on a finite set, its average surprise, and the Kullback–Leibler (KL) divergence $D_{\mathrm{KL}}(p\,\|\,q) = \sum_x p(x)\log\frac{p(x)}{q(x)}$, the average extra surprise of using q where p holds; Section 37.1 develops both in full. We use natural logarithms, so all three quantities are in *nats*. Adding and subtracting $\sum_x p(x)\log p(x)$,

$$\mathrm{CE}(p,q) = -\sum_x p(x)\log q(x) = -\sum_x p(x)\log p(x) + \sum_x p(x)\log\frac{p(x)}{q(x)} = H(p) + D_{\mathrm{KL}}(p\,\|\,q).$$

The entropy depends on p alone; only the KL term depends on the model. By Gibbs' inequality Equation 36.52 that term is never negative, and it vanishes only when $q = p$.

With $p = \widehat{p}_{\text{data}}$ and $q = p_\theta$, the training loss can therefore never fall below the entropy of the observed frequencies, and it reaches that floor only if the model family can represent those frequencies. All that training can reduce is the KL divergence from the data frequencies to the model. For the coin, the floor is $H(9/13) = -\frac{9}{13}\log\frac{9}{13} - \frac{4}{13}\log\frac{4}{13} = 0.617$ nats. A fair-coin model has cross-entropy $\log 2 = 0.693$ nats, so its KL gap is 0.076 nats; a model with $\theta = 0.9$ has a gap of 0.164 nats. Figure 36.26 draws the whole curve.

The same identity holds for a classifier. It models $p_\theta(y \mid \mathbf{x})$, and its average NLL is the average of $-\log p_\theta(y_i \mid \mathbf{x}_i)$. Grouping the examples by their input gives the identity at each distinct input, with the empirical label frequencies at that input in the place of $\widehat{p}_{\text{data}}$ (Exercise 2). For one-hot labels and a softmax output, this average NLL is the softmax cross-entropy loss of Section 3.1. In a typical data set every training input occurs once, the label frequencies at each input are one-hot, and the floor of the training loss is zero however noisy the labels are. The floor a practitioner meets is on held-out data, where the expected loss cannot fall below the conditional entropy of the labels given the inputs (Section 37.1.3.1).

36.6.2.2 Densities

For continuous data the likelihood needs one more step. The probability of observing any exact real value is zero, so taken literally the likelihood is 0 for every θ and cannot choose between them. We therefore ask for a match only to within a small tolerance ϵ and show that ϵ drops out of the minimizer.

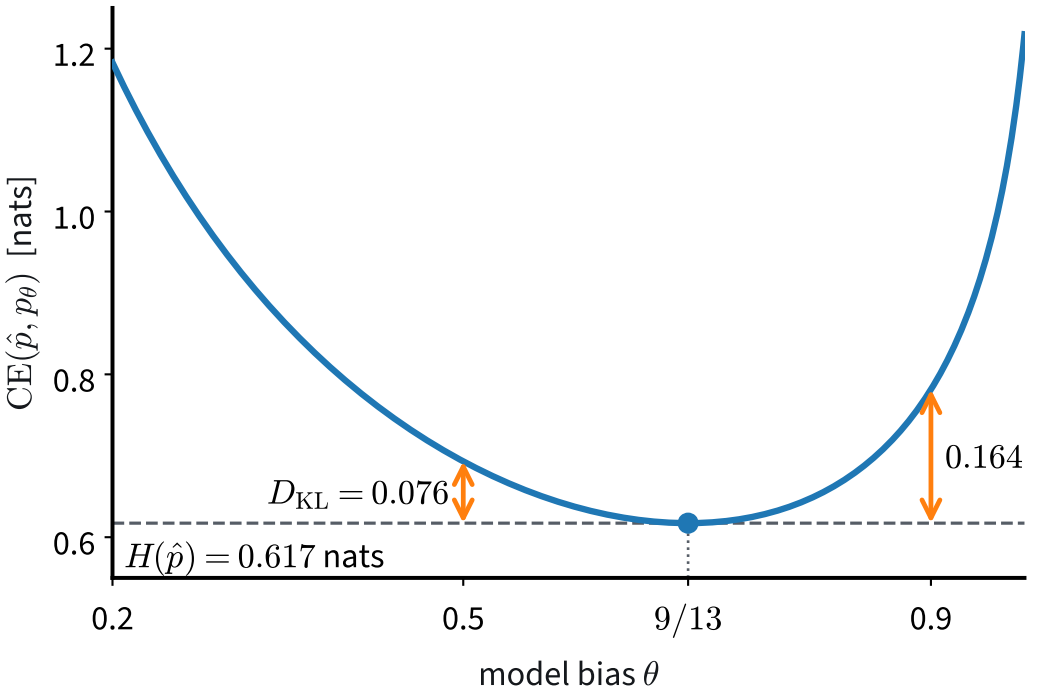


Figure 36.26: The per-flip cross-entropy of the coin's data, 9 heads and 4 tails, as a function of the model bias θ. It never falls below the entropy of the observed frequencies, 0.617 nats (dashed), touches that floor at the MLE $9/13$, and its excess at any other θ is the KL divergence from the data frequencies to the model (orange): 0.076 nats at $\theta = 1/2$ and 0.164 nats at $\theta = 0.9$.

For i.i.d. observations $x_1, \dots, x_n$ with density $p(x \mid \theta)$, the probability that each lands in a window of width ϵ is, to first order in ϵ,

$$P(X_i \in [x_i, x_i + \epsilon] \ \forall i \mid \theta) \approx \epsilon^n \prod_{i=1}^{n} p(x_i \mid \theta).$$

Taking the negative logarithm,

$$-\log P(X_i \in [x_i, x_i + \epsilon] \ \forall i \mid \theta) \approx -n \log \epsilon - \sum_i \log p(x_i \mid \theta).$$

The tolerance enters only through the additive constant $-n \log \epsilon$, which does not depend on θ. Demanding four digits of precision or four hundred changes that constant but never the minimizer, so we drop it and minimize $-\sum_i \log p(x_i \mid \theta)$, as in the discrete case. A density value is not itself a probability: only $\epsilon\, p(x_i \mid \theta)$ approximates one, so p carries the inverse units of x and can exceed 1.

The population form of the cross-entropy identity carries over to continuous data. The average negative log-density at the sample is still the objective, and its expectation under the data-generating density $p_\star$ is

$$\mathbb{E}_{x \sim p_\star}\big[-\log p(x \mid \theta)\big] = h(p_\star) + D_{\mathrm{KL}}(p_\star \,\|\, p_\theta), \qquad h(p_\star) = -\int p_\star(x) \log p_\star(x)\, dx,$$

by the same add-and-subtract step with integrals in place of sums. The *differential entropy* $h(p_\star)$ does not depend on θ. So for the population, maximum likelihood still minimizes a KL divergence. The finite-sample identity of the discrete case has no continuous counterpart, because the empirical distribution of n distinct real numbers is not a density.

36.6.2.3 Gaussian Noise and Squared Error

The most-used regression loss comes out of the same principle once we choose a model for the noise in the targets. Section 2.1.3 derived it for linear regression, and nothing in the derivation depends on the model being linear. Suppose each target is the model's prediction plus Gaussian noise of *fixed* variance, $y_i \sim \mathcal{N}(\hat{y}_i, \sigma^2)$ with $\hat{y}_i = f_\theta(\mathbf{x}_i)$. The negative logarithm of the Gaussian density $(2\pi\sigma^2)^{-1/2} \exp\big(-(y-\hat{y})^2/(2\sigma^2)\big)$, summed over the n independent examples, is

$$-\log \prod_{i=1}^{n} \mathcal{N}(y_i;\, \hat{y}_i,\, \sigma^2) = \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 + \frac{n}{2} \log(2\pi\sigma^2). \tag{36.82}$$

The second term is the same for every θ because the variance is fixed, and the factor $1/(2\sigma^2)$ is a positive constant, so the minimizer of this NLL is the minimizer of $\sum_i (y_i - \hat{y}_i)^2$. Squared-error regression is maximum likelihood under the assumption that the residuals are Gaussian. The same recipe turns many common losses, including the four in the table below, into the NLL of a chosen conditional distribution $p(y \mid \mathbf{x})$. Each row follows by taking the negative logarithm of the distribution and dropping terms that do not depend on θ.

Noise model $p(y \mid \mathbf{x})$	Negative log-likelihood $-\log p(y \mid \mathbf{x})$	Deep-learning loss
Gaussian, fixed variance σ^2	$\frac{1}{2\sigma^2}(y-\hat{y})^2 + \text{const}$	mean squared error (MSE)
Bernoulli	$-y \log \hat{p} - (1-y)\log(1-\hat{p})$	binary cross-entropy (BCE)
Categorical (one-hot y)	$-\sum_k y_k \log \hat{p}_k$	softmax cross-entropy
Laplace, fixed scale s	$\frac{1}{s}\lvert y-\hat{y}\rvert + \text{const}$	mean absolute error (MAE)

For these losses, choosing the loss is choosing a noise model. Not every loss arises this way: the hinge loss of support vector machines has no normalizing constant independent of the model's output, so it is not the NLL of any conditional distribution, even up to an additive constant (Exercise 4).

The choice of noise model matters as soon as the data contain an outlier. Suppose we fit a single constant c to the five readings $\{1, 2, 3, 4, 100\}$, the last from a faulty sensor. Under Gaussian noise we minimize $\frac{1}{2}\sum_i (y_i - c)^2$. Its derivative $-\sum_i (y_i - c)$ vanishes at the mean, $c = 22$. Under Laplace noise we minimize $\sum_i |y_i - c|$, a sum of V-shapes whose slope at c is the number of readings below c minus the number above. That slope is negative below the median and positive above it, so the minimum is the median, $c = 3$. Had the last reading been 5, both fits would give 3. The faulty reading drags the mean to 22 and leaves the median where it was, as Figure 36.27 shows.

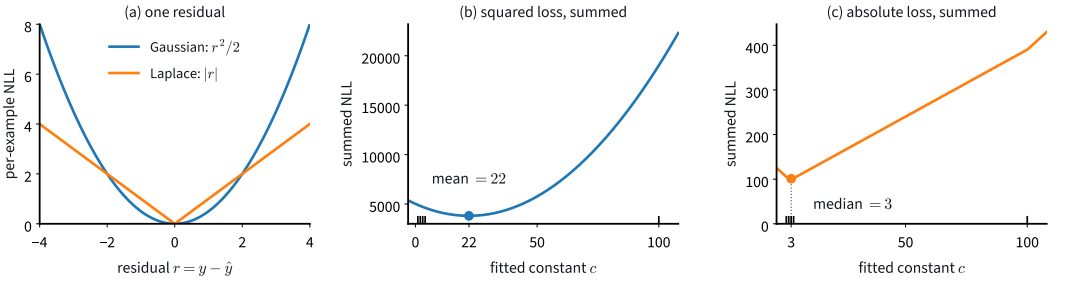


Figure 36.27: Fitting one constant c to the readings $1, 2, 3, 4, 100$. (a) The per-example negative log-likelihood as a function of the residual: quadratic for Gaussian noise, V-shaped for Laplace noise. (b) The summed squared loss is a parabola whose minimum, the mean 22, is pulled far toward the outlier. (c) The summed absolute loss has a kink at every reading (black ticks) and its minimum at the median 3; each point pulls on c with one unit of slope, however far away it lies.

36.6.3 Accuracy of the Maximum Likelihood Estimate

We now return to the question of the opening: how sure are we about $9/13$? For a sample proportion, Section 36.5 answered it with the standard error $\sqrt{\hat{\theta}(1-\hat{\theta})/n}$, the estimated standard deviation of $\hat{\theta}$ across repeated samples. The MLE of a general model is not a sample mean, so we need a formula that reads the error bar off the loss. The ingredients are the per-example gradient we met for the coin and the curvature of the NLL.

36.6.3.1 The Score and Fisher Information

The **score** of one observation is the gradient of its log-probability with respect to the parameters,

$$s(x;\theta) = \nabla_\theta \log p(x \mid \theta).$$

It extends the coin's per-flip gradient to any model, and $\sum_i s(x_i;\hat\theta) = \mathbf{0}$ is the first-order condition that locates the MLE.

At the true parameter the score has mean zero. Differentiating the normalization $\int p(x \mid \theta)\,dx = 1$ under the integral sign gives

$$\mathbf{0} = \nabla_\theta \int p(x \mid \theta)\,dx = \int \nabla_\theta\, p(x \mid \theta)\,dx = \int s(x;\theta)\, p(x \mid \theta)\,dx = \mathbb{E}_\theta\big[\, s(x;\theta)\,\big],$$

where the third equality uses $\nabla_\theta\, p = p\,\nabla_\theta \log p = p\, s$. On average over data drawn from the model, the gradient of the log-likelihood is zero at the parameter that generated the data. Because the score has mean zero, its variance equals its second moment. That variance is the **Fisher information** (Fisher 1925),

$$I(\theta) = \mathrm{Var}_\theta\big[\, s(x;\theta)\,\big] = \mathbb{E}_\theta\big[\, s(x;\theta)\, s(x;\theta)^\top\,\big] = -\,\mathbb{E}_\theta\big[\nabla_\theta^2 \log p(x \mid \theta)\big]. \tag{36.83}$$

The middle equality is the mean-zero property. The last equality is the *information identity*: the variance of the score is the expected curvature of the NLL. It follows by differentiating $\mathbb{E}_\theta[s] = \mathbf{0}$ once more, with the same interchange granted. The product rule splits the derivative into two terms,

$$\mathbf{0} = \nabla_\theta \int s(x;\theta)\, p(x \mid \theta)\,dx = \int \nabla_\theta^2 \log p(x \mid \theta)\; p(x \mid \theta)\,dx + \int s(x;\theta)\,\big(\nabla_\theta\, p(x \mid \theta)\big)^\top dx,$$

and substituting $\nabla_\theta\, p = p\, s$ in the second term turns it into $\mathbb{E}_\theta[\, s\, s^\top]$, so

$$\mathbb{E}_\theta\big[\nabla_\theta^2 \log p(x \mid \theta)\big] + \mathbb{E}_\theta\big[\, s\, s^\top\,\big] = \mathbf{0},$$

which is the last equality of Equation 36.83 rearranged. A sharply curved NLL pins the parameter down, and a flat one leaves it loose. The Fisher information is the curvature of one observation's NLL, averaged over data drawn from the model.

Both derivations moved a derivative inside an integral without comment. The results of this part rest on a few such *regularity conditions*, which we state once:

- the density is twice continuously differentiable in θ;
- its support, the set of x with $p(x \mid \theta) > 0$, does not depend on θ;
- derivative and integral may be interchanged (the tool of Section 34.7); this is the second equality of each derivation above, and the step that a moving support would break;

- the true parameter lies in the interior of the parameter set;
- the Fisher information is finite and nonsingular.

Exercises 9 and 10 show what breaks when the information is singular and when the support moves with the parameter.

For the coin, a single flip $x \in \{0, 1\}$ has log-likelihood $\log p(x \mid \theta) = x \log \theta + (1 - x) \log(1 - \theta)$, so the score and its derivative are

$$s(x; \theta) = \frac{x}{\theta} - \frac{1 - x}{1 - \theta}, \qquad \frac{d}{d\theta}\, s(x; \theta) = -\frac{x}{\theta^2} - \frac{1 - x}{(1 - \theta)^2}.$$

Taking $-\mathbb{E}[\,\cdot\,]$ with $\mathbb{E}[x] = \theta$ gives the per-flip Fisher information and its inverse divided by n,

$$I(\theta) = \frac{1}{\theta} + \frac{1}{1 - \theta} = \frac{1}{\theta(1 - \theta)}, \qquad \frac{I(\theta)^{-1}}{n} = \frac{\theta(1 - \theta)}{n}. \tag{36.84}$$

The per-flip information is smallest at $\theta = 1/2$: a fair coin is the hardest to pin down, because each flip carries the least information about the bias.

A model with two parameters shows the matrix form. For a Gaussian with mean μ and standard deviation σ, the NLL of one observation is $\log \sigma + (x - \mu)^2/(2\sigma^2)$ plus a constant. Its second derivative in μ is $1/\sigma^2$; in σ it is $-1/\sigma^2 + 3(x-\mu)^2/\sigma^4$, whose expectation is $2/\sigma^2$; and the mixed derivative $2(x-\mu)/\sigma^3$ has expectation zero. So $I(\mu, \sigma) = \operatorname{diag}(1/\sigma^2,\, 2/\sigma^2)$: the NLL is twice as curved in σ as in μ, and the data pin down the scale more tightly than the mean. Figure 36.28 shows this bowl for 30 draws.

For the coin and the Gaussian, the MLE is a sample average and we could find its spread without any of this machinery. A model whose MLE is not an average shows what the theory adds. The *Cauchy location model* has density

$$p(x \mid \theta) = \frac{1}{\pi\big(1 + (x - \theta)^2\big)},$$

a bell shape centered at θ with tails so heavy that, as Section 36.1 showed, its mean does not exist. With $u = x - \theta$, the score is $s = 2u/(1 + u^2)$ and its derivative is $\partial_\theta s = -2(1 - u^2)/(1 + u^2)^2$. The Fisher information is the second moment of the score, and a change of variable gives it at sight. Write $u = \tan\varphi$. Since $d\varphi = du/(1 + u^2)$, the angle φ of a standard Cauchy variable is uniform on $(-\pi/2, \pi/2)$, and

$$s = \frac{2\tan\varphi}{1 + \tan^2\varphi} = \sin 2\varphi, \qquad \text{so} \qquad I = \mathbb{E}\big[\sin^2 2\varphi\big] = \frac{1}{2}.$$

Each Cauchy observation carries half a unit of information about its center, the same as a Gaussian observation with $\sigma^2 = 2$. The score never exceeds 1 in size and tends to zero for a far-away point, which is why outliers barely move the MLE.

A model without such a trick still yields both forms of I to automatic differentiation. The cell draws 10,000 Cauchy observations at $\theta^\star = 0$ and prints the mean of the score, its variance,

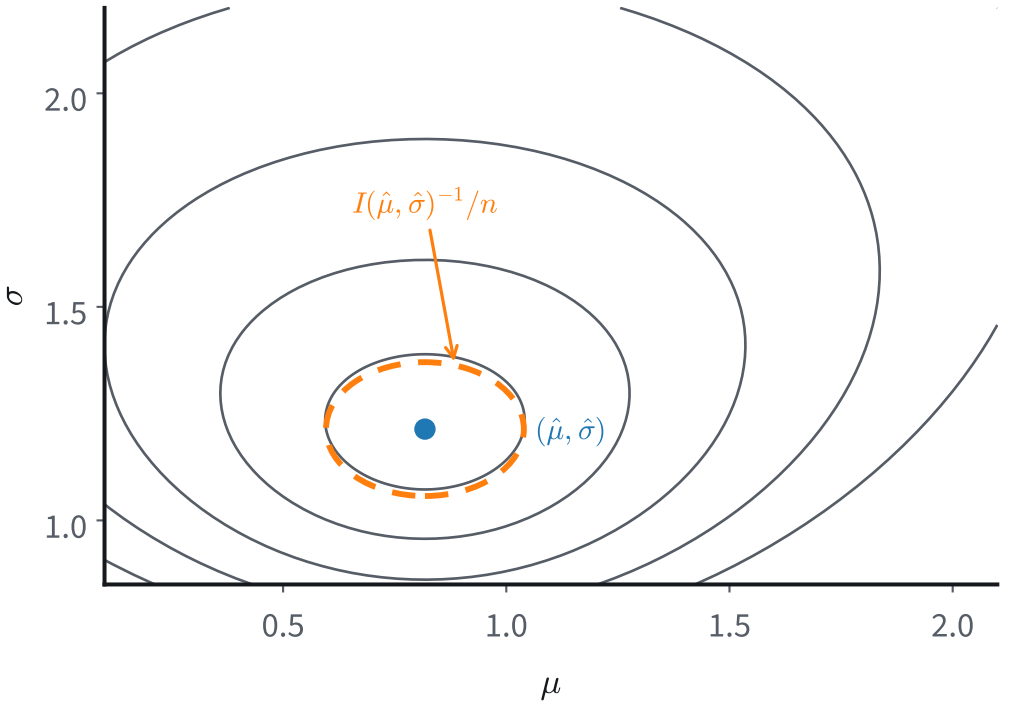


Figure 36.28: The negative log-likelihood of a Gaussian's parameters (μ, σ) for $n = 30$ draws from $\mathcal{N}(1, 1.5^2)$, with the MLE marked by the dot. The dashed ellipse is $I(\hat{\mu}, \hat{\sigma})^{-1}/n$; it nearly traces the innermost contour, where the NLL lies one half above its minimum, and it is wider along μ than along σ.

and minus the mean curvature. By the uniform angle, their Monte Carlo standard errors are about 0.007, 0.004 and 0.008.

```
def log_p(theta, x):  # log density of the Cauchy location model
    return -torch.log1p((x - theta)**2) - np.log(np.pi)

x = torch.tensor(np.random.default_rng().standard_cauchy(10_000),
                 dtype=torch.float32)  # theta* = 0
score = torch.func.vmap(torch.func.grad(log_p), in_dims=(None, 0))
curv = torch.func.vmap(torch.func.grad(torch.func.grad(log_p)),
                       in_dims=(None, 0))
s, h = score(torch.tensor(0.0), x), curv(torch.tensor(0.0), x)
print(f'mean score {s.mean():.3f}, variance of score {s.var():.3f}, '
      f'-mean curvature {-h.mean():.3f}')
```

```
mean score 0.004, variance of score 0.498, -mean curvature 0.502
```

`torch.func.grad` differentiates the log-density of one observation, and `vmap` maps it over all $10{,}000$ observations; applying `grad` twice gives the curvature.

The three numbers land within two or three of these standard errors of 0, 0.5 and 0.5.

36.6.3.2 The Error Bar of the Coin

The Fisher information gives the error bar for $9/13$ that the opening asked for. The next subsection shows that for large n the MLE has approximately the variance $1/(nI(\theta))$. We take that on trust for a moment and apply it. For the coin, $1/(nI) = \theta(1-\theta)/n$ by Equation 36.84, and plugging in $\hat{\theta}$ for the unknown θ gives the estimated standard error

$$\widehat{\mathrm{se}} = \sqrt{\frac{\hat{\theta}(1-\hat{\theta})}{n}} = \sqrt{\frac{(9/13)(4/13)}{13}} = 0.128.$$

So $\hat{\theta} = 0.69 \pm 0.13$. The approximate 95% band $\hat{\theta} \pm 1.96\,\widehat{\mathrm{se}} = [0.44,\ 0.94]$ contains $1/2$: thirteen flips cannot tell this coin from a fair one. Pinning θ down to ± 0.05 at 95% would take about $1.96^2\,\hat{\theta}(1-\hat{\theta})/0.05^2 = 1.96^2 \cdot 0.213/0.05^2 \approx 330$ flips.

The same number comes straight out of the loss. The second derivative of the coin's NLL at $\hat{\theta}$ is

$$\ell''(\hat{\theta}) = \frac{n_H}{\hat{\theta}^2} + \frac{n_T}{(1-\hat{\theta})^2} = \frac{9}{(9/13)^2} + \frac{4}{(4/13)^2} = 61.0,$$

and $1/\sqrt{61.0} = 0.128$. For the coin this agreement is no coincidence. With $n_H = n\hat{\theta}$ and $n_T = n(1-\hat{\theta})$, the curvature is $n/(\hat{\theta}(1-\hat{\theta})) = nI(\hat{\theta})$. For a model that satisfies the conditions above, the same holds approximately at large n: a training run can compute the curvature of the NLL at its minimum, the Hessian, and one over its square root is the error bar. For several parameters, the error bars are the square roots of the diagonal of the inverse Hessian. A deep network does not satisfy the conditions, and Section 36.6.3.5 says what is left of the recipe there. The curvature also explains the likelihood plot at the start

of the section: ten times the data multiplies the curvature by ten and narrows the peak by $\sqrt{10} \approx 3.2$.

36.6.3.3 Consistency and Asymptotic Normality

Why is the variance of the MLE about $1/(nI)$? We first need the MLE to head for the right value at all, and then a description of how it fluctuates around that value. Three tools from earlier sections do the work: consistency, $\hat{\theta} \xrightarrow{P} \theta^\star$, and the weak law of large numbers Equation 36.65, both from Section 36.5, and the central limit theorem of Section 36.3, by which a sum of n i.i.d. mean-zero terms with variance v, divided by $\sqrt{n}$, converges in distribution ($\xrightarrow{d}$) to $\mathcal{N}(0, v)$.

The cross-entropy identity of Section 36.6.2.1 explains why the MLE approaches the true parameter. Suppose the data come from the model with parameter $\theta^\star$. The average NLL is an average of i.i.d. terms, so by the law of large numbers it converges to the population NLL

$$\mathbb{E}_{\theta^\star}\big[-\log p(x \mid \theta)\big] = H(p_{\theta^\star}) + D_{\mathrm{KL}}(p_{\theta^\star} \,\|\, p_\theta),$$

with h in place of H for densities (Section 36.6.2.2). By Gibbs' inequality Equation 36.52 this is smallest where the KL term vanishes, at $\theta = \theta^\star$. The minimizer is unique if the model is **identifiable**: distinct parameters give distinct distributions. Neural networks are not. Permuting the hidden units of a layer, together with their weights, changes the parameter vector but not the function it computes. For such models, consistency is a statement about the fitted distribution, or about the class of parameters that give it, never about one parameter vector.

Under the following conditions the MLE is consistent (Wasserman 2013), a result we state without proof: the data are i.i.d. from $p_{\theta^\star}$, the model is identifiable, the population optimum is well separated (no θ far from $\theta^\star$ comes arbitrarily close to the optimal value; on a compact parameter set with a continuous objective, uniqueness suffices), and the average NLL converges to the population NLL uniformly in θ. The uniformity is the hypothesis that is easy to overlook. The law of large numbers holds at each fixed θ, and convergence at each point does not force the minimizers to converge. For the coin no theorem is needed: $\hat{\theta} = n_H/n \to \theta^\star$ directly by the law of large numbers.

Consistency says where $\hat{\theta}$ lands. The error bar needs the size and shape of its fluctuation. The derivation is short for a scalar θ, and its one non-routine step is to Taylor-expand the first-order condition rather than the NLL. Write $S_n(\theta) = \sum_i s(x_i; \theta)$ for the total score.

I. The MLE solves $S_n(\hat{\theta}) = 0$. Expanding around $\theta^\star$ to first order,

$$0 = S_n(\hat{\theta}) \approx S_n(\theta^\star) + S_n'(\theta^\star)\,(\hat{\theta} - \theta^\star).$$

II. Solving for the error and multiplying by $\sqrt{n}$ splits it into a ratio of two averages,

$$\sqrt{n}\,(\hat{\theta} - \theta^\star) \approx \frac{S_n(\theta^\star)/\sqrt{n}}{-S_n'(\theta^\star)/n}.$$

III. The numerator is a sum of n i.i.d. scores, each with mean zero and variance $I(\theta^\star)$, divided by $\sqrt{n}$; by the central limit theorem it converges in distribution to $\mathcal{N}(0, I(\theta^\star))$. The denominator is an average of n i.i.d. terms $-\partial_\theta s(x_i;\theta^\star)$ whose mean is $I(\theta^\star)$ by the information identity; by the law of large numbers it converges to $I(\theta^\star)$. A Gaussian with variance I divided by the constant I has variance $I/I^2 = 1/I$.

The numerator used the variance form of I and the denominator its curvature form. The information identity makes the two the same number, which is why the result is so simple. The vector case runs the same way with matrices, giving the covariance $I^{-1} I I^{-1} = I^{-1}$. A full proof also controls the Taylor remainder, and the integrable bound on the curvature in the proposition below does this.

Proposition (asymptotic normality). *Under the regularity conditions, the conditions for consistency above, and a bound $\|\nabla_\theta^2 \log p(x \mid \theta)\| \le M(x)$ with $\mathbb{E}_{\theta^\star}[M(x)] < \infty$ for all θ near $\theta^\star$,*

$$\sqrt{n}\,(\hat\theta - \theta^\star) \;\xrightarrow{d}\; \mathcal{N}\!\left(\mathbf{0},\; I(\theta^\star)^{-1}\right) \tag{36.85}$$

(Wasserman 2013). We do not prove it beyond the three steps above. It is the central result of the section: the error bar, the efficiency of the MLE and the limits of the theory all hang on it. The error of the MLE is approximately Gaussian, shrinks at the rate $1/\sqrt{n}$, and has the inverse Fisher information as its covariance. For a finite n we read it as the approximation $\mathrm{Cov}(\hat\theta) \approx I(\theta^\star)^{-1}/n$, which is the error bar of the previous subsection. Halving the error bar requires four times the data. When the model is wrong, the variance and curvature forms of I differ, and the covariance becomes the "sandwich" of Section 36.6.3.5.

To test the proposition we need a model whose MLE is not a sample mean, and the Cauchy location model is one. Its likelihood equation $\sum_i 2u_i/(1+u_i^2) = 0$ has no closed form, so we solve it by **Fisher scoring**: Newton's method with the expected curvature $nI = n/2$ in place of the observed one,

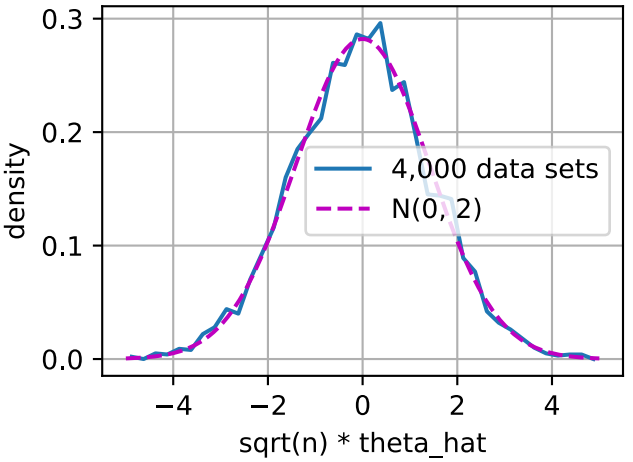
$$\theta \leftarrow \theta + \frac{S_n(\theta)}{nI} = \theta + \frac{2}{n}\,S_n(\theta).$$

The expected curvature is always positive, while the observed curvature of a Cauchy NLL can be negative away from the minimum, where a Newton step would head uphill. We start each run at the sample median, which is already close. We fit $4{,}000$ data sets of $n = 100$ observations at $\theta^\star = 0$ with ten scoring steps each. If the asymptotic theory is already accurate at $n = 100$, the histogram of $\sqrt{n}\,\hat\theta$ follows $\mathcal{N}(0, 1/I) = \mathcal{N}(0, 2)$.

```
reps, n = 4000, 100
x = np.random.default_rng().standard_cauchy((reps, n))  # theta* = 0
theta = np.median(x, axis=1)
for _ in range(10):  # Fisher scoring: theta += S_n / (n I) with I = 1/2
    u = x - theta[:, None]
    theta += (2 / n) * (2 * u / (1 + u**2)).sum(axis=1)
z = np.sqrt(n) * theta  # sqrt(n) (theta_hat - theta*)
hist, edges = np.histogram(z, bins=40, range=(-5, 5), density=True)
grid = np.linspace(-5, 5, 201)
d2l.plot([(edges[:-1] + edges[1:]) / 2, grid],
```

```
        [hist, np.exp(-grid**2 / 4) / np.sqrt(4 * np.pi)],
        'sqrt(n) * theta_hat', 'density', legend=['4,000 data sets', 'N(0, 2)'])
print(f'standard deviation of sqrt(n) * theta_hat: {z.std():.2f}')
```

```
standard deviation of sqrt(n) * theta_hat: 1.43
```



The histogram sits on the predicted curve in center, width, and height. The standard deviation of $\sqrt{n}\,\hat{\theta}$ comes out between about 1.40 and 1.47 from run to run, against the limit $\sqrt{2} = 1.41$; at $n = 100$ the variance is still a few percent above its limiting value.

36.6.3.4 The Cramér–Rao Bound

Is the MLE's variance good, or could a cleverer estimator do better? The Fisher information also sets a floor that no estimator which is right on average can beat. As in Section 36.5, an estimator $\tilde{\theta}$ is *unbiased* if $\mathbb{E}_\theta[\tilde{\theta}] = \theta$ for every θ.

Proposition (Cramér–Rao bound). *Under the regularity conditions, any unbiased estimator $\tilde{\theta}$ of a scalar parameter from n i.i.d. observations satisfies*

$$\operatorname{Var}(\tilde{\theta}) \;\geq\; \frac{1}{n\,I(\theta)} \tag{36.86}$$

(Cramér 1946; Radhakrishna Rao 1945). No estimator that is right on average can be more precise than the curvature of the likelihood allows. The proof takes three lines, and its one idea is to correlate the estimator with the score.

Proof. Let $S_n = \partial_\theta \log P(X \mid \theta)$ be the score of the whole data set, a sum of n independent per-example scores; it has mean zero and variance $nI(\theta)$. Unbiasedness says $\int \tilde{\theta}(X)\,P(X \mid \theta)\,dX = \theta$. Differentiating both sides under the integral, with $\partial_\theta P = P\,S_n$,

$$1 = \int \tilde{\theta}(X)\,S_n\,P(X \mid \theta)\,dX = \mathbb{E}[\tilde{\theta}\,S_n] = \operatorname{Cov}(\tilde{\theta},\,S_n),$$

where the last step uses $\mathbb{E}[S_n] = 0$. The Cauchy–Schwarz inequality for covariances gives $1 \le \mathrm{Var}(\tilde{\theta})\mathrm{Var}(S_n) = \mathrm{Var}(\tilde{\theta})\, nI(\theta)$. ■

Unbiasedness enters through the "1" on the left. The fixed support enters when we differentiate under the integral; Exercise 10 shows an unbiased estimator that beats the bound when the support moves with θ. For a vector parameter the bound reads $\mathrm{Cov}(\tilde{\theta}) \succeq I(\theta)^{-1}/n$ in the positive semidefinite order.

The MLE's limiting variance Equation 36.85 equals this floor, the smallest variance any unbiased estimator can have; in this sense the MLE is *asymptotically efficient*. For the coin the floor is reached at every n, not only in the limit, because n_H/n is an unbiased sample mean with variance $\theta(1-\theta)/n = 1/(nI)$. In general the MLE meets the floor only as $n \to \infty$. At finite n it can be biased, and then the bound does not apply to it; the Gaussian variance below is an example.

How much is efficiency worth? For the Cauchy location model the sample median is the natural competitor, and the sample mean the naive one; by symmetry the MLE and the median are both unbiased. The floor for n times the variance is $1/I = 2$, and the asymptotic theory of the median predicts $\pi^2/4 \approx 2.47$. For the mean, guess before reading the output. The cell reuses the $4{,}000$ data sets.

```
print(f'n Var(MLE) = {n * theta.var():.2f}, '
      f'n Var(median) = {n * np.median(x, axis=1).var():.2f}, '
      f'n Var(mean) = {n * x.mean(axis=1).var():.3g}')
```

```
n Var(MLE) = 2.06, n Var(median) = 2.50, n Var(mean) = 7.79e+05
```

The MLE comes out between about 1.9 and 2.2, close to the floor of 2, and the median near 2.5, so the MLE needs about 20% fewer observations for the same error bar. The sample mean has an enormous variance that changes wildly from run to run: the mean of n Cauchy observations is again Cauchy, so averaging never helps.

36.6.3.5 Limits of the Theory

The guarantees of this part hold under the stated conditions and as $n \to \infty$. Four of the five cases below break one condition each, and the exercises work them out. The fifth, a wrong model, is the case every deep learning reader lives in, so we work it here.

- *Finite-sample bias.* The MLE of a Gaussian's variance divides the sum of squares by n, so on average it underestimates σ^2 by the factor $(n-1)/n$ (Section 36.5). The MLE is optimal in the limit, not unbiased at every n.
- *The boundary.* A parameter near the edge of its range, such as a coin with bias 0.95, needs far more data before the Gaussian limit is accurate; what matters is the expected number of rare outcomes, not n. On the edge itself the MLE can fail to exist (Exercise 5).
- *A support that moves with the parameter.* The uniform distribution on $[0, \theta]$ has the MLE $\max_i x_i$, an error of order $1/n$ rather than $1/\sqrt{n}$, and an unbiased estimator that beats the Cramér–Rao floor (Exercise 10).
- *Singular information.* The rescaling symmetry of a ReLU network gives I a null vector, so its inverse does not exist (Exercise 9).

Misspecification. When no p_θ is the distribution of the data, the MLE converges to the parameter of smallest KL divergence to the truth (White 1982). The score still has mean zero there,

but the variance of the score, B, and the expected curvature, A, now differ, and step III gives the variance B/A^2 in place of $1/I$, in matrix form the *sandwich* $\mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}$ (Exercise 11). Fit, for example, a Poisson model with rate λ to counts with mean m and variance $m + m^2/\alpha$. The MLE is the sample mean, so it still recovers m, with variance $(m + m^2/\alpha)/n$. The Poisson model's Fisher information $I = 1/\lambda$ reports m/n, too small by the factor $1 + m/\alpha$, which is 3 for $m = 4$ and $\alpha = 2$.

A network is both misspecified and non-identifiable, so no error bar attaches to a single weight. What the theory still supports are statements about the fitted distribution and its predictions, which the symmetries leave unchanged. The Fisher matrix itself returns twice in the book: as the Hessian of the KL divergence between neighbouring members of a family (Section 37.3), and as the curvature with which natural-gradient methods precondition a step (Section 35.5).

36.6.4 MAP Estimation and Regularization

Thirteen flips gave a usable estimate. Three flips may not. Suppose they come up HHH. The MLE is then $\hat{\theta} = 3/3 = 1$: it assigns probability zero to tails, so the first tail we see afterwards has infinite NLL. A classifier built from counts that has never seen a feature together with a class is in the same position. Nobody who has handled coins believes this one can never land tails. What we need is a way to say, before seeing any data, that coins are rarely perfectly biased.

A probability distribution over the parameters expresses such a belief. Maximum likelihood had no place for one, since it assigned probabilities to data sets only. With a **prior** $p(\theta)$, Bayes' rule turns the likelihood into a distribution over the parameters given the data,

$$p(\theta \mid X) = \frac{P(X \mid \theta)\, p(\theta)}{P(X)}, \qquad P(X) = \int P(X \mid \theta)\, p(\theta)\, d\theta. \tag{36.87}$$

The left side is the **posterior**, what we believe after seeing the data. The denominator, the **evidence**, is the probability of the data with the parameters averaged out; it does not depend on θ. The **maximum a posteriori** (MAP) estimate is the peak of the posterior. The evidence does not move the peak, so maximizing the posterior and taking negative logarithms gives

$$\hat{\theta}_{\text{MAP}} = \operatorname*{argmax}_{\theta}\; P(X \mid \theta)\, p(\theta) = \operatorname*{argmin}_{\theta}\; \Big[\underbrace{-\textstyle\sum_i \log p(x_i \mid \theta)}_{\text{data fit (NLL)}} \; \underbrace{-\log p(\theta)}_{\text{penalty}} \Big]. \tag{36.88}$$

MAP estimation minimizes the familiar NLL plus a penalty, the negative log-prior. A prior that puts little mass on extreme parameters penalizes them, which is what a regularizer does. For HHH, a prior that adds one imaginary head and one imaginary tail (the Beta$(2, 2)$ prior below) gives $\hat{\theta}_{\text{MAP}} = (3 + 1)/(3 + 2) = 4/5$. A later tail then costs $-\log(1/5) = 1.6$ nats, a finite surprise.

36.6.4.1 Gaussian Prior and Weight Decay

Weight decay, the most common regularizer in deep learning, is the MAP penalty of a Gaussian prior. Section 2.7 derives this for linear regression; here it is in the notation of this

section, for any model with Gaussian noise. Take every parameter to be a priori small, $\theta_j \sim \mathcal{N}(0, \tau^2)$ independently, with τ the typical size we expect. The prior density is proportional to $\exp(-\|\theta\|_2^2/(2\tau^2))$, so its negative logarithm is $\|\theta\|_2^2/(2\tau^2)$ plus a constant. With the Gaussian NLL Equation 36.82, the MAP objective Equation 36.88 becomes

$$\hat{\theta}_{\text{MAP}} = \operatorname*{argmin}_{\theta} \; \frac{1}{2\sigma^2} \sum_i (y_i - \hat{y}_i)^2 + \frac{1}{2\tau^2} \|\theta\|_2^2.$$

Multiplying by the positive constant σ^2/n, which does not move the minimizer, gives the weight-decay objective Equation 2.32,

$$\frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \left(y_i - \hat{y}_i\right)^2 + \frac{\lambda}{2} \|\theta\|_2^2, \qquad \lambda = \frac{\sigma^2}{n\tau^2}.$$

The book's version leaves the bias b unpenalized, which amounts to a flat prior on b. The prior's weight in the averaged loss falls as $1/n$: regularization matters most when data are scarce. Section 35.7.3.2 reads the same penalty as a constraint on the norm of the weights.

A *Laplace* prior $p(\theta) \propto e^{-\|\theta\|_1/s}$ with scale s gives the L_1 penalty $\|\theta\|_1/s$, whose kink at the origin produces exact zeros, that is, sparse parameter vectors. Figure 36.29 shows the compromise in one dimension: the NLL favors the MLE, the negative log-prior favors the prior mean, and the MAP estimate lies between them.

The same picture explains the two limits. As the prior flattens ($\tau \to \infty$, or $s \to \infty$) the penalty vanishes, and as the data grow the NLL deepens in proportion to n while the prior stays fixed. Either way MAP approaches the MLE.

36.6.4.2 Beta Prior and Pseudo-Counts

For the coin the prior must live on $[0, 1]$, and the Beta family is the convenient choice. A Beta(a, b) prior with $a, b > 0$ has density proportional to $\theta^{a-1}(1-\theta)^{b-1}$. Multiplying by the likelihood Equation 36.80 adds the observed counts to the exponents, so the posterior is again a Beta distribution, the conjugate update Equation 36.58 of Section 36.4:

$$p(\theta \mid X) \;\propto\; \theta^{\,n_H + a - 1} \, (1-\theta)^{\,n_T + b - 1}.$$

The posterior has the shape of a coin likelihood with $n_H + a - 1$ heads and $n_T + b - 1$ tails. When both exponents are positive, its peak follows from the same one-variable maximization as the coin's MLE, "head exponent over total exponent":

$$\hat{\theta}_{\text{MAP}} \;=\; \frac{n_H + a - 1}{n + a + b - 2}, \qquad n = n_H + n_T. \tag{36.89}$$

For the mode, then, the prior counts as $a - 1$ imaginary heads and $b - 1$ imaginary tails, appended to the data before taking the frequency. This agrees with the convention fixed in Section 36.4: a and b are the *pseudo-counts* of the posterior mean, $a - 1$ and $b - 1$ those of the posterior mode. A Beta$(2, 2)$ prior adds one imaginary flip of each kind to the mode: it turns our $9/13 \approx 0.692$ into $(9 + 1)/(13 + 2) = 2/3$ and HHH into $4/5$. A Beta$(30, 30)$ prior holds the MAP estimate near $1/2$ until the data outweigh its 58 imaginary flips.

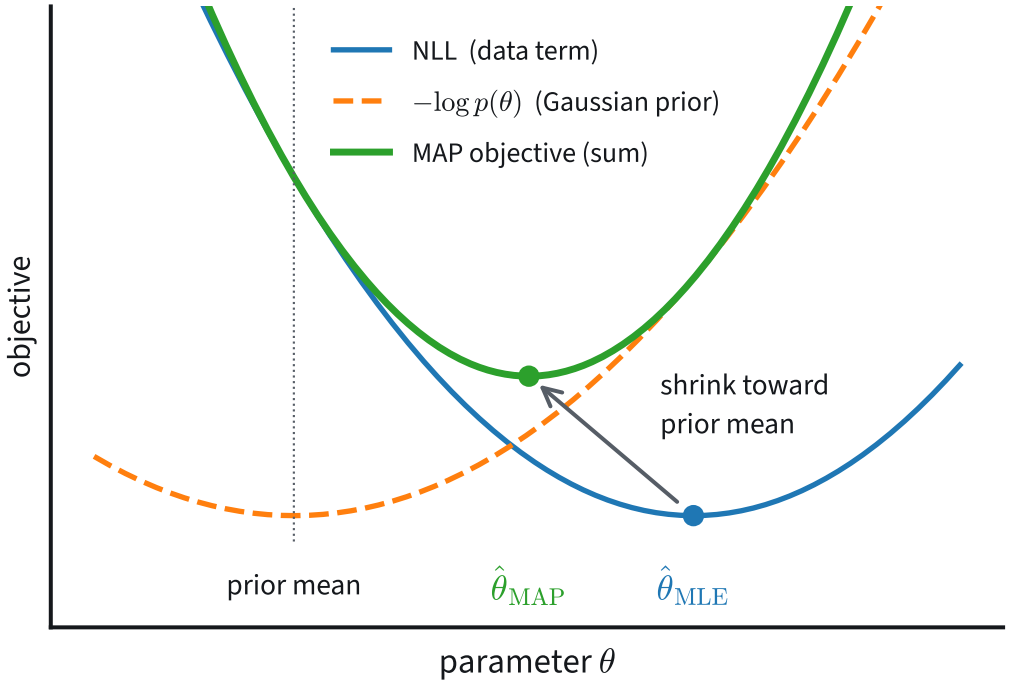


Figure 36.29: Maximum a posteriori balances data against prior. The negative log-likelihood (blue) is minimized at the MLE; adding the negative log of a Gaussian prior centered at the prior mean (orange, dashed) yields the MAP objective (green), whose minimum is pulled from the MLE toward the prior mean. A tighter prior (smaller τ) pulls harder; as $\tau \to \infty$ the prior flattens and MAP returns to the MLE.

The two readings meet in a familiar formula. The posterior mean under the flat prior $\mathrm{Beta}(1,1)$ is Laplace's rule of succession $(n_H+1)/(n+2)$ (Laplace 1814), known in classifiers built from counts as add-one smoothing. The same number is the MAP estimate under $\mathrm{Beta}(2,2)$, while MAP under the flat $\mathrm{Beta}(1,1)$ is the plain MLE. So add-one smoothing is a posterior mean under a flat prior, not a flat-prior MAP estimate.

36.6.4.3 Posterior Mode and Posterior Distribution

The mode is a fragile summary, because it depends on how the parameter is written. A nonlinear change of variables multiplies a density by a Jacobian, which moves its peak while leaving probabilities of regions unchanged. $\mathrm{Beta}(1,1)$ is flat in θ, but the same belief written for the log-odds $\eta = \log \frac{\theta}{1-\theta}$ is not flat, and MAP in η gives $10/15$ instead of $9/13$ (Exercise 7). The likelihood has no Jacobian, because its values are probabilities of the data attached to points of the parameter space. The MLE is therefore **equivariant**: if $\widehat{\theta}$ is the MLE of θ, then $g(\widehat{\theta})$ is the MLE of $g(\theta)$ for any one-to-one g (Wasserman 2013). The MLE of the coin's log-odds is $\log \frac{9/13}{4/13} = \log \frac{9}{4}$. This is why maximum likelihood is defined without any prior, rather than as MAP with a flat one, whose answer would depend on how the parameter is written down. Equivariance has a cost of its own: the same exercise shows that no estimator can be both equivariant and unbiased for every parameterization.

The deeper limit is that MAP, like the MLE, reports a single parameter vector. The posterior $p(\theta \mid X)$ itself says how uncertain the parameters remain. A prediction that uses it averages over that uncertainty, $\int p(x \mid \theta)\, p(\theta \mid X)\, d\theta$, while a prediction that plugs in the mode discards it. Such averages are rarely available in closed form, and computing them is the subject of Section 36.8.

A nearer obstacle comes first. Everything in this section assumed that the probability of an observation can be written down as a formula in the parameters. When each data point comes from one of several sources and nobody recorded which, that probability is a sum over the possibilities, and the logarithm of a sum does not fall apart into one term per example. Section 36.7 shows what to maximize instead.

36.6.5 Summary

Maximum likelihood chooses the parameters under which the observed data are most probable, which in practice means minimizing the negative log-likelihood, a sum of one surprise per example. Choosing a loss is choosing a noise model: squared error for Gaussian noise, absolute error for Laplace noise, and for discrete data the cross-entropy from the observed frequencies to the model, which cannot fall below their entropy. The curvature of the same loss measures precision: under regularity conditions the MLE's error is approximately Gaussian with variance $1/(nI)$, where the Fisher information I is both the variance of the score and the expected curvature of the NLL, and no unbiased estimator does better. These guarantees fail for a misspecified or non-identifiable model, and a deep network is both. A prior adds its negative logarithm as a penalty: a Gaussian prior is weight decay, a Beta prior adds pseudo-counts, and the MAP estimate is still a single point.

36.6.6 Exercises

1. code **Learning rate and loss scale.** The cell of Section 36.6.1.2 minimizes the summed coin NLL by gradient descent with `lr` $= 10^{-9}$ for the counts `n_H` $= 8675309$ and `n_T` $=$

256245. Near its minimizer $\hat{\theta}$ the NLL is approximately quadratic with curvature $c = n\,I(\hat{\theta})$, where $n = n_H + n_T$ and I is the per-flip Fisher information Equation 36.84. On such a quadratic, each step multiplies the distance to $\hat{\theta}$ by $1 - \texttt{lr} \cdot c$, so gradient descent converges if and only if $\texttt{lr} < 2/c$.

1. Predict the learning rate above which the cell stops converging. Rerun it with $\texttt{lr} \in \{5 \times 10^{-9},\ 6.5 \times 10^{-9},\ 10^{-8}\}$ and print the last four iterates of θ. What do the iterates do just above the threshold?
2. Multiply `n_H` and `n_T` by 10^{-3}, keeping $\texttt{lr} = 10^{-9}$. Predict θ after the 100 steps once from the gradient at the starting point $\theta = 0.5$ and once from the contraction factor $1 - \texttt{lr} \cdot c$ of the quadratic model. Which prediction matches the run, and why?
3. Multiply the original counts by 10 instead, keeping $\texttt{lr} = 10^{-9}$. Show that, in exact arithmetic, scaling both counts by k at a fixed `lr` produces the iterates of the original counts with learning rate $k \cdot \texttt{lr}$. Which run of (a) does the factor 10 reproduce? Do its iterates leave $(0, 1)$ within the 100 steps?
4. Divide the loss by n before differentiating. What is the threshold now, and why does it no longer depend on n?

2. **The cross-entropy floor.** For discrete outcomes the average NLL equals $H(\hat{p}_{\text{data}}) + D_{\text{KL}}(\hat{p}_{\text{data}} \,\|\, p_\theta)$ by Section 36.6.2.1, so it cannot fall below the empirical entropy.
   1. Evaluate this floor in nats for the coin sequence with 9 heads and 4 tails, and the KL gap of the fair-coin model $\theta = 1/2$.
   2. A classifier models $p(y \mid \mathbf{x})$ and can represent any distribution over labels separately for each distinct input. Show that its smallest achievable average NLL on a training set is the empirical conditional entropy $\sum_{\mathbf{x}} \hat{p}_{\text{data}}(\mathbf{x})\, H(\hat{p}_{\text{data}}(\cdot \mid \mathbf{x}))$. Evaluate it for a binary task in which each of 100 distinct inputs occurs 10 times and carries the wrong label in exactly one of its occurrences.
   3. Now every training input occurs once, as in a typical image data set. What is the floor of (b)? Why does a training loss near zero then say nothing about how noisy the labels are? Which quantity that we can measure has the population conditional entropy $H(Y \mid X)$ as its floor?
3. **Learned noise variance.** The Gaussian NLL Equation 36.82 holds the variance σ^2 fixed. Here the variance is learned as well.
   1. Let one shared σ^2 be a parameter alongside θ. Minimize the NLL over σ^2 for fixed θ and show that the remaining objective is $\frac{n}{2} \log\big(\frac{1}{n}\sum_i (y_i - \hat{y}_i)^2\big)$ plus a constant. Is the minimizer over θ still the least-squares fit?
   2. A network outputs a mean $\hat{y}(\mathbf{x})$ and a variance $\hat{\sigma}^2(\mathbf{x}) > 0$ for each input. Up to an additive constant, its NLL on the training set is

$$\sum_{i=1}^{n} \left[\frac{(y_i - \hat{y}(\mathbf{x}_i))^2}{2\hat{\sigma}^2(\mathbf{x}_i)} + \frac{1}{2} \log \hat{\sigma}^2(\mathbf{x}_i) \right].$$

      Show that this objective is unbounded below if the network can fit one training point exactly while shrinking its variance. Which property of the Gaussian mixture likelihood in Section 36.7 has the same cause?
   3. Compute the gradient of example i's term with respect to $\hat{y}(\mathbf{x}_i)$. Suppose that the predicted variance tracks the squared residual, $\hat{\sigma}^2(\mathbf{x}_i) \approx r_i^2$ with $r_i = y_i - \hat{y}(\mathbf{x}_i)$. How does the size of the gradient then depend on $|r_i|$, and how does that compare with squared error? What does this weighting do to the fit of the mean in regions where the fit is currently poor (Seitzer et al. 2022)?
4. **Losses that are not likelihoods.** Let $\phi(y, o)$ be a loss of a label y and a model output

o, and let $Z(o) = \sum_y e^{-\phi(y,o)}$. If $Z(o)$ does not depend on o, then $e^{-\phi(y,o)}/Z$ is a conditional distribution of y given o whose NLL is ϕ plus a constant, so minimizing ϕ is maximum likelihood.

1. For labels $y \in \{-1, +1\}$, show that the logistic loss $\phi(y, o) = \log(1 + e^{-yo})$ has $Z(o) = 1$, and identify the probability that the model assigns to $y = +1$.
2. Compute $Z(o)$ for the hinge loss $\phi(y, o) = \max(0, 1 - yo)$ of support vector machines, separately for $|o| \le 1$ and for $|o| > 1$. Conclude that the hinge loss is not the NLL, up to a constant, of any conditional distribution of y given o. Which term must be added to the hinge loss to obtain the NLL of $e^{-\phi(y,o)}/Z(o)$, and how does that term behave as $|o|$ grows?
3. For a real target y and a prediction $\hat{y}$, take any loss of the residual, $\phi(y, \hat{y}) = \rho(y - \hat{y})$ with $\int e^{-\rho(r)}\, dr < \infty$, such as the Huber loss. Show that $Z(\hat{y}) = \int e^{-\rho(y-\hat{y})}\, dy$ does not depend on $\hat{y}$. Why is the condition of this problem automatic for such regression losses but restrictive for classification losses?

5. code **Boundaries: separable data and rare outcomes.** Maximum likelihood misbehaves when the best parameter sits on the edge of its range.
   1. Two labeled points, $x = -1$ with $y = 0$ and $x = +1$ with $y = 1$, are fit by logistic regression $P(y = 1 \mid x) = \text{sigmoid}(wx)$ without a bias. Show that the NLL is $2\log(1 + e^{-w})$, that it decreases for every w and tends to 0, and that no finite w attains it. Which coin sequence of the text has the same problem, and what does $w \to \infty$ predict for a new point at $x = 0.1$?
   2. Add the prior $w \sim \mathcal{N}(0, \tau^2)$. Show that the MAP estimate solves $2\,\text{sigmoid}(-w) = w/\tau^2$, and solve it numerically for $\tau = 1$. Generalize: for any linearly separable data set the logistic NLL has infimum 0 and no minimizer, and a Gaussian prior restores a finite MAP estimate.
   3. For the coin, the skewness of $\hat{\theta} = n_H/n$ is $(1 - 2\theta)/\sqrt{n\theta(1-\theta)}$. Evaluate it for $\theta = 0.7$ and for $\theta = 0.95$ at $n = 400$, and find the n at which $\theta = 0.95$ reaches the skewness that $\theta = 0.7$ has at $n = 400$. For θ near 1, express the skewness through the expected number of tails $n(1 - \theta)$. Why is it that count, and not n, which decides whether the Gaussian limit is accurate?
   4. Simulate $20{,}000$ data sets of $n = 400$ flips (`rng.binomial(n, theta, 20000) / n`), and histogram $\sqrt{n}(\hat{\theta} - \theta)$ for $\theta = 0.7$ and $\theta = 0.95$ against $\mathcal{N}(0, \theta(1 - \theta))$. Describe how the $\theta = 0.95$ histogram departs from the curve.
6. **Gaussian versus Laplace shrinkage.** Observe one value $z \sim \mathcal{N}(\theta, 1)$ of an unknown scalar θ.
   1. With the prior $\theta \sim \mathcal{N}(0, \tau^2)$, show that the MAP estimate is $\tau^2 z/(1 + \tau^2)$. The pair (θ, z) is then jointly Gaussian, so the posterior of θ given z is Gaussian. Why does the MAP estimate therefore equal the posterior mean? For which z is it exactly zero?
   2. With the Laplace prior $p(\theta) \propto e^{-|\theta|/s}$ of Section 36.6.4.1, write the MAP objective. Solve it with the soft-thresholding formula: the minimizer of $\frac{1}{2}(z - \theta)^2 + \lambda|\theta|$ is $\text{sign}(z)\max(|z| - \lambda, 0)$. For which z is the estimate exactly zero?
   3. Keep the Laplace prior and let $z > 0$. Compare the posterior density at θ and at $-\theta$ for $\theta > 0$, and conclude that the posterior mean is strictly positive. What is the posterior probability that $\theta = 0$? What does this imply for reading the exact zeros of an L_1-regularized fit as evidence that a coefficient is zero?
7. **MAP in log-odds.** Parameterize the coin by its log-odds $\eta = \log \frac{\theta}{1-\theta}$, so that $\theta = \text{sigmoid}(\eta)$ and $\frac{d\theta}{d\eta} = \theta(1 - \theta)$. Keep the uniform prior $\text{Beta}(1, 1)$ on θ and the se-

quence with 9 heads and 4 tails.

1. Use the change-of-variables formula for densities, $p_\eta(\eta) = p_\theta(\theta)\,|d\theta/d\eta|$ of Section 36.2 to find the prior density of η that this belief implies. Show that it is the logistic density $e^{-\eta}/(1+e^{-\eta})^2$, the distribution of $\log \frac{U}{1-U}$ for a uniform U, and explain the result with inverse-transform sampling, without a calculation.
2. Find the MAP estimate of η and map it back to θ. Compare it with the MAP estimate in the θ parameterization, with the MLE $\log \frac{9}{4}$ of Section 36.6.4.3, and with Laplace's rule of succession.
3. Repeat (b) for a general prior $\mathrm{Beta}(a, b)$ and counts n_H, n_T. Which summary of the posterior over θ does the log-odds MAP estimate reproduce?
4. The MLE is equivariant, so the MLE of θ^2 is $\widehat{\theta}^2$. Show that its bias is $\mathbb{E}[\widehat{\theta}^2] - \theta^2 = \mathrm{Var}(\widehat{\theta}) = \theta(1-\theta)/n$, and evaluate it at $\theta = 9/13$, $n = 13$. Conclude that no estimator can be both equivariant under all smooth one-to-one reparameterizations and unbiased for all of them.

8. **Densities carry units.** ▣ A density value is not a probability: if x is measured in metres, $p(x)$ has units of inverse metres (Section 36.6.2.2).
   1. A Gaussian is fit by maximum likelihood to n lengths measured in metres and then refit to the same lengths in millimetres. How do $\widehat{\mu}$, $\widehat{\sigma}$, and the minimal NLL change?
   2. Model A is a density for positive data x. Model B is a Gaussian density for $\log x$. Use the change-of-variables formula of Section 36.2 to write model B's NLL for the data $x_1, \dots, x_n$. Which term must be added to B's NLL on the values $\log x_i$ before the two models can be compared?
   3. Images have integer pixel values $\mathbf{x} \in \{0, \dots, 255\}^D$. A density fit to such values can place a narrow spike on every grid point and so make its average NLL arbitrarily low, by the mechanism of problem 3b. Theis et al. (2016) instead fit a density q on $[0, 256)^D$ to the dequantized values $\mathbf{x} + \mathbf{u}$, with $\mathbf{u}$ uniform on $[0, 1)^D$. Show that $Q(\mathbf{x}) = \int_{[0,1)^D} q(\mathbf{x} + \mathbf{u})\, d\mathbf{u}$ is a probability distribution on the pixel grid. Use Jensen's inequality, as in Equation 36.92, to show that $-\log Q(\mathbf{x}) \le -\mathbb{E}_{\mathbf{u}}\big[\log q(\mathbf{x} + \mathbf{u})\big]$. What does this bound guarantee about an NLL reported on dequantized data?

9. **Symmetries of a ReLU network.** ▣ Consider the network $f_\theta(x) = \sum_{j=1}^{h} a_j\, \mathrm{ReLU}(w_j x)$ with a scalar input x and parameters $\theta = (a_1, w_1, \dots, a_h, w_h)$, and the regression model $y \sim \mathcal{N}(f_\theta(x), \sigma^2)$ with fixed σ^2. Identifiability is defined in Section 36.6.3.
   1. Show that replacing (a_j, w_j) by $(c\,a_j,\ w_j/c)$ for any $c > 0$ leaves f_θ unchanged. How many distinct parameter vectors with the same function does permuting the hidden units produce when no two units coincide?
   2. Show from Equation 36.83 that the Fisher information of this model is $I(\theta) = \mathbb{E}_x\big[\nabla_\theta f_\theta(x)\, \nabla_\theta f_\theta(x)^\top\big]/\sigma^2$. Differentiate the rescaling of (a) with respect to c at $c = 1$ to find a nonzero vector $\mathbf{v}$ with $I(\theta)\mathbf{v} = \mathbf{0}$. Which hypothesis behind Equation 36.85 fails?
   3. The Hessian entry of the mean squared error $\frac{1}{n}\sum_i (y_i - f_\theta(x_i))^2$ with respect to a_j is $\frac{2}{n}\sum_i \mathrm{ReLU}(w_j x_i)^2$. How does this entry change under the rescaling of (a)? Compute the entry for w_j, and explain why for a ReLU unit it contains no residual term (away from the kinks $w_j x_i = 0$). How does it change under the rescaling? What does this imply for measures of the flatness of a minimum that are based on Hessian eigenvalues (Dinh, Pascanu, et al. 2017)?

10. **A support that moves.** ▯ Let $x_1, \dots, x_n$ be independent and uniform on $[0, \theta]$ with unknown $\theta > 0$. This model violates one of the regularity conditions of Section 36.6.3.
    1. Write the likelihood as a function of θ and show that the MLE is $\hat{\theta} = \max_i x_i$. Why does setting a derivative to zero not find it?
    2. Show that $P(n(\theta - \hat{\theta}) > t) = (1 - \frac{t}{n\theta})^n$ for $0 \le t \le n\theta$, and find the limit as $n \to \infty$. At which rate in n does the error shrink, and is the limit Gaussian?
    3. On the support, the score is $s(x;\theta) = -1/\theta$. Compute $\mathbb{E}_\theta[s]$. Which step in the derivation of the mean-zero score fails (compare exercise 1 of Section 34.7)?
    4. The unbiased estimator $\frac{n+1}{n}\hat{\theta}$ has variance $\theta^2/(n(n+2))$. Compare it with the value $1/(n\,\mathbb{E}_\theta[s^2])$ that Equation 36.86 would give as a floor. Which step of the proof of the bound fails here?
11. [extended] **Misspecification and the sandwich.** ▯▯ This problem carries out the heuristic derivation of asymptotic normality when the two forms of the Fisher information differ. When no member of the model family is the data-generating law, the MLE converges to the pseudo-true parameter $\theta^\dagger$ that minimizes the population KL divergence (Section 36.6.3.5), and its spread is no longer I^{-1}/n (White 1982). Write $\mathbf{A} = -\mathbb{E}[\nabla^2_\theta \log p(x \mid \theta^\dagger)]$ and $\mathbf{B} = \mathbb{E}[s(x;\theta^\dagger)\, s(x;\theta^\dagger)^\top]$, with expectations under the data-generating law.
    1. Expand the first-order condition $\sum_i s(x_i;\hat{\theta}) = \mathbf{0}$ to first order around $\theta^\dagger$, and apply the central limit theorem to $\frac{1}{\sqrt{n}}\sum_i s(x_i;\theta^\dagger)$ to argue that
    $$\sqrt{n}\,(\hat{\theta} - \theta^\dagger) \xrightarrow{d} \mathcal{N}\big(\mathbf{0},\ \mathbf{A}^{-1}\mathbf{B}\mathbf{A}^{-1}\big). \tag{36.90}$$
    Why does $\mathbb{E}[s(x;\theta^\dagger)] = \mathbf{0}$ hold although the model is wrong? Show that the covariance reduces to the one in Equation 36.85 when the model is correct.
    2. Fit a Poisson model with rate λ to counts from a negative binomial distribution with mean m and shape α, whose variance is $m + m^2/\alpha$. Find $\hat{\lambda}$, $\lambda^\dagger$, $\mathbf{A}$, and $\mathbf{B}$. By which factor does the inverse Fisher information understate the variance of $\hat{\lambda}$?
    3. Fit the Poisson regression $\log \lambda(x) = w_0 + w_1 x$ with one feature by Newton's method, which for this model reads $\mathbf{w} \leftarrow \mathbf{w} + (\mathbf{Z}^\top \operatorname{diag}(\lambda_i)\mathbf{Z})^{-1}\mathbf{Z}^\top(\mathbf{y} - \lambda)$, with rows $(1, x_i)$ in $\mathbf{Z}$ and $\lambda_i = \lambda(x_i)$. An observation is now a pair (x_i, y_i), and $\log p(y_i \mid x_i, \mathbf{w})$ takes the place of $\log p(x \mid \theta)$ in $\mathbf{A}$ and $\mathbf{B}$; estimate both by sample averages at the fitted $\hat{\mathbf{w}}$. Generate $2{,}000$ data sets of $n = 500$ pairs with $x \sim \mathcal{N}(0,1)$, $\mathbf{w} = (1, 0.5)$, and negative-binomial counts with $\alpha = 2$: draw a rate from the Gamma distribution with shape α and mean $\lambda(x)$, then a Poisson count with that rate. Compare three standard errors of $\hat{w}_1$: from the inverse Fisher information, from the sandwich, and from the spread of $\hat{w}_1$ across data sets. Which fraction of the nominal 95% intervals of the first two covers $w_1 = 0.5$? Repeat with $\alpha = 10^6$.
    4. Exercise 6c of Section 36.4 shows that the Poisson fit still recovers $\mathbf{w}$ when the counts are overdispersed. Which part of the model would have to be wrong for $\mathbf{w}^\dagger$ to differ from $\mathbf{w}$? Which parameter does the sandwich interval of (c) cover in that case?

36.7 Latent Variables, EM, and the ELBO

Every likelihood in Section 36.6 was a product of terms that we could write down, one per observation, and its logarithm was a sum that we could differentiate term by term. That convenience rested on an assumption we never stated: that we see everything the model generates.

Often we do not. Suppose that 500 measurements come from two populations, 300 from one and 200 from the other, and that nobody recorded which measurement belongs to which. Had the labels survived, the fit would take one line per population: a fraction, a mean, a variance. Without them, the probability of a measurement is a sum over the two populations it might have come from. The logarithm of a sum does not split, every parameter appears in every term, and setting the gradient to zero leads to equations with no closed-form solution. The same happens whenever a model explains its data through something unobserved: the cluster of a customer, the topic of a document, the random code that a generator turns into an image. Such unobserved quantities are called *latent variables*.

The idea of this section is to guess what we cannot see, in probabilities rather than in hard labels, and to fit as if the guess were data. The surprise is that the guess need not be right for this to make sense. For *any* guess, the objective of the easy problem lies below the true log-likelihood, and it touches the log-likelihood when the guess is the posterior distribution of the hidden labels. This lower bound is the result to remember. Expectation–maximization climbs it exactly. Variational inference and variational autoencoders climb it approximately, because for them the posterior is out of reach.

The section has this one result and is short. We set up mixtures, derive the bound and measure its gap, and then run the resulting algorithm on the 500 points, once from a start where it succeeds and once from a start where it cannot.

36.7.1 Mixture Models

In symbols, a model with a latent variable z defines the joint distribution

$$p(x, z \mid \theta) = p(z \mid \theta)\, p(x \mid z, \theta),$$

of which we observe only x. The likelihood of an observation is then a *marginal*, obtained by summing the latent out (for a continuous latent, by integrating):

$$p(x \mid \theta) = \sum_{z} p(z \mid \theta)\, p(x \mid z, \theta). \tag{36.91}$$

The model's probability of x adds up all the ways of producing x, one for each value of the hidden variable.

The standard example is the **Gaussian mixture model** (GMM). The latent $z \in \{1, \dots, K\}$ picks a component with probability π_k, and the observation is then drawn from that component, $x \mid z{=}k \sim \mathcal{N}(\mu_k, \sigma_k^2)$. The NLL of the data is

$$\ell(\theta) = -\sum_{i=1}^{n} \log\Big(\sum_{k=1}^{K} \pi_k\, \mathcal{N}(x_i;\, \mu_k, \sigma_k^2)\Big), \qquad \theta = (\pi_k, \mu_k, \sigma_k)_{k=1}^{K}.$$

The difficulty is the sum *inside* the logarithm. The logarithm stops at the mixture and never reaches an individual Gaussian, whose exponent it would have turned into a harmless square. Contrast the *complete-data* problem. If an oracle revealed each z_i, the likelihood would factor by component, and maximum likelihood would collapse to estimators we already have. The weight π_k would be the fraction of points in component k, which is how the coin of Section 36.6 was estimated, $\hat{\theta} = n_H/n$. And μ_k, σ_k^2 would be the sample mean and variance of those points. The problem is hard only because the labels are missing, which suggests a plan: guess the missing labels in probabilities, then solve the easy problem.

The guess for one point is its posterior over components. Take a single observation $x = 0.5$ and two equally weighted components $\mathcal{N}(-1, 1)$ and $\mathcal{N}(1, 1)$. The densities are proportional to $e^{-(1.5)^2/2} = e^{-1.125}$ and $e^{-(0.5)^2/2} = e^{-0.125}$, so Bayes' rule gives the left component the posterior probability

$$\frac{e^{-1.125}}{e^{-1.125} + e^{-0.125}} = \frac{1}{1+e} = 0.27.$$

The point is split 27% to 73% between the two components, rather than assigned to one of them. A point at $x = 3$ would go 99.8% to the right, and one at $x = 0$ would be split evenly: the guess is confident far from the midpoint and undecided at it.

36.7.2 The Evidence Lower Bound

To turn this plan into an algorithm with a guarantee, we bound the log-likelihood from below by a quantity in which the logarithm reaches the joint density again. Let $q(z)$ be any distribution over the latent, our probabilistic guess. Insert it into the marginal and apply Jensen's inequality: the logarithm is concave, so $\log \mathbb{E}[\cdot] \geq \mathbb{E}[\log(\cdot)]$ (Section 35.3.3.1). Then

$$\log p(x \mid \theta) = \log \sum_z q(z)\, \frac{p(x, z \mid \theta)}{q(z)} \;\geq\; \sum_z q(z) \log \frac{p(x, z \mid \theta)}{q(z)} \;=:\; \mathcal{L}(q, \theta). \tag{36.92}$$

The sums run over the z with $q(z) > 0$. If q vanishes at some z to which the joint distribution gives positive probability, the first equality becomes $\geq$, and the bound still holds. The marginal $p(x \mid \theta)$ plays for the hidden z the role that the evidence $P(X)$, the denominator of Bayes' rule in Section 36.6.4, plays for the parameters: it is the probability of what we actually observed, with the unobserved quantity summed out. It is also called the *evidence*, and the right-hand side is the **evidence lower bound** ELBO. Expanding the fraction splits the bound into two parts,

$$\mathcal{L}(q, \theta) = \mathbb{E}_q\big[\log p(x, z \mid \theta)\big] + H(q).$$

The ELBO is the expected *complete-data* log-likelihood, the easy objective averaged over the guess, plus the entropy of the guess, which does not involve θ. How far does the bound fall short of the log-likelihood? By a KL divergence, and both EM and variational inference are built on this identity.

Proposition (the ELBO gap is a KL divergence). *For every θ and every q that puts no mass outside the support of the posterior $p(z \mid x, \theta)$,*

$$\log p(x \mid \theta) = \mathcal{L}(q, \theta) + D_{\mathrm{KL}}(q(z) \,\|\, p(z \mid x, \theta)). \tag{36.93}$$

Hence $\mathcal{L}(q, \theta) \le \log p(x \mid \theta)$, with equality if and only if q is the posterior $p(z \mid x, \theta)$.

Proof. Expand the KL term using Bayes' rule, $p(z \mid x, \theta) = p(x, z \mid \theta)/p(x \mid \theta)$:

$$D_{\mathrm{KL}}(q \,\|\, p(z \mid x, \theta)) = \mathbb{E}_q\left[\log \frac{q(z)\, p(x \mid \theta)}{p(x, z \mid \theta)}\right] = \log p(x \mid \theta) - \mathcal{L}(q, \theta),$$

where $\log p(x \mid \theta)$ leaves the expectation because it does not involve z. Rearranging gives Equation 36.93. Gibbs' inequality Equation 36.52 makes the KL term nonnegative and zero only when its two arguments agree. ■

For the point $x = 0.5$ of the previous subsection, the log-evidence is $\log p(x) = -1.42$. The lazy guess $q = (\frac{1}{2}, \frac{1}{2})$ gives an ELBO of -1.54, and the missing 0.12 nats are the KL divergence from $(\frac{1}{2}, \frac{1}{2})$ to the posterior $(0.27, 0.73)$. The bound falls short by the distance of the guess from the posterior, and by nothing else. Figure 36.30 draws the bound as a function of the parameters.

36.7.3 Expectation–Maximization

The proposition turns the hard problem into **coordinate ascent** on $\mathcal{L}(q, \theta)$: maximize over one of the two arguments while holding the other fixed, then swap, and repeat. For a data set $X = \{x_1, \dots, x_n\}$ of independent observations the bound applies example by example and sums, with one guess q_i per data point. The **expectation–maximization** (EM) algorithm alternates the two coordinates (Dempster et al. 1977):

- **E-step.** Hold $\theta^{(t)}$ fixed and maximize over the guesses. By the equality condition of Equation 36.93, the best q_i is the posterior, $q_i^{(t)}(z) = p(z \mid x_i, \theta^{(t)})$. This closes every gap and makes the bound *tight*: $\sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t)}) = \log P(X \mid \theta^{(t)})$.
- **M-step.** Hold the guesses fixed and maximize over the parameters:

  $$\theta^{(t+1)} = \operatorname*{argmax}_{\theta} \sum_i \mathbb{E}_{q_i^{(t)}}\big[\log p(x_i, z \mid \theta)\big],$$

  since the entropy terms do not involve θ. This is the easy complete-data objective, with soft assignments standing in for the oracle's labels.

Proposition (EM never decreases the likelihood). *Each EM iteration satisfies* $\log P(X \mid \theta^{(t+1)}) \ge \log P(X \mid \theta^{(t)})$.

Proof. Chain three facts:

$$\log P(X \mid \theta^{(t+1)}) \;\ge\; \sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t+1)}) \;\ge\; \sum_i \mathcal{L}(q_i^{(t)}, \theta^{(t)}) \;=\; \log P(X \mid \theta^{(t)}).$$

The first inequality is the ELBO bound Equation 36.92 at the new parameters; the second is the definition of the M-step; the equality is the tightness that the E-step achieved. ■

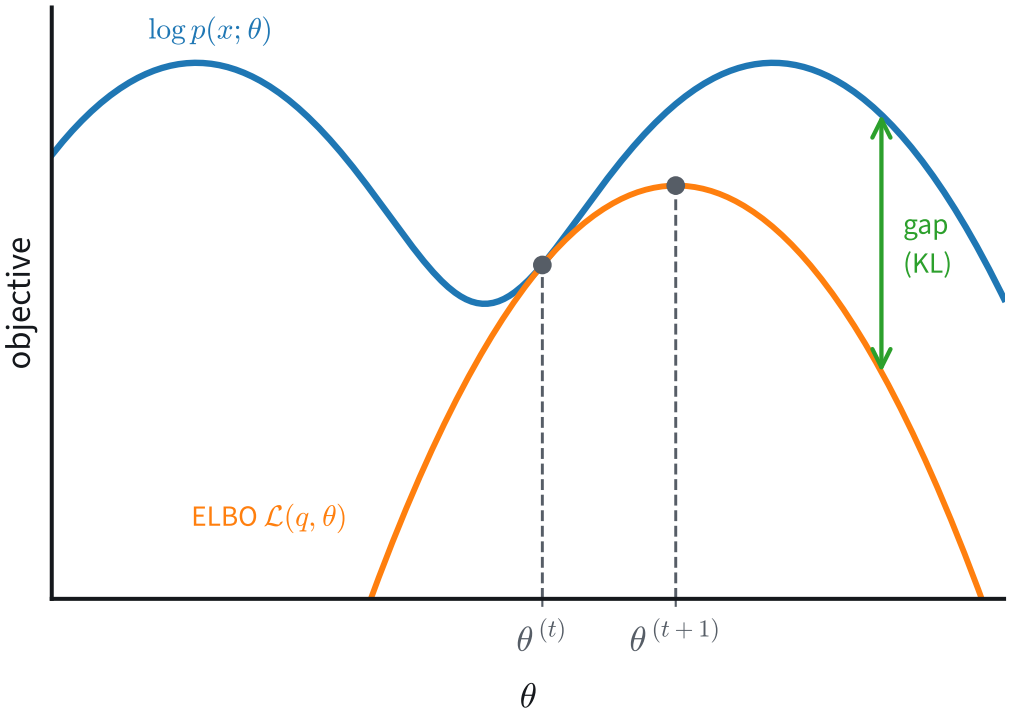


Figure 36.30: The evidence and its lower bound. For a fixed guess q, the ELBO $\mathcal{L}(q,\theta)$ runs below the evidence $\log p(x \mid \theta)$ at every θ, and the gap between the curves is the KL divergence from q to the posterior $p(z \mid x,\theta)$. The E-step of EM picks q to be the posterior at the current iterate $\theta^{(t)}$, closing the gap so the bound touches the evidence there; the M-step climbs the bound to its peak $\theta^{(t+1)}$. Because the bound touches the evidence at $\theta^{(t)}$ and never exceeds it anywhere, the evidence at $\theta^{(t+1)}$ is at least what it was at $\theta^{(t)}$.

For the Gaussian mixture both steps have closed forms. The E-step computes the **responsibilities**, the posterior probability that point i came from component k, as in the hand computation above,

$$r_{ik} = \frac{\pi_k \, \mathcal{N}(x_i;\, \mu_k, \sigma_k^2)}{\sum_j \pi_j \, \mathcal{N}(x_i;\, \mu_j, \sigma_j^2)}.$$

The M-step refits each component by responsibility-weighted counting and averaging. With effective counts $n_k = \sum_i r_{ik}$,

$$\pi_k = \frac{n_k}{n}, \qquad \mu_k = \frac{1}{n_k} \sum_i r_{ik} \, x_i, \qquad \sigma_k^2 = \frac{1}{n_k} \sum_i r_{ik} \, (x_i - \mu_k)^2. \tag{36.94}$$

These are the complete-data estimators, the fraction, the mean and the variance, softened: instead of voting for one component, each point spreads its single vote across the components in proportion to its responsibilities. If all components share one fixed variance σ^2 and we let $\sigma^2 \to 0$, the responsibilities become hard assignments to the nearest mean, and EM becomes the K-means algorithm.

The two steps fit in a dozen lines. We draw 300 points from $\mathcal{N}(-2, 0.8^2)$ and 200 from $\mathcal{N}(1.5, 0.6^2)$, and implement EM for a mixture of K Gaussians in one dimension; `em` returns the log-likelihood at every step together with the final parameters.

```
rng = np.random.default_rng()
x = np.concatenate([rng.normal(-2.0, 0.8, 300), rng.normal(1.5, 0.6, 200)])

def gauss(x, mu, sigma):
    return np.exp(-0.5 * ((x - mu) / sigma)**2) / (sigma * np.sqrt(2 * np.pi))

def em(x, pi, mu, sigma, steps):
    """EM for a 1-D Gaussian mixture: log-likelihood trace and final fit."""
    pi, mu, sigma = (np.array(v, dtype=float) for v in (pi, mu, sigma))
    lls = []
    for _ in range(steps):
        joint = pi * gauss(x[:, None], mu, sigma)  # (n, K)
        lls.append(np.log(joint.sum(axis=1)).sum())
        r = joint / joint.sum(axis=1, keepdims=True)  # E-step
        n_k = r.sum(axis=0)  # M-step
        pi, mu = n_k / len(x), (r * x[:, None]).sum(axis=0) / n_k
        sigma = np.sqrt((r * (x[:, None] - mu)**2).sum(axis=0) / n_k)
    return np.array(lls), pi, mu, sigma
```

The proposition says that the trace cannot go down. It says nothing about speed. We start from a poor guess, two heavily overlapping components at ± 0.5 with unit scale, run 30 steps, and print the first six values and the last.

```
lls, pi, mu, sigma = em(x, [0.5, 0.5], [-0.5, 0.5], [1, 1], 30)
print('steps 0-5:', lls[:6].round(1), ' step 29:', lls[-1].round(1))
print(f'pi = {pi.round(2)}, mu = {mu.round(2)}, sigma = {sigma.round(2)}')
```

```
steps 0-5: [-1298.1  -989.9  -942.   -871.   -848.6  -847. ]  step 29: -846.8
pi = [0.59 0.41], mu = [-2.04  1.54], sigma = [0.74 0.6 ]
```

The log-likelihood climbs by more than 400 nats, from about $-1{,}300$ to between -850 and -890 (the exact values depend on the draw). Nearly all of the climb happens at once: after six steps the trace is within about a nat of its final value. The recovered weights, means, and scales lie within about 0.2 of the generating values $(0.6, 0.4)$, $(-2, 1.5)$, and $(0.8, 0.6)$.

EM has four limits. The first is harmless. Relabeling the K components gives $K!$ equally good copies of every maximum. This *label switching* is the same non-identifiability as permuting the hidden units of a network, and it makes no maximum worse than another.

The second limit is that EM stops at the first stationary point it reaches. A symmetric start makes the point. Before running the cell, predict what EM does when both components begin with the same mean and scale.

```
lls2, pi2, mu2, sigma2 = em(x, [0.5, 0.5], [0, 0], [1, 1], 30)
print(f'log-likelihood {lls2[0]:.1f} -> {lls2[-1]:.1f}, '
      f'{lls[-1] - lls2[-1]:.1f} below the previous fit')
print(f'pi = {pi2.round(2)}, mu = {mu2.round(2)}, sigma = {sigma2.round(2)}')
print(f'mean of x = {x.mean():.2f}, standard deviation of x = {x.std():.2f}')
```

```
log-likelihood -1438.7 -> -1027.5, 180.7 below the previous fit
pi = [0.5 0.5], mu = [-0.59 -0.59], sigma = [1.89 1.89]
mean of x = -0.59, standard deviation of x = 1.89
```

The two components never separate. Their responsibilities are equal for every point, so the M-step keeps them identical, and EM settles on the single-Gaussian fit, the overall mean and standard deviation of the data, more than 100 nats below the two-cluster solution. Every fixed point of EM is a stationary point of the likelihood, not necessarily a maximum, so in practice EM is restarted from several random initializations. Random initialization breaks the same symmetry among the hidden units of a layer (Section 4.4): units that start equal receive equal gradients and stay equal.

The third limit is that the mixture likelihood is unbounded. Put the second component on the first data point, $\mu_2 = x_1$, with weight $\pi_2 = 1/n$ and scale ϵ. Its density at x_1 is $\frac{1}{n}(2\pi)^{-1/2}\epsilon^{-1}$, while the first component still explains the other points, so

$$\ell(\theta) \le \log\left(n\sqrt{2\pi}\right) + \log\epsilon + \text{const},$$

where the constant collects the other $n-1$ points. As $\epsilon \to 0$ the NLL goes to $-\infty$: the maximum likelihood "solution" is a spike on one data point. A floor on the variances, or a prior on them, removes the spike. With the set of parameters at least as good as the starting point then compact, every limit point of EM is a stationary point of the floored problem (Wu 1983).

The fourth limit is that the monotonicity proof needs the exact posterior in the E-step, and approximate E-steps lose the guarantee. A *variational autoencoder* (Kingma and Welling 2014) keeps the ELBO but replaces the per-point posterior by a neural network $q_\phi(z \mid x)$, trained jointly with the generator, because for a deep decoder the posterior has no closed form. It climbs the bound by stochastic gradients rather than by exact EM steps. Here the bound outgrows the algorithm that introduced it. The latent variable need not be a cluster label. Put the parameters themselves in its place, and the posterior that the E-step asks for

is the posterior over parameters, the object that Section 36.6 ended by asking for. It is rarely available either, and Section 36.8 is about what can be computed instead.

36.7.4 Summary

With hidden variables, the likelihood has a sum inside its logarithm, and the fit that would be counting and averaging if the variables were observed has no closed form. A probabilistic guess q for the hidden variables restores the easy problem as a lower bound: the ELBO lies below the log-evidence by the KL divergence from q to the posterior of the latent variable. EM alternates between closing that gap and climbing the bound. It never decreases the likelihood, but it stops at the first stationary point it reaches, and the mixture likelihood needs a floor on the variances to have a maximum at all.

36.7.5 Exercises

1. code **EM speed and missing information.** ⊠ Consider the mixture $p(x \mid \pi) = \pi f_0(x) + (1-\pi) f_1(x)$ with known components $f_0 = \mathcal{N}(\cdot\,; 0, 1)$ and $f_1 = \mathcal{N}(\cdot\,; \Delta, 1)$, in which only the weight π is unknown.
   1. Show that EM updates the weight by $\pi^{(t+1)} = M(\pi^{(t)})$ with $M(\pi) = \frac{1}{n}\sum_i r_i(\pi)$ and responsibilities $r_i(\pi) = \pi f_0(x_i)/p(x_i \mid \pi)$.
   2. Near a fixed point $\hat\pi = M(\hat\pi)$, each iteration multiplies the error by about $M'(\hat\pi)$. Define the observed information $I_{\text{obs}} = -\frac{d^2}{d\pi^2}\sum_i \log p(x_i \mid \pi)$ at $\hat\pi$, and the complete-data information $I_{\text{com}} = n/(\hat\pi(1-\hat\pi))$ that the labels would carry if they were observed (compare Equation 36.84). Show that
   $$M'(\hat\pi) = 1 - \frac{I_{\text{obs}}}{I_{\text{com}}}. \tag{36.95}$$
   Express both sides through the responsibilities and use $\sum_i r_i(\hat\pi) = n\hat\pi$. Dempster et al. (1977) prove the general form of this result: the rate of EM is set by the fraction of the information that the latent variables hide.
   3. Draw $n = 10{,}000$ points with $\pi = 0.3$ for each $\Delta \in \{0.5, 1, 2, 4\}$, run EM from $\pi^{(0)} = 0.5$, and compare the measured ratio of successive errors with the prediction of (b). With $\hat\pi$ taken from a long run, how many iterations does each Δ need to bring the error below 10^{-6}?
   4. What does the ratio $I_{\text{obs}}/I_{\text{com}}$ say about the standard error of $\hat\pi$, compared with a data set of the same size whose labels were observed?
2. **The bound by hand.** Take the single observation $x = 0.5$ and the two equally weighted components $\mathcal{N}(-1, 1)$ and $\mathcal{N}(1, 1)$ of the text.
   1. Compute $\log p(x)$, and the ELBO Equation 36.92 for the guess $q = (\frac{1}{2}, \frac{1}{2})$ and for the posterior $q = (0.27, 0.73)$. Check the two values quoted in the text.
   2. Verify Equation 36.93 for the uniform guess by computing the KL divergence directly.
   3. A hard guess puts all of q on the right component. Compute its ELBO and its gap. An algorithm that alternates hard guesses with M-steps still never decreases the ELBO. Why is its likelihood no longer guaranteed to increase?
3. **K-means as a limit.** Let all K components of a Gaussian mixture have equal weights and one fixed variance σ^2, so that only the means are fitted.
   1. Write the responsibilities r_{ik} as a softmax of $-(x_i - \mu_k)^2/(2\sigma^2)$ over k. What plays the role of the temperature? Show that, as $\sigma^2 \to 0$, the responsibility of

the nearest mean tends to 1 whenever the nearest mean is unique.

2. Show that the M-step Equation 36.94 then sets each mean to the average of the points assigned to it, and that the two steps together never increase $\sum_i (x_i - \mu_{z_i})^2$, where z_i is the component assigned to point i.
3. In the symmetric start of the text, both means begin at the same place. What does K-means do from such a start, and what does this say about the tie-breaking rule of an implementation?

36.8 Bayesian Computation

The Basics part trains each model to a single parameter vector, by maximum likelihood or, with a penalty, by the maximum a posteriori (MAP) estimate, the peak of the posterior. Section 36.6 put an error bar on such a vector. The predictions, however, still came from that one vector, although many others fit the data almost as well. How much do they disagree, and what should we predict when they do?

Take logistic regression with an intercept and a slope, fitted to twenty labeled points whose inputs lie between 0 and 4. At an input of -3, far below the data, the fitted model gives the label 1 a probability of four in a thousand. Vectors that fit the data almost as well disagree there by orders of magnitude. Their average, with each vector weighted by how plausible it is given the data, is almost five times larger. At an input of 2, inside the data, the two answers agree to within 0.01. For a rare outcome, such as a fraud flag or a diagnosis, a factor of several in its probability can decide what we do.

The weighted average is the Bayesian prediction, and computing it is as old a problem as Bayes' rule: the posterior divides by the evidence, an integral over all parameter vectors that rarely has a closed form. Laplace, in 1774, replaced the posterior of a proportion by the Gaussian that matches its peak. Metropolis and his colleagues, in 1953, let a random walk visit states in proportion to a distribution whose normalizer they did not know (Metropolis et al. 1953). In the 1990s the integral became an optimization problem. One idea runs through all of these methods: we can average over a distribution known only up to a constant, because each method uses only ratios of it, and the constant cancels in a ratio.

We ask four questions in turn. Can we replace the posterior by the Gaussian that matches its peak? Can we draw from a simpler distribution and correct each draw? Can a random walk spend its time in each region in proportion to the posterior? Can we optimize a simple distribution to be as close to the posterior as possible? Each of the first three answers repairs a failure of the one before; the fourth replaces sampling by optimization. A small posterior is the cleanest case of a distribution known only up to a constant; the book meets others, and Section 36.8.5 says where each idea returns. All four methods compute the **posterior predictive** distribution of a label $y_\star$ at a new input $\mathbf{x}_\star$,

$$p(y_\star \mid \mathbf{x}_\star, \mathcal{D}) = \int p(y_\star \mid \mathbf{x}_\star, \theta)\, p(\theta \mid \mathcal{D})\, d\theta, \tag{36.96}$$

which averages the predictions of all parameter vectors, each weighted by its posterior density given the data $\mathcal{D}$.

36.8.1 The Posterior of Bayesian Logistic Regression

To compare approximations we need a posterior that has no closed form but that we can still check. A prior is *conjugate* to a likelihood when the posterior stays in the prior's family, as a Beta prior on a coin stays Beta after any number of flips (Section 36.4). A Gaussian prior is not conjugate to the logistic likelihood: the posterior can be evaluated at any point but not integrated in closed form. With only two parameters, a fine grid still gives a reference for every approximation.

36.8.1.1 The Model and Its Unnormalized Posterior

Logistic regression gives each binary label $y_i \in \{0, 1\}$ a probability that depends on the input x_i through a line,

$$P(y_i = 1 \mid x_i, \theta) = \text{sigmoid}(\theta_0 + \theta_1 x_i) = \text{sigmoid}(\mathbf{z}_i^\top \theta), \qquad \text{sigmoid}(z) = \frac{1}{1 + e^{-z}},$$

where $\mathbf{z}_i = (1, x_i)^\top$. The probability crosses $\frac{1}{2}$ where the line crosses zero, at $x = -\theta_0/\theta_1$, and the slope θ_1 sets how sharply it rises there. We give the intercept and the slope independent Gaussian priors $\theta_j \sim \mathcal{N}(0, \tau^2)$, where τ is the prior standard deviation. Bayes' rule multiplies the likelihood of the n labels by the prior:

$$p(\theta \mid \mathcal{D}) \propto \left[\prod_{i=1}^{n} \text{sigmoid}(\mathbf{z}_i^\top \theta)^{y_i} \big(1 - \text{sigmoid}(\mathbf{z}_i^\top \theta)\big)^{1 - y_i} \right] \exp\left(-\frac{\|\theta\|^2}{2\tau^2} \right). \tag{36.97}$$

The bracket is the likelihood and the exponential is the prior. The missing constant is the **evidence** $p(\mathcal{D}) = \int p(\mathcal{D}, \theta)\, d\theta$, an integral over all parameter vectors that we cannot do in closed form. We write $\tilde{p}(\theta) = p(\mathcal{D}, \theta)$ for the product on the right, the *unnormalized posterior*, so that the posterior is $\tilde{p}/p(\mathcal{D})$.

The running data have $n = 20$ inputs drawn uniformly from $[0, 4]$ and labels drawn from the line with $\theta^\star = (-3, 1.5)$, which crosses $\frac{1}{2}$ at $x = 2$ and rises from $\text{sigmoid}(-3) \approx 0.05$ at $x = 0$ to $\text{sigmoid}(3) \approx 0.95$ at $x = 4$. The prior has $\tau^2 = 4$. Twenty points leave the posterior wide enough for each effect in this section to be visible. The twenty points are listed in the cell rather than drawn afresh, so that the numbers below can be quoted, and they were chosen with some care. We simulated 1,500 data sets from this recipe and kept the one closest to the medians of the quantities this section measures: the widths and the correlation of the posterior, the offset of its mean from its mode, and the ratio of the two predictions far from the data. What follows is therefore the typical case and neither a lucky nor an unlucky one. The data are fixed, but every sampler below still draws fresh random numbers, so their outputs are quoted as ranges. In the code, `log_joint` is $\log \tilde{p}$; it uses `logaddexp(0, z)` for $\log(1 + e^z)$ so that large logits do not overflow.

```
rng = np.random.default_rng()
n, prior_var = 20, 4.0                          # prior_var is tau^2
theta_star = np.array([-3.0, 1.5])

def sigmoid(z):
```

```
    return np.exp(-np.logaddexp(0.0, -z))

x = np.array([0.1, 0.3, 0.3, 0.4, 0.8, 1.4, 1.5, 1.6, 1.8, 2.1,
              2.2, 2.3, 2.6, 2.7, 3.0, 3.1, 3.4, 3.5, 3.6, 3.7])
y = np.array([0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1])
Z = np.column_stack([np.ones(n), x])          # rows z_i = (1, x_i)

def log_joint(theta, Z=Z, y=y, prior_var=prior_var):
    """log p(D, theta) up to a constant; theta has final dimension 2."""
    logits = np.einsum('...d,nd->...n', theta, Z)
    log_lik = (y * logits - np.logaddexp(0.0, logits)).sum(axis=-1)
    return log_lik - 0.5 * (theta**2).sum(axis=-1) / prior_var

print(f'{n} points, {y.sum()} of them labeled 1')
```

```
20 points, 10 of them labeled 1
```

36.8.1.2 The Gradient and the Hessian

Several methods below also use the slope and the curvature of $\log \tilde{p}$, so we derive them once. Differentiating $y_i \, \mathbf{z}_i^\top \theta - \log(1 + e^{\mathbf{z}_i^\top \theta})$ gives $(y_i - p_i)\, \mathbf{z}_i$ with $p_i = \text{sigmoid}(\mathbf{z}_i^\top \theta)$, and the prior contributes $-\theta/\tau^2$. Collecting the rows $\mathbf{z}_i^\top$ into the $n \times 2$ matrix $\mathbf{Z}$, the gradient is $\nabla \log \tilde{p}(\theta) = \mathbf{Z}^\top(\mathbf{y} - \mathbf{p}) - \theta/\tau^2$: each point pushes θ along $\mathbf{z}_i$ by its residual, and the prior pulls toward zero. Differentiating once more gives the negative Hessian

$$\mathbf{H}(\theta) = -\nabla^2 \log \tilde{p}(\theta) = \mathbf{Z}^\top \operatorname{diag}(p_i(1 - p_i))\mathbf{Z} + \tau^{-2}\mathbf{I}. \tag{36.98}$$

The data term is a Gram matrix of the inputs, each point weighted by $p_i(1-p_i) \le \frac{1}{4}$, so the points the model is unsure about count most. The prior adds τ^{-2} to every eigenvalue, so $\mathbf{H}$ is positive definite for every θ. The log posterior is therefore strictly concave, with a single maximum. The maximum is finite even when the data are linearly separable, because the log-likelihood is never positive while the prior term falls without bound in every direction.

```
def grad_log_joint(theta, Z=Z, y=y, prior_var=prior_var):
    """Gradient of log p(D, theta); one row per row of theta."""
    theta = np.atleast_2d(theta)
    return (y - sigmoid(theta @ Z.T)) @ Z - theta / prior_var

def neg_hessian(theta, Z=Z, y=y, prior_var=prior_var):
    """H(theta) = Z^T diag(p (1 - p)) Z + I / tau^2."""
    p = sigmoid(Z @ theta)
    return Z.T @ (Z * (p * (1 - p))[:, None]) + np.eye(2) / prior_var
```

For two parameters these derivatives are easy by hand. For a larger model we would write only $\log \tilde{p}$ and leave its gradient and Hessian to automatic differentiation.

Every method in this section touches the posterior only through $\tilde{p}$: its values, its gradient, or its Hessian.

36.8.1.3 The Posterior Mode and the Laplace Approximation

The cheapest approximation reuses an optimization we already know how to run. The peak of the posterior is the MAP estimate, the penalized fit of Section 36.6, and Newton's method $\theta \leftarrow \theta + \mathbf{H}(\theta)^{-1}\nabla \log \tilde{p}(\theta)$ reaches it in a few steps. One more Hessian turns the peak into a Gaussian.

We look for a Gaussian centered at the mode. Expanding $\log \tilde{p}$ to second order around θ_{MAP}, with $\mathbf{H} = \mathbf{H}(\theta_{\text{MAP}})$, gives

$$\log \tilde{p}(\theta) \approx \log \tilde{p}(\theta_{\text{MAP}}) - \tfrac{1}{2}(\theta - \theta_{\text{MAP}})^{\top}\mathbf{H}(\theta - \theta_{\text{MAP}}). \tag{36.99}$$

The linear term is missing because the gradient vanishes at the mode. The right side is, up to a constant, the log density of a Gaussian with mean θ_{MAP} and covariance $\mathbf{H}^{-1}$, so exponentiating gives the **Laplace approximation**

$$p(\theta \mid \mathcal{D}) \approx \mathcal{N}(\theta_{\text{MAP}}, \mathbf{H}^{-1}). \tag{36.100}$$

It needs $\log \tilde{p}$ to be twice differentiable at a mode in the interior of the parameter space, and $\mathbf{H}$ to be positive definite there; both hold here. A sharp peak, with large curvature, gives a small variance. The Gaussian matches the position and the curvature of the posterior's peak, and nothing else.

The smallest case is one-dimensional. Take the posterior Gamma$(4, 1)$, with density proportional to $\theta^3 e^{-\theta}$ on $\theta > 0$. Its log, $3\log\theta - \theta$, has derivative $3/\theta - 1$, which vanishes at the mode $\theta = 3$, and curvature $3/\theta^2 = \frac{1}{3}$ there. The Laplace approximation is $\mathcal{N}(3, 3)$, with standard deviation 1.73. The true posterior has mean 4 and variance 4, so the Gaussian sits too far left, is too narrow, and puts $\Phi(-\sqrt{3}) \approx 4.2\%$ of its mass below $\theta = 0$, where the posterior is zero (Figure 36.31).

For the coin with 9 heads and 4 tails under a uniform prior, the same computation returns the maximum likelihood estimate $9/13$ and its standard error 0.128 from Section 36.6.3.2: under a flat prior the Laplace approximation is the MLE with its error bar (Exercise 4).

The evidence never enters: $\log p(\mathcal{D})$ is an additive constant in $\log p(\theta \mid \mathcal{D})$, so it drops out of the curvature. Integrating the exponential of the right side of Equation 36.99 even estimates that constant, $p(\mathcal{D}) \approx \tilde{p}(\theta_{\text{MAP}})(2\pi)^{d/2}\det(\mathbf{H})^{-1/2}$ for d parameters (Exercise 8).

The approximation is local. It sees no skew, no heavy tail, no boundary and no second mode, and its center is the mode, not the mean. For a network with d parameters $\mathbf{H}$ has d^2 entries, so in practice one keeps only its diagonal, or only the Hessian of the last layer.

In code, ten Newton steps from zero find the mode, and one matrix inverse gives the covariance.

```
theta_map = np.zeros(2)
for _ in range(10):                                  # Newton's method
    theta_map += np.linalg.solve(neg_hessian(theta_map),
                                 grad_log_joint(theta_map)[0])
laplace_cov = np.linalg.inv(neg_hessian(theta_map))
laplace_sd = np.sqrt(np.diag(laplace_cov))
laplace_rho = laplace_cov[0, 1] / (laplace_sd[0] * laplace_sd[1])
```

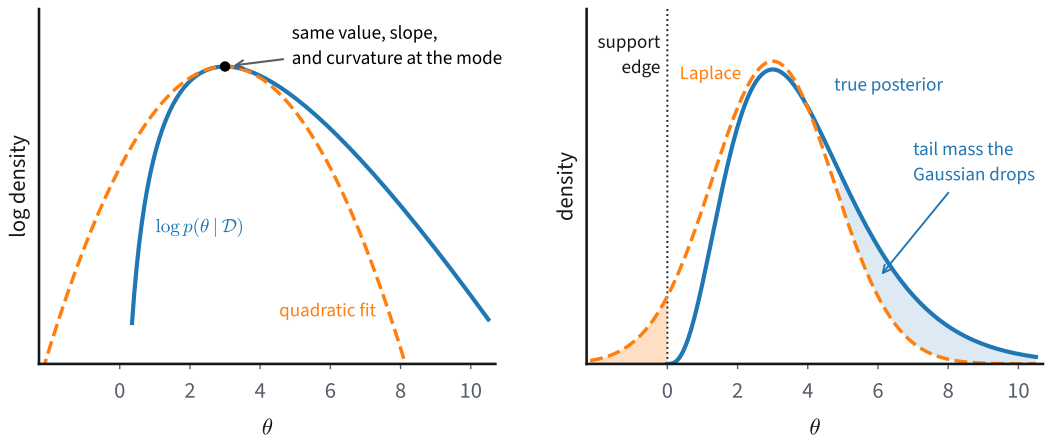


Figure 36.31: The Laplace approximation for a skewed one-dimensional posterior, the Gamma(4,1) density with mode 3. Left: the quadratic fit to the log density matches its value, slope and curvature at the mode and nothing else. Right: after exponentiating, the Laplace Gaussian $\mathcal{N}(3, 3)$ has the right peak but misses the skew. It puts 4.2% of its mass (shaded orange) below the support edge $\theta = 0$, where the posterior is zero, and drops tail mass on the right; the true mean is 4.

```
print('MAP                :', theta_map.round(3))
print('Laplace sd         :', laplace_sd.round(3))
print('Laplace correlation:', round(laplace_rho, 3))
```

```
MAP                : [-2.138  1.117]
Laplace sd         : [1.004 0.457]
Laplace correlation: -0.861
```

The mode is $(-2.14, 1.12)$, a shallower line than the one that generated the data. The Laplace standard deviations are 1.00 for the intercept and 0.46 for the slope, so the generating values $(-3, 1.5)$ lie a little under one standard deviation away in each coordinate. The correlation is strongly negative, -0.86: a steeper line must start lower to pass through the same points.

36.8.1.4 A Grid Reference

To judge the Laplace approximation we need the exact answer. With two parameters we can evaluate $\log \tilde{p}$ on an $m \times m$ grid, exponentiate after subtracting the maximum, and normalize the sum. In d dimensions such a grid needs m^d evaluations, so it serves only as a check in low dimension. Even the grid must know where to look. We center it at the MAP and extend it seven Laplace standard deviations in each direction, which leaves a negligible part of the posterior outside.

```
k, m = 7, 241                    # half-width in Laplace sds, points per axis
grid_axes = [np.linspace(c - k * s, c + k * s, m)
             for c, s in zip(theta_map, laplace_sd)]
B0, B1 = np.meshgrid(*grid_axes, indexing='ij')
```

```
grid_theta = np.stack([B0.ravel(), B1.ravel()], axis=1)
grid_logp = log_joint(grid_theta)
grid_weight = np.exp(grid_logp - grid_logp.max())
grid_weight /= grid_weight.sum()
post_mean = grid_weight @ grid_theta
centered = grid_theta - post_mean
post_cov = (centered * grid_weight[:, None]).T @ centered
post_sd = np.sqrt(np.diag(post_cov))
post_rho = post_cov[0, 1] / (post_sd[0] * post_sd[1])
print('grid mean       :', post_mean.round(3), '  MAP:', theta_map.round(3))
print('grid sd         :', post_sd.round(3), '  Laplace:', laplace_sd.round(3))
print('grid correlation:', round(post_rho, 3))
print('(mean - MAP)/sd :', ((post_mean - theta_map) / post_sd).round(2))
```

```
grid mean       : [-2.347  1.237]   MAP: [-2.138  1.117]
grid sd         : [1.042 0.48 ]   Laplace: [1.004 0.457]
grid correlation: -0.858
(mean - MAP)/sd : [-0.2   0.25]
```

The slope's posterior mean, 1.24, lies above its MAP, 1.12, by a quarter of a posterior standard deviation, and the intercept's mean, -2.35, lies below its MAP by a fifth. The Laplace standard deviations are 4 to 5% too small, and the correlation agrees to two digits. The widths are a minor matter. More serious is that the center of the Laplace Gaussian is in the wrong place: the posterior is skewed toward steep lines, and its mean is not its mode.

36.8.1.5 The Plug-in and the Posterior Predictive

We return to the question of the opening. The *plug-in* prediction $\text{sigmoid}(\mathbf{z}_\star^\top \theta_{\text{MAP}})$ uses the mode alone. The posterior predictive Equation 36.96 averages $\text{sigmoid}(\mathbf{z}_\star^\top \theta)$ over the grid, with the grid weights. We evaluate both at $x_\star \in \{-3, 2, 7\}$, below, inside and above the data, together with the range that holds the middle 90% of the posterior's predictions.

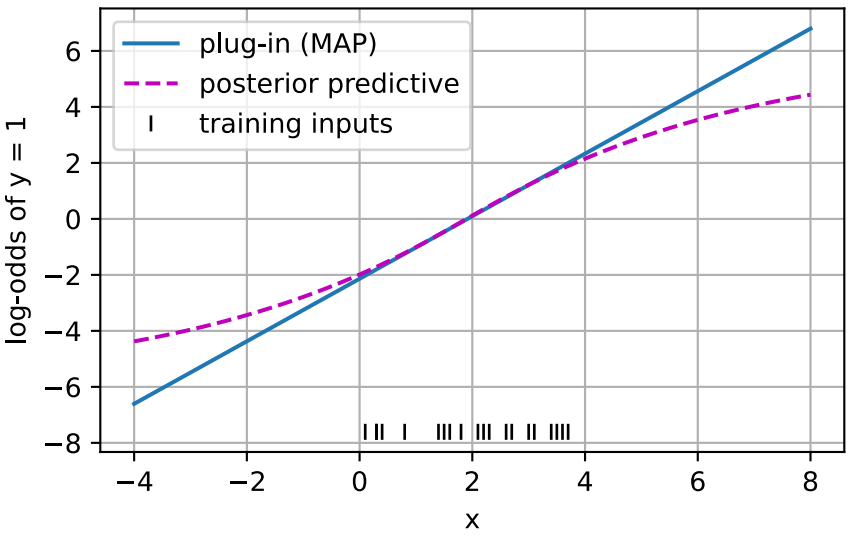
```
x_new = np.array([-3.0, 2.0, 7.0])
Z_new = np.column_stack([np.ones(len(x_new)), x_new])
grid_probs = sigmoid(grid_theta @ Z_new.T)     # each grid vector's answer
plug_in = sigmoid(Z_new @ theta_map)
predictive = grid_weight @ grid_probs
for j, x0 in enumerate(x_new):
    order = np.argsort(grid_probs[:, j])
    cdf = np.cumsum(grid_weight[order])
    lo, hi = grid_probs[order[np.searchsorted(cdf, [0.05, 0.95])], j]
    print(f'x={x0:+.0f}: plug-in {plug_in[j]:.4f}, predictive '
          f'{predictive[j]:.4f}, middle 90%: [{lo:.2g}, {hi:.4f}]')
a = grid_theta @ Z_new[0]                      # logits at x = -3
mu = grid_weight @ a
s2 = grid_weight @ (a - mu)**2
print(f'x=-3, log scale: plug-in {np.log(plug_in[0]):.2f}, '
      f'mean logit {mu:.2f}, plus s^2/2 {mu + s2 / 2:.2f},')
print(f'                 log E[e^a] {np.log(grid_weight @ np.exp(a)):.2f}, '
      f'predictive {np.log(predictive[0]):.2f}')
```

```
x=-3: plug-in 0.0041, predictive 0.0187, middle 90%: [3.8e-05, 0.0887]
x=+2: plug-in 0.5238, predictive 0.5292, middle 90%: [0.32, 0.7364]
x=+7: plug-in 0.9966, predictive 0.9826, middle 90%: [0.92, 1.0000]
x=-3, log scale: plug-in -5.49, mean logit -6.06, plus s^2/2 -3.19,
                 log E[e^a] -3.66, predictive -3.98
```

At $x_\star = 2$ the two predictions, 0.524 and 0.529, agree to within 0.01, and the plausible vectors keep their predictions between 0.32 and 0.74. At $x_\star = -3$ the middle 90% of the predictions run from 4×10^{-5} to 0.09, more than three orders of magnitude, and the posterior predictive, 0.019, is 4.6 times the plug-in, 0.0041. At $x_\star = 7$ the same holds for the probability of a 0 label, 0.017 against 0.003. Probabilities this close to 0 and 1 are easiest to compare as log-odds $\log \frac{P}{1-P}$, so the plot shows both predictions on that scale over a wider range than the data, with the training inputs marked along the bottom.

```
def log_odds(p):
    return np.log(p) - np.log1p(-p)

xs = np.linspace(-4, 8, 121)
Z_xs = np.column_stack([np.ones_like(xs), xs])
plug_xs = Z_xs @ theta_map                        # log-odds of the plug-in
pred_xs = log_odds(grid_weight @ sigmoid(grid_theta @ Z_xs.T))
rug = np.full(n, min(plug_xs.min(), pred_xs.min()) - 1)
d2l.plot([xs, xs, x], [plug_xs, pred_xs, rug],
         xlabel='x', ylabel='log-odds of y = 1', fmts=('-', 'm--', 'k|'),
         legend=['plug-in (MAP)', 'posterior predictive', 'training inputs'],
         figsize=(5, 3))
```



On the log-odds scale the plug-in is the straight line $\theta_0 + \theta_1 x$. Inside the data the posterior predictive lies on top of it; outside $[0, 4]$ it bends away from the line toward 0. Two effects produce the gap, and they point in opposite directions. First, the mode is not the mean. The posterior mean has a steeper slope than the MAP, and a steeper line moves the prediction away from $\frac{1}{2}$ far out on either side. Second, averaging sigmoid over a spread of logits moves

the prediction toward $\frac{1}{2}$, because the sigmoid flattens at its ends. On the far side of the data the plausible vectors with a shallow slope give probabilities that are large compared with the typical one, and they dominate the average.

Which effect wins, and by how much? A short calculation gives the size of the second. Where the probability is small, $\text{sigmoid}(a) \approx e^a$, so the predictive is roughly the average of e^a over the posterior of the logit $a = \mathbf{z}_\star^\top \theta$. If a were Gaussian with mean μ and variance s^2, that average would be $\mathbb{E}[e^a] = e^{\mu + s^2/2}$: averaging would lift the log-probability by $s^2/2$ above its value at the mean logit. At $x_\star = -3$ the grid gives $s^2/2 = 2.87$, which promises a factor of $e^{2.87} \approx 18$. We measured 4.6, a rise of only 1.51 on the log scale. The formula is wrong by almost a factor of two in the exponent, and it is worth seeing where the other 1.36 went. The last two lines of the cell walk from the plug-in to the predictive one step at a time:

step at $x_\star = -3$	log-probability	change
plug-in: the logit at the mode	-5.49	
the posterior mean of the logit	-6.06	-0.57
add $s^2/2$, as for a Gaussian logit	-3.19	$+2.87$
average e^a over the logit's actual distribution	-3.66	-0.47
average $\text{sigmoid}(a)$ instead of e^a: the predictive	-3.98	-0.32

Three corrections, all downward, separate 2.87 from 1.51. The first is the first effect above. The rise of $s^2/2$ starts from the mean logit, not from the plug-in's, and the mean logit lies 0.57 lower, because the posterior mean has the steeper slope. The second is that the logit is not Gaussian. The skew of the posterior gives it a long tail toward very negative values and a short one toward zero, and the average of e^a is made almost entirely by the tail toward zero; with less of that tail than a Gaussian would have, the average is lower by 0.47. The third is the approximation $\text{sigmoid}(a) \approx e^a$ itself. The vectors that dominate the average are the ones whose probabilities at $x_\star$ are no longer small, and there the sigmoid has begun to flatten and lies below e^a, which takes another 0.32. Together, $2.87 - 0.57 - 0.47 - 0.32 = 1.51$.

So the Gaussian formula is a guide and no more: it gives the direction of the effect and its order of magnitude. It also gives the scaling, and that part survives. The logit $\mathbf{z}_\star^\top \theta$ has variance $\text{Var}(\theta_0) + 2x_\star \text{Cov}(\theta_0, \theta_1) + x_\star^2 \text{Var}(\theta_1)$, which grows quadratically with the distance from the data, while the shift of the mean grows only linearly. So the second effect wins far out, and inside the data both effects are small (Exercise 5).

The larger number is not necessarily closer to the truth. The line that generated these data gives $\text{sigmoid}(-3 - 1.5 \cdot 3) = \text{sigmoid}(-7.5) \approx 0.0006$ at $x_\star = -3$, below both numbers: twenty points cannot pin the slope down, and the prior pulls it toward zero. That value does lie inside the printed 90% range, as it should for a typical data set. The posterior predictive is the probability that the model and the prior assign to the label once the disagreement among plausible vectors is counted. It is exact in one sense: if nature draws the parameters from the prior and the labels from the model, then among all the worlds that produce these

twenty points the label at $x_\star$ is 1 with this probability. The rest of the section asks how to compute such averages when no grid is possible.

36.8.2 Rejection and Importance Sampling

The Laplace Gaussian is centered at the mode and has no skew. Draws from the exact posterior would get every moment right at once, because their average estimates any posterior expectation. The two methods of this section turn draws from a simple distribution q, the *proposal*, into draws or weighted draws from the posterior, and they use only the ratio $\tilde{p}/q$.

36.8.2.1 Rejection Sampling

The simplest exact sampler keeps some draws and discards the rest. Choose a proposal q that we can sample and evaluate, and a constant M with $Mq(\theta) \geq \tilde{p}(\theta)$ for every θ, so that Mq is an envelope over the unnormalized posterior. Draw $\theta \sim q$ and keep it with probability

$$a(\theta) = \frac{\tilde{p}(\theta)}{M\, q(\theta)}. \tag{36.101}$$

The acceptance probability is the height of the target as a fraction of the height of the envelope at the drawn point. The kept draws then have density $p(\theta \mid \mathcal{D})$, and a draw is kept with probability $p(\mathcal{D})/M$. To see this, note that a draw lands in a small region $d\theta$ around θ with probability $q(\theta)\, d\theta$ and is kept with probability $a(\theta)$, so it lands there and is kept with probability $\tilde{p}(\theta)\, d\theta/M$. Integrating over θ, a draw is kept with probability $\int \tilde{p}/M = p(\mathcal{D})/M$. Dividing the first probability by the second, a kept draw has density $\tilde{p}(\theta)/p(\mathcal{D}) = p(\theta \mid \mathcal{D})$.

The evidence appears only in the acceptance rate, never in the decision to keep a draw. With the prior as the proposal, the ratio $\tilde{p}/q$ is the likelihood and M is its maximum, and the rule reads: draw parameters from the prior, and keep each with probability likelihood over best likelihood.

On the coin with 9 heads and 4 tails under a uniform prior, the likelihood $\theta^9(1-\theta)^4$ is largest at $\theta = 9/13$, where it equals $M = (9/13)^9(4/13)^4 \approx 3.3 \times 10^{-4}$. The evidence is $p(\mathcal{D}) = \int_0^1 \theta^9(1-\theta)^4\, d\theta = 9!\,4!/14! \approx 1.0 \times 10^{-4}$, so the acceptance rate is 0.305. With ten times the data, 90 heads and 40 tails, it is 0.10, and with a hundred times it is 0.03. The rate falls like $1/\sqrt{n}$ because the posterior's width does, while the prior's stays fixed. In d dimensions the posterior's volume shrinks like $n^{-d/2}$, and so does the acceptance rate. A simulation of the coin with 100,000 draws from the prior agrees.

```
N = 100_000
theta_coin = rng.random(N)                    # draws from the uniform prior
lik = theta_coin**9 * (1 - theta_coin)**4     # likelihood of 9 heads, 4 tails
M = (9 / 13)**9 * (4 / 13)**4                 # its maximum, at theta = 9/13
kept = theta_coin[rng.random(N) < lik / M]
print(f'acceptance {len(kept) / N:.3f}, mean {kept.mean():.3f}, '
      f'sd {kept.std():.3f}')
```

```
acceptance 0.304, mean 0.666, sd 0.118
```

Two limits remain. The bound M must be known, which it rarely is beyond toy models, and for a fixed amount of data per parameter the acceptance rate falls at least exponentially with the dimension.

36.8.2.2 Importance Weights

Rejection sampling discards most of its draws. We can keep every draw instead and give it the weight $w = \tilde{p}/q$ that rejection sampling used to flip a coin. To see that this works, write a posterior expectation as a ratio of two integrals and multiply and divide by q inside both:

$$\mathbb{E}_{p(\cdot\mid\mathcal{D})}[h(\theta)] = \frac{\int h\,\tilde{p}\,d\theta}{\int \tilde{p}\,d\theta} = \frac{\int h\,\frac{\tilde{p}}{q}\,q\,d\theta}{\int \frac{\tilde{p}}{q}\,q\,d\theta} = \frac{\mathbb{E}_q[w(\theta)h(\theta)]}{\mathbb{E}_q[w(\theta)]}, \qquad w(\theta) = \frac{\tilde{p}(\theta)}{q(\theta)}. \tag{36.102}$$

The denominator $\mathbb{E}_q[w]$ equals the unknown evidence $p(\mathcal{D}) = \int \tilde{p}$, and we never need its value, because the same constant scales the numerator. Replacing both expectations by averages over draws $\theta_1, \dots, \theta_N \sim q$ gives **self-normalized importance sampling**, $\sum_s w_s h(\theta_s) / \sum_s w_s$. By the law of large numbers it converges to the posterior expectation; at finite N it is slightly biased, because it is a ratio of two random averages. Products of many likelihood terms underflow, so the weights are computed in log space with the largest log-weight subtracted, as in the log-sum-exp trick. With the prior as the proposal the weight is the likelihood: each draw from the prior counts by how well it explains the data.

36.8.2.3 The Effective Sample Size

A weighted sample in which one draw carries all the weight is worth one draw, whatever N is. The **effective sample size** (ESS) of the normalized weights $\bar{w}_s = w_s / \sum_r w_r$,

$$N_{\text{eff}} = \frac{1}{\sum_{s=1}^{N} \bar{w}_s^2}, \tag{36.103}$$

equals N when all weights are equal and 1 when a single draw carries all of them. It estimates, roughly, the number of independent posterior draws that the weighted sample is worth, and we shall see two ways in which it misleads.

Proposition. As $N \to \infty$,

$$\frac{N_{\text{eff}}}{N} \to \frac{(\mathbb{E}_q[w])^2}{\mathbb{E}_q[w^2]} = \frac{1}{1 + \chi^2(p \,\|\, q)}, \qquad \chi^2(p \,\|\, q) = \int \frac{p^2}{q}\,d\theta - 1, \tag{36.104}$$

where p is the normalized posterior. With the same proposal, this limit is at least the acceptance rate $p(\mathcal{D})/M$ of rejection sampling.

Proof. Multiplying numerator and denominator of the definition by $1/N^2$ gives $N_{\text{eff}}/N = (\frac{1}{N}\sum_s w_s)^2 / (\frac{1}{N}\sum_s w_s^2)$, and the law of large numbers applies to both averages. Writing $w = p(\mathcal{D})\,p/q$ gives $\mathbb{E}_q[w^2] = p(\mathcal{D})^2 \int p^2/q$ and $\mathbb{E}_q[w] = p(\mathcal{D})$, hence the χ^2 form. Finally, $w \le M$ gives $\mathbb{E}_q[w^2] \le M\,\mathbb{E}_q[w]$, so the limit is at least $\mathbb{E}_q[w]/M = p(\mathcal{D})/M$. ■

The ESS is therefore a sample estimate of how far q is from p. For large N, importance sampling keeps at least as many effective draws as rejection sampling would accept, and it needs no bound M. On the coin with the uniform prior as the proposal, $\mathbb{E}_q[w^2] = \int_0^1 \theta^{18}(1-\theta)^8\, d\theta$, and the limit is 0.42, against the acceptance rate 0.305. We reweight the draws that the rejection sampler made, instead of discarding two thirds of them.

```
w = lik / lik.sum()                          # normalized weights
post_coin = w @ theta_coin
print(f'ESS fraction {1 / (w @ w) / N:.3f}, mean {post_coin:.3f}, '
      f'sd {np.sqrt(w @ (theta_coin - post_coin)**2):.3f}')
```

```
ESS fraction 0.420, mean 0.666, sd 0.118
```

The limit also shows how badly a proposal can fail. For a Gaussian target with standard deviation s_p and a Gaussian proposal at the same center with standard deviation $s_q = r s_p$, the limit is $\sqrt{2r^2 - 1}/r^2$ (Exercise 6). It is 0.83 at $r = 0.8$, 1 at $r = 1$, 0.66 at $r = 2$ and 0.46 at $r = 3$, and it is zero for $r \le 1/\sqrt{2}$, where $\mathbb{E}_q[w^2] = \infty$. A proposal that is too narrow fails far more abruptly than one that is too wide.

Two conditions make the identity Equation 36.102 hold: $q(\theta) > 0$ wherever $\tilde{p}(\theta) > 0$, and $\mathbb{E}_q|wh| < \infty$. A central limit theorem for the estimator, and with it error bars of order $1/\sqrt{N}$, holds when $\mathbb{E}_q[w^2]$ and $\mathbb{E}_q[w^2 h^2]$ are also finite. Even then the ESS can mislead, as Figure 36.32 shows.

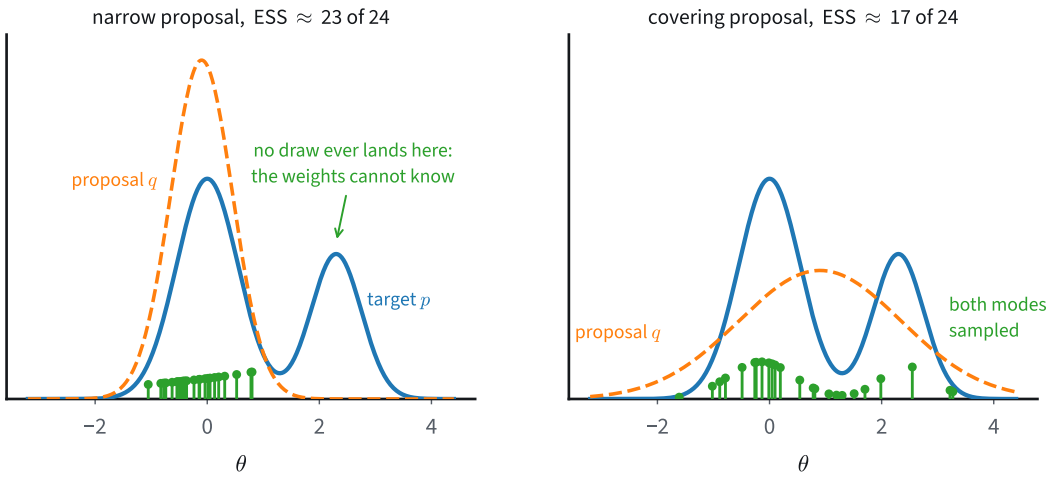


Figure 36.32: Importance sampling of a two-mode target with two proposals; stems show the normalized weights of 24 draws. Left: a narrow proposal never reaches the right-hand mode. Its weights are nearly equal (ESS 23 of 24), and its estimate is wrong. Right: a broader proposal samples both modes. Its weights are less equal (ESS 17 of 24), yet its estimate is far better. The ESS measures how unequal the weights are, not whether the proposal found the mass.

36.8.2.4 Proposals for the Logistic Posterior

Which proposal should we use on the logistic posterior? We compare two Gaussians, the prior with 20,000 draws and the Laplace approximation with 5,000, by the ESS as a fraction of the draws, the largest normalized weight, and the error of the estimated posterior mean in units of posterior standard deviations, measured against the grid.

```
def gaussian_logpdf(samples, mean, cov):
    diff = samples - mean
    _, logdet = np.linalg.slogdet(cov)
    maha = np.einsum('ni,ij,nj->n', diff, np.linalg.inv(cov), diff)
    return -0.5 * (len(mean) * np.log(2 * np.pi) + logdet + maha)

def importance(log_target, mean, cov, N):
    """Self-normalized importance sampling from a Gaussian proposal."""
    samples = rng.multivariate_normal(mean, cov, N)
    log_w = log_target(samples) - gaussian_logpdf(samples, mean, cov)
    w = np.exp(log_w - log_w.max())
    w /= w.sum()
    return w @ samples, 1 / (w @ w), w.max()

for name, mean, cov, N in [('prior', np.zeros(2), prior_var * np.eye(2), 20000),
                           ('Laplace', theta_map, laplace_cov, 5000)]:
    estimate, ess, w_max = importance(log_joint, mean, cov, N)
    error = np.abs(estimate - post_mean) / post_sd
    print(f'{name:8s}: ESS {ess / N:.1%} of {N}, largest weight '
          f'{w_max:.4f}, error in sds {error.round(3)}')
```

```
prior   : ESS 4.2% of 20000, largest weight 0.0025, error in sds [0.013 0.001]
Laplace : ESS 85.5% of 5000, largest weight 0.0030, error in sds [0.006 0.003]
```

With the prior as the proposal, the ESS is between about 1% and 10% of the draws, because most draws land where the likelihood is negligible. The Laplace proposal, aimed at the posterior, keeps most of its draws as effective, typically between about 55% and 95%. Both estimates usually agree with the grid mean to within a few hundredths of a posterior standard deviation, and rarely miss by more than about 0.15.

The Laplace proposal has a defect that the printed ESS does not show. Far out, the posterior's tails are those of the prior: along any ray $\theta = t\mathbf{v}$ the log-likelihood falls at most linearly in t (Exercise 10), while the log prior falls like $t^2/(2\tau^2)$. The integral $\mathbb{E}_q[w^2] = \int \tilde{p}^2/q$ for a Gaussian proposal $\mathcal{N}(\mathbf{m}, \mathbf{S})$ is then infinite as soon as one eigenvalue of $\mathbf{S}$ lies below $\tau^2/2 = 2$. Both eigenvalues of the Laplace covariance, about 1.17 and 0.05, are far below 2, so its weights have infinite variance in every direction, not along one thin axis.

The region where the weights blow up lies many Laplace standard deviations out, and most runs of 5,000 draws never visit it. In a few runs out of a hundred a single far draw takes a large share of the weight, and the ESS drops far below its usual value. A single run cannot reveal the defect; repeated runs, or the analysis of the tails, can. A Student-t proposal, whose tails are heavier than any Gaussian's, removes the defect (Exercise 10). So does the defensive mixture $q = 0.9\,\mathcal{N}(\theta_{\text{MAP}}, \mathbf{H}^{-1}) + 0.1$ prior: since q is at least a tenth of the prior, every weight is at most ten times the likelihood, which is at most one.

The evidence cancelled in the ratio of the two averages. What remains is the need for one

global proposal that covers the whole posterior, and in high dimension no simple distribution does (Exercise 11).

36.8.3 Markov Chain Monte Carlo

Instead of one proposal for the whole posterior, we can make local moves and let a random walk spend time in each region in proportion to its posterior mass. The walk is a Markov chain. Its next state depends on the current state only, through a *transition rule* $T(\theta' \mid \theta)$, the probability of moving from θ to θ'. A distribution π is *stationary* for the chain if states distributed as π are still distributed as π after one step:

$$\sum_{\theta} \pi(\theta)\, T(\theta' \mid \theta) = \pi(\theta') \quad \text{for every } \theta', \tag{36.105}$$

with an integral in place of the sum for continuous parameters. We want a chain whose stationary distribution is the posterior.

36.8.3.1 The Metropolis Rule

The Metropolis rule builds such a chain from any symmetric proposal $g(\theta' \mid \theta) = g(\theta \mid \theta')$, such as a Gaussian step $\theta' = \theta + \mathbf{L}\epsilon$. From the current state θ, propose $\theta' \sim g(\cdot \mid \theta)$. If $\tilde{p}(\theta') \geq \tilde{p}(\theta)$, move to θ'. Otherwise move with probability $\tilde{p}(\theta')/\tilde{p}(\theta)$, and stay at θ with the remaining probability. A rejection repeats the current state, and the repeat counts as a state of the chain (Figure 36.33).

Hastings (Hastings 1970) extended the rule to asymmetric proposals. The only change is a factor, $g(\theta \mid \theta')/g(\theta' \mid \theta)$, that corrects for a proposal that favors one direction. A proposal is then accepted with probability

$$\alpha(\theta, \theta') = \min\left(1, \frac{\tilde{p}(\theta')\, g(\theta \mid \theta')}{\tilde{p}(\theta)\, g(\theta' \mid \theta)}\right). \tag{36.106}$$

A move toward a state the proposal rarely offers is accepted more readily, and for a symmetric proposal the factor is 1. The evidence cancels in the ratio of the two values of $\tilde{p}$.

The smallest case has two states, with target $\pi = (\frac{1}{4}, \frac{3}{4})$, and a proposal that always names the other state. From state 1 the ratio is 3, so the chain always moves. From state 2 the ratio is $\frac{1}{3}$, so it moves with probability $\frac{1}{3}$ and stays otherwise. The transition matrix, with rows for the current state, and the check that π is stationary are

$$\mathbf{T} = \begin{pmatrix} 0 & 1 \\ \frac{1}{3} & \frac{2}{3} \end{pmatrix}, \qquad \left(\tfrac{1}{4}, \tfrac{3}{4}\right)\mathbf{T} = \left(\tfrac{3}{4}\cdot\tfrac{1}{3},\ \tfrac{1}{4} + \tfrac{3}{4}\cdot\tfrac{2}{3}\right) = \left(\tfrac{1}{4}, \tfrac{3}{4}\right).$$

If we counted only accepted states, the sequence would alternate 2, 1, 2, 1 and give $(\frac{1}{2}, \frac{1}{2})$. The repeated 2s after rejections give state 2 its correct weight (Figure 36.34).

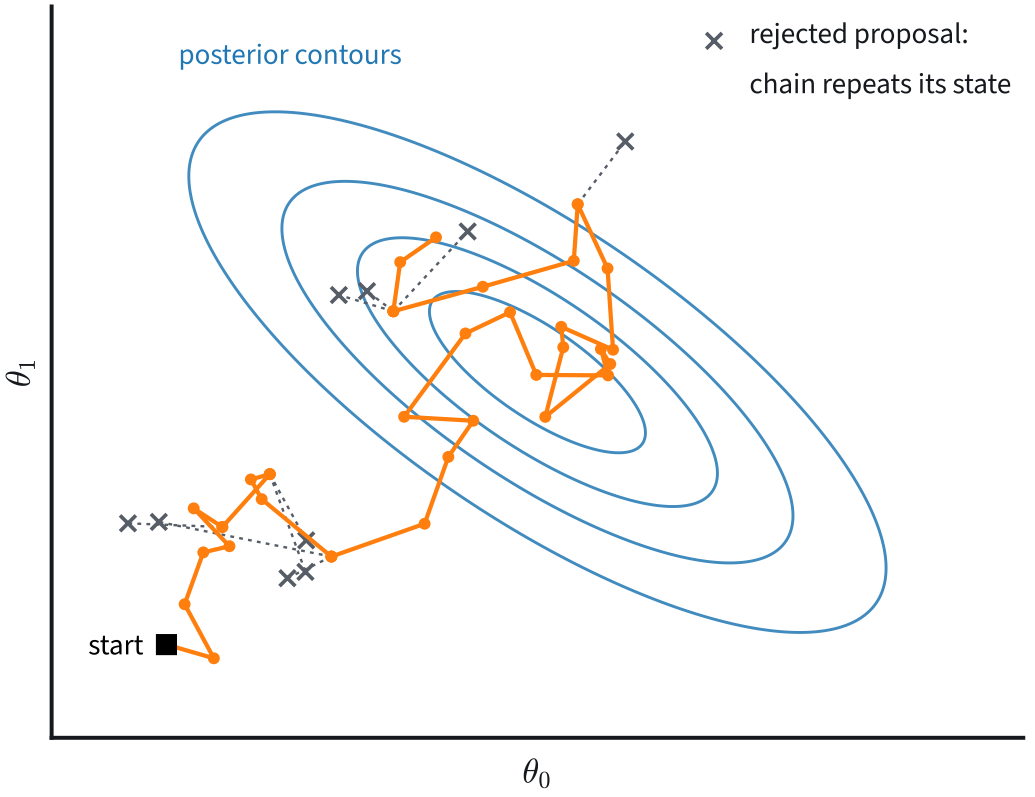


Figure 36.33: A Metropolis random walk on posterior contours. From the starting square, each step proposes a local move; uphill moves are always accepted (path), downhill moves only sometimes, and a rejected proposal (crosses) leaves the chain in place for another step. In the long run the time the walk spends in a region is proportional to its posterior mass.

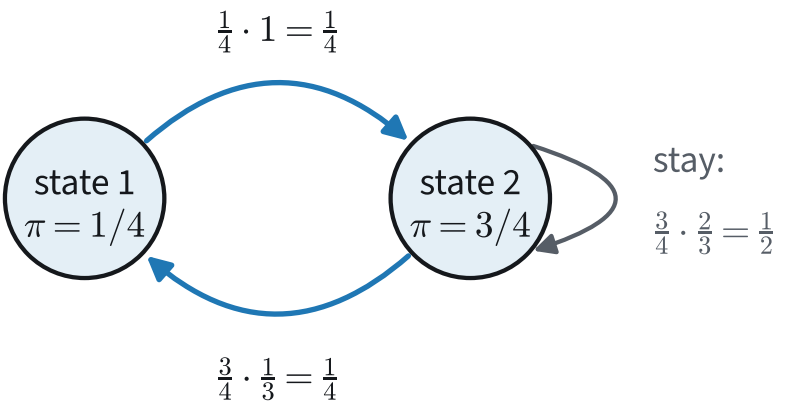


Figure 36.34: The two-state Metropolis chain with target $\pi = (1/4, 3/4)$. The flow from state 1 to state 2 is $\frac{1}{4} \cdot 1$ and the flow back is $\frac{3}{4} \cdot \frac{1}{3}$; both are $1/4$, so the occupation $(1/4, 3/4)$ does not change from step to step. The self-loop carries the rejected moves from state 2; counting only accepted states would give $(1/2, 1/2)$.

36.8.3.2 Detailed Balance and Stationarity

The figure shows why the chain keeps its distribution: the probability that flows from state 1 to state 2 in one step equals the probability that flows back, so no state gains or loses mass. This balance of flows holds for every pair of states, and it is the reason Metropolis works. If the reader keeps one result of this section, it should be this one.

Proposition. If

$$\pi(\theta)\, T(\theta' \mid \theta) = \pi(\theta')\, T(\theta \mid \theta') \quad \text{for all } \theta, \theta', \tag{36.107}$$

a condition called *detailed balance*, then π is stationary. The Metropolis–Hastings chain satisfies detailed balance with $\pi \propto \tilde{p}$.

Proof. Sum both sides over θ. The left side becomes the left side of Equation 36.105, and the right side is $\pi(\theta') \sum_{\theta} T(\theta \mid \theta') = \pi(\theta')$, because the chain moves somewhere from θ'. For the second claim, detailed balance is trivial when $\theta' = \theta$. When $\theta' \neq \theta$, a move happens only if it is proposed and accepted, so

$$\pi(\theta)\, T(\theta' \mid \theta) = \pi(\theta)\, g(\theta' \mid \theta)\, \alpha(\theta, \theta') = \min\big\{\pi(\theta)\, g(\theta' \mid \theta),\ \pi(\theta')\, g(\theta \mid \theta')\big\},$$

which is symmetric in the two states. The normalizer of π cancels inside α, so $\tilde{p}$ may be used in place of π. ■

The one step that is not routine is multiplying πg into the minimum, which turns the acceptance rule into a symmetric expression. For continuous parameters the rejection mass needs care (Exercise 1).

Stationarity alone does not make a chain converge to π from any start. The chain must also be able to reach every region of positive posterior mass, and it must not cycle deterministically between regions. Under these two conditions the long-run fraction of time it spends in a region equals the posterior mass of the region.

The rule takes a few lines of code. The Gaussian step has the shape of the Laplace covariance, $\mathbf{L}\mathbf{L}^{\top} = \mathbf{H}^{-1}$.

```
def metropolis(log_target, start, proposal_chol, n_steps):
    current = np.array(start, dtype=float)
    current_logp = log_target(current)
    states, accepted = np.empty((n_steps, len(current))), 0
    for t in range(n_steps):
        proposal = current + proposal_chol @ rng.standard_normal(len(current))
        proposal_logp = log_target(proposal)
        if np.log(rng.random()) < proposal_logp - current_logp:   # accept
            current, current_logp = proposal, proposal_logp
            accepted += 1
        states[t] = current            # a rejection repeats the state
    return states, accepted / n_steps
```

We run four chains of 12,000 steps from starts scattered twice as widely as the Laplace Gaussian, $\theta_{\text{MAP}} + 2\mathbf{L}\epsilon$, and propose steps $1.1\,\mathbf{L}\epsilon$. The first states of a chain still depend on where it started, so we discard the first 2,000 of each chain, the *warm-up*.

```
def run_chains(scale, n_chains=4, n_steps=12000, warmup=2000):
    L = np.linalg.cholesky(laplace_cov)
    chains, rates = [], []
    for _ in range(n_chains):
        start = theta_map + 2 * L @ rng.standard_normal(2)  # over-dispersed
        states, rate = metropolis(log_joint, start, scale * L, n_steps)
        chains.append(states[warmup:])
        rates.append(rate)
    return np.array(chains), np.array(rates)

chains, rates = run_chains(1.1)
print('acceptance rates:', rates.round(3))
```

```
acceptance rates: [0.523 0.529 0.533 0.526]
```

About half of the proposals are accepted in every chain. The acceptance rule, not the Gaussian, fixes the stationary distribution, so a poorly shaped proposal would slow the chain down but not bias it.

36.8.3.3 Diagnostics for Markov Chain Mixing

Consecutive states of a chain are correlated, and a chain starts wherever we put it. Before averaging its states we ask three questions and answer each with one tool.

Have the chains lost their starts and agreed? A trace plot of each chain shows sticking and drift, and $\widehat{R}$ puts a number on the disagreement. For several chains of N states each, let W be the average of their variances and B be N times the variance of their means. Then

$$\widehat{R}^2 = \frac{\frac{N-1}{N} W + \frac{1}{N} B}{W}. \tag{36.108}$$

The numerator estimates the posterior variance and overstates it while the chains still disagree, whereas W understates it while each chain has seen only part of the posterior. So $\widehat{R}$ compares two estimates of the same variance, and it falls toward 1 as the chains come to agree. The **split** $\widehat{R}$ applies the formula to the two halves of every chain, so that a single chain that drifts looks like two chains that disagree. A value within 0.01 of 1 is necessary for trusting the chains, but not sufficient: chains that all miss the same mode agree with each other.

How many independent draws are the states worth? Let ρ_k be the correlation between states k steps apart. The integrated autocorrelation time $\kappa = 1 + 2\sum_{k \geq 1} \rho_k$ is the number of states that carry as much information as one independent draw, so c chains of N states each are worth $N_{\text{eff}} = cN/\kappa$ draws, the same symbol and the same meaning as for the importance weights.

The smallest case is the autoregressive sequence of order one, AR(1), $x_t = \phi x_{t-1} + \varepsilon_t$ with independent noise ε_t and $|\phi| < 1$: each state keeps a fraction ϕ of the previous one. Its autocorrelations are $\rho_k = \phi^k$, so

$$\kappa = 1 + 2\sum_{k \geq 1} \phi^k = \frac{1 + \phi}{1 - \phi}.$$

With $\phi = 0.9$, $\kappa = 19$, and 10,000 states are worth about 530 independent draws; with $\phi = 0.99$, $\kappa = 199$, and they are worth about 50.

In practice the estimated far autocorrelations are mostly noise, so the code sums them in pairs $\rho_{2k} + \rho_{2k+1}$ and stops at the first negative pair. Production software refines both tools, with rank-normalized $\widehat{R}$ and separate effective sample sizes for the bulk and the tails.

How wrong is the estimated mean? By the central limit theorem for the chain, the **Monte Carlo standard error** (MCSE) of a posterior mean is the posterior standard deviation divided by $\sqrt{N_{\text{eff}}}$.

```
def split_rhat(chains):
    """Split R-hat per coordinate; chains has shape (c, N, d)."""
    half = chains.shape[1] // 2
    halves = np.concatenate([chains[:, :half], chains[:, half:2 * half]])
    W = halves.var(axis=1, ddof=1).mean(axis=0)
    B = half * halves.mean(axis=1).var(axis=0, ddof=1)
    return np.sqrt(((half - 1) / half * W + B / half) / W)

def autocorrelation(x):
    """rho_k of one sequence for k = 0, ..., N - 1, by FFT."""
    x = x - x.mean()
    f = np.fft.rfft(x, n=2 * len(x))
    acov = np.fft.irfft(f * np.conj(f))[:len(x)]
    return acov / acov[0]
```

```
def chain_ess(chains):
    """c N / kappa per coordinate, summing rho in pairs (Geyer)."""
    c, N, d = chains.shape
    ess = np.empty(d)
    for j in range(d):
        rho = np.mean([autocorrelation(chain[:, j]) for chain in chains],
                      axis=0)
        pairs = rho[:N - 1:2] + rho[1:N:2]        # rho_2k + rho_2k+1
        negative = np.flatnonzero(pairs < 0)
        n_pairs = negative[0] if len(negative) else len(pairs)
        ess[j] = c * N / (-1 + 2 * pairs[:n_pairs].sum())
    return ess
```

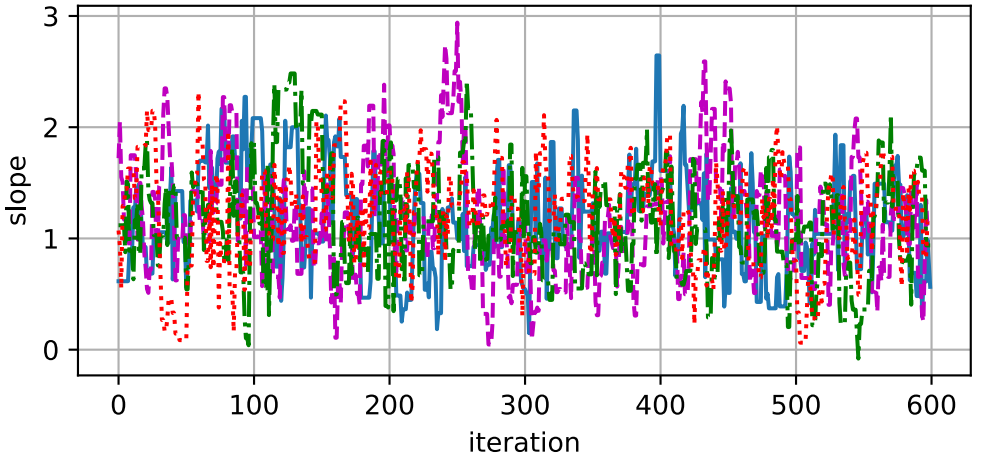
Now the test. If the error bar is right, the MCMC mean of our four chains should differ from the grid mean by about one MCSE. We also compare the measured κ of the slope with the AR(1) formula evaluated at the lag-1 autocorrelation ρ_1.

```
flat = chains.reshape(-1, 2)
ess = chain_ess(chains)
mcmc_mean, mcmc_sd = flat.mean(axis=0), flat.std(axis=0, ddof=1)
mcse = mcmc_sd / np.sqrt(ess)
rho1 = np.mean([autocorrelation(chain[:, 1])[1] for chain in chains])
print('split R-hat       :', split_rhat(chains).round(3))
print('ESS               :', ess.round().astype(int), 'of', len(flat))
print('(MCMC - grid)/MCSE:', ((mcmc_mean - post_mean) / mcse).round(1))
print(f'slope: rho_1 {rho1:.2f}, AR(1) kappa {(1 + rho1) / (1 - rho1):.1f},'
      f' measured kappa {len(flat) / ess[1]:.1f}')
```

```
split R-hat       : [1.001 1.001]
ESS               : [3798 3697] of 40000
(MCMC - grid)/MCSE: [ 0.6 -0.4]
slope: rho_1 0.81, AR(1) kappa 9.4, measured kappa 10.8
```

A trace plot of the slope's first 600 states after warm-up completes the checks.

```
d2l.plot(np.arange(600), [chain[:600, 1] for chain in chains],
         xlabel='iteration', ylabel='slope', figsize=(6, 2.5))
```



The split $\widehat{R}$ is within 0.01 of 1, and the four traces overlap. The ESS is about a tenth of the 40,000 states, so $\kappa \approx 10$: the lag-1 autocorrelation is about 0.8, and the AR(1) formula predicts a κ close to the measured one. The MCMC mean differs from the grid mean by about one MCSE. This is the strongest check in the section, because it tests the error bar against the truth.

36.8.3.4 Proposal Scale and Mixing Failures

The acceptance rate is the number a sampler reports most readily, and on its own it says little. We rerun the chains with the proposal scale set to 0.05 and to 10 in place of 1.1. Before reading the output, guess which of the two runs is worth fewer draws.

```
for scale in [0.05, 10]:
    bad_chains, bad_rates = run_chains(scale)
    print(f'scale {scale:5}: acceptance {bad_rates.mean():.3f}, max R-hat '
          f'{split_rhat(bad_chains).max():.2f}, '
          f'min ESS {chain_ess(bad_chains).min():.0f}')
```

```
scale  0.05: acceptance 0.975, max R-hat 1.13, min ESS 43
scale    10: acceptance 0.020, max R-hat 1.00, min ESS 689
```

With tiny steps almost every proposal is accepted, about 97%. Yet $\widehat{R}$ is well above 1.01 and the 40,000 states are worth a few dozen draws, because each accepted step moves the chain only a short distance. With huge steps about 2% of the proposals are accepted, and the chain repeats each state many times, so the ESS is a few hundred. The best scale lies between (Exercise 2). A random walk moves uphill only by chance, whereas a proposal that follows the gradient of $\log \tilde{p}$ moves uphill by construction. Such proposals come from Langevin dynamics (Section 38.5.2), and their Metropolis-adjusted version is Exercise 12.

36.8.4 Variational Inference

Sampling estimates each expectation anew and carries its Monte Carlo error. *Variational inference* replaces the integration by one optimization. It chooses a simple family of distributions q_ψ with parameters ψ and optimizes ψ so that q_ψ is as close to the posterior as

possible. Expectations are then computed under q_ψ, in closed form or by cheap sampling.

36.8.4.1 The Evidence Lower Bound

We need an objective that measures closeness to the posterior although the posterior's normalizer is unknown. We already have one. In Section 36.7 a guess q about a latent variable gave a lower bound on the log evidence, and the gap was a KL divergence Equation 36.93. Let the parameters θ play the role of the latent variable and the data set $\mathcal{D}$ that of the observation. The identity then reads

$$\log p(\mathcal{D}) = \underbrace{\mathbb{E}_{q_\psi}\big[\log p(\mathcal{D},\theta) - \log q_\psi(\theta)\big]}_{\mathcal{L}(\psi)} + D_{\mathrm{KL}}(q_\psi \,\|\, p(\cdot \mid \mathcal{D})). \tag{36.109}$$

Because the KL divergence is non-negative, $\mathcal{L}$ is the **evidence lower bound** (ELBO) of q_ψ,

$$\mathcal{L}(\psi) = \mathbb{E}_{q_\psi}[\log p(\mathcal{D},\theta) - \log q_\psi(\theta)] \le \log p(\mathcal{D}). \tag{36.110}$$

Raising $\mathcal{L}$ lowers the KL divergence by the same amount, because their sum is the constant $\log p(\mathcal{D})$. Maximizing the ELBO over ψ therefore minimizes the KL divergence, and the unknown normalizer drops out as an additive constant. Every term inside the expectation can be evaluated pointwise, and the expectation can be estimated by sampling from q_ψ.

36.8.4.2 The Direction of the KL Divergence

Variational inference minimizes $D_{\mathrm{KL}}(q\|p) = \int q \log(q/p)$, the *reverse* KL divergence, with q first. Its integrand is large wherever $q > 0$ and $p \approx 0$, so q retreats from regions the posterior does not cover and tends to settle on one mode: reverse KL is *mode-seeking*. The *forward* divergence $D_{\mathrm{KL}}(p\|q) = \int p \log(p/q)$ is large wherever $p > 0$ and $q \approx 0$, so it pushes q to cover all of the posterior's mass: forward KL is *mass-covering*. For a Gaussian family its minimizer matches the posterior's mean and covariance, but it takes its expectation under the posterior, which we cannot sample; the reverse direction needs only draws from q. Figure 36.35 shows both on a target with two modes.

36.8.4.3 A Mean-Field Gaussian Fit

The simplest family is the *mean-field* Gaussian $q = \mathcal{N}(\mathbf{m}, \mathrm{diag}(\mathbf{s}^2))$, which treats the coordinates as independent. To follow the gradient of the ELBO we write a draw as $\theta = \mathbf{m} + \mathbf{s} \odot \epsilon$ with $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. The distribution of ϵ does not depend on $\mathbf{m}$ and $\mathbf{s}$, so the gradient moves inside the expectation (Section 34.7.3). The entropy term $-\mathbb{E}_q[\log q]$ equals $\sum_j \log s_j$ plus a constant, and the gradients are

$$\nabla_{\mathbf{m}} \mathcal{L} = \mathbb{E}[\nabla_\theta \log p(\mathcal{D},\theta)], \qquad \nabla_{\log \mathbf{s}} \mathcal{L} = \mathbb{E}[\nabla_\theta \log p(\mathcal{D},\theta) \odot \mathbf{s} \odot \epsilon] + \mathbf{1}, \tag{36.111}$$

where $\mathbf{1}$ is the derivative of the entropy $\sum_j \log s_j$ with respect to $\log \mathbf{s}$. The first gradient moves the mean uphill on average; the second widens q until the entropy's push is balanced by the posterior's curvature. Before running it, we can predict the answer on a Gaussian target.

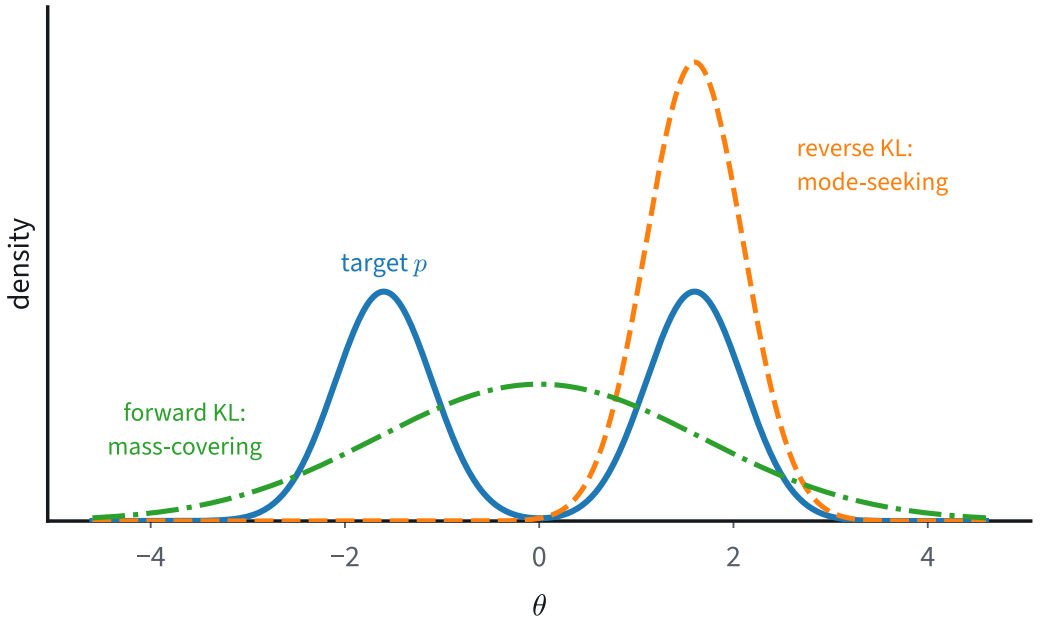


Figure 36.35: The direction of the KL divergence decides how a too-simple approximation fails. For a two-mode target, the best reverse-KL Gaussian, the one variational inference finds, settles on a single mode and understates the uncertainty; the best forward-KL Gaussian covers the mass of both modes and puts its bulk where the target is small.

Proposition. For the target $p = \mathcal{N}(\mu, \Lambda^{-1})$ with precision matrix Λ, the mean-field Gaussian that minimizes $D_{\mathrm{KL}}(q\|p)$ has mean μ and variances $s_j^2 = 1/\Lambda_{jj}$, the *conditional* variances of the target. For two coordinates with correlation ρ, each s_j is $\sqrt{1-\rho^2}$ times the marginal standard deviation.

Proof. Up to a constant, $D_{\mathrm{KL}}(q\|p) = -\sum_j \log s_j + \frac{1}{2}\mathbb{E}_q[(\theta-\mu)^\top \Lambda(\theta-\mu)]$. Under q the coordinates are independent with means m_j and variances s_j^2, so the cross terms of the fluctuations have mean zero and the expectation is $\sum_j \Lambda_{jj} s_j^2 + (\mathbf{m}-\mu)^\top \Lambda(\mathbf{m}-\mu)$. The part in $\mathbf{m}$ is a positive definite quadratic, minimized at $\mathbf{m} = \mu$. Setting the derivative in s_j to zero, $-1/s_j + \Lambda_{jj} s_j = 0$, gives $s_j^2 = 1/\Lambda_{jj}$. For $d = 2$, inverting the covariance gives $1/\Lambda_{jj} = \sigma_j^2(1-\rho^2)$, where σ_j is the marginal standard deviation. ■

A factorized q cannot tilt its axes to follow a correlation, and under the reverse KL it also shrinks to the width of a slice through the posterior rather than the width of the whole posterior (Figure 36.36). The shrinkage is a property of the mean-field family under the reverse KL: a full-covariance Gaussian fitted by the same objective to a Gaussian target is exact. In the running example $\rho = -0.86$, so the mean-field standard deviations should be about $\sqrt{1-0.86^2} = 0.51$ times the true ones, about half.

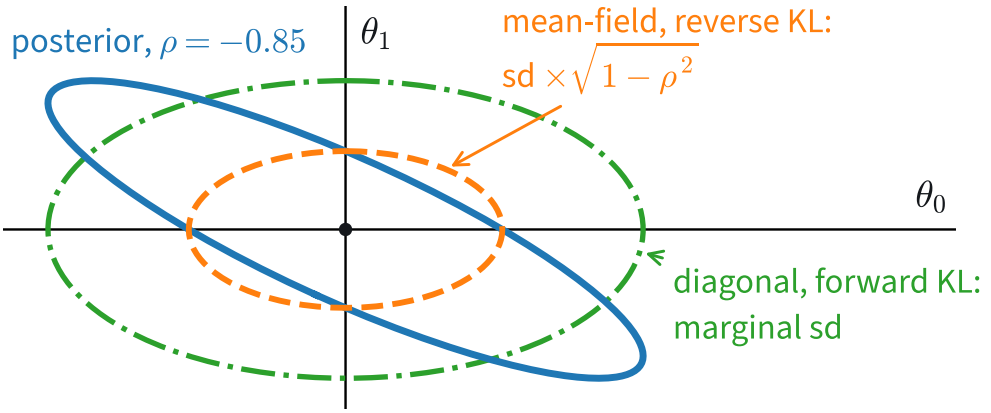


Figure 36.36: Diagonal Gaussians fitted to a correlated Gaussian posterior with standard deviations $(1, 0.5)$ and correlation $\rho = -0.85$, drawn as 2-sd ellipses. A factorized family cannot tilt. Under the reverse KL (orange) it also shrinks to the conditional width, $\sqrt{1-\rho^2} \approx 0.53$ times the marginal standard deviation; under the forward KL (green) it keeps the marginal width and covers mass the posterior does not have.

The fit starts at the Laplace Gaussian and takes 3,000 steps of stochastic gradient ascent on Equation 36.111, with 256 draws per step and a step size that decays like $1/\sqrt{t}$. We report the average of the last 1,000 iterates, which removes most of the optimization noise, next to the prediction of the proposition from the grid's own ρ.

```
m_vi, log_s_vi = theta_map.copy(), np.log(laplace_sd)
m_avg, log_s_avg = np.zeros(2), np.zeros(2)
n_steps, n_avg = 3000, 1000
```

```
    for t in range(1, n_steps + 1):
        eps = rng.standard_normal((256, 2))
        s_vi = np.exp(log_s_vi)
        g = grad_log_joint(m_vi + s_vi * eps)        # at theta = m + s * eps
        lr = 0.05 / np.sqrt(t)
        m_vi += lr * g.mean(axis=0)
        log_s_vi += lr * ((g * eps * s_vi).mean(axis=0) + 1)
        if t > n_steps - n_avg:                      # average the last iterates
            m_avg += m_vi / n_avg
            log_s_avg += log_s_vi / n_avg
    s_avg = np.exp(log_s_avg)
    print('             mean             sd')
    print('grid      ', post_mean.round(3), post_sd.round(3))
    print('Laplace   ', theta_map.round(3), laplace_sd.round(3))
    print('mean-field', m_avg.round(3), s_avg.round(3))
    print('predicted sd, grid sd * sqrt(1 - rho^2):',
          (post_sd * np.sqrt(1 - post_rho**2)).round(3))
```

```
             mean             sd
grid       [-2.347  1.237] [1.042 0.48 ]
Laplace    [-2.138  1.117] [1.004 0.457]
mean-field [-2.276  1.218] [0.533 0.244]
predicted sd, grid sd * sqrt(1 - rho^2): [0.536 0.247]
```

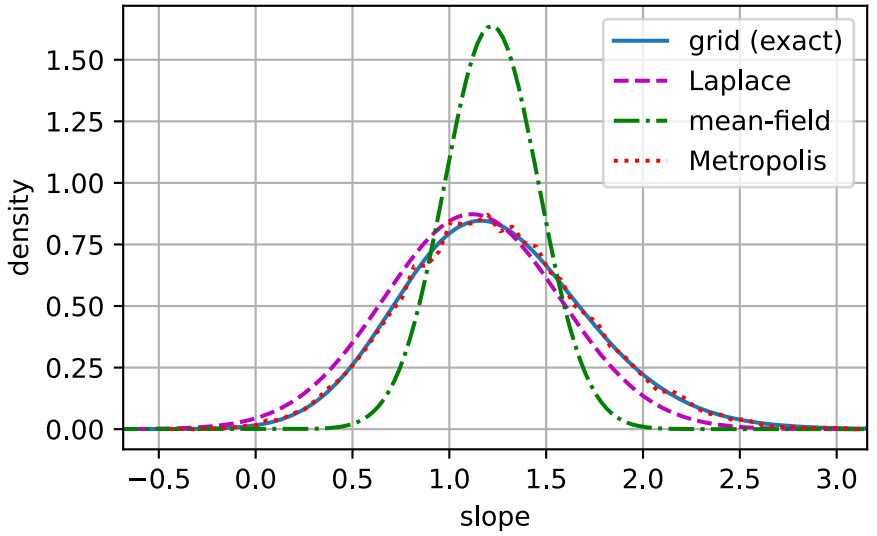
The mean-field mean lies within about a tenth of a posterior standard deviation of the grid mean. Its standard deviations are about half the grid's and match the predicted values, 0.54 and 0.25, to within about 2%. The optimization has converged, and the remaining error comes from the family. Richer families remove it at a higher cost: full-covariance Gaussians (Exercise 7), mixtures, and normalizing flows. A converged ELBO shows that the optimization within the family has stopped; it does not show that the family fits the posterior.

36.8.5 Comparison of the Methods

We now have four answers for the same posterior, and one plot compares them on a single marginal, the density of the slope θ_1: the exact density from the grid, the two Gaussians, and a histogram of the Metropolis states.

```
    def gauss_pdf(t, mean, sd):
        return np.exp(-0.5 * ((t - mean) / sd)**2) / (sd * np.sqrt(2 * np.pi))

    b1 = grid_axes[1]
    exact = grid_weight.reshape(m, m).sum(axis=0) / (b1[1] - b1[0])
    hist, edges = np.histogram(flat[:, 1], bins=60, density=True)
    d2l.plot([b1, b1, b1, (edges[:-1] + edges[1:]) / 2],
             [exact, gauss_pdf(b1, theta_map[1], laplace_sd[1]),
              gauss_pdf(b1, m_avg[1], s_avg[1]), hist],
             xlabel='slope', ylabel='density', figsize=(5, 3),
             xlim=(post_mean[1] - 4 * post_sd[1], post_mean[1] + 4 * post_sd[1]),
             legend=['grid (exact)', 'Laplace', 'mean-field', 'Metropolis'])
```



The Metropolis histogram follows the exact curve, skew included. The Laplace curve is centered at the mode, to the left of the exact mean, and slightly too narrow. The mean-field curve is about half as wide as the exact one, and the mean-field fit also reports a correlation of zero against the grid's $\rho = -0.86$.

Method	Representation	Main diagnostic	Characteristic failure
Grid	weighted points on a lattice	mass at the edge of the grid	cost m^d in dimension d
Laplace	one Gaussian at the mode	comparison with samples	skew, tails, boundaries, other modes
Rejection sampling	independent exact draws	acceptance rate	rate falls like $n^{-d/2}$; needs a bound M
Importance sampling	independent weighted draws	weight ESS, largest weight, reruns	light-tailed or misplaced proposal
MCMC	correlated posterior draws	traces, split $\widehat{R}$, ESS, MCSE	slow mixing, unvisited modes
Variational inference	optimized member of a family	comparison with samples	family too narrow; factorized q too confident

Which one should we use? With a handful of parameters, a grid. For a smooth posterior with one mode, the Laplace approximation is the fast first answer and a good proposal. An expectation that needs an error bar calls for MCMC with its three diagnostics. Variational inference suits a fit that sits inside a training loop, provided we remember that its widths are too small. The two Gaussian approximations carry no check of their own, so their answers should be compared with a sampler whose error bar we trust.

None of the four scales cleanly to the posterior of a large neural network. Importance weights degenerate as the dimension grows (Exercise 11), a random walk needs more steps

per independent draw, the Hessian has d^2 entries, and a factorized q is too narrow by the factor computed here. A network's symmetries also copy every mode of its posterior, once for every permutation and sign flip of its hidden units (Exercise 9). The book uses the ideas all the same, for other distributions known only up to a constant. The evidence lower bound trains the variational autoencoder (Kingma and Welling 2014) of Section 17.6, where an encoder network produces q, and discrete diffusion (Section 17.8). Importance weights and their effective sample size let a policy gradient reuse old data (Section 15.2). The gradient-following proposal, without its accept step, is the Langevin sampler of score-based models (Section 17.1). The sampling methods return averages over finitely many draws, and the next section asks how far such an average can be trusted when the draws are independent.

36.8.6 Summary

A model trained to one parameter vector predicts as if that vector were certain. Far from the data the plausible vectors disagree by orders of magnitude, and the posterior predictive, which averages them, can give a rare label several times the plug-in probability. The average is an integral over a posterior known only up to its normalizer, and four methods compute it from quantities in which the normalizer cancels: the curvature of $\log \tilde{p}$ at the mode (Laplace), the weight $\tilde{p}/q$ (rejection and importance sampling), the acceptance ratio $\tilde{p}(\theta')/\tilde{p}(\theta)$ (Metropolis), and an objective that differs from a KL divergence by a constant (variational inference). The samplers carry their own checks (the weight ESS; the split $\widehat{R}$, κ and the MCSE), and the two Gaussian approximations are checked against samples. None of the four reaches the posterior of a large network, but the evidence lower bound, importance weights and Langevin steps return in the book for other distributions known only up to a constant.

36.8.7 Exercises

1. **Metropolis–Hastings acceptance.** The acceptance probability Equation 36.106 turns any proposal into a chain that targets the posterior.
   1. On a continuous state space the transition kernel has a density part $g(\theta' \mid \theta)\alpha(\theta, \theta')$ and an atom at the current state θ of mass $r(\theta) = 1 - \int g(\theta' \mid \theta)\alpha(\theta, \theta')\, d\theta'$, the probability of a rejection. Show that the posterior is stationary, accounting for the atom. *Hint:* the atom contributes $\pi(\theta')r(\theta')$ to the density at θ' after one step.
   2. An independence sampler proposes $\theta' \sim q$ regardless of the current state. Show that its acceptance probability is $\min(1, w(\theta')/w(\theta))$ with the importance weight w of Equation 36.102. What does the chain do after it reaches a state whose weight is far larger than the weights of typical proposals?
   3. Barker's rule accepts a symmetric proposal with probability $\tilde{p}(\theta')/(\tilde{p}(\theta)+\tilde{p}(\theta'))$. Show that it satisfies detailed balance. Write its transition matrix for the two-state chain of the text, and show that the Metropolis chain moves more often from both states. For a two-state chain the autocorrelation of any function of the state is $\rho_k = \lambda^k$, where λ is the second eigenvalue of the transition matrix. Compute λ and $\kappa = (1+\lambda)/(1-\lambda)$ for both rules. Which gives the smaller Monte Carlo error for the same number of steps? *Hint:* Barker gives the off-diagonal entries $3/4$ and $1/4$, against Metropolis' 1 and $1/3$.
2. code **Proposal scale.** The function `run_chains` multiplies the Cholesky factor of

`laplace_cov` by a scale, 1.1 in the text.

1. Sweep the scale over $\{0.05, 0.2, 0.5, 1.1, 1.7, 2.4, 3, 10\}$, keeping the 12,000 steps per chain and the warm-up. For each value, record the acceptance rate, the split $\widehat{R}$, and the smallest ESS of `chain_ess` divided by the $48{,}000$ log-joint evaluations of the four chains. Plot the ESS per evaluation against the acceptance rate. Over which range of acceptance rates does the ESS per evaluation stay within a factor of two of its maximum?
2. For a d-dimensional standard Gaussian target and the proposal $\mathcal{N}(\theta, (\ell^2/d)\mathbf{I})$, Roberts et al. (1997) show that as $d \to \infty$ the efficiency is largest at $\ell \approx 2.38$, where the acceptance rate is about 0.234. The logistic posterior is close to its Laplace approximation, so the Cholesky factor makes it roughly standard. Which scale does this result suggest for $d = 2$? How do the best scale of your sweep and its acceptance rate compare?
3. Which of the diagnostics of the text flag the run with scale 0.05? Is the acceptance rate one of them? How far is that run's posterior mean from the grid mean, in units of its own MCSE?

3. code **Monte Carlo standard error.** A stationary AR(1) sequence $x_t = \phi x_{t-1} + \varepsilon_t$ with $|\phi| < 1$ has the autocorrelations $\rho_k = \phi^k$ and the autocorrelation time $\kappa = (1 + \phi)/(1 - \phi)$.
   1. From the ESS that your run prints for the slope, compute κ and the AR(1) coefficient ϕ with this autocorrelation time. Compare ϕ with the lag-1 autocorrelation ρ_1 that the cell prints. Is the chain more or less correlated at long lags than an AR(1) sequence with the same ρ_1?
   2. How many post-warm-up states per chain does the same sampler need to bring the MCSE of the slope's posterior mean down to 0.001?
   3. Batch means estimate the MCSE without autocorrelations. Split each chain into batches of b consecutive states, and divide the standard deviation of all batch means by the square root of the number of batches. Compute this estimate for $b \in \{25, 100, 500\}$ and compare it with the printed MCSE. What goes wrong when b is not much larger than κ?
4. **Laplace and parameterization.** A coin shows 9 heads and 4 tails. Under a uniform prior, the posterior of its heads probability θ is $\mathrm{Beta}(10, 5)$, with mean $2/3$, standard deviation 0.118, and central 95% interval $[0.419, 0.872]$.
   1. The Laplace approximation Equation 36.100 in the parameter θ has mean $9/13$ and standard deviation 0.128, as computed in the text. Give its central 95% interval and the probability it places outside $[0, 1]$.
   2. In the log-odds $\eta = \log(\theta/(1 - \theta))$ the posterior density is proportional to $\mathrm{sigmoid}(\eta)^{10}(1 - \mathrm{sigmoid}(\eta))^{5}$, because the Jacobian $d\theta/d\eta = \theta(1 - \theta)$ adds one count to each side. Show that its mode is $\eta = \log 2$, compute the curvature of the log density there, and map the central 95% interval of the Laplace approximation back to θ.
   3. Which of the two approximations is closer to the exact posterior? Why does the Laplace approximation depend on the parameterization, whereas the exact posterior probability of an event such as $\theta > 1/2$ does not?
5. code **Moderated predictions.** ⊠ Under the Laplace approximation Equation 36.100, the logit $a = \mathbf{z}_\star^\top \theta$ at a test input with $\mathbf{z}_\star = (1, x_\star)^\top$ is Gaussian with mean $\mu_a = \mathbf{z}_\star^\top \theta_{\mathrm{MAP}}$ and variance $s_a^2 = \mathbf{z}_\star^\top \mathbf{H}^{-1} \mathbf{z}_\star$. The probit approximation $\mathrm{sigmoid}(a) \approx \Phi(\lambda a)$ with $\lambda^2 = \pi/8$, where Φ is the standard normal c.d.f., turns the predictive integral Equation 36.96 into a closed form (MacKay 1992).
   1. Show that $\mathbb{E}[\Phi(\lambda a)] = \Phi(\lambda \mu_a / \sqrt{1 + \lambda^2 s_a^2})$ for $a \sim \mathcal{N}(\mu_a, s_a^2)$. *Hint:* write

$\Phi(\lambda a)$ as $P(U \le \lambda a)$ for an independent standard normal U. Conclude that the moderated prediction sigmoid$(\mu_a/\sqrt{1+\pi s_a^2/8})$ lies between sigmoid(μ_a) and $1/2$.

2. Compute the moderated predictions at $x_\star \in \{-3, 2, 7\}$ from `theta_map` and `laplace_cov`, and compare them with the plug-in and posterior predictive values that your run prints.
3. Compute three quantities at the same inputs: the plug-in sigmoid$(\mathbf{z}_\star^\top \theta_{\text{MAP}})$, the prediction at the posterior mean sigmoid$(\mathbf{z}_\star^\top \mathbb{E}[\theta \mid \mathcal{D}])$, and the posterior predictive $\mathbb{E}[\text{sigmoid}(\mathbf{z}_\star^\top \theta) \mid \mathcal{D}]$. Which of the two steps, from mode to mean and from mean to average, moves the prediction away from $1/2$, and which toward it? Why are both steps small at $x_\star = 2$ but not at $x_\star = -3$ or $x_\star = 7$? What does this imply for using the Laplace approximation to moderate overconfident predictions?

6. code **Predicting the weight ESS.** ⊠ By the proposition of the text, N_{eff}/N approaches $(\mathbb{E}_q[w])^2/\mathbb{E}_q[w^2]$ as the number N of draws grows.
   1. Let the normalized target be $\mathcal{N}(\mu, s_p^2)$ and the proposal $\mathcal{N}(\nu, s_q^2)$ in one dimension, and let $r = s_q/s_p$. Show that for $s_q^2 > s_p^2/2$
   $$\frac{\mathbb{E}_q[w^2]}{(\mathbb{E}_q[w])^2} = \frac{r^2}{\sqrt{2r^2-1}} \exp\left(\frac{(\mu-\nu)^2}{2s_q^2 - s_p^2}\right),$$
   and that the ratio is infinite for $s_q^2 \le s_p^2/2$. Which r minimizes the ratio when $\mu = \nu$?
   2. Approximate the posterior by independent Gaussians with the grid means and standard deviations, ignoring the correlation, so that the ratio factorizes over the coordinates. Predict the ESS of the 20,000 prior-proposal draws and compare it with the printed value. Repeat for the prior proposals with $\tau = 0.5$ and $\tau = 10$: set `prior_var` $= \tau^2$ in the model cell, rerun the cells through the importance cell, and compare prediction and measurement. Which factor of the formula makes a narrow prior a poor proposal, and which makes a broad prior a poor one?
   3. Under the same approximation, for which coordinates of your run does the proposal $\mathcal{N}(\theta_{\text{MAP}}, 0.3\,\mathbf{I})$ have infinite weight variance? Run it ten times with 5,000 draws and record the ESS, the largest normalized weight, and the estimated posterior mean. Why does a run whose estimate looks plausible not show that the proposal is adequate?
   4. Rejection sampling from the prior keeps a draw with probability $L(\theta)/\max L$, where L is the likelihood and $\max L$ its value at the maximum-likelihood estimate (for separable data, its supremum 1). Measure the acceptance rate on your data and on data with ten times as many points. Explain the decay with n from the volume of the posterior, and compare the rate with the ESS fraction of importance sampling from the same prior.

7. code **Variational families.** ⊠ The variational cell fits a mean-field Gaussian by stochastic gradient ascent on the ELBO Equation 36.110, following the gradients Equation 36.111 with a decaying step.
   1. Predict the mean-field standard deviations from the grid standard deviations and correlation, and compare them with the printed values. What would the forward-KL diagonal Gaussian give?
   2. Rerun the cell three times. Compare the spread of the final iterate `m_vi` across runs with the spread of the averaged iterate `m_avg`. How much of the gap between the mean-field mean and the grid mean is approximation error, and how much

is optimization noise?

3. Replace the mean-field family by a full-covariance Gaussian $\theta = \mathbf{m} + \mathbf{L}\epsilon$, where $\mathbf{L}$ is lower triangular with diagonal entries e^{ℓ_1} and e^{ℓ_2}. Extend Equation 36.111 to the gradients with respect to $\mathbf{m}$, ℓ_1, ℓ_2, and the off-diagonal entry of $\mathbf{L}$. Implement them, and compare the fitted standard deviations and correlation with the grid values.

8. code **Estimating the evidence.** ▯ Let q be a normalized proposal, let $w = p(\mathcal{D}, \theta)/q(\theta)$, and let $\hat{Z}_N = \frac{1}{N}\sum_{s=1}^{N} w(\theta_s)$ for independent draws $\theta_s \sim q$.
   1. Show that $\hat{Z}_N$ is an unbiased estimate of $p(\mathcal{D})$ and that $\mathbb{E}[\log \hat{Z}_N] \leq \log p(\mathcal{D})$. Show that $\mathbb{E}[\log \hat{Z}_1]$ is the ELBO Equation 36.110 of q and that $\mathbb{E}[\log \hat{Z}_N]$ does not decrease with N (Burda et al. 2016).
   2. The function `log_joint` omits the normalizing constant $-\log(2\pi\tau^2)$ of the two-dimensional prior. Add it, and compute $\log p(\mathcal{D})$ in three ways: by summing the density over the grid and multiplying by the area of a grid cell, by $\hat{Z}_N$ with 5,000 Laplace-proposal draws, and by the Laplace evidence $\log \tilde{p}(\theta_{\text{MAP}}) + \log 2\pi - \frac{1}{2}\log\det \mathbf{H}$. The Laplace estimate of $p(\mathcal{D})$ comes out a few percent low, about 0.04 in $\log p(\mathcal{D})$. Which feature of the posterior, seen in the grid cell, explains the sign?
   3. Estimate the ELBO of three Gaussians: the Laplace approximation, the mean-field fit, and the full-covariance fit of Exercise 7c. Use 10^5 draws from each and report the MCSE of each estimate. By Equation 36.109, the gap between $\log p(\mathcal{D})$ and the ELBO of q is $D_{\text{KL}}(q \| p(\cdot \mid \mathcal{D}))$. Rank the three Gaussians by this divergence, and explain where the Laplace Gaussian falls in the ranking.

9. code **A posterior with two modes.** ▯ Fifty observations are drawn as $y_i \sim \mathcal{N}(\theta^2, 1)$ with the true value $\theta = 1$, and θ has the prior $\mathcal{N}(0, 4)$. The likelihood depends on θ only through θ^2, just as the loss of a neural network does not change when the incoming weights, the bias, and the outgoing weights of a tanh hidden unit flip sign together. The posterior is therefore symmetric under $\theta \mapsto -\theta$.
   1. Compute the posterior on a one-dimensional grid. Report $\mathbb{E}[\theta \mid \mathcal{D}]$, $\mathbb{E}[\theta^2 \mid \mathcal{D}]$, and the ratio of the posterior density at $\theta = 0$ to its value at a mode.
   2. Show that the Gaussian that minimizes the forward divergence $D_{\text{KL}}(p\|q)$ matches the mean and the variance of p. Where does this Gaussian put its mode for the present posterior, and how plausible is that point under p?
   3. Fit three local approximations: the Laplace approximation at the positive mode, a one-dimensional Gaussian fitted by the ELBO ascent of the variational cell started at $m = 0.5$, and four Metropolis chains with proposal standard deviation 0.2 started at $\theta \in \{0.5, 1, 1.5, 2\}$. Then rerun the chains from $\theta \in \{-2, -0.5, 0.5, 2\}$. Which methods report a single mode? For each run, which of its diagnostics (split $\widehat{R}$ and the ESS for the chains, the ELBO for the variational fit) reveal the problem?
   4. Which posterior expectations does a single-mode approximation still estimate correctly? A network with h hidden tanh units has the sign flip of this problem for every unit and, in addition, the $h!$ permutations of its hidden units (Exercise 9). How many equivalent copies of a mode does its posterior have under an isotropic Gaussian prior? What does this suggest about approximating the posterior of a neural network by one of the methods of this section?

10. code **Tails of the logistic posterior.** ▯▯ A central limit theorem for importance sampling needs a finite second moment of the weights. The Laplace proposal of the importance cell usually reaches a high ESS. Throughout, $\tau^2 = \texttt{prior_var} = 4$ is the prior

variance in the posterior Equation 36.97.

1. Fix a unit vector $\mathbf{v}$ and let $\theta = t\mathbf{v}$. Show that the log-likelihood in Equation 36.97 equals $-c(\mathbf{v})t + O(1)$ as $t \to \infty$, where $c(\mathbf{v})$ is the sum of $|\mathbf{z}_i^\top \mathbf{v}|$ over the observations with $y_i = 1$ and $\mathbf{z}_i^\top \mathbf{v} < 0$ or with $y_i = 0$ and $\mathbf{z}_i^\top \mathbf{v} > 0$. For which datasets is $c(\mathbf{v}) > 0$ for every $\mathbf{v}$?
2. Conclude that a Gaussian proposal $\mathcal{N}(\mathbf{m}, \mathbf{S})$ has $\mathbb{E}_q[w^2] = \infty$ when some eigenvalue of $\mathbf{S}$ is smaller than $\tau^2/2$, and $\mathbb{E}_q[w^2] < \infty$ when every eigenvalue of $\mathbf{S}$ exceeds $\tau^2/2$. Which of the two proposals of the importance cell has weights with finite variance? Compare this answer with the one that the formula of Exercise 6a gives for the Laplace proposal when the posterior is approximated by a Gaussian.
3. Along each eigenvector of `laplace_cov`, in both directions, find the Mahalanobis distance from θ_{MAP} beyond which $\tilde{p}^2/q$ exceeds its value at θ_{MAP}. How likely is a single Laplace-proposal draw to land that far out? Reconcile your answer with the ESS that your run prints.
4. Geweke (1989) builds importance densities from multivariate Student-t approximations at the posterior mode. Use a Student-t proposal with 3 degrees of freedom, location θ_{MAP}, and scale matrix `laplace_cov`, and show that its weights have finite variance. Compare it with the Laplace proposal over 20 runs on the ESS and on the estimate of the tail probability $P(\theta_1 > \bar{\theta}_1 + 3s_1 \mid \mathcal{D})$, where $\bar{\theta}_1$ and s_1 are the grid mean and standard deviation of the slope; for a reference value, evaluate the posterior on a grid twice as wide. Also try the defensive mixture of the text, and explain why its weights are bounded.

11. [extended] **Approximations in higher dimension.** ▯▯ Extend the running example to d parameters: an intercept and $d-1$ features with independent standard normal entries, $n = 10d$ observations whose labels are drawn from a true parameter with independent $\mathcal{N}(0, 1.5^2/d)$ entries, and the prior $\mathcal{N}(\mathbf{0}, 4\mathbf{I})$. No grid is available, so a long Metropolis run serves as the reference.
    1. For $d \in \{2, 5, 10, 20, 50\}$, compute the Laplace approximation and the ESS of 5,000 Laplace-proposal draws. Repeat with $n = 40d$. Explain how N_{eff}/N depends on d and on n/d.
    2. Run four Metropolis chains with the proposal $(2.38/\sqrt{d})\mathbf{L}$, where $\mathbf{L}$ is the Cholesky factor of the Laplace covariance (Exercise 2b). Measure the acceptance rate and the smallest ESS per log-joint evaluation over the coordinates as functions of d. Which power of d describes the efficiency?
    3. At $d = 50$ and $n = 500$, give importance sampling and Metropolis the same budget of 20,000 log-joint evaluations, and repeat each run five times. Measure their errors in the posterior mean against the reference, in units of posterior standard deviations, and compare them with the errors that each method's own ESS predicts. Report one row per method with the number of evaluations, the ESS, the predicted error, and the measured error. Which method would you use at this budget?
12. code **Langevin proposals.** ▯▯ The Metropolis-adjusted Langevin algorithm proposes $\theta' = \theta + \frac{\epsilon^2}{2}\mathbf{C}\nabla \log \tilde{p}(\theta) + \epsilon\, \mathbf{C}^{1/2}\xi$ with $\xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and a preconditioner $\mathbf{C}$; take $\mathbf{C}$ to be the Laplace covariance $\mathbf{H}^{-1}$ and $\mathbf{C}^{1/2}$ its Cholesky factor.
    1. Write down the proposal density $g(\theta' \mid \theta)$ and the Hastings factor of the acceptance rule Equation 36.106. Why is the proposal not symmetric?
    2. Implement the sampler with `grad_log_joint` and run four chains for $\epsilon \in$

$\{0.5, 1, 1.5\}$. Report the acceptance rate, the split $\widehat{R}$, and the ESS per gradient evaluation, and compare with the random walk's ESS per log-joint evaluation.

3. Without the accept step the chain is the unadjusted Langevin algorithm. Run it at $\epsilon = 1$ and compare its mean and standard deviations with the grid. Which of the two errors, the bias of the discretization or the Monte Carlo error, does the Metropolis step remove?

36.9 Concentration Inequalities

A test accuracy is a measurement, and Section 36.5 put an error bar on it. That error bar, $\pm 1.96\,\hat{\sigma}/\sqrt{n}$, comes from a Gaussian approximation to the sampling distribution, which is exact only as $n \to \infty$. Section 36.1 gave a bound that holds at every n, Chebyshev's inequality, but it is weak. How far can a test-set error be from the truth, with a guarantee that holds at the sample size we actually have and stays useful when we demand that it almost never fails?

Take a classifier that makes 830 mistakes on 10,000 test images, an error rate of 8.3%. Chebyshev's inequality bounds the chance that the average of n independent draws misses its mean by t or more by $\sigma^2/(nt^2)$, and a 0/1 outcome has variance $\sigma^2 \le \frac{1}{4}$. So the true error lies within $\pm 2.2\%$ of 8.3% with 95% confidence. Ask for a guarantee that fails only once in a million, and Chebyshev answers $\pm 500\%$, which says nothing. Yet the exact probability that an average of 0/1 outcomes strays far falls exponentially in n, and a bound that captures this answers $\pm 1.4\%$ and $\pm 2.7\%$ for the same two questions.

The idea behind such bounds is to apply Markov's inequality not to the average but to its exponential, which turns a sum of independent terms into a product. Sergei Bernstein found bounds of this kind in the 1920s, Herman Chernoff used the method in 1952 to measure how fast statistical tests become reliable, and Wassily Hoeffding gave in 1963 the form used throughout machine learning (Hoeffding 1963).

Why ask for one failure in a million? A learner that compares many models and keeps the best needs a guarantee for all of them at once, and only a tail that shrinks exponentially gives one at a practical sample size; that is the subject of Section 36.10. This section sizes a test set from a required precision. The same calculation, applied to the coordinates of a random vector, shows that a standard Gaussian vector in $\mathbb{R}^{784}$ has a length within a few percent of 28 and that two random directions are nearly orthogonal. Initialization scales and embeddings rest on both facts.

Three results are worth remembering: the Chernoff method Equation 36.112, Hoeffding's interval Equation 36.116, and the concentration of the norm Equation 36.121.

```
rng = np.random.default_rng()
```

36.9.1 Exponential Tail Bounds

The Gaussian interval is narrow because Gaussian tails fall like $e^{-z^2/2}$, a rate it takes from the central limit theorem and that therefore holds only in the limit. We want a rate of this kind together with a guarantee at every finite n. First we measure how far Chebyshev's bound is from the truth.

36.9.1.1 Chebyshev's Rate

Chebyshev's bound is Markov's inequality Equation 36.13 applied to a squared deviation. Here the deviation is that of $\bar{X}$, the mean of n independent draws with mean μ and finite variance σ^2. The expectation of $(\bar{X} - \mu)^2$ is $\mathrm{Var}(\bar{X}) = \sigma^2/n$, so

$$P(|\bar{X} - \mu| \geq t) \leq \frac{\sigma^2}{nt^2}.$$

The miss probability falls like $1/n$. To push it from 1 to 10^{-6} at a fixed deviation t, the sample must grow a million-fold.

How small is the true miss probability? The smallest case where we can compute it exactly is a fair coin. Let $\hat{p}$ be the fraction of heads in n flips, and ask for the probability that it misses $\frac{1}{2}$ by $t = 0.1$ or more. A flip has variance $\frac{1}{4}$, so Chebyshev's bound is $0.25/(0.01\,n) = 25/n$. The exact probability is a binomial sum, which we evaluate in log space so that $n = 5{,}000$ does not underflow.

```
def exact_tail(n, t):
    """P(|p_hat - 1/2| >= t) for n fair flips, a binomial sum in log space."""
    k = np.arange(n + 1)
    log_pmf = n * np.log(0.5) + np.concatenate(
        ([0.0], np.cumsum(np.log((n - k[1:] + 1) / k[1:]))))
    in_tail = log_pmf[np.abs(k / n - 0.5) >= t - 1e-12]  # fp-safe boundary
    m = in_tail.max()
    return np.exp(m) * np.exp(in_tail - m).sum()  # stable log-sum-exp
```

At $n = 1{,}000$ Chebyshev allows a miss in one experiment out of forty. Before reading the table, guess how often such a miss really happens.

```
t = 0.1
print(f'{"n":>6} {"exact":>10} {"Chebyshev":>10}')
for n in (10, 100, 1000, 5000):
    print(f'{n:6d} {exact_tail(n, t):10.2e} {0.25 / (n * t**2):10.2e}')
```

```
     n      exact  Chebyshev
    10   7.54e-01   2.50e+00
   100   5.69e-02   2.50e-01
  1000   2.73e-10   2.50e-02
  5000   1.30e-45   5.00e-03
```

At $n = 10$ the exact tail is 0.75 and the bound, 2.5, says nothing. At $n = 100$ the bound is 0.25, about four times the exact 0.057. Beyond that the two part ways. At $n = 1{,}000$ the exact tail is 2.7×10^{-10} while Chebyshev allows 0.025, eight orders of magnitude more, and at $n = 5{,}000$ the gap is forty-two orders. Each tenfold increase of n divides Chebyshev's bound by ten, while the exact tail loses a fixed number of decimal places per hundred flips: on a log scale it falls along a straight line.

Chebyshev's bound cannot be tightened without more information about the distribution. Section 36.1 showed that it holds with equality for a distribution on three points, so no bound that uses only the mean and the variance can improve on it for every distribution. The coin

has more structure than two moments: it is bounded, so all its moments are small. A bound that uses all moments at once needs the moment generating function.

36.9.1.2 The Chernoff Method

To use more than two moments, look at what Markov's and Chebyshev's inequalities have in common. If a function g lies above the indicator $\mathbf{1}\{x \geq t\}$ everywhere, a *majorant* of the indicator, then $P(X \geq t) = \mathbb{E}[\mathbf{1}\{X \geq t\}] \leq \mathbb{E}[g(X)]$. Markov's inequality uses $g(x) = x/t$ for a nonnegative X. Chebyshev's uses $g(x) = (x/t)^2$, in practice with the deviation $X - \mu$ in place of X. The exponential $e^{\lambda(x-t)}$, for any $\lambda > 0$, lies above the indicator on the whole real line. Panel (a) of Figure 36.37 draws the three functions for one nonnegative X. The bound is the expectation of g, so what matters is the value of g where the mass of X sits.

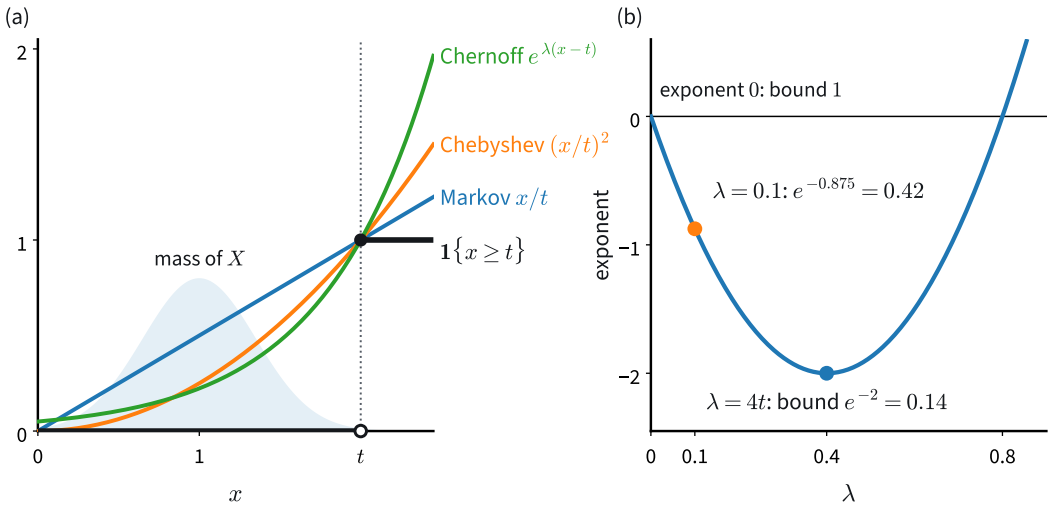


Figure 36.37: A tail bound is the expectation of a function lying above the indicator. (a) For a threshold $t = 2$, the indicator $\mathbf{1}\{x \geq t\}$ (black) and three functions above it: Markov's x/t, Chebyshev's $(x/t)^2$, and Chernoff's $e^{\lambda(x-t)}$ with $\lambda = 1.5$. The shaded bump marks where the mass of X sits; each bound is the average of its function under that mass, and raising λ makes the exponential small there. (b) The Chernoff exponent $-\lambda nt + n\lambda^2/8$ for the mean of $n = 100$ fair coins at $t = 0.1$. Every $\lambda > 0$ gives a valid bound; the best, at the minimum $\lambda = 4t$ of the parabola, is e^{-2}.

The expectation of the exponential has a name. The *moment generating function* of X is $\mathbb{E}[e^{\lambda X}]$, as a function of λ, wherever the expectation is finite. For a fair coin centered at zero, $Z \in \{-\frac{1}{2}, \frac{1}{2}\}$, it is $\frac{1}{2}(e^{\lambda/2} + e^{-\lambda/2}) = \cosh(\lambda/2)$. It is finite for every λ when X is bounded, and infinite for every $\lambda > 0$ when the right tail falls only polynomially, as for a Student-t variable; for such variables the method below gives nothing.

Markov's inequality applied to the nonnegative variable $e^{\lambda X}$ gives the bound. The map $x \mapsto e^{\lambda x}$ is increasing, so the event is unchanged, and for every $\lambda > 0$

$$P(X \geq t) = P(e^{\lambda X} \geq e^{\lambda t}) \leq e^{-\lambda t}\,\mathbb{E}[e^{\lambda X}], \qquad \text{hence} \qquad P(X \geq t) \leq \inf_{\lambda > 0} e^{-\lambda t}\,\mathbb{E}[e^{\lambda X}]. \tag{36.112}$$

Every $\lambda > 0$ gives a valid bound, and we keep the best one. This is the *Chernoff method*.

The exponential is the right majorant for a second reason. If $S = X_1 + \cdots + X_n$ is a sum of independent terms, then $e^{\lambda S} = \prod_i e^{\lambda X_i}$ is a product of independent factors, and the expectation of such a product is the product of the expectations:

$$\mathbb{E}[e^{\lambda S}] = \prod_{i=1}^{n} \mathbb{E}[e^{\lambda X_i}].$$

The moment generating function of a sum is the product of those of its terms, so the logarithm of the Chernoff bound for a sum has n terms and grows in proportion to n. No polynomial majorant factorizes this way: $\mathbb{E}[S^4]$ is not the product of the fourth moments. What remains is to bound the moment generating function of one summand.

36.9.1.3 The Fair Coin

The fair coin is the smallest case, and working it to the end shows every step of the general proof. A centered flip $Z = X - \frac{1}{2}$ takes the values $\pm\frac{1}{2}$ with probability $\frac{1}{2}$ each, so $\mathbb{E}[e^{\lambda Z}] = \cosh(\lambda/2)$. Compare the Taylor series of $\cosh u$ and of $e^{u^2/2}$ term by term:

$$\cosh u = \sum_{k \geq 0} \frac{u^{2k}}{(2k)!} \leq \sum_{k \geq 0} \frac{(u^2/2)^k}{k!} = e^{u^2/2},$$

because $(2k)! \geq 2 \cdot 4 \cdots (2k) = 2^k k!$: the even factors of $(2k)!$ alone give $2^k k!$, and the odd ones are at least 1. With $u = \lambda/2$, one centered flip has $\mathbb{E}[e^{\lambda Z}] \leq e^{\lambda^2/8}$ for every λ. A Gaussian $\mathcal{N}(0, \sigma^2)$ has $\mathbb{E}[e^{\lambda X}] = e^{\lambda^2\sigma^2/2}$ (complete the square in the integral), so a flip's moment generating function is no larger than that of a Gaussian with standard deviation $\frac{1}{2}$.

For n independent flips, the event $\hat{p} - \frac{1}{2} \geq t$ is the event $\sum_i Z_i \geq nt$. The Chernoff method and the product rule give

$$P(\hat{p} - \tfrac{1}{2} \geq t) \leq e^{-\lambda n t} \prod_{i=1}^{n} \mathbb{E}[e^{\lambda Z_i}] \leq \exp\Big(-\lambda n t + \frac{n\lambda^2}{8}\Big).$$

The exponent is a parabola in λ. Panel (b) of Figure 36.37 draws it for $n = 100$ and $t = 0.1$, where it reads $-10\lambda + 12.5\lambda^2$. Any $\lambda > 0$ gives a bound, for instance $e^{-0.875} \approx 0.42$ at $\lambda = 0.1$. The best one sits at the minimum, $\lambda = 4t$, where the exponent is $-4nt^2 + 2nt^2 = -2nt^2$. The lower tail is the same by symmetry, and adding the two tails gives

$$P(|\hat{p} - \tfrac{1}{2}| \geq t) \leq 2e^{-2nt^2}.$$

The probability of missing by t falls exponentially in nt^2. At $n = 100$ and $t = 0.1$ the bound is $2e^{-2} \approx 0.27$. At $n = 1{,}000$ it is $2e^{-20} \approx 4.1 \times 10^{-9}$, where Chebyshev allowed 0.025.

Why is the exponential majorant so effective? The function $e^{\lambda(S-nt)}$ equals 1 at the threshold, but at the mean of the sum, where the mass is, it equals $e^{-\lambda nt}$, exponentially small in n. What limits λ is that the function grows with S, so outcomes above the mean count for more than $e^{-\lambda nt}$. The product of the moment generating functions, at most $e^{n\lambda^2/8}$, measures by how much, and the best λ balances the two factors.

The proof used one fact about the coin: the moment generating function of a centered flip is at most $e^{\lambda^2/8}$, that of a Gaussian with standard deviation $\frac{1}{2}$. The $\frac{1}{2}$ is half the range of a flip. Hoeffding's lemma says that every variable confined to an interval of length $b-a$ satisfies the same bound with $(b-a)/2$ in place of $\frac{1}{2}$.

36.9.1.4 Hoeffding's Lemma and Hoeffding's Inequality

A test-set loss is rarely a fair coin. An error rate of 8.3% is a biased coin, and a clipped loss takes many values. We want the coin's bound for every bounded variable. The lemma that gives it is the hard step of this section, and its proof rests on one fact about variances.

Proposition (variance of a bounded variable). *If Y takes values in $[a,b]$, then $\mathrm{Var}(Y) \le (b-a)^2/4$, with equality when Y takes the values a and b with probability $\frac{1}{2}$ each.*

Proof. The mean minimizes the expected squared deviation, since $\mathbb{E}[(Y-c)^2] = \mathrm{Var}(Y) + (\mathbb{E}[Y]-c)^2$ for every center c. Take c to be the midpoint. Every point of $[a,b]$ lies within $(b-a)/2$ of it, so

$$\mathrm{Var}(Y) \le \mathbb{E}\Big[\Big(Y - \frac{a+b}{2}\Big)^2\Big] \le \Big(\frac{b-a}{2}\Big)^2.$$

Both inequalities are equalities for the distribution with equal mass at a and b. ■

The fair coin is this extreme case, with variance $\frac{1}{4}$ on $[0,1]$.

Proposition (Hoeffding's lemma). *Let X be a random variable with $\mathbb{E}[X]=0$ taking values in $[a,b]$. Then for every $\lambda \in \mathbb{R}$,*

$$\mathbb{E}\big[e^{\lambda X}\big] \le \exp\left(\frac{\lambda^2 (b-a)^2}{8}\right). \tag{36.113}$$

Proof. We show that $\psi(\lambda) = \log \mathbb{E}[e^{\lambda X}]$ grows no faster than the parabola $\lambda^2(b-a)^2/8$. Since X is bounded, ψ is finite and smooth, and we may differentiate under the expectation. At zero, $\psi(0)=0$, and

$$\psi'(\lambda) = \frac{\mathbb{E}[X e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]}, \qquad \text{so} \qquad \psi'(0) = \mathbb{E}[X] = 0.$$

The second derivative is a variance, as it is for the log-partition function of an exponential family (Section 36.4); ψ is one. Let q_λ be the *tilted* distribution, which reweights that of X by $e^{\lambda x}$ and renormalizes. Then $\psi'(\lambda)$ is the mean of X under q_λ, and

$$\psi''(\lambda) = \frac{\mathbb{E}[X^2 e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]} - \left(\frac{\mathbb{E}[X e^{\lambda X}]}{\mathbb{E}[e^{\lambda X}]}\right)^2 = \mathrm{Var}_{q_\lambda}(X).$$

The tilt moves no mass outside $[a,b]$, so the variance bound applies at every λ: $\psi''(\lambda) \le (b-a)^2/4$. Taylor's theorem with the Lagrange remainder finishes the proof. For some ξ between 0 and λ,

$$\psi(\lambda) = \psi(0) + \lambda\,\psi'(0) + \frac{\lambda^2}{2}\,\psi''(\xi) \le \frac{\lambda^2 (b-a)^2}{8},$$

and exponentiating gives Equation 36.113. ■

A bounded, centered variable has a moment generating function no larger than that of a Gaussian with standard deviation $(b-a)/2$. For the centered coin, $b-a=1$, and the lemma returns the bound $\cosh(\lambda/2) \le e^{\lambda^2/8}$ that we proved by series. The proof of the coin's tail now goes through for every bounded variable.

Proposition (Hoeffding's inequality). *Let $X_1, \dots, X_n$ be independent random variables with $X_i \in [a,b]$, and let $\bar X = \frac{1}{n}\sum_i X_i$. Then for every $t > 0$,*

$$P(|\bar X - \mathbb{E}[\bar X]| \ge t) \le 2\exp\!\left(-\frac{2nt^2}{(b-a)^2}\right). \tag{36.114}$$

Proof. Write $Z_i = X_i - \mathbb{E}[X_i]$. Each Z_i is centered and lies in an interval of length $b-a$, so the lemma applies. For any $\lambda > 0$, the Chernoff method Equation 36.112 and the product rule give

$$P(\bar X - \mathbb{E}[\bar X] \ge t) = P\Big(\sum_i Z_i \ge nt\Big) \le e^{-\lambda n t} \prod_{i=1}^{n} \mathbb{E}\big[e^{\lambda Z_i}\big] \le \exp\!\left(-\lambda n t + \frac{n\lambda^2 (b-a)^2}{8}\right).$$

This is the coin's parabola with $(b-a)^2$ in place of 1. It is minimized at $\lambda = 4t/(b-a)^2$, where its value is $-2nt^2/(b-a)^2$. The same argument applied to $-Z_i$ bounds the lower tail, and adding the two tails gives the factor 2. ■

The mean of n independent bounded terms misses its expectation by t with a probability that falls exponentially in nt^2, with t measured in units of the range. The bound needs no Gaussianity, no variance estimate and no asymptotics. It needs only boundedness and independence, and it holds at every finite n. Independence is a real restriction: the bound does not cover the correlated draws of a Markov chain.

The proof never used that the ranges agree. For independent centered $Y_i \in [a_i, b_i]$, the product of the lemma's bounds has $\sum_i (b_i - a_i)^2$ where it had $n(b-a)^2$, and the same optimization gives for the sum

$$P\Big(\Big|\sum_i Y_i\Big| \ge s\Big) \le 2\exp\!\Big(-\frac{2s^2}{\sum_i (b_i - a_i)^2}\Big). \tag{36.115}$$

We need this form for random directions below.

For losses in $[0,1]$, $b-a=1$, and the bound is at most δ once $n \ge \log(2/\delta)/(2t^2)$. To estimate an error rate within $t = 0.01$ with failure probability $\delta = 0.05$, this gives $\log 40/(2\cdot$

$10^{-4}) = 18{,}444.4$, so $18{,}445$ examples. This is the number that Chapter 3.6 quoted, about twice the 9,604 examples that the Gaussian approximation asks for.

Setting the right-hand side of Equation 36.114 equal to δ and solving for t gives the form used in practice: with probability at least $1 - \delta$,

$$\left|\bar{X} - \mathbb{E}[\bar{X}]\right| \le (b - a)\sqrt{\frac{\log(2/\delta)}{2n}}. \tag{36.116}$$

This is a confidence interval whose half-width shrinks like $1/\sqrt{n}$, the same rate as the Gaussian interval $\hat{\mu} \pm 1.96\,\hat{\sigma}/\sqrt{n}$, but it is valid at every n.

For the classifier with 830 errors on 10,000 images, at $\delta = 0.05$, Hoeffding gives $\pm 1.36\%$ and the Gaussian interval $\pm 1.96\sqrt{0.083 \cdot 0.917/10^4} = \pm 0.54\%$. The two differ in their constants. Hoeffding's half-width is the Gaussian one with $\hat{\sigma}$ replaced by its worst case $(b - a)/2 = 0.5$ and the quantile 1.96 replaced by $\sqrt{2\log(2/\delta)} = 2.72$. With $\hat{\sigma} = 0.28$ the ratio is 2.5. When the spread of the data is much smaller than their range, Hoeffding is conservative, and Bernstein's inequality below repairs this.

The constant $\frac{1}{8}$ in Hoeffding's lemma cannot be improved for all bounded variables: a variable with mass $\frac{1}{2}$ at each endpoint of its interval attains it to second order in λ (Exercise 2).

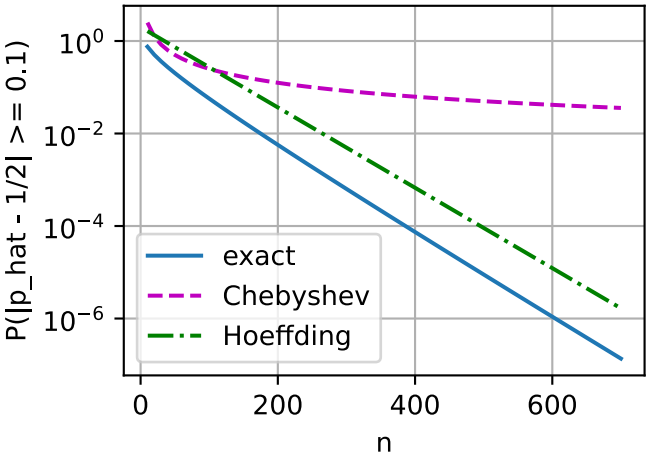
36.9.1.5 The Fair Coin in Numbers

We set Hoeffding's bound $2e^{-2nt^2}$ next to the exact tail of the fair coin at $t = 0.1$, with their ratio, and plot both together with Chebyshev's bound against n on a log scale.

```
print(f'{"n":>6} {"exact":>10} {"Hoeffding":>10} {"ratio":>8}')
for n in (10, 100, 1000, 5000):
    hoeffding = 2 * np.exp(-2 * n * t**2)
    print(f'{n:6d} {exact_tail(n, t):10.2e} {hoeffding:10.2e} '
          f'{hoeffding / exact_tail(n, t):8.1f}')

ns = np.arange(10, 701, 10)
d2l.plot(ns, [np.array([exact_tail(n, t) for n in ns]),
              0.25 / (ns * t**2), 2 * np.exp(-2 * ns * t**2)],
         'n', 'P(|p_hat - 1/2| >= 0.1)',
         legend=['exact', 'Chebyshev', 'Hoeffding'], yscale='log')
```

```
     n      exact  Hoeffding    ratio
    10   7.54e-01   1.64e+00      2.2
   100   5.69e-02   2.71e-01      4.8
  1000   2.73e-10   4.12e-09     15.1
  5000   1.30e-45   7.44e-44     57.2
```



At $n = 10$ both bounds exceed 1. At $n = 100$ Chebyshev's 0.25 is a shade tighter than Hoeffding's 0.27: at small n the constants decide, and Chebyshev uses the true variance $\frac{1}{4}$. For large n the rate decides, and Hoeffding stays within a factor of 15 of the exact tail at $n = 1{,}000$ and of 57 at $n = 5{,}000$.

On the log plot the exact tail and Hoeffding's bound are nearly parallel lines. The logarithm of Hoeffding's bound falls by $2t^2 = 0.02$ per flip, and that of the exact tail by about 0.0201. The second number is the Kullback–Leibler divergence $D_{\mathrm{KL}}(0.6 \| 0.5) = 0.6 \log 1.2 + 0.4 \log 0.8$ between a coin of bias 0.6 and the fair coin (Section 37.3), and the Chernoff method with the exact moment generating function of a coin attains this rate (Exercise 4).

At small n no bound helps. The coin of the chapter introduction landed heads 9 times in 13 flips, a deviation of $t = 2.5/13$ from $\frac{1}{2}$. Hoeffding's bound on a deviation that large is $2e^{-2 \cdot 13 \cdot (2.5/13)^2} = 2e^{-0.96} \approx 0.77$, which rules out nothing. The exact two-sided probability is 0.27, so with 13 flips not even the exact tail rules out a fair coin.

36.9.1.6 Sub-Gaussian and Sub-Exponential Variables

Hoeffding's interval for the 8.3% classifier was 2.5 times wider than the Gaussian one because it uses the range of a 0/1 outcome and ignores its spread. Bernstein's inequality uses both.

Proposition (Bernstein's inequality). *Let $X_1, \dots, X_n$ be independent with $\mathrm{Var}(X_i) \le \sigma^2$ and $|X_i - \mathbb{E}[X_i]| \le M$. Then for every $t > 0$,*

$$P\big(|\bar{X} - \mathbb{E}[\bar{X}]| \ge t\big) \le 2 \exp\left(- \frac{n t^2}{2(\sigma^2 + M t/3)} \right). \tag{36.117}$$

The proof is the Chernoff method once more (Boucheron et al. 2013), with a bound on the moment generating function that uses the variance as well as the range; Exercise 8 carries it out. For small t the denominator is close to $2\sigma^2$, and the bound is the Gaussian tail with the *true* variance. For large t the term $Mt/3$ takes over and the bound decays only exponentially in t.

How much does the variance help? A classifier with an error rate of 2% on $n = 10{,}000$ examples has per-example variance $\sigma^2 = 0.02 \cdot 0.98$, and its centered losses satisfy $M = 1$. Setting the bound equal to δ gives $nt^2/(2(\sigma^2 + t/3)) = \log(2/\delta)$, a quadratic in t. We solve it at $\delta = 0.05$ for both error rates we have met, next to Hoeffding and the Gaussian interval, first with the measured rate in the variance (plug-in) and then in the form that is a guarantee, explained below.

```
def half_widths(p, n, delta=0.05, z=1.96):
    """Hoeffding, Bernstein (plug-in, guaranteed), Gaussian half-widths."""
    L = np.log(2 / delta)
    hoeffding = np.sqrt(L / (2 * n))
    c = 2 * L / 3  # Bernstein with M = 1: n t^2 - c t - 2 L var = 0
    bernstein = lambda var: (c + np.sqrt(c**2 + 8 * n * L * var)) / (2 * n)
    plug_in = t = bernstein(p * (1 - p))
    for _ in range(20):  # true rate at most p + t: a fixed point in t
        t = bernstein((p + t) * (1 - p - t))
    return hoeffding, plug_in, t, z * np.sqrt(p * (1 - p) / n)
```

```
for p in (0.083, 0.02):
    hoeffding, plug_in, bernstein, gauss = half_widths(p, 10000)
    print(f'error {p:.1%}: Hoeffding ±{hoeffding:.2%}, Bernstein '
          f'±{plug_in:.2%} (plug-in) ±{bernstein:.2%} (guaranteed), '
          f'Gaussian ±{gauss:.2%}')
```

```
error 8.3%: Hoeffding ±1.36%, Bernstein ±0.76% (plug-in) ±0.79% (guaranteed), Gaussian ±0.54%
error 2.0%: Hoeffding ±1.36%, Bernstein ±0.39% (plug-in) ±0.43% (guaranteed), Gaussian ±0.27%
```

At an error rate of 2%, Bernstein's half-width is $\pm 0.39\%$ against Hoeffding's $\pm 1.36\%$, three and a half times tighter, and within a factor of 1.4 of the asymptotic Gaussian $\pm 0.27\%$. At 8.3% the gain is smaller, $\pm 0.76\%$ against $\pm 1.36\%$.

These two half-widths are not yet guarantees. The cell put the measured rate into the variance $p(1-p)$, whereas Bernstein's inequality asks for a bound on the true variance: the substitution of $\hat{\sigma}$ for σ that we held against the Gaussian interval. The repair is cheap. The true rate is at most $\hat{p} + t$, so we solve the same equation with variance $(\hat{p} + t)(1 - \hat{p} - t)$, a fixed-point equation in t that the cell solves by iteration. It gives the guaranteed 0.43% and 0.79% in place of 0.39% and 0.76%. This answers the opening question for the classifier with 830 errors: with 95% confidence its true error lies within about 0.8% of the measured 8.3%. Error rates of a few percent are where practitioners work, and there the variance-aware bound matters most.

Hoeffding's proof used one property of the summands: a moment generating function bounded by a Gaussian's. Giving that property a name lets us apply the proof to unbounded variables, and shows where it stops working. A random variable X is **sub-Gaussian** with *variance proxy* σ^2 if its centered moment generating function is no larger than that of $\mathcal{N}(0, \sigma^2)$:

$$\mathbb{E}\left[e^{\lambda(X - \mathbb{E}[X])}\right] \le \exp\left(\frac{\lambda^2 \sigma^2}{2}\right) \qquad \text{for all } \lambda \in \mathbb{R}. \tag{36.118}$$

The tails of such a variable are no heavier than a Gaussian's: the Chernoff method with

$\lambda = t/\sigma^2$ gives $P(|X-\mathbb{E}[X]| \geq t) \leq 2e^{-t^2/(2\sigma^2)}$. Here σ^2 is the proxy and not the variance. The proxy is never smaller than the variance (compare the two sides of Equation 36.118 to second order in λ), and it can be much larger: a 0/1 outcome with error rate 2% has variance 0.0196, while Hoeffding's lemma supplies the proxy $\frac{1}{4}$. That is why Hoeffding's interval was wide.

A Gaussian $\mathcal{N}(\mu, \sigma^2)$ has proxy σ^2, with equality. A variable on $[a, b]$ has proxy $(b-a)^2/4$ by Hoeffding's lemma. A random sign $\xi \in \{-1, +1\}$ has proxy 1, since $\mathbb{E}[e^{\lambda\xi}] = \cosh\lambda \leq e^{\lambda^2/2}$ by the series comparison above. Sub-Gaussianity also survives averaging: if $X_1, \dots, X_n$ are independent with proxy σ^2, the product rule gives $\bar{X}$ the proxy σ^2/n, and hence $P(|\bar{X} - \mathbb{E}\bar{X}| \geq t) \leq 2e^{-nt^2/(2\sigma^2)}$.

Not every variable with all moments finite is sub-Gaussian, and the one we need next is not. The squared length of a Gaussian vector is a sum of squares of standard Gaussians, and the square of a standard Gaussian x has

$$\mathbb{E}\big[e^{\lambda(x^2-1)}\big] = \frac{e^{-\lambda}}{\sqrt{1-2\lambda}} \qquad \text{for } \lambda < \tfrac{1}{2},$$

and the expectation is infinite for $\lambda \geq \frac{1}{2}$. So Equation 36.118 fails for x^2 at every variance proxy. Its right tail falls like $e^{-t/2}$, too slowly for a Gaussian bound.

A Gaussian-type bound does hold on a window of λ around zero. A random variable X is **sub-exponential** with parameters (ν, α) if

$$\mathbb{E}\big[e^{\lambda(X-\mathbb{E}[X])}\big] \leq \exp\left(\frac{\lambda^2\nu^2}{2}\right) \qquad \text{for } |\lambda| \leq \frac{1}{\alpha}. \tag{36.119}$$

Here ν^2 plays the part of the variance proxy near $\lambda = 0$, and $1/\alpha$ is the half-width of the window, which limits how large a λ the Chernoff method may use. Section 36.9.2.1 shows that x^2 is sub-exponential with $\nu = 2$ and $\alpha = 4$. For an average of such variables, the Chernoff optimization splits at the edge of the window. The proof can be skipped on a first reading; the next part uses only the statement.

Proposition (sub-exponential tail bound). *Let $X_1, \dots, X_n$ be independent and (ν, α)-sub-exponential. Then for every $t > 0$,*

$$P(|\bar{X} - \mathbb{E}[\bar{X}]| \geq t) \leq 2\exp\left(-\frac{n}{2}\min\Big(\frac{t^2}{\nu^2}, \frac{t}{\alpha}\Big)\right). \tag{36.120}$$

Proof. For $0 < \lambda \leq 1/\alpha$, the Chernoff method and the product rule bound $P(\bar{X} - \mathbb{E}[\bar{X}] \geq t)$ by $\exp(-\lambda nt + n\lambda^2\nu^2/2)$. If $t \leq \nu^2/\alpha$, the parabola's minimizer $\lambda = t/\nu^2$ lies in the window and gives $e^{-nt^2/(2\nu^2)}$. If $t > \nu^2/\alpha$, the minimizer lies outside the window, and we take the largest admissible value, $\lambda = 1/\alpha$. The exponent becomes $-nt/\alpha + n\nu^2/(2\alpha^2)$, and since $\nu^2/\alpha < t$ it is at most $-nt/\alpha + nt/(2\alpha) = -nt/(2\alpha)$. In both cases the exponent is at most $-\frac{n}{2}\min(t^2/\nu^2, t/\alpha)$. The lower tail follows in the same way from $\lambda \in [-1/\alpha, 0)$. ■

Averages of bounded or sub-Gaussian terms have tails e^{-ct^2} at every scale. A sub-exponential average has such a tail only for deviations below ν^2/α, and an exponential tail, $e^{-nt/(2\alpha)}$, beyond it.

36.9.2 Probability in High Dimension

The bounds so far protect a single average. A random vector in $\mathbb{R}^d$ has d independent coordinates, and its length and its angles with other vectors are averages over those coordinates. Concentration therefore governs the geometry of high-dimensional random vectors. A reader who starts here needs two results from above: the sub-exponential tail bound Equation 36.120 for squared lengths and Hoeffding's inequality in the form Equation 36.115 for inner products.

36.9.2.1 The Thin Shell

Draw $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_{784})$, a Gaussian vector with as many coordinates as a 28×28 image. Its density is largest at the origin. Yet the probability that $\mathbf{x}$ lands within distance 14 of the origin, half its typical length, is about 10^{-110}, the probability that a sum of 784 squared standard Gaussians is at most 196. No sample we could ever draw will contain such a point. A typical draw has norm 28.0 ± 0.7.

The reason is that $\|\mathbf{x}\|^2 = \sum_{i=1}^{d} x_i^2$ is a sum of d independent terms with mean 1, so $\|\mathbf{x}\|^2/d$ is a sample mean. Its terms are unbounded, so Hoeffding's inequality does not apply, but they are sub-exponential.

Proposition (norm concentration). *Let* $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$. *For every* $0 < \varepsilon \le 1$,

$$P\left(\left|\frac{\|\mathbf{x}\|}{\sqrt{d}} - 1\right| \ge \varepsilon\right) \le 2\exp\left(-\frac{d\,\varepsilon^2}{8}\right). \tag{36.121}$$

Proof. The proof has three steps. First, the moment generating function of a centered square, $e^{-\lambda}/\sqrt{1-2\lambda}$, satisfies

$$\mathbb{E}\big[e^{\lambda(x_i^2-1)}\big] \le e^{2\lambda^2} \qquad \text{for } |\lambda| \le \tfrac{1}{4}, \tag{36.122}$$

by a series comparison (Exercise 5). In the language of Equation 36.119, x_i^2 is sub-exponential with $\nu = 2$ and $\alpha = 4$. Second, the sub-exponential tail bound Equation 36.120 with $n = d$ and $t = \varepsilon$ gives

$$P\Big(\Big|\frac{\|\mathbf{x}\|^2}{d} - 1\Big| \ge \varepsilon\Big) \le 2\exp\Big(-\frac{d}{2}\min\Big(\frac{\varepsilon^2}{4}, \frac{\varepsilon}{4}\Big)\Big) = 2e^{-d\varepsilon^2/8} \qquad \text{for } \varepsilon \le 1.$$

Third, we pass from squares to norms. Write $z = \|\mathbf{x}\|/\sqrt{d} \ge 0$. If $|z-1| \ge \varepsilon$, then $|z^2 - 1| = |z-1|\,(z+1) \ge \varepsilon$, because $z + 1 \ge 1$. So the event for the norm is contained in the event for the squared norm and inherits its bound. ■

The norm of a standard Gaussian vector misses $\sqrt{d}$ by a fixed fraction with a probability that falls exponentially in d. The restriction $\varepsilon \le 1$ matters only for the squared norm, whose tail beyond $\varepsilon = 1$ is exponential, $e^{-d\varepsilon/8}$ (Exercise 5). The norm keeps its Gaussian tail at

every scale: a norm above $(1+\varepsilon)\sqrt{d}$ means a squared norm above $(1+2\varepsilon+\varepsilon^2)d$, which has probability at most $e^{-d(2\varepsilon+\varepsilon^2)/8} \le e^{-d\varepsilon^2/8}$.

How thick is the shell? Since $\mathbb{E}[x^4] = 3$, each square has $\mathrm{Var}(x^2) = 3-1 = 2$, so $\mathrm{Var}(\|\mathbf{x}\|^2) = 2d$. The tangent line of the square root at d gives $\|\mathbf{x}\| \approx \sqrt{d} + (\|\mathbf{x}\|^2 - d)/(2\sqrt{d})$, so the standard deviation of $\|\mathbf{x}\|$ is about $\sqrt{2d}/(2\sqrt{d}) = 1/\sqrt{2} \approx 0.71$, whatever the dimension. A standard Gaussian in $\mathbb{R}^d$ is a shell of radius $\sqrt{d}$ and thickness of order one.

Why is the mass so far from the mode? The density $(2\pi)^{-d/2} e^{-\|\mathbf{x}\|^2/2}$ depends on $\mathbf{x}$ only through $r = \|\mathbf{x}\|$, and the sphere of radius r has a surface area proportional to r^{d-1}. The density of $\|\mathbf{x}\|$ is therefore proportional to $r^{d-1} e^{-r^2/2}$. The first factor grows with r, the second falls, and their product peaks where the derivative of $(d-1)\log r - r^2/2$ vanishes, at $r = \sqrt{d-1}$. Panel (a) of Figure 36.38 shows the two factors and their product at $d = 20$.

At a typical point, of norm $\sqrt{d}$, the density is $e^{-d/2}$ times its value at the origin, 10^{-170} at $d = 784$, but there are vastly more such points. Panel (b) shows the exact density of $\|\mathbf{x}\|$ for d from 2 to 512. The centers move out like $\sqrt{d}$ while the widths stay near $1/\sqrt{2}$.

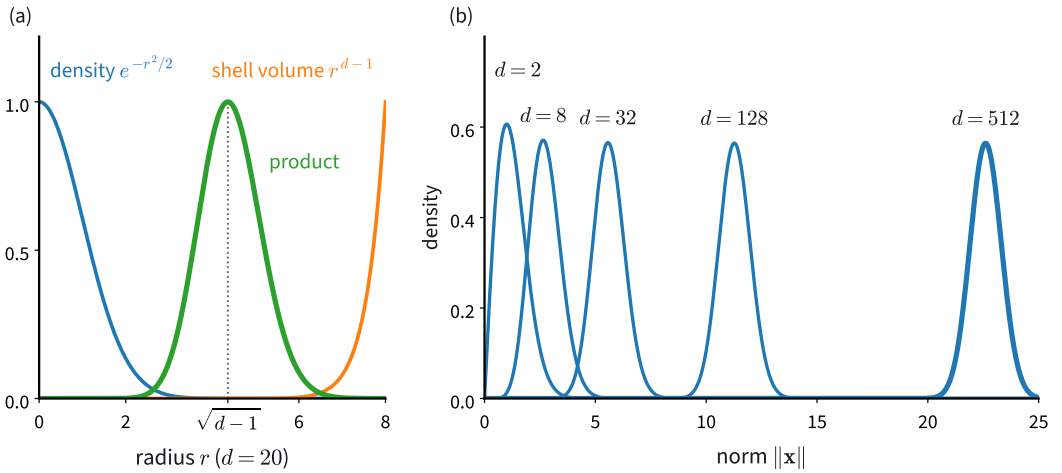


Figure 36.38: The thin shell. (a) At $d = 20$ the Gaussian density $e^{-r^2/2}$ falls with the radius r, the volume r^{d-1} of the shell at radius r grows, and their product, the density of $\|\mathbf{x}\|$, peaks at $\sqrt{d-1}$, far from the origin where the density is largest; each curve is scaled to maximum 1. (b) The exact density of $\|\mathbf{x}\|$ for $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ and $d = 2, 8, 32, 128, 512$. The centers move out like $\sqrt{d}$; the widths stay near $1/\sqrt{2}$.

The bound captures the form of the number we started from, but not its size. A norm below 14 at $d = 784$ means $\varepsilon = \frac{1}{2}$, and Equation 36.121 gives $2e^{-784/32} = 2e^{-24.5} \approx 5 \times 10^{-11}$, against the exact 10^{-110}. The constant 8 in the exponent is loose; the exponential dependence on d is right.

36.9.2.2 Near-Orthogonality of Random Directions

Section 33.1 showed that the cosine between a fixed unit vector and a random direction in $\mathbb{R}^d$ has variance $1/d$, which with Chebyshev's inequality gives a tail of order $1/(d\varepsilon^2)$.

Concentration replaces it by an exponential tail, and with that tail we can count how many random directions are nearly orthogonal at once.

Vectors of random signs are the cleanest case. Fix a unit vector $\mathbf{u}$ and let the $x_i \in \{-1, +1\}$ be independent uniform signs. Each term $u_i x_i$ of $\langle \mathbf{x}, \mathbf{u} \rangle$ is centered and lies in $[-|u_i|, |u_i|]$, an interval of length $2|u_i|$. In Equation 36.115 the denominator is then $4 \sum_i u_i^2 = 4$, and

$$P(|\langle \mathbf{x}, \mathbf{u} \rangle| \geq s) \leq 2e^{-s^2/2}. \tag{36.123}$$

The inner product of a random sign vector with any fixed unit vector is of order one, with a Gaussian tail: each coordinate contributes $\pm|u_i|$, and the signs cancel. The norm, meanwhile, is exactly $\sqrt{d}$, so with $s = \varepsilon\sqrt{d}$ we get $P(|\cos(\mathbf{x}, \mathbf{u})| \geq \varepsilon) \leq 2e^{-d\varepsilon^2/2}$. For a Gaussian $\mathbf{x}$ the same rate holds approximately: $\langle \mathbf{x}, \mathbf{u} \rangle$ is a standard Gaussian whatever $\mathbf{u}$ is, with the same tail $2e^{-s^2/2}$ (Exercise 1), and the norm is within a few percent of $\sqrt{d}$ by the thin shell.

The exponential tail lets us count, and the count needs one more tool. The *union bound* says that the probability that at least one of several events occurs is at most the sum of their probabilities, whether or not the events are independent. Take m independent sign vectors. They form $m(m-1)/2 < m^2/2$ pairs, and for each pair the cosine exceeds ε in absolute value with probability at most $2e^{-d\varepsilon^2/2}$. By the union bound, the probability that any pair does is below $m^2 e^{-d\varepsilon^2/2}$. So with probability at least $1 - \delta$ all pairwise cosines are below ε whenever $m \leq \sqrt{\delta}\, e^{d\varepsilon^2/4}$, a number that grows exponentially in d.

At $d = 784$ and $\delta = 0.05$ this guarantees about 570 directions with all pairwise $|\cos| < 0.2$, and about 10^7 with $|\cos| < 0.3$. These are guarantees from a loose bound, not estimates of how many such directions exist. A typical $|\cos|$ is much smaller, $\sqrt{2/(\pi d)} \approx 0.028$, the mean absolute value of a Gaussian with standard deviation $1/\sqrt{d}$.

Trained embeddings are not independent random directions. A cosine of 0.3 at $d = 784$ lies more than eight standard deviations ($1/\sqrt{784} = 0.036$) above what independent directions produce, so they serve as the null model against which structure in embeddings is measured.

36.9.2.3 The Thin Shell in Numbers

The bounds above have loose constants, so we measure. The tangent-line calculation predicts a standard deviation of the norm near $1/\sqrt{2}$ at every d, and a Gaussian with standard deviation $1/\sqrt{d}$ predicts a mean $|\cos|$ of $\sqrt{2/(\pi d)}$. We draw 10,000 pairs of vectors at $d = 2, 20, 200, 2000$; one row will not fit.

```
for d in (2, 20, 200, 2000):
    x, y = rng.standard_normal((2, 10000, d))
    nx = np.linalg.norm(x, axis=1)
    cos = (x * y).sum(1) / (nx * np.linalg.norm(y, axis=1))
    print(f'd = {d:4d}: mean ||x||/sqrt(d) = {(nx / np.sqrt(d)).mean():.3f}, '
          f'sd ||x|| = {nx.std():.2f}, mean |cos| = {np.abs(cos).mean():.4f}'
          f' vs {np.sqrt(2 / (np.pi * d)):.4f}')
```

```
d =    2: mean ||x||/sqrt(d) = 0.886, sd ||x|| = 0.65, mean |cos| = 0.6377 vs 0.5642
d =   20: mean ||x||/sqrt(d) = 0.986, sd ||x|| = 0.71, mean |cos| = 0.1827 vs 0.1784
d =  200: mean ||x||/sqrt(d) = 0.999, sd ||x|| = 0.70, mean |cos| = 0.0560 vs 0.0564
d = 2000: mean ||x||/sqrt(d) = 1.000, sd ||x|| = 0.71, mean |cos| = 0.0179 vs 0.0178
```

The standard deviation of the norm is about 0.65 at $d = 2$ and about 0.7 after that, while the norm itself grows from about 1.3 to about 45. The row that does not fit is $d = 2$: its mean $|\cos|$ is about 0.64. That is $2/\pi$, the mean of $|\cos\theta|$ for a uniform angle; the formula $\sqrt{2/(\pi d)}$ describes high dimensions only. From $d = 20$ on, the measured values, about 0.18, 0.057 and 0.018, agree with it to within a few percent.

Two further questions. How much of the mass lies within 10% of the typical norm, in the shell $(1 \pm 0.1)\sqrt{d}$? And do distances still separate near points from far ones? We measure the second by the *contrast* of a query point, $(d_{\max} - d_{\min})/d_{\min}$ over its distances to 200 random points, and report the median over 100 queries.

```
eps = 0.1
for d in (2, 20, 200, 2000):
    x = rng.standard_normal((10000, d))
    ratio = np.linalg.norm(x, axis=1) / np.sqrt(d)
    shell = np.mean(np.abs(ratio - 1) <= eps)          # mass in the 10% shell
    query, points = x[:100], x[100:300]                # 100 queries, 200 points
    dist = np.linalg.norm(query[:, None] - points[None], axis=2)  # (100, 200)
    contrast = (dist.max(1) - dist.min(1)) / dist.min(1)
    print(f'd = {d:4d}: mass in (1 ± 0.1) sqrt(d) = {shell:6.1%}, '
          f'median contrast = {np.median(contrast):.3f}')
```

```
d =    2: mass in (1 ± 0.1) sqrt(d) =  14.5%, median contrast = 30.417
d =   20: mass in (1 ± 0.1) sqrt(d) =  46.7%, median contrast = 1.179
d =  200: mass in (1 ± 0.1) sqrt(d) =  95.6%, median contrast = 0.270
d = 2000: mass in (1 ± 0.1) sqrt(d) = 100.0%, median contrast = 0.079
```

At $d = 2$ the shell holds about 15% of the mass, at $d = 20$ about 47%, and at $d = 200$ about 95%. The last number is what the tangent-line calculation predicts: the ratio $\|\mathbf{x}\|/\sqrt{d}$ has standard deviation $1/\sqrt{2d} = 0.05$ at $d = 200$, so the shell spans two standard deviations on either side. At $d = 2000$ every one of the 10,000 draws lies in the shell. The median contrast falls from about 30 at $d = 2$ (between roughly 25 and 40 from run to run) to about 1.2, 0.27 and 0.08. At $d = 2000$ the farthest of 200 random points is, for a typical query, less than 10% farther from it than the nearest.

36.9.2.4 Consequences for Deep Learning

Three practices in deep learning rest on these facts.

Initialization. LeCun initialization (LeCun, Leon Bottou, et al. 1998) draws the entries of a $d \times d$ weight matrix independently with variance $1/d$, and He initialization (He et al. 2015) uses $2/d$ for ReLU networks (Section 4.4). The argument for these scales computes an expectation over all draws of the weights: at LeCun's scale each pre-activation $\langle \mathbf{w}, \mathbf{x} \rangle$ has variance $\|\mathbf{x}\|^2/d$, which norm concentration puts close to 1 for inputs with coordinates of unit variance. We train one draw. Concentration says when the expectation describes it.

For one He-initialized ReLU layer of width d, the ratio $\rho = \|\mathbf{h}_k\|^2/\|\mathbf{h}_{k-1}\|^2$ of squared activation norms has mean 1 and variance $5/d$ (Exercise 6), so one layer changes the squared

norm by about $\sqrt{5/d}$, 10% at $d = 512$. Through K layers the fluctuations add up in the logarithm, and $\log(\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2)$ has variance about $5K/d$. The variance calculation therefore describes a typical network only while $K \ll d$. At $K = 50$ and $d = 512$ the log-variance is about 0.5, so the squared norm of a typical network is off by a factor of two in either direction.

Distances and nearest neighbors. For two independent draws, $\mathbf{x} - \mathbf{y} \sim \mathcal{N}(\mathbf{0}, 2\mathbf{I})$, so norm concentration applies to $(\mathbf{x} - \mathbf{y})/\sqrt{2}$: each distance lies in $(1 \pm \varepsilon)\sqrt{2d}$ except with probability $2e^{-d\varepsilon^2/8}$. The pair count used for directions then puts all distances among n points in that band at once when $d \geq (8/\varepsilon^2)\log(n^2/\delta)$, a dimension that grows only with the logarithm of the number of points. When all distances are nearly equal, the nearest and the farthest neighbors of a query differ by a small relative margin, as the contrast measured above shows. Real embeddings keep their contrast because their data lie near a set of low intrinsic dimension. The same calculation also gives a construction: a random projection to k dimensions preserves all pairwise distances among n points once k is of order $\log n$, whatever the original dimension (Exercise 7).

Latent codes. A decoder trained on codes $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ sees inputs of norm close to $\sqrt{d}$. The prior mean, of norm 0, and the midpoint of the chord between two codes, of norm close to $\sqrt{d/2}$, are inputs it never saw during training. For this reason, latent codes are usually interpolated along the sphere rather than the chord (Exercise 3).

Every bound in this section protects a quantity fixed *before* the data were drawn. A learner picks the best of many candidates *after* seeing the data, and the best of many fair coins looks biased. Section 36.10 shows how to account for that choice.

36.9.3 Summary

How far can an average of n independent terms be from its expectation, at the n we have? Chebyshev's inequality answers with a probability that falls like $1/n$, too slowly for a guarantee that must almost never fail. The Chernoff method applies Markov's inequality to $e^{\lambda X}$, where independence turns the moment generating function of a sum into a product, and optimizes over λ. With Hoeffding's lemma it gives, for losses in $[0, 1]$, the tail $2e^{-2nt^2}$ and an interval of half-width $\sqrt{\log(2/\delta)/(2n)}$ that is valid at every n; Bernstein's inequality tightens it when the variance is small. Applied to the d coordinates of a random vector, the same calculation shows that a Gaussian vector lies in a shell of radius $\sqrt{d}$ and thickness about $1/\sqrt{2}$, and that exponentially many random directions are pairwise nearly orthogonal. These results size test sets, tell when an initialization scale computed in expectation describes the one network we train, and give the null model for cosines between embeddings.

36.9.4 Exercises

1. **Hoeffding with unequal ranges.** The proof of Equation 36.114 assumed that every X_i takes values in the same interval $[a, b]$.
   1. Redo the proof for independent $X_i \in [a_i, b_i]$ and show that

$$P(|\bar{X} - \mathbb{E}[\bar{X}]| \geq t) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right).$$

Check that this bound reduces to Equation 36.114 when all ranges agree.

2. Let $x_1, \dots, x_d$ be independent uniform random signs and let $\mathbf{u} \in \mathbb{R}^d$ be a unit vector. Apply the bound of (a) to the sum $\sum_i u_i x_i$ and derive the tail bound for the inner product, Equation 36.123 in the text.
3. Now let the x_i be independent standard Gaussians. Derive the same bound $2e^{-s^2/2}$ from the Gaussian moment generating function $\mathbb{E}[e^{\lambda g}] = e^{\lambda^2/2}$ and the Chernoff method Equation 36.112, and explain why the bound of (a) cannot be applied here.

2. **Sub-Gaussian random signs.** A random sign ξ takes the values -1 and $+1$ with probability $\tfrac{1}{2}$ each, so that $\mathbb{E}[e^{\lambda \xi}] = \cosh \lambda \le e^{\lambda^2/2}$.
   1. Conclude with the Chernoff method Equation 36.112 that the average $\bar{\xi}$ of n independent random signs satisfies $P(|\bar{\xi}| \ge t) \le 2e^{-nt^2/2}$.
   2. Hoeffding's lemma Equation 36.113 with $[a,b] = [-1,1]$ also gives a variance proxy for ξ. Compare it with the bound $\cosh \lambda \le e^{\lambda^2/2}$. Expand $\log \cosh \lambda = \lambda^2/2 - \lambda^4/12 + \cdots$ around $\lambda = 0$, and explain why no constant smaller than $\tfrac{1}{8}$ can replace the $\tfrac{1}{8}$ in Equation 36.113 for all bounded variables.
3. **Latent interpolation.** A generative model decodes latent codes $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ with $d = 512$. Let $\mathbf{z}_1, \mathbf{z}_2$ be two independent codes and $\mathbf{z}(\gamma) = (1-\gamma)\mathbf{z}_1 + \gamma \mathbf{z}_2$ for $\gamma \in [0,1]$ their linear interpolation.
   1. Find the distribution of $\mathbf{z}(\gamma)$ and the approximate value of $\|\mathbf{z}(\gamma)\|$. At which γ is the norm smallest?
   2. Use Equation 36.121 to bound the probability that the midpoint $\mathbf{z}(\tfrac{1}{2})$ has norm at least $0.9\sqrt{d}$, that is, reaches the shell $(1 \pm 0.1)\sqrt{d}$ in which typical codes lie. Then compute the exact probability, which is $P(\chi^2_{512} \ge 2 \cdot 0.81 \cdot 512)$ for a chi-square variable with 512 degrees of freedom (`scipy.stats.chi2.sf`). By how many orders of magnitude is the bound loose?
   3. Spherical interpolation (White 2016) replaces the chord by the arc
   $$\tilde{\mathbf{z}}(\gamma) = \frac{\sin((1-\gamma)\theta)}{\sin \theta}\, \mathbf{z}_1 + \frac{\sin(\gamma \theta)}{\sin \theta}\, \mathbf{z}_2, \tag{36.124}$$
   where θ is the angle between $\mathbf{z}_1$ and $\mathbf{z}_2$. Show that $\|\tilde{\mathbf{z}}(\gamma)\| = r$ for all γ when $\|\mathbf{z}_1\| = \|\mathbf{z}_2\| = r$. Which value does θ take approximately for $d = 512$, and why?
4. code **Chernoff bound for coins.** ⊠ Let $\hat{p}$ be the fraction of heads in n independent flips of a coin with bias p, and let $a > p$.
   1. Apply the Chernoff method Equation 36.112 to $n\hat{p}$, whose moment generating function is $(1 - p + pe^{\lambda})^n$. Optimize over λ and show that $P(\hat{p} \ge a) \le e^{-n\, D_{\mathrm{KL}}(a \| p)}$, where $D_{\mathrm{KL}}(a\|p) = a \log \frac{a}{p} + (1-a) \log \frac{1-a}{1-p}$.
   2. Show that $D_{\mathrm{KL}}(a\|p) \ge 2(a-p)^2$ by examining the second derivative in a, so that the bound of (a) is never weaker than one side of Hoeffding's bound. Evaluate both exponents for $p = \tfrac{1}{2}$ and $a = 0.6$.
   3. Add the two-sided bound $2e^{-n\, D_{\mathrm{KL}}(0.6 \| 0.5)}$ as a column to the table of Section 36.9.1.5, for $n \in \{10, 100, 1000, 5000\}$. The ratio of the exact tail to this bound still decreases with n. How does it scale with n, and what does the scaling say about the factor that the Chernoff method loses?
5. **Chi-square tail.** ⊠ The proof of Equation 36.121 rests on the bound Equation 36.122 for $x^2 - 1$ with $x \sim \mathcal{N}(0,1)$, which the proof states without derivation.
   1. Show that $\mathbb{E}[e^{\lambda(x^2-1)}] = e^{-\lambda}/\sqrt{1-2\lambda}$ for $\lambda < \tfrac{1}{2}$ by completing the square in the Gaussian integral.

2. Expand $\log \mathbb{E}[e^{\lambda(x^2-1)}]$ into a power series in 2λ and prove Equation 36.122. The series starts at the quadratic term.
3. The proof of Equation 36.121 takes $\lambda = \varepsilon/4$, which is admissible only for $\varepsilon \le 1$. For $\varepsilon > 1$, set $\lambda = \frac{1}{4}$ and bound $P(\|\mathbf{x}\|^2 \ge (1+\varepsilon)d)$. Does the result decay like $e^{-c\varepsilon^2}$ or like $e^{-c\varepsilon}$ in ε, and which regime of Equation 36.120 does it belong to?

6. code **Activation norms through depth.** ⊠ Consider a ReLU network of width d and depth K without biases, $\mathbf{h}_k = \mathrm{ReLU}(\mathbf{W}_k \mathbf{h}_{k-1})$, whose weights are independent draws from $\mathcal{N}(0, 2/d)$ (He initialization, Section 4.4). Let $\rho_k = \|\mathbf{h}_k\|^2 / \|\mathbf{h}_{k-1}\|^2$. How such norms fluctuate with depth and width was analyzed by Hanin and Nica (2020) .
   1. Show that, given $\mathbf{h}_{k-1} \neq \mathbf{0}$, the ratio ρ_k is distributed as $\frac{2}{d}\sum_{j=1}^{d} \mathrm{ReLU}(g_j)^2$ with independent $g_j \sim \mathcal{N}(0,1)$, whatever the direction of $\mathbf{h}_{k-1}$. Conclude that $\mathbb{E}[\rho_k] = 1$ and $\mathrm{Var}(\rho_k) = 5/d$, using $\mathbb{E}[g^2] = 1$ and $\mathbb{E}[g^4] = 3$.
   2. Write $\log(\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2)$ as a sum of K independent terms ($\rho_k = 0$ has probability 2^{-d} per layer; ignore it). Expand $\log \rho$ to second order around 1 and predict the mean and variance of this sum in terms of K/d.
   3. Simulate $1{,}000$ networks for each width $d \in \{32, 128\}$ and depth $K \in \{1, 10, 50\}$, each with a fresh standard Gaussian input. Compare the mean and variance of $\log(\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2)$ with (b), and report the mean and the median of the ratio $\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2$ itself. The variance calculation of Section 4.4 gives $\mathbb{E}[\|\mathbf{h}_K\|^2/\|\mathbf{h}_0\|^2] = 1$ at every depth. Repeat the run at $K = 50$ and $d = 32$ several times. Why does the sample mean over $1{,}000$ networks scatter so widely around 1? Why does the expectation not describe a typical network there?

7. code **Random projections preserve distances.** ⊠ Let $\mathbf{A} \in \mathbb{R}^{k \times d}$ have independent $\mathcal{N}(0, 1/k)$ entries.
   1. Show that for a fixed $\mathbf{v} \neq \mathbf{0}$ the ratio $\|\mathbf{A}\mathbf{v}\|^2/\|\mathbf{v}\|^2$ has the distribution of $\|\mathbf{g}\|^2/k$ with $\mathbf{g} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_k)$, so that by Equation 36.121 $P(|\|\mathbf{A}\mathbf{v}\|/\|\mathbf{v}\| - 1| \ge \varepsilon) \le 2e^{-k\varepsilon^2/8}$. Hint: rotate so that $\mathbf{v}$ lies along the first axis.
   2. For n points, apply (a) to the $n(n-1)/2$ differences and a union bound. Show that all pairwise distances are preserved within a factor $1 \pm \varepsilon$ with probability at least $1 - \delta$ once $k \ge (8/\varepsilon^2)\log(n^2/\delta)$, a bound that does not depend on d. This is the Johnson–Lindenstrauss lemma. Evaluate it for $n = 1{,}000$, $\varepsilon = 0.3$ and $\delta = 0.05$.
   3. Verify it on $1{,}000$ standard Gaussian points in $d = 2{,}000$. Project to $k \in \{50, 200, 800\}$ and report the largest distortion $|\|\mathbf{A}(\mathbf{x}_i - \mathbf{x}_j)\|/\|\mathbf{x}_i - \mathbf{x}_j\| - 1|$ over all pairs, together with the typical fluctuation $1/\sqrt{2k}$. Compute the pairwise distances from the Gram matrix, $\|\mathbf{x}_i - \mathbf{x}_j\|^2 = \|\mathbf{x}_i\|^2 + \|\mathbf{x}_j\|^2 - 2\,\mathbf{x}_i^\top \mathbf{x}_j$, not by broadcasting a $1{,}000 \times 1{,}000 \times 2{,}000$ array. Explain the ratio between the two numbers from the number of pairs. Already at $k = 200$ the largest distortion is below $\varepsilon = 0.3$, while (b) asked for a much larger k. Which part of the bound is loose: the constant 8 or the $\log n$?

8. **Bernstein's inequality.** ⊠ Let X be centered with $|X| \le M$ and $\mathrm{Var}(X) = \sigma^2$.
   1. Show that $\mathbb{E}[X^k] \le \sigma^2 M^{k-2}$ for $k \ge 2$, and hence that $\mathbb{E}[e^{\lambda X}] \le 1 + \sigma^2(e^{\lambda M} - 1 - \lambda M)/M^2$ for $\lambda > 0$.
   2. Show that $e^u - 1 - u \le u^2/(2(1-u/3))$ for $0 \le u < 3$ by comparing $u^k/k!$ with $(u^2/2)(u/3)^{k-2}$, that is, by using $k! \ge 2 \cdot 3^{k-2}$. Conclude with $1 + y \le e^y$ that $\mathbb{E}[e^{\lambda X}] \le \exp(\lambda^2\sigma^2/(2(1-\lambda M/3)))$ for $0 < \lambda < 3/M$.
   3. Apply the Chernoff method to an average of n such variables with $\lambda = t/c$, where $c = \sigma^2 + Mt/3$. Check that $\lambda < 3/M$ and that $1 - \lambda M/3 = \sigma^2/c$, and conclude

that the exponent is exactly $-nt^2/(2c)$, which proves Equation 36.117.

4. Show that (b) makes X sub-exponential in the sense of Equation 36.119 with $\nu^2 = 2\sigma^2$ and $\alpha = 2M/3$.
5. With $n = 10{,}000$ and $\delta = 0.05$, show that Bernstein's bound is weaker than Hoeffding's at the worst-case variance $\sigma^2 = \frac{1}{4}$ of a 0/1 outcome. Below which variance does it win, and which error rates does that correspond to?

9. **Comparing two classifiers on one test set.** ⊠ Classifiers A and B are scored on the same $n = 10{,}000$ examples. For 0/1 losses let $D_i = \ell_A(i) - \ell_B(i) \in \{-1, 0, 1\}$, so that the mean of the D_i is the difference of the two error rates.
   1. Apply Hoeffding's inequality to the D_i, whose range has length 2. Show that the half-width at $\delta = 0.05$ is about $\pm 2.7\%$, so a difference of one percentage point cannot be certified this way.
   2. Suppose the two models disagree on a fraction q of the examples. Show that $\mathrm{Var}(D_i) \le q$ and $|D_i - \mathbb{E}[D_i]| \le 2$, and apply Equation 36.117 with $M = 2$. Compute the half-widths for $q = 0.05$, 0.10 and 0.20. Below which disagreement rate can a one-point gap be certified?
   3. Compare with the asymptotic Gaussian interval $1.96\sqrt{q/n}$. What does Bernstein give up in exchange for validity at every n?
   4. Why is scoring both models on the *same* examples better than scoring them on two independent test sets of 10,000 examples each? Hint: compare the variance $2p(1-p)/n$ of a difference of two independent error estimates at $p = 0.083$ with q/n at $q = 0.05$.
10. code **Mode versus mass.** Let $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$.
    1. Show that the density at a point of norm $\sqrt{d}$ is $e^{-d/2}$ times the density at the origin, and evaluate this ratio at $d = 784$.
    2. Show that the density of $\|\mathbf{x}\|$ is proportional to $r^{d-1}e^{-r^2/2}$ and peaks at $r = \sqrt{d-1}$.
    3. Compute $P(\|\mathbf{x}\| \le \sqrt{d}/2)$ at $d = 784$ in three ways: with the bound Equation 36.121, exactly with `scipy.stats.chi2.cdf`, and by Monte Carlo with 10^6 draws. Which statement would you quote, and why does the exponent rather than the constant carry the message?
    4. A decoder with $d = 512$ latent dimensions is evaluated at the prior mean $\mathbf{0}$. Using the standard deviation $1/\sqrt{2}$ of the norm, how many standard deviations is that input from the norm of a typical code?

36.10 Uniform Convergence and Generalization

The previous section put an error bar on a number computed from data. A classifier scored on 10,000 held-out images has a true error within about 1.4 percentage points of its test error, with 95% confidence. The guarantee rests on a hypothesis that is easy to overlook: the classifier was fixed before the test images were drawn. Little of what we report is like that. We train many configurations, look at their validation scores, and keep the best one. The reported score belongs to the winner of a contest, and the winner is partly the configuration whose luck was best.

Chapter 3.6 met this problem and gave the remedy of Vapnik and Chervonenkis (Vapnik and Chervonenkis 1971): ask for a guarantee that holds for every candidate function at once, so that it also holds for whichever one the learner picks. For a finite list of candidates the guarantee widens only with the logarithm of their number. We only recall that result here,

with one experiment that shows how much luck there is to choose from: flip each of 1,000 fair coins 100 times and keep the coin with the most heads. How many does it show?

The main work of the section is the infinite classes that we actually train. For them a later idea measures a class by how well it can fit labels that are pure coin flips, its Rademacher complexity. It explains why the capacity of a linear model depends on its weight norm rather than on its number of parameters, and it shows where this way of arguing stops: at models that fit random labels, as the networks of the main text do. The last part repeats the double-descent experiment of Section 4.5.2 in a model small enough that its mechanism can be computed.

```
rng = np.random.default_rng()
```

36.10.1 Uniform Convergence

A test-set certificate and a learner differ only in the order of two events: choosing the function and drawing the sample. This part measures how much choosing after the sample distorts the score, and how much more data a guarantee needs once the order is reversed.

36.10.1.1 The Best of Many Coins

We first state the guarantee for one function in the notation used below. A sample $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ consists of n independent draws from an unknown distribution. A loss ℓ with values in $[0, 1]$ scores a prediction, for instance the 0-1 loss, which is 1 for a mistake and 0 otherwise.

The *empirical risk* of a function f is its average loss on the sample, $\hat{R}(f) = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\mathbf{x}_i), y_i)$, and its *risk* is the expected loss on a fresh draw, $R(f) = \mathbb{E}[\ell(f(\mathbf{x}), y)]$. (For the 0-1 loss these are the empirical error $\epsilon_S(f)$ and the population error $\epsilon(f)$ of the main text.) If f is fixed before S is drawn, $\hat{R}(f)$ is an average of n independent terms in $[0, 1]$ with mean $R(f)$, and Hoeffding's inequality (Section 36.9) bounds $P(|\hat{R}(f) - R(f)| \geq t)$ by $2e^{-2nt^2}$. Setting this to $\delta = 0.05$ and solving for t gives the half-width $\sqrt{\log(2/\delta)/(2n)}$, which the main text calls the Hoeffding radius. For $n = 10{,}000$ it is 0.0136, the 1.4 points above.

A learner reverses the order. It searches a class $\mathcal{F}$ of functions and returns the $\hat{f}$ with the smallest $\hat{R}$ on the sample it saw. Each $\hat{R}(f)$ fluctuates around its own $R(f)$, and the learner keeps the function whose fluctuation was most favorable. The empirical risk $\hat{R}(\hat{f})$ is therefore a minimum of fluctuating quantities, and an inequality that holds for each f separately does not cover it.

The coins show by how much. Let $k = 1{,}000$ fair coins each be flipped $m = 100$ times. For one coin named in advance, Hoeffding at $\delta = 0.05$ allows a deviation of $\sqrt{\log 40/200} = 0.136$ in the fraction of heads, so the coin shows between 37 and 63 heads with probability at least 95% (the exact probability is 99.3%). The best of the 1,000 coins has no such promise. We repeat the experiment 2,000 times and record how far the best coin strays and how often it leaves the single-coin band.

```
k, m, repeats = 1000, 100, 2000          # coins, flips per coin, experiments
best = rng.binomial(m, 0.5, size=(repeats, k)).max(axis=1) / m
one = np.sqrt(np.log(2 / 0.05) / (2 * m))       # Hoeffding, one fixed coin
many = np.sqrt(np.log(2 * k / 0.05) / (2 * m))  # union bound over k coins
print(f'best of {k} coins: median {np.median(best):.2f}, 90% of experiments '
      f'in [{np.percentile(best, 5):.2f}, {np.percentile(best, 95):.2f}]')
print(f'95% half-width: one coin {one:.3f}, all {k} coins {many:.3f}')
print(f'experiments in which the best coin leaves the one-coin band: '
      f'{np.mean(best > 0.5 + one):.0%}')
```

```
best of 1000 coins: median 0.66, 90% of experiments in [0.64, 0.69]
95% half-width: one coin 0.136, all 1000 coins 0.230
experiments in which the best coin leaves the one-coin band: 97%
```

The best coin shows 66 heads in a typical experiment and between 64 and 69 in nine experiments out of ten. It leaves the single-coin band in 96 to 97% of the experiments, so a guarantee that is valid for every coin named in advance fails almost always for the coin chosen afterwards. No coin is biased. The selected coin is off by 16 points because we looked at 1,000 fluctuations and reported the largest.

Figure 36.39 shows why. Each coin leaves the band with probability 0.0033 on the upper side, a rare event, but among 1,000 coins about 3.3 are expected to do so. The tail that one coin almost never reaches is populated when a thousand coins are drawn.

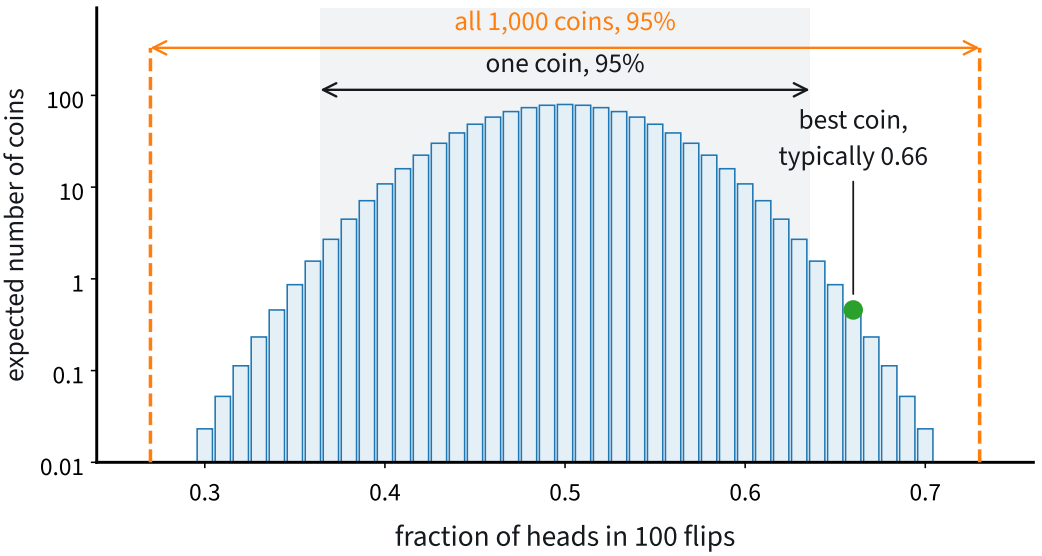


Figure 36.39: The expected histogram of the fraction of heads of 1,000 fair coins flipped 100 times each, on a logarithmic scale. The single-coin 95% band from Hoeffding's inequality covers 99% of the coins, yet about 3.3 coins are expected beyond its upper edge, and the best coin lies near 0.66 in a typical experiment. The band of the union bound for all 1,000 coins is wider and covers the best coin.

A validation set is the same experiment with the roles renamed: each configuration is a coin, each example a flip. Its configurations share their examples and differ in their true errors,

but the bound below needs neither independence nor equal means.

What would cover the chosen function? If the largest deviation over the whole class,

$$\sup_{f \in \mathcal{F}} \left| \hat{R}(f) - R(f) \right|,$$

is at most t, then every function's empirical risk is within t of its risk, including the one the learner picked. Bounds on this supremum are called *uniform convergence* bounds (Vapnik and Chervonenkis 1971). Choosing after seeing the data is safe if the guarantee was made for all candidates before seeing it, because the one we pick is one of them.

36.10.1.2 Finite Classes and the Union Bound

The simplest class is a finite list of candidates fixed in advance, such as a grid of hyperparameter configurations. For it the main text already has the bound, Equation 3.32, and its proof takes two steps. Hoeffding's inequality bounds the probability that one fixed f deviates by t or more by $2e^{-2nt^2}$, and the union bound (Section 36.9) over the $|\mathcal{F}|$ such events gives $2|\mathcal{F}|e^{-2nt^2}$. Setting this to δ and solving for t: for a loss with values in $[0,1]$ and an i.i.d. sample of size n, with probability at least $1-\delta$,

$$\left| \hat{R}(f) - R(f) \right| \le \sqrt{\frac{\log(2|\mathcal{F}|/\delta)}{2n}} \qquad \text{simultaneously for every } f \in \mathcal{F}. \tag{36.125}$$

Compared with one function, choosing among $|\mathcal{F}|$ candidates adds $\log|\mathcal{F}|$ under the square root, and nothing else.

On the coins, $|\mathcal{F}| = 1{,}000$ and $n = 100$ give the half-width $\sqrt{\log(2{,}000/0.05)/200} = 0.230$, so with probability at least 95% every coin, the best one included, shows between 27 and 73 heads. The bound is rather conservative: the exact probability that some coin falls outside 27 to 73 heads is about 0.17%, against the 5% the bound allows. Almost all of this factor of 30 is lost in Hoeffding's inequality for the single coin, which leaves that range with probability 1.7×10^{-6} where the inequality allows 5×10^{-5}. The union bound is nearly exact for rare independent events. Conservative or not, the bound is valid, which the single-coin band was not.

For a validation set of 10,000 examples the main text lists the half-widths: 0.014 for one function and 0.030 for a million. Solving the bound for the sample size shows the dependence more directly:

$$n = \frac{\log(2/\delta) + \log|\mathcal{F}|}{2t^2}.$$

Each additional unit of $\log|\mathcal{F}|$ adds $1/(2t^2)$ examples. At a tolerance of $t = 0.01$, doubling the number of candidates costs $\log 2/(2 \cdot 10^{-4}) \approx 3{,}500$ more validation examples.

It is the exponential tail of Hoeffding's inequality that makes the dependence logarithmic. With Chebyshev's inequality in its place (a $[0,1]$-valued loss has variance at most $\frac{1}{4}$), the same union bound gives $t = \sqrt{|\mathcal{F}|/(4n\delta)}$. For 2,000 configurations on 10,000 examples at $\delta = 0.05$ this is $t = 1$, the whole range of the loss, where Hoeffding gives $t = 0.024$.

The proof uses one hypothesis, in its first step: the class is fixed before the sample is drawn. A public leaderboard breaks it, because each submission is chosen after seeing the scores of earlier ones on the same test set (Dwork et al. 2015). The guarantee then degrades from a deviation of order $\sqrt{\log k/n}$ for a fixed list of k candidates to one of order $\sqrt{k/n}$ for the majority vote that the main text builds from k random guessers (exercise 5).

36.10.2 Rademacher Complexity

The classes we train are infinite. Every weight vector $\mathbf{w} \in \mathbb{R}^d$ defines a different linear classifier, and a union bound over infinitely many events is vacuous. The classical remedies either replace the class by finitely many representatives (covering numbers) or count the labelings it can produce on n points (the VC dimension, sketched in Chapter 3.6). The tool of this part (Bartlett and Mendelson 2002) asks a question a practitioner can act on: how well can the class fit pure noise?

The two results of the section that are worth remembering are in this part: the uniform convergence theorem Equation 36.127 and the complexity $Br/\sqrt{n}$ of a norm-bounded linear class, Equation 36.129. The one hard step is the symmetrization in the proof of the first.

36.10.2.1 Correlation with Random Signs

If a class can fit labels that carry no information, it can also fit the noise in real labels, and then its empirical risks say little about its risks. To measure this we feed the class fictitious labels. Let $\varepsilon_1, \dots, \varepsilon_n$ be independent fair random signs, each $+1$ or -1 with probability $\frac{1}{2}$; they are called *Rademacher variables*. For a sign vector ε, the quantity $\frac{1}{n}\sum_i \varepsilon_i f(\mathbf{x}_i)$ is the correlation of f with the signs on the sample, and the best a class can do is its supremum over the class.

Definition. The *empirical Rademacher complexity* of a class $\mathcal{F}$ of real-valued functions on the points $S = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ is the expected best correlation the class achieves with random signs,

$$\widehat{\mathfrak{R}}_S(\mathcal{F}) = \mathbb{E}_\varepsilon \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \varepsilon_i \, f(\mathbf{x}_i) \right], \tag{36.126}$$

and its *Rademacher complexity* $\mathfrak{R}_n(\mathcal{F}) = \mathbb{E}_S[\widehat{\mathfrak{R}}_S(\mathcal{F})]$ is its average over samples of size n.

A class that can match any sign pattern scores high; a class that cannot choose scores zero. The scale is set by the range of the functions. For a class with values in $[-1, 1]$ each correlation is at most 1, and the value 1 is reached by the class of all functions into $[-1, 1]$, which contains $f(\mathbf{x}_i) = \varepsilon_i$ for every pattern (when the points are distinct). For a class with values in $[0, 1]$ the largest possible value is $\frac{1}{2}$: the best choice is $f(\mathbf{x}_i) = 1$ exactly where $\varepsilon_i = +1$, which scores the fraction of plus signs, $\frac{1}{2}$ on average. An unbounded class, such as the linear class below, has no such ceiling: if it fits every sign pattern on the sample, its complexity is at least 1, and it can be larger.

The smallest case with something to choose is the class of the two constant functions $\{+1, -1\}$. Its best correlation with a sign vector is $\max(\bar\varepsilon, -\bar\varepsilon) = |\bar\varepsilon|$, where $\bar\varepsilon$ is the average of the signs. For $n = 2$ the four patterns $(+,+), (+,-), (-,+), (-,-)$ give $|\bar\varepsilon| = 1, 0, 0, 1$, so $\widehat{\mathfrak{R}}_S = \frac{1}{2}$. For $n = 4$ the sum of the signs is ± 4 with probability $\frac{2}{16}$, ± 2 with probability

$\frac{8}{16}$ and 0 otherwise, which gives $\widehat{\mathfrak{R}}_S = \frac{3}{8}$. For large n the average $\bar{\varepsilon}$ is approximately normal with variance $1/n$, so $\widehat{\mathfrak{R}}_S \approx \sqrt{2/(\pi n)}$; at $n = 100$ the exact value is 0.0796 against 0.0798. Even a class with only two members correlates with noise at the rate $1/\sqrt{n}$, because the average of n signs has that spread. A class with one member has complexity 0: with nothing to choose, $\mathbb{E}[\frac{1}{n}\sum_i \varepsilon_i f(\mathbf{x}_i)] = 0$.

For a linear class the best correlation has a geometric meaning, which Figure 36.40 previews: it is the length of a signed sum of the inputs, divided by n. Section 36.10.2.3 proves this.

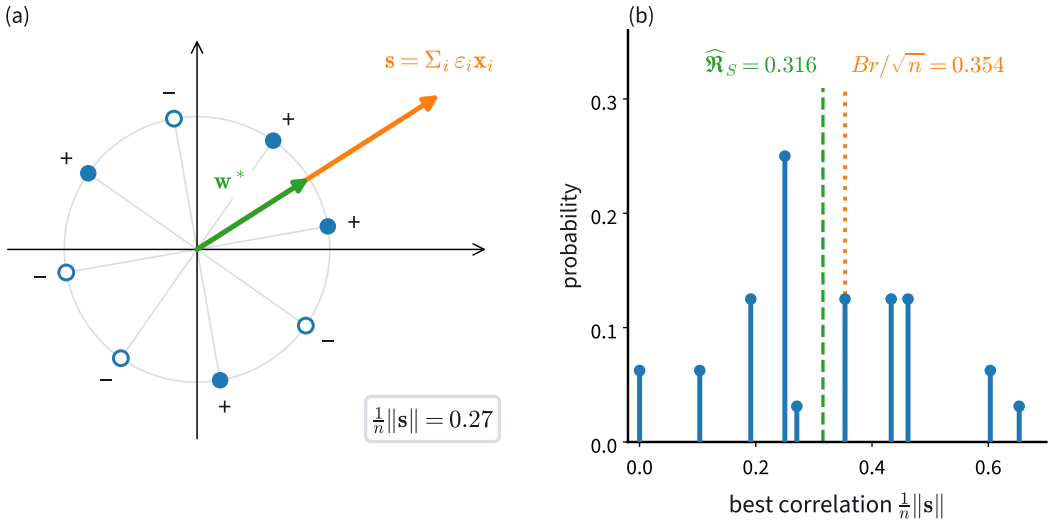


Figure 36.40: Rademacher complexity of a linear class, for eight unit vectors in the plane and the class of unit-norm linear functions. (a) For one sign pattern, filled dots are $+1$ and hollow dots -1. The best correlation is attained by the weight vector $\mathbf{w}^*$ aligned with the signed sum $\mathbf{s} = \sum_i \varepsilon_i \mathbf{x}_i$, and equals $\|\mathbf{s}\|/n = 0.27$. (b) The exact distribution of the best correlation over all 256 sign patterns. Its mean, the empirical Rademacher complexity, is 0.316, against the bound $Br/\sqrt{n} = 0.354$ proved in the text.

A finite class with values in $[-1, 1]$ obeys *Massart's lemma*, $\widehat{\mathfrak{R}}_S(\mathcal{F}) \le \sqrt{2\log|\mathcal{F}|/n}$ (exercise 2), so its complexity grows only logarithmically with its size. Rademacher complexity measures how freely the class can vary on the actual sample points, whatever the number of its parameters or members.

36.10.2.2 The Uniform Convergence Theorem

We now show that the Rademacher complexity bounds the largest deviation between empirical risk and risk. One distinction must be made first, because the predictions and the losses live in different classes. The predictor class $\mathcal{F}$ contains the functions we fit. The functions whose averages we compare are the losses, so define the *loss class*

$$\mathcal{G} = \{(\mathbf{x}, y) \mapsto \ell(f(\mathbf{x}), y) : f \in \mathcal{F}\},$$

with values in $[0, 1]$. For $g \in \mathcal{G}$ write $z = (\mathbf{x}, y)$, $R(g) = \mathbb{E}[g(z)]$ and $\hat{R}_S(g) = \frac{1}{n}\sum_i g(z_i)$. The risk of a predictor is the risk of its loss.

Proposition (uniform convergence via Rademacher complexity) (Bartlett and Mendelson 2002). *Let $S = (z_1, \dots, z_n)$ be an i.i.d. sample and let $\mathcal{G}$ be a class of functions with values in $[0,1]$. Then with probability at least $1-\delta$,*

$$R(g) \le \hat{R}_S(g) + 2\,\mathfrak{R}_n(\mathcal{G}) + \sqrt{\frac{\log(1/\delta)}{2n}} \qquad \text{simultaneously for every } g \in \mathcal{G}. \tag{36.127}$$

The bound has the shape of the finite-class one: an empirical risk, a term for the richness of the class, and a confidence term that would be there for a single function. The richness term is twice the class's ability to fit random signs.

Proof. We show $\mathbb{E}_S[\sup_g (R(g) - \hat{R}_S(g))] \le 2\mathfrak{R}_n(\mathcal{G})$ in three steps; the high-probability form follows afterwards.

I. *Ghost sample.* Let $S' = (z'_1, \dots, z'_n)$ be an independent copy of S, used only inside the proof. Since $R(g) = \mathbb{E}_{S'}[\hat{R}_{S'}(g)]$, and a supremum of expectations is at most the expectation of the supremum,

$$\mathbb{E}_S\Big[\sup_{g\in\mathcal{G}} (R(g) - \hat{R}_S(g))\Big] \le \mathbb{E}_{S,S'}\Big[\sup_{g\in\mathcal{G}} \frac{1}{n}\sum_{i=1}^{n} (g(z'_i) - g(z_i))\Big]. \tag{36.128}$$

The unknown risk has been replaced by a second sample.

II. *Sign swap.* Exchanging z_i with z'_i leaves the joint distribution of (S, S') unchanged, because the two points are independent draws from the same distribution. The exchange flips the sign of the i-th difference $g(z'_i) - g(z_i)$ for every g at once. So for any fixed signs $\varepsilon_1, \dots, \varepsilon_n$ the right-hand side of Equation 36.128 does not change if the i-th difference is multiplied by ε_i. Nor does it change when the signs are drawn at random:

$$\mathbb{E}_{S,S'}\Big[\sup_g \frac{1}{n}\sum_i (g(z'_i) - g(z_i))\Big] = \mathbb{E}_{S,S',\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i \varepsilon_i (g(z'_i) - g(z_i))\Big].$$

III. *Split.* The supremum of a sum is at most the sum of the suprema, so the right-hand side is at most

$$\mathbb{E}_{S',\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i \varepsilon_i\, g(z'_i)\Big] + \mathbb{E}_{S,\varepsilon}\Big[\sup_g \frac{1}{n}\sum_i (-\varepsilon_i)\, g(z_i)\Big] = 2\,\mathfrak{R}_n(\mathcal{G}),$$

because $-\varepsilon$ has the same distribution as ε, so each term is $\mathfrak{R}_n(\mathcal{G})$. The split produces the factor 2.

For the high-probability form, consider $\Phi(S) = \sup_g (R(g) - \hat{R}_S(g))$. Changing one sample point changes each $\hat{R}_S(g)$ by at most $1/n$, and therefore changes Φ by at most $1/n$. *McDiarmid's inequality* (McDiarmid 1989), the extension of Hoeffding's inequality from sums to functions that no single input can move much, then gives $P(\Phi - \mathbb{E}\Phi \ge t) \le e^{-2nt^2}$ (exercise 1). Setting the right-hand side to δ and combining with the bound on $\mathbb{E}\Phi$ gives Equation 36.127. ■

The intuition is that the ghost sample stands in for the population. Swapping a real point with its ghost changes nothing, because the two are exchangeable, and after the swaps only coin flips remain: the deviation of the class from the truth is controlled by how well it can follow random signs.

The theorem uses the expected complexity $\mathfrak{R}_n$, whereas what we can compute is $\widehat{\mathfrak{R}}_S$ on the sample at hand. Changing one sample point moves $\widehat{\mathfrak{R}}_S(\mathcal{G})$ by at most $1/n$ too, so a second use of McDiarmid's inequality gives Equation 36.127 with $\widehat{\mathfrak{R}}_S(\mathcal{G})$ in place of $\mathfrak{R}_n(\mathcal{G})$ and the confidence term $3\sqrt{\log(2/\delta)/(2n)}$. The computations below rely on this form.

The theorem can be checked at its ends. For a class with one member, $\mathfrak{R}_n = 0$ and Equation 36.127 is one-sided Hoeffding. For a finite class, Massart's lemma turns it into Equation 36.125 with larger constants: for 2,000 configurations on 10,000 examples the complexity term alone is $2\sqrt{2\log 2000/n} = 0.078$, against the union bound's whole half-width of 0.024. The Rademacher route is weaker for finite classes, but it also applies to infinite ones. For classes of binary classifiers, counting the labelings the class can realize on n points bounds $\mathfrak{R}_n$ by a constant times $\sqrt{d_{\mathrm{VC}}\log n/n}$ (Boucheron et al. 2005). Here d_{VC} is the VC dimension of the main text, the largest number of points on which the class realizes every labeling.

At the other end, a loss class with values in $[0,1]$ has $\mathfrak{R}_n(\mathcal{G}) \le \tfrac{1}{2}$. When equality holds, the complexity term is 1 and the bound allows any risk whatever. For classifiers with the 0-1 loss, the two complexities stand in a fixed ratio. Let the predictors take values in $\{-1,+1\}$ and use the 0-1 loss, $g(\mathbf{x},y) = (1 - y f(\mathbf{x}))/2$. Then $\sum_i \varepsilon_i g(z_i) = \tfrac{1}{2}\sum_i \varepsilon_i + \tfrac{1}{2}\sum_i (-\varepsilon_i y_i) f(\mathbf{x}_i)$. The first sum has mean zero and does not depend on f, and for fixed labels $-\varepsilon_i y_i$ is again a fair random sign, so

$$\widehat{\mathfrak{R}}_S(\mathcal{G}) = \tfrac{1}{2}\,\widehat{\mathfrak{R}}_S(\mathcal{F}).$$

Such a predictor class, if it fits every sign pattern on the sample, has $\widehat{\mathfrak{R}}_S(\mathcal{F}) = 1$. Twice the loss complexity is then 1, and the theorem promises only an error rate below 100%.

36.10.2.3 The Linear Class

The theorem is useful only if the complexity can be computed for a class we care about. For linear functions with a bounded weight norm the computation takes four lines, and its answer does not involve the dimension.

Proposition (Rademacher complexity of a norm-bounded linear class). *Let $\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x}\rangle : \|\mathbf{w}\|_2 \le B\}$ and suppose the sample satisfies $\|\mathbf{x}_i\|_2 \le r$ for all i. Then*

$$\widehat{\mathfrak{R}}_S(\mathcal{F}) = \frac{B}{n}\,\mathbb{E}_{\varepsilon}\Big\|\sum_{i=1}^{n}\varepsilon_i \mathbf{x}_i\Big\| \le \frac{B\,r}{\sqrt{n}}. \tag{36.129}$$

Proof. The supremum is explicit. By linearity and Cauchy–Schwarz, with equality at $\mathbf{w}$ of length B aligned with the signed sum,

$$\sup_{\|\mathbf{w}\|\le B} \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i\langle \mathbf{w}, \mathbf{x}_i\rangle = \sup_{\|\mathbf{w}\|\le B}\frac{1}{n}\Big\langle \mathbf{w},\, \sum_i \varepsilon_i \mathbf{x}_i\Big\rangle = \frac{B}{n}\Big\|\sum_{i=1}^{n}\varepsilon_i\mathbf{x}_i\Big\|.$$

Taking the expectation gives the equality in the proposition. For the bound, the square root is concave, so $\mathbb{E}\|\cdot\| \le \sqrt{\mathbb{E}\|\cdot\|^2}$, and expanding the square,

$$\mathbb{E}_\varepsilon\Big\|\sum_i \varepsilon_i \mathbf{x}_i\Big\| \le \sqrt{\sum_{i,j} \mathbb{E}[\varepsilon_i\varepsilon_j]\,\langle \mathbf{x}_i, \mathbf{x}_j\rangle} = \sqrt{\sum_i \|\mathbf{x}_i\|^2} \le r\sqrt{n}.$$

The cross terms vanish because independent signs have $\mathbb{E}[\varepsilon_i\varepsilon_j] = 0$ for $i \ne j$. Multiplying by B/n gives $Br/\sqrt{n}$. ■

Panel (a) of Figure 36.40 draws the first step of this proof: the best weight vector points along the signed sum.

Only one step of the proof loses anything. The identity $\mathbb{E}\|\sum_i \varepsilon_i \mathbf{x}_i\|^2 = \sum_i \|\mathbf{x}_i\|^2$ is exact, so the slack is all in the step from the mean length of the signed sum to its root-mean-square length. That step is tight when the length varies little from one sign pattern to the next. For the eight points of the figure the slack is 11% (0.316 against 0.354). In the plane, a signed sum of many unit vectors in spread-out directions is approximately a two-dimensional Gaussian vector, whose length has mean $\sqrt{\pi}/2 \approx 0.89$ times its root-mean-square, so the slack stays near 11% however many points there are. In twenty dimensions the length of the signed sum concentrates, by the thin shell of Section 36.9.2.1, and the slack falls to about 1%.

The dimension d does not appear in Equation 36.129: it matters only through its effect on B and r. In this bound the capacity of a linear model is set by the product of its weight norm and its input radius, whether it acts on ten features or on a million. Weight decay (Section 2.7) keeps $\|\mathbf{w}\|$ small, and so keeps the class that contains the fitted model small. Nor is this confined to linear models: a two-layer ReLU network with output weights of ℓ_1 norm at most B_1 and hidden weight vectors of norm at most B_2 has complexity at most $2B_1B_2r/\sqrt{n}$, whatever its width (exercise 10).

A reader may object that the proposition needs B fixed before the data, while the norm of a trained weight vector is known only afterwards. The objection is right, and a union bound over radii answers it. Apply the bound to each radius $B_j = 2^j$ with failure probability $\delta_j = \delta/(j(j+1))$, which sum to δ, and use the smallest radius that contains the fitted vector. For $\|\mathbf{w}\| > 1$ this loses at most a factor 2 in the complexity term and adds a term of order $\sqrt{\log\log\|\mathbf{w}\|/n}$ (exercise 3).

The theorem needs the complexity of the loss class, and the proposition computes that of the predictor class. A Lipschitz loss connects them.

Lemma (contraction) (Boucheron et al. 2013). *Let $\phi_1, \dots, \phi_n : \mathbb{R} \to \mathbb{R}$ each be L-Lipschitz. Then for any class $\mathcal{F}$ and points $\mathbf{x}_1, \dots, \mathbf{x}_n$,*

$$\mathbb{E}_\varepsilon\Big[\sup_{f\in\mathcal{F}} \sum_{i=1}^n \varepsilon_i\, \phi_i(f(\mathbf{x}_i))\Big] \le L\,\mathbb{E}_\varepsilon\Big[\sup_{f\in\mathcal{F}} \sum_{i=1}^n \varepsilon_i\, f(\mathbf{x}_i)\Big]. \tag{36.130}$$

Composing with a function that stretches distances by at most L multiplies the complexity by at most L. Each ϕ_i may differ, which is what a loss needs: at the point i the loss is the function $u \mapsto \ell(u, y_i)$ of the prediction, with the label y_i fixed. No centering is required, since adding a constant c to ϕ_i adds $c\,\varepsilon_i$, whose expectation is zero. We state the lemma without proof.

The theorem also needs losses in $[0, 1]$. The hinge and logistic losses are not bounded by 1: on the class they reach about $1 + Br$, so they must be divided by that maximum before the theorem applies, and the result is a bound on the rescaled loss. A cleaner route is a loss that is bounded from the start, the *ramp loss* on the margin $u = yf(\mathbf{x})$,

$$\phi_\gamma(u) = \min\{1, \max\{0,\ 1 - u/\gamma\}\},$$

which is 1 for a mistake, 0 for a correct prediction with margin at least γ, and linear in between. It lies above the 0-1 loss and is $(1/\gamma)$-Lipschitz.

Write $\hat{R}_\gamma(\mathbf{w})$ for the average of the ramp loss $\frac{1}{n}\sum_i \phi_\gamma(y_i\langle \mathbf{w}, \mathbf{x}_i\rangle)$ on the sample. With $\|\mathbf{x}\| \le r$ for every input, the contraction lemma and the linear-class proposition give a loss-class complexity of at most $Br/(\gamma\sqrt{n})$, and the theorem gives, with probability at least $1-\delta$, for every $\mathbf{w}$ with $\|\mathbf{w}\| \le B$,

$$P(y\langle \mathbf{w}, \mathbf{x}\rangle \le 0) \le \hat{R}_\gamma(\mathbf{w}) + \frac{2Br}{\gamma\sqrt{n}} + \sqrt{\frac{\log(1/\delta)}{2n}}. \tag{36.131}$$

The left side is the true error rate of the linear classifier; the right side is its training ramp loss plus two terms we can evaluate. As an illustration (the ratio is chosen, not measured), take $Br/\gamma = 10$, $n = 50{,}000$ and $\delta = 0.05$. The complexity term is 0.089 and the confidence term 0.005, together 0.095. With $B = 10\gamma/r$ fixed in advance, a linear classifier in the class that separates its training data with margin γ has, with probability at least 95%, a true error below 9.5%. Fitting random labels with a margin forces Br/γ up. Once it passes about 10.6 the two terms exceed 0.1, and near $Br/\gamma \approx 110$ the complexity term alone reaches 1.

36.10.2.4 Rademacher Complexity in Numbers

The proposition gives the complexity of the linear class as an expected length, and the expectation can be estimated by averaging over random sign vectors. We do this for 50 unit-norm points in $d = 2$, 20 and $1{,}000$ dimensions. The slack analysis above predicts an estimate about 11% below the bound $Br/\sqrt{n} = 1/\sqrt{50}$ in the plane and about 1% below it in twenty dimensions.

```
n, B, num_signs = 50, 1.0, 2000
signs = rng.choice([-1.0, 1.0], size=(num_signs, n))  # rows: sign vectors
for d in (2, 20, 1000):
    X = rng.standard_normal((n, d))
    X /= np.linalg.norm(X, axis=1, keepdims=True)       # ||x_i|| = r = 1
    sup_corr = B * np.linalg.norm(signs @ X, axis=1) / n  # closed-form sup
    print(f'd = {d:4d}: Monte-Carlo R_S = {sup_corr.mean():.3f}, '
          f'bound B r / sqrt(n) = {B / np.sqrt(n):.3f}')
```

```
d =    2: Monte-Carlo R_S = 0.123, bound B r / sqrt(n) = 0.141
d =   20: Monte-Carlo R_S = 0.141, bound B r / sqrt(n) = 0.141
d = 1000: Monte-Carlo R_S = 0.141, bound B r / sqrt(n) = 0.141
```

The estimate is about 0.125 in the plane and 0.139 to 0.141 in 20 and 1,000 dimensions, against the bound 0.141, as predicted. Going from 20 to 1,000 dimensions leaves the complexity unchanged.

The other end of the scale is a class rich enough to *interpolate*, that is, to reproduce any labels exactly on the sample. Least squares on $p = 2n$ Gaussian features, fit with the pseudoinverse of Section 33.5.3.1, returns for every sign vector the interpolant $\mathbf{w} = \Phi^{+}\varepsilon$ of smallest norm, where $\Phi \in \mathbb{R}^{n \times p}$ holds the features of the points as rows. Its correlation with the signs is 1 for every pattern, by construction. What the proposition tells us is how large a ball of weights this takes: a class that interpolates every sign pattern has $\widehat{\mathfrak{R}}_S \geq 1$, the proposition gives $\widehat{\mathfrak{R}}_S \leq Br/\sqrt{n}$, hence any ball that contains all the interpolants has $Br/\sqrt{n} \geq 1$. How close to this minimum do actual fits come? The cell fits 2,000 sign vectors and prints their norms (the predicted mean $\sqrt{n/(p-n-1)}$ is derived in exercise 4) and the value of $Br/\sqrt{n}$ for the smallest ball that contains them.

```
p = 2 * n                                   # more features than points
Phi = rng.standard_normal((n, p))
W = np.linalg.pinv(Phi) @ signs.T           # min-norm fit of each sign vector
w_norm = np.linalg.norm(W, axis=0)
r = np.linalg.norm(Phi, axis=1).max()       # input radius: largest row norm
row_norm = np.linalg.norm(Phi, axis=1).mean()
print(f'mean ||w|| = {w_norm.mean():.2f} (predicted '
      f'{np.sqrt(n / (p - n - 1)):.2f}), largest {w_norm.max():.2f}')
print(f'typical fit: mean ||w|| * mean row norm / sqrt(n) = '
      f'{w_norm.mean() * row_norm / np.sqrt(n):.2f}')
print(f'r = {r:.1f}, B r / sqrt(n) with B = largest ||w||: '
      f'{w_norm.max() * r / np.sqrt(n):.1f}')
```

```
mean ||w|| = 0.99 (predicted 1.01), largest 1.31
typical fit: mean ||w|| * mean row norm / sqrt(n) = 1.40
r = 12.4, B r / sqrt(n) with B = largest ||w||: 2.3
```

The fits have norm about 1 and the rows of Φ length about $\sqrt{p} = 10$, so a typical fit sits at $\|\mathbf{w}\| r/\sqrt{n} \approx 1.4$. The largest of the 2,000 norms is 1.3 to 1.8 and the longest row 11 to 12.5, which puts the ball around all the fits at 2 to 3. Interpolating noise therefore takes a norm only one and a half to three times the minimum that the proposition allows: the bound $Br/\sqrt{n}$ is nearly tight for this class, even at a complexity where it has stopped being useful.

The same arithmetic applies to networks. Take the class of all classifiers that an architecture can represent. If training on labels ε returns a member that agrees with them at nearly every point, the supremum in Equation 36.126 is close to 1 for that ε, and if this happens for typical sign vectors, so is the complexity. In Section 4.5.1.1 an MLP fit random labels of its training images exactly, as the image classifiers of Zhang et al. (2021) did. (These experiments tried a few labelings with ten classes, so they are evidence, not a proof.) Twice the complexity of the loss class is then about 1, and Equation 36.127 allows any error rate. Yet the same network generalizes on real labels. What decides its test error is therefore invisible to a bound over the whole class: the data, the optimizer, and the solution it reached. The next part examines a model in which the solution can be computed in closed form.

36.10.3 Interpolation and Double Descent

Section 4.5.2 swept the width h of a network whose hidden layer stayed frozen at its random initialization. The test error peaked at $h = n$, where an exact fit first became possible and the smallest singular value of the hidden representation was tiny, and fell again beyond it. Here we repeat the construction on synthetic data, where every quantity in the mechanism

can be computed, and add two things to the main text's account: a formula for the fit in terms of singular values, and a split of the test error into the part due to the target and the part due to label noise. The features are $\phi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{V}\mathbf{x})$ with a frozen random matrix $\mathbf{V}$, a *random-features* model (Rahimi and Recht 2007), and only the linear head $\mathbf{w}$ is fit. This part can be skipped on a first reading.

36.10.3.1 The Minimum-Norm Interpolant

To see why the threshold is special we need the solution in closed form. With p features and n points, the feature matrix $\Phi \in \mathbb{R}^{n\times p}$ has the feature vectors of the training points as rows, and we fit $\mathbf{w} = \Phi^{+}\mathbf{y}$: least squares below $p = n$, the interpolant of smallest norm above it.

The singular value decomposition shows what happens below, at and above the threshold. Let Φ have full rank $q = \min(n, p)$ and singular value decomposition $\Phi = \sum_{i=1}^{q} \sigma_i \mathbf{u}_i \mathbf{v}_i^{\top}$, with $\sigma_1 \geq \cdots \geq \sigma_q > 0$. Then

$$\mathbf{w} = \Phi^{+}\mathbf{y} = \sum_{i=1}^{q} \frac{\langle \mathbf{u}_i, \mathbf{y}\rangle}{\sigma_i}\, \mathbf{v}_i. \tag{36.132}$$

Every component of $\mathbf{y}$ along a left singular vector is divided by its singular value, so the smallest singular value $\sigma_{\min} = \sigma_q$ amplifies most. Whatever part of $\mathbf{y}$ lies along $\mathbf{u}_q$ enters the solution with the factor $1/\sigma_{\min}$. This holds for label noise and equally for the part of the target that the features represent badly.

For $p < n$ the fit $\Phi\mathbf{w}$ is the projection of $\mathbf{y}$ onto the p directions $\mathbf{u}_i$, and the rest of $\mathbf{y}$ is left as residual. At $p = n$ there are n directions, the fit is exact, and nothing is left over to absorb the badly represented part. A random square ReLU-feature matrix is almost always invertible, but rarely *well* invertible: its smallest singular value is small, and the condition-number analysis of Section 33.5.3.2 says that the solution then moves far in the direction $\mathbf{v}_q$. Adding features beyond n gives the matrix more room, and its smallest singular value grows again.

Past the threshold the norm of the solution can only fall as features are added, provided the features are nested (the model with $p + 1$ features uses the features of the model with p, plus one more). This is the padding argument of Section 4.5.2: give the new feature zero weight, and the minimum-norm interpolant with p features becomes an interpolant with $p+1$ features of the same norm. The argument needs exact interpolation. Below $p = n$ the least-squares solution is not an interpolant, and its norm grows with p: in the sweep below it rises from about 0.5 at $p = 2$ to nearly 2 at $p = 15$.

The linear-class bound suggests a quantity to track along the sweep: the *capacity product* $\|\mathbf{w}\| \cdot \max_i \|\phi(\mathbf{x}_i)\|/\sqrt{n}$, which is $Br/\sqrt{n}$ for the smallest ball that contains the fitted head. It is borrowed from classification. The sweep is a regression with squared loss on unbounded targets, which no theorem of this section covers, so the product is a diagnostic and not a bound. The feature radius grows like $\sqrt{p}$ while the norm of the solution falls past the threshold, so the product could go either way.

36.10.3.2 A Double-Descent Sweep

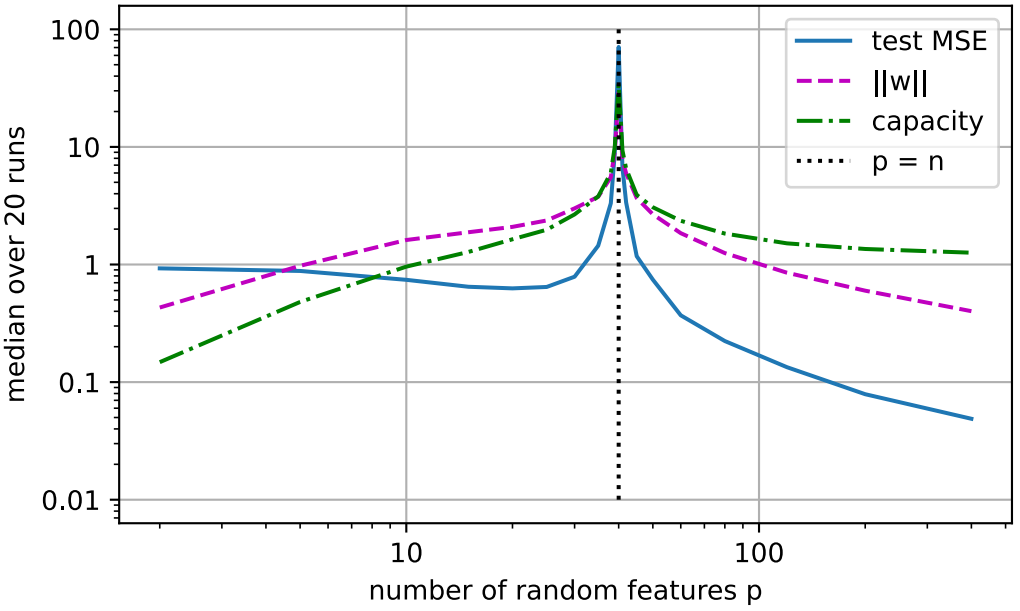
The data are forty training points from a linear teacher $y = \langle \beta, \mathbf{x}\rangle$ in fifteen dimensions, with $\|\beta\| = 1$ and label noise of standard deviation 0.1. The features are ReLU random

features with nested columns, and the head is fit by `pinv`: least squares below the threshold, the minimum-norm interpolant above it. We sweep p from 2 to 400 through $p = n = 40$, with the neighbours $p = 39$ and 41 included so that the peak is resolved. For every fit we record the test error, the training error, $\|\mathbf{w}\|$, $\sigma_{\min}$ and the capacity product, and also the two parts of the test error that Section 36.10.3.3 separates: the clean part, the error of the same fit made to noise-free labels, and the noise part, the extra error that the label noise adds. Medians over 20 independent runs keep the curve readable.

```
n, n_test, d_in, p_max, trials, noise = 40, 500, 15, 400, 20, 0.1
p_grid = np.array([2, 5, 10, 15, 20, 25, 30, 35, 38, 39, 40, 41, 42, 45, 50,
                   60, 80, 120, 200, 400])
keys = ('test', 'train', 'norm', 's_min', 'capacity', 'clean', 'noise',
        'ratio_q', 'ratio')
stats = np.zeros((len(keys), trials, len(p_grid)))
ols = np.zeros(trials)                              # least squares on x itself
for tr in range(trials):
    beta = rng.standard_normal(d_in)
    beta /= np.linalg.norm(beta)                    # teacher: y = <beta, x>
    X, X_te = (rng.standard_normal((size, d_in)) for size in (n, n_test))
    y_clean, y_te = X @ beta, X_te @ beta
    y = y_clean + noise * rng.standard_normal(n)    # noisy training labels
    ols[tr] = np.mean((X_te @ np.linalg.lstsq(X, y, rcond=None)[0] - y_te) ** 2)
    V = rng.standard_normal((p_max, d_in)) / np.sqrt(d_in)
    Phi, Phi_te = np.maximum(X @ V.T, 0), np.maximum(X_te @ V.T, 0)
    for j, p in enumerate(p_grid):                  # nested feature prefixes
        F, F_te = Phi[:, :p], Phi_te[:, :p]
        P = np.linalg.pinv(F)                       # least squares / min norm
        w = P @ y
        r = np.linalg.norm(F, axis=1).max()         # feature radius
        _, S, Vt = np.linalg.svd(F, full_matrices=False)
        ratio = ((np.linalg.norm(F_te @ Vt.T, axis=0) ** 2 / n_test)
                 / (S ** 2 / n))                    # test/train feature power
        stats[:, tr, j] = (
            np.mean((F_te @ w - y_te) ** 2),                # test MSE
            np.mean((F @ w - y) ** 2),                      # training MSE
            np.linalg.norm(w),                              # ||w||
            S.min(),                                        # sigma_min
            np.linalg.norm(w) * r / np.sqrt(n),             # capacity
            np.mean((F_te @ (P @ y_clean) - y_te) ** 2),    # clean part
            noise ** 2 * np.linalg.norm(F_te @ P) ** 2 / n_test,  # noise part
            ratio[-1], ratio.mean())                # along v_q, average
med = dict(zip(keys, np.median(stats, axis=1)))
```

The plot shows the medians of the test error, the solution norm and the capacity product on logarithmic axes.

```
d2l.plot([p_grid] * 3 + [[n, n]],
         [med['test'], med['norm'], med['capacity'], [1e-2, 1e2]],
         'number of random features p', 'median over 20 runs',
         legend=['test MSE', '||w||', 'capacity', 'p = n'], xscale='log',
         yscale='log', fmts=('-', 'm--', 'g-.', 'k:'), figsize=(6, 3.5))
```



The table gives the numbers at seven feature counts. Its last two columns belong to Section 36.10.3.3, and its last line is the classical model that fits the raw inputs.

```
print('   p ' + ' '.join(f'{key:>8}' for key in keys))
for p in (15, 38, 40, 42, 50, 80, 400):
    row = [med[key][p_grid == p][0] for key in keys]
    print(f'{p:4d} ' + ' '.join(f'{v:8.2g}' for v in row))
print(f'least squares on the {d_in} raw inputs: test MSE {np.median(ols):.4f}')
```

```
   p     test     train      norm     s_min capacity     clean     noise  ratio_q     ratio
  15     0.65      0.28       1.9         1      1.3      0.67    0.0066      3.7       1.8
  38      3.3    0.0081       5.5      0.08        6       2.9      0.31  6.8e+02        32
  40       71   2.3e-28        29     0.016       34        58       4.2  1.5e+04   4.2e+02
  42      3.4   1.3e-29         6     0.076      6.5       2.9      0.26  5.8e+02        26
  50     0.74   6.6e-30       2.7      0.25      3.1      0.64     0.063       62       6.3
  80     0.22     7e-30       1.3      0.71      1.8      0.19     0.021       10       2.1
 400    0.049   5.3e-30       0.4       2.6      1.3      0.04    0.0095      2.3      0.95
least squares on the 15 raw inputs: test MSE 0.0066
```

The table is best read in three stretches: below the threshold, at it, and beyond it.

Below the threshold the model is a classical one, with fewer parameters than points. Its test error first falls, from about 1 at $p = 2$ to roughly 0.6 to 0.8 near $p = 15$, and then rises again as p approaches 40. This is the U-curve that the classical account leads us to expect. Two things must be said about this regime before anything is compared with it. First, it is weak. The targets have variance 1, so a test error of 1 is what predicting zero achieves, and the best fit here removes only a quarter to a third of it. Second, a far better classical model exists. The teacher is linear in the fifteen raw inputs, and least squares on those inputs has an expected test error of $0.01 \cdot 15/(40 - 15 - 1) \approx 0.006$, the noise variance times $d/(n - d - 1)$ for $d = 15$. The last line of the table confirms it. The random features were handicapped by construction: they have to assemble a linear function out of ReLU ridges,

and forty points are too few for that.

At the threshold three columns move together. The training error drops from a few times 10^{-3} at $p = 38$ to about 10^{-28}, which is exact interpolation up to floating point. The smallest singular value falls to about 0.01 to 0.04, several times below its values at $p = 38$ and $p = 42$ on either side. And since Equation 36.132 divides by it, the solution is inflated: the norm jumps to between about 6 and 20, and the test error to somewhere between 5 and 50, one to two orders of magnitude above the minimum of the first stretch. These ranges are wide because the peak is heavy-tailed, as it is for a square Gaussian matrix, where $\mathbb{E}[1/\sigma_{\min}^2]$ is infinite. Even a median over 20 trials moves from one run of the cell to the next.

Beyond the threshold everything that rose comes down again. The smallest singular value recovers, to about 2.8 at $p = 400$. The norm falls steadily, as nesting requires (in every trial, not only in the medians), down to about 0.4. The test error follows: about 0.2 at $p = 80$, and 0.04 to 0.07 at $p = 400$. One column has nothing more to tell us. The training error is zero up to floating point everywhere past the threshold, so no criterion based on it can separate $p = 42$ from $p = 400$, whose test errors differ by well over an order of magnitude. The norm, the smallest singular value and the capacity product all can.

What, then, has the sweep shown? Less than its plot suggests. The widest model beats every narrower model of its own family, the best one below the threshold included, by roughly a factor of ten. It still loses to least squares on the raw inputs by about another factor of ten. Within one nested family of random features, more features are better, even far past the point where the model fits its noise. That is a real and surprising fact. Whether interpolation can beat a well-specified small model is a different question, and in this experiment the answer is no.

The capacity product peaks at the threshold with values of about 8 to 25, falls past it, and reaches about 1.2 to 1.3 at $p = 400$. It follows the shape of the test curve, but it is no guarantee: no theorem here covers this loss, the product changes when the targets are rescaled, and its B was read off the fit rather than fixed in advance.

The peak is not caused by the label noise. Rerunning the sweep with `noise = 0` leaves a peak of the same size, and the table shows why. At $p = 40$ the clean part of the test error is larger than the noise part, usually several times larger. The linear teacher is not a combination of finitely many ReLU features, and the part of the target that the features represent badly is what $\sigma_{\min}$ divides. With a target that is an exact combination of the first ten features and no noise, there is no peak at all (exercise 8).

36.10.3.3 Benign Overfitting

One puzzle remains. The $p = 400$ model reproduces every noisy training label exactly, yet it is the best of the sweep. Why does fitting the noise cost so little there?

The test error separates into two parts we can compute. Write the training labels as $\mathbf{y} = \mathbf{y}_0 + \xi$, with $\mathbf{y}_0$ the noise-free labels and ξ independent noise of variance $\sigma_\xi^2 = 0.01$ per label. Then $\mathbf{w} = \Phi^+\mathbf{y}_0 + \Phi^+\xi$, and the predictions at the test points are $\Phi_{\text{te}}\Phi^+\mathbf{y}_0 + \Phi_{\text{te}}\Phi^+\xi$. In the squared test error, the cross term between these two parts has mean zero because ξ does. Averaged over the noise, the test error over n_{te} test points is

$$\mathbb{E}_\xi[\text{test error}] = \underbrace{\frac{1}{n_{\text{te}}}\left\|\Phi_{\text{te}}\Phi^+\mathbf{y}_0 - \mathbf{y}_{\text{te}}\right\|^2}_{\text{clean part}} + \underbrace{\frac{\sigma_\xi^2}{n_{\text{te}}}\left\|\Phi_{\text{te}}\Phi^+\right\|_F^2}_{\text{noise part}}.$$

The last two columns of the table print the two parts. At $p = 40$ the noise part is about 1 to 4, a hundred times or more the noise variance 0.01, and the clean part is larger still. At $p = 42$ the noise part is a few tenths, at $p = 80$ about 0.02, and at $p = 400$ just under 0.01. The clean part falls from roughly 4 to 20 at the peak to about 0.05 at $p = 400$, so most of the gain past the threshold comes from a better approximation of the target.

Why does the division by small singular values, which produced the peak, do so little harm at $p = 400$? By Equation 36.132 a noise component along $\mathbf{u}_i$ enters the weights with the factor $1/\sigma_i$ along $\mathbf{v}_i$, and it moves a test prediction by that weight times the component of the test feature vector along $\mathbf{v}_i$. Since $\|\Phi_{\text{te}}\Phi^+\|_F^2 = \sum_i \|\Phi_{\text{te}}\mathbf{v}_i\|^2/\sigma_i^2$ and $\sigma_i^2 = \|\Phi\mathbf{v}_i\|^2$, the noise part is, for $p \geq n$,

$$\sigma_\xi^2 \cdot \frac{1}{n}\sum_{i=1}^{n} \frac{\|\Phi_{\text{te}}\mathbf{v}_i\|^2/n_{\text{te}}}{\|\Phi\mathbf{v}_i\|^2/n},$$

the noise variance times the average over directions of the ratio of the mean squared test feature to the mean squared training feature along $\mathbf{v}_i$. A small σ_i hurts only if the test features are not small in the same direction. At $p = n$ the smallest singular value is small by an accident of the forty training points: along $\mathbf{v}_q$ the test features are typically thousands of times larger in mean square than the training features. At $p = 400$ a small σ_i marks a direction in which the features vary little for training and test points alike. The table's last two columns show it: the ratio along $\mathbf{v}_q$ (`ratio_q`) is about 2, and the average over all directions (`ratio`) is just below 1, so the small singular value in the denominator is cancelled by a small test feature in the numerator. Overfitting is called *benign* in this sense. Benign means bounded here, not vanishing: at $p = 400$ the predictions still carry about as much noise as a single training label.

Bartlett et al. (2020) make this precise for linear regression, through two effective ranks of the covariance of the inputs, and exercise 9 measures them. In deep networks the features are learned and change the spectrum during training, and a theory that predicts a given network's test error from first principles is still open. Every bound in this section covers the worst function in a class fixed before the data, while the function we use is the one an algorithm reached. A guarantee that depends on the algorithm and the data, rather than on the class alone, is the open question this section leaves.

36.10.4 Summary

A guarantee for one function fixed before the data does not cover the function chosen because it looked best on the data. Uniform convergence repairs this by bounding the deviation of every function in the class at once. For a finite class, Hoeffding's inequality and the union bound do so with $\log|\mathcal{F}|/(2t^2)$ additional examples at tolerance t. For infinite classes, symmetrization with a ghost sample turns the unknown risk into a correlation with random signs, and the bound adds twice the Rademacher complexity of the loss class. For the linear class with weight norm B and inputs of norm r this complexity is at most $Br/\sqrt{n}$, whatever the dimension. The argument stops at classes that fit every sign pattern, such as

a network that fits random labels: for them the bound allows any error rate. In the random-features sweep the test error peaks where $\sigma_{\min}$ is smallest and falls as the minimum-norm solution shrinks, and these quantities belong to the solution rather than to the class.

36.10.5 Exercises

1. **McDiarmid's inequality.** ▣ The inequality of McDiarmid (1989) extends Hoeffding's from sums to functions with *bounded differences*. Suppose that changing only the i-th argument of $g(x_1, \dots, x_n)$ moves its value by at most c_i, and that $X_1, \dots, X_n$ are independent. Then

$$P\big(g(X_1,\dots,X_n) - \mathbb{E}[g(X_1,\dots,X_n)] \ge t\big) \le \exp\left(-\frac{2t^2}{\sum_{i=1}^{n} c_i^2}\right), \tag{36.133}$$

   and the same bound holds for the lower tail. Take this statement on faith; its proof uses martingales.

   1. Recover Hoeffding's inequality for an average of independent terms $X_i \in [a_i, b_i]$ from it.
   2. Let $\Phi(S) = \sup_{g \in \mathcal{G}} (R(g) - \hat{R}_S(g))$ for a loss class $\mathcal{G}$ with values in $[0,1]$. Show that Φ has bounded differences $c_i = 1/n$. Combine McDiarmid's inequality with the symmetrization bound $\mathbb{E}_S[\Phi(S)] \le 2\mathfrak{R}_n(\mathcal{G})$ of the text and derive Equation 36.127.
   3. For data in $[0,1]$, the sample mean has bounded differences $1/n$. Construct a sample of odd size n in $[0,1]$ in which changing one point moves the sample median by 1. What does this imply for using McDiarmid's inequality to put an error bar on the bootstrap estimate of the median in Section 36.5?

2. **Massart's lemma and ℓ_1 balls.** ▣ Massart's lemma for finite classes and the Rademacher complexity of ℓ_1-bounded linear classes rest on one bound for the expected maximum of sub-Gaussian variables.

   1. Let $Z_1, \dots, Z_N$ be centered and sub-Gaussian with variance proxy σ^2, that is, $\mathbb{E}[e^{\lambda Z_j}] \le e^{\lambda^2 \sigma^2 / 2}$ for all λ, but not necessarily independent. Show that $\mathbb{E}[\max_j Z_j] \le \sigma \sqrt{2 \log N}$ by bounding $e^{\lambda \mathbb{E}[\max_j Z_j]}$ by $\sum_j \mathbb{E}[e^{\lambda Z_j}]$ and optimizing over $\lambda > 0$. Deduce Massart's lemma, $\widehat{\mathfrak{R}}_S(\mathcal{F}) \le \sqrt{2 \log |\mathcal{F}| / n}$, for a finite class with values in $[-1,1]$.
   2. Let $\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_1 \le B\}$ and $\|\mathbf{x}_i\|_\infty \le r$ for all i. Follow the proof of Equation 36.129, with the duality $\sup_{\|\mathbf{w}\|_1 \le B} \langle \mathbf{w}, \mathbf{v} \rangle = B \|\mathbf{v}\|_\infty$ in place of Cauchy–Schwarz, and show that $\widehat{\mathfrak{R}}_S(\mathcal{F}) \le B r \sqrt{2 \log(2d)/n}$. Why does the dimension d now enter, and why only logarithmically?
   3. Let the inputs lie in $\{-1,+1\}^d$ and let the target weight vector have k nonzero entries, each equal to ± 1. Compare the bound of (b) with Equation 36.129, each with the smallest radius B that contains the target. For which k is the ℓ_1 bound smaller when $d = 10^4$?

3. **Margin bound.** ▣ Let $\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_2 \le B\}$ with $\|\mathbf{x}\|_2 \le r$, let the labels be $y \in \{-1,+1\}$, and let the ramp loss $\phi_\gamma(u) = \min\{1, \max\{0, 1 - u/\gamma\}\}$ with a fixed $\gamma > 0$ act on the margin $u = y f(\mathbf{x})$.

1. Show that $\phi_\gamma(yf(\mathbf{x}))$ is at least the 0-1 loss $\mathbf{1}\{yf(\mathbf{x}) \le 0\}$ and that ϕ_γ is $(1/\gamma)$-Lipschitz. Show that multiplying each prediction $f(\mathbf{x}_i)$ by a fixed label y_i leaves $\widehat{\mathfrak{R}}_S(\mathcal{F})$ unchanged.
2. Combine (a), the contraction lemma Equation 36.130, Equation 36.127, and Equation 36.129 into Equation 36.131. For $n = 50{,}000$ and $\delta = 0.05$, find the largest ratio Br/γ for which the complexity and confidence terms together stay below 0.1.
3. The bound holds for a radius B fixed before the data are drawn, whereas the norm of a trained weight vector is known only afterwards. Apply the bound with radius $B_j = 2^j$ and confidence $\delta_j = \delta/(j(j+1))$ for every $j \ge 1$, and take a union bound over j. State the bound that holds for a weight vector with $\|\mathbf{w}\| > 1$ chosen after seeing the data. Show that the freedom to choose the radius costs at most a factor 2 in the complexity term and an additional term of order $\sqrt{\log\log \|\mathbf{w}\|/n}$.

4. code **Fitting random signs.** ▯ The second cell of Section 36.10.2.4 fits $2{,}000$ sign vectors $\varepsilon \in \{-1,+1\}^n$ with $\mathbf{w} = \Phi^+\varepsilon$ on $p = 2n$ standard Gaussian features and measures the norms of the fits.
   1. Let $p < n$ and let Φ have full column rank. Show that the correlation $\frac{1}{n}\sum_i \varepsilon_i(\Phi\mathbf{w})_i$ has expectation p/n over the signs. The matrix $\Phi\Phi^+$ is a projection.
   2. Let $p > n$, let r be the largest row norm of Φ, and let B be large enough that the class $\{\|\mathbf{w}\| \le B\}$ contains an interpolant of every sign vector. Explain why $Br/\sqrt{n}$ cannot then be smaller than 1. For a standard Gaussian Φ with $p > n+1$, $\mathbb{E}[(\Phi\Phi^\top)^{-1}] = \mathbf{I}_n/(p-n-1)$. Use this identity to predict the typical $\|\mathbf{w}\|$, and the typical value of $\|\mathbf{w}\|$ times a typical row norm of Φ divided by $\sqrt{n}$, as functions of p and n.
   3. Rerun the cell with $n = 50$ for $p \in \{10, 25, 45, 55, 100, 400, 1000\}$ (for $p < n$ print the correlation instead of the norm) and compare with (a) and (b). What happens to the capacity value of (b) as $p \to \infty$, and why does the version with the largest row norm stay above 1?

5. code **Leaderboard attack.** ▯▯ A leaderboard reports the 0-1 loss of each submitted label vector on a holdout set of $n = 4{,}000$ examples with labels $\mathbf{y} \in \{0,1\}^n$. The *boosting attack* of Blum and Hardt (2015) uses no knowledge of the task: submit k uniformly random label vectors, keep those with reported loss at most $\frac{1}{2}$, and submit their coordinatewise majority vote. On fresh data, the majority vote has loss $\frac{1}{2}$.
   1. Implement the attack for uniformly random $\mathbf{y}$ and $k \in \{10, 30, 100, 300, 1000, 3000\}$, and record the reported loss of the majority vote. Compare its distance from $\frac{1}{2}$ with the deviation that Equation 36.125 allows for a pre-registered pool of $k+1$ models at $\delta = 0.05$. From which k on does the attack exceed this deviation, and which hypothesis of the proposition does it violate?
   2. Blum and Hardt (2015) prove that the reported loss is $\frac{1}{2} - \Omega(\sqrt{k/n})$ but do not give the constant. Derive it heuristically with Gaussian approximations: estimate the expected excess agreement with $\mathbf{y}$ of a kept vector, then the probability that the majority of about $k/2$ such vectors is correct on one coordinate. Compare the constant with the measurements of (a).
   3. The *Ladder* mechanism of Blum and Hardt (2015) releases a new score only when

a submission beats the best released score by more than a step η, rounds it to a multiple of η, and otherwise repeats the previous score. Against the Ladder, the attacker can only keep the vectors that lowered the released score. Implement both for $\eta \in \{0.005, 0.01\}$ and $k \in \{100, 1000, 3000\}$. How far below $\frac{1}{2}$ does the majority vote now get, and how does this compare with the deviation of (a)?

6. **Ranking configurations on a holdout.** ▣ A validation set has $n = 10{,}000$ examples, the loss takes values in $[0, 1]$, and $\delta = 0.05$.

   1. From Equation 36.125, how many configurations, fixed in advance, can be ranked with a simultaneous guarantee of $\pm 2\%$, $\pm 2.5\%$ and $\pm 3\%$? Why is the count so sensitive to the tolerance?
   2. Repeat (a) with Chebyshev's inequality (variance at most $\frac{1}{4}$) in place of Hoeffding's, with the same union bound. How many configurations does it allow at $\pm 2\%$, and how many at $\pm 5\%$?
   3. *The winner's curse.* Let all k configurations have true error 10%, and treat their validation errors as independent. With the Gaussian approximation to each error and the estimate $\mathbb{E}[\max] \approx \sqrt{2 \log k}$ for the maximum of k independent standard normals, show that the best reported error is about $10\% - 0.3\% \cdot \sqrt{2 \log k}$, and evaluate it at $k = 1{,}000$. Then compute the exact median and the 5th and 95th percentiles of the minimum of $k = 1{,}000$ independent Binomial$(10{,}000, 0.1)$ counts, divided by n, from the distribution function of the binomial.
   4. The reported score of the chosen configuration is biased downward by about one point. Explain in one sentence why scoring it on a fresh test set removes the bias. Does the bias violate a hypothesis of Equation 36.125, and does the bound cover it?

7. **Thresholds on the line.** ▣ Let $\mathcal{F} = \{x \mapsto \mathbf{1}\{x \geq \theta\} : \theta \in \mathbb{R}\}$ act on n distinct real points $x_1 < \cdots < x_n$.

   1. Show that $\mathcal{F}$ realizes exactly $n + 1$ labelings of the points and that its VC dimension (the largest number of points on which it realizes all labelings) is 1. Show that with both orientations, $\mathbf{1}\{x \geq \theta\}$ and $\mathbf{1}\{x \leq \theta\}$, the count is $2n$ and the VC dimension is 2.
   2. Compute $\widehat{\mathfrak{R}}_S(\mathcal{F})$ for the one-sided class exactly for $n = 2$ and $n = 3$ by enumerating the sign patterns.
   3. Show that in general $\widehat{\mathfrak{R}}_S(\mathcal{F}) = \frac{1}{n}\mathbb{E}\big[\max(0, \max_k T_k)\big]$, where T_k is the sum of the signs at the k largest points, a simple random walk. Argue with the reflection principle, or simulate, that this is about $\sqrt{2/(\pi n)}$ for large n, and check the values at $n = 100$ and $n = 1{,}000$.
   4. Compare with Massart's lemma applied to the $n + 1$ labelings. By what factor is it loose at $n = 100$, and which property of the labelings does it ignore?

8. code **The square case.** ▣ Use the sweep of the double-descent experiment in Section 36.10.3.2.

   1. Show from the stored norms that the nesting monotonicity fails below the threshold, and explain why the padding argument does not apply there.
   2. At $p = n$, decompose $\mathbf{w}$ along Equation 36.132. Report, as a median over trials, the coefficient $|\langle \mathbf{u}_n, \mathbf{y} \rangle| / \sigma_{\min}$ of the smallest singular direction next to $\|\mathbf{w}\|$. Which share of the norm does this one direction carry?

3. Rerun the sweep with `noise = 0`. Then replace the teacher by a target $y = \langle \mathbf{c}, \phi_{1:10}(\mathbf{x}) \rangle$ that is an exact combination of the first ten features, with $\|\mathbf{c}\| = 1$ and no noise. Describe the two curves. In each case, what does $\sigma_{\min}$ divide? Why does the minimum-norm interpolant become *worse* as features are added when the target uses only ten of them?
4. Add label noise of standard deviation 0.1 to the target of (c). Where is the test error now highest, how does the peak compare with the noise variance, and where is the error lowest?

9. [extended] **Benign spectra.** ▣▣ Bartlett et al. (2020) characterize when the minimum-norm interpolant in linear regression has small excess risk, the mechanism that Section 36.10.3.3 describes in words. Let $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ with $\Sigma = \operatorname{diag}(\lambda_1, \dots, \lambda_p)$ and $\lambda_1 \geq \lambda_2 \geq \cdots$, let $y = \langle \theta^*, \mathbf{x} \rangle + \xi$ with $\xi \sim \mathcal{N}(0, \sigma^2)$, stack $n < p$ training inputs into $\mathbf{X} \in \mathbb{R}^{n \times p}$, and let $\hat{\theta} = \mathbf{X}^{+}\mathbf{y}$. The excess risk is $(\hat{\theta} - \theta^*)^{\top} \Sigma (\hat{\theta} - \theta^*)$.

   1. Show that its expectation over the noise is the sum of a bias term $\theta^{*\top}(\mathbf{I} - \mathbf{P})\Sigma(\mathbf{I} - \mathbf{P})\theta^*$, where $\mathbf{P} = \mathbf{X}^{+}\mathbf{X}$, and a variance term $\sigma^2 \operatorname{tr}\big((\mathbf{X}^{+})^{\top} \Sigma \mathbf{X}^{+}\big)$. Implement both terms; no test set is needed.
   2. The paper's effective ranks are $\rho_k = \sum_{i>k} \lambda_i / \lambda_{k+1}$ and $\varrho_k = \big(\sum_{i>k} \lambda_i\big)^2 / \sum_{i>k} \lambda_i^2$. With k^* the smallest k such that $\rho_k \geq bn$, the variance term is of order $\sigma^2 (k^*/n + n/\varrho_{k^*})$ up to constants; take $b = 1$. Use $n = 100$, $\sigma = 1$, five strong directions with $\lambda_i = 1$ that contain a unit-norm θ^*, and q weak directions with $\lambda_i = 1/q$ each, for $q \in \{100, 300, 10^3, 3 \cdot 10^3, 10^4\}$. Compare the measured variance with $\sigma^2 (k^*/n + n/\varrho_{k^*})$. What happens at $q = 100$, and why?
   3. Repeat with power-law spectra $\lambda_i = i^{-\alpha}$ truncated at $p = 40n$, for $\alpha \in \{0.5, 1, 2\}$, $\theta^* = \mathbf{e}_1$, and $n \in \{50, 100, 200, 400\}$. For which spectra does the excess risk decrease with n? For the others, does the bias or the variance fail, and how do ρ_0/n, k^*/n, and n/ϱ_{k^*} show it?

10. **Norms, not widths: a two-layer network.** ▣▣ Let $\mathcal{F}_1 = \{\mathbf{x} \mapsto \operatorname{ReLU}(\langle \mathbf{v}, \mathbf{x} \rangle) : \|\mathbf{v}\|_2 \leq B_2\}$ and $\mathcal{F}_2 = \{\mathbf{x} \mapsto \sum_{j=1}^{h} w_j \operatorname{ReLU}(\langle \mathbf{v}_j, \mathbf{x} \rangle) : \|\mathbf{w}\|_1 \leq B_1,\ \|\mathbf{v}_j\|_2 \leq B_2\}$, on points with $\|\mathbf{x}_i\| \leq r$.

   1. Show that $\widehat{\mathfrak{R}}_S(\mathcal{F}_1) \leq B_2 r / \sqrt{n}$.
   2. Show that $\mathcal{F}_2$ is contained in B_1 times the convex hull of $\mathcal{F}_1 \cup (-\mathcal{F}_1)$, and that taking the convex hull of a class does not change its empirical Rademacher complexity. Hint: $\sum_j w_j f_j = \|\mathbf{w}\|_1 \sum_j \frac{|w_j|}{\|\mathbf{w}\|_1} (\pm f_j)$.
   3. Using $0 \in \mathcal{F}_1$, show that $\mathbb{E}_{\varepsilon} \sup_{f \in \mathcal{F}_1} |\sum_i \varepsilon_i f(\mathbf{x}_i)| \leq 2n\, \widehat{\mathfrak{R}}_S(\mathcal{F}_1)$, and conclude that $\widehat{\mathfrak{R}}_S(\mathcal{F}_2) \leq 2 B_1 B_2 r / \sqrt{n}$.
   4. The width h does not appear in the bound. What does the bound say about weight decay on both layers? Why does it not explain the random-label experiment of Section 4.5.1.1?

36.11 Naive Bayes

In Section 3.4 we trained softmax regression on Fashion-MNIST. It found 7,850 numbers, one weight for each pixel and class plus ten offsets, by gradient descent on the probability of the label given the image. It reached about 83% accuracy, and its probabilities passed a reliability check. Gradient descent is not the only way to obtain such numbers. We could instead describe how each class produces its images, what a typical coat or sandal looks like, and let Bayes' rule turn that description into a classifier. Can this be done, and what do we give up?

The obstacle is size. A description of how coats look must give a probability to every possible picture. Even if we record each pixel only as inked or blank, there are far more such pictures than atoms in the universe, and we have 60,000 training images. The *naive Bayes* classifier removes the obstacle with one assumption: once the class is known, the features tell us nothing about each other.

The idea goes back to Maron's work on sorting documents by the words they contain (Maron 1961), and the spam filters of the 1990s were built on it (Sahami et al. 1998). Under the assumption, a filter needs only how often each word appears in spam and in ordinary mail, and a garment classifier only how often each pixel is inked in each class. Training is counting, and prediction is multiplying.

Naive Bayes is the simplest *generative* classifier, one built from a model of how each class produces its inputs. It is fitted in one pass without an optimizer, and it can ignore a missing feature by dropping its factor. Its assumption is false for images, as it is for most data, and no other model shows as clearly what a false independence assumption does to a classifier.

This section is the chapter's case study. After the bounds of Section 36.10 the mathematics is light; the work lies in using the chapter's tools on one model. Maximum likelihood fits it, the Beta posterior smooths it, the bootstrap puts an error bar on its accuracy, and a reliability check tests its probabilities.

The code is plain NumPy; the framework supplies only the data loader.

```
d2l.use_svg_display()
```

36.11.1 Bayes' Rule for Classification

There are two ways to arrive at the probability of a label given an input. The main text learned it directly. Here we go the long way round, through a model of the inputs, find that the long way is impassable as it stands, and clear it with the assumption that gives the classifier its name.

36.11.1.1 The Generative Route

A classifier assigns a label $y \in \{1, \dots, K\}$ to an input $\mathbf{x}$. If we knew the posterior $p(y \mid \mathbf{x})$, the probability of each label given the input, the guess that is right most often would be the most probable label,

$$\hat{y} = \operatorname*{argmax}_{y} \, p(y \mid \mathbf{x}). \tag{36.134}$$

For Fashion-MNIST $K = 10$. If each of the 784 pixels is recorded as inked or blank, $\mathbf{x}$ is one of 2^{784} patterns, and since $784 \log_{10} 2 = 236.0$ that is about 10^{236}. A table of $p(y \mid \mathbf{x})$ would need a row for every pattern, and 60,000 training images fill almost none of them.

Bayes' rule (Section 36.2) writes the posterior as $p(y \mid \mathbf{x}) = p(\mathbf{x} \mid y)\, p(y)/p(\mathbf{x})$. The denominator is the same for every class, so it cannot change which class is largest:

$$\widehat{y} = \operatorname*{argmax}_{y}\ p(\mathbf{x} \mid y)\, p(y). \tag{36.135}$$

The classifier compares, for each class, how probable the image is under that class times how common the class is. We call the product $p(\mathbf{x} \mid y)\, p(y)$ the *Bayes numerator*. Dividing the K numerators by their sum restores the denominator and gives back the posterior.

A *generative classifier* models how the inputs of each class arise, the class-conditional distribution $p(\mathbf{x} \mid y)$, together with the prior $p(y)$, and then applies Bayes' rule. A *discriminative classifier* models $p(y \mid \mathbf{x})$ directly and never says how the inputs arise; softmax regression (Section 3.1) is one.

A reader may object that Bayes' rule has saved nothing: $p(\mathbf{x} \mid y)$ is still a distribution over 2^{784} patterns. The objection is correct, and the chain rule of probability shows where the size comes from. Written out one feature at a time,

$$p(\mathbf{x} \mid y) = p(x_1 \mid y)\, p(x_2 \mid x_1, y) \cdots p(x_d \mid x_1, \ldots, x_{d-1}, y). \tag{36.136}$$

Each factor conditions on the whole history of earlier features, and the last one alone is a table over 2^{d-1} histories. The next subsection removes the histories.

36.11.1.2 The Naive Assumption

We need a model of $p(\mathbf{x} \mid y)$ with few enough numbers to estimate. The naive assumption says, in words: once we know the garment, learning that one pixel is inked tells us nothing more about any other pixel. In symbols, the features are *conditionally independent given the label*, so that every factor of the chain rule forgets its history:

$$p(\mathbf{x} \mid y) = \prod_{i=1}^{d} p(x_i \mid y). \tag{36.137}$$

For binary pixels each factor is fixed by one number, the probability that the pixel is inked in that class. An unrestricted $p(\mathbf{x} \mid y)$ has $2^{784} - 1$ free probabilities per class; the factorized one has 784. Each of these 784 numbers is a frequency, which 6,000 training images per class estimate with a standard error of at most $0.5/\sqrt{6000} \approx 0.0065$ (Section 36.5). Both models need $K - 1 = 9$ further numbers for the prior.

The assumption is about all d features at once, and checking pairs does not establish it. Within one class, let x_1 and x_2 be independent fair bits and let $x_3 = x_1 \oplus x_2$ be 1 when exactly one of them is 1. Any two of the three features are independent, yet the third is determined by the other two. The pattern $(1, 1, 1)$ therefore has probability 0, while the product of the three marginals assigns it $\frac{1}{8}$ (exercise 1).

Drawn as a graph, with an arrow from each variable to the variables it directly influences, the factorization is a star: the label is a parent of every feature, and no edges join the features (Figure 36.41, left). A model that keeps some dependence between features adds edges among them (right).

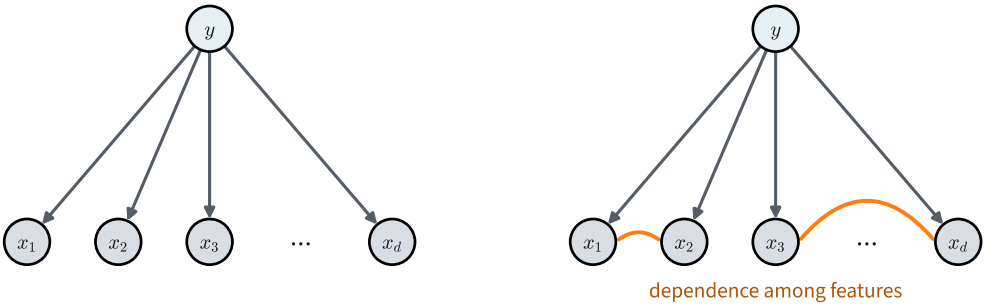


Figure 36.41: Conditional-independence structures. Left: in naive Bayes the label y is a parent of each feature and no edges join the features, which represents $p(\mathbf{x} \mid y) = \prod_i p(x_i \mid y)$. Right: a model that keeps conditional dependence between features adds edges among them.

Substituting the factorization Equation 36.137 into Bayes' rule gives the *naive Bayes rule* (Maron 1961):

$$\hat{y} = \operatorname*{argmax}_{y} \; p(y) \prod_{i=1}^{d} p(x_i \mid y). \tag{36.138}$$

The assumption is false for images. Neighbouring pixels of a garment are inked together or blank together, because they belong to the same sleeve or the same hem. How false it is, and what the error does to the classifier, we measure on Fashion-MNIST below.

36.11.1.3 A Two-Word Spam Filter

Before turning to 784 pixels, we work the smallest case to the end. A mail filter looks at two features: whether a mail contains the word "free" and whether it contains "meeting". Its training set has ten mails, 4 spam and 6 ordinary, with these counts:

	spam (4 mails)	ordinary (6 mails)
contains "free"	3	1
contains "meeting"	0	4

Counting gives every number the model needs. The prior is $\hat{p}(\text{spam}) = 4/10 = 0.4$. "Free" appears in $3/4$ of the spam and in $1/6$ of the ordinary mails; "meeting" appears in $0/4$ of the spam and in $4/6$ of the ordinary mails. A new mail contains "free" and not "meeting". Its two Bayes numerators are

$$\text{spam: } 0.4 \cdot \tfrac{3}{4} \cdot (1 - \tfrac{0}{4}) = 0.30, \qquad \text{ordinary: } 0.6 \cdot \tfrac{1}{6} \cdot (1 - \tfrac{4}{6}) = \tfrac{1}{30} \approx 0.033,$$

so the posterior probability of spam is $0.30/(0.30+0.033) = 0.90$. The same computation reads more clearly as odds. Each feature multiplies the prior odds by its likelihood ratio, the probability of what we saw under spam divided by its probability under ordinary mail:

$$\frac{p(\text{spam} \mid \mathbf{x})}{p(\text{ordinary} \mid \mathbf{x})} = \underbrace{\frac{0.4}{0.6}}_{\text{prior}} \cdot \underbrace{\frac{3/4}{1/6}}_{\text{"free"}} \cdot \underbrace{\frac{1}{2/6}}_{\text{no "meeting"}} = \frac{2}{3} \cdot 4.5 \cdot 3 = 9. \tag{36.139}$$

Odds of 9 to 1 are a probability of $9/10$. The 784-pixel classifier below is this computation with 784 likelihood ratios instead of two. The filter will return three times: when a count is zero, when we take logarithms, and when a feature is counted twice.

36.11.2 Learning by Counting

The naive Bayes rule needs the prior and one probability per class and feature. Where do these numbers come from, and how does a computer evaluate a product of 785 of them?

36.11.2.1 Maximum Likelihood and Smoothing

The numbers of the spam filter were frequencies. Maximum likelihood justifies this choice and also shows where it fails. Write $\theta_{iy} = p(x_i = 1 \mid y)$ for the probability that feature i is on in class y. Suppose class y occurs n_y times among n training examples, and feature i is on in n_{iy} of them. Under the naive model the log-likelihood of the training set splits into one term for the prior and one term per class and feature:

$$\sum_y n_y \log p(y) + \sum_y \sum_{i=1}^{d} \Big[n_{iy} \log \theta_{iy} + (n_y - n_{iy}) \log(1 - \theta_{iy}) \Big]. \tag{36.140}$$

Each bracket is the log-likelihood of n_{iy} heads in n_y flips of a coin with bias θ_{iy}, so each parameter is fitted on its own. The maximum-likelihood estimate of a coin's bias is its observed frequency (Section 36.6), which gives $\hat{\theta}_{iy} = n_{iy}/n_y$ and $\hat{p}(y) = n_y/n$.

Frequencies have one hazard. If feature i is never on in the training examples of class y, then $\hat{\theta}_{iy} = 0$, and every new example with feature i on gets a Bayes numerator of 0 for class y, whatever its other features say. One feature vetoes the class. The spam filter has such a count: "meeting" never occurred in spam, so a mail containing "free" and "meeting" has spam numerator $0.4 \cdot \frac{3}{4} \cdot 0 = 0$ and is declared ordinary with certainty. The training cell below counts how often a pixel of Fashion-MNIST is never inked in a class.

The usual fix is *Laplace smoothing* (Laplace 1814): add one imagined example with the feature on and one with it off,

$$\hat{\theta}_{iy} = \frac{n_{iy} + 1}{n_y + 2}. \tag{36.141}$$

The estimate moves a little toward $\frac{1}{2}$ and is never 0 or 1.

The smoothed estimate is a Bayesian one. Under a uniform prior on θ, n_{iy} successes in n_y trials give the posterior $\mathrm{Beta}(n_{iy}+1,\, n_y-n_{iy}+1)$, whose mean is $(n_{iy}+1)/(n_y+2)$ (Section 36.4); Laplace called this the rule of succession.

Smoothing the spam filter changes its estimates to $4/6$ and $1/6$ for "free" and "meeting" in spam, and to $2/8$ and $5/8$ in ordinary mail; we leave the prior at 0.4. The mail with both words now has the numerators $0.4 \cdot \frac{4}{6} \cdot \frac{1}{6} = 0.044$ and $0.6 \cdot \frac{2}{8} \cdot \frac{5}{8} = 0.094$, a spam probability of 0.32 instead of 0. The first mail, "free" without "meeting", drops from 0.90 to 0.80, because four spam mails are weak evidence that spam never mentions a meeting.

36.11.2.2 Scores in Log Space

The naive Bayes rule multiplies $d+1$ numbers between 0 and 1, and for an image the product is tiny. If every factor were $\frac{1}{2}$, the product of 784 of them would be $2^{-784} \approx 10^{-236}$. The smallest positive single-precision number is about $1.4 \cdot 10^{-45}$ and the smallest positive double about $4.9 \cdot 10^{-324}$, so the product underflows to 0 in single precision. A double survives this example, but not the real scores computed below, some of which are near 10^{-1000}.

The logarithm turns the product into a sum. Because log is increasing, it does not change which class is largest (Section 35.8 discusses this precaution in general). Define the *class score* $s_y(\mathbf{x})$ as the logarithm of the Bayes numerator. The rule becomes

$$\widehat{y} = \operatorname*{argmax}_y \; s_y(\mathbf{x}), \qquad s_y(\mathbf{x}) = \log p(y) + \sum_{i=1}^{d} \log p(x_i \mid y). \tag{36.142}$$

A sum of 785 logarithms replaces the product of 785 probabilities, and every term is finite as long as no estimate is 0, which smoothing guarantees. For the unsmoothed spam filter the scores of the first mail are $\log 0.30 = -1.20$ and $\log \frac{1}{30} = -3.40$; their gap, $2.20 = \log 9$, is the logarithm of the odds we computed by hand.

36.11.2.3 Linear Decision Boundaries

With binary features the class score has a simple form, and that form tells us which classifiers naive Bayes can represent. A binary feature contributes $\log \theta_{iy}$ when it is on and $\log(1-\theta_{iy})$ when it is off, so $\log p(x_i \mid y) = x_i \log \theta_{iy} + (1-x_i)\log(1-\theta_{iy})$. Collecting the terms that multiply x_i and the terms that do not,

$$s_y(\mathbf{x}) = \mathbf{w}_y^{\top}\mathbf{x} + b_y, \qquad w_{iy} = \log \frac{\theta_{iy}}{1-\theta_{iy}}, \qquad b_y = \log p(y) + \sum_{i=1}^{d} \log(1-\theta_{iy}). \tag{36.143}$$

Each inked pixel adds w_{iy}, the log-odds that the pixel is inked in class y, and the offset b_y is the score of a blank image. Every class score is affine in $\mathbf{x}$, so the boundary between any two classes, where their scores are equal, is a hyperplane (Bishop 2006). This is the first of the two results of the section worth remembering: the counted model is a linear classifier of the same family as softmax regression, with arrays of the same shapes, 784×10 weights and 10 offsets. The two methods differ only in how they choose the numbers, by counting or by gradient descent on $p(y \mid \mathbf{x})$. With two classes the posterior is a sigmoid of the score

difference, $p(y = 1 \mid \mathbf{x}) = \text{sigmoid}\big((\mathbf{w}_1 - \mathbf{w}_0)^\top \mathbf{x} + b_1 - b_0\big)$, which is the form of logistic regression (exercise 2).

The linear form depends on the kind of feature. Gaussian naive Bayes models a continuous feature x_i in class y by a Gaussian with mean μ_{iy} and variance σ_{iy}^2, so the class score contains $-(x_i - \mu_{iy})^2/(2\sigma_{iy}^2)$. If each feature has the same variance in every class, the x_i^2 terms are equal in all scores, cancel in every comparison, and the boundary is again linear. With class-specific variances they do not cancel, and the boundary is quadratic in each x_i, though still without cross terms $x_i x_j$ (exercise 3).

The Gaussian case lets us draw how much the way of choosing the numbers matters. In Figure 36.42 the two features are correlated within each class, so that each cloud of points is tilted. The naive fit models each class by a Gaussian with independent features, drawn as axis-aligned ellipses, with the same variances in both classes, so its boundary is a line. The ellipses ignore the tilt, and the line cuts across both clouds: a point from either class falls on the wrong side with probability 19%. The right panel shows the line that a direct fit of $p(y \mid \mathbf{x})$ approaches with enough data. It runs along the tilt and errs with probability 7%, and no classifier of any kind does better on these two classes.

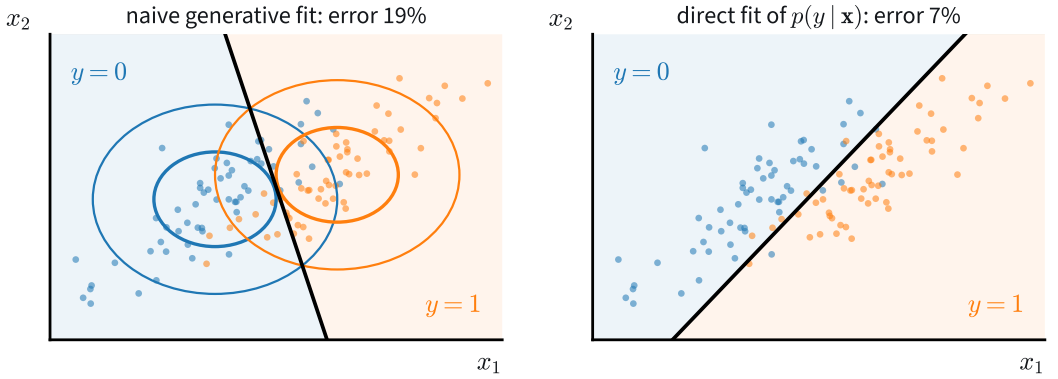


Figure 36.42: Two routes to a linear classifier on the same points; the two features are correlated within each class. Left: the naive generative fit models each class by a Gaussian with independent features (ellipses at one and two standard deviations) and takes the boundary from Bayes' rule. Right: the line fitted to $p(y \mid \mathbf{x})$ directly. Errors are population values for equal priors.

Text, not pixels, is the standard application of naive Bayes: a spam filter is the two-word example above with tens of thousands of words (Manning et al. 2008). A document can be modeled by which words occur or by how often each occurs, and the score is linear in both cases. We stay with pixels; exercise 5 develops the two event models for text, which can disagree about the same document.

36.11.3 Fashion-MNIST by Counting

A spam filter with two words can be checked by hand. Whether the same arithmetic survives 784 features that visibly depend on each other can only be tried, and Fashion-MNIST lets us compare the result with a model we already trained on the same images.

36.11.3.1 Binary Pixels

Fashion-MNIST (Xiao et al. 2017) has 60,000 training and 10,000 test images of ten kinds of garments, 28×28 grayscale pixels each (Section 3.2). Its background pixels are exactly 0, so we set $x_i = 1$ where pixel i has any value above 0, that is, where it belongs to the garment, and $x_i = 0$ on the background. The loader returns the raw images of both splits; we flatten each image into a vector of 784 binary features.

```
data = d2l.FashionMNIST()
def binarize(images):  # 1 where the pixel belongs to the garment
    return (images.reshape(-1, 784) > 0).astype(np.float32)

X, Y = binarize(data.train.data.numpy()), data.train.targets.numpy()
X_test, Y_test = binarize(data.val.data.numpy()), data.val.targets.numpy()
print(X.shape, X_test.shape, f'ink fraction {X.mean():.2f}')
```

```
(60000, 784) (10000, 784) ink fraction 0.50
```

Half of all pixels are inked. Each class has 6,000 training and 1,000 test images, so the estimated prior is 0.1 for every class and the prior term is the same in every score; the code keeps it anyway.

36.11.3.2 Templates and Samples

Training is two counts: the images per class, n_y, and for each class and pixel the images in which the pixel is inked, n_{iy}. The code stores the smoothed estimates $\widehat{\theta}_{iy}$ in the array `P_xy`, one row per class. While counting, the cell settles what the discussion of smoothing left open: how many pixels are never inked in the training images of a class, and how many test images ink one of them?

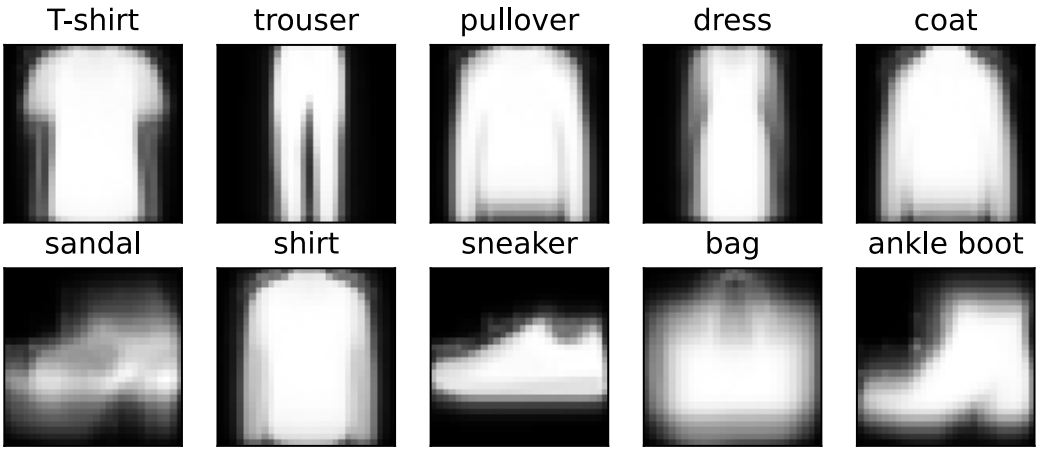
```
n_y = np.bincount(Y, minlength=10)              # images per class
n_x = np.stack([X[Y == y].sum(axis=0) for y in range(10)])
P_y = n_y / n_y.sum()                           # prior p(y)
P_xy = (n_x + 1) / (n_y + 2)[:, None]           # smoothed theta_iy
print('prior:', P_y)
print('never-inked pixels per class:', (n_x == 0).sum(axis=1))
unseen = (X_test * (n_x[Y_test] == 0)).sum(axis=1) > 0
print('test images inking one of their class:', unseen.sum())
```

```
prior: [0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1]
never-inked pixels per class: [ 0 13  1  0  3  3  0 57  0  1]
test images inking one of their class: 5
```

Zero counts are rare here: none for half the classes, 13 for trousers, and 57 for sneakers, which are flat and leave the top of the image blank. Only 5 test images ink a pixel that their own class never inked in training. Without smoothing, each of these five would get a score of $-\infty$ for its true class and be misclassified whatever its other 783 pixels say.

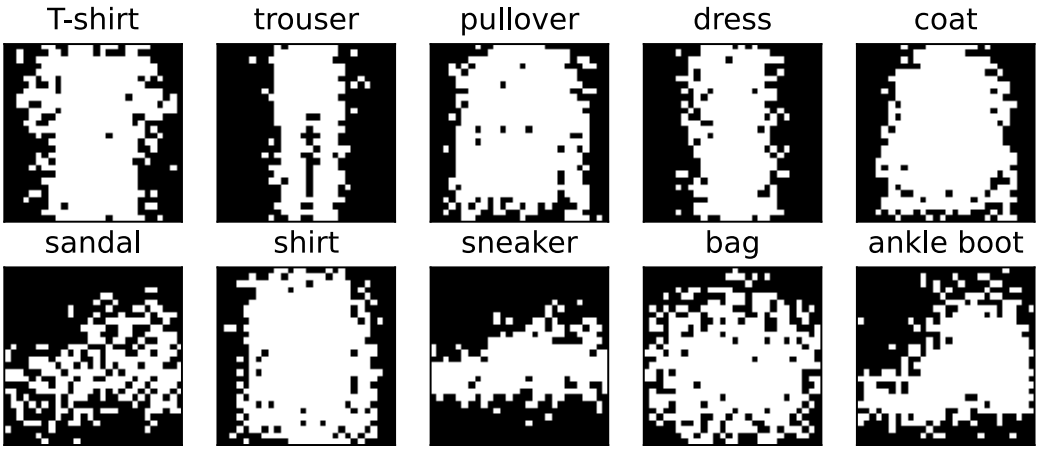
Each row of `P_xy` is an image: the fraction of the class's training images in which each pixel is inked. We show the ten rows as 28×28 pictures, one per class.

```
d2l.show_images(P_xy.reshape(10, 28, 28), 2, 5,
                titles=data.text_labels(range(10)), cmap='gray');
```



The templates are blurred outlines of the garments. Each template is the per-pixel mean of its class, so the blur shows how much the garments vary in placement and shape. Any model that matches the per-pixel frequencies of these images, with or without independence, has the same templates; the blur is a property of the data. The naive assumption adds something else: these means, together with the prior, are all the model keeps. To see what that implies, we draw images from the model, inking every pixel independently with probability $\hat{\theta}_{iy}$.

```
rng = np.random.default_rng()
samples = (rng.random((1000, 10, 784)) < P_xy).astype(np.float32)
d2l.show_images(samples[0].reshape(10, 28, 28), 2, 5,
                titles=data.text_labels(range(10)), cmap='gray');
```



A sample has the right silhouette, but a ragged, speckled outline, because every pixel near an edge is decided by its own coin. We can put a number on the difference. Along a row of a real image, ink and background alternate a few times; in a sample they should alternate

more often. We count the ink/background switches per row, and the amount of ink, in the training images and in 1,000 samples per class.

```
def switches(images):  # ink/background changes along a row, mean per row
    return np.abs(np.diff(images.reshape(-1, 28, 28), axis=2)).sum(axis=2).mean()

print(f'ink fraction: data {X.mean():.3f}, samples {samples.mean():.3f}')
print('switches per row, data:   ',
      np.round([switches(X[Y == y]) for y in range(10)], 1))
print('switches per row, samples:',
      np.round([switches(samples[:, y]) for y in range(10)], 1))
```

```
ink fraction: data 0.498, samples 0.498
switches per row, data:    [3.8 3.2 3.1 3.1 3.1 3.  3.3 1.5 3.1 2.6]
switches per row, samples: [5.1 4.2 4.8 4.7 4.8 8.2 5.4 4.  8.2 5.9]
```

The samples carry the same amount of ink as the data, since each pixel is inked with the right frequency. But they switch between ink and background far more often: for pullovers about 5 times per row instead of 3, for sandals and bags about 8 times instead of 3. Real garments have smooth edges because neighbouring pixels agree, and that agreement is what the model discards.

36.11.3.3 Checking the Assumption

The samples show the assumption failing over the whole image; one pair of pixels shows it with numbers. We take two neighbouring pixels on the bottom hem of pullovers, in row 26 and columns 14 and 15. If they were independent given the class, the fraction of pullovers with both inked would equal the product of the two fractions. The cell compares this product with the joint frequency, in the training pullovers and in the 1,000 sampled ones.

```
i, j = 26 * 28 + 14, 26 * 28 + 15     # two neighbouring hem pixels
for name, Z in (('data', X[Y == 2]), ('model', samples[:, 2])):
    p_i, p_j, both = Z[:, i].mean(), Z[:, j].mean(), (Z[:, i] * Z[:, j]).mean()
    print(f'{name:5s}: P(i)={p_i:.2f}  P(j)={p_j:.2f}  '
          f'P(i)P(j)={p_i * p_j:.2f}  P(i,j)={both:.2f}')
```

```
data : P(i)=0.55  P(j)=0.55  P(i)P(j)=0.30  P(i,j)=0.54
model: P(i)=0.54  P(j)=0.51  P(i)P(j)=0.28  P(i,j)=0.26
```

Each pixel is inked in about 55% of the pullovers. Independence predicts that both are inked in $0.55 \times 0.55 \approx 30\%$ of them; in the data both are inked in 54%, almost as often as either one alone. The hem either reaches row 26 or it does not, so the two pixels are nearly one feature recorded twice. The model's samples, by construction, follow the product.

36.11.3.4 Test Accuracy

Prediction uses the affine form Equation 36.143. We compute the weights w_{iy} and offsets b_y from `P_xy` and `P_y`, score all test images and classes with one matrix product, and keep the predictions for the rest of the section. Softmax regression reached about 83% with scores of this form. How much does counting lose?

```
W = np.log(P_xy) - np.log1p(-P_xy)              # w_iy, log-odds per pixel
b = np.log1p(-P_xy).sum(axis=1) + np.log(P_y)   # b_y, score of a blank image
s = X_test @ W.T + b                            # s_y(x), shape (10000, 10)
pred = s.argmax(axis=1)
print(f'test accuracy {(pred == Y_test).mean():.4f}, training accuracy '
      f'{((X @ W.T + b).argmax(axis=1) == Y).mean():.4f}')
```

```
test accuracy 0.7059, training accuracy 0.7171
```

Naive Bayes classifies about 71% of the test images correctly, against 10% for guessing. The number does not change from run to run, because counting involves no randomness. How precise it is we ask under error bars below.

The second number deserves a moment. On the 60,000 images it was fitted to, the model is right 72% of the time, one point more than on images it has never seen. A model that had memorized its training set would show a large gap here; this one has almost none. Its mistakes are therefore not a matter of too little data. It is wrong about its own training images in the same way, and for the same reason, as about new ones: in the terms of Section 36.5, the error is bias and not variance. More images would not help. A better model would, and softmax regression will show below that one exists within the same family of classifiers.

The scores also show what the raw product of Equation 36.138 would have done. Each score is the natural logarithm of a Bayes numerator, so dividing by $\log 10$ gives its decimal exponent. We compare the numerators with the smallest positive single-precision number and with the smallest *normal* double, below which a double loses precision.

```
log10 = s / np.log(10)                    # decimal exponent of each numerator
print(f'log10 of the numerators: largest {log10.max():.0f}, smallest '
      f'{log10.min():.0f}, median best {np.median(log10.max(axis=1)):.0f}')
f32 = -149 * np.log(2)     # log of 2**-149, the smallest positive float32
f64 = -1022 * np.log(2)    # log of 2**-1022, the smallest normal float64
print(f'images whose best numerator underflows float32: '
      f'{(s.max(axis=1) < f32).mean():.2f}')
print(f'numerators below the smallest normal float64: {(s < f64).mean():.2f}, '
      f'images whose best one is: {(s.max(axis=1) < f64).mean():.4f}')
print(f'accuracy of the raw float64 product: '
      f'{(np.exp(s).argmax(axis=1) == Y_test).mean():.4f}')
```

```
log10 of the numerators: largest -32, smallest -1002, median best -81
images whose best numerator underflows float32: 0.96
numerators below the smallest normal float64: 0.44, images whose best one is: 0.0003
accuracy of the raw float64 product: 0.7058
```

The Bayes numerators lie between about 10^{-32} and 10^{-1000}, and the best one of a typical image is about 10^{-81}. For 96% of the test images even the largest of the ten numerators is below the smallest positive single-precision number, so the raw rule would see ten zeros and could not classify them at all. In double precision 44% of the numerators fall below the smallest normal number, but nearly all of these belong to losing classes. The largest numerator stays an ordinary double for all but three images in ten thousand, and the raw product in double precision reaches the accuracy of the log scores to within one image in ten thousand. So on 784 pixels a double still classifies, but only just: the best numerator of a typical image is 10^{-81}, and an input four times as long would push it under as well. The

log scores never leave the range of ordinary numbers.

36.11.4 Evaluating the Classifier

An accuracy of 71% raises three questions. How precise is it? Which mistakes make up the missing 29%? And can the probabilities the model reports be trusted?

36.11.4.1 Error Bars

The test accuracy is computed from 10,000 random test images, so another test set would give a slightly different number. Each image is a Bernoulli outcome, correct or not. This is the coin with which the chapter opened, flipped once per test image, so the accuracy $\hat{p}$ is the mean of n flips and has the standard error $\sqrt{\hat{p}(1-\hat{p})/n}$ (Section 36.5). For $\hat{p} = 0.706$ and $n = 10{,}000$ this is

$$\sqrt{\frac{0.706 \cdot 0.294}{10^4}} \approx 0.0046, \qquad 1.96 \times 0.0046 \approx 0.009,$$

so a 95% confidence interval reaches about 0.9 points to either side. The bootstrap of Section 36.5 gives a second, formula-free answer: resample the test images with replacement, recompute the accuracy on each resample, and read off the middle 95% of the results.

```
rng = np.random.default_rng()
correct = pred == Y_test                          # per-image 0/1 outcomes
n, p = len(correct), correct.mean()
boot = correct[rng.integers(0, n, (1000, n))].mean(axis=1)  # 1,000 resamples
print(f'accuracy {p:.4f}, bootstrap 95% interval '
      f'({np.percentile(boot, 2.5):.3f}, {np.percentile(boot, 97.5):.3f})')
print(f'binomial half-width 1.96 se = {1.96 * np.sqrt(p * (1 - p) / n):.4f}')
```

```
accuracy 0.7059, bootstrap 95% interval (0.696, 0.715)
binomial half-width 1.96 se = 0.0089
```

The bootstrap interval runs from about 0.70 to 0.715, in agreement with the binomial half-width of 0.009. The accuracy is therefore $71\% \pm 0.9$ points; the digits after the decimal point of "70.59%" are noise. Two classifiers evaluated on the same test set make correlated errors, so a paired analysis of their per-image differences compares them more sensitively than two separate intervals (exercise 6).

36.11.4.2 Confusions and Weights

The error rate says how often the model is wrong; the confusion matrix of Section 3.5 says how. We count it in the orientation of Equation 3.20: rows are the predicted classes and columns the true classes, so each column shows where the 1,000 test images of one garment went.

```
conf = np.zeros((10, 10), dtype=int)
np.add.at(conf, (pred, Y_test), 1)       # N[i, j]: class j predicted as i
print(conf)
```

```
[[748   9   8  50   2   0 218   0   0   0]
 [  0 907   0  53   3   0   4   0   3   0]
 [ 14  18 665   5 372   0 232   0  14   0]
 [122  48  12 808 104   2  93   0  68   7]
 [  8  16 242  45 483   0 222   0   6   0]
 [  0   0   0   0   0 680   0   5   1   0]
 [ 83   0  50  35  25   2 167   0  20   0]
 [  1   0   0   0   0 232   1 893  55 113]
 [ 24   2  23   4  11   9  63   0 832   4]
 [  0   0   0   0   0  75   0 102   1 876]]
```

A hundred counts are easier to read from their largest entries. We list the six largest off-diagonal counts, true class first, and the accuracy of each class, its diagonal entry divided by its column sum.

```
names = data.text_labels(range(10))
off = conf - np.diag(np.diag(conf))      # errors only
order = np.argsort(off, axis=None, kind='stable')[::-1][:6]  # largest first
pairs = [f'{names[t]}->{names[q]} {off[q, t]}'  # true -> predicted
         for q, t in zip(*np.unravel_index(order, off.shape))]
print(', '.join(pairs[:3]))
print(', '.join(pairs[3:]))
print('per-class accuracy:', np.round(np.diag(conf) / conf.sum(axis=0), 2))
```

```
coat->pullover 372, pullover->coat 242, sandal->sneaker 232
shirt->pullover 232, shirt->coat 222, shirt->T-shirt 218
per-class accuracy: [0.75 0.91 0.66 0.81 0.48 0.68 0.17 0.89 0.83 0.88]
```

The errors are concentrated. Coats are taken for pullovers 372 times and pullovers for coats 242 times; shirts are taken for pullovers, coats and t-shirts more than 200 times each; sandals for sneakers 232 times. Trousers (91%) and sneakers (89%) are easy, coats (48%) and shirts (17%) are not. These are the two groups that softmax regression confused in Section 3.3, the four upper-body garments and the three kinds of shoes, but the counted model does much worse inside the first: there shirts were recognized 40% to 60% of the time, here one time in six.

Is this the best a linear classifier can do on these pixels? Softmax regression draws hyperplanes of the same family, but chooses them by minimizing the cross-entropy of $p(y \mid \mathbf{x})$. We train it on the same binary pixels with minibatch stochastic gradient descent, so that only the method of fitting differs. The code uses the fact from Section 3.1 that the gradient of the cross-entropy with respect to the logits is the vector of predicted probabilities minus the one-hot label.

```
rng = np.random.default_rng()
W_sr, b_sr = np.zeros((10, 784)), np.zeros(10)
for epoch in range(20):                  # minibatches of 256 images
    for idx in np.array_split(rng.permutation(len(X)), len(X) // 256):
        O = X[idx] @ W_sr.T + b_sr                  # logits
        E = np.exp(O - O.max(axis=1, keepdims=True))
        G = E / E.sum(axis=1, keepdims=True) - np.eye(10)[Y[idx]]
        W_sr -= 0.05 * G.T @ X[idx] / len(idx)      # mean-loss gradient
        b_sr -= 0.05 * G.mean(axis=0)
```

```
pred_sr = (X_test @ W_sr.T + b_sr).argmax(axis=1)
conf_sr = np.zeros((10, 10), dtype=int)
np.add.at(conf_sr, (pred_sr, Y_test), 1)
print(f'accuracy: softmax regression {(pred_sr == Y_test).mean():.3f}, '
      f'naive Bayes {(pred == Y_test).mean():.3f}')
print('pullover<->coat errors: softmax regression',
      conf_sr[2, 4] + conf_sr[4, 2], 'naive Bayes', conf[2, 4] + conf[4, 2])
```

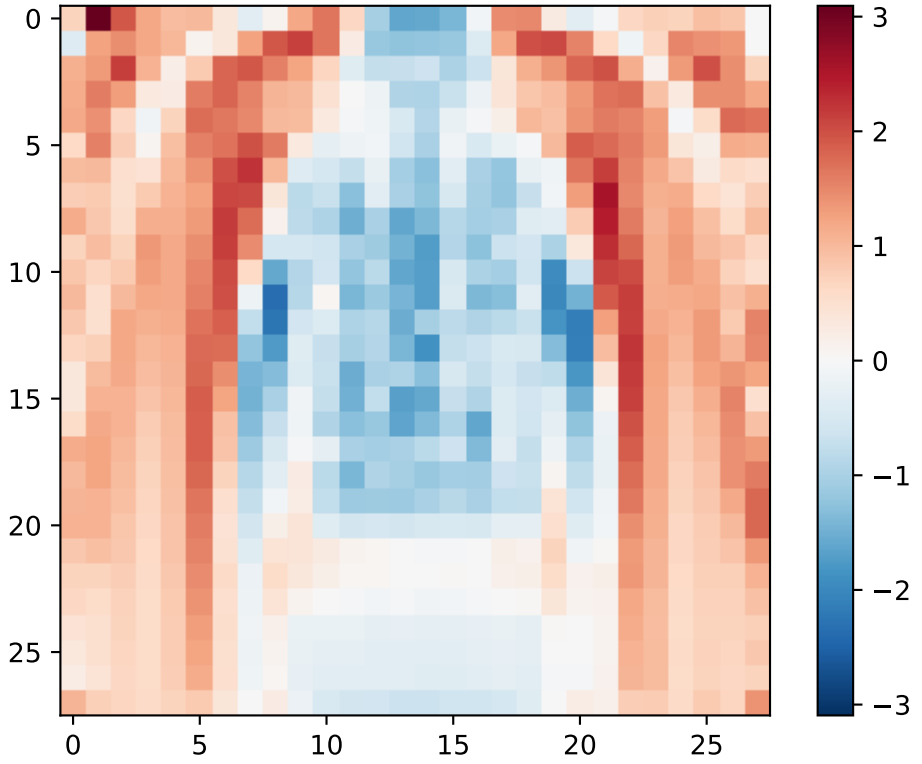
```
accuracy: softmax regression 0.831, naive Bayes 0.706
pullover<->coat errors: softmax regression 301 naive Bayes 614
```

Softmax regression reaches about 83%, twelve points above naive Bayes, and roughly halves the confusions between pullovers and coats. The family of hyperplanes is therefore not the limit, and the loss lies in how counting sets the weights. The run does not say what is wrong with them; we return to this under calibration.

The weights themselves show what naive Bayes uses to tell two classes apart. The score difference between pullover and coat is affine with the weight vector $\mathbf{w}_{\text{pullover}} - \mathbf{w}_{\text{coat}}$, and an inked pixel moves the decision toward pullover by its entry, compared with the same pixel left blank. We display this vector as an image, red for pullover and blue for coat, and print its largest entry together with the counts behind it.

```
w = (W[2] - W[4]).reshape(28, 28)        # pullover minus coat, in nats
m = np.abs(w).max()
d2l.plt.imshow(w, cmap='RdBu_r', vmin=-m, vmax=m)
d2l.plt.colorbar()
k = np.abs(w).argmax()
print(f'largest |weight| {m:.1f} nats at row {k // 28}, column {k % 28}: '
      f'inked in {n_x[2, k]:.0f} pullovers and {n_x[4, k]:.0f} coats')
```

```
largest |weight| 3.1 nats at row 0, column 1: inked in 21 pullovers and 0 coats
```



Red pixels lie along both sides of the image and in the top rows, where pullovers are inked more often than coats. Blue pixels fill the torso, with weights near -1, and it would be wrong to read them as strong evidence for a coat. A weight is the score of an inked pixel relative to a blank one. A torso pixel is inked in about 99% of the pullovers and in more than 99.5% of the coats, so finding it inked says almost nothing, about -0.01 nats. The weight of -1 is what the model gains, in favour of pullover, from the rare image whose torso pixel is blank. The decision is made along the sides. The largest weight, about 3 nats, sits in a corner that is inked in about twenty pullovers and in no coat, so its size is set by the pseudo-count of smoothing rather than by much evidence.

A generative model has one advantage that the direct fit lacks. If some pixels are missing at test time, say hidden by an occlusion, naive Bayes drops their factors from the product. Under the model this is exact, since summing $p(x_i \mid y)$ over the two values of a hidden x_i gives 1. Softmax regression has no such operation and must fill in a value (exercise 11).

36.11.4.3 Calibration

The scores do more than rank the classes. Exponentiating them restores the Bayes numerators, and normalizing restores the denominator that we dropped, so the model's posterior is the softmax of its scores:

$$p(y \mid \mathbf{x}) = \frac{\exp s_y(\mathbf{x})}{\sum_{y'} \exp s_{y'}(\mathbf{x})}. \tag{36.144}$$

Every prediction therefore comes with a confidence, the largest of these probabilities. As

in Section 3.5, a classifier is *calibrated* if its confidences match frequencies: among the predictions made with confidence c, a fraction c is correct. To check, we sort the test images into bins by confidence and compare, in each bin, the mean claimed confidence with the achieved accuracy.

The *expected calibration error* summarizes the check in one number, the gap between claimed confidence and achieved accuracy averaged over the bins with the fraction of images in each as weight. The softmax is computed in its stable form, with the largest score subtracted first (Section 35.8.2). The cell also prints the median gap between the best and the second-best score.

```
post = np.exp(s - s.max(axis=1, keepdims=True))      # softmax, stably
post /= post.sum(axis=1, keepdims=True)
c = post.max(axis=1)                                 # claimed confidence
print(f'mean confidence {c.mean():.3f}, accuracy {correct.mean():.3f}')
edges, ece = [0, 0.9, 0.99, 0.999, 1], 0
for lo, hi in zip(edges[:-1], edges[1:]):
    m = (c > lo) & (c <= hi)
    ece += m.mean() * abs(c[m].mean() - correct[m].mean())
    print(f'({lo}, {hi}]: {m.sum():5d} images, claimed {c[m].mean():.3f}, '
          f'achieved {correct[m].mean():.3f}')
top2 = np.sort(s, axis=1)[:, -2:]
print(f'ECE {ece:.2f}, median top-two score gap '
      f'{np.median(top2[:, 1] - top2[:, 0]):.0f} nats')
```

```
mean confidence 0.992, accuracy 0.706
(0, 0.9]:   257 images, claimed 0.726, achieved 0.350
(0.9, 0.99]:   297 images, claimed 0.961, achieved 0.384
(0.99, 0.999]:   265 images, claimed 0.996, achieved 0.419
(0.999, 1]:  9181 images, claimed 1.000, achieved 0.735
ECE 0.29, median top-two score gap 61 nats
```

The model claims 99% confidence on average and is right 71% of the time. Most test images, about 92%, land in the top bin, where the model claims certainty to three decimals and is right on about 73% of them. In the three lower bins it claims between 73% and 99.6% and achieves only 35% to 42%. Every bin is overconfident, so the expected calibration error equals the mean confidence minus the accuracy, about 0.29; softmax regression in Section 3.4 stayed below 0.03. The ordering still carries some information, since accuracy rises from bin to bin, but the probabilities cannot be used as stated.

The cause is the one we met at the hem: correlated pixels are counted as independent evidence. Its simplest case is a feature that is stored several times, and the calculation takes three lines. Take two classes with priors π_0 and π_1 and a feature x with likelihood ratio $r(x) = p(x \mid y = 1)/p(x \mid y = 0)$, and suppose the data contain k identical copies of x. The copies always agree, so together they carry the evidence of one, and the correct log posterior odds are $\log(\pi_1/\pi_0) + \log r(x)$. Naive Bayes treats the copies as conditionally independent, multiplies k equal factors $p(x \mid y)$, and reports

$$\log \frac{\pi_1}{\pi_0} + k \log r(x). \tag{36.145}$$

Duplicated evidence multiplies the log-odds. With equal priors the sign of the log-odds does not change, so naive Bayes predicts the same class for every k, but it claims the odds $\bar{r}^k$

where $\bar{r}$ is correct, with $\bar{r} = \max(r, 1/r)$. This is the second result to remember: the ranking survives and the confidence does not.

Take a clue that is right 80% of the time, $p(x = 1 \mid y = 1) = 0.8$ and $p(x = 1 \mid y = 0) = 0.2$, so that $r = 4$ when the clue is on. One copy gives the correct confidence 0.80; two, three, four and five copies make naive Bayes claim 0.94, 0.985, 0.996 and 0.999, while its predictions stay right 80% of the time. Duplicating the word "free" in the spam filter has the same effect: the odds of the first mail rise from 9 to $\frac{2}{3} \cdot 4.5^2 \cdot 3 = 40.5$, a spam probability of 0.976.

The calculation protects the ranking only for equal priors, and the restriction is real. With prior odds $\pi_1/\pi_0 = 1/5$ and $r = 4$, one copy gives posterior odds $4/5$ and the correct prediction, class 0; two copies give $16/5$ and predict class 1. Repeated evidence can overturn a prior that the evidence alone should not overturn. The ranking is safe when the evidence points the same way as the prior, or when a single copy already outweighs it.

On Fashion-MNIST the 784 pixels are not identical copies of one another, but many of them are strongly correlated. The two hem pixels are nearly one pixel counted twice, and many such groups add up to the median score gap of about 60 nats between the best and second-best class. Since $e^{-60} \approx 10^{-26}$, the softmax turns such a gap into a confidence of 1. That the classes are still ranked usefully is typical of naive Bayes (Domingos and Pazzani 1997). The ranking is not untouched, however. Correlated groups differ in size, so a large region of redundant pixels can outvote a small region that carries independent information. This is a plausible reason for the twelve points lost to softmax regression, which is free to discount pixels that repeat their neighbours, but our run did not test it.

The overconfidence has a cheap repair, which we only name here. Temperature scaling (Section 3.5) divides all scores by one number T fitted on held-out data, which leaves the ranking as it is; for k exact copies and equal priors, $T = k$ restores the correct posterior. Exercise 10 fits T for the counted model. Nor is overconfidence peculiar to naive Bayes. Deep networks are often overconfident in the same direction, though usually by less (C. Guo et al. 2017), and the reliability check above applies to any classifier whose probabilities will be used.

One question is left open. With 6,000 images per class the direct fit won. With few examples the order is often reversed: Ng and Jordan (2002) show that counted parameters settle after a number of examples logarithmic in d, while the direct fit can need a number linear in d. Exercise 12 tests this on pullovers and coats.

36.11.5 Summary

Bayes' rule builds a classifier from a model of how each class produces its inputs, but a table of $p(\mathbf{x} \mid y)$ over 10^{236} binary images cannot be estimated. The naive assumption, that the features are conditionally independent given the class, reduces the model to one frequency per class and feature, smoothed by one pseudo-count, so training is counting. The log of the naive Bayes numerator is affine in binary features, so Bernoulli naive Bayes is a linear classifier of the same family as softmax regression.

On Fashion-MNIST it reaches about 71% $\pm$ 0.9 points, twelve points below the direct fit of the same family. It claims 99% confidence, because correlated pixels are counted as independent evidence and duplicated evidence multiplies the log-odds. Its overconfidence is the reason to check the calibration of any classifier.

36.11.6 Exercises

1. **XOR and the naive assumption.** The four points $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ carry the labels $y = x_1 \oplus x_2$, that is 0, 1, 1 and 0. Each point occurs once in the training set.
   1. Compute the maximum-likelihood estimates of $p(y)$ and $\theta_{iy} = p(x_i = 1 \mid y)$, and the class scores of all four points. Which predictions does the naive Bayes rule Equation 36.138 make?
   2. Show that no choice of Bernoulli naive Bayes parameters classifies all four points correctly, using the affine form Equation 36.143 of the scores. Why can degenerate parameters $\theta_{iy} \in \{0,1\}$ not help? (Each class must give positive probability to its own two points.)
   3. Replace the two factors $p(x_1 \mid y)\,p(x_2 \mid y)$ by a joint table $p(x_1, x_2 \mid y)$. In the language of Bayesian networks (Koller and Friedman 2009), this adds the edge between x_1 and x_2 that the right panel of Figure 36.41 shows. Estimate this model from the four points and show that it classifies them correctly. Why is its class score no longer affine in $\mathbf{x}$?
   4. Within one class, let x_1 and x_2 be independent fair bits and let $x_3 = x_1 \oplus x_2$. Show that every pair of the three features is independent. Compare the probability of each of the eight patterns with the product of the three marginals. Why can a check of all pairwise correlations within a class not confirm the assumption Equation 36.137?
2. **Naive Bayes as logistic regression.** Consider two classes $y \in \{0,1\}$ with the prior $\pi_1 = p(y=1)$ and the Bernoulli parameters $\theta_{iy} = p(x_i = 1 \mid y)$.
   1. Show that the naive Bayes posterior is $p(y=1 \mid \mathbf{x}) = \operatorname{sigmoid}(\mathbf{w}^\top \mathbf{x} + b)$ with $\operatorname{sigmoid}(z) = 1/(1+e^{-z})$ and $w_i = \log \frac{\theta_{i1}(1-\theta_{i0})}{\theta_{i0}(1-\theta_{i1})}$. Express b through the θ_{iy} and π_1. Which sign does w_i have for a pixel that is inked more often in class 1 than in class 0?
   2. Show that, conversely, every pair $(\mathbf{w}, b)$ arises from some choice of $\theta_{iy} \in (0,1)$ and $\pi_1 \in (0,1)$. Naive Bayes and logistic regression therefore define the same set of posteriors and differ in how they fit it.
   3. Suppose the two classes of the training set are linearly separable. How does the conditional likelihood of logistic regression change when a separating $(\mathbf{w}, b)$ is multiplied by a factor $t \to \infty$? What does this imply for the maximum-likelihood estimate of $(\mathbf{w}, b)$? Which $(\mathbf{w}, b)$ does naive Bayes with Laplace smoothing return on the same data?
3. **Gaussian class-conditionals.** A single feature has $x \mid y = 0 \sim \mathcal{N}(0,1)$ and $x \mid y = 1 \sim \mathcal{N}(0,4)$, and both classes have prior $1/2$.
   1. Find the set of x that the Bayes classifier assigns to class 1. Why can no threshold on x produce this set? Which transformed feature turns the boundary into a threshold?
   2. In d dimensions, Gaussian naive Bayes with class-specific variances has the class score
   $$\log p(y) - \sum_{i=1}^{d} \left[\frac{(x_i - \mu_{iy})^2}{2\sigma_{iy}^2} + \frac{1}{2} \log\left(2\pi\sigma_{iy}^2\right) \right]. \tag{36.146}$$
   Show that the boundary between two classes is a quadratic surface without cross terms $x_i x_j$.
   3. Two classes in two dimensions have equal priors and the class-conditionals $\mathcal{N}(\mathbf{0}, \Sigma_+)$ and $\mathcal{N}(\mathbf{0}, \Sigma_-)$ with $\Sigma_\pm = \left(\begin{smallmatrix} 1 & \pm\rho \\ \pm\rho & 1 \end{smallmatrix}\right)$ for some $0 < \rho < 1$. Show

that the difference of the two class scores is $2\rho x_1 x_2/(1-\rho^2)$, so that the Bayes classifier assigns class $+$ exactly when $x_1 x_2 > 0$, and that each single score also depends on $x_1^2 + x_2^2$. The two classes have the same marginals, so Gaussian naive Bayes is at chance even with unlimited data. Under $\mathcal{N}(\mathbf{0}, \Sigma_+)$, $P(x_1 x_2 > 0) = \frac{1}{2} + \frac{1}{\pi} \arcsin \rho$ (Sheppard's formula). Find the Bayes error as a function of ρ, and evaluate it for $\rho = 0.1$ and $\rho = 0.9$.

4. code **Smoothing and unseen pixels.** The `#naive-bayes-train` cell estimates θ_{iy} by $(n_{iy} + 1)/(n_y + 2)$.
   1. Generalize the cell to a pseudocount α, estimating θ_{iy} by $(n_{iy} + \alpha)/(n_y + 2\alpha)$, and report the test accuracy for $\alpha \in \{0, 0.01, 0.1, 1, 10, 100, 1000\}$. In floating point, $0 \cdot (-\infty)$ is NaN. Which values does the affine score of the `#naive-bayes-predict` cell take at $\alpha = 0$, and which class does `argmax` then return for every image?
   2. Rewrite the score so that each pixel selects one of its two log probabilities, for example with `np.where` one class at a time, instead of multiplying both by x_i and $1 - x_i$. Rerun $\alpha = 0$. Which fraction of the class scores is $-\infty$, for how many test images is every class score $-\infty$, and what is the accuracy? Was the failure in (a) a property of the unsmoothed model or of its implementation?
   3. The estimate with pseudocount α is the posterior mean of θ_{iy} under a Beta prior. Which prior is it? Why does the accuracy fall as α grows large?
5. **Two event models for text.** A vocabulary V has the four words *goal*, *match*, *vote* and *poll*. The training set contains three sports documents, "goal goal match", "goal match match" and "match vote", and three politics documents, "vote poll poll", "vote vote match" and "poll vote". The multinomial model treats a document as a sequence of independent word draws from a class-specific distribution over V. The Bernoulli model records for each word of V only whether it occurs in the document.
   1. Estimate both models with Laplace smoothing. For the multinomial model, add 1 to each word count of a class and $|V|$ to the class's total count. For the Bernoulli model, estimate the probability that a word occurs by $(n_{iy} + 1)/(n_y + 2)$, where n_{iy} counts the class-y documents that contain word i.
   2. Classify the document "goal match poll poll" with both models. Which words drive each decision?
   3. Concatenate a document with itself k times. How do the log posterior odds of the two models change? In real text, a word that has occurred once in a document tends to occur again (Church and Gale 1995), a finding that exercise 6 of Section 36.3 reproduces. Which assumption of the multinomial model does this violate?
6. code **Paired comparison.** The text compared naive Bayes with softmax regression by their accuracies, `pred` against `pred_sr`.
   1. Bootstrap the per-image difference in correctness, using the same resampled indices for both models, and report a 95% interval for the difference of the accuracies.
   2. Compare the width of this interval with that of the two marginal intervals and with the unpaired standard error $\sqrt{p_1(1-p_1)/n + p_2(1-p_2)/n}$. Compute the correlation of the two vectors of correctness, and use it to explain why pairing helps.
   3. Count the test images that both models get wrong, that only naive Bayes gets right, and that only softmax regression gets right. Which of these counts determine the difference of the accuracies? What would the table look like for a model that

differs from naive Bayes only by correcting some of its errors?

7. **Duplicated and correlated evidence.** Two classes have equal priors, and a binary feature has $p(x = 1 \mid y = 1) = 0.8$ and $p(x = 1 \mid y = 0) = 0.2$.
   1. A pipeline stores the feature k times. Find the smallest k for which naive Bayes claims at least 99.9% confidence for predictions that are right 80% of the time.
   2. Now store two copies that agree only partly: given y, the second copy equals the first with probability q and is otherwise a fresh draw from the same class-conditional distribution. Show that both copies have the same marginal as x, so naive Bayes still claims $16/17 \approx 0.94$ when both copies are 1. Show that the correct posterior is $r/(1 + r)$ with $r = 4(0.8 + 0.2q)/(0.2 + 0.8q)$, and evaluate it for $q \in \{1, 0.75, 0.5, 0.25, 0\}$. How does the over-count depend on the correlation of the copies?
   3. Show that dividing every class score by $T = k$ before the softmax restores the correct posterior in (a). Why are equal priors needed for this? (The prior term is divided by k too.)
8. code **Label shift.** Under label shift (Section 3.7), the target prior $p_T(y)$ differs from the source prior $p_S(y)$, while $p(\mathbf{x} \mid y)$ stays the same.
   1. Show that the posterior under the target prior is $p_T(y \mid \mathbf{x}) \propto p_S(y \mid \mathbf{x})\, p_T(y)/p_S(y)$. Why does naive Bayes need no retraining to use it?
   2. Build a test set in which pullovers make up 90% of the images: all 1,000 test pullovers and 111 images drawn at random from the other classes. Compare the accuracy of `pred` on it with that of the corrected rule of (a), with $p_T(\text{pullover}) = 0.9$ and $p_T(y) = 1/90$ for every other class.
   3. The correction adds $\log 9 \approx 2.2$ nats to the pullover score and subtracts 2.2 from the others. Compare this with the score gaps of the `#mdl-naive-bayes-calibration-3` cell and explain the size of the change you measured in (b).
9. code **Score gaps under the model.** ⊠ For two classes u and v, let $g(\mathbf{x}) = s_u(\mathbf{x}) - s_v(\mathbf{x})$ be the difference of their scores, so that the rule ranks v above u exactly when $g(\mathbf{x}) < 0$. By Equation 36.143, $g(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b_u - b_v$ with $\mathbf{w} = \mathbf{w}_u - \mathbf{w}_v$.
   1. Let $\mathbf{x}$ follow the model's own class-u distribution, with independent pixels $x_i \sim \text{Bernoulli}(\theta_{iu})$. Show that $\mathbb{E}[g]$ equals the Kullback–Leibler divergence $\sum_{\mathbf{x}} p(\mathbf{x} \mid u) \log(p(\mathbf{x} \mid u)/p(\mathbf{x} \mid v))$ of the two class-conditional distributions of the model plus $\log(p(u)/p(v))$, and that $\text{Var}[g] = \sum_i w_i^2\, \theta_{iu}(1 - \theta_{iu})$.
   2. Let u be pullover and v coat. Compute g on the test pullovers, and compare its empirical mean and standard deviation with the two values of (a). From the model's mean and variance, use a Gaussian approximation to predict the fraction of these images with $g < 0$, and compare it with the observed fraction.
   3. On real images, $\text{Var}[g] = \mathbf{w}^\top \Sigma_u \mathbf{w}$, where Σ_u is the covariance matrix of the pixels of class u. Which entries of Σ_u does the variance of
      (a) omit? Show that their contribution $\sum_{i \neq j} w_i w_j \Sigma_{u,ij}$ is positive when positively correlated neighbours carry weights of the same sign, and check both the correlation and the sign agreement for the hem pixels of the text.
10. code **Temperature scaling.** ⊠ Exercise 7 shows that dividing the class scores by $T = k$ undoes a k-fold duplication of features. For Fashion-MNIST the effective duplication is unknown, and temperature scaling (C. Guo et al. 2017) fits T on held-out data instead: the posterior becomes the softmax of $s_y(\mathbf{x})/T$.
    1. Refit the model on the first $50{,}000$ training images, and choose T by minimizing the average negative log-likelihood of the posterior on the remaining $10{,}000$. Why does T not change the test accuracy?

2. Apply the fitted T to the scores of the same $50{,}000$-image model on the test set, and repeat the reliability table of the `#mdl-naive-bayes-calibration-3` cell with the bin edges 0, 0.5, 0.7, 0.9, 0.99 and 1. Compare the mean claimed confidence with the accuracy.
3. In the setting of exercise 7, show that the standard deviation of the score gap on real data is $\sqrt{k}$ times the standard deviation that the model predicts. Compute the square of this ratio for the pair pullover/coat of exercise 9 and for t-shirt/shirt and pullover/shirt, and compare it with the fitted T.
4. Apply the prior correction of exercise 8 to the temperature-scaled posterior, on the test set built there. Why does the corrected prior change many more predictions after temperature scaling?

11. code **Missing pixels.** Hide a random half of the pixels of every test image with a Boolean mask `M` of the shape of `X_test` that is `True` for the visible pixels.
    1. Show that under the model, marginalizing a hidden pixel out of $p(\mathbf{x} \mid y)$ removes its factor, so that the score of the visible pixels is $\log p(y) + \sum_{i \text{ visible}} \log p(x_i \mid y)$. Implement it with two matrix products, `(X_test * M) @ np.log(P_xy).T + ((1 - X_test) * M) @ np.log(1 - P_xy).T` plus the log prior, and compare the accuracy with the 71% of the full images.
    2. Instead fill the hidden pixels with 0, with 1, or with the mean of the training images, and score with the affine form. Explain each accuracy from Equation 36.143; what does filling with 0 add to the score of every class?
    3. Score softmax regression from the text on the images with hidden pixels set to 0. Which operation does a discriminative model lack that the generative model used in (a)?
12. [extended] **Generative versus discriminative learning.** ⊠⊠ Ng and Jordan (2002) compare naive Bayes with logistic regression, which by exercise 2 fits the same family of posteriors by maximizing the conditional likelihood. They prove that the naive Bayes parameters approach their population values after a number of examples logarithmic in d. For logistic regression, the number of examples needed to approach its asymptotic error is at most linear in d, and in the worst case it is also at least linear in d. On many of their 15 datasets, naive Bayes has the lower test error for small training sets and logistic regression for large ones.
    1. Let each class have at least m training examples. Use Hoeffding's inequality (Section 36.9) and a union bound to show that all $2d$ unsmoothed estimates $\widehat{\theta}_{iy}$ of a two-class model lie within ε of their population values with probability at least $1 - \delta$ when
    $$m \geq \frac{\log(4d/\delta)}{2\varepsilon^2}. \tag{36.147}$$
    Evaluate the bound for $d = 784$, $\varepsilon = 0.05$ and $\delta = 0.05$.
    2. A uniform bound on $|\widehat{\theta}_{iy} - \theta_{iy}|$ does not by itself bound the change of the class scores. Why not, for the pixels with θ_{iy} close to 0 or 1, such as the corner pixel that carries the largest pullover-coat weight in the text? What must be assumed about the θ_{iy} for the scores to converge at the rate of the parameters?
    3. Restrict the data to pullovers and coats. For the total training-set sizes $n \in \{10, 20, 50, 100, 200, 500, 1000, 2000, 5000\}$ and for all $12{,}000$ images, draw random training subsets that contain both classes, 20 for each size up to $n = 200$ and 5 for each larger size. Fit naive Bayes with Laplace smoothing, and logistic regression without weight decay. For small n the training data are linearly separable, and by exercise 2c the logistic loss has no minimizer, so run a fixed number

of full-batch gradient steps from zero initialization. Plot the average test errors of both classifiers against n on a logarithmic axis with `d2l.plot`. In which range of n, if any, does naive Bayes have the lower error?

4. Repeat (c) with synthetic data for which the naive assumption holds: draw balanced labels and sample each pixel independently from Bernoulli(θ_{iy}), with the θ_{iy} estimated from all training images of the two classes. How do the two learning curves and their asymptotic errors change?
5. Theorem 4 of Ng and Jordan (2002) bounds the excess error of naive Bayes by the probability that the score gap of the population model falls near zero. Use the score-gap variances of exercise 9 to compare this probability for real pixels and for the synthetic data of (d). How does this comparison, together with the asymptotic errors of (c) and (d), account for the two pairs of learning curves?

36.12 Further Reading

The introduction of this chapter (Chapter 36) lists books and courses for what the chapter teaches. This section is about what it leaves out: each entry, in the chapter's order, starts from a limit of something the chapter did. Two are of immediate practical use, conformal prediction and confidence sequences, which repair the intervals and tests of Section 36.5 under the conditions of real experiments. The VC dimension is mainly of theoretical interest, and whether gradient noise is heavy-tailed is still disputed. Nothing else in the book depends on this section.

36.12.1 Probability Models

Three topics continue the sections on distributions: a kind of distribution that the chapter's classification misses, the limit of sums that have no variance, and a known flaw of the softmax as a model of choice.

36.12.1.1 Distributions Without Atoms or Densities

Section 36.1 notes that a variable with a density has no atoms, and treats every variable as discrete, as having a density, or as a mixture of the two. The converse fails. Write a number in base 3 using only the digits 0 and 2, each chosen by a fair coin. Its c.d.f. is continuous, so it has no atoms, yet all of its probability lies on the Cantor set, which has length zero, so no density describes it. The Lebesgue decomposition says that every distribution on the line is a mixture of a discrete part, a part with a density, and such a singular part. "Has a density" means *absolutely continuous* with respect to length, and the Radon–Nikodym theorem then supplies the density. Billingsley (1995) builds probability on this footing, in which an expectation is one integral against the distribution and the sum and the integral of the chapter are two cases of it.

36.12.1.2 Heavy Tails and Stable Laws

The average of Cauchy draws is again Cauchy, and averages of draws with a power-law tail settle more slowly than $1/\sqrt{n}$ (exercise 6). What do they converge to? If $P(|X| > x)$ falls like $x^{-\alpha}$ with $0 < \alpha < 2$, the variance is infinite, and suitably centered sums, scaled by $n^{1/\alpha}$ instead of $\sqrt{n}$, converge to an α-*stable* law rather than to a Gaussian. The Cauchy is the symmetric stable law with $\alpha = 1$ and the Gaussian the one with $\alpha = 2$. Gnedenko

and Kolmogorov (1954) give the classical theory of these limits. In deep learning, heavy tails have been reported in minibatch gradient noise, modeled as α-stable by Simsekli et al. (2019), and in the eigenvalue spectra of trained weight matrices (Martin and Mahoney 2021). Each such claim rests on an estimate of the tail index, which depends strongly on how much of the tail it uses.

36.12.1.3 Choice Models and Irrelevant Alternatives

Section 36.4.4 derives the softmax from Gumbel noise added to the scores of the options. The ratio $p_i/p_j = e^{z_i - z_j}$ then does not depend on any other option, which fails for near-duplicates. A commuter who splits evenly between a car and a red bus should still take the car half the time when an identical blue bus is added, but the softmax gives each of the three options $\frac{1}{3}$. Letting the noise of similar options be correlated (nested logit, or a probit with correlated Gaussian noise) keeps the car near $\frac{1}{2}$, but the choice probabilities then have no closed form. McFadden (1974) derived the logit from Gumbel noise and discussed this property, Luce's *independence of irrelevant alternatives*, as its main restriction.

36.12.2 Estimation and Testing

The intervals and tests of Section 36.5 leaned on a Gaussian approximation, on a model for the errors, or on a sample size fixed in advance. Each topic here drops one of the three.

36.12.2.1 Better Bootstrap Intervals

The percentile interval of Section 36.5 inherits the bias and skew of the replicate distribution, and for small samples or skewed statistics it covers the parameter less often than it claims. The *studentized* bootstrap resamples the pivot $(\hat{\theta}^* - \hat{\theta})/\widehat{\mathrm{se}}^*$ instead of $\hat{\theta}^*$, and the *BCa* interval shifts the percentiles to correct for the bias and for how the standard error changes with θ. For a one-sided interval both reduce the coverage error from order $1/\sqrt{n}$ to order $1/n$ (Hall 1992). An accuracy on thousands of test examples does not need them; a small evaluation set or a ratio of two error rates does. DiCiccio and Efron (1996) review these intervals.

36.12.2.2 Conformal Prediction

A trained model gives a prediction, but no interval with a guaranteed coverage unless its errors follow a known model. *Split conformal prediction* scores n held-out calibration points by how badly the model fits them, for example by $|y - f(\mathbf{x})|$, and returns for a new input all labels whose score is at most the $\lceil (n+1)(1-\alpha) \rceil$-th smallest calibration score. If the calibration and test points are exchangeable, the set contains the true label with probability at least $1 - \alpha$, for any model and any data distribution. The argument is the one behind the permutation test: under exchangeability the rank of the test score among all $n+1$ scores is uniform. The guarantee holds on average over inputs, not for each input separately. Vovk et al. (2022) develop the theory, and Angelopoulos and Bates (2023) give a tutorial with code for classifiers and regression models.

36.12.2.3 Confidence Sequences

The z-test of Section 36.5 is valid at a sample size fixed in advance. Checking a running comparison after every batch and stopping at the first $p < 0.05$ makes a false alarm certain if the

checks go on long enough, and exercise 14 builds a test that stays valid however often it is inspected, from a Gaussian mixture martingale. *Confidence sequences* extend this to intervals that hold simultaneously at every sample size, for any bounded or sub-Gaussian data rather than one known model. They are wider than a fixed-n interval, but for the best boundaries only by a factor that grows like $\sqrt{\log \log n}$. Howard et al. (2021) construct them from non-negative supermartingales, and Johari et al. (2017) describe the always-valid p-values that a commercial A/B testing platform adopted.

36.12.3 Computing with Posteriors

Section 36.8 had one Markov chain, the random walk, and a posterior with two parameters. These topics are the samplers used in practice and what can be done for a network.

36.12.3.1 Gibbs Sampling and Hamiltonian Monte Carlo

Random-walk Metropolis (Section 36.8.3) needs a proposal scale for every direction and moves a short distance per step, so strongly correlated or high-dimensional posteriors take many steps per independent draw. *Gibbs sampling* updates one coordinate at a time from its exact conditional distribution, so every proposal is accepted and no scale is tuned; it needs conditionals that can be sampled, and it is slow when coordinates are strongly correlated (Geman and Geman 1984). *Hamiltonian Monte Carlo* treats $-\log \tilde{p}$ as a potential energy, adds a random momentum, and simulates the resulting motion for many steps with the gradient. A proposal can then travel across the posterior and still be accepted, because the simulation nearly conserves energy. Neal (2011) explains the method and its tuning, and Hoffman and Gelman (2014) choose the trajectory length automatically with the No-U-Turn Sampler, the default sampler of Stan.

36.12.3.2 Posteriors of Neural Networks

Every method of Section 36.8 evaluates the full log-joint at each step, and the Laplace approximation stores a $d \times d$ Hessian; a network has millions of examples and parameters. *Stochastic-gradient Langevin dynamics* takes the unadjusted Langevin step of exercise 12 with a minibatch gradient and a decreasing step size, giving up the Metropolis correction for a cost per step equal to that of SGD (Welling and Teh 2011). Laplace approximations scale by keeping only part of the curvature, a Kronecker-factored block per layer or the Hessian of the last layer alone; Daxberger et al. (2021) report that these are competitive with more expensive approximations at a small fraction of their cost.

36.12.4 Concentration and Learning Theory

The first two topics extend the tools of Section 36.9 and Section 36.10, to functions that are not averages and to older measures of a class. The last two take up the question on which the chapter's theory ended: guarantees for the solution an algorithm reaches, not for the worst member of its class.

36.12.4.1 Bounded Differences and Lipschitz Functions

The Chernoff method of Section 36.9 needs a sum of independent terms. Many quantities, such as the largest gap between risk and empirical risk over a class, are functions of independent inputs but not sums. *McDiarmid's inequality* replaces "each term lies in an interval of length c_i" by "changing input i moves the function by at most c_i" and keeps the tail

$e^{-2t^2/\sum_i c_i^2}$; its proof replaces the product of moment generating functions by a martingale. For a standard Gaussian vector, every L-Lipschitz function f satisfies $P(|f - \mathbb{E}f| \geq t) \leq 2e^{-t^2/(2L^2)}$ in any dimension. The norm is the case $L = 1$, which is why its fluctuation in the thin shell does not grow with d. Boucheron et al. (2013) develop both results and the methods behind them.

36.12.4.2 VC Dimension and Covering Numbers

Section 36.10 bounds infinite classes by their Rademacher complexity. The older route counts labelings. If a binary class cannot realize all 2^{d+1} labelings of any $d + 1$ points (halfspaces in $\mathbb{R}^m$ have $d = m + 1$), Sauer's lemma says that it realizes at most $(en/d)^d$ labelings of $n \geq d$ points, and the union bound over labelings gives a complexity of order $\sqrt{d \log n / n}$; Vapnik and Chervonenkis (1971) introduced this argument. For real-valued classes one covers the class at each scale ϵ by finitely many functions instead, and chaining sums the scales and removes the $\log n$ (Vershynin 2018).

36.12.4.3 PAC-Bayes Bounds

Every bound of Section 36.10 covers the worst function in a class fixed before the data. A *PAC-Bayes* bound covers a distribution Q over functions chosen after the data, against a prior P fixed before, and the class complexity is replaced by $D_{\mathrm{KL}}(Q\|P)/n$ under a square root (McAllester 1999). A posterior that stays close to the prior is cheap even if the class is huge. Dziugaite and Roy (2017) optimized a Gaussian posterior around the weights found by SGD and obtained nonvacuous bounds for stochastic networks with far more parameters than training examples, on a binary MNIST task.

36.12.4.4 Algorithmic Stability

The random-label experiment (Section 4.5.1.1) shows that the class a network can represent is far larger than what SGD reaches on real data. Stability bounds the *algorithm* instead. If replacing one training example changes the loss of the output at any point by at most β, the expected generalization gap is of order β, and McDiarmid's inequality turns this into a high-probability bound (Bousquet and Elisseeff 2002). For convex losses that are smooth and Lipschitz, SGD is stable with β proportional to the sum of its step sizes divided by n, so the bound is smaller for fewer or smaller steps (Hardt et al. 2016).

36.12.5 Classifiers from Counts

Two topics continue Section 36.11: probabilities that one temperature cannot repair, and a second thing that a generative classifier can do and a direct fit cannot, which is to learn from inputs without labels.

36.12.5.1 Calibration Beyond One Temperature

Temperature scaling (exercise 10) assumes that the scores are right up to a scale. *Platt scaling* fits a slope and an offset of a logistic map from a binary score to a probability (Platt 1999), and isotonic regression fits any monotone map; Zadrozny and Elkan (2002) used it to calibrate naive Bayes and other classifiers. A more flexible map needs more held-out data, and for modern networks a single temperature is often as good (C. Guo et al. 2017).

36.12.5.2 Semi-Supervised Naive Bayes

Labels are scarce and unlabeled inputs are not. Because naive Bayes models how each class produces its inputs, an unlabeled example enters the likelihood with its class as a latent variable. The EM algorithm of Section 36.7 then alternates between class posteriors for the unlabeled examples and counting with those posteriors as fractional labels. Nigam et al. (2000) showed large gains for text classification with few labeled documents, and that unlabeled data can hurt when the generative model fits the data poorly.

Chapter 37

Information Theory and Divergences

Most models in this book were trained by minimizing a cross-entropy. We chose it because it works: it turns "give high probability to what actually happened" into a number that gradient descent can push down. We never asked what that number means. When a language model reports a loss of 1.2 nats per character, is that good? How low could it go? And when we compare two distributions in other ways, with a KL penalty in a variational autoencoder or with a discriminator in a generative adversarial network, what exactly are we measuring?

Information theory answers these questions, although it began with a different one. In 1948 Claude Shannon asked how efficiently a message can be sent over a telephone line, and found that the answer depends on a single property of the source: how predictable it is. His measure of unpredictability, *entropy*, is the average surprise of an outcome. It is also the floor below which no model's cross-entropy can fall.

The main text states this floor without proof (Section 3.1.2.3). This chapter proves it, and three of its results are worth carrying back there. A cross-entropy exceeds its floor by the *KL divergence* between the data and the model, which is zero only for the right model. Count a model's total loss on a message in bits, and the message can be stored in that many bits plus at most two, so the loss of a language model (Section 8.3.2) is a file size. And a discriminator reports only a lower bound on how far apart two distributions are: to a weak one, distant distributions look close. Adversarial training (Section 16.2) rests on such bounds, and so does contrastive learning, where picking the true partner of an example among N candidates certifies at most $\log N$ nats of shared information.

The first of the chapter's three parts asks what a cross-entropy measures (Section 37.1) and why it is the length of a file (Section 37.2). The second begins with a defect. The KL divergence is infinite as soon as a model gives zero probability to something that occurs. If the model's samples miss the data altogether, no other function of the ratio of the two densities can tell a near miss from a distant one (Section 37.3). Comparing averages of a function under the two distributions can, and so can the cost of moving probability mass between them (Section 37.4). A model whose density is known only up to a constant factor is compared through the slope of its log-density (Section 37.5).

The third part turns to two variables: how much one tells about the other (Section 37.6),

which is what the KL penalty of an autoencoder limits, and how much of that samples can show (Section 37.7). It uses nothing from the two sections just before it, which can wait until Chapter 16 or Chapter 17 sends you there.

We assume Jensen's inequality (Chapter 35), conditional distributions and maximum likelihood (Chapter 36). Entropy and the KL divergence are developed from the beginning. Practical error-correcting codes, Kolmogorov complexity and information geometry are left to Section 37.8.

Resources and Further Reading

The resources below accompany the sections of this chapter; each names the section it goes with. What the chapter does not teach, from calibration to compressors that need no model, is collected in Section 37.8.

Books

- Elements of Information Theory — Cover & Thomas — the graduate reference that develops entropy, the KL divergence (which it calls relative entropy), mutual information and their chain rules from first principles, with chapters on source coding, channel capacity and rate–distortion; it accompanies Section 37.1, Section 37.2 and Section 37.6.
- Information Theory, Inference, and Learning Algorithms — David MacKay — free to read online; treats information theory and machine learning together, and its chapters on symbol codes and stream codes (including arithmetic coding) accompany Section 37.2.
- Information Theory: From Coding to Learning — Polyanskiy & Wu — a modern graduate text oriented toward statistics and learning, whose treatment of f-divergences, their variational forms and data processing accompanies Section 37.3 and Section 37.6; the lecture notes of MIT 6.441, from which the book grew, are free.
- Computational Optimal Transport — Peyré & Cuturi — Wasserstein distances, entropic regularization and Sinkhorn iterations for data science, the transport part of Section 37.4; the full text is on arXiv.
- The Minimum Description Length Principle — Peter Grünwald — develops learning as data compression through two-part and universal codes, the subject of Section 37.2.2.3; the core ideas also appear in a freely available tutorial.

Courses, tutorials and software

- Stanford EE376A Information Theory — Tsachy Weissman — an undergraduate-to-graduate course with public, student-scribed lecture notes on entropy, prefix codes and the typical set, for Section 37.1 and Section 37.2.
- Visual Information Theory — Christopher Olah — a visual introduction that reaches entropy and cross-entropy through the lengths of codewords and ends with mutual information, for Section 37.1, Section 37.2 and Section 37.6.
- From GAN to WGAN — Lilian Weng — compares the KL, Jensen–Shannon and Wasserstein losses of generative models, the comparison of Section 37.3 and Section 37.4.
- POT: Python Optimal Transport — Flamary et al. — a library with solvers for the linear program of transport and for its entropic regularization by Sinkhorn iterations, for computing the distances of Section 37.4.2 on your own data.
- Generative Modeling by Estimating Gradients of the Data Distribution — Yang Song — a tutorial post on why a score needs no normalizing constant and how score matching

fits one to data, the material of Section 37.5, before it goes on to sampling and diffusion models.

Foundational papers

- Language Modeling Is Compression — Delétang et al. — drives an arithmetic coder with a language model's next-token probabilities, the construction of Section 37.2.2.2.
- f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization — Nowozin et al. — shows that every f-divergence yields an adversarial objective through its Fenchel lower bound, the bound of Section 37.3.4.
- A Kernel Two-Sample Test — Gretton et al. — defines the maximum mean discrepancy, its unbiased estimate from two samples and the test built on it, the comparison of Section 37.4.1.
- Wasserstein GAN — Arjovsky et al. — replaces the Jensen–Shannon divergence by the Wasserstein-1 distance, which stays informative when the supports do not overlap; it accompanies Section 37.4.2.
- Estimation of Non-Normalized Statistical Models by Score Matching — Aapo Hyvärinen — fits a model whose normalizing constant is unknown by matching the slope of its log-density to that of the data, and removes the unknown slope of the data by an integration by parts, the two steps of Section 37.5.
- A Kernelized Stein Discrepancy for Goodness-of-fit Tests — Liu, Lee & Jordan — tests a sample against a model that is known only through its score, with a statistic computed from kernel evaluations on pairs of data points, the test of Section 37.5.3.
- The Information Bottleneck Method — Tishby, Pereira & Bialek — the original formulation of compressing X while keeping information about Y, the problem of Section 37.6.4.
- Estimating Mutual Information — Kraskov, Stögbauer & Grassberger — estimates the mutual information of two continuous variables from the distances between each sample and its nearest neighbors, the estimator of Section 37.7.2.2; scikit-learn's `mutual_info_regression` is built on it.
- On Variational Bounds of Mutual Information — Poole et al. — derives the Barber–Agakov, Donsker–Varadhan, NWJ and InfoNCE bounds in one framework and compares their bias and variance, the material of Section 37.7.3.
- Representation Learning with Contrastive Predictive Coding — van den Oord et al. — introduces the InfoNCE loss as a lower bound on mutual information for self-supervised learning, the bound of Section 37.7.4.

37.1 Entropy, Cross-Entropy, and KL Divergence

A loss curve is the first thing we look at when a classifier trains, and we have learned to read its shape: a falling curve is good news, a flat one is suspicious. The numbers on its axis are another matter. So far we can explain one of them. A freshly initialized classifier with ten classes reports a loss of about 2.30, and Section 4.2 used this value to check that an untrained network is set up correctly. The value is $\ln 10$, the cost of naming one of ten labels when the model still finds all ten equally likely. The other end of the curve raises harder questions. Is a final loss of 0.6 good? Is a plateau a bug or a property of the data? If some of the labels are wrong, how low can any model go?

The loss is the average of $-\log q(y)$, the model's surprise at the label y that occurred. That average splits into two parts. The first is the average surprise of someone who knows the true distribution of the labels, the *entropy*. The data fix it, and no model can reduce it. The

second is the extra surprise of using the wrong distribution, the *Kullback–Leibler (KL) divergence*. It is never negative, and it is zero only for the right model. A loss can therefore be judged only against its floor.

Two facts are already in hand from the previous chapter: maximum likelihood minimizes a cross-entropy (Section 36.6.2.1), and the KL divergence is never negative (Section 36.4.2), which is Gibbs' inequality. There the inequality was a tool. Here it is the subject. We start from a deck of cards and a source with four symbols, where entropy counts yes/no questions. We then compare two distributions, and return to the classifier to find its floor. Finally we ask what changes when the targets are softened and when the outcomes are real numbers.

The logarithm base is a choice of unit: base 2 gives *bits*, base e gives *nats*. **We work in nats throughout**, as the library `log` does and as training losses are reported. Bits suit yes/no questions, and a subscript 2 marks a quantity in bits, as in H_2 for an entropy. One bit is $\ln 2 \approx 0.693$ nats. Table 37.1 collects the symbols of the chapter.

Table 37.1: Symbols of the information theory chapter. The binary entropy $H_b(p)$ is a function of a number, not of a random variable. The mutual information $I(X;Y)$ and the last five rows are defined in later sections.

Symbol	Meaning	Unit
$I(x) = -\log p(x)$	self-information (surprisal) of the outcome x (by convention the letter is shared with the mutual information $I(X;Y)$)	nats
$H(X)$, $H(P)$	entropy of a discrete random variable or distribution; a subscript 2, as in H_2 and CE_2, marks bits	nats
$H_b(p)$	binary entropy of a coin with heads probability p	nats
$h(X)$	differential entropy of a continuous random variable	nats, depends on the units of X
$D_{\mathrm{KL}}(P\|Q)$	KL divergence from P to Q	nats
$\mathrm{CE}(P,Q)$	cross-entropy from P to Q	nats
k	number of classes (size of the set of outcomes)	
n	number of examples, or of symbols in a message; the dimension in $\mathbb{R}^n$	
$\mathbf{p}$, $\mathbf{q}$	probability vectors on k classes (truth, model)	
$\mathbf{z}$	logits, with $\mathbf{q} = \mathrm{softmax}(\mathbf{z})$	
ϵ	label-smoothing mass (in later sections, a noise variable)	
τ	temperature	

Symbol	Meaning	Unit
ρ	label-noise rate: the probability that a recorded label is not the clean class	
$H(X \mid Y)$, $I(X;Y)$	conditional entropy, mutual information	nats
$D_f(P\|Q)$	f-divergence with divergence function f	
T	critic: a function whose averages under two distributions bound a divergence	
g	test function of an integral probability metric	
γ	coupling (transport plan) of two distributions	
N	number of points in each of two samples that are compared; number of candidates in a contrastive batch	

37.1.1 Information and Entropy

37.1.1.1 Self-Information

Consider reports about a shuffled deck of cards. "I see a card" conveys nothing, because the event is certain. "I see a heart" picks one of four equally likely suits, "It is the three of spades" one of 52 equally likely cards, and the order of the whole deck one of $52!$ orderings. The less probable an event, the more we learn by observing it. Shannon (Shannon 1948) measured this by the *self-information* of an outcome x with probability $p(x)$,

$$I(x) = -\log p(x).$$

It is also called the *surprisal* of the outcome, the name used in Section 3.1.2.3: the less we expected an outcome, the more it surprises us. The surprise belongs to whoever assigned the probability, so a confidently wrong model finds ordinary events surprising. A certain event carries no information, and halving the probability of an outcome adds $\ln 2$ nats, one bit, to its surprisal. The first three card reports carry 0, $\ln 4 \approx 1.39$ and $\ln 52 \approx 3.95$ nats. In bits, where one bit is the information of one fair coin flip, they carry 0, 2 and 5.70.

Why a logarithm? We want three properties: a certain event carries zero information, rarer events carry more, and the information of two independent events adds up. Since independent probabilities multiply, the third property says $I(p_1 p_2) = I(p_1) + I(p_2)$, and the only decreasing functions with this property are multiples of $-\log p$ (Csiszár (2008); Exercise 1 gives the argument). The choice of multiple is the choice of unit.

Figure 37.1 plots the curve: decreasing, convex, zero at $p = 1$, and diverging as $p \to 0$. The fourth card report, the order of the deck, lies far out on it, at $\ln 52!$. We compute this

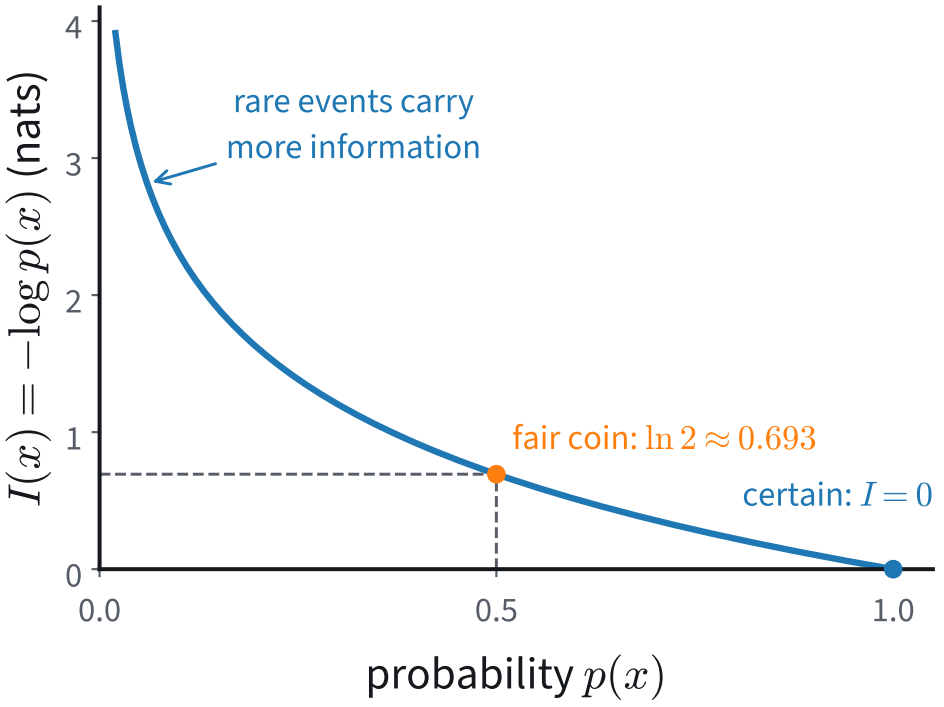


Figure 37.1: Self-information $I(x) = -\log p(x)$ in nats as a function of the probability p. Certain events ($p = 1$) carry zero information, the fair coin ($p = \frac{1}{2}$) carries $\ln 2 \approx 0.693$ nats, and the curve diverges as $p \to 0$.

value through the log-gamma function, $\ln 52! = \ln \Gamma(53)$, without ever forming the 68-digit number $52!$.

```
deck_order = math.lgamma(53)          # ln 52! without forming 52!
print(f'the order of the deck: {deck_order:.2f} nats '
      f'= {deck_order / math.log(2):.1f} bits')
```

```
the order of the deck: 156.36 nats = 225.6 bits
```

Naming the order of a shuffled deck carries 156 nats, as much information as 226 fair coin flips.

37.1.1.2 Entropy

A surprisal is known only once the outcome has arrived. Before that we can say how surprised we expect to be, by averaging the surprisal over the outcomes. Let X be a discrete random variable that takes the value x_i with probability p_i. Its *entropy* is the expected self-information of its outcome,

$$H(X) = -\mathbb{E}_{x\sim P}[\log p(x)] = -\sum_i p_i \log p_i, \tag{37.1}$$

with the convention $0 \log 0 = 0$, justified by continuity ($t \log t \to 0$ as $t \to 0$): an impossible outcome contributes nothing. Since the entropy depends only on the distribution P of X, we write $H(P)$ as well as $H(X)$.

On a finite set, entropy in bits has a concrete meaning. It counts the yes/no questions that a good questioner needs, on average, to identify the outcome. Take the distribution $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ on four symbols a, b, c, d. We carry it through the section and call it the *dyadic source*, since all its probabilities are powers of $\frac{1}{2}$. A good questioner first asks "Is it a?", which settles the matter half the time. If not, they ask "Is it b?", and then "Is it c?", where "no" means d. The expected number of questions is

$$\tfrac{1}{2}\cdot 1 + \tfrac{1}{4}\cdot 2 + \tfrac{1}{8}\cdot 3 + \tfrac{1}{8}\cdot 3 = 1.75,$$

and the entropy in bits is the same number, $H_2(P) = \frac{1}{2}\log_2 2 + \frac{1}{4}\log_2 4 + 2\cdot\frac{1}{8}\log_2 8 = 1.75$ bits, or 1.213 nats. Each outcome is identified with $\log_2(1/p_i)$ questions, its self-information in bits.

A questioner who ignores the probabilities asks "Is it a or b?" and then "Which of the two?", and always needs 2 questions. A questioning strategy is a binary code in disguise: write 1 for yes and 0 for no, and the answers that identify an outcome form its codeword, here 1, 01, 001 and 000 for a, b, c and d. The count of questions matched the entropy here because every probability is a power of $\frac{1}{2}$. In the language of codes, Section 37.2 proves that the count is never far off: for every distribution on a finite set, the best strategy needs between H_2 and $H_2 + 1$ questions on average.

In code, the function `entropy` masks the zero probabilities, which implements $0 \log 0 = 0$. We use it for the rest of the section, first on the dyadic source.

```python
def entropy(p):
    """Entropy of a probability vector, in nats (0 log 0 = 0 by masking)."""
    p = np.asarray(p, dtype=float)
    p = p[p > 0]
    return float(-np.dot(p, np.log(p)))

p_dyadic = np.array([1 / 2, 1 / 4, 1 / 8, 1 / 8])
h = entropy(p_dyadic)
print(f'H = {h:.3f} nats = {h / math.log(2):.2f} bits')
```

```
H = 1.213 nats = 1.75 bits
```

It returns the 1.75 bits that we counted in questions.

Every term $-p_i \log p_i$ is non-negative because $0 \le p_i \le 1$, so $H(X) \ge 0$, with equality only for a point mass, a variable whose value is certain. The entropy also depends only on the list of probabilities, not on the values or names of the outcomes. A variable that takes the values 1, 100, 10^6, and 10^9 with the dyadic probabilities also has 1.75 bits.

37.1.1.3 Joint and Conditional Entropy

A label comes with an input, and a character with the text before it. We therefore need the entropy of two variables together, and the entropy left in one of them once the other is known. The first needs nothing new. A pair (X, Y) with joint mass function $p(x, y)$ is a single random variable whose outcomes are pairs, and the entropy of the pair is the *joint entropy* $H(X, Y) = -\mathbb{E}[\log p(X, Y)]$. The *conditional entropy* is the surprisal left in Y once X is known, averaged over both,

$$H(Y \mid X) = -\mathbb{E}[\log p(Y \mid X)] = \sum_x p(x)\, H(Y \mid X = x),$$

where $H(Y \mid X = x)$ is the entropy of the conditional distribution $p(\cdot \mid x)$. Since $\log p(x, y) = \log p(x) + \log p(y \mid x)$, taking expectations gives the *chain rule*

$$H(X, Y) = H(X) + H(Y \mid X). \tag{37.2}$$

The entropy of a pair is the entropy of the first variable plus what the second adds once the first is known. Applied repeatedly, the rule reads $H(X_1, \dots, X_n) = \sum_{i=1}^{n} H(X_i \mid X_1, \dots, X_{i-1})$: the entropy of a sequence is the sum of the entropies of its symbols, each given the symbols before it. A language model predicts text in this order, one symbol at a time.

The questions game is an instance. Let X be the answer to the first question, "Is it a?", and let Y be the outcome of the dyadic source. The answer is a fair coin, so $H_2(X) = 1$ bit. After "yes" nothing is left to ask. After "no" the outcome is b, c or d with probabilities $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$, a distribution with 1.5 bits, so $H_2(Y \mid X) = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 1.5 = 0.75$ bits. The chain rule gives $H_2(X, Y) = 1 + 0.75 = 1.75$ bits: the first answer supplies one bit and the later questions the rest. The sum is the entropy of the outcome alone, as it has to be. The outcome determines the answer, so $H(X \mid Y) = 0$, and the chain rule in the other order reads $H(X, Y) = H(Y) + 0$.

37.1.1.4 Maximum Entropy on a Finite Set

How large can the entropy of a variable with k outcomes be? We start with the smallest case, a coin. A Bernoulli variable with heads probability p has the *binary entropy*

$$H_b(p) = -p \log p - (1-p) \log(1-p),$$

a function of the number p rather than of a random variable. Figure 37.2 plots it. The entropy is zero at $p \in \{0, 1\}$, where the outcome is certain, and largest at the fair coin, $H_b(\frac{1}{2}) = \ln 2 \approx 0.693$ nats, one bit. A coin with $p = 0.1$ has $H_b(0.1) \approx 0.325$ nats, because its outcome is easy to guess.

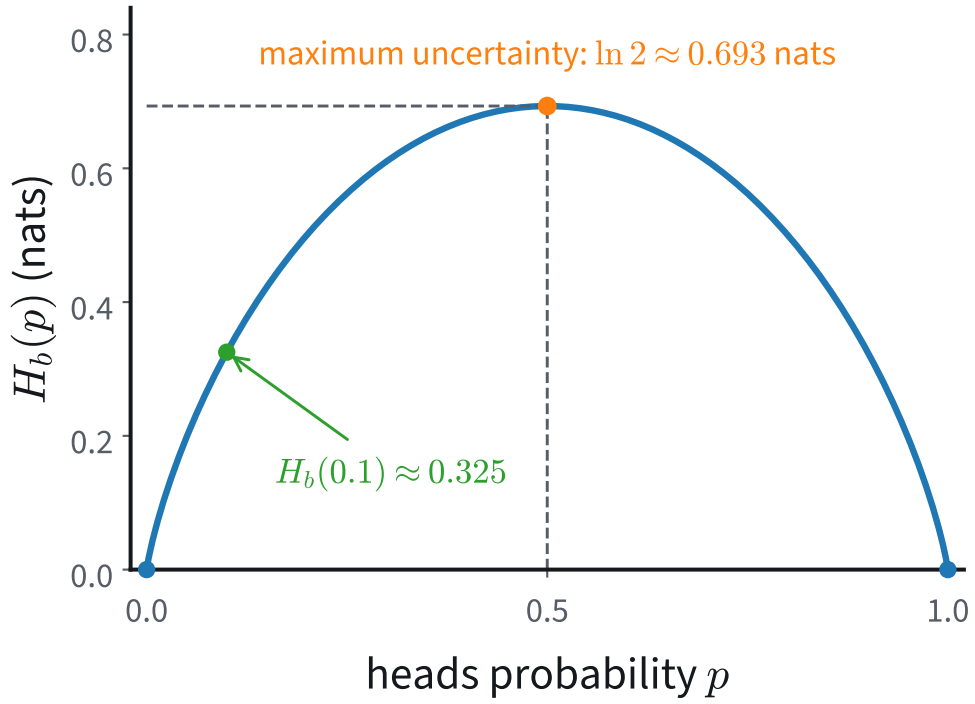


Figure 37.2: The binary entropy $H_b(p)$ in nats, symmetric about $p = \frac{1}{2}$, where it peaks at $\ln 2 \approx 0.693$ nats. At the deterministic endpoints $p = 0$ and $p = 1$ the outcome carries no surprise and the entropy vanishes. The curve is concave: any chord lies below it.

The curve is concave: the chord joining any two of its points lies below it. It peaks at the fair coin, and the same holds for any number of outcomes. If X takes at most k values, then

$$H(X) \leq \log k,$$

with equality if and only if X is uniform on k values. The uniform distribution hedges the most, so it is the hardest to guess. For four outcomes the bound is $\log_2 4 = 2$ bits, 0.25 bits above the 1.75 bits of the dyadic source. We postpone the proof until Gibbs' inequality is available, where it takes one line. That inequality compares two distributions, and so we turn to the comparison first.

37.1.2 KL Divergence and Cross-Entropy

37.1.2.1 The Kullback–Leibler Divergence

We now want to measure how badly a model Q misrepresents a truth P. Information theory answers with the *Kullback–Leibler (KL) divergence*, also called *relative entropy* (Kullback and Leibler 1951): the average extra surprisal of someone who believes Q when the outcomes are drawn from P. In the questions game, the KL divergence in bits is the average number of extra questions that a strategy designed for Q needs on outcomes drawn from P, up to rounding to whole questions. Let P and Q have mass functions (or densities) p and q. The KL divergence from P to Q is

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{x\sim P}\left[\log \frac{p(x)}{q(x)}\right]. \tag{37.3}$$

The term inside the expectation is $\log p(x) - \log q(x) = (-\log q(x)) - (-\log p(x))$: the surprisal of the believer in Q minus the surprisal of the believer in P. It is positive where Q underestimates the outcome ($q(x) < p(x)$) and negative where Q overestimates it. KL averages this difference over outcomes drawn from the truth.

The definition is asymmetric. P supplies both the samples and the numerator, so in general $D_{\mathrm{KL}}(P\|Q) \neq D_{\mathrm{KL}}(Q\|P)$. Outcomes with $p(x) = 0$ contribute nothing, by the convention $0\log(0/q) = 0$. If some outcome has $p(x) > 0$ but $q(x) = 0$, however, then $D_{\mathrm{KL}}(P\|Q) = +\infty$.

A small pair makes the definition concrete, and we carry it through the section. Let $P = (0.6, 0.3, 0.1)$ be the truth and $Q = (0.2, 0.5, 0.3)$ a model on three outcomes. Then

$$D_{\mathrm{KL}}(P\|Q) = 0.6\ln\frac{0.6}{0.2} + 0.3\ln\frac{0.3}{0.5} + 0.1\ln\frac{0.1}{0.3} = 0.659 - 0.153 - 0.110 = 0.396 \text{ nats},$$

and the reverse direction gives $D_{\mathrm{KL}}(Q\|P) = 0.365$ nats. In bits the two directions are 0.571 and 0.527. They differ, and the first says that a questioner who believes Q asks 0.57 more questions per outcome than necessary, on average over long runs of outcomes drawn from P.

Here is the discrete case in code, with $0\log 0 = 0$ implemented by masking the zero probabilities of P, as in `entropy`. We call it on the running pair in both directions, and on a model $Q' = (0.5, 0.5, 0)$ that declares the third outcome impossible.

```
def kl_divergence(p, q):
    """KL(P || Q) for two probability vectors, in nats; inf where q = 0 < p."""
    p, q = np.asarray(p, dtype=float), np.asarray(q, dtype=float)
    m = p > 0
    if np.any(q[m] == 0):
        return float('inf')
    return float(np.dot(p[m], np.log(p[m] / q[m])))

p = np.array([0.6, 0.3, 0.1])
q = np.array([0.2, 0.5, 0.3])
print(f'KL(P||Q) = {kl_divergence(p, q):.4f} nats, '
```

```
        f'KL(Q||P) = {kl_divergence(q, p):.4f} nats')
print(f"KL(P||Q') = {kl_divergence(p, [0.5, 0.5, 0.0])}")
```

```
KL(P||Q) = 0.3961 nats, KL(Q||P) = 0.3653 nats
KL(P||Q') = inf
```

The second line is the one to remember. The model Q' gives zero probability to an outcome that occurs one time in ten, and its divergence is infinite: a model that declares an event impossible is infinitely surprised when it happens.

37.1.2.2 Gibbs' Inequality

Can the extra surprisal be negative? That would mean a wrong model is less surprised, on average, than the truth itself. Gibbs' inequality (Gibbs 1902) says that it cannot: no model beats the truth on average. Everything that follows rests on it, so we prove it in full, as a statement about averaging a concave function. For a concave function f and a random variable X with finite mean, *Jensen's inequality* states that

$$\mathbb{E}[f(X)] \le f(\mathbb{E}[X]).$$

Averaging first and applying f second gives the larger value. For an X with two values, this is the chord below the curve of Figure 37.2. For strictly concave f, equality holds only when X is constant with probability one. Section 35.3.3.1 proves the general case. We need it for one function, the logarithm, which is strictly concave: the logarithm of an average is at least the average of the logarithms.

Proposition (Gibbs' inequality). *For any distributions P and Q on the same space,*

$$D_{\mathrm{KL}}(P\|Q) \ge 0,$$

with equality if and only if $P = Q$.

Proof. If $q(x) = 0$ for some x with $p(x) > 0$, the divergence is $+\infty$ and there is nothing to prove, so assume otherwise. Write the divergence as minus the average of $\log(q(x)/p(x))$ over $x \sim P$, and apply Jensen's inequality to that logarithm:

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) &= -\mathbb{E}_{x\sim P}\left[\log\frac{q(x)}{p(x)}\right] \ge -\log \mathbb{E}_{x\sim P}\left[\frac{q(x)}{p(x)}\right] \\ &= -\log \sum_{x\,:\,p(x)>0} q(x) = -\log Q(\operatorname{supp} P) \ge 0. \end{aligned}$$

The minus sign turns Jensen's "at most" into "at least". Everything else happens in the second line. The expectation runs over the support of P, the set of outcomes with $p(x) > 0$. The factors $p(x)$ cancel, and what remains is the Q-mass of that support, a number at most 1, whose negative logarithm is non-negative. For densities, the same computation runs with integrals. For equality, both inequalities must be tight. Because the logarithm is strictly concave, the first is an equality only when $q(x)/p(x)$ equals a constant c for P-almost every x (for all x outside a set of P-probability zero), and the second requires $Q(\operatorname{supp} P) = 1$. Summing (or integrating) $q = c\,p$ over the support of P then gives $c = 1$, so $q = p$, in the case of densities again outside a set of probability zero. ■

The proof has one idea. Averaged over the truth, the ratio q/p is at most one, so its logarithm averages to at most zero, with equality only when there is nothing to average. A model incurs no extra surprisal only when it is right. Exercise 2 compares this proof with the tangent-line proof.

The inequality settles the uniform bound that we left unproved. For a distribution P on k outcomes and the uniform distribution U on the same outcomes, $D_{\mathrm{KL}}(P\|U) = \sum_i p_i \log(k\,p_i) = \log k - H(P)$, which is non-negative and zero only for $P = U$.

It also shows that conditioning cannot increase entropy. Compare the joint distribution of a pair with the product of its marginals. The log-ratio is $\log \frac{p(x,y)}{p(x)\,p(y)} = \log p(y \mid x) - \log p(y)$, and its average over the joint distribution is $H(Y) - H(Y \mid X)$. That average is a KL divergence, so

$$H(Y) - H(Y \mid X) = D_{\mathrm{KL}}(p_{X,Y} \,\|\, p_X\, p_Y) \geq 0,$$

with equality only when X and Y are independent. On average, knowing X never makes Y more uncertain. A particular observation can. Let Y be tails for certain when $X = 0$, and a fair coin when $X = 1$, which happens one time in five. Then Y shows heads with probability 0.1, and $H(Y) = H_b(0.1) \approx 0.325$ nats. Observing $X = 1$ raises the uncertainty about Y to $\ln 2 \approx 0.693$ nats, yet the average $H(Y \mid X) = 0.2 \cdot 0.693 \approx 0.139$ nats is less than half of $H(Y)$.

37.1.2.3 Cross-Entropy

Classifiers are trained with a close relative of the KL divergence. The *cross-entropy* from P to Q is the average surprisal of the believer in Q on outcomes drawn from P,

$$\mathrm{CE}(P, Q) = -\mathbb{E}_{x \sim P}[\log q(x)]. \tag{37.4}$$

Adding and subtracting $\log p(x)$ inside the expectation, which is legitimate whenever $H(P)$ is finite, splits the model's surprisal into the truth's own surprisal plus the excess:

$$\mathrm{CE}(P, Q) = H(P) + D_{\mathrm{KL}}(P\|Q). \tag{37.5}$$

The cross-entropy is an entropy floor plus a never-negative excess: by Gibbs' inequality, $\mathrm{CE}(P, Q) \geq H(P)$ with equality if and only if $Q = P$. For the running pair, $\mathrm{CE}(P, Q) = -(0.6 \ln 0.2 + 0.3 \ln 0.5 + 0.1 \ln 0.3) = 1.294$ nats and $H(P) = 0.898$ nats; their difference, 0.396 nats, is the KL divergence computed above. Since $H(P)$ does not depend on Q, minimizing the cross-entropy in Q is minimizing the KL divergence. Maximum likelihood on discrete data does just that: its average negative log-likelihood is the cross-entropy from the empirical distribution of the data to the model (Section 36.6.2.1).

Figure 37.3 draws the decomposition in the questions game, for the dyadic source and the uniform model $U = (\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$. Asking adaptively takes $H_2(P) = 1.75$ questions on average. The two fixed questions of the uniform questioner take $\mathrm{CE}_2(P, U) = 2$, and the difference, $D_{\mathrm{KL}}(P\|U) = 0.25$ bits, is the waste. Since a questioning strategy is a code, the same numbers are average codeword lengths in bits per symbol. For this pair the reverse divergence $D_{\mathrm{KL}}(U\|P)$ also happens to be 0.25 bits, which is why we showed the asymmetry of KL on the other pair.

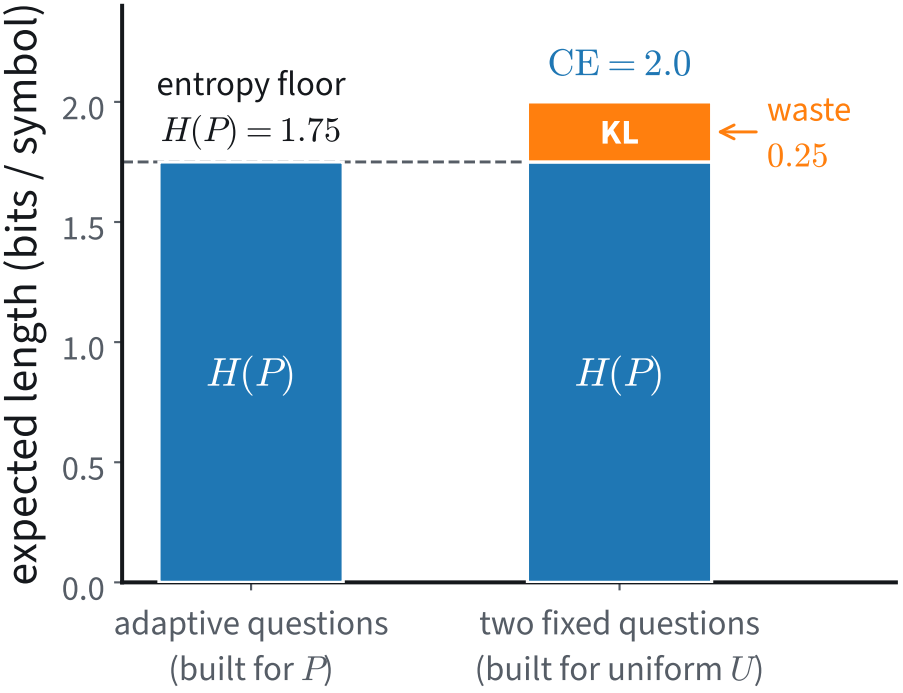


Figure 37.3: The decomposition Equation 37.5 for the dyadic source $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ and the uniform model U, in bits. Asking the questions adapted to P takes $H_2(P) = 1.75$ questions on average, asking two fixed questions takes $\mathrm{CE}_2(P, U) = 2.0$, and the waste is $D_{\mathrm{KL}}(P \| U) = 0.25$ bits.

37.1.3 The Classification Loss

In a k-class classification problem, the target for one example is the one-hot distribution that puts all its mass on the recorded label y, and the model supplies a predicted distribution $\mathbf{q} = (q_1, \dots, q_k)$. The cross-entropy Equation 37.4 between them collapses to a single term, $-\log q_y$, the model's surprisal at the label. Averaged over n examples with labels y_i and predictions $\mathbf{q}_i$, it is the *cross-entropy loss*

$$\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{n} \log q_{i,y_i},$$

in nats. A built-in loss computes this number from the *logits* $\mathbf{z}$, the unnormalized scores whose softmax is $\mathbf{q}$. It applies a stable log-softmax itself, because forming the probabilities first overflows for large logits and ends in the logarithm of zero for very negative ones (Section 3.4.2; Section 35.8.2 derives the stable form). The subsections that follow read that number.

37.1.3.1 The Irreducible Loss

The opening asked how low the loss can go when some labels are wrong. On a fixed training set the answer is disappointing. A network large enough to memorize the set can fit every recorded label, the wrong ones included, and push the training loss toward zero. The question has content only for examples that the model has not seen, and there the one-hot target is not the truth. In real data the label is not a function of the input: the same image can carry different labels, from ambiguity or from annotation errors. The truth about an input $\mathbf{x}$ is the distribution from which its label is drawn, and the recorded label is one draw from it.

Let the input $\mathbf{x}$ and the label $y \in \{1, \dots, k\}$ have a joint distribution, let $p(\cdot \mid \mathbf{x})$ be the true conditional distribution of the label, and let $q(\cdot \mid \mathbf{x})$ be any model's prediction. As the number of fresh examples grows, their average loss approaches its expectation over the pair $(\mathbf{x}, y)$. We call this expectation the *population loss*. No model can bring it below the average entropy of the label given the input.

Proposition (the irreducible classification loss). *The population cross-entropy loss of any model satisfies*

$$\mathbb{E}_{(\mathbf{x},y)}\big[-\log q(y \mid \mathbf{x})\big] = \mathbb{E}_{\mathbf{x}}\big[H(p(\cdot \mid \mathbf{x}))\big] + \mathbb{E}_{\mathbf{x}}\big[D_{\mathrm{KL}}(p(\cdot \mid \mathbf{x}) \,\|\, q(\cdot \mid \mathbf{x}))\big] \geq \mathbb{E}_{\mathbf{x}}\big[H(p(\cdot \mid \mathbf{x}))\big], \tag{37.6}$$

with equality if and only if $q(\cdot \mid \mathbf{x}) = p(\cdot \mid \mathbf{x})$ for almost every $\mathbf{x}$, that is, for all inputs outside a set of probability zero.

Proof. Condition on $\mathbf{x}$. The inner expectation over y is the cross-entropy $\mathrm{CE}(p(\cdot \mid \mathbf{x}), q(\cdot \mid \mathbf{x}))$, and the decomposition Equation 37.5 applies at each $\mathbf{x}$; the entropies are finite, at most $\log k$. Averaging over $\mathbf{x}$ gives the identity. Gibbs' inequality makes every KL term non-negative, and an average of non-negative terms is zero only when almost all of them vanish. ■

The floor is the conditional entropy of the label given the input (Section 37.1.1.3). It is a property of the data: no architecture, data volume, or training time lowers it. Of the results

of this section, it is the one to have in mind when reading a loss curve. Three cases show what it means.

- A model that predicts the uniform distribution has cross-entropy $\ln k$ at every input, whatever the labels are: the 2.30 of the opening for $k = 10$.
- If the label is a function of the input, every $H(p(\cdot \mid \mathbf{x}))$ is zero, and a perfect model reports 0.
- Under symmetric label noise, the clean class is determined by the input, and the observed label is the clean class with probability $1-\rho$ and each of the other $k-1$ classes with probability $\rho/(k-1)$. Then $H = -(1-\rho)\log(1-\rho) - \rho \log \frac{\rho}{k-1} = H_b(\rho) + \rho \log(k-1)$ at every input, which for $\rho = 0.1$ and $k = 10$ is $0.325 + 0.220 = 0.545$ nats. As long as $1 - \rho > \rho/(k-1)$, the best prediction of the class is the clean one, so the best accuracy is $1 - \rho$, here 90%. Exercise 4 treats inputs whose clean class is itself uncertain.

Before running the next cell, guess the loss of a predictor that puts 0.99 on the clean class in the noisy case. It is right nine times in ten, and nearly certain each time. The cell compares it with the uniform predictor and with the predictor that reports the true distribution, $q = p$, which is called the *Bayes predictor*.

```
k, rho = 10, 0.1
p_noisy = np.array([1 - rho] + [rho / (k - 1)] * (k - 1))  # clean class first
predictors = {'uniform': np.full(k, 1 / k),
              'confident 0.99': np.array([0.99] + [0.01 / (k - 1)] * (k - 1)),
              'Bayes q = p': p_noisy}
print(f'floor = H(p) = {entropy(p_noisy):.3f} nats')
for name, q_hat in predictors.items():
    ce = float(-np.dot(p_noisy, np.log(q_hat)))
    print(f'{name:15s} CE = {ce:.3f} nats')
```

```
floor = H(p) = 0.545 nats
uniform         CE = 2.303 nats
confident 0.99  CE = 0.689 nats
Bayes q = p     CE = 0.545 nats
```

The confident predictor reports 0.689, well above the floor: every flipped label surprises it, and its confidence loses more on the tenth of the labels that are flipped than it gains on the rest. Only the Bayes predictor reaches the floor.

The opening asked whether a final loss of 0.6 is good. On clean labels, where the floor is zero, it leaves room for a better model. On these noisy labels it lies within 0.06 nats of the floor and leaves almost none. A plateau at the floor is a property of the data, and only a plateau above it is the fault of the model or of its training. All of this holds for fresh data. There a loss below the floor cannot come from a better model, and Exercise 3 asks what it signals instead.

The floor was derived for one penalty, $-\log q_y$, for labels as the data deliver them, and for outcomes that can be counted. The rest of the section changes these one at a time: other penalties, softened targets, and outcomes that are real numbers.

37.1.3.2 Proper Scoring Rules

The floor is reached only by the true conditional probabilities, which is what we want of a loss whose outputs we read as probabilities. The property has a name. A *scoring rule* assigns a penalty $S(\mathbf{q}, y)$ for having reported the probability vector $\mathbf{q}$ when class y then occurs. The rule is *proper* if truthful reporting is optimal: when the labels follow $\mathbf{p}$, the expected penalty $\mathbb{E}_{y\sim\mathbf{p}}[S(\mathbf{q}, y)]$ is minimized by reporting $\mathbf{q} = \mathbf{p}$. It is *strictly proper* if this report is the unique optimum. The cross-entropy loss is the *log score* $S(\mathbf{q}, y) = -\log q_y$, and it is strictly proper, since its expected penalty is the cross-entropy $\mathrm{CE}(\mathbf{p}, \mathbf{q})$, which is smallest only at $\mathbf{q} = \mathbf{p}$.

At the population optimum over a model class that contains the truth, the predicted probabilities are therefore the true conditional probabilities. In particular they are *calibrated*: among the inputs that receive probability 0.8 for a class, that class occurs 80% of the time. A finite sample or a restricted model class can still leave a trained model miscalibrated.

The log score is not the only strictly proper rule. The *Brier score* $S(\mathbf{q}, y) = \sum_j (q_j - \mathbf{1}(j = y))^2$ is strictly proper too, with a bounded penalty on confident errors where the log score's penalty diverges (Gneiting and Raftery 2007). Exercise 6 computes its excess penalty and tests an improper rule.

37.1.3.3 Label Smoothing

So far the target was the one-hot label. A softer target changes what the loss demands of the logits and where its floor lies, and there is a reason to want one: one-hot targets ask for something that a softmax cannot deliver. Probability 1 on the true class needs a logit infinitely far above the others, so the loss has no minimizer, and training pushes the logits apart for as long as it runs. *Label smoothing* (Szegedy et al. 2016) replaces the one-hot target $\mathbf{e}_y$ by a mixture with a fraction ϵ of the uniform distribution over all k classes,

$$\mathbf{p}^\epsilon = (1 - \epsilon)\, \mathbf{e}_y + \frac{\epsilon}{k}\, \mathbf{1}. \tag{37.7}$$

The true class gets target probability $1 - \epsilon + \epsilon/k$ and each other class ϵ/k. The training loss is the ordinary cross-entropy $\mathrm{CE}(\mathbf{p}^\epsilon, \mathbf{q})$ to this target, the smoothed loss of Section 7.6. It is smallest at $\mathbf{q} = \mathbf{p}^\epsilon$, and a softmax reaches that prediction with a finite gap between the logit of the true class and each other logit,

$$z_y - z_j = \log \frac{1 - \epsilon + \epsilon/k}{\epsilon/k} = \log\Big(1 + \frac{k(1-\epsilon)}{\epsilon}\Big).$$

For $k = 10$ classes and $\epsilon = 0.1$ the gap is $\ln 91 \approx 4.51$, and the loss at the optimum is the entropy of the target, $H(\mathbf{p}^\epsilon) = 0.500$ nats.

It is instructive to see how fast the logits run away without smoothing. Take one example, start its logits at zero, and descend the loss with step size 1. The gradient with respect to the logits is $\mathbf{q} - \mathbf{p}^\epsilon$. With a one-hot target the wrong classes stay tied, and a step widens the gap g between the true class and the rest by $(1 - q_y) + q_j = k/(e^g + k - 1)$, about ke^{-g} once the gap is large. A step of that size multiplies e^g by $e^{ke^{-g}} \approx 1 + ke^{-g}$, which adds about k to it. So $e^g \approx kt$ and $g \approx \ln(kt)$ after t steps. The gap never stops growing, but ten times the training adds only $\ln 10 \approx 2.30$ to it. After $10{,}000$ steps we expect $\ln 10^5 \approx 11.51$. Here is the descent, with and without smoothing.

```
k, checkpoints = 10, (10, 100, 1000, 10000)
for eps in (0.0, 0.1):
    p_eps = np.full(k, eps / k)
    p_eps[0] += 1 - eps                  # the true class first
    z = np.zeros(k)                      # the logits of one example
    for step in range(1, checkpoints[-1] + 1):
        q = np.exp(z - z.max())
        z -= q / q.sum() - p_eps         # gradient step, step size 1
        if step in checkpoints:
            log_q = z - z.max() - np.log(np.exp(z - z.max()).sum())
            print(f'eps = {eps}  step {step:5d}  gap = {z[0] - z[1]:5.2f}  '
                  f'loss = {-p_eps @ log_q:.4f}')
print(f'optimum for eps = 0.1: gap = ln 91 = {math.log(91):.2f}, '
      f'loss = H(p_eps) = {entropy(p_eps):.4f}')
```

```
eps = 0.0  step    10  gap =  4.29  loss = 0.1158
eps = 0.0  step   100  gap =  6.87  loss = 0.0093
eps = 0.0  step  1000  gap =  9.21  loss = 0.0009
eps = 0.0  step 10000  gap = 11.51  loss = 0.0001
eps = 0.1  step    10  gap =  3.81  loss = 0.5243
eps = 0.1  step   100  gap =  4.51  loss = 0.5003
eps = 0.1  step  1000  gap =  4.51  loss = 0.5003
eps = 0.1  step 10000  gap =  4.51  loss = 0.5003
optimum for eps = 0.1: gap = ln 91 = 4.51, loss = H(p_eps) = 0.5003
```

With one-hot targets the gap reads 4.29, 6.87, 9.21 and 11.51. The prediction $\ln(kt)$ is 4.61, 6.91, 9.21 and 11.51 at these steps: the gap starts a little below it, where ke^{-g} is still a rough approximation, and agrees with it to two decimals from $1{,}000$ steps on. The loss keeps falling without ever arriving at zero. In a network large enough to fit its training set, the same growth shows as weights that keep growing and as probabilities near 1 on every training label, the wrong ones included. With $\epsilon = 0.1$ the gap is 4.51 after 100 steps and stays there, and the loss stops at 0.500.

Training can now come to rest, but a smoothed model has to be read differently. Its training loss levels off at $H(\mathbf{p}^\epsilon)$, not at zero, even when it predicts the target without error. Its probabilities are no longer the true conditional ones: no class receives more than $1-\epsilon+\epsilon/k$, here 0.91. In practice $\epsilon = 0.1$ is the usual choice, and the recipe of Section 7.6 uses it. Exercise 7 asks what smoothing does to the logits of the wrong classes.

37.1.3.4 Knowledge Distillation

A soft target can also come from another model. *Knowledge distillation* (Hinton et al. 2015) trains a small *student* network to imitate a large *teacher*, and Section 26.1 applies it to language models. The target is the teacher's whole distribution over the classes, not only its most probable class. That a "2" looks somewhat like a "7" is information a one-hot label does not carry. It is stored in the teacher's logits for the wrong classes, and Exercise 7 asks why a teacher trained with label smoothing passes on less of it.

As with any target, the loss is smallest when the student reports the target itself, so the decomposition Equation 37.5 has nothing new to say here. What needs an explanation is the scale of the loss: as we are about to see, it carries a factor that looks arbitrary at first sight.

A trained teacher's softmax is nearly one-hot, and the probabilities it gives to the wrong classes are too small to influence the student. Both distributions are therefore first *softened*: the logits are divided by a *temperature* $\tau > 0$ before the softmax,

$$p_j^{(\tau)} = \mathrm{softmax}(\mathbf{z}^{\mathrm{tea}}/\tau)_j, \qquad q_j^{(\tau)} = \mathrm{softmax}(\mathbf{z}^{\mathrm{stu}}/\tau)_j.$$

A temperature above one shrinks the gaps between logits and moves probability toward the wrong classes. The distillation loss is the KL divergence between the softened distributions, scaled by τ^2:

$$\mathcal{L}_{\mathrm{distill}} = \tau^2 \, D_{\mathrm{KL}}\big(\mathbf{p}^{(\tau)} \,\|\, \mathbf{q}^{(\tau)}\big). \tag{37.8}$$

The loss is zero only when the student matches the softened teacher. To see where the factor τ^2 comes from, differentiate the KL term with respect to a student logit. Only $-\sum_j p_j^{(\tau)} \log q_j^{(\tau)}$ depends on the student, and the derivative of a log-softmax at temperature τ is the usual softmax-minus-target expression scaled by $1/\tau$:

$$\frac{\partial}{\partial z_j^{\mathrm{stu}}} D_{\mathrm{KL}}(\mathbf{p}^{(\tau)} \| \mathbf{q}^{(\tau)}) = \frac{1}{\tau}\big(q_j^{(\tau)} - p_j^{(\tau)}\big).$$

Each student logit is pushed by the gap between the two softened probabilities of its class, divided by τ.

The gap itself shrinks as well. For large τ the softened distributions flatten toward uniform. Subtract the mean logit $\bar{z}$, which leaves the softmax unchanged, and expand $e^{(z_j - \bar{z})/\tau} \approx 1 + (z_j - \bar{z})/\tau$ to first order. The denominator becomes k because the deviations sum to zero, so $\mathrm{softmax}(\mathbf{z}/\tau)_j \approx \frac{1}{k}\big(1 + (z_j - \bar{z})/\tau\big)$. The difference $q_j^{(\tau)} - p_j^{(\tau)}$ therefore shrinks like $1/\tau$, and the gradient like $1/\tau^2$:

$$\frac{1}{\tau}\big(q_j^{(\tau)} - p_j^{(\tau)}\big) \;\approx\; \frac{1}{k\,\tau^2}\Big(\big(z_j^{\mathrm{stu}} - \bar{z}^{\mathrm{stu}}\big) - \big(z_j^{\mathrm{tea}} - \bar{z}^{\mathrm{tea}}\big)\Big).$$

Multiplying the loss by τ^2 therefore keeps the size of the gradient roughly constant as the temperature is tuned. This matters because the distillation loss is rarely used alone. In practice it is added, with a fixed weight, to the ordinary cross-entropy on the recorded labels, and without the factor every change of τ would also change the balance between the two terms. With the factor, the temperature decides only how much of the teacher's small probabilities the student gets to see.

We check the scaling on teacher logits $(5, 2, -1)$ and student logits $(3, 3, 0)$. Both have mean 2, so τ^2 times the gradient should approach $\frac{1}{3}\big((1, 1, -2) - (3, 0, -3)\big) = (-0.667, 0.333, 0.333)$, a vector of norm 0.816. We let automatic differentiation compute the gradient of the KL term at five temperatures, and print its norm without and with the factor τ^2.

```
z_tea = torch.tensor([5.0, 2.0, -1.0])
z_stu = torch.tensor([3.0, 3.0, 0.0], requires_grad=True)
print(' tau  |grad|  |tau^2 grad|  tau^2 grad')
```

```
    for tau in (1.0, 2.0, 5.0, 10.0, 20.0):
        p_tau = torch.softmax(z_tea / tau, 0)
        kl = (p_tau * (torch.log(p_tau) - torch.log_softmax(z_stu / tau, 0))).sum()
        g = torch.autograd.grad(kl, z_stu)[0].numpy()
        print(f'{tau:4.0f}  {np.linalg.norm(g):.4f}  '
              f'{np.linalg.norm(tau**2 * g):12.3f}  {np.round(tau**2 * g, 3)}')
    limit = (z_stu - z_stu.mean()) - (z_tea - z_tea.mean())
    limit = limit.detach().numpy() / len(limit)
    print(f'limit {"":6s}  {np.linalg.norm(limit):12.3f}  {np.round(limit, 3)}')
```

```
 tau  |grad|  |tau^2 grad|  tau^2 grad
   1  0.6391         0.639  [-0.462  0.441  0.022]
   2  0.2190         0.876  [-0.672  0.549  0.123]
   5  0.0368         0.920  [-0.741  0.478  0.263]
  10  0.0088         0.884  [-0.719  0.413  0.306]
  20  0.0021         0.854  [-0.697  0.374  0.322]
limit                0.816  [-0.667  0.333  0.333]
```

Without the factor the gradient collapses. Its norm falls from 0.64 at $\tau = 1$ to 0.0021 at $\tau = 20$, about three hundred times smaller, and the last doubling of τ divides it by about four, as $1/\tau^2$ predicts. With the factor, the norm stays between 0.64 and 0.92. It overshoots the predicted limit 0.816, peaks near $\tau = 5$, and comes back slowly, at a rate of order $1/\tau$: at $\tau = 20$ it still reads 0.854. At $\tau = 1$ the loss is the ordinary KL divergence, and with a one-hot teacher, which has zero entropy, it is the ordinary cross-entropy loss, so standard training is a special case.

37.1.4 Entropy and KL Divergence for Densities

So far every outcome could be counted. A regression target is a real number with a density, and the sums become integrals. The KL divergence survives this change. The entropy does not: it can be negative, and it comes to depend on the units in which the variable is measured. The loss of a continuous model can therefore fall below zero, and its floor moves with the units. What keeps its meaning is the excess over the floor, and with it the difference between two models on the same data. A first reading can skip these three subsections and resume at the last paragraph of the section.

37.1.4.1 Differential Entropy

For a continuous random variable X with density p, the sum in the entropy becomes an integral. The result is called the *differential entropy*,

$$h(X) = -\int p(x) \log p(x)\, dx. \tag{37.9}$$

It is the average surprisal of a density value, and density values are not probabilities: they carry the units of $1/x$, and they can exceed one. We write a lowercase h to keep it apart from the discrete H, because the two behave differently.

The smallest case shows how. The uniform density on $[0, a]$ equals $1/a$ there, so $h = -\int_0^a \frac{1}{a} \log \frac{1}{a}\, dx = \log a$. For $a = \frac{1}{2}$ this is $-\ln 2 \approx -0.693$ nats, a negative entropy,

which no discrete distribution can have. Measuring the same variable in centimetres instead of metres multiplies a by 100 and adds $\ln 100 \approx 4.61$ nats, although the uncertainty has not changed.

A shift of this kind occurs under any smooth change of variables. If $y = g(x)$ is invertible with derivative g', the density of Y at $g(x)$ is $p_X(x)/|g'(x)|$, and so

$$h(Y) = h(X) + \mathbb{E}\big[\log|g'(X)|\big].$$

For a vector $\mathbf{x} \in \mathbb{R}^n$ the definition and the shift keep their form, with an integral over $\mathbb{R}^n$ and the Jacobian determinant $|\det \partial g/\partial \mathbf{x}|$ in place of $|g'|$. Stretching a variable raises its differential entropy, and squeezing it lowers the entropy without limit. A Gaussian with standard deviation σ has $h = \frac{1}{2}\log(2\pi e\sigma^2)$, as we compute in Section 37.1.4.3, which is negative once $\sigma < (2\pi e)^{-1/2} \approx 0.24$.

The KL divergence does not share this defect. Apply the same change of variables to a model density q as to the data: both densities are divided by the same factor, which cancels in the ratio p/q, so the KL divergence does not change. For two uniform densities p_1 on $[0, a_1]$ and p_2 on $[0, a_2]$ with $a_1 \le a_2$, for instance, it is $\int p_1 \log(p_1/p_2)\,dx = \log(a_2/a_1)$, which for $a_1 = \frac{1}{2}$ and $a_2 = 1$ is $\ln 2 \approx 0.693$ nats in metres and in centimetres alike. The cross-entropy $-\mathbb{E}_P[\log q]$, on which a continuous model is trained, shifts by the same constant as h. The constant does not depend on the model, so the gradients and the minimizer of the cross-entropy are unchanged. When $h(p)$ is finite, the decomposition Equation 37.5 holds with h in place of H, and maximum likelihood for densities minimizes $D_{\mathrm{KL}}(p\|q_\theta)$ at the population level.

Differential entropy is not the limit of discrete entropy. Divide the line into cells of width Δ, and let X_Δ record which cell contains X. Then $P(X_\Delta = i) \approx p(x_i)\Delta$ for a point x_i in cell i, and

$$H(X_\Delta) + \log \Delta \to h(X) \quad \text{as } \Delta \to 0$$

for densities regular enough that the sums converge to the integrals (for example, when p is continuous and p and $p \log p$ are Riemann-integrable). The discrete entropy itself grows like $\log(1/\Delta)$, since a finer grid takes more questions, and h is what remains after that growth is subtracted. For the uniform density on $[0, 1]$ with $\Delta = \frac{1}{8}$, the eight cells are equally likely, $H_2(X_\Delta) = 3$ bits and $\log_2 \Delta = -3$, which sum to $0 = h$. Exercise 9 proves the limit and checks it on a Gaussian.

37.1.4.2 KL Divergence Between Gaussians

When a model reports a real number together with its uncertainty, it usually reports a Gaussian: a mean and a variance. To compare such a model with the data, or two such models with each other, we need the KL divergence between two Gaussians. It has a closed form. For $P = \mathcal{N}(\mu_1, \sigma_1^2)$ and $Q = \mathcal{N}(\mu_2, \sigma_2^2)$,

$$D_{\mathrm{KL}}\big(\mathcal{N}(\mu_1, \sigma_1^2)\,\big\|\,\mathcal{N}(\mu_2, \sigma_2^2)\big) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \tag{37.10}$$

The divergence is a log ratio of standard deviations plus the variance and the squared mean gap of P, both measured in units of the model's variance. It is non-negative and zero only when $\mu_1 = \mu_2$ and $\sigma_1 = \sigma_2$, as Gibbs' inequality requires. It is not symmetric, because σ_1 and σ_2 enter differently. Exercise 10 derives the formula from the definition.

The smallest case is a unit mean shift with equal variances, $P = \mathcal{N}(0, 1)$ and $Q = \mathcal{N}(1, 1)$, where the formula gives $(\mu_1 - \mu_2)^2/(2\sigma^2) = \frac{1}{2}$ nat in either direction. Now let the variances differ: $P = \mathcal{N}(0, 1)$ and $Q = \mathcal{N}(0, 4)$, so $\sigma_2 = 2$. The formula gives $D_{\mathrm{KL}}(P\|Q) = \ln 2 + \frac{1}{8} - \frac{1}{2} \approx 0.318$ and $D_{\mathrm{KL}}(Q\|P) = -\ln 2 + 2 - \frac{1}{2} \approx 0.807$ nats, a factor of 2.5 apart.

Figure 37.4 shows where each of the two divergences comes from. In both directions the integrand changes sign at $|x| = (8 \ln 2/3)^{1/2} \approx 1.36$, where the two densities cross. In panel (a) the wide model explains narrow data. It is too surprised in the core $|x| < 1.36$, which contributes $+0.424$, and less surprised than the truth in the tails, which contribute -0.106. In panel (b) the narrow model has to explain wide data. The core contributes -0.240, and the tails, where the narrow model gives little probability to outcomes that occur, contribute $+1.046$.

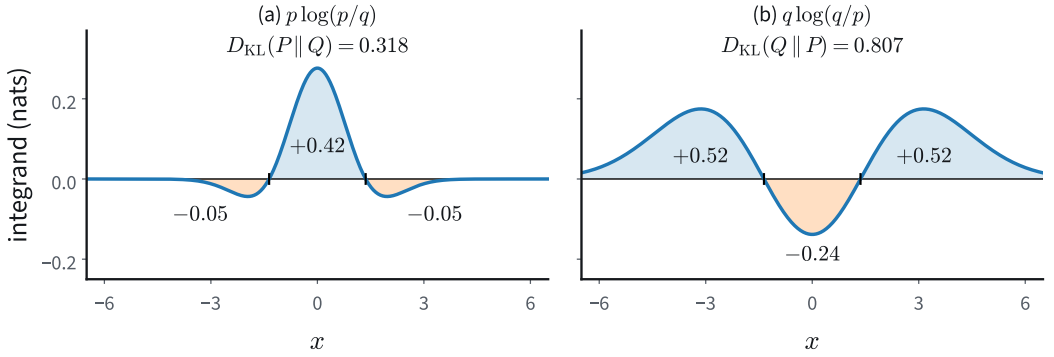


Figure 37.4: Where the KL divergence between $\mathcal{N}(0, 1)$ and $\mathcal{N}(0, 4)$ comes from. (a) The narrow distribution explained by the wide model: a positive core and slightly negative tails sum to 0.318 nats. (b) The wide distribution explained by the narrow model: the tails, where the narrow model assigns little probability, contribute 1.05 nats against a core of -0.24, for a total of 0.807. The sign changes at $|x| = 1.36$ in both directions.

A narrow model is far more surprised by wide data than a wide model is by narrow data. The KL divergence grows large when outcomes occur that the model has nearly ruled out, and it stays small when the model merely spreads probability over outcomes that do not occur. What this asymmetry does to a model that is fitted in one direction or the other is the subject of Section 37.3.2.

The definition suggests a second way to compute a KL divergence: draw samples from P and average $\log p(x) - \log q(x)$. This is how a divergence is estimated when no closed form exists, so it is worth knowing how accurate such averages are. The next cell defines the closed form and the Monte Carlo estimate with its standard error.

```
def gaussian_kl(mu1, sigma1, mu2, sigma2):
    """Closed-form KL(N(mu1, sigma1^2) || N(mu2, sigma2^2)), in nats."""
    return (math.log(sigma2 / sigma1)
```

```
            + (sigma1 ** 2 + (mu1 - mu2) ** 2) / (2 * sigma2 ** 2)
            - 0.5)

def gaussian_log_pdf(x, mu, sigma):
    """Log-density of N(mu, sigma^2) at x."""
    return (-0.5 * math.log(2 * math.pi * sigma ** 2)
            - (x - mu) ** 2 / (2 * sigma ** 2))

def mc_kl(mu1, sigma1, mu2, sigma2, n=200000):
    """Monte Carlo estimate of the same KL from n samples x ~ P, with its
    standard error."""
    x = np.random.default_rng().normal(loc=mu1, scale=sigma1, size=n)
    r = gaussian_log_pdf(x, mu1, sigma1) - gaussian_log_pdf(x, mu2, sigma2)
    return float(r.mean()), float(r.std() / math.sqrt(n))
```

For the mean shift the accuracy can be predicted. The log-ratio is $\log p(x) - \log q(x) = \frac{1}{2} - x$, which has variance one under P, so the average of $n = 2 \cdot 10^5$ samples has standard error $1/\sqrt{n} = 0.0022$.

For the pair with different variances, the two directions are not equally easy to estimate. Both average the same function up to its sign, $\ln 2 - \frac{3}{8}x^2$, one over samples from the narrow Gaussian and the other over samples from the wide one, where x^2 fluctuates four times as much. We estimate all three divergences.

```
pairs = {'N(0,1) || N(1,1)': (0.0, 1.0, 1.0, 1.0),
         'N(0,1) || N(0,4)': (0.0, 1.0, 0.0, 2.0),
         'N(0,4) || N(0,1)': (0.0, 2.0, 0.0, 1.0)}
print(f'{"P || Q":18s} {"closed":>7s} {"sampled":>8s} {"std err":>8s}')
for name, args in pairs.items():
    estimate, std_err = mc_kl(*args)
    print(f'{name:18s} {gaussian_kl(*args):7.4f} {estimate:8.4f} {std_err:8.4f}')
```

```
P || Q             closed  sampled  std err
N(0,1) || N(1,1)   0.5000   0.5010   0.0022
N(0,1) || N(0,4)   0.3181   0.3169   0.0012
N(0,4) || N(0,1)   0.8069   0.8017   0.0047
```

The standard errors are 0.0022, 0.0012 and 0.0047, and every estimate lies within a few of its standard errors of the closed form. The direction that draws from the wide Gaussian is four times noisier than its reverse, so it needs sixteen times as many samples for the same accuracy. Panel (b) of Figure 37.4 shows why: this divergence is collected far out in the tails, where the log-ratio is large and the samples thin out. Section 37.7 meets the same difficulty at a much larger scale.

37.1.4.3 Maximum Entropy and the Gaussian

On a finite set the uniform distribution is the most uncertain one. Which density on the real line is? Without a constraint there is none. No density is uniform on $\mathbb{R}$, and stretching a density raises its differential entropy without limit. So we fix the two numbers that a Gaussian model reports, the mean μ and the variance σ^2. Among all densities with these two moments, the Gaussian has the largest entropy. This is the sense in which a Gaussian noise model assumes the least: it commits to a mean and a variance, and to nothing else.

The proof is the one-line argument for the uniform bound, with one change. There $-\log u(x) = \log k$ was the same for every outcome, so every distribution P had the same cross-entropy $\log k$ to the uniform model. Here let φ be the Gaussian density with the given moments. Its surprisal $-\log \varphi(x) = \frac{1}{2}\log(2\pi\sigma^2) + \frac{(x-\mu)^2}{2\sigma^2}$ is not constant, but it is a quadratic in x, and the average of a quadratic depends on P only through the two moments that we fixed. For every density P with these moments,

$$\mathrm{CE}(P,\varphi) = \tfrac{1}{2}\log(2\pi\sigma^2) + \frac{\mathbb{E}_P\big[(x-\mu)^2\big]}{2\sigma^2} = \tfrac{1}{2}\log(2\pi\sigma^2) + \tfrac{1}{2} = \tfrac{1}{2}\log(2\pi e\sigma^2).$$

The cross-entropy to the Gaussian does not depend on which such P produced the data. For $P = \varphi$ the cross-entropy is the entropy, so this value is the Gaussian's own differential entropy, the closed form quoted whenever a Gaussian entropy is needed. For every other such P, the decomposition Equation 37.5, with h in place of H, and Gibbs' inequality give

$$h(P) = \mathrm{CE}(P,\varphi) - D_{\mathrm{KL}}(P\|\varphi) \le \tfrac{1}{2}\log(2\pi e\sigma^2) = h(\mathcal{N}(\mu,\sigma^2)), \tag{37.11}$$

with equality if and only if $P = \mathcal{N}(\mu,\sigma^2)$. (If $h(P) = -\infty$, there is nothing to prove.) The gap in this inequality, $D_{\mathrm{KL}}(P\|\varphi)$, has a use of its own: it is the excess loss of the best Gaussian model on data that are not Gaussian (Exercise 11).

The uniform and the Gaussian are two cases of one mechanism. Whenever the surprisal of a candidate averages to the same value under every distribution that meets the constraints, the candidate has the largest entropy among them. Section 36.4.2 proves this for constraints on the averages of any set of statistics, where the candidates form an exponential family. The geometric distribution on the counts $\{0, 1, 2, \ldots\}$ is a third case. Its surprisal is affine in the outcome, so it has the largest entropy among the distributions with its mean.

The model Q' gave zero probability to an outcome that occurs, and its KL divergence was infinite; the narrow Gaussian of Figure 37.4 came close, with a divergence that arose in tails it had nearly ruled out. A model that turns a few random numbers into an image puts all its probability on a thin surface in the space of pixels. If the data do not lie on that surface, the KL divergence is infinite whether the surface passes close by the data or far from them. Section 37.3 shows that every divergence built on the ratio p/q shares this blindness, and Section 37.4 builds comparisons that still tell near from far. A second question is closer at hand. We counted questions and read them as codewords, but only for a source whose surprisals are whole numbers of bits. Section 37.2 builds a code for every source and asks how close a real compressor gets to the entropy.

37.1.5 Summary

A cross-entropy loss is the model's average surprisal at the data. It is the entropy of the data, which no model can lower, plus the KL divergence from the data to the model, which by Gibbs' inequality is never negative and vanishes only for the right model. A loss must therefore be read against its floor, the conditional entropy of the label given the input, which on fresh data only the true conditional probabilities reach. The floor is zero when the input determines the label, and $H_b(\rho) + \rho\log(k-1)$ under symmetric label noise. In bits, the entropy of a finite source is the average number of yes/no questions needed to identify an outcome, to within one question, and it is at most $\log_2 k$ for k outcomes. Softened targets move the optimum: label smoothing to a finite logit gap with a floor above zero, distillation

to the softened teacher, where the factor τ^2 holds the gradient scale fixed as the temperature changes. For densities the entropy depends on the units of the data, while the KL divergence and the minimizer of the cross-entropy do not.

37.1.6 Exercises

1. **Uniqueness of self-information.** ⊠ Show that self-information is unique up to scale: if a strictly decreasing function $I(p)$ on $(0, 1]$ satisfies $I(1) = 0$ and $I(p_1 p_2) = I(p_1) + I(p_2)$ for all $p_1, p_2 \in (0, 1]$, then $I(p) = -c \log p$ for some constant $c > 0$. Work with the function $g(t) = I(e^{-t})$ for $t \geq 0$.

2. **Two proofs of Gibbs' inequality.** The text proves Gibbs' inequality with Jensen's inequality. Section 36.4.2 proves it from the tangent line $\ln u \leq u - 1$.

   1. Show that the tangent-line proof, with $u = q(x)/p(x)$ on the support of P, gives $D_{\mathrm{KL}}(P\|Q) \geq 1 - Q(\operatorname{supp} P)$, where the text obtains $-\log Q(\operatorname{supp} P)$. Which bound is larger?
   2. Find a pair $P \neq Q$ for which the bound of the text holds with equality. Can the tangent-line bound hold with equality for $P \neq Q$?
   3. Why are the two proofs nevertheless the same argument? Hint: the standard proof of Jensen's inequality for a concave f bounds f by its tangent line at $\mathbb{E}[X]$.

3. code **The label-noise floor.** Labels on k classes are correct with probability $1 - \rho$ and otherwise uniform over the other $k - 1$ classes, independently of the input.

   1. Assume that the clean label is a function of the input. Show that the population cross-entropy of any classifier is at least $H_b(\rho) + \rho \log(k - 1)$, that the optimal prediction is the noisy conditional distribution itself, and that the optimal accuracy is $1 - \rho$ whenever $1 - \rho > \rho/(k - 1)$.
   2. Evaluate the floor for $k = 10$ and $\rho = 0.05, 0.1, 0.2$.
   3. A colleague reports a validation loss of 0.40 nats on a 10-class data set known to have 10% such noise in every split. List possible explanations, and say which of them the floor rules out and which it does not.
   4. Train the softmax regression of Section 3.3 on Fashion-MNIST with 10% of the training and test labels flipped in this way. Compare the test loss with 0.545 and the test accuracy with 90%. The gap is large. Which assumption of (a) does Fashion-MNIST violate, what does the linear model's own error add, and how could you estimate the floor that does apply?

4. **Label noise on an ambiguous input.** The floor of the previous exercise assumes that the input determines the clean class. Fix one input at which it does not: the clean class C has the distribution $\mathbf{c}$ on k classes, and the recorded label Y equals C with probability $1 - \rho$ and is otherwise uniform over the other $k - 1$ classes.

   1. Show that $H(Y \mid C) = H_b(\rho) + \rho \log(k - 1)$, whatever $\mathbf{c}$ is. Call this value H_{noise}.
   2. Write the chain rule Equation 37.2 for the pair (C, Y) in both orders, and conclude that $H_{\text{noise}} \leq H(Y) \leq H(C) + H_{\text{noise}}$. Which result of the text gives the lower bound? For which $\mathbf{c}$ and ρ does the upper bound hold with equality?
   3. For $k = 2$ and $\rho = 0.1$, compute $H(Y)$ for $\mathbf{c} = (1, 0)$, $(0.9, 0.1)$ and $(\frac{1}{2}, \frac{1}{2})$, and compare each value with the two bounds. How much does the noise add to the entropy of the label when the clean class is a fair coin, and why?

4. A data set has 10% label noise on ten classes, and some of its inputs are ambiguous. Is the value 0.545 nats of Section 37.1.3.1 a lower or an upper bound on its floor?

5. code **The initial-loss check.** A freshly initialized network produces small, random logits.
   1. Let the k logits be independent with mean 0 and variance s^2. Expand $\log \sum_j e^{z_j}$ to second order in the z_j to show $\mathbb{E}[\text{CE}] \approx \ln k + \frac{s^2}{2}(1 - \frac{1}{k})$.
   2. Check the formula by sampling Gaussian logits for $k = 10$ and $s = 0.1, 0.5, 1$. Where does the second-order approximation start to fail, and in which direction?
   3. A 10-class model reports a loss of 4.1 at initialization. What does the formula say about the scale of its logits, and what would you change in the initialization? Given what you found in (b), is the formula still trustworthy at that scale?

6. **Other scoring rules.** A scoring rule $S(\mathbf{q}, y)$ is strictly proper if the expected penalty $\mathbb{E}_{y\sim\mathbf{p}}[S(\mathbf{q}, y)]$ is uniquely minimized at $\mathbf{q} = \mathbf{p}$. For the log score, this property is Gibbs' inequality.
   1. Show that the expected Brier score, with $S(\mathbf{q}, y) = \sum_j (q_j - \mathbf{1}(j = y))^2$, exceeds the expected Brier score of the truthful report by $\|\mathbf{q} - \mathbf{p}\|^2$, so that the Brier score is strictly proper (Gneiting and Raftery 2007).
   2. The linear score $S(\mathbf{q}, y) = -q_y$ rewards the probability placed on the observed class. Which report minimizes its expected penalty? Is the linear score proper? What does a classifier trained with it report for an input whose label is class 1 with probability 0.6 and class 2 with probability 0.4?

7. **Label smoothing on ambiguous inputs.** ⊠ Label smoothing trains with the target Equation 37.7.
   1. In the training distribution, an input $\mathbf{x}$ carries the label y with probability $p(y \mid \mathbf{x})$. Show that the expected smoothed loss for this input is minimized by $\mathbf{q}^*(\cdot \mid \mathbf{x}) = (1 - \epsilon)\, p(\cdot \mid \mathbf{x}) + \frac{\epsilon}{k}\mathbf{1}$. How can $p(\cdot \mid \mathbf{x})$ be recovered from $\mathbf{q}^*$? What is the largest probability that the minimizer can assign to a class?
   2. For one example with logits $\mathbf{z}$ and $\mathbf{q} = \text{softmax}(\mathbf{z})$, the gradient of $\text{CE}(\mathbf{p}^\epsilon, \mathbf{q})$ with respect to z_j is $q_j - p_j^\epsilon$. For $\epsilon > 0$, show that all wrong-class logits are equal at every stationary point.
   3. With one-hot targets ($\epsilon = 0$), the gradient with respect to a wrong-class logit is q_j, and the loss has no stationary point. Show that gradient descent on the logits of one example still shrinks the gap between two wrong-class logits, and explain why this shrinkage stalls as $q_y \to 1$. In a network, the logits of all examples are computed from shared features. Which differences between wrong-class logits can therefore survive training with $\epsilon = 0$? Why does training with $\epsilon > 0$ remove them?
   4. Müller et al. (2019) observe that teachers trained with label smoothing produce worse students under the distillation loss Equation 37.8. Explain this observation with the results of (b) and (c).

8. code **Distillation at high temperature.** ⊠ Hinton et al. (2015) state in their Section 2.1 that, at high temperature, distillation reduces to matching zero-mean logits by squared error.
   1. Write $q_j = p_j(1 + u_j)$, so that $\sum_j p_j u_j = 0$, and expand $D_{\text{KL}}(\mathbf{p}\|\mathbf{q}) =$

$-\sum_j p_j \log(1+u_j)$ to second order to obtain

$$D_{\mathrm{KL}}(\mathbf{p}\|\mathbf{q}) = \frac{1}{2}\sum_j \frac{(q_j-p_j)^2}{p_j} + O(\|\mathbf{q}-\mathbf{p}\|^3).$$

Combine this expansion with the large-τ approximation of $\mathrm{softmax}(\mathbf{z}/\tau)$ in Section 37.1.3.4 to show that, as $\tau \to \infty$,

$$\tau^2 D_{\mathrm{KL}}(\mathbf{p}^{(\tau)} \,\|\, \mathbf{q}^{(\tau)}) \to \frac{1}{2k}\sum_{j=1}^{k}\Big((z_j^{\mathrm{stu}} - \bar{z}^{\mathrm{stu}}) - (z_j^{\mathrm{tea}} - \bar{z}^{\mathrm{tea}})\Big)^2.$$

2. Evaluate $\tau^2 D_{\mathrm{KL}}(\mathbf{p}^{(\tau)}\|\mathbf{q}^{(\tau)})$ for $\tau = 1, 2, 5, 10, 100$ with the logits $\mathtt{z_tea} = (5, 2, -1)$ and $\mathtt{z_stu} = (3, 3, 0)$ of Section 37.1.3.4, and compare it with the limit.
3. At low temperature, which teacher logits have almost no influence on the loss? Hinton et al. (2015) argue that this can be an advantage and also a drawback. Give one argument for each side, based on what the teacher's own cross-entropy training does to these logits.

9. code **Quantized entropy against differential entropy.** Let X have density p, and let X_Δ record which cell $[i\Delta, (i+1)\Delta)$ contains X.
   1. Show that $H(X_\Delta) + \log\Delta \to h(X)$ as $\Delta \to 0$ when p is continuous and p and $p\log p$ are Riemann-integrable. Write $P(X_\Delta = i) = p(x_i)\Delta$ for some x_i in the cell, by the mean value theorem for integrals.
   2. Check the limit by hand for the uniform density on $[0,1]$ with $\Delta = \frac{1}{8}$. What happens for a cell width that does not divide 1, such as $\Delta = 0.3$?
   3. Compute $H(X_\Delta) + \log\Delta$ numerically for $X \sim \mathcal{N}(0,1)$ with $\Delta = 0.5, 0.1, 0.01$, using the Gaussian distribution function for the cell probabilities, and compare it with $h = \frac{1}{2}\log(2\pi e) \approx 1.4189$ nats. (At $\Delta = 0.5$ you should find 1.4292.)
   4. Why does $H(X_\Delta)$ itself diverge as $\Delta \to 0$? A regression target is binned into cells of width Δ and predicted with a softmax classifier. Which quantity does its loss approach at best, and what must be added to it before it can be compared with the negative log-density of a continuous model?

10. **Gaussian KL divergence.** Let $P = \mathcal{N}(\mu_1, \sigma_1^2)$.
    1. Derive the closed form Equation 37.10 by writing out $\mathbb{E}_{x\sim P}[\log p(x) - \log q(x)]$ and using $\mathbb{E}_{x\sim P}[x] = \mu_1$ and $\mathbb{E}_{x\sim P}[(x-\mu_1)^2] = \sigma_1^2$.
    2. Let Q have mean $\mu_1 + \delta$ and standard deviation $\sigma_1 e^{\eta}$. Expand both $D_{\mathrm{KL}}(P\|Q)$ and $D_{\mathrm{KL}}(Q\|P)$ to second order in δ and η. At which order do the two directions first differ?
    3. Evaluate the second-order approximation for $P = \mathcal{N}(0,1)$ and $Q = \mathcal{N}(0,4)$, and compare it with the exact values 0.318 and 0.807 nats of the two directions.

11. **The excess loss of a Gaussian model.** A regression model reports a Gaussian $\mathcal{N}(m, s^2)$ for a target whose density P has mean μ, variance σ^2 and a finite differential entropy.

1. Show that $\mathrm{CE}(P, \mathcal{N}(m, s^2))$ is smallest at $m = \mu$ and $s^2 = \sigma^2$, and that this best Gaussian model exceeds the floor $h(P)$ by $D_{\mathrm{KL}}(P\|\varphi) = h(\varphi) - h(P)$, where φ is the density of $\mathcal{N}(\mu, \sigma^2)$.
2. Evaluate the excess for the uniform density on $[0, a]$, which has variance $a^2/12$, and for the Laplace density $\frac{1}{2b}e^{-|x|/b}$, which has variance $2b^2$ and differential entropy $1 + \log(2b)$. (You should find about 0.176 and 0.072 nats.)
3. Neither answer depends on the scale a or b. Explain why without computing, from what Section 37.1.4.1 says about a change of units.
4. The Gaussian has the largest entropy for a given variance, so no density has a higher floor. Is there also a largest excess? Consider densities that concentrate near the two points $\mu \pm \sigma$. What does your answer say about a Gaussian model for a target with two well-separated modes?

37.2 Codes, Compression, and Perplexity

A model that reaches a loss of x bits per symbol on a text should be able to store that text in about x bits per symbol. Section 37.1 suggested as much when it read an entropy and a cross-entropy as average codeword lengths. It did so for one source, whose surprisals are whole numbers of bits, and we never built a code for any other. The claim means little until such a code exists. This section asks two questions: is there a code whose length is a model's log-loss, and how close do real compressors come to it?

Compressors are familiar tools, and they already suggest the answer. Take *The Time Machine*, the novel of Section 8.2, reduced to 173,428 characters over 27 symbols, the 26 letters and the space. The general-purpose compressor `zlib`, which replaces repeated strings by pointers and then codes one symbol at a time, stores it in 2.84 bits per character. A small model that predicts each character from the three before it, learning its counts as it reads, codes the same text in 2.55 bits per character. The better predictor writes the shorter file, and this section shows why. Once a code exists whose length is the log-loss, a loss such as the 1.2 nats per character of the chapter introduction is a file size, and two models can be compared by the length of the files they write.

We build the code in steps, and each step answers a question. How short can codewords be when the receiver gets no separators between them? What does it cost that every codeword has a whole number of bits, and can a code avoid that cost? How short can a file of text get, where each character depends on those before it? And what does a perplexity, the number that language models report, say about the file? We count in bits throughout, because codewords are binary strings; x bits are $x \ln 2$ nats.

Beyond the book's standard imports, the section needs `re` to clean the text and `zlib` as the compressor to beat.

```
import re
import zlib
```

37.2.1 Symbol Codes

We start with the simplest sender. A stream of symbols, drawn independently from a distribution P over k outcomes, must travel as one binary string with no separators, and the receiver must split it back into symbols.

37.2.1.1 Prefix Codes and the Kraft Inequality

Each outcome i gets a binary string, its *codeword*, of length ℓ_i. The receiver can split the stream without separators if no codeword is a prefix of another; such a code is called *prefix-free*, or a *prefix code*. Telephone numbers work this way: once the dialed digits match a number, no longer number starts with the same digits, so the exchange knows that the number is complete. The code $0, 10, 110, 111$ for four symbols is prefix-free, and the stream `110010111` splits in only one way, as `110|0|10|111`.

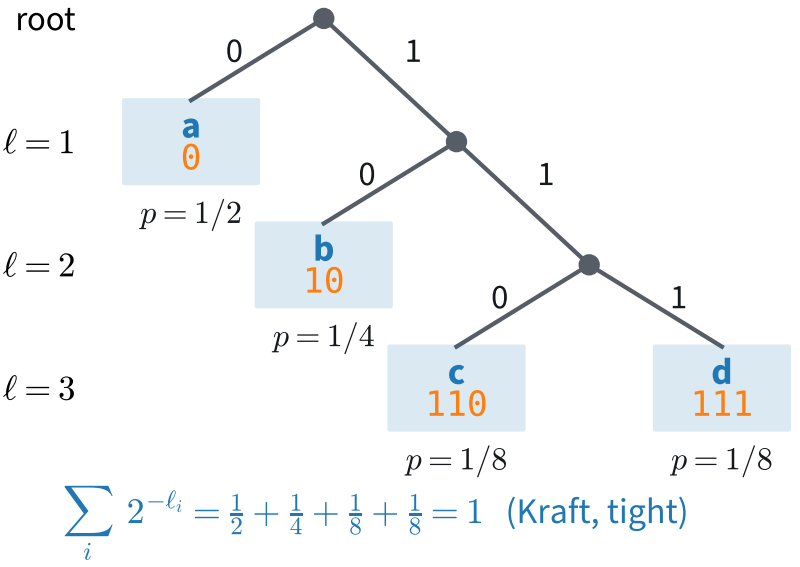


Figure 37.5: The prefix-free code $0, 10, 110, 111$ as a binary tree. Each codeword is the path from the root to a leaf (left edge 0, right edge 1), and no codeword continues through another. A leaf at depth ℓ uses up a fraction $2^{-\ell}$ of the tree, and the four leaves use up all of it, $2^{-1} + 2^{-2} + 2^{-3} + 2^{-3} = 1$. The symbols a, b, c, d and the probabilities below them are those of the previous section's dyadic source.

Figure 37.5 draws this code as a binary tree. A codeword of length ℓ ends at a leaf and removes everything below it, a fraction $2^{-\ell}$ of the tree. The fractions removed by all codewords must fit into one tree, a budget of 1, so short codewords are expensive. The lengths $(1, 2, 3, 3)$ of the figure use $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{8} = 1$, the whole tree. Four codewords of length 1 would use 2 trees, which is impossible: only two strings of length one exist. The lengths $(1, 2, 4)$ use $\frac{1}{2} + \frac{1}{4} + \frac{1}{16} = 0.8125$ and fit with room to spare, for instance as $0, 10, 1100$.

The tree is also a strategy for the questions game of Section 37.1. Every internal node asks a yes/no question, and a codeword is the list of answers. With 0 for yes, the code $0, 10, 110, 111$ asks "Is it the first symbol?", then "the second?", then "the third?". The previous section wrote 1 for yes and obtained the mirror image, $1, 01, 001, 000$ (Exercise 1).

Proposition (Kraft inequality). *A prefix-free binary code with codeword lengths $\ell_1, \dots, \ell_k$ exists if and only if*

$$\sum_{i=1}^{k} 2^{-\ell_i} \le 1. \tag{37.12}$$

Proof. Identify each codeword c of length ℓ with the half-open interval $[0.c,\ 0.c{+}2^{-\ell})$ of all real numbers in $[0,1)$ whose binary expansion begins with c. For example, the codeword 10 owns $[0.10_2, 0.11_2) = [\frac{1}{2}, \frac{3}{4})$, of length 2^{-2}. In Figure 37.5 this interval is the leaf of b together with everything that the leaf removes below it. Such *dyadic* intervals are either nested or disjoint. If c is a prefix of c', every number that begins with c' begins with c, so the intervals are nested. Otherwise c and c' differ at some digit within the shorter length, no number can begin with both, and the intervals are disjoint. A prefix-free code therefore owns disjoint intervals inside $[0,1)$, and their total length $\sum_i 2^{-\ell_i}$ is at most 1.

Conversely, let the lengths satisfy Equation 37.12. The key step is to sort them, $\ell_1 \le \cdots \le \ell_k$, and to lay intervals of length $2^{-\ell_i}$ side by side from 0. The i-th interval starts at $s_i = \sum_{m<i} 2^{-\ell_m}$ and ends at $s_i + 2^{-\ell_i} \le 1$. Every earlier length satisfies $\ell_m \le \ell_i$, so each term of s_i is a multiple of $2^{-\ell_i}$, and so is s_i. The i-th interval is therefore the dyadic interval of the codeword formed by the first ℓ_i binary digits of s_i. The intervals are disjoint, so by the first part no codeword is a prefix of another. ■

The budget binds more codes than the prefix-free ones. A code is *uniquely decodable* if every concatenation of codewords parses in only one way, prefix-free or not. McMillan showed that such codes obey the same inequality (McMillan 1956) (Exercise 2). Every length profile that a uniquely decodable code achieves, a prefix code therefore achieves too, and restricting to prefix codes costs nothing.

37.2.1.2 Shannon Lengths and the Entropy Bound

Given the budget, which lengths minimize the expected code length $\mathbb{E}[\ell] = \sum_i p_i \ell_i$? Frequent symbols should get short codewords, but a codeword of length 1 already spends half the budget.

To see where the answer comes from, let the lengths be real numbers for a moment. Unspent budget never helps, because shortening a codeword lowers $\mathbb{E}[\ell]$, so at the optimum $\sum_i 2^{-\ell_i} = 1$. Lagrange's rule for a minimum under an equality constraint says that there the gradient of the objective is a multiple of the gradient of the constraint (Section 34.2.4). The derivative of $\mathbb{E}[\ell]$ in ℓ_i is p_i, and that of $\sum_i 2^{-\ell_i}$ is $-2^{-\ell_i} \ln 2$. Hence $2^{-\ell_i} = c\, p_i$ for one constant c and all i. Summing both sides over i gives $c = 1$, and the best real lengths and their expected length are

$$\ell_i = \log_2 \frac{1}{p_i}, \qquad \mathbb{E}[\ell] = \sum_i p_i \log_2 \frac{1}{p_i} = H_2(P).$$

So each symbol gets as many bits as its *surprisal* $\log_2(1/p_i)$, the self-information of Section 37.1 in bits, and the expected length is $H_2(P)$, the entropy in bits. Real codewords need whole bits. For the dyadic source $P = (\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$ the surprisals $1, 2, 3, 3$ are already integers, the code of Figure 37.5 has these lengths, and $\mathbb{E}[\ell] = 1.75$ bits $= H_2(P)$, with nothing wasted. For $P = (0.6, 0.3, 0.1)$, the first of the chapter's running pair of distributions, the surprisals are 0.74, 1.74 and 3.32 bits. Rounding them up gives the *Shannon*

lengths $1, 2, 4$, the profile that fit the tree with room to spare. Its Kraft sum is 0.8125, and $\mathbb{E}[\ell] = 0.6 + 0.6 + 0.4 = 1.6$ bits, against $H_2(P) = 1.295$.

Whole-bit lengths chosen with more care cannot beat the entropy, and the Shannon lengths never land more than one bit above it. Both statements rest on one fact from Section 37.1, the decomposition of the cross-entropy, which we restate in bits; the subscript 2 and the superscript (2) mark base-2 logarithms. The expected log-loss of a model Q on data from P, the cross-entropy, splits into the entropy of the data and the Kullback–Leibler divergence,

$$\mathrm{CE}_2(P, Q) = -\sum_i p_i \log_2 q_i = H_2(P) + D^{(2)}_{\mathrm{KL}}(P\|Q), \qquad D^{(2)}_{\mathrm{KL}}(P\|Q) \geq 0, \tag{37.13}$$

where the last inequality is Gibbs': no model beats the entropy of the data, and only $Q = P$ reaches it. For the running pair, the P above and $Q = (0.2, 0.5, 0.3)$, the three terms are $1.867 = 1.295 + 0.571$ bits, up to rounding in the last digit.

Proposition (entropy bounds the code length). *Let P have all $p_i > 0$ (symbols of probability zero need no codeword). Every prefix-free binary code has $\mathbb{E}[\ell] \geq H_2(P)$, with equality if and only if every p_i equals $2^{-\ell_i}$. The* Shannon code *with lengths $\ell_i = \lceil \log_2(1/p_i) \rceil$ is prefix-free and has $\mathbb{E}[\ell] < H_2(P) + 1$.*

Proof. For the lower bound, compare the code with the distribution it implies. Let $Z = \sum_i 2^{-\ell_i}$, which is at most 1 by Kraft, and let $q_i = 2^{-\ell_i}/Z$. Then $\ell_i = -\log_2 q_i - \log_2 Z$, and averaging under P gives

$$\mathbb{E}[\ell] = \sum_i p_i \ell_i = -\sum_i p_i \log_2 q_i - \log_2 Z = H_2(P) + D^{(2)}_{\mathrm{KL}}(P\|Q) - \log_2 Z \geq H_2(P). \tag{37.14}$$

The inequality holds because the KL term is never negative and $-\log_2 Z \geq 0$. Equality needs both to vanish: $Q = P$ and $Z = 1$, that is, $p_i = 2^{-\ell_i}$. For the upper bound, the Shannon lengths satisfy $2^{-\ell_i} \leq p_i$, so they pass Kraft's test ($\sum_i 2^{-\ell_i} \leq \sum_i p_i = 1$) and a prefix code with these lengths exists. Each length is less than $\log_2(1/p_i) + 1$, and averaging gives $\mathbb{E}[\ell] < H_2(P) + 1$. $\blacksquare$

The identity Equation 37.14 is worth more than the bound it proves. A code can waste bits in only two ways: by implying a distribution Q other than P, and by leaving budget unspent. For the running P the Shannon code wastes $1.6 - 1.295 = 0.305$ bits, of which 0.005 is mismatch and 0.300 is unspent budget (Exercise 4).

The proposition places the best expected length of a symbol code between $H_2(P)$ and $H_2(P) + 1$ bits. This is the bound on the number of yes/no questions that Section 37.1 promised. The best length equals the entropy only when every probability is a power of $\tfrac{1}{2}$.

How much of the one bit can a better code recover? Huffman's algorithm (Huffman 1952) builds the best symbol code by merging the two least probable symbols into one, again and again (Exercise 3), and it is the symbol code inside `zlib`. For the running P it finds the lengths $1, 2, 2$ and spends 1.4 bits, only 0.1 above the entropy, so two thirds of the Shannon code's waste was avoidable. But no codeword is shorter than one bit, and for a skewed source that is nearly the whole of the allowed slack.

Take a source that emits the symbol a with probability 0.9 and the symbol b with probability 0.1. Its entropy is 0.47 bits per symbol, and each a carries only $\log_2(1/0.9) = 0.15$ bits of surprise. Yet every symbol code spends a whole bit on an a, more than six times its information. At probability 0.999 the entropy is 0.011 bits per symbol, and the cost is still 1. The "$+1$" is the cost of coding one symbol at a time. Coding blocks of symbols shrinks it. A block of n symbols is one symbol of a larger alphabet, and its code rounds up once for the whole block, so the slack per original symbol is below $1/n$. Huffman's code for pairs of a/b symbols already spends only 0.645 bits per symbol.

37.2.1.3 Mismatched Codes and Cross-Entropy

In practice the code is built from a model Q, and the data come from P. The Shannon code for Q has lengths $\ell_i = \lceil \log_2(1/q_i) \rceil$, which requires $q_i > 0$ for every symbol that P can produce. On data from P its expected length satisfies

$$\mathrm{CE}_2(P, Q) \;\le\; \mathbb{E}_P[\ell] \;<\; \mathrm{CE}_2(P, Q) + 1, \tag{37.15}$$

because termwise $\log_2(1/q_i) \le \ell_i < \log_2(1/q_i) + 1$, and we average under P. Cross-entropy is therefore the length of a code built from the model, to within one bit, and by Equation 37.13 the KL divergence is the extra length over the code built from the truth.

The cell builds the Shannon code of a distribution q, averages its lengths under the data distribution p, and prints the Kraft sum and the cross-entropy next to the result. It does so for the two sources of this section, each with its own code and with a mismatched one: the dyadic source with the code of the uniform distribution, and the running P with the code of $Q = (0.2, 0.5, 0.3)$.

```
def shannon_lengths(q):
    return [math.ceil(math.log2(1 / q_i)) for q_i in q]

def code_report(name, p, q):
    """The Shannon code built from q, used on data from p (all in bits)."""
    ells = shannon_lengths(q)
    kraft = sum(2 ** -ell for ell in ells)
    avg = sum(p_i * ell for p_i, ell in zip(p, ells))
    ce2 = -sum(p_i * math.log2(q_i) for p_i, q_i in zip(p, q))
    print(f'{name:<16} lengths {ells}  Kraft {kraft:.4f}  '
          f'E[ell] {avg:.3f}  CE2 {ce2:.3f}')

dyadic, uniform = [1/2, 1/4, 1/8, 1/8], [1/4] * 4
p, q = [0.6, 0.3, 0.1], [0.2, 0.5, 0.3]
code_report('dyadic, own', dyadic, dyadic)
code_report('dyadic, uniform', dyadic, uniform)
code_report('P, own', p, p)
code_report('P, code from Q', p, q)
```

```
dyadic, own      lengths [1, 2, 3, 3]  Kraft 1.0000  E[ell] 1.750  CE2 1.750
dyadic, uniform  lengths [2, 2, 2, 2]  Kraft 1.0000  E[ell] 2.000  CE2 2.000
P, own           lengths [1, 2, 4]  Kraft 0.8125  E[ell] 1.600  CE2 1.295
P, code from Q   lengths [3, 1, 2]  Kraft 0.8750  E[ell] 2.300  CE2 1.867
```

The two rows with a matched code repeat the hand calculations: there the column CE2 is the entropy, and the expected lengths are 1.75 and 1.6 bits. The mismatched rows are new. The code built from Q has lengths $3, 1, 2$, so the most frequent symbol of P gets the longest codeword. On data from P it averages 2.3 bits, inside the window $1.867 \leq 2.3 < 2.867$ of Equation 37.15. The uniform code spends two bits on every symbol of the dyadic source. Nothing is rounded there, so the lower bound holds with equality, and $2.0 = 1.75 + 0.25$ bits is the matched length plus a KL divergence of 0.25 bits. The Kraft column shows which codes leave budget unspent: none for the dyadic source, 0.1875 for the Shannon code of P, and 0.125 for the code from Q.

The bound Equation 37.15 covers the Shannon code of Q, not every code someone might build from Q. Huffman's code for Q has lengths $2, 1, 2$, spends the whole budget, and costs $1.2 + 0.3 + 0.2 = 1.7 < 1.867$ bits on data from P. Its implied distribution is $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$, not Q, and it happens to be closer to P. By Equation 37.14 the waste of a code depends on the distribution it implies, and Q enters only through the lengths it suggests.

37.2.2 Stream Codes and Description Length

A symbol code rounds once per symbol. We now build a code that rounds once per message, let a language model drive it, and then ask what the model itself costs to send.

37.2.2.1 Arithmetic Coding

Blocks remove the rounding loss, but they need a table. A code for blocks of 100 binary symbols has 2^{100} codewords, more than any computer can store. What we want is the length of the block code without its table: one codeword for the whole message, computed from the message itself. A code of this kind is a *stream code*. *Arithmetic coding* (Rissanen 1976; Witten et al. 1987) is the standard stream code. It represents the message as an interval. Figure 37.6 shows it on the message aaaaaaaaba, ten symbols from the skewed source above that emits a with probability 0.9, coded with the true model.

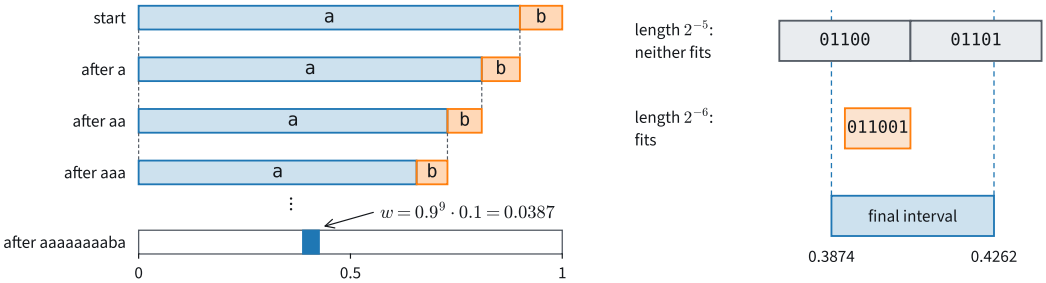


Figure 37.6: Arithmetic coding of the message aaaaaaaaba under the model $q(\mathtt{a}) = 0.9$. (a) Each symbol keeps the slice of the current interval that the model assigns to it; after ten symbols the interval has width $w = 0.9^9 \cdot 0.1 = 0.0387$, the probability of the message. (b) The final interval is named by a dyadic interval inside it. Neither dyadic interval of length 2^{-5} that meets it fits inside, but the one of length 2^{-6} from 0.390625 to 0.40625 does, so the codeword is its address 011001: six bits for a message of 4.69 bits of surprisal.

The coder keeps a current interval, initially $[0, 1)$. To encode the next symbol it splits the current interval into one slice per possible symbol, with widths proportional to the model's conditional probabilities $q(\cdot \mid x_{<i})$, and keeps the slice of the symbol that occurred. Each

step multiplies the width by the probability of the observed symbol, so after the message $x_1, \dots, x_n$ the interval has width

$$w = \prod_{i=1}^{n} q(x_i \mid x_{<i}),$$

the probability that the model assigns to the whole message. The final interval identifies the message: a decoder that replays the model sees which slice the transmitted number falls into, recovers each symbol in turn, and narrows its own interval in step. The decoder must also know n, or the model must include an end-of-message symbol, and we do not count either cost here. What remains is to name a point of the final interval in binary, and the dyadic intervals of the Kraft proof do this.

Any interval of width w contains a dyadic interval of length $2^{-\ell}$ when $\ell = \lceil \log_2(1/w) \rceil + 1$. Then $2^{-\ell} \le w/2$, so the first multiple of $2^{-\ell}$ at or above the interval's left end lies in its left half, and the dyadic interval of length $2^{-\ell}$ starting there ends inside the interval. Panel (b) of Figure 37.6 shows both sides of this: the dyadic intervals of length 2^{-5} stick out of the final interval, while one of length 2^{-6} fits. Since $\ell < \log_2(1/w) + 2$, the codeword length satisfies

$$\ell < \sum_{i=1}^{n} \log_2 \frac{1}{q(x_i \mid x_{<i})} + 2, \tag{37.16}$$

the message's total surprisal under the model plus at most two bits, for a message length n known to the decoder. The rounding now happens once per message, not once per symbol. The codewords of different messages of length n name dyadic intervals inside disjoint message intervals, so by the first half of the Kraft proof they form a prefix code for whole messages. The entropy bound therefore applies to messages as it did to symbols: whatever the model, the expected code length is at least the entropy of the message.

We run the coder on `aaaaaaaaba`. The code narrows the interval, constructs the codeword from the interval's ends, checks that the codeword's dyadic interval lies inside the final interval, and counts what the symbol-by-symbol Shannon code spends on the same message.

```
msg = 'aaaaaaaaba'                    # nine a's, one b
q = {'a': 0.9, 'b': 0.1}              # an i.i.d. model of the source

lo, hi = 0.0, 1.0
for ch in msg:                        # keep the observed symbol's slice
    width = hi - lo
    if ch == 'a':                     # 'a' owns the left 90% of the interval
        hi = lo + width * q['a']
    else:                             # 'b' owns the right 10%
        lo = lo + width * q['a']

ell = math.ceil(-math.log2(hi - lo)) + 1      # codeword length
m = math.ceil(lo * 2 ** ell)                  # first multiple of 2^-ell >= lo
assert lo <= m / 2 ** ell and (m + 1) / 2 ** ell <= hi
surprisal = -math.log2(math.prod(q[ch] for ch in msg))
shannon_bits = sum(math.ceil(math.log2(1 / q[ch])) for ch in msg)
```

```
print(f'final interval [{lo:.4f}, {hi:.4f}), width {hi - lo:.4f}')
print(f'surprisal {surprisal:.2f} bits, codeword {m:0{ell}b} ({ell} bits)')
print(f'symbol-by-symbol Shannon code: {shannon_bits} bits')
```

```
final interval [0.3874, 0.4262), width 0.0387
surprisal 4.69 bits, codeword 011001 (6 bits)
symbol-by-symbol Shannon code: 13 bits
```

The final width 0.0387 is the product $0.9^9 \cdot 0.1$, as the construction promises. The message has 4.69 bits of surprisal and the arithmetic code spends 6 bits on it, the codeword `011001`, while the symbol-by-symbol code spends 13, a whole bit on each of the nine cheap `a`'s. This message has the source's average composition, nine `a`'s and one `b`, so its surprisal equals $10\, H_2(0.9, 0.1) = 4.69$ bits. A message with two `b`'s would cost more.

Floating-point narrowing works only for short messages. The width shrinks by a factor q at every step, and after about a hundred symbols the ends of the interval can no longer be told apart in the 53-bit significand of a double (Exercise 5). Practical coders use integer arithmetic and emit the leading bits as soon as both ends agree on them, which keeps the working interval at a fixed precision (Witten et al. 1987). A message can then be as long as we like, and the two extra bits vanish into the total: a text of n symbols is stored in n times the model's log-loss per symbol, in bits.

37.2.2.2 A Language Model as a Compressor

Nothing in the construction required independent symbols. Any conditional model $q(\cdot \mid x_{<i})$ drives the coder, provided the encoder and the decoder compute the same probabilities, bit for bit in finite precision. An autoregressive language model with an arithmetic coder is therefore a lossless compressor. With a shared tokenizer, model and coder, the code length of a document is its total surprisal under the model plus at most two bits. A file format adds framing, and the model itself must be shared once, which this length does not count.

The smallest model that learns is a counting model. An *adaptive order-m model* predicts each character from the m characters before it, its *context* s, with

$$q(x \mid s) = \frac{c(s, x) + \frac{1}{2}}{c(s) + \frac{k}{2}},$$

where $c(s, x)$ counts how often x has followed s so far, $c(s) = \sum_x c(s, x)$, and k is the alphabet size. The half counts keep every probability positive, so no character ever costs infinitely many bits. After coding a character, the model adds it to the counts. The decoder repeats the same counting on the characters it has decoded, so the encoder never sends any parameters.

We try it on *The Time Machine*. The first cell loads the text, keeps letters and spaces, and defines the adaptive code length in bits per character.

```
fname = d2l.download(d2l.DATA_URL + 'timemachine.txt',
                     sha1_hash='090b5e7e70c295757f55df93cb0a180b9691891a')
with open(fname) as f:
```

```
    text = re.sub('[^A-Za-z]+', ' ', f.read()).lower()   # 26 letters, space

def adaptive_bits(text, m, k):
    """Bits per character of the adaptive order-m model with half counts."""
    counts, totals, bits = {}, {}, 0.0
    for i, ch in enumerate(text):
        s = text[max(0, i - m):i]                # the context
        c = counts.setdefault(s, {})
        bits -= math.log2((c.get(ch, 0) + 0.5) / (totals.get(s, 0) + k / 2))
        c[ch] = c.get(ch, 0) + 1                 # learn after coding
        totals[s] = totals.get(s, 0) + 1
    return bits / len(text)
```

The second cell codes the text with contexts of zero to four characters and compresses it with `zlib` at its highest level. A longer context predicts better once its counts are learned, but it has more counts to learn, and the output shows which effect wins at each order.

```
k, n = len(set(text)), len(text)
freq = [text.count(ch) / n for ch in set(text)]
h_freq = -sum(f * math.log2(f) for f in freq)
print(f'{n} characters over {k} symbols, letter-frequency entropy '
      f'{h_freq:.3f} bits')
bits = [adaptive_bits(text, m, k) for m in range(5)]
for m, b in enumerate(bits):
    print(f'order {m}: {b:.2f} bits/char = {b * math.log(2):.2f} '
          f'nats/char, perplexity 2^bits = {2 ** b:.2f}')
print(f'zlib -9: {8 * len(zlib.compress(text.encode(), 9)) / n:.2f} bits/char')
print(f'order 0 minus the letter-frequency entropy: {bits[0] - h_freq:.4f} '
      f'bits/char, {(bits[0] - h_freq) * n:.0f} bits in all')
```

```
173428 characters over 27 symbols, letter-frequency entropy 4.078 bits
order 0: 4.08 bits/char = 2.83 nats/char, perplexity 2^bits = 16.91
order 1: 3.29 bits/char = 2.28 nats/char, perplexity 2^bits = 9.81
order 2: 2.75 bits/char = 1.90 nats/char, perplexity 2^bits = 6.70
order 3: 2.55 bits/char = 1.77 nats/char, perplexity 2^bits = 5.84
order 4: 2.78 bits/char = 1.92 nats/char, perplexity 2^bits = 6.85
zlib -9: 2.84 bits/char
order 0 minus the letter-frequency entropy: 0.0011 bits/char, 183 bits in all
```

Context lowers the code length from 4.08 bits per character with no context to 2.55 with three characters, which beats `zlib`'s 2.84. A fourth character makes it worse: the four-character model needs 2.78 bits per character. Most of its $27^4 \approx 530{,}000$ possible contexts are seen too rarely to learn their next-character distribution, and the bits spent while each context learns are part of the code length.

With no context there is little to learn. The first line of the output gives the entropy of the text's letter frequencies, 4.078 bits per character, which is what a code without context that knew the frequencies in advance would spend. The order-0 model starts without them, and the last line gives what learning them cost: 0.0011 bits per character, 183 bits for the whole text. Twenty-six frequencies are cheap to learn, and half a million contexts are not. The columns in nats and as a perplexity are read in Section 37.2.4.1.

An adaptive model transmits no parameters, but it codes the early text poorly while it learns.

A pretrained model codes every file well from the start, but its description must be shared once. Delétang et al. (2023) take the second route with a 70-billion-parameter language model. A coder driven by its next-token probabilities compresses ImageNet image patches to 43.4% of their raw size, against 58.5% for PNG, and LibriSpeech audio to 16.4%, against 30.3% for FLAC. These rates leave out the model. At two bytes per parameter it occupies 140 GB, against 1 GB for each data set, and the authors report that with the model counted it no longer competes.

37.2.2.3 Minimum Description Length

The code lengths for *The Time Machine* charged each adaptive model for learning its counts. Without that charge the comparison comes out differently. Take the counts from the whole text first and then score the same text with them: now order 4 beats order 3, and order 5 beats order 4, because a table that has already seen the text can memorize it (Exercise 6 computes these lengths). Judged by the code length of the data alone, the richer of two models never loses. A receiver who does not have the model, however, must first be told it. The *minimum description length* (MDL) principle chooses the model $\mathcal{M}$ with the shortest complete code, model and data together,

$$\widehat{\mathcal{M}} = \operatorname*{argmin}_{\mathcal{M}} \big[L(\mathcal{M}) + L(\mathcal{D} \mid \mathcal{M}) \big], \tag{37.17}$$

where $L(\mathcal{M})$ is the length of the model's description in bits and $L(\mathcal{D} \mid \mathcal{M})$ is the log-loss of the data $\mathcal{D}$ under the model, a code length by the previous subsections. A model must therefore save more bits on the data than it costs to describe. MDL is Occam's razor with simplicity measured in bits. With a code built from a prior, $L(\mathcal{M}) = -\log_2 P(\mathcal{M})$, the total is minus the log prior minus the log likelihood, and MDL becomes maximum a posteriori estimation (Section 36.6.4).

A coin shows the two terms at work. A sequence of $n = 1{,}000$ flips contains 600 heads. The fair coin $\mathcal{M}_0$ has nothing to describe, and it codes each flip in one bit: $L(\mathcal{D} \mid \mathcal{M}_0) = 1{,}000$ bits. The model $\mathcal{M}_1$, a coin with bias θ, must first send θ, and a real number needs a precision. The frequency of heads in n flips varies by an amount of order $1/\sqrt{n}$ from one sequence to the next, so digits of θ finer than that describe noise (Exercise 7). A grid of spacing $1/\sqrt{n}$ has $\sqrt{n}$ points, and naming one of them costs $L(\mathcal{M}_1) = \log_2 \sqrt{n} = \frac{1}{2} \log_2 1{,}000 \approx 5$ bits. The data then cost $L(\mathcal{D} \mid \mathcal{M}_1) \approx 1{,}000\, H_2(0.6, 0.4) = 971$ bits. The total $976 < 1{,}000$ selects the biased coin: it saves 29 bits on the data for 5 bits of parameter. With 520 heads the saving would be 1.2 bits, less than the 5, and MDL keeps the fair coin. Charging $\frac{1}{2} \log_2 n$ bits for every real parameter in this way is known as the *Bayesian information criterion*.

The compressor experiment had already applied this rule. The adaptive code length is a complete description, with the cost of learning the model inside it. For the order-0 model that cost is small. At $\frac{1}{2} \log_2 n \approx 8.7$ bits for each of its 26 free frequencies the criterion predicts about 226 bits, and the output reported 183: the criterion keeps only the leading term of the cost. The order-4 model must learn a separate set of frequencies for every context that occurs, and that cost exceeded what the fourth character saved. The adaptive code length therefore selected order 3.

A parameter count would not have found this. The order-4 model has $27^4 \cdot 26 \approx 14$ million free frequencies, and at 8.7 bits each the criterion would charge it 120 million bits, for a text

that takes 1.4 million bits at eight bits per character. The criterion assumes far more data than parameters. Most of the contexts never occur, and the adaptive code charges nothing for them. For a model with more parameters than data, a neural network included, the adaptive code length is the description length to use.

37.2.3 Entropy Rate and Typical Sequences

So far the model was given, and we asked how long its code is. We now ask how short the code of the best possible model can be.

37.2.3.1 The Entropy Rate

How few bits per character can any code achieve on a source with memory, such as text? For independent symbols the floor is the entropy. Text is not independent: after `q` comes `u`. Let the source $X_1, X_2, \ldots$ be *stationary*, meaning that the joint distribution of any block of symbols does not change when the block is shifted in time.

We need the joint and the conditional entropy of Section 37.1.1.3. For the rest of the section the entropy H of random variables is in bits, and $H_2(p_1, \ldots, p_k)$ remains the entropy in bits of a distribution written out by its probabilities. The joint entropy $H(X_1, \ldots, X_n)$ treats the block as one random variable. The conditional entropy of the next symbol given the past, $H(X_n \mid X_{<n}) = \mathbb{E}\big[-\log_2 p(X_n \mid X_{<n})\big]$, is the expected log-loss of the best possible model of the next symbol, the source itself. The chain rule $H(X, Y) = H(X) + H(Y \mid X)$ of that subsection, applied repeatedly, gives $H(X_1, \ldots, X_n) = \sum_{i=1}^{n} H(X_i \mid X_{<i})$. The *entropy rate* is the limit of the conditional entropy, or equivalently of the entropy per symbol of long blocks,

$$H_{\text{rate}} = \lim_{n\to\infty} \frac{1}{n} H(X_1, \ldots, X_n) = \lim_{n\to\infty} H(X_n \mid X_{<n}). \tag{37.18}$$

By the chain rule the block entropy is a sum of n conditional entropies, so the first expression is their average and the second is the last of them. For a stationary source both limits exist and agree, and $\frac{1}{n} H(X_1, \ldots, X_n)$ decreases monotonically to the limit (Cover and Thomas 1999). We grant these facts. Intuitively, each extra symbol of context can only help prediction, because conditioning never increases entropy (Section 37.1), and the average of a nonincreasing sequence of losses is itself nonincreasing. For independent, identically distributed symbols both limits equal $H(X_1)$.

The smallest source with memory is a two-state chain. Let it emit `a` or `b`, keep its current symbol with probability 0.9, and switch with probability 0.1. Both symbols are equally frequent, so a single symbol has entropy $H(X_1) = 1$ bit. Given the previous symbol, the next is a coin of bias 0.1 from either state, so $H(X_n \mid X_{n-1}) = H_2(0.9, 0.1) = 0.47$ bits, and longer context adds nothing. Memory cuts the floor from 1 bit to 0.47. Exercise 8 does the general two-state chain.

For a stationary source no uniquely decodable code averages fewer than H_{rate} bits per symbol. A code for blocks of n symbols is a symbol code over the alphabet of blocks. The entropy bound, which extends to uniquely decodable codes by McMillan's theorem, says that it spends at least $H(X_1, \ldots, X_n)$ bits per block, and the monotone decrease gives $H(X_1, \ldots, X_n) \geq n H_{\text{rate}}$.

The floor can be reached. An arithmetic coder driven by the source's own conditional probabilities spends less than the surprisal of the message plus two bits, by Equation 37.16, and the expected surprisal of a block under its own distribution is $H(X_1, \dots, X_n)$. Per symbol the coder therefore averages less than $\frac{1}{n} H(X_1, \dots, X_n) + \frac{2}{n}$ bits, which tends to H_{rate}.

Shannon estimated the entropy rate of printed English at about one bit per character (Shannon 1951), far below the $\log_2 27 = 4.75$ bits of uniformly random letters and spaces. The bits per character that a compressor achieves estimate an *upper* bound on the entropy rate, and the 2.55 of the order-3 model is one such estimate. No compressor reports a lower bound. The loss of 1.2 nats per character that the chapter introduction asked about is 1.73 bits. That is below the counting model's 2.55 bits and still well above Shannon's estimate of one bit.

37.2.3.2 Typical Sequences

The entropy bound concerns averages. What does one long message look like, and how many likely messages are there? The answer explains the entropy a second time, as an exponent that counts the likely messages. Nothing later depends on it, and the subsection can be skipped on a first reading. Take independent bits that are 1 with probability 0.3, and messages of length $n = 200$.

The most probable message, all zeros, has probability $0.7^{200} = 2^{-103}$. A message with 60 ones, the expected number, has probability $0.3^{60}\, 0.7^{140} = 2^{-176.3}$, which is 2^{-nH} with $H = H_2(0.3, 0.7) = 0.881$ bits. There are $\binom{200}{60} = 2^{172.2}$ such messages. The number of messages peaks at 100 ones, with $2^{195.9}$ messages, but each of them has probability 2^{-225}, so together they carry only 2^{-29} of the probability. Most messages are unlikely, and the most likely message is unrepresentative.

Write $x_{1:n}$ for a message $x_1, \dots, x_n$. Its surprisal is a sum of independent surprisals, $-\log_2 P(x_{1:n}) = \sum_i -\log_2 p(x_i)$, each with mean $H = H_2(P)$. Per symbol it is therefore a sample mean, and the law of large numbers says that such a mean converges in probability to its expectation:

$$-\frac{1}{n} \log_2 P(X_{1:n}) = \frac{1}{n} \sum_{i=1}^{n} -\log_2 p(X_i) \xrightarrow{P} H. \tag{37.19}$$

So for any fixed $\delta > 0$ a long message has, with probability tending to one, surprisal within $n\delta$ of nH bits. This is the *asymptotic equipartition property* (AEP). The messages it singles out form the *typical set* with tolerance δ bits per symbol,

$$A_\delta = \left\{ x_{1:n} : \left| -\frac{1}{n} \log_2 P(x_{1:n}) - H \right| < \delta \right\}. \tag{37.20}$$

Proposition (AEP). *Let $X_1, \dots, X_n$ be independent draws from a distribution P on a finite alphabet, with entropy H in bits, and let $\delta > 0$. (i) $P(A_\delta) \to 1$ as $n \to \infty$. (ii) Every $x_{1:n} \in A_\delta$ has $2^{-n(H+\delta)} < P(x_{1:n}) < 2^{-n(H-\delta)}$. (iii) $|A_\delta| \le 2^{n(H+\delta)}$, and $|A_\delta| \ge (1-\eta)\, 2^{n(H-\delta)}$ whenever $P(A_\delta) \ge 1 - \eta$.*

Proof. Part (i) is Equation 37.19, and part (ii) rewrites the definition of A_δ. For (iii), add the bounds of (ii) over the typical set: $1 \ge P(A_\delta) > |A_\delta|\, 2^{-n(H+\delta)}$, and $1 - \eta \le P(A_\delta) < |A_\delta|\, 2^{-n(H-\delta)}$. ■

On the exponential scale, then, the typical set has about 2^{nH} members of probability about 2^{-nH} each. "About" means up to a factor $2^{\pm n\delta}$, which is large: at $n = 200$ and $\delta = 0.1$ it is 2^{20}.

The AEP gives Shannon's source-coding theorem (Shannon 1948; Cover and Thomas 1999). To compress, give each typical message an index of $\lceil n(H+\delta) \rceil$ bits and give up on the rest. This fixed-length code spends $H + \delta$ bits per source symbol, up to rounding, and the bits per source symbol are called the *rate* of a code. It fails with probability $P(A_\delta^c) \to 0$. Since δ is arbitrary, every rate above H is achievable. (A flag bit and an uncompressed fallback for the atypical messages make the code lossless at the same rate.) Conversely, a fixed-length code of rate $R < H$ has only 2^{nR} codewords. It can name a set B of at most 2^{nR} messages, and with $\delta < H - R$

$$P(B) \le P(B \cap A_\delta) + P(A_\delta^c) < 2^{nR}\, 2^{-n(H-\delta)} + P(A_\delta^c) \to 0.$$

So every fixed-length code of rate below H fails with probability tending to one. For stationary ergodic sources the same holds with H_{rate} in place of H, a theorem we grant (Cover and Thomas 1999).

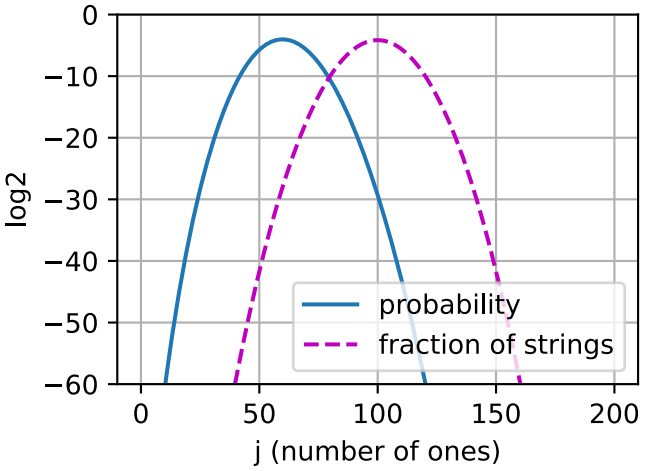
For the Bernoulli source the typical set can be counted *exactly*, because all messages with the same number j of ones share one probability. The cell groups the 2^{200} messages by j and adds up the counts and the probabilities of the groups that are typical at $\delta = 0.1$. The round figure to compare the count with is $2^{nH} = 2^{176.3}$.

```
n, p, delta = 200, 0.3, 0.1
j = np.arange(n + 1)                                  # number of ones
log_fact = np.concatenate([[0.0], np.cumsum(np.log(np.arange(1, n + 1)))])
log2_count = (log_fact[n] - log_fact[j] - log_fact[n - j]) / np.log(2)
bits_per_symbol = -(j * np.log2(p) + (n - j) * np.log2(1 - p)) / n
log2_prob_j = log2_count - n * bits_per_symbol        # log2 P(j ones)
h = -(p * np.log2(p) + (1 - p) * np.log2(1 - p))
typical = np.abs(bits_per_symbol - h) < delta
log2_size = np.logaddexp2.reduce(log2_count[typical])
prob = (2 ** log2_prob_j[typical]).sum()
print(f'H = {h:.3f} bits/symbol, nH = {n * h:.1f} bits')
print(f'typical: 2^{log2_size:.1f} of 2^{n} strings, '
      f'{j[typical].min()} <= j <= {j[typical].max()}')
print(f'probability they carry: {prob:.4f}')
print(f'bounds: {prob:.3f} * 2^{n * (h - delta):.1f} <= |A| '
      f'<= 2^{n * (h + delta):.1f}')
```

```
H = 0.881 bits/symbol, nH = 176.3 bits
typical: 2^188.8 of 2^200 strings, 44 <= j <= 76
probability they carry: 0.9893
bounds: 0.989 * 2^156.3 <= |A| <= 2^196.3
```

The $2^{188.8}$ typical messages, those with 44 to 76 ones, are one in 2^{11} of all messages and carry probability 0.989. The count exceeds 2^{nH} by a factor of several thousand: with $\delta = 0.1$ the set reaches out to 76 ones, and the messages there are its most numerous. It does lie between the proposition's bounds $0.989 \cdot 2^{156.3}$ and $2^{196.3}$, which is all that the proposition promised. The plot shows where the probability and the messages sit.

```
d2l.plot(j, [log2_prob_j, log2_count - n], 'j (number of ones)', 'log2',
         legend=['probability', 'fraction of strings'],
         ylim=[-60, 0])
```



The two curves peak far apart. The probability of having j ones peaks at $j = 60$, where the messages are a fraction 2^{-28} of all strings. The fraction of strings peaks at $j = 100$, where the probability is 2^{-29}. The single most probable message, at $j = 0$, has probability 2^{-103} and falls off the chart. The typical messages are neither the most probable messages nor the most numerous ones. They sit where the number of messages times the probability of each is largest.

The typical set serves proofs, and it makes a poor coder. Indexing it costs 189 bits for every typical message, 13 bits above $nH = 176.3$, and the code fails on one message in a hundred. Arithmetic coding of the same source with the true model never fails and averages under $nH + 2 = 178.3$ bits by Equation 37.16. What the typical set shows is that a long message almost surely has the source's composition, and that about 2^{nH} messages have that composition.

37.2.4 Perplexity

Language models rarely report a file size. They report one number on held-out text, the perplexity, and it is the code length of this section in another unit.

37.2.4.1 Perplexity as a Branching Factor

For a text of n tokens and a model with next-token probabilities $q(\cdot \mid x_{<i})$, the loss per token in nats and its exponential, the *perplexity* (Jelinek et al. 1977), are

$$\mathrm{CE} = -\frac{1}{n}\sum_{i=1}^{n} \log q(x_i \mid x_{<i}), \qquad \mathrm{PPL} = \exp(\mathrm{CE}) = \Big(\prod_{i=1}^{n} q(x_i \mid x_{<i})\Big)^{-1/n}. \tag{37.21}$$

Written without arguments, CE is the loss of the model on this one text, and its expectation

over texts is a cross-entropy between source and model. The second form of the perplexity is an inverse geometric mean of probabilities. Section 8.3.2 introduces perplexity and reads it as an *effective branching factor*: a model that is uniformly undecided among k tokens at every step has perplexity k, and a perfect model of a deterministic sequence has perplexity 1.

The output of Section 37.2.2.2 shows the reading on real text, with characters as tokens. A model that knew nothing would hesitate among all 27 symbols. The letter frequencies alone bring the perplexity down to 16.91, about 17 equally likely symbols, and three characters of context bring it to 5.84: the model is then, on average, about as uncertain of the next character as of the roll of a die. The cell computes perplexity as 2 to the power of the bits per character, and e to the power of the nats gives the same values, because $\text{PPL} = b^{\text{CE}_b}$ for the cross-entropy CE_b in any log base b (Exercise 11).

37.2.4.2 The Perplexity Floor

Perplexity has a floor. For a stationary source it is the exponentiated entropy rate, and it holds only in expectation. The loss is in nats and the entropy rate H_{rate} of Equation 37.18 is in bits, hence the factor $\ln 2$.

Proposition (perplexity floor). *Let the text $X_1, \dots, X_n$ come from a stationary source, let q be any model, and let CE and PPL be the loss and the perplexity Equation 37.21 of this text under q. Then*

$$\mathbb{E}[\text{CE}] \geq H_{\text{rate}} \ln 2, \qquad \mathbb{E}[\text{PPL}] = \mathbb{E}\big[e^{\text{CE}}\big] \geq 2^{H_{\text{rate}}}. \tag{37.22}$$

Proof. Fix a prefix $x_{<i}$. The expected loss of the model on the next token is a cross-entropy between the source's conditional distribution $p(\cdot \mid x_{<i})$ and the model's, and by Gibbs' inequality it is at least the entropy of the source's conditional distribution. Averaging over prefixes gives $\mathbb{E}[-\log q(X_i \mid X_{<i})] \geq \mathbb{E}[-\log p(X_i \mid X_{<i})]$. The log-probability of a text is the sum of its next-token log-probabilities, $\log p(x_{1:n}) = \sum_i \log p(x_i \mid x_{<i})$, so summing over i and dividing by n gives $\mathbb{E}[\text{CE}] \geq \frac{\ln 2}{n} H(X_1, \dots, X_n) \geq H_{\text{rate}} \ln 2$, the last step by the monotone decrease of the block entropy per symbol. Finally, e^x is convex, so its average lies above its value at the average (Jensen's inequality): $\mathbb{E}[e^{\text{CE}}] \geq e^{\mathbb{E}[\text{CE}]} \geq 2^{H_{\text{rate}}}$. ■

No model can be less perplexed, on average, than the source's own randomness allows. If English has Shannon's one bit per character, the floor for a character model is a perplexity of 2, one coin flip per letter, and the counting model's 5.84 is far above it.

The proposition bounds an expectation, and the perplexity of a single short text can fall below the floor. A source that emits `a` with probability 0.9 has $2^H = 1.38$, yet its own true model scores a text of ten symbols below that whenever the text contains no `b`, which happens with probability $0.9^{10} = 35\%$. A long text from this source still lands below 2^H almost half the time, but by a margin that shrinks with its length (Exercise 10). In general, for an ergodic source and the true model, the perplexity of one long text converges to $2^{H_{\text{rate}}}$ as $n \to \infty$, a fact we grant.

37.2.4.3 Perplexity across Tokenizers

Perplexity is computed per token, so it depends on the tokenizer, and two models are comparable by perplexity only under one tokenizer. Section 8.3.2.4 shows this on three tokenizers and repairs it by dividing the total surprisal by the number of bytes of the text. In the

terms of this section the repair is a code length per unit of text. A text of n_{char} characters split into n_{tok} tokens costs $n_{\text{tok}} \log_2 \text{PPL}$ bits, so

$$\text{bpc} = \frac{n_{\text{tok}}}{n_{\text{char}}} \log_2 \text{PPL}, \tag{37.23}$$

and the denominator now depends on the text alone. Where every character is one byte, as in our 27-symbol text, bits per character are the main text's bits per byte.

A perplexity, like every code length in this section, needs a model whose probability for each symbol the encoder and the decoder can both evaluate. Two distributions known only through samples, such as a generator's samples and the data, give no code to measure. Section 37.3 and Section 37.4 build comparisons that work from samples.

37.2.5 Summary

There is a code whose length is a model's log-loss, so a loss is a file size. A symbol code must round every codeword to whole bits, and its best expected length lies between $H_2(P)$ and $H_2(P) + 1$ bits. Arithmetic coding rounds once per message: it stores a message in its total surprisal under any next-symbol model, a language model included, plus at most two bits, so a model Q costs the cross-entropy and the KL divergence is its excess over the true model. Real compressors compete with models on these terms. On *The Time Machine* a counting model with three characters of context beats `zlib` and a fourth character makes it worse, because the cost of learning a model is part of its code length, which is the rule by which MDL chooses between models. For a source with memory no code averages fewer bits per symbol than the entropy rate, and a perplexity is that code length exponentiated: at least $2^{H_{\text{rate}}}$ in expectation, and comparable across models only under one tokenizer.

37.2.6 Exercises

1. **Questions as codewords.** A friend draws one of k outcomes from a known distribution P, and you identify it with yes/no questions. Each question may depend on the answers so far.
    1. Show that every such strategy is a prefix-free code, with the list of answers as the codeword, and that every prefix-free code is a strategy. Use the Kraft inequality Equation 37.12 to show that no strategy identifies every one of k outcomes within m questions unless $k \le 2^m$. How many objects can the parlor game of twenty questions tell apart?
    2. *Guessing* asks "Is it outcome 1?", then "Is it outcome 2?", and so on, most probable outcome first. Find its codeword lengths. Show that it needs 1.75 questions on average for the dyadic source, which is the entropy, and $(k+1)/2 - 1/k$ questions for k equally likely outcomes. For $k = 8$, name a strategy that needs fewer.
2. **McMillan's theorem.** A code is uniquely decodable if every concatenation of codewords parses in only one way.
    1. Show that the code $\{0, 01, 011\}$ is uniquely decodable but not prefix-free, and check that it satisfies the Kraft inequality.
    2. Prove that every uniquely decodable code satisfies the Kraft inequality Equation 37.12. Hint: expand $\left(\sum_i 2^{-\ell_i}\right)^m$ as a sum over all strings of m codewords

of $2^{-(\text{total length})}$, and group the strings by their total length L, where $m \le L \le m\ell_{\max}$. Unique decodability allows at most 2^L strings of total length L, so the sum is at most $m\ell_{\max}$. Take m-th roots and let $m \to \infty$.

3. **Huffman codes.** Huffman's algorithm (Huffman 1952) builds a prefix code by repeatedly merging the two least probable symbols into one symbol of their summed probability, and then assigning 0 and 1 to the two members of each merge.
    1. Run it on $P = (0.6, 0.3, 0.1)$ and on $(0.4, 0.3, 0.2, 0.1)$. Report the codeword lengths and the expected length of each code, next to the Shannon code and H_2.
    2. Prove that Huffman's code has the smallest expected length of all prefix codes. First show that in an optimal code $p_i > p_j$ implies $\ell_i \le \ell_j$, and that the two longest codewords have the same length and can be taken to be siblings in the tree. Then show that merging the two least probable symbols i, j reduces the problem by one symbol with $\mathbb{E}[\ell] = \mathbb{E}[\ell'] + p_i + p_j$, where ℓ' are the lengths of the merged code. Conclude by induction. Hint: exchange two codewords and compare the expected lengths.
    3. For the source that emits `a` with probability 0.9, show that every symbol code spends at least one bit per symbol. Then code *pairs* of symbols as a four-symbol source with Huffman's algorithm and check the text's 0.645 bits per original symbol. Do the same for triples. How does the slack over the entropy compare with the bound $1/n$ in each case?
    4. The guessing strategy of Exercise 1 is the code with lengths $1, 2, \dots, k-1, k-1$. Sort the probabilities, $p_1 \ge \cdots \ge p_k$, and show that Huffman's algorithm can return these lengths whenever $p_i \ge \sum_{j>i} p_j$ for every i, so that guessing is then a best strategy. The source $(0.4, 0.3, 0.2, 0.1)$ violates the condition. Is guessing a best strategy for it?

4. **Where the slack goes.** A code with lengths ℓ_i spends $Z = \sum_i 2^{-\ell_i} \le 1$ of its budget and implies the distribution $q_i = 2^{-\ell_i}/Z$, and Equation 37.14 writes $\mathbb{E}[\ell] - H_2(P) = D_{\mathrm{KL}}^{(2)}(P\|Q) - \log_2 Z$.
    1. The text splits the waste of the Shannon code of $P = (0.6, 0.3, 0.1)$ into 0.005 bits of mismatch and 0.300 of unspent budget. Check the split by computing Z and Q for its lengths $1, 2, 4$, and do the same for Huffman's lengths $1, 2, 2$. Which code wastes bits on unspent budget, and which on mismatch?
    2. Two codes are built from the model $Q = (0.2, 0.5, 0.3)$ and used on data from P: the Shannon code, with lengths $3, 1, 2$, and Huffman's, with lengths $2, 1, 2$. Split the excess of each over $H_2(P)$ into the two terms. The distribution implied by the Shannon code is further from P than Q itself, for which $D_{\mathrm{KL}}^{(2)}(P\|Q) = 0.571$ bits. Which symbol is responsible?
    3. Show that the Shannon code of any P leaves less than one bit unspent, $-\log_2 Z < 1$, and that it leaves nothing unspent only when every probability is a power of $\frac{1}{2}$.

5. code **Exact arithmetic coding.** ⚄ The interval narrowing of Section 37.2.2.1 stores `lo` and `hi` as floating-point numbers. Draw a message of $n = 1000$ symbols, each of which is `a` with probability 0.9 and `b` otherwise.
    1. Run the floating-point narrowing on this message with the model $q(\texttt{a}) = 0.9$. After how many symbols does `hi - lo` first differ from the product of the probabilities by more than 0.1%, and after how many does it become zero? Relate

both answers to the 53-bit significand of a double-precision number.
2. Repeat the narrowing with exact rationals (`fractions.Fraction`). Name a point of the final interval as the text's cell does, write a decoder that replays the model to recover the message, and compare the code length with the surprisal of the message and with $nH_2(0.9, 0.1)$.
3. Encode the same message with the mismatched model $q(\mathtt{a}) = 0.7$. Compare the increase in code length with $nD_{\mathrm{KL}}^{(2)}(P\|Q)$. Why do the two numbers differ for a single message?

6. [extended] **Adaptive context-model compression.** ▣▣ The text's `adaptive_bits` is a complete description length in the sense of Equation 37.17, with the cost of learning the model inside it. By Equation 37.16, an arithmetic coder driven by its predictions spends at most two bits more. Blier and Ollivier (2018) use such prequential codes to measure the description length of deep networks.
   1. Extend the text's experiment to $m = 0, 1, \ldots, 8$. For each m, compare the adaptive code length with the per-character cross-entropy of the order-m model whose counts are taken from the entire text, and with `zlib`.
   2. Explain why the adaptive code length has a minimum over m while the cross-entropy of the fitted model keeps falling. Which order does MDL select? Which term of Equation 37.17 does the cross-entropy of the fitted model leave out?
   3. Design an adaptive model that combines contexts of several orders, for example by falling back to shorter contexts when a long context has few counts. Report a table of bits per character for every model you tried, together with `zlib`, and a paragraph on the design choice that mattered most.

7. **Two-part code for a coin.** ▣ A sequence $\mathcal{D}$ of n coin flips contains j heads, and $\hat{\theta} = j/n$. Hypothesis $\mathcal{M}_0$ is the fair coin: it has no parameter to describe, and $L(\mathcal{D} \mid \mathcal{M}_0) = n$ bits. Hypothesis $\mathcal{M}_1$ is a coin with bias θ. Its encoder sends the point θ_q nearest to $\hat{\theta}$ on a grid of spacing Δ in $(0, 1)$, which costs $L(\mathcal{M}_1) = \log_2(1/\Delta)$ bits, and the decoder knows n and Δ.
   1. Use the decomposition Equation 37.13 to show that $L(\mathcal{D} \mid \mathcal{M}_1) = nH_2(\hat{\theta}, 1-\hat{\theta}) + nD_{\mathrm{KL}}^{(2)}(\hat{\theta}\,\|\,\theta_q)$, where the KL divergence is that of the Bernoulli distributions with these parameters. In nats, this KL divergence is $(\theta_q - \hat{\theta})^2/(2\hat{\theta}(1-\hat{\theta}))$ to second order. Derive the expansion, and show that the total length Equation 37.17 of $\mathcal{M}_1$ exceeds $nH_2(\hat{\theta}, 1-\hat{\theta})$ by at most about $\log_2(1/\Delta) + n\Delta^2/(8\hat{\theta}(1-\hat{\theta})\ln 2)$ bits.
   2. Which spacing Δ minimizes this bound? The decoder must know Δ before it decodes θ_q. Why can the code therefore not use the minimizer? Show that the spacing $\Delta = 1/\sqrt{n}$, which depends on n alone, gives an excess of $\frac{1}{2}\log_2 n + O(1)$ bits whenever $\hat{\theta}$ stays bounded away from 0 and 1.
   3. Take $n = 1000$ and the approximation $L(\mathcal{M}_1) + L(\mathcal{D} \mid \mathcal{M}_1) \approx nH_2(\hat{\theta}, 1-\hat{\theta}) + \frac{1}{2}\log_2 n$ bits. The text showed that $\mathcal{M}_1$ wins at $j = 600$ by 24 bits. Which hypothesis wins at $j = 520$? Approximately how far from $n/2$ must j lie for $\mathcal{M}_1$ to win, and how many standard deviations $\sqrt{n}/2$ of j under the fair coin is that?

8. **Entropy rate of a two-state chain.** A chain on $\{\mathtt{a}, \mathtt{b}\}$ moves from `a` to `b` with probability 0.1 and from `b` to `a` with probability 0.3, and it starts from its stationary distribution.

1. Find the stationary distribution and $H(X_1)$ in bits.
2. Show that $H(X_n \mid X_{n-1}) = \sum_s \pi_s H_2(r_s, 1-r_s)$ bits, where π is the stationary distribution and r_s is the exit probability of state s, and that the entropy rate equals this value. Compute it.
3. Show that $\frac{1}{n}H(X_1, \dots, X_n) = \frac{1}{n}\big(H(X_1) + (n-1)H_{\text{rate}}\big)$ and that it decreases to H_{rate}.
4. Consider two limits. With both exit probabilities equal to $\frac{1}{2}$ the chain is i.i.d. and the rate is 1 bit. With both exit probabilities equal to r, the rate tends to 0 as $r \to 0$, while $H(X_1)$ stays at 1 bit. What does $H(X_1)$ say about how well the chain compresses?
5. code Simulate 10^5 steps of the first chain and code them with the text's `adaptive_bits` at orders 0, 1 and 2. Which orders reach the rate, and what does the order-0 length converge to?

9. code **Typical and high-probability sets.** ⊠ The code of Section 37.2.3.2 declares a binary string of length $n = 200$ from a Bernoulli source with $p = 0.3$ typical when its surprisal per symbol lies within $\delta = 0.1$ bit of the entropy H in bits. It finds $2^{188.8}$ typical strings, inside the bounds $0.989 \cdot 2^{156.3} \le |A_\delta| \le 2^{196.3}$ of the AEP proposition and well above $2^{nH} = 2^{176.3}$.

   1. Which string is the most probable one? Compute its surprisal per symbol and decide whether it is typical. Why does this string not matter for compression?
   2. Let A be the typical set and let B be any set of strings with $P(B) \ge 1-\eta$. Show that $P(A \cap B) \ge 1 - \eta - P(A^c)$, and conclude that $|B| \ge \big(1 - \eta - P(A^c)\big)\, 2^{n(H-\delta)}$.
   3. The smallest set with probability at least 0.9 collects strings in order of decreasing probability. Group the strings by their number j of ones, as the code does, and compute $\frac{1}{n}\log_2$ of the size of this set for $n = 200$, 1000, 5000, and 20000. Compare it with H and with $\frac{1}{n}\log_2 |A_\delta|$ at the same n. Which of the two sets grows like 2^{nH}, and why does the typical set with a fixed δ not?

   Adapted from David Tse, EE376A/Stats376A: Information Theory, Homework 2, Problem 6.

10. **The perplexity floor.**

    1. A Bernoulli source emits `a` with probability 0.9 and `b` otherwise, and its own true model scores a text of n symbols. Write the loss CE of Equation 37.21 as a function of the number j of `b`'s and show that PPL $< 2^H$ exactly when $j < n/10$. The text gives the probability of this event for $n = 10$. Compute it for $n = 100$ and $n = 1000$. What does it tend to, and how does the typical distance of PPL from 2^H change with n?
    2. A leaderboard reports one perplexity on one held-out set. Explain why Equation 37.22 does not bound that number, and why a *lower* reported perplexity, under the same tokenizer, nevertheless always means a smaller total surprisal of that set, which is its arithmetic code length to within two bits.

11. **Perplexity across tokenizers.** Perplexity is computed per token, so it depends on how a text is split into tokens.

    1. Show that b^{CE_b} is the same number for every log base b, where CE_b is the per-token cross-entropy in base-b units.
    2. Two language models are evaluated on the same text of 10^6 characters. The tokenizer of model A splits it into 2.5×10^5 tokens, and A reaches a per-token

perplexity of 20. The tokenizer of model B splits it into 6.7×10^5 tokens, and B reaches a per-token perplexity of 8. Convert both results to bits per character with Equation 37.23. Which model predicts the text better?

3. Take the entropy rate of the text to be one bit per character, the estimate of Shannon (1951) . By Equation 37.22 the expected per-token perplexity is at least 2^H, where H is the entropy rate per *token* in bits. What is the lowest perplexity any model can reach in expectation with the tokenizer of A, and with that of B? How far is each model from its floor?
4. Explain in two sentences why the two perplexities cannot be compared directly while the two values of bits per character can.

37.3 f-Divergences

Every model in this book that was fitted by maximum likelihood was compared with its data through one number, the KL divergence. Section 37.1 ended on its weak point: it is infinite as soon as the model gives zero probability to something that occurs. A classifier rarely notices, because a softmax never outputs an exact zero. A network that turns a low-dimensional noise vector into an image notices at once, because its samples lie on a thin surface in image space that can miss the data altogether. How else can two distributions be compared, and which comparisons can tell a model that is close from one that is far off?

The smallest case already shows what can go wrong. Let the data sit at a single point, $P = \delta_0$, and the model's samples at a single point a distance d away, $Q = \delta_d$. A model at $d = 0.01$ is nearly right and one at $d = 100$ is far off, so a useful loss should tell them apart. Table 37.2 shows what five common comparisons report.

Table 37.2: Five comparisons of the point masses δ_0 and δ_d. The MMD row uses the RBF kernel with length scale 1.

comparison	$d = 0.01$	$d = 1$	$d = 100$
KL divergence $D_{\mathrm{KL}}(P\|Q)$	∞	∞	∞
Jensen–Shannon divergence	$\log 2$	$\log 2$	$\log 2$
total variation distance	1	1	1
maximum mean discrepancy MMD^2	0.0001	0.79	2.00
Wasserstein distance W_1	0.01	1	100

Three of the five rows do not change with d at all. A model trained on any of them receives the same loss at $d = 0.01$ as at $d = 100$, hence no gradient in d. The original generative adversarial network (GAN) of Chapter 16 was in this position: its loss is the Jensen–Shannon row up to constants.

The three rows that do not move are of one kind. KL compares P and Q through the ratio p/q of their densities, and if we keep the ratio and change the function applied to it we obtain a family, the *f-divergences*. Its members share one proof of non-negativity, one way

of being estimated from samples, and one reason for being blind to d. The two rows that move are built without the ratio and wait for Section 37.4.

Once the family is built we ask three questions that the main text keeps meeting. Does it matter which distribution comes first in KL, the data as in maximum likelihood or the model as in variational inference (Section 37.3.2)? What does a small divergence promise to someone who tries to tell P from Q (Section 37.3.3)? And what can be computed when both distributions are known only through samples (Section 37.3.4)? Section 37.2 ended on this last question, and adversarial training rests on its answer.

All logarithms are natural, so KL and Jensen–Shannon are in nats. Throughout, P is the data or the target and Q the model, p and q are their probability mass functions or densities, and $u = p/q$ is their *density ratio*.

```
from scipy import integrate, optimize
```

The table is computed. The next cell treats δ_0 and δ_d as the probability vectors $(1, 0)$ and $(0, 1)$ on the two points $\{0, d\}$ and evaluates each row from its definition. Three of the definitions follow in this section and two in the next.

```
def kl_div(p, q):
    """D_KL(P||Q) on a finite support: 0 log 0 = 0, and inf if q = 0 < p."""
    s = p > 0
    with np.errstate(divide='ignore'):
        return np.sum(p[s] * np.log(p[s] / q[s]))

def five_comparisons(atoms, p, q, ell=1.0):
    """KL, JS, TV, RBF MMD^2 and W1 between pmfs p, q on sorted atoms."""
    m = (p + q) / 2                                  # the mixture for JS
    K = np.exp(-(atoms[:, None] - atoms[None, :]) ** 2 / (2 * ell ** 2))
    F_gap = np.abs(np.cumsum(p) - np.cumsum(q))[:-1]  # |F_P - F_Q| between atoms
    return {'KL': kl_div(p, q), 'JS': (kl_div(p, m) + kl_div(q, m)) / 2,
            'TV': np.abs(p - q).sum() / 2, 'MMD^2': (p - q) @ K @ (p - q),
            'W1': np.sum(F_gap * np.diff(atoms))}

delta_0, delta_d = np.array([1.0, 0.0]), np.array([0.0, 1.0])  # on atoms (0, d)
for d in (0.01, 1.0, 100.0):
    vals = five_comparisons(np.array([0.0, d]), delta_0, delta_d)
    print(f'd = {d:6}: ' + '  '.join(f'{k} {v:.4g}' for k, v in vals.items()))
```

```
d =   0.01: KL inf  JS 0.6931  TV 1  MMD^2 0.0001  W1 0.01
d =    1.0: KL inf  JS 0.6931  TV 1  MMD^2 0.7869  W1 1
d =  100.0: KL inf  JS 0.6931  TV 1  MMD^2 2  W1 100
```

37.3.1 The f-Divergence Family

Before we generalize KL we fix what a comparison of two distributions has to satisfy. The family then comes from asking what the proof that KL is never negative used.

37.3.1.1 Divergences and Metrics

The least that a "distance between distributions" must do is to be zero for equal distributions and positive for unequal ones. A **divergence** is a function of two distributions with this property:

$$D(P,Q) \geq 0, \qquad D(P,Q) = 0 \quad \text{if and only if} \quad P = Q.$$

Nothing else is required. A **metric**, like the distance $\|\mathbf{x} - \mathbf{y}\|$ between two vectors (Section 1.3), must in addition be symmetric, $D(P,Q) = D(Q,P)$, and satisfy the triangle inequality, $D(P,R) \leq D(P,Q) + D(Q,R)$. The KL divergence is a divergence: Gibbs' inequality (Section 37.1) says that $D_{\text{KL}}(P\|Q) \geq 0$, with equality only for $P = Q$. It fails both extra conditions, and two Gaussians on the line show the failures with numbers. That section gave the KL divergence between them,

$$D_{\text{KL}}(\mathcal{N}(\mu_1,\sigma_1^2)\,\|\,\mathcal{N}(\mu_2,\sigma_2^2)) = \log\frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1-\mu_2)^2}{2\sigma_2^2} - \frac{1}{2}. \tag{37.24}$$

For $P = \mathcal{N}(0,1)$ and $Q = \mathcal{N}(0,4)$ it gave $D_{\text{KL}}(P\|Q) \approx 0.318$ nats but $D_{\text{KL}}(Q\|P) \approx 0.807$ nats, so KL is not symmetric. The triangle inequality fails for a different reason. For unit-variance Gaussians centered at 0, 1 and 2 the formula gives $D_{\text{KL}} = \frac{1}{2}$ for each adjacent pair but 2 for the outer pair, and $2 > \frac{1}{2} + \frac{1}{2}$. KL grows like the *square* of the distance between the means, and squared distances do not satisfy the triangle inequality.

Neither failure needs repair. Minimizing a divergence uses neither property, and the asymmetry turns out to be a modeling decision (Section 37.3.2). A proof that has to chain two comparisons, from P to Q and from Q to R, does need the triangle inequality, and for it the family we are about to build also contains metrics.

37.3.1.2 The f-Divergence Template

Gibbs' inequality is the reason KL is a divergence. Which properties of KL did its proof use, and what else has them?

First, we write KL as an average under Q of a function of the ratio $u(x) = p(x)/q(x)$:

$$D_{\text{KL}}(P\|Q) = \sum_x p(x)\log\frac{p(x)}{q(x)} = \sum_x q(x)\,\frac{p(x)}{q(x)}\log\frac{p(x)}{q(x)} = \mathbb{E}_{x\sim Q}\big[u(x)\log u(x)\big].$$

KL weighs each outcome by Q and gives it the penalty $u\log u$. Our running pair is the one that Section 37.1 carried through its examples, $P = (0.6, 0.3, 0.1)$ and $Q = (0.2, 0.5, 0.3)$. Its ratios are $u = 3$, 0.6 and $\frac{1}{3}$, with weights 0.2, 0.5 and 0.3:

$$0.2\,(3\log 3) + 0.5\,(0.6\log 0.6) + 0.3\,(\tfrac{1}{3}\log\tfrac{1}{3}) = 0.659 - 0.153 - 0.110 = 0.396 \text{ nats},$$

the KL divergence computed in that section.

Second, we look at the proof. Section 37.1 applied Jensen's inequality once, to $-\log(q/p)$ under P. Run instead under Q, on the function $u \log u$, the same one-step argument needs only two facts about that function: it is convex, and it is zero at $u = 1$. Third, we replace $u \log u$ by any function with these two properties. Let $f : (0, \infty) \to \mathbb{R}$ be convex with $f(1) = 0$, and set $f(0) = \lim_{u \downarrow 0} f(u)$, which may be $+\infty$. Assume for now that $p(x) = 0$ wherever $q(x) = 0$, which we write $P \ll Q$, so that an average under Q sees all of P. The **f-divergence** of P with respect to Q is (Csiszár 1967)

$$D_f(P\|Q) = \mathbb{E}_{x\sim Q}\left[f\left(\frac{p(x)}{q(x)}\right)\right]. \tag{37.25}$$

The ratio says, point by point, how much P over- or under-represents x relative to Q, and f sets the penalty, which is zero at $u = 1$. We call f the *divergence function*. The literature calls it the generator of the divergence, and the code below keeps the functions in a dictionary named `generators`. In the text, as in Section 16.2, a generator is always a network that produces samples.

One more definition is needed to say when the divergence vanishes. We call f *strictly convex at* 1 if some line through $(1, 0)$ lies strictly below f away from 1: $f(u) > c\,(u - 1)$ for all $u \neq 1$ and some slope c.

Proposition (non-negativity). *For any convex f with $f(1) = 0$ and $P \ll Q$,*

$$D_f(P\|Q) \geq 0,$$

and if f is strictly convex at 1, then $D_f(P\|Q) = 0$ if and only if $P = Q$.

Proof. Jensen's inequality (Section 35.3.3.1) says that $\mathbb{E}[f(U)] \geq f(\mathbb{E}[U])$ for a convex f and a random variable U. Apply it to $U = u(x)$ with $x \sim Q$. Everything rests on one fact: the ratio averages to one under Q, $\mathbb{E}_Q[u] = \sum_x q(x)\, p(x)/q(x) = \sum_x p(x) = 1$. Hence

$$D_f(P\|Q) = \mathbb{E}_Q[f(u)] \geq f(\mathbb{E}_Q[u]) = f(1) = 0.$$

For the equality case, take the line $c\,(u-1)$ below f. Then $D_f(P\|Q) \geq \mathbb{E}_Q[c\,(u-1)] = 0$, and the gap is the average of $f(u) - c\,(u-1)$, which is positive wherever $u \neq 1$. So $D_f = 0$ forces $u(x) = 1$ at every x with $q(x) > 0$, that is, $P = Q$. ■

The fact that carried the proof, $\mathbb{E}_Q[u] = 1$, has a second consequence that we shall use several times. Adding a multiple $c\,(u - 1)$ to f changes no divergence, because $\mathbb{E}_Q[c\,(u - 1)] = 0$. The functions $u \log u$ and $u \log u - u + 1$, for instance, both define KL.

The definition assumed $P \ll Q$, and the point masses of the opening violate it. A generative model begins training in just this regime, with its samples and the data in disjoint regions, so we must say what D_f means there. Outcomes with $p(x) = q(x) = 0$ contribute nothing. Where $p(x) = 0 < q(x)$, the term is $q(x)\, f(0)$, as the definition already says. Where $q(x) = 0 < p(x)$ the ratio is infinite, and we take the term to be the limit of $q\, f(p/q)$ as $q \downarrow 0$. With $t = p/q$ the term is $p\, f(t)/t$, so the limit is $p(x)\, f'(\infty)$ with $f'(\infty) := \lim_{t\to\infty} f(t)/t$. For a convex f the ratio $f(t)/t$ and the slope $f'(t)$ have the same limit, hence the notation. Both $f(0)$ and $f'(\infty)$ may be $+\infty$.

On the point masses δ_0 and δ_d with $d \neq 0$, the outcome 0 has $q = 0 < p = 1$ and the outcome d has $p = 0 < q = 1$, so

$$D_f(\delta_0 \| \delta_d) = f(0) + f'(\infty) \qquad \text{for every } d \neq 0.$$

The distance d does not appear on the right, and this says more than the table did: every f-divergence is constant on disjoint supports. It cannot tell a near miss from a distant one, and a model trained on it receives no gradient from the distance between the supports. For KL, $u \log u$ has $f(0) = 0$ and $f'(\infty) = \lim_{t\to\infty} \log t = \infty$, the first row of Table 37.2. The other two rows that do not move follow as soon as we have their functions.

37.3.1.3 Common Divergence Functions

Each row of Table 37.3 is one choice of f, and Figure 37.7 plots the curves. The column $f''(1)$, the curvature at $u = 1$, is needed in the next subsection.

Table 37.3: Common divergence functions. Each row gives the function f of one divergence, its curvature at $u = 1$, and whether the divergence is symmetric and whether it, or its square root, is a metric.

Divergence	$f(u)$	$f''(1)$	Symmetric?	Metric?
Kullback–Leibler (forward)	$u \log u$	1	no	no
reverse KL	$-\log u$	1	no	no
Pearson χ^2	$(u-1)^2$	2	no	no
squared Hellinger d_{H}^2	$(\sqrt{u}-1)^2$	$\frac{1}{2}$	yes	d_{H} is a metric
total variation	$\frac{1}{2}\lvert u-1\rvert$	none (corner)	yes	yes
Jensen–Shannon	$\frac{u}{2}\log u - \frac{u+1}{2}\log\frac{u+1}{2}$	$\frac{1}{4}$	yes	$\sqrt{\mathrm{JS}}$ is a metric
α-divergence ($\alpha \neq 0, 1$)	$\dfrac{u^\alpha - \alpha u + \alpha - 1}{\alpha(\alpha-1)}$	1	only at $\alpha = \frac{1}{2}$	square root at $\alpha = \frac{1}{2}$

Reverse KL uses $f(u) = -\log u$, since $\mathbb{E}_Q[-\log(p/q)] = \mathbb{E}_Q[\log(q/p)] = D_{\mathrm{KL}}(Q\|P)$. The asymmetry of KL is therefore a choice of f, and the two directions are two rows of the table. Exercise 1a gives the general rule for swapping arguments.

Total variation uses $f(u) = \frac{1}{2}|u-1|$, and $\sum_x q(x)\,\frac{1}{2}|p(x)/q(x)-1| = \frac{1}{2}\sum_x |p(x)-q(x)|$. We write it $\mathrm{TV}(P, Q)$. It is half the ℓ_1 distance between the probability vectors, and on the running pair $\mathrm{TV} = \frac{1}{2}(0.4 + 0.2 + 0.2) = 0.4$. What this number means for anyone who tries to tell P from Q is the subject of Section 37.3.3. On the point masses its function gives $f(0) + f'(\infty) = \frac{1}{2} + \frac{1}{2} = 1$, the third row of Table 37.2.

Pearson's χ^2 divergence uses the parabola $f(u) = (u-1)^2$, so that $\chi^2(P\|Q) = \sum_x q(x)\,(p(x)/q(x)-1)^2 = \sum_x (p(x)-q(x))^2/q(x)$. On the running pair it is $0.16/0.2 + 0.04/0.5 + 0.04/0.3 = 1.0133$. A disagreement counts for more where Q is small, and the first outcome, with $q = 0.2$, contributes 0.8 of the total.

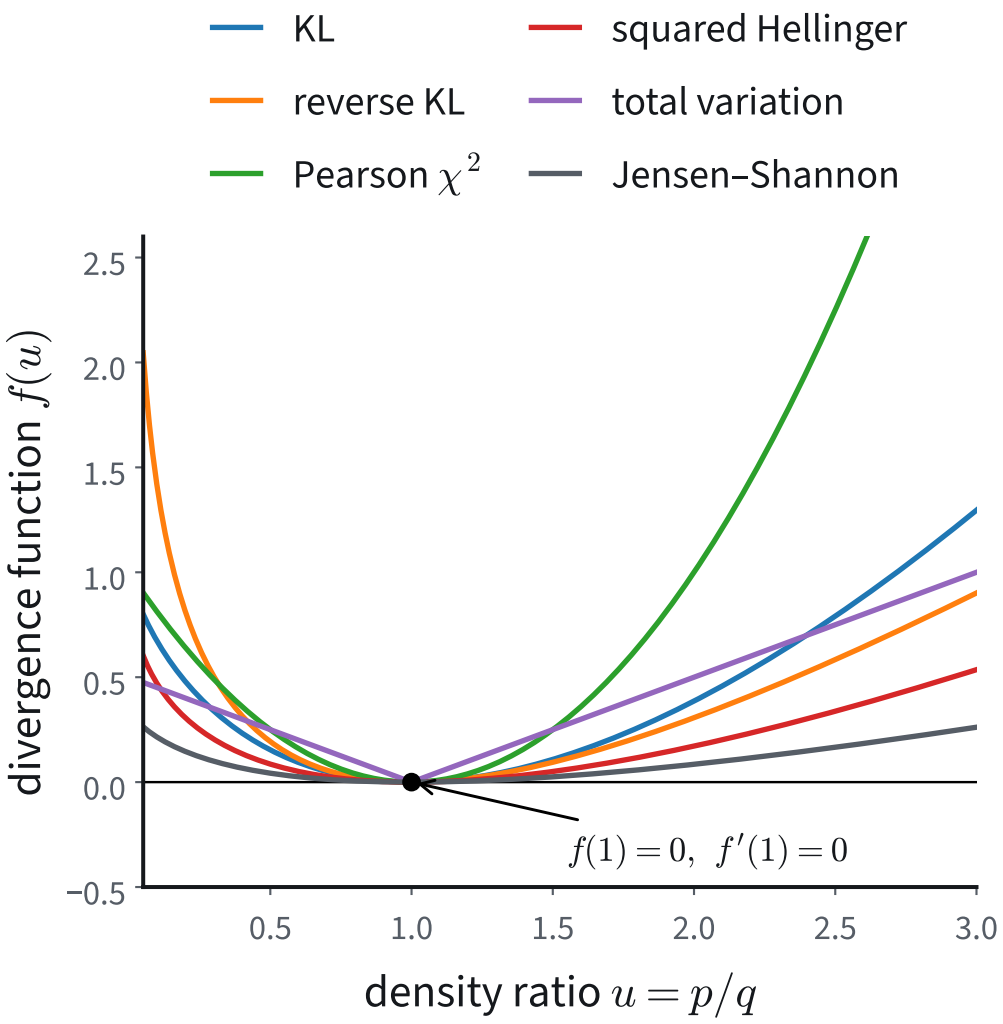


Figure 37.7: Six divergence functions, the smooth ones shifted by the multiple $f'(1)(u-1)$, which changes no divergence. Every curve is non-negative and zero at $u = 1$. The smooth ones are tangent to zero there, with the curvatures $1, 1, 2, \frac{1}{2}, \frac{1}{4}$ of the table, and total variation has a corner instead. Far from $u = 1$ the functions penalize over- and under-representation very differently: reverse KL diverges as $u \to 0$, KL and χ^2 grow superlinearly, and total variation grows linearly.

The **Jensen–Shannon divergence** (Lin 1991) symmetrizes KL through the even mixture $M = \frac{1}{2}(P + Q)$, with mass function m:

$$\mathrm{JS}(P, Q) = \frac{1}{2} D_{\mathrm{KL}}(P \,\|\, M) + \frac{1}{2} D_{\mathrm{KL}}(Q \,\|\, M). \tag{37.26}$$

Each distribution is compared with the mixture, which has mass wherever either one does. Since $p/m = 2p/(p+q) \le 2$, each KL term is at most $\log 2$. So JS is symmetric and bounded, $0 \le \mathrm{JS}(P, Q) \le \log 2$, and it reaches $\log 2$ only when P and Q have disjoint supports, where $p/m = 2$ wherever $p > 0$. Its square root is a metric (Endres and Schindelin 2003). Expanding Equation 37.26 in the ratio u gives the function in the table (Exercise 1c). That function has $f(0) = f'(\infty) = \frac{1}{2}\log 2$, and the sum is the second row of Table 37.2. The one identity $D_f(\delta_0 \| \delta_d) = f(0) + f'(\infty)$ has now accounted for all three rows that do not move.

The **squared Hellinger divergence** is $d_{\mathrm{H}}^2(P, Q) = \sum_x \left(\sqrt{p(x)} - \sqrt{q(x)}\right)^2$. So d_{H} is the Euclidean distance between the vectors $\sqrt{p}$ and $\sqrt{q}$, which is why it is a metric (Exercise 2b).

The table's last row is a family within the family, the α**-divergences** (Amari 2016), with

$$f_\alpha(u) = \frac{u^\alpha - \alpha u + \alpha - 1}{\alpha(\alpha - 1)}, \tag{37.27}$$

which is convex with $f_\alpha(1) = 0$ and curvature 1 at $u = 1$ for every $\alpha \neq 0, 1$. As $\alpha \to 1$ it tends to $u \log u - u + 1$ and as $\alpha \to 0$ to $-\log u + u - 1$. A multiple of $u - 1$ changes no divergence, so the two ends are forward and reverse KL. Halfway, at $\alpha = \frac{1}{2}$, the function is $2(\sqrt{u} - 1)^2$, twice squared Hellinger, and $\alpha = 2$ gives half of χ^2. One parameter thus leads from reverse KL through Hellinger to forward KL, and on to χ^2 (Exercise 1d relates the family to Rényi's divergence).

We now evaluate the first six rows on the running pair, in both argument orders. KL should print 0.396 and 0.365 nats, the two values of Section 37.1, and TV and $\chi^2(P\|Q)$ the values just found by hand.

```
def f_divergence(f, p, q):
    """D_f(P||Q) = sum_x q(x) f(p(x)/q(x)); nats for KL/JS, else unitless."""
    return np.sum(q * f(p / q))

generators = {
    'KL':          lambda u: u * np.log(u),
    'reverse KL':  lambda u: -np.log(u),
    'chi^2':       lambda u: (u - 1) ** 2,
    'Hellinger^2': lambda u: (np.sqrt(u) - 1) ** 2,
    'TV':          lambda u: 0.5 * np.abs(u - 1),
    'JS':          lambda u: u / 2 * np.log(u) - (u + 1) / 2 * np.log((u + 1) / 2),
}

p = np.array([0.6, 0.3, 0.1])
q = np.array([0.2, 0.5, 0.3])
for name, f in generators.items():
```

```
        print(f'{name:12s} D_f(P||Q) = {f_divergence(f, p, q):.4f}   '
              f'D_f(Q||P) = {f_divergence(f, q, p):.4f}')
```

```
KL           D_f(P||Q) = 0.3961   D_f(Q||P) = 0.3653
reverse KL   D_f(P||Q) = 0.3653   D_f(Q||P) = 0.3961
chi^2        D_f(P||Q) = 1.0133   D_f(Q||P) = 0.8000
Hellinger^2  D_f(P||Q) = 0.1862   D_f(Q||P) = 0.1862
TV           D_f(P||Q) = 0.4000   D_f(Q||P) = 0.4000
JS           D_f(P||Q) = 0.0911   D_f(Q||P) = 0.0911
```

The KL row is asymmetric, and the reverse-KL row is the KL row with its columns swapped. The χ^2 row is asymmetric too (1.0133 against 0.8000). Squared Hellinger (0.1862), TV (0.4) and JS are symmetric, and JS, at 0.0911 nats, sits far below its ceiling of $\log 2 \approx 0.693$. Six divergences have looked at one pair of distributions and returned values from 0.09 to 1.01. Multiply KL by 2, squared Hellinger by 4 and JS by 8, however, and the three products are 0.79, 0.74 and 0.73. Why should three different functions of the ratio nearly agree, and where do the factors come from?

37.3.1.4 The Local Quadratic

The functions in Figure 37.7 differ greatly away from $u = 1$. Near $u = 1$ they turn out to be one function up to a constant, which is perhaps the most surprising fact about the family. The rest of the section does not depend on this subsection, and it can be skipped on a first reading.

A multiple of $u-1$ changes no divergence, so the value $f(1) = 0$ and the slope $f'(1)$ carry no information, and the first property that distinguishes one divergence function from another is the curvature $f''(1)$. Near $u = 1$ every smooth f looks like the parabola $\frac{1}{2}f''(1)(u-1)^2$, and the average of $(u-1)^2$ under Q is the χ^2 divergence, $\mathbb{E}_Q[(u-1)^2] = \sum_x (p(x) - q(x))^2/q(x)$. Let $r = \max_x |u(x) - 1|$ measure how far the ratio strays from 1.

Proposition (local quadratic). *Let $P \ll Q$, let $0 < r < 1$, let f be three times differentiable on $[1-r, 1+r]$ with $|f'''| \le B$ there, and let $|u(x) - 1| \le r$ for every x with $q(x) > 0$. Then*

$$\left| D_f(P\|Q) - \frac{f''(1)}{2}\,\chi^2(P\|Q) \right| \le \frac{Br}{6}\,\chi^2(P\|Q). \tag{37.28}$$

Proof. Taylor's theorem with remainder gives $f(u) = f'(1)(u-1) + \frac{1}{2}f''(1)(u-1)^2 + R(u)$ with $|R(u)| \le B|u-1|^3/6$, since $f(1) = 0$. Take $\mathbb{E}_Q$ term by term. The linear term vanishes because $\mathbb{E}_Q[u-1] = 0$, and with it the only first-order difference between one f and another. The quadratic term is $\frac{1}{2}f''(1)\,\chi^2(P\|Q)$. The remainder obeys $\mathbb{E}_Q|R| \le \frac{B}{6}\,\mathbb{E}_Q|u-1|^3 \le \frac{Br}{6}\,\mathbb{E}_Q(u-1)^2 = \frac{Br}{6}\,\chi^2$. ■

So $D_f \approx \frac{1}{2}f''(1)\,\chi^2$ with a relative error proportional to r: close to Q, all smooth f-divergences are multiples of one another. The factors 2, 4 and 8 that we applied to KL, squared Hellinger and JS are $2/f''(1)$, and the number that the three products approximate is $\chi^2(P\|Q) = 1.013$. They fall about a quarter short of it, which does not contradict the proposition: the running pair has $r = 2$, because its largest ratio is 3, and the hypothesis asks for $r < 1$. To see the approximation take hold, we move the running pair's P toward Q along the segment $P_t = Q + t\,(P - Q)$, which multiplies every $u-1$ by t, so that $r = 2t$. For

four divergences we print the ratio $D_f(P_t\|Q)/(\frac{1}{2}f''(1)\,\chi^2(P_t\|Q))$, which should approach 1 at a rate proportional to t.

```
curvature = {'KL': 1.0, 'reverse KL': 1.0, 'Hellinger^2': 0.5, 'JS': 0.25}
for t in (0.5, 0.1, 0.01):
    p_t = q + t * (p - q)                    # moves P_t toward Q as t -> 0
    chi2 = f_divergence(generators['chi^2'], p_t, q)
    ratios = [f_divergence(generators[name], p_t, q) / (c / 2 * chi2)
              for name, c in curvature.items()]
    print(f't = {t:4}: ' + '  '.join(f'{name} {r:.3f}'
                                     for name, r in zip(curvature, ratios)))
```

```
t =  0.5: KL 0.844  reverse KL 0.747  Hellinger^2 0.789  JS 0.783
t =  0.1: KL 0.956  reverse KL 0.917  Hellinger^2 0.936  JS 0.935
t = 0.01: KL 0.995  reverse KL 0.990  Hellinger^2 0.993  JS 0.993
```

At $t = 0.5$ the four divergences are about 75% to 85% of their quadratics, at $t = 0.1$ more than 91%, and at $t = 0.01$ more than 99%. Between the last two rows t falls tenfold and so, nearly, does each shortfall, as the proposition's factor r predicts. The bound itself is a worst case. For KL at $t = 0.1$ we have $r = 0.2$ and $|f'''(u)| = 1/u^2 \le 1/0.8^2$, so it allows a relative error of about 10%, and the cell shows 4%. Smooth f-divergences disagree only about distant distributions, which is the regime early in training, when the model is far from the data.

On a parametric family the common quadratic is the Fisher information $\mathbf{I}(\theta) = \mathbb{E}[\nabla_\theta \log q_\theta \, \nabla_\theta \log q_\theta^\top]$ of Section 36.6. For a small step $\mathbf{h}$ the ratio of $q_{\theta+\mathbf{h}}$ to q_θ is $u \approx 1 + \mathbf{h}^\top \nabla_\theta \log q_\theta$, so $\chi^2(q_{\theta+\mathbf{h}}\|q_\theta) = \mathbb{E}[(u-1)^2] \approx \mathbf{h}^\top \mathbf{I}(\theta)\,\mathbf{h}$, and every smooth D_f has Hessian $f''(1)\,\mathbf{I}(\theta)$ at $\mathbf{h} = \mathbf{0}$. The natural gradient of Section 35.5, which measures steps by KL, would therefore be the same method under any other smooth f-divergence. Total variation is the exception. Its divergence function has a corner at $u = 1$ and no curvature, and Exercise 4b shows what that changes when P approaches Q.

37.3.2 Forward and Reverse KL

Maximum likelihood, the criterion behind most models in this book, minimizes KL with the data in the first argument. Variational inference minimizes KL with the model in the first argument. The two orders are the two KL rows of Table 37.3, and when we fit a model Q_θ, with density q_θ, to a target P that it cannot match, they ask for different things:

- **Forward KL**, $D_{\mathrm{KL}}(P\,\|\,Q_\theta) = \mathbb{E}_P[\log(p/q_\theta)]$, averages over the *truth*. If $q_\theta(x) \to 0$ while $p(x) > 0$, the divergence blows up. Forward KL is *zero-avoiding*, or *mode-covering*: the model must cover every mode of the data, that is, every region where p is concentrated, even if that smears mass over regions P never visits.
- **Reverse KL**, $D_{\mathrm{KL}}(Q_\theta\,\|\,P) = \mathbb{E}_{Q_\theta}[\log(q_\theta/p)]$, averages over the *model*. The model is punished for putting mass where the truth has none, while modes it never visits leave the objective untouched. Reverse KL is *zero-forcing*, or *mode-seeking*: the model concentrates on the mass it can explain.

Which direction to use is seldom a free choice, because each needs something different. Forward KL needs samples from P and the density of the model, and minimizing it is maximum likelihood: Section 36.6.2.1 shows that the average negative log-likelihood is the cross-entropy from the empirical distribution to the model, which differs from forward KL by a

constant. Reverse KL needs samples from the model and the density of the target, and a density known only up to its normalizing constant will do. Variational inference (Section 36.7) is in that position. Its target P is a posterior whose normalizing constant is unknown, and it minimizes $D_{\mathrm{KL}}(Q_\theta \| P)$ over approximations Q_θ.

Where samples and densities are available on both sides, the direction is a decision. A student network distilled from a teacher (Section 26.1) spreads over everything the teacher might say when the teacher comes first, and keeps to the teacher's likeliest outputs when the student does. The KL penalty of Section 15.3 puts the policy first, which keeps it away from actions that the reference policy never takes.

When the model family contains the target, both directions agree on the answer. They differ for a *misspecified* family, and the simplest instance is a single Gaussian fitted to a bimodal target. The forward fit needs no computation. Forward KL is a constant minus $\mathbb{E}_P[\log q(X)]$, so its minimizer is the maximum-likelihood Gaussian with P in place of the empirical distribution, and maximum likelihood in an exponential family matches the expected sufficient statistics (Section 36.4.3.2): $\mu^* = \mathbb{E}_P[X]$ and $(\sigma^*)^2 = \mathrm{Var}_P(X)$, provided the variance is finite. The reverse direction has no such closed form, and it can have one local optimum per mode.

For the mixture $P = 0.7\,\mathcal{N}(-2, 0.6^2) + 0.3\,\mathcal{N}(2, 0.6^2)$ we can therefore predict the forward fit. Its mean is $0.7(-2) + 0.3(2) = -0.8$. Its variance is the variance within a component plus the variance of the component means, which for weights w_1, w_2 and means Δ apart is $w_1 w_2 \Delta^2$: $0.36 + 0.7 \cdot 0.3 \cdot 4^2 = 3.72$. The reverse fit we leave to a derivative-free optimizer over $\theta = (\mu, \log \sigma)$, started once at the origin and once near the minor mode, with each KL evaluated by quadrature on a grid. For every fit the cell prints the mean, the standard deviation and both divergences.

```
w, mus, sigmas = np.array([0.7, 0.3]), np.array([-2.0, 2.0]), np.array([0.6, 0.6])
x = np.linspace(-8.0, 8.0, 4001)

def log_components(x, w, mus, sigmas):
    """log(w_k N(x; mu_k, sigma_k^2)) for every component k; shape (k, len(x))."""
    return (np.log(w)[:, None] - 0.5 * np.log(2 * np.pi * sigmas[:, None] ** 2)
            - (x - mus[:, None]) ** 2 / (2 * sigmas[:, None] ** 2))

def log_mix(x, w, mus, sigmas):
    """Log-density of the mixture sum_k w_k N(mu_k, sigma_k^2), computed stably."""
    c = log_components(x, w, mus, sigmas)
    c_max = c.max(axis=0)
    return c_max + np.log(np.exp(c - c_max).sum(axis=0))

log_p = lambda x: log_mix(x, w, mus, sigmas)

def log_q(x, theta):
    mu, log_sigma = theta
    return (-0.5 * np.log(2 * np.pi) - log_sigma
            - (x - mu) ** 2 / (2 * np.exp(2 * log_sigma)))

def kl(log_a, log_b, x):
    """D_KL(A||B) by quadrature on the grid x; in nats."""
    return integrate.trapezoid(np.exp(log_a) * (log_a - log_b), x)
```

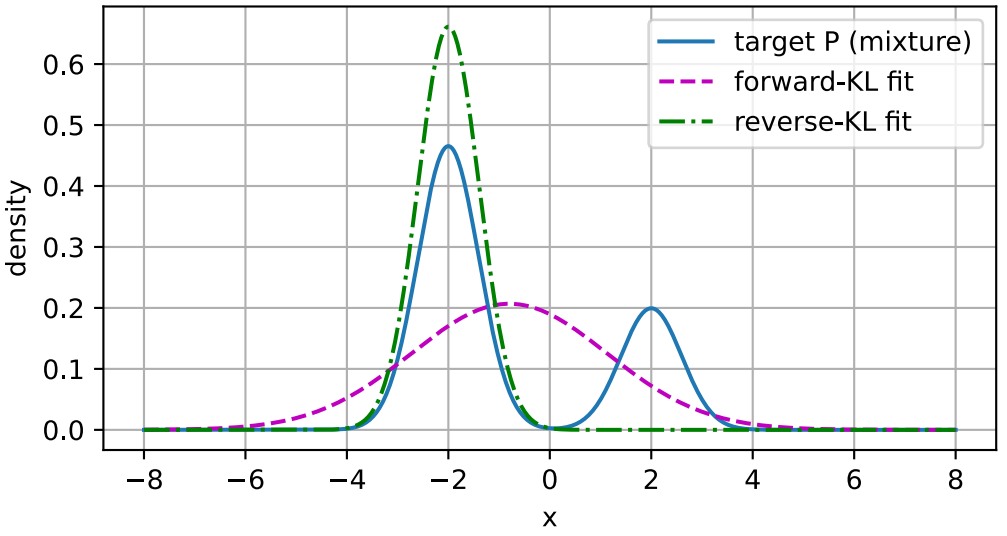
```
fwd = lambda th: kl(log_p(x), log_q(x, th), x)     # D_KL(P || Q_theta)
rev = lambda th: kl(log_q(x, th), log_p(x), x)     # D_KL(Q_theta || P)

th_fwd = optimize.minimize(fwd, x0=[0.0, 0.0], method='Nelder-Mead').x
th_rev = optimize.minimize(rev, x0=[0.0, 0.0], method='Nelder-Mead').x
th_rev2 = optimize.minimize(rev, x0=[2.5, -0.5], method='Nelder-Mead').x

print(f'{"fit":26s}{"mu":>8s}{"sigma":>7s}'
      f'{"forward KL":>12s}{"reverse KL":>12s}')
for name, th in [('forward-KL fit', th_fwd), ('reverse-KL fit', th_rev),
                 ('reverse-KL fit, 2nd start', th_rev2)]:
    print(f'{name:26s}{th[0]:+8.3f}{np.exp(th[1]):7.3f}'
          f'{fwd(th):12.3f}{rev(th):12.3f}')

d2l.plot(x, [np.exp(log_p(x)), np.exp(log_q(x, th_fwd)),
             np.exp(log_q(x, th_rev))], 'x', 'density',
         legend=['target P (mixture)', 'forward-KL fit', 'reverse-KL fit'],
         figsize=(6, 3))
```

```
fit                           mu  sigma  forward KL  reverse KL
forward-KL fit            -0.800  1.929       0.558       1.274
reverse-KL fit            -1.998  0.603       5.981       0.356
reverse-KL fit, 2nd start +1.995  0.607      14.534       1.202
```



The forward fit lands on the prediction, $\mu = -0.800$ and $\sigma = 1.929 \approx \sqrt{3.72}$. It is a broad Gaussian draped across both modes, with substantial mass in the valley between them where P has almost none. The reverse fit lands at $\mu = -1.998$, $\sigma = 0.603$: it matches the dominant component and assigns little mass to the minor one. Its reverse KL, 0.356 nats, is close to $\log(1/0.7) \approx 0.357$, the divergence of treating the 70% component as the whole distribution. Started near the minor mode, the optimizer converges instead to $\mu = +1.995$, where the reverse KL is 1.202 nats, close to $\log(1/0.3) \approx 1.204$. Which optimum a variational method finds therefore depends on its initialization.

Reverse KL is an average under the model, so a mode that the model never visits is never inspected. All that the objective registers is that the mode it does visit carries weight 0.7, or 0.3, in place of 1. This is why variational posteriors tend to be too narrow: here the reverse fit has width 0.60 where the target has standard deviation 1.93.

The last two columns show how differently the two criteria judge the same omission. Forward KL, the criterion of maximum likelihood, puts the fit on the dominant mode at 5.98 nats, more than ten times the 0.558 of the broad fit, and the fit on the minor mode at 14.5. In both, the dropped mode lies more than six of the model's standard deviations from its mean, where $\log q$ is very negative (Exercise 3c). Reverse KL ranks the first two fits the other way, 0.356 nats against 1.27: what it penalizes is the broad fit's mass in the valley, where P has almost none.

The conventions for vanishing densities say the same in the limit. The function $u \log u$ has $f'(\infty) = \infty$, which gives any mass of P where Q has none an infinite penalty, and $f(0) = 0$, so that mass of Q where P has none adds no term at all. For $-\log u$ the two values are exchanged.

37.3.3 Total Variation, Pinsker's Inequality, and Data Processing

A divergence of 0.4 says little until we know what it permits and what it forbids. For total variation the answer is plain, and KL inherits a share of it through an inequality.

37.3.3.1 Total Variation

The function $\frac{1}{2}|u - 1|$ is the plainest in Table 37.3: its penalty is the distance of the ratio from 1 and nothing else. Its divergence, half the ℓ_1 distance between p and q, has a second form that says what the number means. Total variation is the largest disagreement between P and Q about the probability of any event A:

$$\mathrm{TV}(P, Q) = \frac{1}{2} \sum_x |p(x) - q(x)| = \sup_A |P(A) - Q(A)|. \tag{37.29}$$

If $\mathrm{TV}(P, Q) = 0.03$, then no event distinguishes the two distributions by more than three percentage points. The supremum is attained at $A^\star = \{x : p(x) > q(x)\}$, the outcomes that P favors. To see this, take any event A. Then $P(A) - Q(A) = \sum_{x \in A} (p(x) - q(x)) \le \sum_{x \in A^\star} (p(x) - q(x))$, since passing from A to $A^\star$ adds only non-negative terms and removes only non-positive ones. Because $\sum_x (p(x) - q(x)) = 0$, the positive part of $p - q$ equals the negative part in magnitude, so the right side is $\frac{1}{2} \sum_x |p(x) - q(x)|$, and the same bound holds for $Q(A) - P(A)$.

On the running pair $A^\star$ is the first outcome and $\mathrm{TV} = 0.6 - 0.2 = 0.4$, as before. Figure 37.8 shows the same for two densities. TV is a metric: both forms are symmetric, and the triangle inequality follows from the first (Exercise 2a).

The second form gives TV its operational meaning. Hand a tester one sample, drawn from P or from Q with probability $\frac{1}{2}$ each, and ask which distribution produced it. A test that answers "P" when the sample lands in an event A, and "Q" otherwise, is right with probability

$$\tfrac{1}{2} P(A) + \tfrac{1}{2}(1 - Q(A)) = \tfrac{1}{2}(1 + P(A) - Q(A)) \le \tfrac{1}{2}(1 + \mathrm{TV}(P, Q)),$$

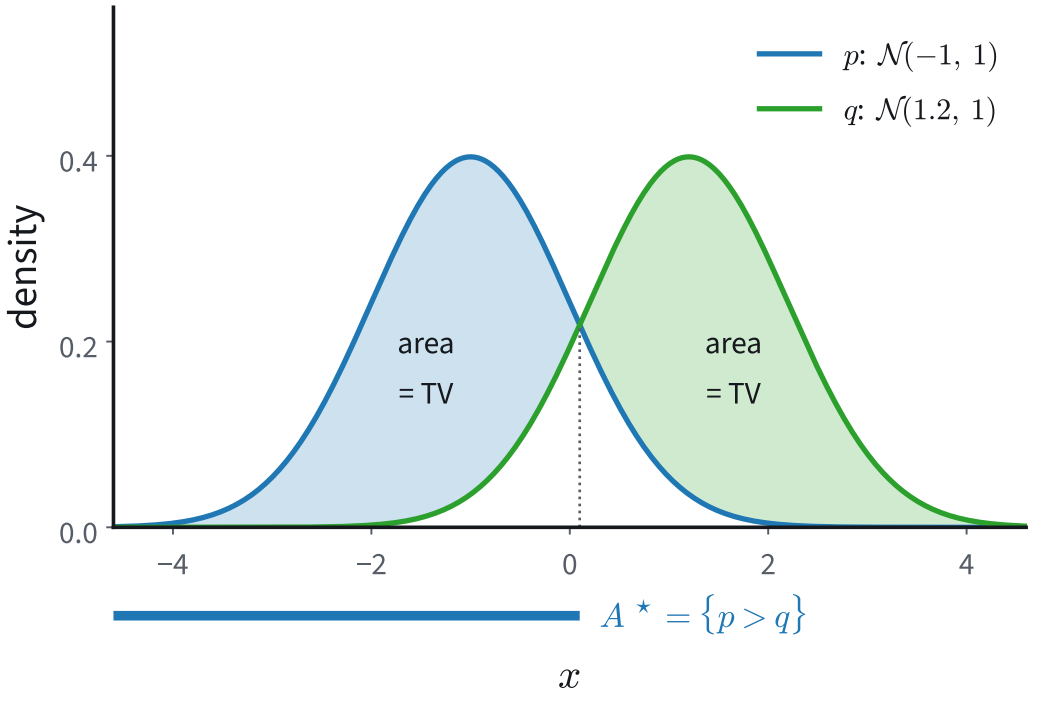


Figure 37.8: Two equal-variance densities and the regions where they disagree. The lobe where $p > q$ and the lobe where $q > p$ have the same area, and each equals the total variation distance. The event that attains the supremum is $A^\star = \{p > q\}$, here the half-line left of the crossing: betting on P there and on Q elsewhere is the best single-sample test.

with equality at $A = A^\star$. Every test is of this form for some A (a randomized test averages such tests and does no better), so the best possible test beats coin flipping by $\mathrm{TV}/2$. On the running pair the best test answers "P" on the first outcome and is right 70% of the time. The discriminator of a GAN, a network trained to tell data from generated samples, and a detector of generated text are testers of this kind, and TV limits what either can achieve from one sample.

37.3.3.2 Pinsker's Inequality

TV limits every test, and KL is what we can usually compute or bound, because it is built from log-probabilities, as losses are. Pinsker's inequality connects the two: a small KL divergence certifies that no test can tell the distributions apart well. The trust regions of Section 15.2 are a case in point. The guarantee that a policy update improves the return is proved for total variation and stated as a limit on KL.

Proposition (Pinsker's inequality). (Pinsker 1964; Csiszár 1967) *For distributions P and Q on a finite or countable set,*

$$\mathrm{TV}(P, Q) \;\le\; \sqrt{\tfrac{1}{2}\, D_{\mathrm{KL}}(P \| Q)}. \tag{37.30}$$

Proof. *Step 1: two outcomes.* Write $\mathrm{Ber}(a)$ for a coin that shows heads with probability a. For $a \in [0,1]$ and $b \in (0,1)$ the KL divergence is $D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) = a \log \frac{a}{b} + (1-a) \log \frac{1-a}{1-b}$ and the TV distance is $|a-b|$, so we must show that $\rho(a) = D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) - 2(a-b)^2$ is non-negative. Fix b. Then $\rho(b) = 0$, the derivative $\rho'(a) = \log \frac{a}{b} - \log \frac{1-a}{1-b} - 4(a-b)$ vanishes at $a = b$, and for $0 < a < 1$

$$\rho''(a) = \frac{1}{a(1-a)} - 4 \ge 0,$$

since $a(1-a) \le \frac{1}{4}$. A convex function with value and slope zero at $a = b$ is non-negative, and by continuity (with $0 \log 0 = 0$) also at the endpoints $a = 0$ and $a = 1$. If b is 0 or 1, the divergence is infinite unless $a = b$, and there is nothing to prove.

Step 2: reduction to two outcomes. Let $A^\star = \{p > q\}$, and set $a = P(A^\star)$, $b = Q(A^\star)$, so that $\mathrm{TV}(P, Q) = a - b$. Merging the outcomes inside $A^\star$ and inside its complement can only *decrease* KL, by the log-sum inequality: for non-negative numbers v_i and w_i,

$$\sum_i v_i \log \frac{v_i}{w_i} \ge \Big(\sum_i v_i\Big) \log \frac{\sum_i v_i}{\sum_i w_i},$$

itself one application of Jensen to $t \mapsto t \log t$. Applied separately to the terms in $A^\star$ and in its complement,

$$D_{\mathrm{KL}}(P\|Q) \;\ge\; D_{\mathrm{KL}}(\mathrm{Ber}(a)\|\mathrm{Ber}(b)) \;\ge\; 2(a-b)^2 = 2\,\mathrm{TV}(P,Q)^2,$$

which rearranges to Equation 37.30. ■

On the running pair the inequality reads $0.4 \le \sqrt{0.396/2} = 0.445$, and Step 2 can be followed by hand. Merging the second and third outcomes leaves the coins $(0.6, 0.4)$

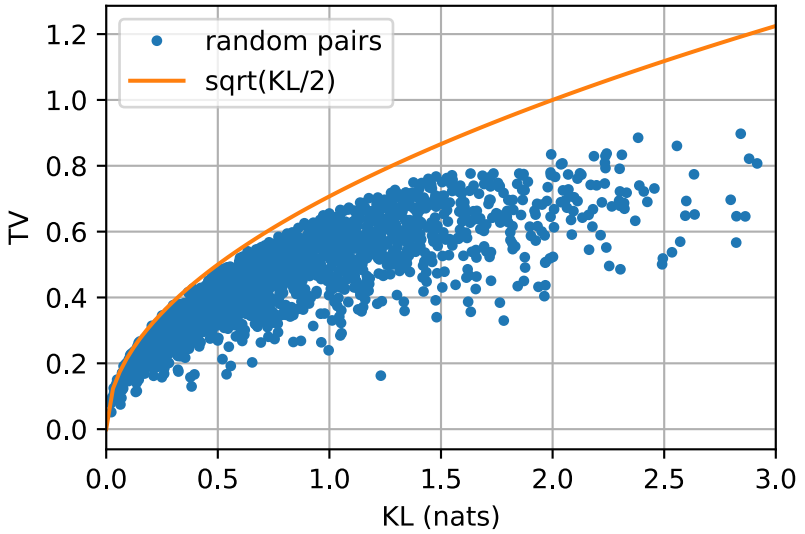
and $(0.2, 0.8)$. Their TV distance is still 0.4, their KL divergence has fallen from 0.396 to $0.6 \log 3 + 0.4 \log \frac{1}{2} = 0.382$ nats, and Step 1 promises at least $2\,(0.4)^2 = 0.32$.

For a coin of bias 0.6 against a fair coin the KL divergence is $0.6 \log 1.2 + 0.4 \log 0.8 = 0.0201$ nats, and Step 1 promises at least $2\,(0.1)^2 = 0.02$: for nearly equal coins the inequality is nearly an equality. It has no useful converse, however. A model that assigns $q = 0$ to a rare event has infinite KL at tiny TV. The bound also goes slack for distant pairs, since TV is at most 1: two unit-variance Gaussians 50 apart have TV ≈ 1 but, by Equation 37.24, KL $= 50^2/2 = 1250$ nats.

How tight is the bound for typical pairs, and can the constant $\frac{1}{2}$ be improved? For the first question we draw $10{,}000$ pairs of distributions on five outcomes, uniformly from all such distributions. For the second we let two coins approach each other. Step 1 lost nothing to second order at $a = b = \frac{1}{2}$, so the ratio $\mathrm{TV}/\sqrt{\mathrm{KL}/2}$ should approach 1 there.

```
rng = np.random.default_rng()
pr, qr = rng.dirichlet(np.ones(5), 10000), rng.dirichlet(np.ones(5), 10000)
tv = 0.5 * np.abs(pr - qr).sum(axis=1)
kl_pq = np.sum(pr * np.log(pr / qr), axis=1)
ratio = tv / np.sqrt(kl_pq / 2)
print(f'TV / sqrt(KL/2) over 10,000 random pairs: median {np.median(ratio):.2f}, '
      f'max {ratio.max():.3f}')
for delta in (0.1, 0.01, 0.001):
    a, b = np.array([0.5, 0.5]), np.array([0.5 + delta, 0.5 - delta])
    print(f'coins 1/2 vs 1/2 + {delta:5}: TV / sqrt(KL/2) = '
          f'{delta / np.sqrt(kl_div(a, b) / 2):.6f}')
grid = np.linspace(0, 3, 100)
d2l.plot([kl_pq[:2000], grid], [tv[:2000], np.sqrt(grid / 2)], 'KL (nats)',
         'TV', legend=['random pairs', 'sqrt(KL/2)'], fmts=('.', '-'),
         xlim=[0, 3], figsize=(4.5, 3))
```

```
TV / sqrt(KL/2) over 10,000 random pairs: median 0.78, max 0.986
coins 1/2 vs 1/2 +   0.1: TV / sqrt(KL/2) = 0.989881
coins 1/2 vs 1/2 +  0.01: TV / sqrt(KL/2) = 0.999900
coins 1/2 vs 1/2 + 0.001: TV / sqrt(KL/2) = 0.999999
```



In the scatter plot every pair lies below the curve $\sqrt{\mathrm{KL}/2}$. The median pair reaches nearly 0.8 of the bound, and the largest ratio among the $10{,}000$ comes within a few hundredths of 1. Beyond $\mathrm{KL} = 2$ the curve exceeds 1 and the bound says nothing. For nearly fair coins the ratio climbs to 0.9899, 0.9999 and 0.999999, so the constant $\frac{1}{2}$ cannot be improved (Exercise 4c). Pinsker's own constant was weaker. The sharp one is due to Csiszár (1967) and, independently, to Kullback and to Kemperman.

37.3.3.3 Data Processing

Can a feature extractor, a quantizer, or any other processing make two distributions easier to tell apart? Pinsker's proof merged outcomes and found that KL can only fall. The same holds for every f-divergence and every kind of processing. A *channel* $\kappa(y \mid x)$ is any rule, deterministic or random, that turns an outcome x into an outcome y. It maps P to the output distribution $(P\kappa)(y) = \sum_x \kappa(y \mid x)\, p(x)$, and Q likewise.

Proposition (data processing for f-divergences). *For any channel κ, convex f with $f(1) = 0$, and $P \ll Q$ on a finite or countable set,*

$$D_f(P\kappa \,\|\, Q\kappa) \;\le\; D_f(P\|Q). \tag{37.31}$$

Proof. Since $\sum_y \kappa(y \mid x) = 1$, $D_f(P\|Q) = \sum_y \sum_x \kappa(y \mid x)\, q(x)\, f(p(x)/q(x))$. Fix y with $(Q\kappa)(y) > 0$. The weights $\kappa(y \mid x)\, q(x)/(Q\kappa)(y)$ sum to one over x, and averaging the ratios $p(x)/q(x)$ with these weights gives $(P\kappa)(y)/(Q\kappa)(y)$. Jensen's inequality therefore bounds the inner sum below by $(Q\kappa)(y)\, f\big((P\kappa)(y)/(Q\kappa)(y)\big)$, which is the y-th term of $D_f(P\kappa\|Q\kappa)$. An outcome y with $(Q\kappa)(y) = 0$ has $\kappa(y \mid x) = 0$ wherever $q(x) > 0$, hence also $(P\kappa)(y) = 0$, and contributes nothing to either side. Summing over y finishes the proof. ■

No processing of the data, whatever the divergence function, makes two distributions easier to tell apart. In deep learning the inequality is mostly read from right to left: a divergence measured on features, on quantized values or on any other function of the data is a lower

bound on the divergence between the inputs. Merging outcomes, as in Pinsker's proof, is the deterministic channel $x \mapsto \mathbf{1}(x \in A^\star)$. On the running pair it lowered KL from 0.396 to 0.382, and it lowers χ^2 from 1.013 to $0.16/0.2 + 0.16/0.8 = 1$. The case of KL is what Section 37.6.2.2 uses for mutual information.

A map that can be undone is a channel in both directions, so for it the inequality is an equality: an f-divergence does not change when the outcomes are relabeled or the axis is rescaled. The last two rows of Table 37.2 do change with the scale. What they gain in sensitivity to distance they lose in this guarantee.

37.3.4 The Fenchel Conjugate and the Critic Bound

A network that generates samples gives us draws from its distribution Q, and we hold draws from the data distribution P. Neither set of draws tells us the ratio $p(x)/q(x)$ that the definition Equation 37.25 needs. Can D_f be computed from samples alone? It can be bounded from below, and the bound turns a divergence into a training objective.

37.3.4.1 Derivation of the Bound

The tool is the convex conjugate of Section 35.7.4. For a convex f it is

$$f^*(t) = \sup_u \big(ut - f(u)\big),$$

so that $-f^*(t)$ is the intercept of the tangent line to f with slope t (Figure 37.9). For $f(u) = u \log u$ the supremum is attained where $\log u + 1 = t$, and $f^*(t) = e^{t-1}$. A convex function lies above each of its tangents, which is the Fenchel–Young inequality

$$f(u) \geq ut - f^*(t) \quad \text{for all } t, \tag{37.32}$$

with equality when t is the slope of f at u, that is, $t = f'(u)$.

We now apply the inequality *pointwise inside the divergence* and let the slope vary with x. We choose a function $T(x)$, the **critic**, and apply Equation 37.32 at $u = p(x)/q(x)$ and $t = T(x)$.

Proposition (critic bound). *Let f be convex and differentiable with $f(1) = 0$, let $P \ll Q$, and let T be any critic with $\mathbb{E}_P|T| < \infty$ and $\mathbb{E}_Q|f^*(T)| < \infty$. Then*

$$D_f(P\|Q) \;\geq\; \mathbb{E}_{x\sim P}[T(x)] - \mathbb{E}_{x\sim Q}[f^*(T(x))]. \tag{37.33}$$

The supremum of the right side over critics equals $D_f(P\|Q)$, and it is attained by $T^\star(x) = f'\big(p(x)/q(x)\big)$ wherever $T^\star$ is finite (Nguyen et al. 2010; Nowozin et al. 2016).

Proof. Multiply Equation 37.32 at $u = p(x)/q(x)$, $t = T(x)$ by $q(x) \geq 0$:

$$q(x)\, f\!\left(\frac{p(x)}{q(x)}\right) \;\geq\; p(x)\,T(x) - q(x)\, f^*(T(x)).$$

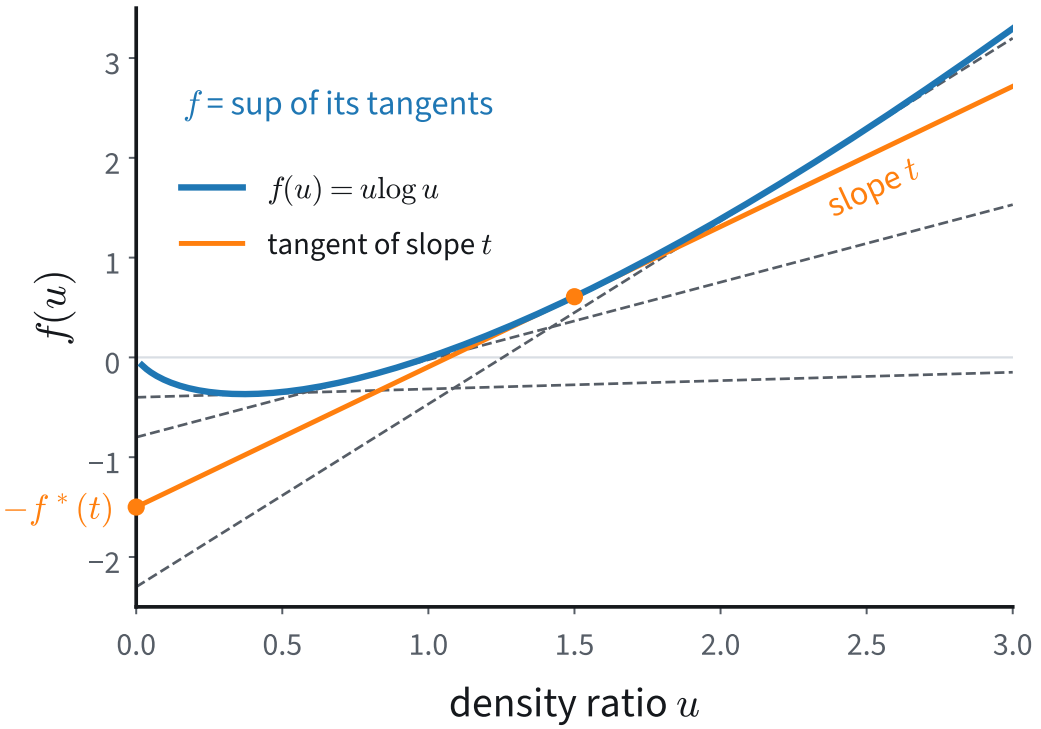


Figure 37.9: A convex divergence function, here $f(u) = u \log u$, is the upper envelope of its tangent lines. The tangent of slope t has the intercept $-f^*(t)$, here $-e^{t-1}$. Keeping one tangent gives the Fenchel–Young lower bound, and taking the supremum over slopes recovers the function. Applied at every point to the density ratio, the same inequality gives the critic bound.

Summing (or integrating) over x gives Equation 37.33. Since Fenchel–Young holds with equality at $t = f'(u)$, the choice $T^\star(x) = f'(p(x)/q(x))$ makes the inequality an equality at every x, hence in expectation. ■

Both expectations on the right of Equation 37.33 are averages, of T over the data and of $f^*(T)$ over the model's samples, and neither needs a density. A neural network T can therefore be trained on minibatches to maximize the bound. For KL the bound reads $D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[e^{T-1}]$ for every critic. The **f-GAN** of Nowozin et al. (2016), after which the bound is also called the f-GAN bound, makes a game of it. The critic maximizes the bound, which equals the divergence at the optimal critic, while the generator, the network that produces the samples of Q, minimizes it. Section 16.2 trains such pairs.

37.3.4.2 The Original GAN

The original GAN of Goodfellow et al. (2014) is the case $f(u) = u\log u - (u+1)\log(u+1)$. Its conjugate is $f^*(t) = -\log(1-e^t)$ for $t < 0$, and writing the critic as the logarithm of a discriminator D with values in $(0,1)$, $T = \log D$, turns the bound into the familiar objective

$$\mathbb{E}_P[T] - \mathbb{E}_Q[f^*(T)] = \mathbb{E}_{x\sim P}[\log D(x)] + \mathbb{E}_{x\sim Q}[\log(1-D(x))].$$

The optimal critic $f'(p/q) = \log\left(p/(p+q)\right)$ is the discriminator $D^\star = p/(p+q)$. With f_{JS} the Jensen–Shannon function of Table 37.3, this f equals $2f_{\mathrm{JS}}(u) - (u-1)\log 2 - \log 4$: twice the Jensen–Shannon function, a multiple of $u-1$, which changes no divergence, and the constant $f(1) = -\log 4$, which only shifts the value. At $D^\star$ the game is therefore worth $2\,\mathrm{JS}(P,Q) - \log 4$, on the running pair $2(0.0911) - 1.386 = -1.204$.

37.3.4.3 Limits of the Bound

The bound has two limits in practice, and the first can be measured. It is tight only at the optimal critic, so a critic that is not fully trained reports less than the divergence. To see how much less, we train one.

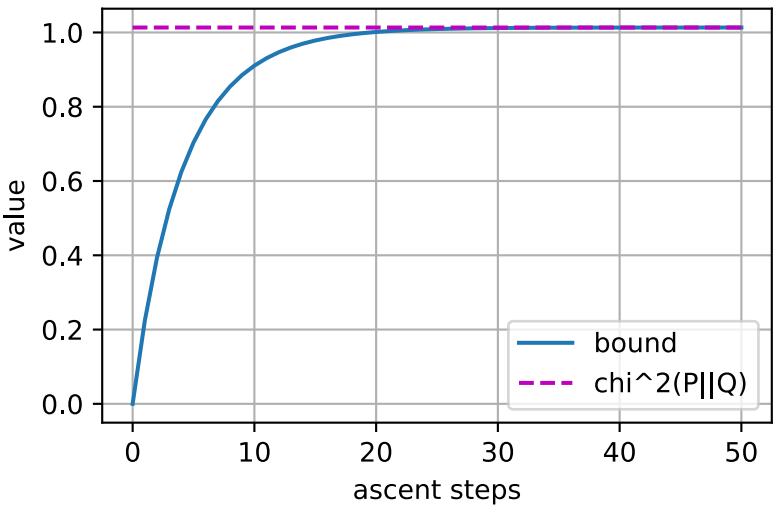
On the running pair expectations are sums, and a critic is three numbers, one per outcome. We take χ^2, with $f(u) = (u-1)^2$, conjugate $f^*(t) = t + t^2/4$ and optimal critic $T^\star = 2(p/q - 1)$. (The conjugate takes the supremum over all real u; Exercise 7a derives it and shows that this loses nothing at $T^\star$.) The critic starts at $T = 0$, where the bound is 0, and takes gradient ascent steps of size 1 on the bound, whose derivative with respect to $T(x)$ is $p(x) - q(x)\,(1 + T(x)/2)$. The proposition says where the run must end: at $\chi^2(P\|Q) = 1.0133$, approached from below. The last line of the cell evaluates a critic that points the right way and is twice too confident, $2T^\star$.

```
def fgan_bound(T, p, q, f_star):
    """E_P[T] - E_Q[f*(T)]: a lower bound on D_f(P||Q) for any critic T."""
    return np.sum(p * T) - np.sum(q * f_star(T))

chi2_star = lambda t: t + t ** 2 / 4    # conjugate of f(u) = (u - 1)^2
exact = f_divergence(generators['chi^2'], p, q)
T, bounds = np.zeros(3), []             # untrained critic: one number per outcome
for step in range(51):
```

```
    bounds.append(fgan_bound(T, p, q, chi2_star))
    T = T + (p - q * (1 + T / 2))       # ascent step 1 along p - q f*'(T)
shown = (0, 1, 2, 5, 10, 20, 50)
print(f'exact chi^2(P||Q) = {exact:.4f}')
print('steps ' + ''.join(f'{s:8d}' for s in shown))
print('bound ' + ''.join(f'{bounds[s]:8.4f}' for s in shown))
T_star = 2 * (p / q - 1)                # the optimal critic f'(p/q)
print(f'overconfident critic 2 T*: bound = '
      f'{fgan_bound(2 * T_star, p, q, chi2_star):.4f}')
d2l.plot(np.arange(51), [bounds, [exact] * 51], 'ascent steps', 'value',
         legend=['bound', 'chi^2(P||Q)'], figsize=(4.5, 2.8))
```

```
exact chi^2(P||Q) = 1.0133
steps        0       1       2       5      10      20      50
bound   0.0000  0.2240  0.3935  0.7036  0.9107  1.0013  1.0133
overconfident critic 2 T*: bound = 0.0000
```



After five steps the critic certifies 0.70 of the 1.01, about 70% of the divergence, after ten steps 90%, and after twenty 99%. At no step does the bound exceed the exact value. The overconfident critic reports zero, no more than the untrained one. With exact expectations, as here, an adversarial estimate of a divergence is therefore an underestimate, by an amount that the estimate itself does not reveal. With averages over samples it remains one, on average, only for samples that the critic was not trained on (Exercise 8d).

The update also tells us where the missing part hides. It moves $T(x)$ at a rate proportional to $q(x)$, so the critic is slowest to learn where the model's samples are rare. On the running pair that is the first outcome, where the ratio is largest and most of the divergence lies (Exercise 7b).

The second limit is the constancy on disjoint supports, now seen from the critic's side. If we extend $T^\star = f'(p/q)$ formally beyond the proposition's hypothesis $P \ll Q$, the ratio is ∞ on the data and 0 on the model's samples, so $T^\star$ is the constant $f'(\infty)$ on one support and $f'(0)$ on the other. The generator is updated through the gradient of the critic at the generated samples, and a critic that is constant there has gradient zero. The generator learns

nothing about which way to move.

Both limits have one source, the ratio p/q. Samples do not supply it, and where the supports separate it is 0 or ∞ and keeps no trace of the distance between them. Section 37.4 asks what can be compared without it.

37.3.5 Summary

The KL divergence is one member of a family: an f-divergence averages a convex function of the density ratio p/q under Q. Jensen's inequality makes every member non-negative, near $P = Q$ all smooth members agree up to the curvature $f''(1)$, and none of them grows when the data are processed. The direction of KL is a modeling decision: forward KL, which maximum likelihood minimizes, covers every mode, and reverse KL, which variational inference minimizes, can settle on one. The best test that sees one sample is right with probability $\frac{1}{2}(1 + \mathrm{TV})$, and TV is at most $\sqrt{D_{\mathrm{KL}}/2}$. A critic bounds any f-divergence from below by two averages, one over each distribution, which is how adversarial training estimates a divergence without densities. The weakness of the family is the ratio itself: on disjoint supports every f-divergence is the constant $f(0) + f'(\infty)$, however near or far the supports are.

37.3.6 Exercises

1. **Divergence functions and argument order.** Each row of Table 37.3 is one choice of f in the template Equation 37.25.
   1. For a divergence function f, define $f^\diamond(u) = u\, f(1/u)$. Show that $f^\diamond$ is convex with $f^\diamond(1) = 0$ and that $D_f(Q\|P) = D_{f^\diamond}(P\|Q)$. Compute $f^\diamond$ for the KL function $u \log u$ and for the χ^2 function $(u-1)^2$, and write the divergence that the second one defines as a sum over x.
   2. Adding $c\,(u-1)$ to f does not change the divergence. Show that D_f is symmetric whenever $f^\diamond(u) = f(u) + c\,(u-1)$ for some constant c. Check this condition for the rows of Table 37.3 that are marked symmetric, and show that $D_{f_\alpha}(Q\|P) = D_{f_{1-\alpha}}(P\|Q)$ for the function Equation 37.27. Use this identity to confirm the entry of the α-divergence row in the "Symmetric?" column.
   3. Expand Equation 37.26 in terms of the ratio $u = p/q$ and recover the Jensen–Shannon function of Table 37.3.
   4. *Rényi divergence.* For $\alpha > 0$, $\alpha \neq 1$, define $D_\alpha(P\|Q) = \frac{1}{\alpha-1} \log \sum_x p(x)^\alpha\, q(x)^{1-\alpha}$ (Rényi 1961). Show that $\sum_x p^\alpha q^{1-\alpha} = 1 + \alpha(\alpha-1)\, D_{f_\alpha}(P\|Q)$ and that D_α is an increasing function of D_{f_α}. Deduce that $D_\alpha \geq 0$ and that D_α satisfies the data-processing inequality Equation 37.31. Find $\lim_{\alpha\to 0} D_\alpha(P\|Q)$, and explain why it is 0 whenever p and q are positive on the same outcomes. Why does the Rényi family therefore *not* reach reverse KL as $\alpha \to 0$, while D_{f_α} does?
2. **Metrics among divergences.** The last column of Table 37.3 records which divergences are metrics.
   1. Prove the triangle inequality for TV from its half-ℓ_1 form.
   2. Show that the Hellinger distance $d_{\mathrm{H}} = \sqrt{d_{\mathrm{H}}^2}$ is the Euclidean distance between the vectors $\left(\sqrt{p(x)}\right)_x$ and $\left(\sqrt{q(x)}\right)_x$, so that d_{H} is a metric. Find three distributions on two outcomes for which both d_{H}^2 and JS violate the triangle inequality.
3. **Cost of dropping a mode.** A model Q misses part of a data distribution P.

1. Let $P = (1-w)\,Q + wR$ with $0 < w < 1$, where R is a distribution on a set to which Q assigns probability zero. Compute $D_{\mathrm{KL}}(P\|Q)$, $D_{\mathrm{KL}}(Q\|P)$, $\mathrm{TV}(P,Q)$ and $d^2_{\mathrm{H}}(P,Q)$ as functions of w. Which direction of KL penalizes the missing mass? How do the finite divergences behave as $w \to 0$?
2. Now let $P = w_1\mathcal{N}(m_1, s^2) + w_2\mathcal{N}(m_2, s^2)$ with separation $\Delta = |m_2 - m_1|$, and let the model $Q = \mathcal{N}(m_1, s^2)$ keep only the first component. Show that $D_{\mathrm{KL}}(Q\|P) \le \log(1/w_1)$ for every Δ, with equality in the limit $\Delta \to \infty$, and relate the limit to (a). Use the bound to explain the two reverse-KL values, 0.356 and 1.202 nats, that the fit of a Gaussian to the mixture prints.
3. Assume that the components are well separated, so that almost every x has non-negligible density under at most one of them. Show that $D_{\mathrm{KL}}(P\|Q) \approx w_2\Delta^2/(2s^2) + w_1\log w_1 + w_2\log w_2$. Evaluate the approximation for the mixture of that experiment, and check it with `kl(log_p(x), log_q(x, [-2.0, np.log(0.6)]), x)`. The experiment prints two values in its forward-KL column to compare with: 5.981 nats at the reverse-KL fit, whose parameters $(-1.998, 0.603)$ are not quite $(-2, 0.6)$, and 0.558 nats at the forward fit.

4. **Local quadratic regime.** The proposition Equation 37.28 bounds the gap between D_f and $\frac{1}{2}f''(1)\,\chi^2$.
   1. Exercise 10b of Section 37.1 finds the order at which the two directions of the Gaussian KL divergence first differ. Which property of the KL and reverse-KL functions explains that order? Carry the Taylor expansion of the proof one term further, and show that the ratio which the local-quadratic experiment prints is $1 + c\,t\,f'''(1)/f''(1) + O(t^2)$ with $c = \mathbb{E}_Q[(u-1)^3]/(3\chi^2(P\|Q))$, about 0.49 for the running pair. Compute $f'''(1)/f''(1)$ for the four functions. Why is the shortfall of reverse KL at $t = 0.01$ twice that of KL, and why do squared Hellinger and JS print the same value there?
   2. The total-variation function $\frac{1}{2}|u-1|$ has no second derivative at 1. Let $P_t = Q + t\,(P-Q)$ for $t \in (0,1]$. How do $\mathrm{TV}(P_t,Q)$ and $\chi^2(P_t\|Q)$ scale with t? Explain from this why Pinsker's inequality Equation 37.30 bounds TV by the square root of KL and not by a multiple of KL.
   3. Let P and Q be coins with heads probabilities b and $b+\delta$. Use the proposition to show that $D_{\mathrm{KL}}(P\|Q) = \delta^2/(2b(1-b)) + O(\delta^3)$. Exercise 7a of Section 37.2 obtains the same expansion directly from the Bernoulli KL divergence. The factor $1/(b(1-b))$ is the Fisher information of the coin (Section 36.6). Find the limit of $\mathrm{TV}/\sqrt{D_{\mathrm{KL}}/2}$ as $\delta \to 0$, and the values of b for which Pinsker's inequality is tight in this limit. The coin experiment that follows Pinsker's inequality takes $b = \frac{1}{2}$ and prints a ratio of 0.999999 at $\delta = 10^{-3}$. Why does the ratio approach 1 at the rate δ^2 and not δ?

5. **Detecting generated text.** A detector receives one text of n tokens. With probability $\frac{1}{2}$ a person wrote it (distribution P over texts), and otherwise a language model sampled it (distribution Q).
   1. Show that no detector is right with probability above $\frac{1}{2}(1 + \mathrm{TV}(P,Q))$. Show also that a detector which sees only a function of the text, such as its length, its word counts or the features of a network, can do no better than one that sees the text.
   2. On human text the model's loss exceeds the entropy of the text by ε nats per token, so that $D_{\mathrm{KL}}(P\|Q) = n\varepsilon$ by the decomposition of the cross-entropy in Section 37.1. Bound the detector's accuracy with Pinsker's inequality. For $\varepsilon = 0.02$, what does the bound allow at $n = 1$ and at $n = 10$, and from which length on

does it allow an accuracy of 90%?

3. The bound of (b) limits detection and promises nothing. Let Q be P conditioned on avoiding a set of texts that has probability 0.02 under P. Compute $\mathrm{TV}(P,Q)$, $D_{\mathrm{KL}}(P\|Q)$ and $D_{\mathrm{KL}}(Q\|P)$. What does the best detector do, and how often is it right? TV is symmetric, so Pinsker's inequality holds with either direction of KL. Which direction is the useful one here?

6. **Hellinger bounds and testing.** ⊠ Pinsker's inequality is vacuous once $D_{\mathrm{KL}} > 2$. Let $\mathrm{BC}(P,Q) = \sum_x \sqrt{p(x)q(x)} = 1 - \frac{1}{2}d_{\mathrm{H}}^2(P,Q)$ be the Bhattacharyya coefficient.
   1. Prove that $\frac{1}{2}d_{\mathrm{H}}^2(P,Q) \le \mathrm{TV}(P,Q)$, using $|p-q| = |\sqrt{p}-\sqrt{q}|\,(\sqrt{p}+\sqrt{q})$.
   2. Prove that $\mathrm{TV}(P,Q) \le d_{\mathrm{H}}(P,Q)\sqrt{1-\frac{1}{4}d_{\mathrm{H}}^2(P,Q)}$ with the Cauchy–Schwarz inequality of Section 33.1, and rewrite the bound as $\mathrm{TV}^2 \le 1 - \mathrm{BC}^2$.
   3. Use Jensen's inequality to show that $\mathrm{BC}(P,Q) \ge e^{-D_{\mathrm{KL}}(P\|Q)/2}$, and conclude the Bretagnolle–Huber inequality $\mathrm{TV}(P,Q) \le \sqrt{1-e^{-D_{\mathrm{KL}}(P\|Q)}}$ (Bretagnolle and Huber 1978; Canonne 2022). For which values of D_{KL} is it tighter than Equation 37.30?
   4. A test sees n independent samples, all from P or all from Q, each case with probability $\frac{1}{2}$. Use (c) and the additivity of KL over independent samples to show that every test errs with probability at least $\frac{1}{4}e^{-nD_{\mathrm{KL}}(P\|Q)}$. How many samples does an error probability of at most η require? Why can Pinsker's inequality not give a bound of this kind?

7. **Conjugates and critics.** The critic bound Equation 37.33 needs the conjugate f^* of the divergence function.
   1. Derive the conjugate $f^*(t) = t + t^2/4$ of the χ^2 function $f(u) = (u-1)^2$, taking the supremum over $u \in \mathbb{R}$, and its optimal critic $T^\star = 2(p/q-1)$. Redo the computation over the function's domain $u \in (0,\infty)$, and show that the supremum is $t + t^2/4$ for $t \ge -2$ and -1 for $t \le -2$.
   2. Use the $\mathbb{R}$-conjugate. Show that for every critic T the gap between $\chi^2(P\|Q)$ and the bound equals $\frac{1}{4}\mathbb{E}_Q\big[(T-T^\star)^2\big]$. Deduce that a scaled critic $T = c\,T^\star$ certifies $\chi^2(P\|Q)\,(2c-c^2)$, which explains the value that the experiment prints for $2T^\star$. Show that one ascent step of the experiment multiplies $T(x) - T^\star(x)$ by $1 - q(x)/2$. Which outcome of the running pair accounts for most of the gap after ten steps, and why? For which step sizes does the ascent converge?
   3. Compute the conjugate of the total-variation function $f(u) = \frac{1}{2}|u-1|$ over $u \in (0,\infty)$. Show that critics with values in $[-\frac{1}{2},\frac{1}{2}]$ suffice to maximize the bound, and conclude that $\mathrm{TV}(P,Q) = \sup_g \left(\mathbb{E}_P[g] - \mathbb{E}_Q[g]\right)$ over all functions g with $|g| \le \frac{1}{2}$, a supremum of the kind that Section 37.4.1 studies. Which g attains it?

8. code **Restricted critics.** ⊠ The bound Equation 37.33 is tight only when the critic class contains the optimal critic. Take the χ^2 function with the conjugate $f^*(t) = t + t^2/4$, and critics $T_\theta(x) = \theta^\top \phi(x)$ that are linear in a feature vector $\phi(x)$. Write $\mathbf{m}_P = \mathbb{E}_P[\phi]$, $\mathbf{m}_Q = \mathbb{E}_Q[\phi]$ and $\Sigma = \mathbb{E}_Q[\phi\phi^\top]$, assumed invertible.
   1. Show that the bound is a concave quadratic in θ, maximized at $\theta^\star = 2\Sigma^{-1}(\mathbf{m}_P - \mathbf{m}_Q)$, with the value $(\mathbf{m}_P - \mathbf{m}_Q)^\top \Sigma^{-1}(\mathbf{m}_P - \mathbf{m}_Q)$.
   2. For affine critics, $\phi(x) = (1, x)$, show that the value is $(\mathbb{E}_P[X] - \mathbb{E}_Q[X])^2/\mathrm{Var}_Q(X)$. What does the restricted bound certify for $P = \mathcal{N}(0,1)$ and $Q = \mathcal{N}(0,4)$?
   3. Use the features $\phi_k(x) = \mathrm{He}_k\big((x-\mu_Q)/\sigma_Q\big)$ for $k = 0, \dots, K$, where He_k are

the probabilists' Hermite polynomials (`numpy.polynomial.hermite_e.hermevander`) and μ_Q, σ_Q are the mean and standard deviation of Q. Compute $\mathbf{m}_P$, $\mathbf{m}_Q$ and Σ by quadrature on a grid, and plot the value against $K \le 10$ for the pairs $(\mathcal{N}(1,1), \mathcal{N}(0,1))$ and $(\mathcal{N}(0,1), \mathcal{N}(0,4))$, together with their exact divergences $\chi^2 = e - 1$ and $4/\sqrt{7} - 1$. For the first pair the values are partial sums of a familiar series. Explain why from the generating function $e^{xt - t^2/2} = \sum_k \mathrm{He}_k(x)\, t^k/k!$ and the orthogonality of the polynomials: for $Z \sim \mathcal{N}(0,1)$, $\mathbb{E}[\mathrm{He}_j(Z)\,\mathrm{He}_k(Z)]$ is $k!$ if $j = k$ and 0 otherwise.

4. For the second pair, replace the expectations by averages over $n = 200$ samples from each distribution and fit $\theta^\star$ on these samples. Evaluate the bound on the same samples and on 10^5 fresh samples, averaged over 200 repetitions, for $K \in \{2, 4, 6, 8, 10\}$. Which of the two evaluations estimates a valid lower bound on χ^2? Why does the other one exceed the restricted value of (c) and, for large K, even χ^2 itself?

37.4 Test Functions and Transport

A model whose samples miss the data learns nothing from an f-divergence. On disjoint supports every member of that family is a constant (Section 37.3.1.2), and a constant has no gradient. The verdict rested on two point masses, the extreme case. Real distributions have spread: two Gaussians overlap everywhere, because their tails never reach zero. Does a little overlap bring the gradient back? The f-divergences left a second complaint as well. They need the ratio p/q, which samples do not supply, so a critic had to be trained before anything could be measured (Section 37.3.4). Can two samples be compared with no critic at all?

We can settle the first question with numbers. Let the data be $P = \mathcal{N}(0,1)$ and the model $Q = \mathcal{N}(d,1)$, the same bell curve d standard deviations away, with d the model's one parameter. After the imports, a cell evaluates the five comparisons of Table 37.2 on this pair, each with its slope in d, the gradient that the model receives. Its formulas are derived in this section and the last; for now, read only the output, which we shall call the opening table.

```
from scipy import optimize
from scipy.stats import norm
```

```
def js(d, x=np.linspace(-12, 20, 6401)):
    """JS divergence of N(0, 1) and N(d, 1) by quadrature on the grid x."""
    log_p, log_q = norm.logpdf(x), norm.logpdf(x, loc=d)
    log_m = np.logaddexp(log_p, log_q) - math.log(2)    # log of the mixture
    terms = np.exp(log_p) * (log_p - log_m) + np.exp(log_q) * (log_q - log_m)
    return terms.sum() * (x[1] - x[0]) / 2

comparisons = {                        # N(0, 1) against N(d, 1), RBF ell = 1
    'KL': lambda d: d ** 2 / 2,
    'JS': js,
    'TV': lambda d: 2 * norm.cdf(d / 2) - 1,
    'MMD^2': lambda d: 2 / math.sqrt(3) * (1 - math.exp(-d ** 2 / 6)),
    'W1': lambda d: d}
```

```
h = 1e-3                                # step of the central difference
print('   d' + ''.join(f'{name:>8s} {"slope":>7s}' for name in comparisons))
for d in (0.5, 1.0, 2.0, 4.0, 8.0):
    print(f'{d:4.1f}' + ''.join(
        f'{f(d):8.3f} {(f(d + h) - f(d - h)) / (2 * h):7.1e}'
        for f in comparisons.values()))
```

d	KL	slope	JS	slope	TV	slope	MMD^2	slope	W1	slope
0.5	0.125	5.0e-01	0.030	1.2e-01	0.197	3.9e-01	0.047	1.8e-01	0.500	1.0e+00
1.0	0.500	1.0e+00	0.111	2.0e-01	0.383	3.5e-01	0.177	3.3e-01	1.000	1.0e+00
2.0	2.000	2.0e+00	0.337	2.2e-01	0.683	2.4e-01	0.562	4.0e-01	2.000	1.0e+00
4.0	8.000	4.0e+00	0.633	6.9e-02	0.954	5.4e-02	1.074	1.1e-01	4.000	1.0e+00
8.0	32.000	8.0e+00	0.693	2.0e-04	1.000	1.3e-04	1.155	7.2e-05	8.000	1.0e+00

At half a standard deviation all five slopes lie between 0.1 and 1, and any of the comparisons would train the model. At $d = 8$ the Jensen–Shannon divergence reads 0.693 and total variation 1.000, the constants $\log 2$ and 1 of the point masses, and both slopes are near 10^{-4}. Overlap in the tails does not help. The KL divergence keeps a slope, d itself, but only because a Gaussian model has a density at every data point; a model whose samples lie on a thin surface has none. That leaves the two comparisons that kept moving for the point masses. MMD^2 grows at first and has stalled by $d = 8$, at 1.155 with a slope below 10^{-4}. W_1 equals d, with slope 1 at every distance.

Neither is built from the ratio p/q. MMD compares how P and Q *average* a class of functions; samples can estimate averages, and the class decides how far the comparison sees. W_1 asks how far mass must travel to turn one distribution into the other, and it turns out to be a comparison of averages too. Two results stand out. Over a fixed set of features the comparison of averages has a closed form, with no second network to train. And the cost of moving mass equals the largest gap in averages over functions of slope at most one, the theorem behind the Wasserstein GAN of Section 16.2. P is the data and Q the model, as before, and the examples are on the line.

37.4.1 Integral Probability Metrics and MMD

37.4.1.1 Classes of Test Functions

Total variation is a metric, and still it went flat in the opening table. Its second form shows why, and what to change: total variation is the largest disagreement between P and Q about the probability of any event A (Section 37.3.3), $\mathrm{TV}(P, Q) = \sup_A |P(A) - Q(A)|$. The probability of an event is the average of its indicator function, so total variation is a supremum, over a class of functions, of the difference between two averages. Nothing obliges us to use indicators. For a class $\mathcal{G}$ of *test functions* g, the **integral probability metric** (IPM) (Müller 1997) is

$$\mathrm{IPM}_{\mathcal{G}}(P, Q) = \sup_{g \in \mathcal{G}} \left(\mathbb{E}_{x \sim P}[g(x)] - \mathbb{E}_{x \sim Q}[g(x)] \right). \tag{37.34}$$

The class $\mathcal{G}$ is a panel of auditors, and the IPM reports the largest gap in averages that any auditor in the panel can find. Our classes contain $-g$ along with every g, so the gap is never negative and is the same in both directions; whether it vanishes only for $P = Q$, as the word metric promises, depends on the class. The critic of Section 37.3.4 was an auditor too. In its

bound, $D_f(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[f^*(T)]$ for every function T, the conjugate f^* keeps an unrestricted critic in check. An IPM averages g itself under both distributions, so nothing restrains g except membership in $\mathcal{G}$, and everything depends on the class.

Three classes matter to us. Functions with $|g| \leq \frac{1}{2}$ give total variation again: an indicator minus $\frac{1}{2}$ is such a function, the constant cancels in the difference, and $g = \frac{1}{2}\operatorname{sign}(p-q)$ attains the supremum. Functions that are linear in a fixed set of features give the **maximum mean discrepancy** (MMD) (Gretton et al. 2012), which we build next. Functions of slope at most one give the Wasserstein distance of Section 37.4.2.

Why not take the largest class every time? A class has to meet two requirements, and size works against the first: averages over a sample must track averages over the distribution. Samples estimate expectations directly, with no density ratio, but a panel that is too large finds gaps that are not there. Two samples of $1{,}000$ points from the same continuous distribution share no point, so the indicator of the first sample has mean 1 on it and 0 on the other: the empirical total variation is 1 whatever the distributions are. A class must be small or smooth enough to exclude such functions.

The second requirement is that the gap keep growing as the distributions move apart, and for that the class needs functions that grow. Total variation has none: its test functions are bounded, so its gap cannot exceed 1. The largest disagreement is about the event $\{p > q\}$, which for the two Gaussians of the opening is the half-line $x < d/2$. Its two probabilities give TV $= \Phi(d/2) - \Phi(-d/2) = 2\Phi(d/2) - 1$, where Φ is the cumulative distribution function (CDF) of the standard normal. This is 0.954 at $d = 4$ and within 10^{-4} of 1 at $d = 8$. Once an indicator separates the two bell curves almost perfectly, moving them further apart changes nothing. Total variation fails both requirements, and we shall hold the next two classes to them.

37.4.1.2 Mean Features and Kernels

We build a better class from the smallest panel up, and try each stage on one pair: $\mathcal{N}(0,1)$ against $\mathcal{N}(0,4)$, which differ in spread and not in mean. First, take linear test functions $g(x) = w\,x$ with $|w| \leq 1$. Then $\mathbb{E}_P g - \mathbb{E}_Q g = w\,(\mathbb{E}_P X - \mathbb{E}_Q X)$, and the supremum is $|\mathbb{E}_P X - \mathbb{E}_Q X|$: the IPM over linear functions is the difference of the means. It cannot see a change of spread, and on our pair it reports 0.

Second, add a feature. Take the feature vector $\varphi(x) = (x, x^2)$ and the test functions $g(x) = \mathbf{w}^\top \varphi(x)$ with $\|\mathbf{w}\| \leq 1$, and write $\mathbf{m}_P = \mathbb{E}_P[\varphi(x)]$ for the *mean feature* of P. Then $\mathbb{E}_P g - \mathbb{E}_Q g = \mathbf{w}^\top(\mathbf{m}_P - \mathbf{m}_Q)$, and by the Cauchy–Schwarz inequality the largest value of a linear function on the unit ball is the norm of its coefficient vector:

$$\mathrm{MMD}(P,Q) = \sup_{\|\mathbf{w}\|\leq 1} \mathbf{w}^\top(\mathbf{m}_P - \mathbf{m}_Q) = \|\mathbf{m}_P - \mathbf{m}_Q\|. \tag{37.35}$$

MMD is the distance between the mean features, and the supremum has gone. On our pair the mean features are $(0,1)$ and $(0,4)$, so $\mathrm{MMD}^2 = 0^2 + 3^2 = 9$: means and second moments are now both compared.

Third, we expand the squared norm. A product of two means is the mean of a product over independent draws, so $\|\mathbf{m}_P\|^2 = \mathbb{E}\langle\varphi(x), \varphi(x')\rangle$ for independent x, x' from P. With independent y, y' from Q as well, $\|\mathbf{m}_P - \mathbf{m}_Q\|^2 = \mathbb{E}\langle\varphi(x), \varphi(x')\rangle + \mathbb{E}\langle\varphi(y), \varphi(y')\rangle -$

$2\,\mathbb{E}\langle\varphi(x),\varphi(y)\rangle$. Only inner products of features appear. We call $k(x,x') = \langle\varphi(x),\varphi(x')\rangle$ the **kernel** and obtain

$$\mathrm{MMD}^2(P,Q) = \mathbb{E}_{x,x'\sim P}[k(x,x')] + \mathbb{E}_{y,y'\sim Q}[k(y,y')] - 2\,\mathbb{E}_{x\sim P,y\sim Q}[k(x,y)]. \quad (37.36)$$

The first two terms measure similarity within each distribution and the third similarity across the two. If draws from P and Q interleave, the three terms cancel; if they form separate clumps, the within terms exceed the across term. The features of the first step have the *linear kernel* $k(x,x') = xx'$, and those of the second the *quadratic kernel* $xx' + x^2x'^2$.

The quadratic kernel still has a blind spot: two distributions with equal means and equal variances are at distance zero. We could add x^3, then x^4, and never finish. The fourth step therefore chooses the kernel directly, as a measure of similarity, and never writes the features down. The *RBF kernel* $k(x,x') = e^{-(x-x')^2/(2\ell^2)}$ calls two points similar when they lie within about a *length scale* ℓ of each other. It is an inner product of features all the same. It factors as $e^{-x^2/(2\ell^2)}\,e^{-x'^2/(2\ell^2)}\,e^{xx'/\ell^2}$, and the power series of the last factor writes it as an inner product of infinitely many features $\varphi_j(x) = x^j e^{-x^2/(2\ell^2)}/(\ell^j\sqrt{j!})$, $j = 0,1,2,\ldots$, damped monomials of every order with weights that decrease quickly. The panel now has infinitely many auditors, and its verdict is still three averages. In $\mathbb{R}^n$ the squared difference in the kernel becomes a squared norm and nothing else changes.

No blind spot is left. With the RBF kernel, $\mathrm{MMD}(P,Q) = 0$ only if $P = Q$ (Gretton et al. 2012), so this MMD is a metric. A kernel with this property is called *characteristic*; the linear and quadratic kernels are not.

For Gaussians the RBF averages can be computed by hand. For $Z \sim \mathcal{N}(\mu, s^2)$, a Gaussian integral gives

$$\mathbb{E}\big[e^{-Z^2/(2\ell^2)}\big] = \Big(1 + \frac{s^2}{\ell^2}\Big)^{-1/2} \exp\Big(-\frac{\mu^2}{2(\ell^2+s^2)}\Big), \quad (37.37)$$

and the difference of two independent Gaussians is Gaussian with the variances added. On our pair at $\ell = 1$ the three differences $x-x'$, $y-y'$ and $x-y$ have mean zero and variances 2, 8 and 5, so $\mathrm{MMD}^2 = 3^{-1/2} + 9^{-1/2} - 2\cdot 6^{-1/2} \approx 0.094$. For the two Gaussians of the opening the same integral gives $\frac{2}{\sqrt{3}}(1 - e^{-d^2/6})$, the MMD^2 column of the opening table (Exercise 1a). For the point masses of Table 37.2 there is nothing to average, and Equation 37.36 gives $1 + 1 - 2k(0,d) = 2 - 2e^{-d^2/2}$, the fourth row of that table. In both formulas only the cross term depends on d, and it is gone within a few length scales. The gap stops growing, and the second requirement is not met. An RBF auditor is smooth, and it is nearsighted.

37.4.1.3 Estimation from Samples

The first requirement is where a kernel does well. From samples, each expectation in Equation 37.36 becomes an average over pairs. The unbiased estimator (Gretton et al. 2012) leaves out the pairs of a point with itself, which would bias the within terms upward. We apply it to $N = 1{,}000$ draws from $\mathcal{N}(0,1)$ and as many from $\mathcal{N}(0,4)$ under the linear, quadratic and RBF kernels, for which we have just found 0, 9 and 0.094. With the RBF kernel we also compare the first sample with a second one from $\mathcal{N}(0,1)$, where the answer is 0,

and with one from $\mathcal{N}(0.5, 1)$, where the opening table says 0.047. What the mathematics does not supply is the size of the sampling error.

```
def mmd2_unbiased(x, y, k):
    """Unbiased MMD^2 between samples x and y for the kernel k(a, b)."""
    kxx = k(x[:, None], x[None, :])
    kyy = k(y[:, None], y[None, :])
    n, m = len(x), len(y)
    return ((kxx.sum() - np.trace(kxx)) / (n * (n - 1))
            + (kyy.sum() - np.trace(kyy)) / (m * (m - 1))
            - 2 * k(x[:, None], y[None, :]).mean())

linear = lambda a, b: a * b                          # features phi(x) = x
quadratic = lambda a, b: a * b + a ** 2 * b ** 2     # features (x, x^2)
rbf = lambda a, b: np.exp(-(a - b) ** 2 / 2)         # RBF kernel, ell = 1

rng = np.random.default_rng()
n = 1000
x_p = rng.standard_normal(n)                 # P = N(0, 1)
y_var = 2 * rng.standard_normal(n)           # N(0, 4)
y_same = rng.standard_normal(n)              # N(0, 1) again
y_shift = rng.standard_normal(n) + 0.5       # N(0.5, 1)
for name, y, k, pop in [
        ('linear,    N(0,1) vs N(0,4)', y_var, linear, 0.0),
        ('quadratic, N(0,1) vs N(0,4)', y_var, quadratic, 9.0),
        ('RBF,       N(0,1) vs N(0,4)', y_var, rbf,
         3 ** -0.5 + 9 ** -0.5 - 2 * 6 ** -0.5),
        ('RBF,       N(0,1) vs N(0,1)', y_same, rbf, 0.0),
        ('RBF,       N(0,1) vs N(0.5,1)', y_shift, rbf,
         2 / np.sqrt(3) * (1 - np.exp(-1 / 24)))]:
    print(f'{name:30s} MMD^2 = {mmd2_unbiased(x_p, y, k):+.4f}  '
          f'(population {pop:.4f})')
```

```
linear,    N(0,1) vs N(0,4)    MMD^2 = -0.0033  (population 0.0000)
quadratic, N(0,1) vs N(0,4)    MMD^2 = +10.2618  (population 9.0000)
RBF,       N(0,1) vs N(0,4)    MMD^2 = +0.1180  (population 0.0942)
RBF,       N(0,1) vs N(0,1)    MMD^2 = +0.0001  (population 0.0000)
RBF,       N(0,1) vs N(0.5,1)  MMD^2 = +0.0351  (population 0.0471)
```

The linear kernel reports a value near zero because the means agree. Its estimate is the squared difference of the two sample means, less an estimate of the variance of that difference, about $(1 + 4)/1{,}000 = 0.005$. It therefore cannot fall much below -0.005, while about one run in twenty exceeds $+0.015$. The quadratic kernel reports about 9, give or take 3; its estimate involves fourth powers and is noisy. The RBF kernel reports about 0.094, give or take 0.03, and for the shifted pair about 0.047, typically within 0.02, an error that can reach a third of the value. Between the two samples of $\mathcal{N}(0, 1)$ the RBF estimate is far steadier, within a few thousandths of zero and sometimes below it, because the errors of the three averages then cancel to first order. Where the indicators found a gap of 1 between two samples of one distribution, the kernel finds none.

The RBF estimator measures the size of a difference poorly and detects its presence well, and this makes it a *two-sample test*: from $1{,}000$ points, a value of a few hundredths is strong evidence that the samples differ in distribution. No critic was trained and no density was

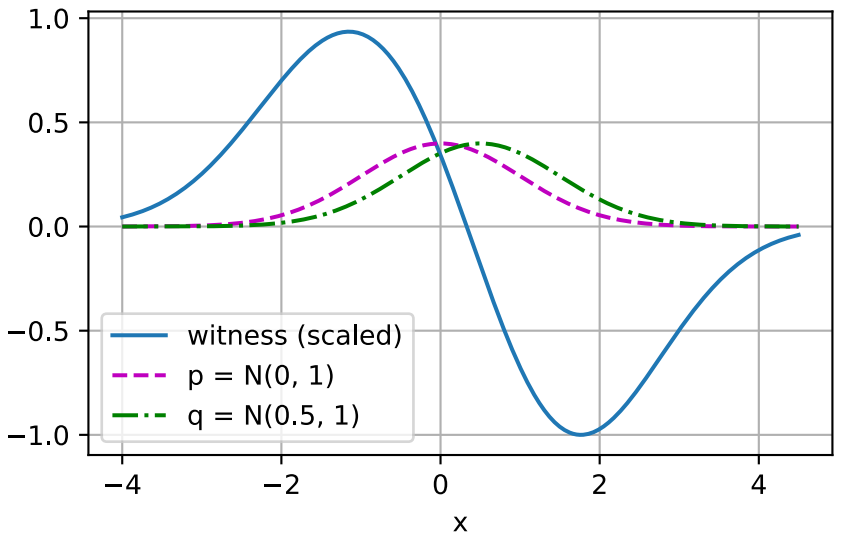
evaluated.

37.4.1.4 The Witness

An IPM is a supremum, so one may ask which auditor found the gap. For MMD the supremum in Equation 37.35 is attained by $\mathbf{w}^\star \propto \mathbf{m}_P - \mathbf{m}_Q$, and the corresponding test function, the **witness**, $g^\star(x) \propto \langle \mathbf{m}_P - \mathbf{m}_Q, \varphi(x) \rangle = \mathbb{E}_P[k(x, X)] - \mathbb{E}_Q[k(x, Y)]$, needs only kernel evaluations. For the shifted pair, $\mathcal{N}(0, 1)$ against $\mathcal{N}(0.5, 1)$, the two densities are mirror images about $x = 0.25$, so the witness of the two distributions changes sign there. We estimate the witness by sample means on a grid and plot it with the two densities. The cell also prints $\mathbb{E}_P g - \mathbb{E}_Q g$ for the unscaled witness $g = \langle \mathbf{m}_P - \mathbf{m}_Q, \varphi \rangle$. This gap is $\|\mathbf{m}_P - \mathbf{m}_Q\|^2$, which is MMD^2 once more, estimated this time with the self-pairs included.

```
g = lambda s: (rbf(s[:, None], x_p[None, :]).mean(axis=1)    # the unscaled
               - rbf(s[:, None], y_shift[None, :]).mean(axis=1))  # witness
grid = np.linspace(-4.0, 4.5, 400)
witness = g(grid)
print(f'witness changes sign at x = {grid[np.argmax(witness < 0)]:.2f}')
print(f'E_P g - E_Q g = {g(x_p).mean() - g(y_shift).mean():.4f}')
d2l.plot(grid, [witness / np.abs(witness).max(), norm.pdf(grid),
                norm.pdf(grid, loc=0.5)], 'x', '',
         legend=['witness (scaled)', 'p = N(0, 1)', 'q = N(0.5, 1)'],
         figsize=(5, 3))
```

```
witness changes sign at x = 0.35
E_P g - E_Q g = 0.0360
```



The witness is positive where P has more mass and negative where Q has more. It changes sign near $x = 0.25$; with $1{,}000$ points per sample the crossing moves by a tenth or two from draw to draw. The printed gap agrees with the last line of the previous output, the unbiased estimate for the same two samples, to about 0.001, and the difference is the self-pairs.

The ends of the plot say more than the crossing. A model that lowers the gap moves each of its samples uphill on the test function, so the slope of the test function at a sample is the push which that sample receives. The witness is steep between the two bell curves, and a few length scales away it levels off toward zero. A model sample out there is told nothing. This is the MMD^2 column of the opening table, seen from one sample.

MMD with a fixed kernel therefore serves twice: as a two-sample test, and as a loss on which a model is trained without an adversary (Li et al. 2015). The kernel inception distance by which generated images are scored is the unbiased estimate of the previous subsection, applied to image features with a cubic kernel (Section 16.5). The length scale has to be chosen, and a poor choice is costly: at $\ell = 1$ the slope at $d = 8$ was 7×10^{-5} (Exercise 1b tries longer scales). Common practice sets ℓ to the median distance between sample points, or adds RBF kernels of several length scales, as Section 16.2 does when it trains the model of a GAN, its *generator*, on MMD.

37.4.2 Optimal Transport and the Wasserstein Distance

One column of the opening table never went flat. W_1 kept its slope because it measures distance itself, which the f-divergences ignore and a kernel sees only nearby: $\delta_{0.01}$ is near δ_0 because its mass need only move 0.01. The idea is old. In 1781 Gaspard Monge asked how a heap of soil can be moved into a pit of the same volume with the least work, and the least work is a measure of how far the heap is from the pit.

37.4.2.1 Transport Plans

Take two distributions on the points $0, 1, \dots, 5$ of the line, with P mostly on the left and Q mostly on the right, as in the top left panel of Figure 37.10. To turn P into Q we must say how much mass travels from each source x to each destination y. Such a *transport plan* is a joint distribution γ of pairs (x, y) whose marginals are P and Q: what leaves each x is what P has there, and what arrives at each y is what Q needs. A plan is also called a *coupling* of P and Q. The **Wasserstein-1 distance**, also called the earth mover's distance, is the least total of mass times distance over all plans. Writing $\Pi(P, Q)$ for the set of plans, for P and Q on $\mathbb{R}^n$ with finite first moments,

$$W_1(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}\big[\|x - y\|\big]. \tag{37.38}$$

The 1 in the name is the power of the distance; the second power comes later. Between the point masses δ_0 and δ_d the only plan moves all mass from 0 to d, so $W_1(\delta_0, \delta_d) = |d|$, with gradient ± 1 for $d \neq 0$. For the two Gaussians of the opening, shifting every point by d is a plan of cost d, and none is cheaper, because $\mathbb{E}_\gamma|x - y| \geq |\mathbb{E}_\gamma[x - y]| = d$ for every plan. In both cases W_1 is the distance itself, the training signal that the f-divergences lack.

37.4.2.2 Kantorovich–Rubinstein Duality

As a recipe for computation the definition is discouraging: it optimizes over joint distributions, which are hard even to parameterize from samples. On finitely many points it is at least a familiar kind of problem, a linear program. The unknowns are the entries of the plan, and the cost and the constraints on the marginals are linear in them. A linear program has a *dual*, a maximization over multipliers for its constraints (Section 35.7.1). The

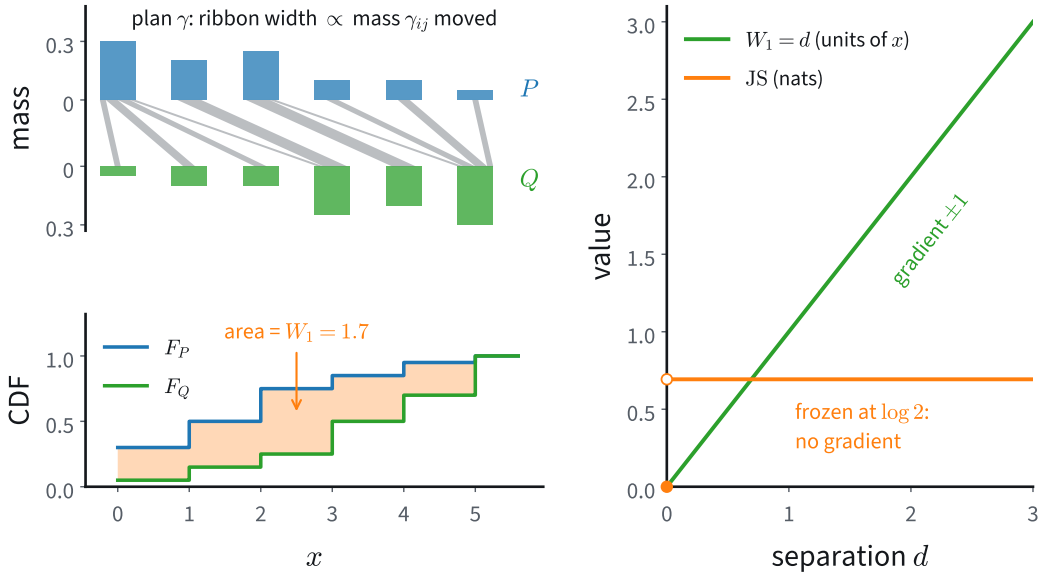


Figure 37.10: Transport on the line. Top left: a plan between two distributions on six points; each ribbon carries mass from a point of P (top) to a point of Q (bottom), and its width is the mass moved. Bottom left: the CDFs of the same P and Q; the shaded area between them is 1.7. Right: for two point masses at separation d, the Jensen–Shannon divergence (in nats) stays at $\log 2$ while W_1 (in units of x) grows with d.

dual of transport optimizes over a single function, and it is the reason W_1 matters to deep learning.

Proposition (Kantorovich–Rubinstein duality). *For distributions P and Q on $\mathbb{R}^n$ with finite first moments,*

$$W_1(P, Q) = \sup_{\|g\|_{\mathrm{Lip}} \le 1} \left(\mathbb{E}_{x \sim P}[g(x)] - \mathbb{E}_{y \sim Q}[g(y)] \right), \tag{37.39}$$

where the supremum runs over all 1-Lipschitz functions g, those with $|g(x) - g(y)| \le \|x - y\|$ for all x, y.

An infimum over plans has become a supremum over one function: W_1 is the IPM Equation 37.34 over test functions of slope at most 1, the third of our classes. Such an auditor cannot jump, so it cannot pick out the points of a sample as an indicator does. It can climb without limit, however, and that is what the second requirement asked for. For the two Gaussians of the opening take $g(x) = -x$. Its averages are 0 under P and $-d$ under Q, a gap of d, which is W_1, so no auditor of slope one does better. Its slope is the same at every model sample, however far from the data the sample lies, and uphill on g is the way to the data.

A network can represent such a function, and the **WGAN** (Arjovsky et al. 2017) trains this dual, with a neural critic in the role of g that is constrained to be approximately 1-Lipschitz. Section 16.2 sets this critic beside the other adversarial losses, and Section 16.4 treats the

gradient penalties that enforce the constraint. What an approximately Lipschitz critic reports is, of course, no longer exactly W_1.

37.4.2.3 Transport on the Line

We prove the duality only on the line, where the proof also yields a closed form and transport needs no optimization at all; Peyré and Cuturi (2019) give the general proof. The idea is to count crossings. At a point t, P has a fraction $F_P(t)$ of its mass at or to the left of t and Q has $F_Q(t)$, so any plan must carry at least $|F_P(t) - F_Q(t)|$ of mass across t. A parcel moved from x to y crosses every t between them, so its distance $|x - y|$ is the length of the set of points it crosses, and the total cost is the mass crossing each t, integrated over t.

Proposition (W_1 on the line). *For distributions P and Q on $\mathbb{R}$ with CDFs F_P, F_Q and finite first moments,*

$$W_1(P, Q) = \int_{-\infty}^{\infty} |F_P(t) - F_Q(t)|\, dt = \sup_{\|g\|_{\mathrm{Lip}} \le 1} \big(\mathbb{E}_P[g] - \mathbb{E}_Q[g]\big), \tag{37.40}$$

so Kantorovich–Rubinstein duality holds on the line.

Proof. *Step 1: every plan costs at least the area.* For $(X, Y) \sim \gamma$, $|X - Y| = \int \mathbf{1}(t \text{ lies between } X \text{ and } Y)\, dt$. The integrand is non-negative, so expectation and integral may be exchanged:

$$\mathbb{E}_\gamma|X-Y| = \int \Big(\gamma(X \le t < Y) + \gamma(Y \le t < X)\Big)\, dt \ge \int \big|\gamma(X \le t < Y) - \gamma(Y \le t < X)\big|\, dt.$$

The step that is not routine is the difference inside the last integral. Add and subtract $\gamma(X \le t, Y \le t)$: then $\gamma(X \le t < Y) - \gamma(Y \le t < X) = \gamma(X \le t) - \gamma(Y \le t) = F_P(t) - F_Q(t)$, because the marginals of γ are P and Q. So every plan costs at least $\int |F_P - F_Q|$.

Step 2: one plan costs exactly the area. Send quantile to quantile: the lowest tenth of P goes to the lowest tenth of Q, and so on upward. In symbols, let U be uniform on $(0, 1)$ and set $X = F_P^{-1}(U)$, $Y = F_Q^{-1}(U)$, with the quantile function $F^{-1}(\lambda) = \inf\{x : F(x) \ge \lambda\}$ for $\lambda \in (0, 1)$, which satisfies $F^{-1}(\lambda) \le t$ exactly when $\lambda \le F(t)$. This is a plan, because $\Pr(X \le t) = \Pr(U \le F_P(t)) = F_P(t)$ and likewise for Y. We call it the *monotone plan*, since X and Y rise together. Then $\gamma(X \le t < Y) = \Pr\big(F_Q(t) < U \le F_P(t)\big)$ and $\gamma(Y \le t < X) = \Pr\big(F_P(t) < U \le F_Q(t)\big)$. At least one of the two is zero: the monotone plan never carries mass both ways across a point. The inequality of Step 1 is therefore an equality for this plan, and the least cost over plans equals the area.

Step 3: no test function beats any plan. This is *weak duality*, and it holds in any dimension. For a 1-Lipschitz g and a plan γ, $g(x) - g(y) \le \|x - y\|$ at every pair, and taking the expectation under γ gives $\mathbb{E}_P[g] - \mathbb{E}_Q[g] \le \mathbb{E}_\gamma\|x - y\|$ (compare the weak duality of Section 35.7.1).

Step 4: one test function attains the area. It suffices to consider $g(x) = g(0) + \int_0^x g'(t)\, dt$ with $|g'| \le 1$: every such g is 1-Lipschitz, and Step 3 already bounds all the others. The constant $g(0)$ cancels in $\mathbb{E}_P[g] - \mathbb{E}_Q[g]$. Writing $\int_0^X g'(t)\, dt = \int_0^\infty g'(t)\, \mathbf{1}(X >$

$t)\,dt - \int_{-\infty}^{0} g'(t)\,\mathbf{1}(X \le t)\,dt$ and swapping expectation and integral (Fubini's theorem from Section 34.6, which the finite first moments justify),

$$\mathbb{E}_P[g] - \mathbb{E}_Q[g] = \int_{-\infty}^{\infty} g'(t)\,(F_Q(t) - F_P(t))\,dt.$$

The choice $g'(t) = \operatorname{sign}(F_Q(t) - F_P(t))$ makes the right side equal to the area. Together, the supremum over test functions is at least the area (Step 4) and at most the least cost over plans (Step 3), which equals the area (Steps 1 and 2). ■

On the line, then, W_1 is the area between the two CDFs, and for two samples of equal size it is the mean absolute difference of the sorted values (Exercise 4a). It is rather surprising that a linear program should collapse this far. We watch it happen once, on the six points of Figure 37.10. Between neighboring points the two CDFs differ by 0.25, 0.35, 0.5, 0.35 and 0.25, which sum to 1.7, the shaded area in the figure. The cell computes W_1 three times. It adds up the five gaps. It solves the linear program over plans, where a plan is a 6×6 matrix with row sums p and column sums q. And it builds the monotone plan of Step 2, in which each route carries the overlap of two quantile intervals, and adds up its cost. The last line of the output compares the plan that the linear program returned with the monotone plan.

```
atoms = np.array([0.0, 1.0, 2.0, 3.0, 4.0, 5.0])
p_w = np.array([0.30, 0.20, 0.25, 0.10, 0.10, 0.05])
q_w = np.array([0.05, 0.10, 0.10, 0.25, 0.20, 0.30])

F_p, F_q = np.cumsum(p_w), np.cumsum(q_w)
w1_cdf = np.sum(np.abs(F_p - F_q)[:-1] * np.diff(atoms))

C = np.abs(atoms[:, None] - atoms[None, :])      # cost matrix |x_i - y_j|
n_atoms = len(atoms)
A_eq = np.zeros((2 * n_atoms, n_atoms * n_atoms))
for i in range(n_atoms):
    A_eq[i, i * n_atoms:(i + 1) * n_atoms] = 1   # row sums    = p
    A_eq[n_atoms + i, i::n_atoms] = 1            # column sums = q
res = optimize.linprog(C.ravel(), A_eq=A_eq,
                       b_eq=np.concatenate([p_w, q_w]), method='highs')
plan_lp = res.x.reshape(n_atoms, n_atoms)
# Monotone plan (Step 2): source i holds the quantiles (F_p[i-1], F_p[i]],
# destination j holds (F_q[j-1], F_q[j]]; route i -> j carries their overlap
lo_p, lo_q = F_p - p_w, F_q - q_w
mono = np.clip(np.minimum(F_p[:, None], F_q[None, :])
               - np.maximum(lo_p[:, None], lo_q[None, :]), 0, None)
print(f'W1 from the CDF gaps     : {w1_cdf:.4f}')
print(f'W1 from the LP           : {res.fun:.4f}')
print(f'cost of the monotone plan: {np.sum(mono * C):.4f}')
print(f'LP plan = monotone plan  : {np.allclose(plan_lp, mono)}, '
      f'mass the LP plan moves leftward: {np.tril(plan_lp, -1).sum():.2f}')
```

```
W1 from the CDF gaps     : 1.7000
W1 from the LP           : 1.7000
cost of the monotone plan: 1.7000
LP plan = monotone plan  : False, mass the LP plan moves leftward: 0.00
```

All three computations give $W_1 = 1.7$: one sum over five gaps replaces a 36-variable linear program. The last line is the unexpected one. The linear program did not return the monotone plan. It returned another plan of the same cost, which also moves no mass to the left.

The optimal plan is not unique here, then, and the reason is simple. $F_P \geq F_Q$ at every point, so no mass has to move left, and for a plan that moves mass only rightward $|x-y| = y-x$. Its cost is then the difference of the two means, $3.35 - 1.65 = 1.7$, whatever else the plan does. The monotone plan, the one drawn in the figure, is one of many optimal plans.

Two tools that we meet elsewhere measure transport with the square of the distance: the Fréchet inception distance, the other score for generated images (Section 16.5), and the minibatch pairing of flow matching, which matches noise to data by the plan of least squared distance (Section 38.6). The **Wasserstein-2 distance** squares the distance before averaging and takes the square root afterwards, $W_2(P,Q) = \left(\inf_\gamma \mathbb{E}_\gamma \|x-y\|^2\right)^{1/2}$. Under the squared cost two ribbons that cross always cost more than the same two uncrossed, so on the line the monotone plan is the only optimum (Exercise 4d). Between two Gaussians on the line with means μ_1, μ_2 and standard deviations σ_1, σ_2 it gives $W_2^2 = (\mu_1-\mu_2)^2 + (\sigma_1-\sigma_2)^2$ (Exercise 6a). The Fréchet inception distance is the same expression for Gaussians fitted to image features, with covariance matrices in place of the variances.

37.4.2.4 Entropic Regularization and Sinkhorn Iterations

This subsection can be skipped on a first reading. Off the line there is in general no CDF formula, and the linear program has N^2 variables for N points on each side. The standard remedy is **entropic regularization** (Cuturi 2013): we add the term $\varepsilon \sum_{ij} \gamma_{ij}(\log \gamma_{ij} - 1)$ to the transport cost $\sum_{ij} \gamma_{ij} C_{ij}$, for a cost matrix $\mathbf{C}$ and a regularizer $\varepsilon > 0$. On plans, $\sum_{ij} \gamma_{ij} = 1$, so the added term is $-\varepsilon$ times the entropy of γ plus a constant; it penalizes plans that concentrate on few routes. With it, transport became usable as a loss in deep learning, and generative models are trained on the regularized cost directly (Genevay et al. 2018).

The regularized problem can be solved almost in closed form. Its Lagrangian, as in Section 35.7.1, adds a multiplier ξ_i for each row-sum constraint $\sum_j \gamma_{ij} = p_i$ and ζ_j for each column-sum constraint $\sum_i \gamma_{ij} = q_j$. Setting its derivative in γ_{ij} to zero gives $C_{ij} + \varepsilon \log \gamma_{ij} - \xi_i - \zeta_j = 0$, so $\gamma_{ij} = e^{-(C_{ij}-\xi_i-\zeta_j)/\varepsilon}$: a route carries mass according to how far its cost exceeds $\xi_i + \zeta_j$, in units of ε. In matrix form,

$$\gamma_\varepsilon = \operatorname{diag}(\mathbf{a})\, \mathbf{K}\, \operatorname{diag}(\mathbf{b}), \qquad \mathbf{K} = e^{-\mathbf{C}/\varepsilon} \text{ (entrywise)}, \tag{37.41}$$

with positive scalings $\mathbf{a} = e^{\xi/\varepsilon}$ and $\mathbf{b} = e^{\zeta/\varepsilon}$. The regularized plan is the *Gibbs kernel* $\mathbf{K}$ (a matrix of positive weights, unrelated to the kernels of MMD), rescaled row by row and column by column. Only the scalings are unknown, and each half of the constraints determines one of them. With $\mathbf{b}$ fixed, $\mathbf{a} = \mathbf{p}/(\mathbf{K}\mathbf{b})$ (entrywise) makes the row sums equal to $\mathbf{p}$. With $\mathbf{a}$ fixed, $\mathbf{b} = \mathbf{q}/(\mathbf{K}^\top \mathbf{a})$ makes the column sums equal to $\mathbf{q}$.

The **Sinkhorn iterations** alternate these two updates, first $\mathbf{b}$ and then $\mathbf{a}$. After the second update the row sums are right, and we stop once the column sums agree with $\mathbf{q}$ to within 10^{-9} as well. Each iteration costs two matrix-vector products, so the iterations run well on a GPU and can be differentiated through. We report the *entropic cost*, the transport cost

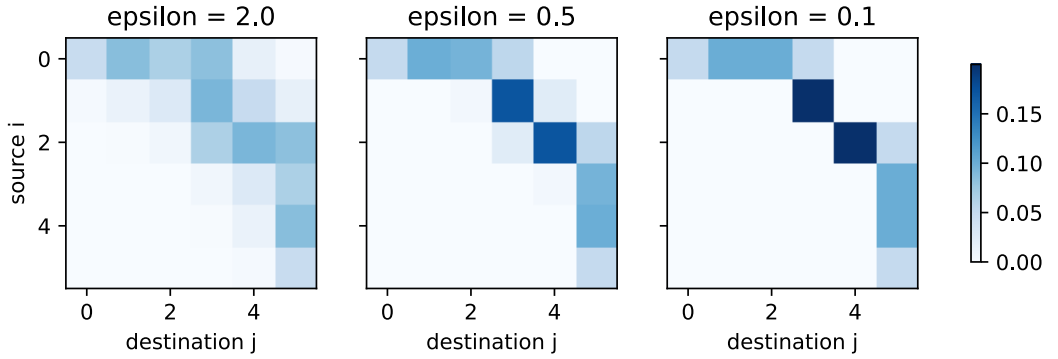
$\sum_{ij}(\gamma_\varepsilon)_{ij}C_{ij}$ of the regularized plan without the entropy term. As $\varepsilon \to 0$ it approaches the unregularized optimum, but the iterations converge ever more slowly (Peyré and Cuturi 2019), and the entries of $\mathbf{K}$ underflow, so practical code iterates on $\log \mathbf{a}$ and $\log \mathbf{b}$ (Exercise 7b).

To see how small ε must be, and what a small ε takes, we use the squared cost $C_{ij} = (x_i - x_j)^2$ on the six-point example, for which the monotone plan of Figure 37.10 is the only optimum and its ten routes cost $W_2^2 = 3.5$. The cell checks this value with the linear program and runs Sinkhorn at three values of ε. For each it prints the number of iterations, the entropic cost, and the number of routes that carry more than 10^{-3} of mass.

```
def sinkhorn(p, q, C, eps, tol=1e-9, max_iters=10 ** 6):
    """Entropic OT by Sinkhorn iterations, stopped once the column sums are
    within tol of q (L1); returns the plan, its cost and the iterations."""
    K = np.exp(-C / eps)                        # the Gibbs kernel
    a = np.ones_like(p)
    for it in range(1, max_iters + 1):
        b = q / (K.T @ a)                       # match the column sums
        a = p / (K @ b)                         # match the row sums ...
        if np.abs(b * (K.T @ a) - q).sum() < tol:  # ... columns still right?
            break
    plan = a[:, None] * K * b[None, :]
    return plan, np.sum(plan * C), it

C2 = (atoms[:, None] - atoms[None, :]) ** 2      # squared cost (x_i - y_j)^2
res2 = optimize.linprog(C2.ravel(), A_eq=A_eq,
                        b_eq=np.concatenate([p_w, q_w]), method='highs')
print(f'squared cost, LP optimum : {res2.fun:.5f}')
plans, eps_values = [], (2.0, 0.5, 0.1)
for eps in eps_values:
    plan, cost, iters = sinkhorn(p_w, q_w, C2, eps)
    plans.append(plan)
    print(f'epsilon = {eps}: {iters:6d} iterations, entropic cost = '
          f'{cost:.5f}, routes above 1e-3 = {(plan > 1e-3).sum()}')
d2l.show_heatmaps(d2l.tensor(np.stack(plans)[None]), 'destination j',
                  'source i', titles=[f'epsilon = {e}' for e in eps_values],
                  figsize=(9, 2.8), cmap='Blues')
```

```
squared cost, LP optimum : 3.50000
epsilon = 2.0:     47 iterations, entropic cost = 4.06489, routes above 1e-3 = 25
epsilon = 0.5:    186 iterations, entropic cost = 3.56963, routes above 1e-3 = 14
epsilon = 0.1: 154325 iterations, entropic cost = 3.50002, routes above 1e-3 = 10
```



The linear program agrees on 3.5. At $\varepsilon = 2$ the plan spreads each source's mass over several destinations (25 of the 36 routes carry more than 10^{-3}), and its cost, 4.06, is well above the optimum. At $\varepsilon = 0.5$ the plan has 14 routes and costs 3.57. At $\varepsilon = 0.1$ only the ten routes of the monotone plan remain, and the cost, 3.50002, lies above the optimum by 2×10^{-5}. In the heat maps a broad band sharpens into a staircase. The yardstick for ε is the difference in cost between competing routes, which is of order 1 on this grid, and a tenth of it was small enough.

The iteration counts show what this accuracy takes. About 50 iterations were enough at $\varepsilon = 2$ and about 190 at $\varepsilon = 0.5$, but $\varepsilon = 0.1$ needed some $150{,}000$. Between the last two runs ε fell fivefold and the work grew 800 times. Sinkhorn iterations are fast only as long as the plan may stay blurred (Exercise 8b measures the growth).

37.4.3 Fixed Kernels and Trained Critics

MMD and W_1 divide the two requirements between them. The class behind a fixed kernel is small. MMD costs three averages, and the error of its estimate from N points is of order $N^{-1/2}$ in any dimension (Gretton et al. 2012), but it sees only a few length scales. The functions of slope at most one see at every distance, which is what a model far from its data needs. They cannot jump, and yet in many dimensions they are still a very large class: between two samples in dimension $n > 2$ the error of W_1 is of order $N^{-1/n}$ (Weed and Bach 2019). In 100 dimensions, halving that error takes 2^{100} times the data. Nobody estimates W_1 that way. A critic network searches a far smaller class than all functions of slope one, and Sinkhorn iterations serve where the problem is small.

Kernel and critic alike begin by drawing samples from the model. For some models that is the expensive step, and Section 37.5 asks what can be compared without it.

37.4.4 Summary

A little overlap does not bring the gradient back: for two unit Gaussians eight standard deviations apart, the Jensen–Shannon divergence and total variation have slopes near 10^{-4}. An integral probability metric compares instead how P and Q average a class of test functions, and the class decides both whether samples can estimate the gap and how far the comparison sees. Functions that are linear in fixed features give MMD, the distance between mean features: it is three kernel averages, which samples estimate with no critic, and it is blind beyond a few length scales. Functions of slope at most one give W_1, the least mass times distance that turns one distribution into the other, which keeps a slope of one at every sep-

aration and is what the critic of a Wasserstein GAN estimates. On the line W_1 is the area between the CDFs; elsewhere it needs a trained critic or, for small problems, Sinkhorn iterations on an entropically regularized plan.

37.4.5 Exercises

1. code **MMD in closed form.** Use the Gaussian integral Equation 37.37 for the RBF kernel with length scale ℓ, with $P = \mathcal{N}(0, 1)$ and $Q = \mathcal{N}(d, 1)$.
    1. Show that $\mathrm{MMD}^2 = \frac{2\ell}{\sqrt{\ell^2+2}}\left(1 - e^{-d^2/(2(\ell^2+2))}\right)$, and check that $\ell = 1$ gives the MMD^2 column of the opening table. Which length scale maximizes MMD^2 for small shifts d? What happens as $\ell \to 0$ and as $\ell \to \infty$?
    2. Evaluate the slope in d at $d = 8$ for $\ell = 1, 4$ and 16. What do the three values say about the length scale for a model that starts far from its data?
2. code **Sampling error of the MMD estimate.** At $\ell = 1$ and $d = 0.5$ the closed form $\frac{2}{\sqrt{3}}\left(1 - e^{-d^2/6}\right)$ gives 0.047.
    1. Recompute `mmd2_unbiased` with the RBF kernel on 200 fresh pairs of samples drawn like `x_p` and `y_shift`, with `n = 250` and with `n = 1000`, and compare the mean and the standard deviation of the estimates with the closed form. How does the standard deviation scale with the sample size? Is the value that your run of the MMD experiment printed within the spread?
    2. Repeat for pairs drawn like `x_p` and `y_same`. The standard deviation now scales differently. How, and why does this help a test of $P = Q$?
3. **Rescaling and the data-processing inequality.** An f-divergence is $D_f(P\|Q) = \mathbb{E}_{x\sim Q}[f(p(x)/q(x))]$ for a convex f with $f(1) = 0$, and it satisfies the data-processing inequality $D_f(P\kappa \,\|\, Q\kappa) \le D_f(P\|Q)$ for every channel κ, that is, for every deterministic or random map applied to both distributions (Section 37.3.3.3). Apply the map $x \mapsto 2x$ to both P and Q.
    1. Show from the definition that no f-divergence changes under an invertible map, and that W_1 doubles.
    2. Show that the RBF MMD^2 at $\ell = 1$ falls from 0.047 to 0.036 for $\mathcal{N}(0, 1)$ against $\mathcal{N}(0.5, 1)$, but rises from 0.235 to 0.787 for δ_0 against $\delta_{0.5}$. Why does the data-processing inequality say nothing about MMD and W_1?
4. code **One-dimensional transport.** ⊠ On the real line, W_1 is the area between the CDFs Equation 37.40.
    1. For two empirical distributions on samples $x_1, \dots, x_N$ and $y_1, \dots, y_N$ of equal size, use the monotone plan of the proof to show that $W_1 = \frac{1}{N}\sum_i |x_{(i)} - y_{(i)}|$, the mean absolute difference of the sorted values. Check this against the linear program of the six-point example on two random samples of size 7.
    2. Compute the sorted-sample W_1 between `x_p` and `y_shift` and between `x_p` and `y_same` of the MMD experiment. Then average it over 20 pairs of independent samples of size N from $\mathcal{N}(0, 1)$, for $N \in \{10^2, 10^3, 10^4\}$, and find the rate at which it decreases. The estimate is never negative. Why does this rule out unbiasedness when $P = Q$?
    3. Change `q_w` to $(0.40, 0.05, 0.05, 0.10, 0.10, 0.30)$, so that the two CDFs cross, and rerun the cell that computes W_1. Build the test function g with $g(0) = 0$ and

slope $\operatorname{sign}(F_Q(t) - F_P(t))$ between neighboring atoms, as in Step 4 of the proof, and check that $\mathbb{E}_P[g] - \mathbb{E}_Q[g]$ equals the LP value. For every route (i, j) that carries mass in the LP solution `plan_lp`, compare $g(x_i) - g(x_j)$ with $|x_i - x_j|$. Why must the two agree on every such route for any optimal plan and any optimal test function?

4. *Uniqueness under the squared cost.* Suppose a plan moves mass from x to y' and from x' to y, with $x < x'$ and $y < y'$, so that the two ribbons cross. Show that under the squared cost, sending the same mass from x to y and from x' to y' saves $2(x' - x)(y' - y)$ per unit of mass, and deduce that the monotone plan is the only optimal one. What does the same exchange save under the cost $|x - y|$ when all four moves go rightward? Find two ribbons that cross in `plan_lp`, the plan that the linear program returned for the six-point example, and check your answer on them.

5. **Optimal critics on the line.** Step 4 of the proof of Equation 37.40 shows that a test function with slope $g'(t) = \operatorname{sign}(F_Q(t) - F_P(t))$ attains W_1.

   1. For $P = \mathcal{N}(0, 1)$ and $Q = \mathcal{N}(d, 1)$ with $d > 0$, show that $F_P > F_Q$ everywhere, so that Step 4 returns $g(x) = -x$, the critic that the text found by inspection.
   2. For $P = \mathcal{N}(0, 1)$ and $Q = \mathcal{N}(0, 4)$, show that the two CDFs cross only at 0, that $g(x) = -|x|$ is optimal, and that $W_1 = \sqrt{2/\pi} \approx 0.80$. (For $X \sim \mathcal{N}(0, \sigma^2)$, $\mathbb{E}|X| = \sigma\sqrt{2/\pi}$.) The linear kernel reported 0 on this pair. Which way does the critic move the samples of Q?
   3. Let $P = \frac{1}{2}\delta_0 + \frac{1}{2}\delta_3$ and $Q = \frac{1}{2}\delta_1 + \frac{1}{2}\delta_3$. Find $W_1(P, Q)$ and describe every optimal critic of the form used in Step 4. Then show in general that such a critic is optimal if and only if $g'(t) = \operatorname{sign}(F_Q(t) - F_P(t))$ at almost every t with $F_P(t) \neq F_Q(t)$, the points that mass has to cross, while its slope elsewhere is free.
   4. A WGAN critic is often trained with a penalty on $(|g'(x)| - 1)^2$ at points x between the data and the generated samples (Gulrajani et al. 2017). Explain this choice by (c). Which critics are optimal when $P = Q$, and which of them does the penalty prefer? Section 16.4 shows what follows for training.

6. code W_2 **between Gaussians and the Fréchet distance.** On the line the monotone plan $X = F_P^{-1}(U)$, $Y = F_Q^{-1}(U)$ is optimal for the squared cost.

   1. Show that for $P = \mathcal{N}(\mu_1, \sigma_1^2)$ and $Q = \mathcal{N}(\mu_2, \sigma_2^2)$ the monotone plan is $Y = \mu_2 + (\sigma_2/\sigma_1)(X - \mu_1)$, and deduce that $W_2^2(P, Q) = (\mu_1 - \mu_2)^2 + (\sigma_1 - \sigma_2)^2$. With the same plan, recover $W_1 = |\mu_1 - \mu_2|$ for equal variances, the W_1 column of the opening table.
   2. Let both standard deviations be σ and the means d apart. Then $W_2^2 = d^2$ for every σ, while the KL divergence between the two Gaussians is $d^2/(2\sigma^2)$. What do the two report as $\sigma \to 0$, when the Gaussians become the point masses of Table 37.2?
   3. In $\mathbb{R}^n$ the squared W_2 between $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ is the squared *Fréchet distance* $\|\mu_1 - \mu_2\|^2 + \operatorname{tr}(\Sigma_1 + \Sigma_2 - 2\,(\Sigma_1^{1/2}\Sigma_2\Sigma_1^{1/2})^{1/2})$ (Dowson and Landau 1982). Check that it reduces to (a) for $n = 1$ and, for diagonal covariance matrices, to the sum of (a) over the coordinates. Prove the diagonal case directly: the squared cost is a sum over coordinates, so no plan costs less than the sum of the coordinatewise optima, and the monotone plans of the coordinates together

attain it.

4. The Fréchet inception distance fits a Gaussian to each of two feature distributions and reports the squared Fréchet distance between the fits. Let P be the mixture $0.7\,\mathcal{N}(-2, 0.6^2) + 0.3\,\mathcal{N}(2, 0.6^2)$ and $Q = \mathcal{N}(-0.8, 3.72)$, which has the same mean and variance. What is the Fréchet distance between the two fits? Estimate $W_2^2(P, Q)$ from 10^5 sorted draws of each (you should find about 0.63). What does a Fréchet distance of zero certify, and what does it not?

7. code **Entropic transport and Sinkhorn divergences.** Regularize the transport LP of the six-point example by the divergence from the independent plan,

$$\mathrm{OT}_\varepsilon(P, Q) = \min_\gamma \sum_{i,j} \gamma_{ij} C_{ij} + \varepsilon\, D_{\mathrm{KL}}(\gamma \,\|\, p\, q^\top),$$

where γ ranges over the plans with row sums p and column sums q.

1. Show that $D_{\mathrm{KL}}(\gamma \,\|\, p\, q^\top) = -H(\gamma) + H(P) + H(Q)$ for every such plan, where H denotes the entropy, so that `sinkhorn` returns the minimizer. Show that $\mathrm{OT}_\varepsilon(P, Q) \to \sum_{i,j} p_i q_j C_{ij}$ as $\varepsilon \to \infty$, and that $\mathrm{OT}_\varepsilon(P, P) > 0$ for every $\varepsilon > 0$ unless P is a point mass. With $C_{ij} = |x_i - x_j|$, show that the distributions P on the atoms that minimize the limit $\sum_{i,j} p_i q_j C_{ij}$ put all their mass on medians of Q, for `q_w` the atoms 3 and 4. What would a model trained on OT_ε with a large ε learn?
2. The Sinkhorn divergence $S_\varepsilon(P, Q) = \mathrm{OT}_\varepsilon(P, Q) - \frac{1}{2}\mathrm{OT}_\varepsilon(P, P) - \frac{1}{2}\mathrm{OT}_\varepsilon(Q, Q)$ removes this bias (Genevay et al. 2018; Feydy et al. 2019). Compute $S_\varepsilon(P, Q)$ for `p_w` and `q_w` with the absolute cost `C` at $\varepsilon \in \{10^{-3}, 10^{-2}, \dots, 10^{3}\}$. At $\varepsilon = 10^{-3}$ the entries of $\mathbf{K}$ underflow, so carry out the Sinkhorn updates on $\log \mathbf{a}$ and $\log \mathbf{b}$ there, with `scipy.special.logsumexp`. Check that S_ε approaches W_1 as $\varepsilon \to 0$ and $\frac{1}{2}\mathcal{E}(P, Q)$ as $\varepsilon \to \infty$, where $\mathcal{E}(P, Q) = 2\mathbb{E}|X - Y| - \mathbb{E}|X - X'| - \mathbb{E}|Y - Y'|$ is the energy distance (Székely and Rizzo 2013), the expression Equation 37.36 with $-|x - y|$ in place of the kernel. Derive the second limit from (a).

8. code **Sinkhorn iterations at small ε.** Use `sinkhorn`, `p_w` and `q_w` of the six-point example.

1. *Stopping early.* Call `sinkhorn` with the squared cost `C2` at $\varepsilon = 0.1$ and `max_iters=5000`, so that the loop ends before the column sums have settled. The cost that it returns is below the optimum 3.5. Print the column sums of the returned matrix minus `q_w`. Which destinations receive too little, and why does that lower the cost? Why can no matrix with row sums `p_w` and column sums `q_w` cost less than 3.5?
2. *The number of iterations.* Count the iterations that `sinkhorn` needs with `C2` at $\varepsilon \in \{2, 1, 0.5, 0.3, 0.2, 0.15, 0.1\}$, and plot the logarithm of the count against $1/\varepsilon$. What kind of growth do the points at small ε show, and what count does it predict for $\varepsilon = 0.05$?
3. *Ties.* Under the absolute cost `C` every plan that moves mass only rightward is optimal for `p_w` and `q_w`. Run `sinkhorn` with `C` at $\varepsilon = 0.1$ and 0.02. How many routes carry mass? Show that the limit plan has the form $\gamma_{ij} = a_i b_j$ for $j \geq i$ and 0 otherwise. On plans the regularizer is $-\varepsilon$ times the entropy plus a constant. Use this to explain why the limit is the plan of largest entropy among the optimal plans.

37.5 Score-Based Comparisons

A softmax turns ten logits into probabilities by exponentiating them and dividing by their sum, and nobody gives the sum a second thought. Now let a network assign such a number not to ten classes but to every possible input. By convention the sign is reversed: the network returns an energy $E(\mathbf{x})$, low for plausible images where a logit would be high, and the density is proportional to $\tilde{q}(\mathbf{x}) = e^{-E(\mathbf{x})}$. The sum becomes an integral over all images, $Z = \int \tilde{q}(\mathbf{x})\, d\mathbf{x}$, and nobody can compute it. Such an *energy-based model* can say how plausible an input is, but not how probable.

None of our comparisons applies to it. The likelihood needs the density $q = \tilde{q}/Z$. The others, by a critic (Section 37.3.4), a kernel or a transport plan (Section 37.4), need samples of the model, and an energy does not hand out samples. Each one has to be produced by a long random walk, many small noisy steps toward lower energy, and every step needs a gradient of the network (Section 38.5.2).

One such gradient is cheap, however, and it is all that we shall ask of the model. The gradient of $\log \tilde{q}$ with respect to the input, the slope of the log-density, costs one backward pass, and Z has disappeared from it, because a constant has no slope. This gradient is called the *score*, and this section compares distributions through scores alone. The mean squared difference of two scores is the objective of score matching, on which diffusion models (Chapter 17) are trained. It has a limit that anyone who trains such a model should know. A slope is local, and it cannot tell how mass is divided between modes that lie far apart. We measure how little it sees.

As before, P is the data, which we hold as samples, and Q is the model, which we know through its score s_Q. We ask what a difference of slopes says about two distributions and what it misses, and then how slopes can be compared at all when the data, a finite set of points, have none. Section 38.5.1 builds on these two answers. A third question, how to test a sample against a model that cannot be sampled, can be skipped on a first reading. The section ends with a table of training objectives and advice on choosing among them.

```
from scipy import integrate
```

The examples are Gaussians and mixtures of two Gaussians on the line. The helper functions evaluate the log-density of a mixture, with the largest term factored out so that the tails do not underflow (Section 35.8.2), and the KL divergence between two densities on a grid, in nats.

```
def log_components(x, w, mus, sigmas):
    """log(w_k N(x; mu_k, sigma_k^2)) for every component k; shape (k, len(x))."""
    return (np.log(w)[:, None] - 0.5 * np.log(2 * np.pi * sigmas[:, None] ** 2)
            - (x - mus[:, None]) ** 2 / (2 * sigmas[:, None] ** 2))

def log_mix(x, w, mus, sigmas):
    """Log-density of the mixture sum_k w_k N(mu_k, sigma_k^2), computed stably."""
    c = log_components(x, w, mus, sigmas)
    c_max = c.max(axis=0)
    return c_max + np.log(np.exp(c - c_max).sum(axis=0))

def kl(log_a, log_b, x):
```

```
    """D_KL(A||B) by quadrature on the grid x; in nats."""
    return integrate.trapezoid(np.exp(log_a) * (log_a - log_b), x)
```

37.5.1 The Score and the Fisher Divergence

37.5.1.1 The Score

The score answers one question at every point: in which direction does the density rise, and how steeply? For a distribution P with a differentiable density p, the **score** is the gradient of the log-density *with respect to the data point*:

$$s_P(\mathbf{x}) = \nabla_{\mathbf{x}} \log p(\mathbf{x}). \tag{37.42}$$

(In classical statistics, "score" means the gradient of the log-likelihood with respect to the parameters, as in Section 36.6. Here the gradient is in $\mathbf{x}$.) For a Gaussian $\mathcal{N}(\mu, \sigma^2)$ on the line it is

$$s(x) = -\frac{x - \mu}{\sigma^2},$$

a spring pulling toward the mean with stiffness $1/\sigma^2$. For a mixture $p = \sum_k w_k\, p_k$ the chain rule gives $s(x) = \sum_k r_k(x)\, s_k(x)$ with $r_k(x) = w_k p_k(x)/p(x)$: each component's spring, weighted by the posterior probability, or *responsibility*, that x came from that component. Near a mode one responsibility is close to 1 and the point feels that mode's spring alone, as Figure 37.11 shows. The code is the same formula, with the responsibilities computed from log-densities.

```
def mix_score(x, w, mus, sigmas):
    """Score of a Gaussian mixture: responsibility-weighted component scores."""
    r = np.exp(log_components(x, w, mus, sigmas) - log_mix(x, w, mus, sigmas))
    return np.sum(r * -(x - mus[:, None]) / sigmas[:, None] ** 2, axis=0)
```

The score does not depend on a normalizing constant, and that is why we are interested in it. If $q(\mathbf{x}) = \tilde{q}(\mathbf{x})/Z$, then

$$\nabla_{\mathbf{x}} \log q(\mathbf{x}) = \nabla_{\mathbf{x}} \log \tilde{q}(\mathbf{x}) - \nabla_{\mathbf{x}} \log Z = \nabla_{\mathbf{x}} \log \tilde{q}(\mathbf{x}),$$

since Z does not depend on $\mathbf{x}$. The unnormalized density $\tilde{q}(x) = e^{-x^4/4}$, for instance, has the score $-x^3$, and we never had to find out that $Z \approx 2.56$.

37.5.1.2 The Fisher Divergence

A comparison built from scores can therefore be applied to an unnormalized model. The natural candidate is the mean squared difference of the two vector fields under the data, the **Fisher divergence**:

$$D_{\mathrm{F}}(P\|Q) = \frac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim P}\Big[\big\|s_P(\mathbf{x}) - s_Q(\mathbf{x})\big\|^2\Big]. \tag{37.43}$$

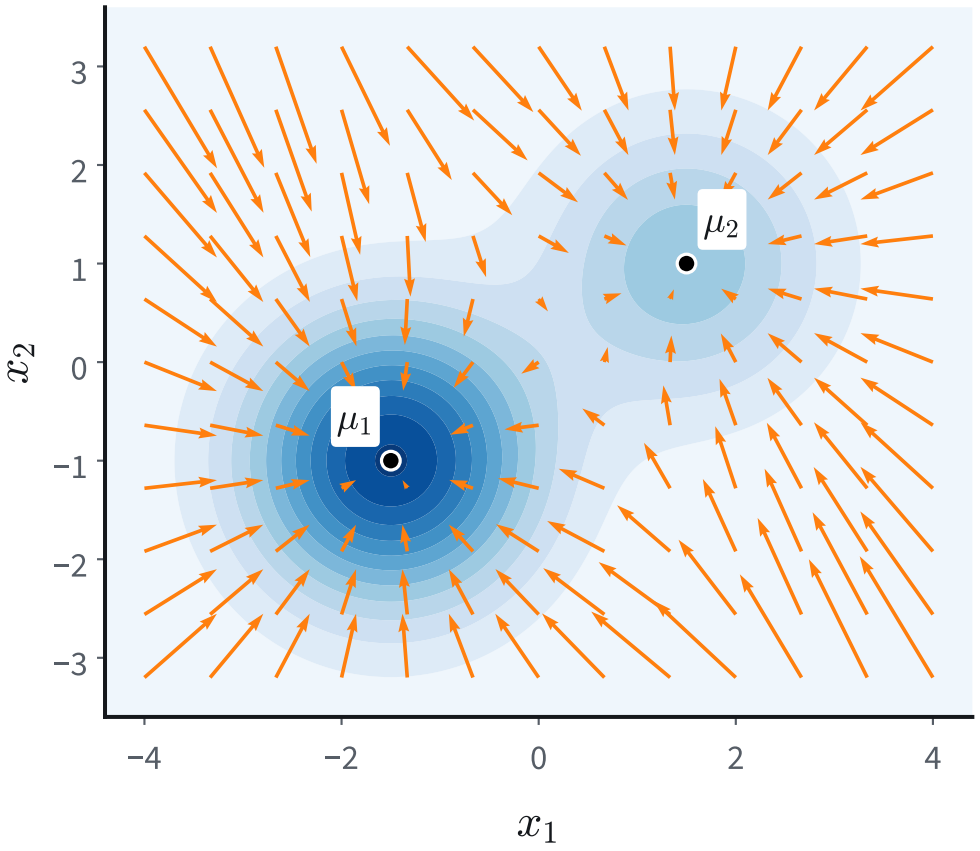


Figure 37.11: The score field of a two-component Gaussian mixture. Arrows show the gradient of the log-density: near each mode the field is a spring pulling toward that mode's mean, and along the valley between the modes the responsibilities hand the point off from one attractor to the other. Density contours are underlaid for reference.

It is non-negative, and it is a divergence, zero only for $P = Q$ (Section 37.3.1.1), when both densities are positive on the same connected region. There, equal scores force $\log p - \log q$ to be constant, and normalization makes the constant zero.

Two Gaussians show how it differs from the KL divergence. With equal variance σ^2 and means a distance Δ apart, the scores differ by the constant Δ/σ^2 at every x, so

$$D_{\mathrm{F}} = \frac{\Delta^2}{2\sigma^4} \qquad \text{whereas} \qquad D_{\mathrm{KL}} = \frac{\Delta^2}{2\sigma^2}$$

by the formula for two Gaussians in Section 37.1. At $\sigma = 1$ the two agree. Measure x in units half as large, and Δ and σ both double: the KL divergence stays where it was, and the Fisher divergence falls to a quarter. It is measured in units of $1/x^2$, as a squared slope must be, and not in nats. Its size therefore depends on the units of the data, and a score-matching loss on one data set cannot be compared with the loss on another, or with a budget in nats. Exercise 1a treats unequal variances, and Exercise 2 a general change of variables.

37.5.1.3 Mixing Weights of Distant Modes

The average in Equation 37.43 runs over the data, and what it averages is a difference of slopes. For data with several modes the two facts combine badly. Take two mixtures with the same components and swapped weights,

$$P_w = w\,\mathcal{N}(-m, 1) + (1 - w)\,\mathcal{N}(m, 1), \qquad w = 0.7 \text{ against } w = 0.3.$$

As distributions the two are far apart. Once the modes are well separated, their KL divergence is that of the weight vectors $(0.7, 0.3)$ and $(0.3, 0.7)$, which is $0.7 \log \frac{7}{3} + 0.3 \log \frac{3}{7} = 0.339$ nats, whatever m is.

The scores hardly differ. Near each mode the responsibility of that mode is close to 1 in both mixtures, so both scores are that mode's spring. Written out (Exercise 3b derives it), the score of P_w is

$$s_w(x) = -x + m \tanh(mx - c_w), \qquad c_w = \tfrac{1}{2} \log \frac{w}{1 - w}.$$

This is the spring $-(x + m)$ far to the left, the spring $-(x - m)$ far to the right, and a hand-over between them around $x = c_w/m$. The weight enters through the place of the hand-over and through nothing else. At $m = 5$ the hand-over sits at $x = 0.085$ for $w = 0.7$ and at $x = -0.085$ for $w = 0.3$, in the valley between the modes, where the density is $e^{-m^2/2}/\sqrt{2\pi}$, about 1.5×10^{-6}. The data almost never visit the one place where the two scores disagree. We compute both divergences by quadrature as the modes move apart, and expect the Fisher divergence to fall roughly like $e^{-m^2/2}$.

```
xs = np.linspace(-14.0, 14.0, 14001)
w_a, w_b, unit = np.array([0.7, 0.3]), np.array([0.3, 0.7]), np.ones(2)
for m in (1.0, 3.0, 5.0):
    mus_m = np.array([-m, m])
    log_a, log_b = log_mix(xs, w_a, mus_m, unit), log_mix(xs, w_b, mus_m, unit)
    score_gap = mix_score(xs, w_a, mus_m, unit) - mix_score(xs, w_b, mus_m, unit)
```

```
    fisher = 0.5 * integrate.trapezoid(np.exp(log_a) * score_gap ** 2, xs)
    print(f'm = {m:.0f}: KL = {kl(log_a, log_b, xs):.3f} nats, '
          f'Fisher = {fisher:.1e}')
```

```
m = 1: KL = 0.184 nats, Fisher = 1.1e-01
m = 3: KL = 0.337 nats, Fisher = 7.1e-03
m = 5: KL = 0.339 nats, Fisher = 4.0e-06
```

The KL divergence settles at 0.339 nats, and the Fisher divergence falls by more than four orders of magnitude, from about 0.1 to about 4×10^{-6}. At $m=5$ it is of the size of $e^{-m^2/2} = e^{-12.5} \approx 4\times10^{-6}$, as the empty valley led us to expect. A loss built from scores cannot see how mass is divided between distant modes. The word "connected" above was a warning: on two separate regions, equal scores allow $\log p - \log q$ a constant of its own on each (Exercise 4). A model fitted to such data by minimizing the Fisher divergence can give both modes the right place and the right shape and still hold 30% of its mass where the data hold 70% (Wenliang and Kanagawa 2020).

Noise fills the valley. Add independent Gaussian noise of standard deviation 2 to both mixtures, and the variance of every mode grows from 1 to 5. The modes, ten standard deviations apart before, are now 4.5 apart, and the density in the valley rises from 1.5×10^{-6} to about 0.015. The data now visit the place where the two scores disagree (Exercise 3c measures what is gained). This is one reason why diffusion models perturb the data with noise at many scales before they match scores. Section 17.1 measures the same blind spot on trained networks.

37.5.2 Stein's Identity

The Fisher divergence has a second defect, which stands in the way of any use at all: it contains s_P, the score of the data, and data come as samples. A finite set of points has no slope. To see where s_P enters, we expand the square. On the line,

$$D_{\mathrm{F}}(P\|Q) = \tfrac{1}{2}\,\mathbb{E}_P[s_P^2] - \mathbb{E}_P[s_P\,s_Q] + \tfrac{1}{2}\,\mathbb{E}_P[s_Q^2]. \tag{37.44}$$

The last term is an average over the data of a function that the model supplies, and a sample estimates it. The first does not depend on the model, so it plays no part when models are fitted or ranked. The obstacle is the cross term in the middle, where the unknown score of the data multiplies the score of the model. An identity that every smooth density satisfies with its own score removes it. We state it for one density on the line and then use it twice: for the data, to remove the cross term, and for the model, to test a sample.

Proposition (Stein's identity). *Let $p>0$ be a differentiable density on $\mathbb{R}$ with score $s=(\log p)'$, and let g be differentiable with $g(x)\,p(x)\to 0$ as $x\to\pm\infty$, $\mathbb{E}_p|g'|<\infty$ and $\mathbb{E}_p|g\,s|<\infty$. Then*

$$\mathbb{E}_{x\sim p}\big[g'(x) + g(x)\,s(x)\big] = 0. \tag{37.45}$$

Proof. The score is $s=p'/p$, so $p'=s\,p$. One integration by parts (Section 34.6.2.3) gives

$$\int g'\,p\,dx = \big[g\,p\big]_{-\infty}^{\infty} - \int g\,p'\,dx = -\int g\,s\,p\,dx,$$

where the boundary term vanishes and both integrals converge by the conditions on g. ■

For the standard normal density the score is $-x$ and the identity reads $\mathbb{E}[g'(X)] = \mathbb{E}[Xg(X)]$. With $g(x) = x$ it says $\mathbb{E}[X^2] = 1$. This case is Stein's lemma for the normal distribution (Stein 1981), and the form for a general density is in (Stein 1986). The identity turns knowledge of a slope into statements about averages, and averages are what samples estimate.

37.5.2.1 Score Matching

Suppose that the data distribution P has a smooth density, even though we see only samples of it. Then the identity holds for the data, with $s = s_P$, and we may take the model's score as the test function, $g = s_Q$. It reads $\mathbb{E}_P[s'_Q + s_Q\, s_P] = 0$. The cross term of Equation 37.44 is therefore $-\mathbb{E}_P[s'_Q]$, an average that no longer contains s_P, and

$$D_{\mathrm{F}}(P\|Q) = \mathbb{E}_{x\sim P}\Big[\tfrac{1}{2}\, s_Q(x)^2 + s'_Q(x)\Big] + \tfrac{1}{2}\,\mathbb{E}_P[s_P^2]. \tag{37.46}$$

The derivative has moved from the data, which have none, to the model. The last term is the same for every model, and the rest is an average over the data of quantities that the model supplies: it can be estimated from a sample and minimized, with no Z and no s_P. This is *score matching*. Hyvärinen (2005) devised it for models without a normalizing constant and fitted a model of natural image patches with it.

The quartic of Section 37.5.1 shows what has been gained. Give it a parameter, $\tilde{q}_\theta(x) = e^{-\theta x^4/4}$, so that $s_Q(x) = -\theta x^3$ and $s'_Q(x) = -3\theta x^2$. With a bar for an average over the data, the part of Equation 37.46 that depends on the model is estimated by

$$\overline{\tfrac{1}{2}\, s_Q^2 + s'_Q} = \tfrac{1}{2}\,\theta^2\,\overline{x^6} - 3\,\theta\,\overline{x^2}, \qquad \text{which is smallest at} \qquad \hat{\theta} = \frac{3\,\overline{x^2}}{\overline{x^6}}.$$

Two moments of the sample fit the model, and nobody had to integrate $e^{-\theta x^4/4}$. Maximum likelihood needs $Z(\theta)$ and arrives at a different function of the data (Exercise 6). For a Gaussian model the two fits coincide (Exercise 5).

On the line, s'_Q is one more derivative. In $\mathbb{R}^n$ it becomes a sum of n second derivatives, one for each coordinate, and the single backward pass that gave the score turns into n of them, far too many for an image. Section 38.5.1 carries the argument to $\mathbb{R}^n$, shows how noise added to the data avoids these derivatives, and derives the training loss of diffusion models from the result.

37.5.2.2 A Test for Samples

The identity can also test a sample against a model. Stated for the model, it says that for every g that meets the conditions of the proposition the function $g' + g\, s_Q$ averages to zero under Q. The map that takes g to $g' + g\, s_Q$ is called the *Stein operator* of the model. If the average of $g' + g\, s_Q$ over a sample is far from zero for some g, the sample did not come from Q. The check needs the model's score at the sample points and nothing else, neither Z nor samples of the model.

How much can one test function detect? Let the model be $Q = \mathcal{N}(0, 1)$, so that the function to average is $g'(x) - x\, g(x)$. For $g = 1$ this is $-x$, and the test checks the mean of the sample. For $g(x) = x$ it is $1 - x^2$, and the test checks the second moment against 1. As a third test function we take $g(x) = \sin x$, which has no such simple reading. The cell draws $200{,}000$ points from the model itself, from $\mathcal{N}(0.5, 1)$, which is shifted, and from $\mathcal{N}(0, 4)$, which is twice as wide, and prints every average with its standard error. From the two moments we expect -0.5 and $1 - 1.25 = -0.25$ on the shifted sample, and 0 and $1 - 4 = -3$ on the wide one.

```
rng = np.random.default_rng()
n = 200_000
# Stein operator of the model Q = N(0, 1), whose score is -x:  g'(x) - x g(x)
tests = [('g = 1', np.ones_like, np.zeros_like),
         ('g = x', lambda t: t, np.ones_like),
         ('g = sin x', np.sin, np.cos)]
samples = [('N(0, 1)', rng.normal(0.0, 1.0, n)),
           ('N(0.5, 1)', rng.normal(0.5, 1.0, n)),
           ('N(0, 4)', rng.normal(0.0, 2.0, n))]
print(f'{"sample":10s}' + ''.join(f'{name:>18s}' for name, _, _ in tests))
for s_name, x in samples:
    vals = [g_prime(x) - x * g(x) for _, g, g_prime in tests]
    print(f'{s_name:10s}' + ''.join(f'{v.mean():+10.3f} ± {v.std() / np.sqrt(n):.3f}'
                                    for v in vals))
```

```
sample                 g = 1              g = x          g = sin x
N(0, 1)       -0.001 ± 0.002    -0.002 ± 0.003    -0.001 ± 0.002
N(0.5, 1)     -0.503 ± 0.002    -0.249 ± 0.004    -0.145 ± 0.002
N(0, 4)       +0.002 ± 0.004    -3.006 ± 0.013    -0.410 ± 0.003
```

On the model's own sample no average is further from zero than a few of its standard errors. All three test functions catch the shifted sample, at about -0.50, -0.25 and -0.15. The wide sample is another matter. Its mean is zero, so $g = 1$ reports no more than a hundredth or so, nothing like the -0.5 of the shift, although a sample with four times the model's variance is hardly a subtle violation. Only $g = x$, at about -3, and $\sin x$, at about -0.4, notice it. A test function checks one property of the sample, and a sample can be wrong in a property that the function does not check.

Why does $g = 1$ see the shift and miss the width? A sample from a distribution P with a smooth density obeys the identity of P, $\mathbb{E}_P[g'] = -\mathbb{E}_P[g\, s_P]$, and therefore

$$\mathbb{E}_P[g' + g\, s_Q] = \mathbb{E}_P[(s_Q - s_P)\, g]. \tag{37.47}$$

The average is the difference of the two scores, weighted by g. For the shifted sample the difference is the constant $-x + (x - 0.5) = -0.5$, and $g = 1$ picks it up in full. For the wide sample it is $-x + x/4 = -\frac{3}{4}x$. This is an odd function, and a distribution that is symmetric about zero averages it to zero when the weight is $g = 1$. No single test function is correlated with every possible difference of scores.

37.5.3 The Kernel Stein Discrepancy

One test function checks one property, so we use many at once: all functions in the unit ball of a feature space, and among them the one whose average is largest. Section 37.4.1

did the same for the difference of two means and found a closed form, the maximum mean discrepancy (MMD). There is one here as well, which is rather remarkable. A supremum over infinitely many test functions reduces to kernel evaluations on pairs of data points.

Write $\mathcal{A}_Q$ for the Stein operator of the model Q, $(\mathcal{A}_Q g)(x) = g'(x) + s_Q(x)\, g(x)$, so that Stein's identity says $\mathbb{E}_Q[\mathcal{A}_Q g] = 0$, and measure the data distribution P by $\sup_g \mathbb{E}_P[\mathcal{A}_Q g]$, the largest amount by which the data violate the identity of the model.

For $g(x) = \langle \mathbf{w}, \varphi(x) \rangle$ with a feature vector φ and $\|\mathbf{w}\| \le 1$, the operator acts on the features: $(\mathcal{A}_Q g)(x) = \langle \mathbf{w}, \psi(x) \rangle$ with $\psi = \varphi' + s_Q\, \varphi$. By the Cauchy–Schwarz inequality the supremum of $\langle \mathbf{w}, \mathbb{E}_P[\psi] \rangle$ over the ball is $\big\|\mathbb{E}_P[\psi]\big\|$, and the square of this norm is $\mathbb{E}\langle \psi(x), \psi(x') \rangle$ for independent draws x, x' from P. Multiplying out leaves four inner products of features, and each is the kernel $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ or one of its derivatives:

$$\langle \varphi'(x), \varphi(x') \rangle = \partial_x k, \qquad \langle \varphi(x), \varphi'(x') \rangle = \partial_{x'} k, \qquad \langle \varphi'(x), \varphi'(x') \rangle = \partial_x \partial_{x'} k.$$

Their sum, with the scores as coefficients, is the **Stein kernel** of Q,

$$k_Q(x, x') = s_Q(x)\, s_Q(x')\, k(x, x') + s_Q(x)\, \partial_{x'} k(x, x') + \partial_x k(x, x')\, s_Q(x') + \partial_x \partial_{x'} k(x, x'), \tag{37.48}$$

and the squared **kernel Stein discrepancy** (KSD) is its average over two independent draws from the data:

$$\mathrm{KSD}^2(P, Q) = \mathbb{E}_{x, x' \sim P}\big[k_Q(x, x')\big]. \tag{37.49}$$

The average is taken under the data alone. A sample $x_1, \dots, x_N$ from P therefore gives the unbiased estimator

$$\widehat{\mathrm{KSD}}^2 = \frac{1}{N(N-1)} \sum_{i \neq j} k_Q(x_i, x_j),$$

the same average over pairs without self-pairs as the MMD estimator. Statisticians call such an average a U-statistic, which is where the name `ksd2_ustat` in the code below comes from.

To see what the KSD measures, we apply Equation 37.47 to each feature. It gives $\mathbb{E}_P[\psi] = \mathbb{E}_P[(s_Q - s_P)\, \varphi]$, and squaring the norm as before,

$$\mathrm{KSD}^2(P, Q) = \mathbb{E}_{x, x' \sim P}\big[(s_Q - s_P)(x)\, k(x, x')\, (s_Q - s_P)(x')\big]. \tag{37.50}$$

The KSD is a kernel-smoothed relative of the Fisher divergence $D_{\mathrm{F}}(P \| Q)$: the same difference of scores under the same data distribution, multiplied across pairs of points with weight k where the Fisher divergence squares it at each point. The blind spot comes along, and it is deeper. On the two mixtures of Section 37.5.1 both points of a pair must now fall into the valley where the scores differ, and KSD^2 decays roughly like e^{-m^2}, the square of the rate of the Fisher divergence (Exercise 3e).

Where both scores are known, Equation 37.50 predicts the value. Let the data be $P = \mathcal{N}(0,1)$ and take three models. Against the correct model $\mathcal{N}(0,1)$ the difference of scores vanishes and $\mathrm{KSD}^2 = 0$. Against the shifted model $\mathcal{N}(1,1)$ the difference is $-(x-1) - (-x) = 1$ at every point, so $\mathrm{KSD}^2 = \mathbb{E}[k(X,X')]$. For the RBF kernel $k(x,x') = e^{-(x-x')^2/(2\ell^2)}$ with length scale ℓ this is the Gaussian integral of Section 37.4.1: the difference $X - X'$ has the distribution $\mathcal{N}(0,2)$, and the average of the kernel over it is $(1 + 2/\ell^2)^{-1/2}$. At $\ell = 1$ we obtain $\mathrm{KSD}^2 = 1/\sqrt{3} \approx 0.577$.

The third model is of the kind that this section is about: the quartic $\tilde{q}(x) = e^{-x^4/4}$ of Section 37.5.1, which we know only up to its constant. Its score is $-x^3$, the difference of scores is $x - x^3$, and the double integral in Equation 37.50 has no such short form. The function `ksd2_population` in the cell evaluates it on a grid. It can do so because we, unlike the test, know the density of the data.

For the RBF kernel on the line the derivatives in Equation 37.48 are $\partial_{x'} k = \frac{x-x'}{\ell^2}\, k$, $\partial_x k = -\frac{x-x'}{\ell^2}\, k$ and $\partial_x \partial_{x'} k = \left(\frac{1}{\ell^2} - \frac{(x-x')^2}{\ell^4}\right) k$. In the code, `dk_dy` is the derivative in the second argument and `kappa` the matrix of all $k_Q(x_i, x_j)$. We test one sample of $2{,}000$ points from $\mathcal{N}(0,1)$ against the three models. The function `ksd2_ustat` receives each model as a score function and as nothing else.

```
def ksd2_ustat(x, score, ell=1.0):
    """U-statistic KSD^2 with the RBF kernel exp(-(a-b)^2 / (2 ell^2))."""
    d = x[:, None] - x[None, :]
    k = np.exp(-d ** 2 / (2 * ell ** 2))
    dk_dx = -d / ell ** 2 * k
    dk_dy = d / ell ** 2 * k
    d2k = (1 / ell ** 2 - d ** 2 / ell ** 4) * k
    s = score(x)
    kappa = (s[:, None] * s[None, :] * k + s[:, None] * dk_dy
             + dk_dx * s[None, :] + d2k)
    n = len(x)
    return (kappa.sum() - np.trace(kappa)) / (n * (n - 1))

def ksd2_population(score, ell=1.0):
    """KSD^2 of the data P = N(0, 1) against a model with this score, from
    the score difference s_Q - s_P weighted by the kernel, by quadrature."""
    t = np.linspace(-10.0, 10.0, 2001)
    h = np.exp(-t ** 2 / 2) / np.sqrt(2 * np.pi) * (score(t) + t)
    k = np.exp(-(t[:, None] - t[None, :]) ** 2 / (2 * ell ** 2))
    return h @ k @ h * (t[1] - t[0]) ** 2

x_p = rng.standard_normal(2000)          # the data sample, P = N(0, 1)
for name, s in [('true model  N(0,1)', lambda t: -t),
                ('wrong model N(1,1)', lambda t: -(t - 1)),
                ('quartic exp(-x^4/4)', lambda t: -t ** 3)]:
    print(f'KSD^2 vs the {name:19s}: {ksd2_ustat(x_p, s):+.4f}  '
          f'(population {ksd2_population(s):.4f})')
```

```
KSD^2 vs the true model  N(0,1) : +0.0010  (population 0.0000)
KSD^2 vs the wrong model N(1,1) : +0.5274  (population 0.5774)
KSD^2 vs the quartic exp(-x^4/4): +0.2835  (population 0.3208)
```

Against the correct model the estimate is within a few thousandths of zero and may be slightly negative, as an unbiased estimate of zero must sometimes be. Against the shifted model, in the second line, it lands close to the predicted $1/\sqrt{3} \approx 0.58$, within about 0.1. Against the quartic it is a few tenths, far from zero. The population value is 0.32 here, and a single sample can miss it by 0.1 and more, because the score $-x^3$ is large at the few points far from the origin and these dominate the average.

Between the three lines of output only the score function changed. The sample is the same, no model was sampled, and the normalizing constant of the quartic, $Z \approx 2.56$, appears nowhere in the code.

With the RBF kernel a value of zero can be trusted. For data with a smooth density, and under mild conditions on the two scores, $\mathrm{KSD}^2(P, Q) = 0$ if and only if $P = Q$ (Q. Liu et al. 2016; Chwialkowski et al. 2016). The two mixtures are a reminder that the value can nevertheless be tiny for distributions that are far apart.

In $\mathbb{R}^n$ only the notation changes: g is a vector field, the operator is $s_Q^\top g + \nabla \cdot g$, the products in Equation 37.48 become inner products of vectors, and the last term becomes $\sum_i \partial_{x_i} \partial_{x_i'} k$. The KSD serves as a goodness-of-fit test for unnormalized models and as a check on the output of a sampler. Applied to the kernel itself, the Stein operator also gives a direction in which to move the points of a sample toward the model, a method called Stein variational gradient descent (Liu and Wang 2016). No training objective in this book uses the KSD.

37.5.4 Training Objectives and Their Divergences

The two sections before this one produced three families of comparisons, and this one has added two comparisons of scores. Figure 37.12 places the families and the members they share. It does not draw the Fisher divergence, which belongs to none of the three. Nor does it draw the KSD, which has the form of the second family, a largest difference of averages over a class of test functions, but with a class that is built from the model's score and changes with the model.

Table 37.4 lists the objectives that are built on these comparisons. Each row names an objective, what it needs from the model and the data, and the divergence that it minimizes when its averages are taken over the whole distributions and, for the adversarial objectives, the critic is the best possible. A critic, as in Section 37.3.4, is a function trained on the two samples, in practice a second network. The fourth column returns to the test case with which Section 37.3 began, the data at a single point, $P = \delta_0$, and the model's samples at a point a distance d away, $Q = \delta_d$. A comparison that does not change with d gives such a model no gradient toward the data.

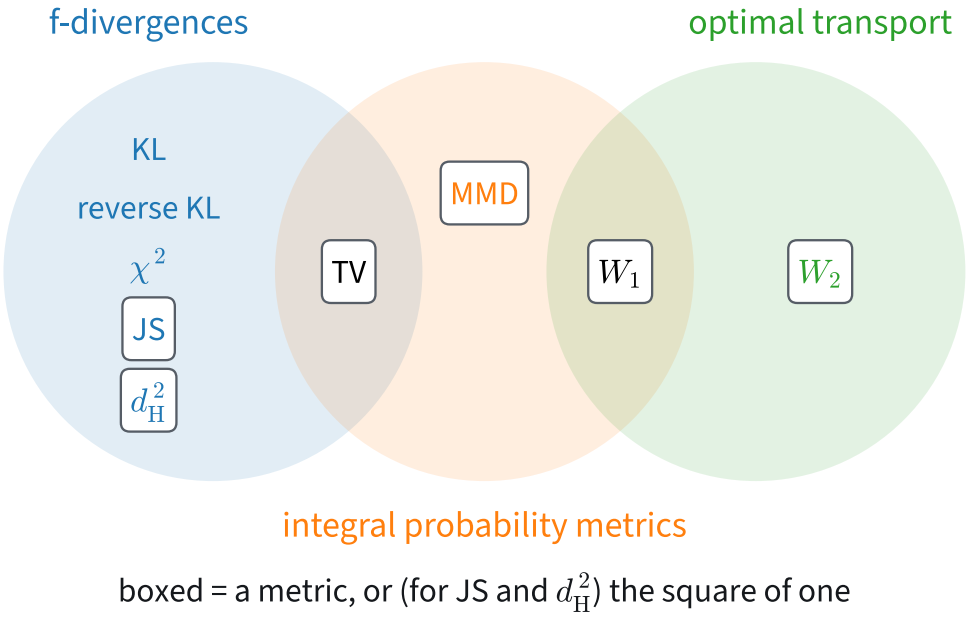


Figure 37.12: The ratio, test-function and transport families and their common members. f-divergences average a convex function of the density ratio, integral probability metrics take a supremum of mean differences over a class of test functions, and optimal-transport distances measure the cost of moving mass. Total variation belongs to the first two families and Wasserstein-1 to the last two. (Total variation is also a transport cost, the one that charges 1 for moving mass by any distance at all, which the figure does not draw.) Boxed names are metrics; JS and the squared Hellinger divergence d_{H}^2 are squares of metrics.

Table 37.4: Training objectives, ordered by what they need, and the divergences they minimize when averages are exact and the critic is optimal. The column on point masses is the value for δ_0 against δ_d with $d \neq 0$; $f(0)$ and $f'(\infty)$ are the limits of $f(u)$ at 0 and of $f(u)/u$ at infinity, and the MMD entry uses the RBF kernel with length scale ℓ.

What the objective needs	Objective	Divergence	Two point masses d apart	Treated in
model density at data points	maximum likelihood (autoregressive models, flows)	forward KL	∞	Section 37.3.2
samples of both, through a critic	original GAN	Jensen–Shannon	$\log 2$, constant	Section 37.3.4
samples of both, through a critic	f-GAN	the chosen D_f	$f(0) + f'(\infty)$, constant	Section 37.3.4
samples of both, no critic for a fixed kernel	moment matching, two-sample tests	MMD	$\text{MMD}^2 = 2 - 2e^{-d^2/(2\ell^2)}$, flat beyond a few ℓ	Section 37.4.1
samples of both, a 1-Lipschitz critic	WGAN	W_1	d	Section 37.4.2
model samples, the model's density at them, the unnormalized target	variational inference, VAE posterior	reverse KL	∞	Section 37.3.2
model score, data samples	score matching, diffusion models	Fisher	undefined (a point mass has no score)	Section 37.5.1
model score, data samples	goodness-of-fit tests, sampler quality	KSD	undefined (a point mass has no score)	Section 37.5.3

Read from the top, the table is advice, because its rows are ordered by what a model can supply. A model whose density can be evaluated, and is positive wherever data may fall, should be trained by maximum likelihood: forward KL needs nothing but the data, its sample average is unbiased, and it leaves no mode of the data uncovered. A model that only produces samples has three choices. A critic trained as in a GAN is the usual one for images (Section 16.1), where nobody knows in advance which features tell the model's samples from the data, and the critic learns them. Its estimate of the divergence is biased low and goes flat when the samples are far from the data (Section 37.3.4). A 1-Lipschitz critic is the choice for a model that starts far away, because W_1 keeps its slope at every distance (Section 16.2 trains with both kinds). A fixed kernel needs no second network, and the error of its estimate is of order $N^{-1/2}$ in any dimension (Section 37.4.1). It suits two-sample tests and the

evaluation of samples, and it loses its gradient beyond a few length scales.

The last three rows are for the cases in which a normalizing constant is missing. When it is the target that is known only up to its constant, and the model can be sampled and has a density, as in variational inference, reverse KL applies, at the risk of settling on one mode. And a model known only through its score calls for the Fisher divergence, which leaves the weights of distant modes undetermined unless noise is added, as diffusion models do (Section 17.1).

The table says which comparison an objective makes. It does not say how well that comparison can be made from data. Every sample-based number of these three sections, the critic bound, MMD^2 and KSD^2, is an estimate with an error, and the critic bound is biased low besides. The next two sections turn from two distributions to two variables. Section 37.6 uses the KL divergence to measure how much one variable tells about another, and Section 37.7 asks how much of it a sample of n pairs can certify.

37.5.5 Summary

A model that is known only up to its normalizing constant can still be compared with data through its score, the gradient of the log-density with respect to the input, from which the constant drops out. The Fisher divergence is the mean squared difference of two scores under the data, and Stein's identity, one integration by parts, removes the unknown score of the data from it. Scores are local: between two mixtures that differ only in the weights of two modes ten standard deviations apart, the Fisher divergence is 4×10^{-6} while the KL divergence is 0.339 nats. The kernel Stein discrepancy uses the same identity to test a sample against a model through the model's score alone. Which comparison to train on is decided by what the model can supply: a density, samples, or a score.

37.5.6 Exercises

1. **Fisher divergence and Stein moments.** The score of $\mathcal{N}(\mu, \sigma^2)$ is $-(x-\mu)/\sigma^2$.
   1. Compute the Fisher divergence Equation 37.43 between $\mathcal{N}(\mu_1, \sigma_1^2)$ and $\mathcal{N}(\mu_2, \sigma_2^2)$ in closed form, and check that it reduces to $(\mu_1 - \mu_2)^2/(2\sigma^4)$ for equal variances. Is the Fisher divergence symmetric?
   2. Apply Stein's identity Equation 37.45 for $\mathcal{N}(0,1)$ to $g(x) = x^k$, derive the recursion $\mathbb{E}[X^{k+1}] = k\,\mathbb{E}[X^{k-1}]$, and compute $\mathbb{E}[X^4]$ and $\mathbb{E}[X^6]$. What does $g(x) = 1$ give for a general density that satisfies the conditions of the identity?
   3. Let the data distribution P on $\mathbb{R}$ have finite moments of all orders, and let $\mathbb{E}_P[g'(X) - Xg(X)] = 0$ for every polynomial g. Explain why $P = \mathcal{N}(0,1)$, using that the normal distribution is determined by its moments. Then let $P(X=1) = P(X=-1) = \frac{1}{2}$, and show that it satisfies the identity for every polynomial g of degree at most 2 but not for $g(x) = x^3$. What does this imply for a goodness-of-fit test that checks the identity only on polynomials up to a fixed degree?
2. **Units and changes of variables.** Let $a > 0$, and write P_a and Q_a for the distributions of aX when X is drawn from P and from Q.
   1. Show that $D_{\mathrm{KL}}(P_a \| Q_a) = D_{\mathrm{KL}}(P \| Q)$ and $D_{\mathrm{F}}(P_a \| Q_a) = D_{\mathrm{F}}(P \| Q)/a^2$. A set of images is stored once with pixel values between 0 and 255 and once with values between 0 and 1. By what factor does the Fisher divergence between the data and a model differ in the two units?

2. Replace the scaling by a smooth increasing map $y = \phi(x)$ of the line. Show that the two scores at y differ by $(s_P(x) - s_Q(x))/\phi'(x)$, although each of them gains a term in ϕ''. Conclude that the Fisher divergence becomes $\frac{1}{2}\,\mathbb{E}_P\big[(s_P - s_Q)^2/\phi'^2\big]$, while the KL divergence does not change.
3. A change of variables can therefore make a Fisher divergence that is not zero as large or as small as we please. For the two mixtures of the text, where on the line would ϕ have to be nearly flat to make the difference of the weights visible?

3. code **Score blindness to mixing weights.** ⊠ Let $P_w = w\,\mathcal{N}(-m, 1) + (1 - w)\,\mathcal{N}(m, 1)$.
   1. The text computes $D_{\mathrm{F}}(P_{0.7}\|P_{0.3})$ and $D_{\mathrm{KL}}(P_{0.7}\|P_{0.3})$ for $m \in \{1, 3, 5\}$. Extend the computation to $m \in \{1, \dots, 6\}$. Show that the KL divergence tends to $0.4 \log \frac{7}{3}$, the value 0.339 of the text, and find the rate at which the Fisher divergence decays by fitting $\log D_{\mathrm{F}}$ against m^2.
   2. Derive the score $s_w(x) = -x + m \tanh(mx - c_w)$ of the text from the responsibilities. Show that $s_{0.7}$ and $s_{0.3}$ differ most at $x = 0$, by $0.8\,m$, and substitute $u = mx$ in the Fisher divergence to show that it behaves like $C\,m\,e^{-m^2/2}$ for large m, with a constant C that depends only on the weights. Compare with the rate you fitted in (a). What does the result imply for the mixing weights of a model fitted by score matching (Wenliang and Kanagawa 2020)?
   3. Add independent $\mathcal{N}(0, \sigma^2)$ noise to samples of both mixtures, so that each component becomes $\mathcal{N}(\pm m, 1+\sigma^2)$. Repeat (a) at $m = 5$ for $\sigma \in \{0, 0.5, 1, 2\}$. How does the noise change the two divergences? Heavy noise shows the weights and blurs the shape of each mode, and light noise does the opposite. Why would a model that matches scores at several noise levels recover both (Song and Ermon 2019)?
   4. Write P_t and Q_t for two distributions after independent $\mathcal{N}(0, t)$ noise has been added to both. For two Gaussians of equal variance σ^2, which the noise raises to $\sigma^2 + t$, check from the two closed forms of the text that $\frac{d}{dt} D_{\mathrm{KL}}(P_t\|Q_t) = -D_{\mathrm{F}}(P_t\|Q_t)$. The identity holds for every pair of smooth densities. Confirm it by a finite difference for the two mixtures at $m = 5$ and $t \in \{1, 4, 9\}$, and conclude that $D_{\mathrm{KL}}(P\|Q) = \int_0^\infty D_{\mathrm{F}}(P_t\|Q_t)\,dt$. At which noise level is the Fisher divergence largest? What does the integral say about matching scores at one noise level against matching them at all levels (Y. Song, Durkan, et al. 2021)?
   5. The kernel Stein discrepancy has the same blind spot. Evaluate Equation 37.50 for the two mixtures by quadrature on a grid, with the RBF kernel at $\ell = 1$ and $m \in \{1, 3, 5\}$, and compare with the Fisher divergences of the text (at $m = 5$ you should find about 10^{-11}). The function `ksd2_population` of the text is the pattern; it is written for the data $\mathcal{N}(0, 1)$, and here the data are $P_{0.7}$. Where must both points of a pair fall to contribute, and how wide is that region compared with ℓ? Conclude that KSD^2 decays like e^{-m^2}. Would a smaller length scale help?

4. **Two separate regions.** Let $p_0(x) = 30\,x^2(1 - x)^2$ for $0 < x < 1$ and $p_0(x) = 0$ elsewhere, and let P_w have the density $w\,p_0(x) + (1 - w)\,p_0(x - 2)$: two humps of the same shape with a gap between them.
   1. Show that the score of P_w does not depend on w at any point where the density is positive. Conclude that $D_{\mathrm{F}}(P_{0.7}\|P_{0.3}) = 0$, and compute $D_{\mathrm{KL}}(P_{0.7}\|P_{0.3})$.
   2. Which step fails in the argument of the text that the Fisher divergence is zero only for $P = Q$? How is this example related to the two Gaussian mixtures of the text as m grows?

3. Let P_w be the data, and fit the weight v of a model P_v by score matching. What does the objective Equation 37.46 say about v?
4. Evaluate Equation 37.50 for $P_{0.7}$ and $P_{0.3}$. The text states that $\mathrm{KSD}^2 = 0$ only for $P = Q$, under conditions that it does not spell out. Name one condition that the statement must contain.

5. **Score matching for a Gaussian model.** Let the data distribution P have mean a and variance v, take the model $Q = \mathcal{N}(\mu, \sigma^2)$, and write $J = \mathbb{E}_P[\frac{1}{2}s_Q^2 + s_Q']$ for the part of Equation 37.46 that depends on the model.
   1. Show that $J = (v + (a-\mu)^2)/(2\sigma^4) - 1/\sigma^2$ and that it is smallest at $\mu = a$ and $\sigma^2 = v$, whatever else is true of P. Compare with the Gaussian that minimizes the forward KL divergence (Section 37.3.2).
   2. The minimum is $J = -1/(2v)$, so the Fisher divergence from P to its best Gaussian fit is $\frac{1}{2}(\mathbb{E}_P[s_P^2] - 1/v)$. Apply Stein's identity for P to $g(x) = x - a$, and then the Cauchy–Schwarz inequality, to show that $\mathbb{E}_P[s_P^2] \geq 1/v$, with equality only if P is Gaussian. Which density of a given variance has the smallest mean squared score?
   3. Write the model as $\tilde{q}_\theta(x) = \exp(\theta_1 x + \theta_2 x^2)$. Show that J is a quadratic function of θ, so that score matching solves two linear equations in the moments of the data. Show the same for every model of the form $\tilde{q}_\theta(x) = \exp(\theta^\top \phi(x))$. What would maximum likelihood have to compute instead?

6. code **An unnormalized quartic.** Let $\tilde{q}_\theta(x) = e^{-\theta x^4/4}$ with $\theta > 0$.
   1. Compute the score. Use Stein's identity with $g(x) = x$ and with $g(x) = x^3$ to show that under the model $\mathbb{E}[X^4] = 1/\theta$ and $\mathbb{E}[X^6] = (3/\theta)\,\mathbb{E}[X^2]$, without computing the normalizing constant. Can the identity with polynomial g also give $\mathbb{E}[X^2]$?
   2. The text finds the score-matching estimate $\hat{\theta} = 3\,\overline{x^2}/\overline{x^6}$, where a bar denotes an average over the data. Show that the normalizing constant is $Z(\theta) = \theta^{-1/4} Z(1)$ and that maximum likelihood gives $\hat{\theta} = 1/\overline{x^4}$. Use (a) to argue that both estimates approach the true θ on a large sample from the model, although they are different functions of the data.
   3. If U has the Gamma distribution with shape $\frac{1}{4}$ and scale 1, then $(4U/\theta)^{1/4}$ with a random sign has the density $\tilde{q}_\theta/Z(\theta)$. Draw $1{,}000$ points at $\theta = 1$, compute both estimates, and repeat $2{,}000$ times. Which estimate has the smaller standard deviation, and by what factor? What does score matching give up for not needing Z?
   4. Now let the data be standard normal, as in the experiment on the KSD, so that no quartic is right. On a large sample the two estimates of (b) approach $3\,\mathbb{E}[X^2]/\mathbb{E}[X^6] = \frac{1}{5}$ and $1/\mathbb{E}[X^4] = \frac{1}{3}$. Evaluate `ksd2_population` at the scores of both fits, and find the θ at which it is smallest. Three criteria have chosen three different quartics. Why is this no contradiction, and when would the three agree?

7. code **A polynomial Stein kernel.** Let the model be $Q = \mathcal{N}(0,1)$ and take the kernel $k(x, x') = \sum_{j=0}^{M} x^j x'^j$, whose features are the monomials $1, x, \ldots, x^M$.
   1. Show that $\mathrm{KSD}^2(P, Q) = \sum_{j=0}^{M} \left(\mathbb{E}_P[jX^{j-1} - X^{j+1}]\right)^2$, a sum of squared averages of the kind that the experiment on Stein's identity prints, one for each monomial. Evaluate it for $M = 1$ on $P = \mathcal{N}(0.5, 1)$ and on $P = \mathcal{N}(0, 4)$, and compare with the first two columns of that experiment's output.
   2. Let $P(X = 1) = P(X = -1) = \frac{1}{2}$, as in Exercise 1c. Show that $\mathrm{KSD}^2(P, Q) =$

0 for $M \le 2$ and that it is 4 for $M = 3$. Why can a kernel with finitely many features not have the property that $\mathrm{KSD}^2 = 0$ only for $P = Q$?

3. The Stein kernel Equation 37.48 needs no density of P. For the RBF kernel with $\ell = 1$ and the two-point distribution of (b), show that $k_Q(x, x) = 2$ and $k_Q(1, -1) = -8e^{-2}$, so that $\mathrm{KSD}^2(P, Q) = 1 - 4e^{-2} \approx 0.46$. Check the value with `ksd2_ustat` on $2{,}000$ draws. Which features of the RBF kernel (Section 37.4.1) detect what the monomials up to degree 2 miss?

8. code **The length scale of the Stein kernel.** Let the data be $P = \mathcal{N}(0, 1)$, as in the experiment of the text, and use the RBF kernel with length scale ℓ.
   1. For the model $\mathcal{N}(\mu, 1)$, show from Equation 37.50 that $\mathrm{KSD}^2 = \mu^2\,(1{+}2/\ell^2)^{-1/2}$. What is its limit as $\ell \to \infty$? Which single test function g gives the same number as the square of its average $\mathbb{E}_P[g' + g\,s_Q]$?
   2. For the model $\mathcal{N}(0, 4)$, whose score is $-x/4$, show that $\mathrm{KSD}^2 = \frac{9}{16}\,\ell\,(\ell^2{+}2)^{-3/2}$. (Write XX' in terms of the independent variables $X + X'$ and $X - X'$.) Show that the value is largest at $\ell = 1$, the standard deviation of the data, and that it tends to zero as $\ell \to \infty$. Explain the limit with the answer to (a).
   3. Check both formulas with `ksd2_population` at $\ell \in \{0.25, 1, 4, 16\}$, and compute the same four values for the quartic, whose score is $-x^3$. Why can no length scale be the best one against every wrong model?

37.6 Mutual Information

The previous sections compared two distributions over the same variable, such as the data and a model. Many questions in deep learning concern a pair of variables instead. An image and its caption share content; so do two crops of one photograph, and a sentence and its translation. How much does one of them tell us about the other?

The correlation coefficient, the usual measure of dependence, answers only for straight-line relations. Take $X \sim \mathcal{N}(0, 1)$ and let Y be its square plus a little noise, $Y = X^2 + 0.3\,\epsilon$ with independent $\epsilon \sim \mathcal{N}(0, 1)$. Their correlation is zero, because $\mathrm{Cov}(X, Y) = \mathbb{E}[X^3] = 0$ by symmetry. Yet knowing X pins Y down up to noise of standard deviation 0.3, while Y on its own spreads from about 0 to 9. The quantity this section defines sees the dependence: it credits this pair with more shared information than a Gaussian pair with correlation 0.9.

We want a number that is zero for independent variables and positive for all others, that grows with their dependence, and that does not change when either variable is measured in other coordinates. We already have what we need to build it: the KL divergence $D_{\mathrm{KL}}(P\|Q)$ of Section 37.1, which by Gibbs' inequality is never negative and is zero only when $P = Q$. Take for P the joint distribution of the pair and for Q the distribution the pair would have if its members were independent. The result is the *mutual information* $I(X; Y)$.

The definition is the easy part. The section is for three results that follow from it. Processing cannot increase mutual information, so no layer of a network knows more about the input than the layer below it. A classifier cannot reach a given error rate unless its input carries a minimum of information about the label. And the test loss of any model yields a lower bound on that information. We then ask Shannon's two questions, how much a noisy line can carry and how little a compressed copy may keep, and a third that grows out of the second: what should a representation keep? Throughout, the distributions are known. Estimating mutual information from samples alone is a harder problem, which Section 37.7 takes up.

We work in nats, so 1 bit $= \ln 2 \approx 0.693$ nats. All logarithms are natural; we write log in formulas and ln where a number is evaluated. The binary entropy is $H_b(p) = -p \log p - (1-p)\log(1-p)$.

37.6.1 Measures of Dependence

Every definition of this section is computed first on one table of four numbers, and the same table returns in each later part.

37.6.1.1 Joint and Conditional Entropy

Let X be a fair bit, and let Y be a copy of it that is flipped with probability $\delta = 0.1$. Read X as a bit sent over a noisy line and Y as the bit received, or read Y as a binary label and X as a feature that agrees with it 90% of the time. The table is symmetric, so nothing below depends on the reading. With rows indexing $x \in \{0,1\}$ and columns $y \in \{0,1\}$, the joint distribution is

$$p_{X,Y} = \begin{pmatrix} 0.45 & 0.05 \\ 0.05 & 0.45 \end{pmatrix},$$

and summing rows and columns gives the marginals $p_X = p_Y = (\tfrac{1}{2}, \tfrac{1}{2})$. We refer to this pair as the *flipped bit*. Subscripts name the variables of a distribution, and we drop them where the arguments already show which distribution is meant, as in the conditional $p(y \mid x)$.

A pair has two entropies beyond those of its members, both defined in Section 37.1.1.3. The *joint entropy* is the entropy of the pair viewed as a single variable, here one with four outcomes:

$$H(X,Y) = -\mathbb{E}_{(x,y)\sim P_{X,Y}}[\log p_{X,Y}(x,y)]. \tag{37.51}$$

On the table, $H(X,Y) = -2(0.45 \ln 0.45) - 2(0.05 \ln 0.05) = 1.018$ nats. That is more than the 0.693 nats of one fair bit and less than the 1.386 nats of two independent ones.

The *conditional entropy* is the surprise left in Y once X is known, which is the average entropy of the rows of the table $p(y \mid x) = p_{X,Y}(x,y)/p_X(x)$:

$$H(Y \mid X) = -\mathbb{E}_{(x,y)\sim P_{X,Y}}[\log p(y \mid x)] = \sum_x p_X(x)\, H(Y \mid X = x). \tag{37.52}$$

For the flipped bit the rows are $(0.9, 0.1)$ and $(0.1, 0.9)$, each with entropy $H_b(0.1) = 0.325$ nats, so $H(Y \mid X) = 0.325$. After X has been seen, Y keeps 0.325 of its 0.693 nats of surprise.

The three numbers satisfy $1.018 = 0.693 + 0.325$. This is the chain rule $H(X,Y) = H(X) + H(Y \mid X)$ of the same subsection: the surprise in the pair is the surprise in X plus what Y adds once X is known.

37.6.1.2 Mutual Information

Conditional entropy is the part of the surprise in Y that X cannot explain. We want the part the two variables share. If X and Y were independent, their joint distribution would be the product of the marginals, $P_X \otimes P_Y$, with $p_X(x)\,p_Y(y)$ in every cell. The *mutual information* is the KL divergence of the actual joint distribution from that product:

$$I(X;Y) = D_{\mathrm{KL}}(P_{X,Y} \,\|\, P_X \otimes P_Y) = \mathbb{E}_{(x,y)\sim P_{X,Y}}\left[\log \frac{p_{X,Y}(x,y)}{p_X(x)\,p_Y(y)}\right]. \tag{37.53}$$

For the flipped bit the product of the marginals puts 0.25 in every cell. The ratio of the joint to the product is therefore $0.45/0.25 = 1.8$ on the diagonal, where X and Y agree, and $0.05/0.25 = 0.2$ off it, so

$$I(X;Y) = 0.9 \ln 1.8 + 0.1 \ln 0.2 = 0.529 - 0.161 = 0.368 \text{ nats}.$$

Agreements occur 1.8 times as often as independence predicts, and disagreements one fifth as often. Mutual information averages the logarithms of these factors under the joint distribution. A KL divergence is also the extra length of a code built from the wrong model (Section 37.2), so $I(X;Y)$ is the number of nats per pair that a code wastes when it treats X and Y as independent. Section 17.8 meets this penalty when a network predicts two masked tokens independently.

The definition can also be read one row of the table at a time. Dividing the joint and the product by $p_X(x)$ turns the ratio into $p(y \mid x)/p_Y(y)$, and the expectation over the pair into an expectation over x of a divergence between two distributions of y:

$$I(X;Y) = \mathbb{E}_{x\sim P_X}\Big[D_{\mathrm{KL}}\big(p(y \mid x) \,\|\, p_Y(y)\big)\Big]. \tag{37.54}$$

Mutual information is how far, on average, learning x moves the distribution of Y away from its marginal. For the flipped bit each row, $(0.9, 0.1)$ or $(0.1, 0.9)$, lies $0.9 \ln 1.8 + 0.1 \ln 0.2 = 0.368$ nats from the marginal $(0.5, 0.5)$. This form suits a pair whose members play different roles, such as an input and what a noisy encoder makes of it, and we return to it in Section 37.6.4.

Proposition (properties of mutual information). *For any pair (X, Y), $I(X;Y) = I(Y;X) \geq 0$, with equality if and only if X and Y are independent. When the entropies involved are finite,*

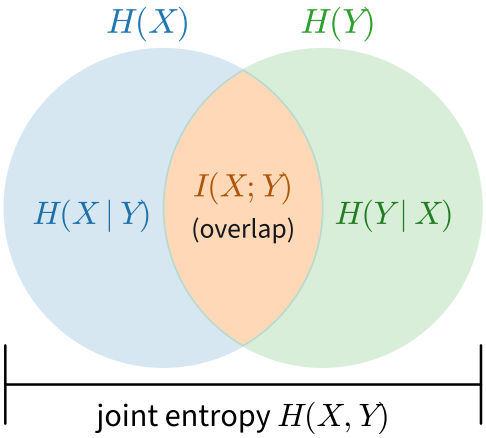
$$I(X;Y) = H(X) + H(Y) - H(X,Y) = H(X) - H(X \mid Y) = H(Y) - H(Y \mid X).$$

Proof. The ratio in Equation 37.53 is unchanged when X and Y trade places, which gives symmetry. Nonnegativity and the equality case are Gibbs' inequality for the two distributions $P_{X,Y}$ and $P_X \otimes P_Y$: the divergence is nonnegative, and it is zero only when the two distributions are the same, $p_{X,Y}(x,y) = p_X(x)\,p_Y(y)$, which is the definition of independence. For the identities, split the logarithm of the ratio into three terms and take expectations:

$$I(X;Y) = \mathbb{E}\left[\log p_{X,Y}(x,y)\right] - \mathbb{E}\left[\log p_X(x)\right] - \mathbb{E}\left[\log p_Y(y)\right] = -H(X,Y) + H(X) + H(Y).$$

Substituting the chain rule in its two orders, $H(X,Y) = H(Y) + H(X \mid Y) = H(X) + H(Y \mid X)$, gives the other two forms. ■

So X says as much about Y as Y says about X. The last form reads mutual information as a reduction in surprise: seeing X lowers the surprise in Y from $H(Y)$ to $H(Y \mid X)$, for the flipped bit from 0.693 to 0.325 nats. Nonnegativity adds that $H(X,Y) \le H(X) + H(Y)$, with equality only at independence: a pair never carries more surprise than its members taken separately. Figure 37.13 draws the identities. Each identity of the proposition is one way of reading off the area of the overlap.



$$\begin{aligned} I(X;Y) &= H(X) - H(X \mid Y) \\ &= H(Y) - H(Y \mid X) \\ &= H(X) + H(Y) - H(X,Y) \end{aligned}$$

Figure 37.13: Mutual information $I(X;Y)$ as the overlap of the entropies $H(X)$ and $H(Y)$: the crescents are the conditional entropies $H(X \mid Y)$ and $H(Y \mid X)$, the union is the joint entropy $H(X,Y)$, and the three equivalent expressions for $I(X;Y)$ follow by reading off the areas.

The function `mutual_information` implements the divergence form for any table, and we reuse it throughout. The cell also evaluates the last identity, with $H(Y \mid X)$ as the average entropy of the rows.

```
p_xy = np.array([[0.45, 0.05],    # rows x, columns y: a fair bit X and
                 [0.05, 0.45]])   # its copy Y, flipped with probability 0.1

def entropy(p):
    """Entropy of a p.m.f. of any shape, in nats (0 log 0 = 0)."""
    p = p[p > 0]
    return float(-(p * np.log(p)).sum())

def mutual_information(p_xy):
    """I(X;Y) of a discrete joint p.m.f. (rows x, columns y), in nats."""
    p_x = p_xy.sum(axis=1, keepdims=True)
    p_y = p_xy.sum(axis=0, keepdims=True)
```

```
    m = p_xy > 0                                  # 0 log 0 = 0
    return float((p_xy[m] * np.log(p_xy[m] / (p_x * p_y)[m])).sum())

p_x, p_y = p_xy.sum(axis=1), p_xy.sum(axis=0)     # marginals from the joint
H_rows = sum(p_x[i] * entropy(p_xy[i] / p_x[i]) for i in range(2))   # H(Y|X)
I = mutual_information(p_xy)
print(f'I(X;Y) = {I:.4f} nats (KL form) = {entropy(p_y) - H_rows:.4f} '
      f'(H(Y) - H(Y|X)) = {I / np.log(2):.3f} bits')
```

```
I(X;Y) = 0.3681 nats (KL form) = 0.3681 (H(Y) - H(Y|X)) = 0.531 bits
```

Both forms return 0.3681 nats, or 0.531 bits: a copy that is wrong one time in ten carries about half a bit about the original.

37.6.1.3 The Gaussian Pair

The tables so far were discrete, and most pairs in deep learning are not: activations and embeddings are continuous. One continuous pair can be computed by hand, the bivariate Gaussian, and the rate–distortion function and the bottleneck later in this section are built on it. For densities, the entropies become the differential entropies h of Section 37.1.4.1, which can be negative and which change with the units of measurement. The KL form of I does neither. The chain-rule algebra never used discreteness, so $I(X;Y) = h(X) + h(Y) - h(X,Y) = h(Y) - h(Y \mid X)$ whenever these are finite. Here $h(Y \mid X)$ is the differential entropy of the conditional density $p(y \mid x)$, averaged over x.

Proposition (Gaussian mutual information). *If (X,Y) is bivariate Gaussian with variances $\sigma_X^2, \sigma_Y^2 > 0$ and correlation coefficient ρ, $\rho^2 < 1$, then*

$$I(X;Y) = -\tfrac{1}{2}\log\left(1-\rho^2\right). \tag{37.55}$$

Proof. Use $I(X;Y) = h(Y) - h(Y \mid X)$. A Gaussian with variance σ^2 has differential entropy $\frac{1}{2}\log(2\pi e\sigma^2)$, as computed in Section 37.1.4.3, so $h(Y) = \frac{1}{2}\log(2\pi e\sigma_Y^2)$. Given $X = x$, the variable Y is Gaussian with variance $\sigma_Y^2(1-\rho^2)$, whatever x is. This is the variance of the residual that is left when the best linear prediction from X is subtracted from Y, and for a Gaussian pair the residual is independent of X (Section 36.3.3.3). So $h(Y \mid X) = \frac{1}{2}\log\left(2\pi e\sigma_Y^2(1-\rho^2)\right)$, and the difference is $-\frac{1}{2}\log(1-\rho^2)$. ■

The answer depends only on $1-\rho^2$, the fraction of the variance of Y that X leaves unexplained. The units of Y have cancelled, a first instance of the invariance proved in Section 37.6.1.5.

At $\rho = 0$ the variables are independent and $I = 0$. As $\rho \to \pm 1$ the joint density concentrates on a line and $I \to \infty$. In between, $\rho = 0.5$ gives 0.144 nats, $\rho = 0.9$ gives 0.830 nats and $\rho = 0.99$ gives 1.959 nats. The pair with standard normal marginals and $\rho = 0.9$ returns throughout the section, and we call it the Gaussian pair at $\rho = 0.9$.

Mutual information can indeed be infinite. The case returns twice below, for the deterministic layers of a network and for the encoder of the bottleneck. If $Y = X$ for a continuous X, knowing X fixes Y to infinitely many digits, and the same happens whenever Y is a function of X that takes no single value with positive probability. It cannot happen when Y is discrete with finite entropy: then $I(X;Y) = H(Y) - H(Y \mid X) \le H(Y)$ whatever

X is, where $H(Y \mid X)$ averages the entropy of $p(\cdot \mid x)$ over x. For a standard normal X, the sign of X carries $\ln 2$ nats. The same bound caps the Jensen–Shannon divergence of Section 16.1 at $\ln 2$: as shown there, it is the mutual information between a sample and the binary label of its source.

37.6.1.4 Pointwise Mutual Information

Mutual information is an average over all pairs of outcomes, and the average hides which pairs contribute. Often a single pair is what we care about: does seeing the word *new* make the next word *york* more likely than chance? The summand of Equation 37.53 answers this. It is the *pointwise mutual information*

$$\operatorname{pmi}(x, y) = \log \frac{p_{X,Y}(x, y)}{p_X(x)\, p_Y(y)}, \tag{37.56}$$

which compares how often the particular pair (x, y) occurs with how often it would occur if X and Y were independent. Positive pmi means that the two outcomes attract (they co-occur more often than chance predicts), negative pmi means that they repel, and $I(X;Y) = \mathbb{E}_{(x,y)\sim P_{X,Y}}[\operatorname{pmi}(x, y)]$ is the average attraction. For the flipped bit the pmi is $\ln 1.8 = +0.588$ on the two agreements and $\ln 0.2 = -1.609$ on the two disagreements, the two logarithms we averaged above.

Pointwise mutual information is widely used in natural language processing (Church and Hanks 1990), where it separates *collocations*, word pairs that belong together, from words that merely co-occur because both are common. Suppose we count consecutive word pairs (bigrams) in a small corpus and tally how often each of the first words *new*, *machine*, *the* is followed by each of *york*, *learning*, *day*:

count	york	learning	day
new	38	3	9
machine	1	46	3
the	11	21	168

The raw counts are misleading. “The day” is by far the most frequent pair (168 occurrences against 38 for “new york”) because *the* and *day* are both frequent words. The cell divides each joint frequency by the product of the frequencies of its two words and prints the logarithm for four pairs, with the mutual information of the whole table as the last line.

```
counts = np.array([[38., 3., 9.],      # new york / new learning / new day
                   [1., 46., 3.],      # machine ...
                   [11., 21., 168.]])  # the ...
first, second = ['new', 'machine', 'the'], ['york', 'learning', 'day']

p = counts / counts.sum()
pmi = np.log(p / (p.sum(axis=1, keepdims=True) * p.sum(axis=0, keepdims=True)))
for (i, j) in [(0, 0), (1, 1), (2, 2), (2, 0)]:
    print(f'pmi({first[i]}, {second[j]}) = {pmi[i, j]:6.3f} nats '
          f'(count {counts[i, j]:.0f})')
print(f'I(first; second) = {float((p * pmi).sum()):.4f} nats')
```

```
pmi(new, york) =  1.517 nats (count 38)
pmi(machine, learning) =  1.372 nats (count 46)
pmi(the, day) =  0.336 nats (count 168)
pmi(the, york) = -1.109 nats (count 11)
I(first; second) = 0.4146 nats
```

After the correction for the marginals, the collocations receive the higher scores: "new york" scores 1.52 nats and "machine learning" 1.37, while the frequent "the day" reaches only 0.34. The pair "the york" comes out negative, -1.11 nats: it occurs three times less often than independence would predict. The mutual information of the table, 0.41 nats, summarizes how strongly first words constrain second words on average. The same correction survives in learned word vectors (Section 19.1). The skip-gram model learns its vectors by telling each observed pair of neighboring words from K random pairs, and in doing so it implicitly factorizes the matrix of shifted pointwise mutual informations, $\text{pmi}(x, y) - \log K$ (Levy and Goldberg 2014), as Exercise 9 of Section 37.7 derives.

37.6.1.5 Mutual Information and Correlation

The correlation coefficient $\rho(X, Y) = \text{Cov}(X, Y)/(\sigma_X \sigma_Y)$ of Section 36.2 detects linear dependence only: $\rho(X, Y) = 0$ does not imply independence. Mutual information is zero only at independence. The pair of the opening shows the difference in numbers.

Let $X \sim \mathcal{N}(0, 1)$ and $Y = X^2 + 0.3\,\epsilon$ with independent $\epsilon \sim \mathcal{N}(0, 1)$, a pair with $\rho = 0$. The identity $I(X; Y) = h(Y) - h(Y \mid X)$ makes I computable from one-dimensional integrals. Given $X = x$, the only randomness in Y is the noise, and shifting by the known constant x^2 does not change differential entropy, so $h(Y \mid X) = h(0.3\,\epsilon) = \frac{1}{2}\ln(2\pi e \cdot 0.09) = 0.215$ nats. For $h(Y)$ a bound comes before the value. Y has variance $\text{Var}(X^2) + 0.09 = 2.09$, and no density with that variance has more differential entropy than the Gaussian's $\frac{1}{2}\ln(2\pi e \cdot 2.09) = 1.79$ nats (Section 37.1.4.3), so $I(X; Y) \le 1.79 - 0.215 = 1.57$ nats. The density of Y is a mixture over x, $p_Y(y) = \int \mathcal{N}(x; 0, 1)\,\mathcal{N}(y; x^2, 0.09)\,dx$, which has no closed form, so we integrate it numerically to obtain $h(Y)$.

```
normal = lambda t, var: np.exp(-t**2 / (2 * var)) / np.sqrt(2 * np.pi * var)
xs = np.linspace(-6, 6, 6001)              # quadrature grid for X
ys = np.linspace(-2.5, 30, 6501)           # grid for Y = X^2 + 0.3 eps
dx, dy = xs[1] - xs[0], ys[1] - ys[0]
p_Y = np.concatenate([                     # p_Y(y) = int N(x;0,1) N(y;x^2,0.09) dx
    (normal(xs, 1) * normal(c[:, None] - xs**2, 0.09)).sum(axis=1) * dx
    for c in np.array_split(ys, 13)])      # 13 chunks keep memory small
h_Y = float(-(p_Y[p_Y > 0] * np.log(p_Y[p_Y > 0])).sum() * dy)
h_noise = 0.5 * np.log(2 * np.pi * np.e * 0.09)   # h(Y|X) = h(0.3 eps)
print(f'h(Y) = {h_Y:.3f}, h(Y|X) = {h_noise:.3f}, '
      f'I(X;Y) = h(Y) - h(Y|X) = {h_Y - h_noise:.3f} nats')
```

```
h(Y) = 1.356, h(Y|X) = 0.215, I(X;Y) = h(Y) - h(Y|X) = 1.141 nats
```

The pair shares 1.141 nats. That is below the bound of 1.57, because the density of Y, piled up near zero with a long tail to the right, is far from Gaussian. It is also more than the 0.830 nats of the Gaussian pair at $\rho = 0.9$, and this from a pair whose correlation is zero. Figure 37.14 shows the three cases side by side.

Correlation and mutual information differ in a second way, which concerns coordinates. Recording a temperature in Fahrenheit instead of Celsius, or a price as its logarithm, leaves

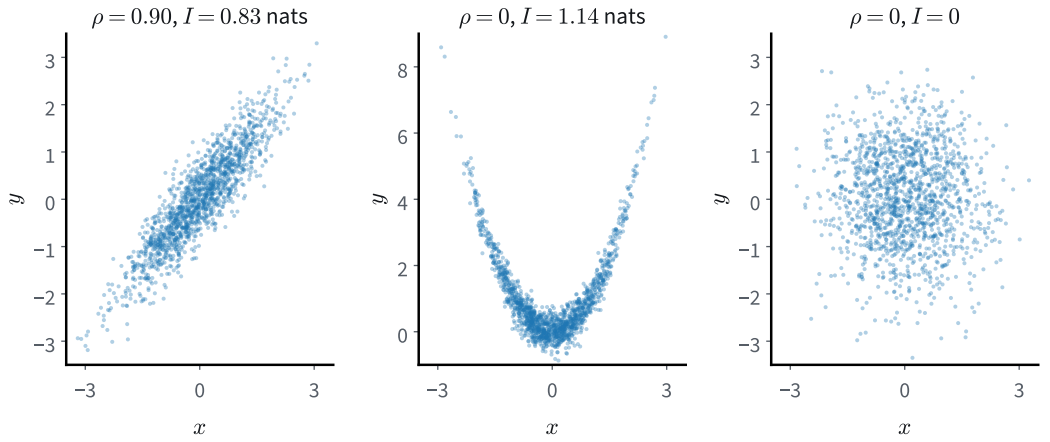


Figure 37.14: Three pairs compared by correlation and by mutual information. (a) A Gaussian pair with correlation 0.9. (b) $Y = X^2 + 0.3\epsilon$, whose correlation vanishes by symmetry although X nearly determines Y; the vertical scale of (b) differs. (c) An independent pair. Correlation cannot tell (b) from (c), and mutual information ranks (b) above (a).

what it tells about anything else as it was, and a measure of dependence should not notice the change.

Proposition (reparameterization invariance). *Let ϕ and ψ be invertible maps of the values of X and of Y. Then $I(\phi(X); \psi(Y)) = I(X; Y)$.*

Proof. For discrete variables, invertible maps only relabel the outcomes: the joint distribution of $(\phi(X), \psi(Y))$ takes the same values as that of (X, Y), at different labels, and the sum in Equation 37.53 is unchanged. For continuous variables we prove the smooth case, which shows the mechanism; Section 37.6.2.2 removes the assumption. Let ϕ and ψ be continuously differentiable with continuously differentiable inverses, and change variables $(x, y) \mapsto (\phi(x), \psi(y))$. The joint density of the new pair at $(\phi(x), \psi(y))$ is $p_{X,Y}(x, y)\,|\det D\phi(x)|^{-1}|\det D\psi(y)|^{-1}$, and the two marginal densities pick up the factors $|\det D\phi(x)|^{-1}$ and $|\det D\psi(y)|^{-1}$. The factors cancel in the ratio, so the integrand of Equation 37.53 takes the same value at corresponding points, and its expectation is the same. A set of exceptional points of probability zero, such as the origin for $x \mapsto x^3$, does not matter. ■

Differential entropy has no such invariance: there the Jacobian factor does not cancel, which is why h changes with the units. Correlation is preserved only by affine maps, up to sign. Apply the invertible map $x \mapsto x^3$ to the first coordinate of the Gaussian pair at $\rho = 0.9$. The mutual information stays at 0.830 nats, while the correlation falls from 0.9 to $3\rho/\sqrt{15} = 0.697$, because $\mathbb{E}[X^3 Y] = 3\rho$ and $\mathrm{Var}(X^3) = 15$ for a standard X. Mutual information asks whether two variables are informative about each other in any coordinates, not whether they line up.

37.6.2 Processing and Prediction

This part holds the three results that the opening announced. The first two are guarantees: processing cannot increase information, and accurate prediction needs a minimum

amount of it. Both rest on one identity, a chain rule for mutual information. The third is the bound that the loss of any model places on the information in its input.

37.6.2.1 Conditional Mutual Information and the Chain Rule

We extend the flipped bit by one step. Let Z be Y flipped once more, independently, with probability 0.2: a fair bit sent over two noisy lines in series, or a label, a noisy feature and a noisy copy of the feature (Figure 37.15). Two flips in series are one flip. The bit arrives flipped if exactly one of the two lines flips it, with probability $0.1 \cdot 0.8 + 0.9 \cdot 0.2 = 0.26$, so the pair (X, Z) is a flipped bit with $\delta = 0.26$, and $I(X;Z) = \ln 2 - H_b(0.26) = 0.120$ nats. The second line has removed two thirds of the 0.368 nats that Y carried.

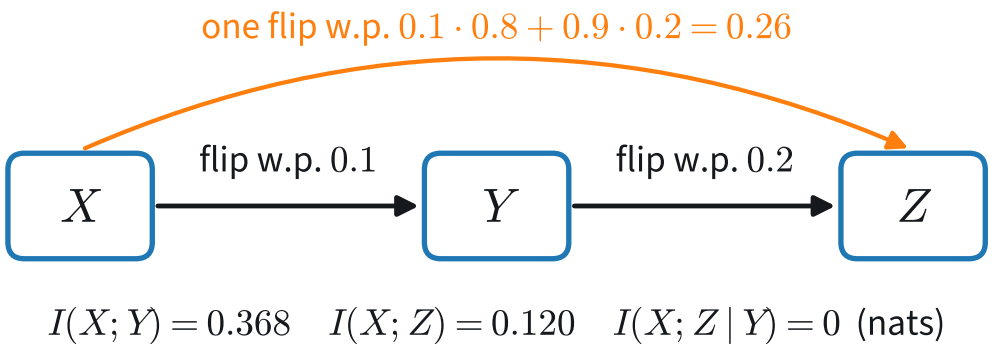


Figure 37.15: Two flips in series: a fair bit X is flipped with probability 0.1 to give Y, and Y is flipped with probability 0.2 to give Z; the two flips compose to one flip with probability 0.26. The second flip removes two thirds of the information, and $I(X;Z \mid Y) = 0$ says that Z knows nothing about X that Y does not.

Three variables raise a question that two do not: how much do X and Y share once Z is known? Like entropy, mutual information has a conditional version for this,

$$I(X;Y \mid Z) = H(X \mid Z) - H(X \mid Y, Z), \tag{37.57}$$

where $H(X \mid Y, Z)$ is the conditional entropy given the pair (Y, Z). Writing each conditional entropy as an expectation and collecting the logarithms gives the divergence form $I(X;Y \mid Z) = \mathbb{E}_z\big[D_{\mathrm{KL}}(P_{X,Y|z} \,\|\, P_{X|z} \otimes P_{Y|z})\big]$, the definition of mutual information applied under each conditional distribution. As an average of divergences it is nonnegative, and it is zero if and only if X and Y are conditionally independent given Z.

The information that the pair (Y, Z) carries about X now splits in two. Subtract and add $H(X \mid Z)$ in $I(X;Y,Z) = H(X) - H(X \mid Y, Z)$:

$$I(X;Y,Z) = \underbrace{H(X) - H(X \mid Z)}_{I(X;Z)} + \underbrace{H(X \mid Z) - H(X \mid Y, Z)}_{I(X;Y|Z)}.$$

This is the *chain rule for mutual information*, and it holds in general when each term is written in its KL form. What (Y, Z) tells about X is what Z tells plus what Y adds once Z is known.

The rule can take Y first just as well, and for the two flips in series the two orders split one number in two ways. Z was made from Y and fresh noise, so once Y is known it says nothing more about X: $I(X;Z \mid Y) = 0$, and with Y first, $I(X;Y,Z) = I(X;Y) + 0 = 0.368$ nats. With Z first the same 0.368 nats are $I(X;Z) + I(X;Y \mid Z) = 0.120 + I(X;Y \mid Z)$, so $I(X;Y \mid Z) = 0.248$ nats. Conditioning on Z has lowered what Y says about X, because here Z already holds part of it. Conditioning does not always do so, and the next subsection has a case where it raises information.

37.6.2.2 The Data-Processing Inequality

A hidden layer is computed from the layer below it, and that layer from the input. Can processing of this kind create information about the input that the earlier layer did not already carry? On the two noisy lines it could not, and the calculation used nothing that is special to bits.

Random variables X, Y, Z form a *Markov chain*, written $X \to Y \to Z$, when Z depends on (X, Y) only through Y: $p(z \mid x, y) = p(z \mid y)$. This is the structure of processing: Y is computed from X (a measurement, a feature map, a hidden layer), then Z is computed from Y alone, possibly with fresh randomness. Equivalently, X and Z are conditionally independent given Y, so $I(X;Z \mid Y) = 0$.

Proposition (data-processing inequality). *If $X \to Y \to Z$ is a Markov chain and the entropies involved are finite, then*

$$I(X;Z) \le I(X;Y),$$

with equality if and only if $I(X;Y \mid Z) = 0$, that is, if and only if $X \to Z \to Y$ is also a Markov chain (Cover and Thomas 1999).

Proof. Expand $I(X;Y,Z)$ by the chain rule in both orders:

$$I(X;Z) + I(X;Y \mid Z) = I(X;Y,Z) = I(X;Y) + I(X;Z \mid Y).$$

The Markov property makes the last term zero, which leaves $I(X;Z) = I(X;Y) - I(X;Y \mid Z) \le I(X;Y)$, since conditional mutual information is nonnegative. Equality holds if and only if $I(X;Y \mid Z) = 0$. ■

No processing of Y, deterministic or random, can increase the information it carries about X. For the two flips in series the inequality reads $0.120 \le 0.368$.

The inequality is a statement about Markov chains, not about conditioning in general, and conditioning can create dependence. Let X and Y be independent fair bits and $Z = X \oplus Y$ their exclusive or. Then X and Y share nothing, but once Z is known, Y determines X. The cell computes the mutual informations of the two flips in series from their three-way table, and then those of this triple.

```
def cond_mi(p_abc):
    """I(A;B|C) of a joint p.m.f. indexed [a, b, c], in nats."""
    p_ac, p_bc, p_c = p_abc.sum(1), p_abc.sum(0), p_abc.sum((0, 1))
    m = p_abc > 0                                   # 0 log 0 = 0
    ratio = ((p_abc * p_c[None, None, :])[m]
```

```
                / (p_ac[:, None, :] * p_bc[None, :, :])[m])
    return float((p_abc[m] * np.log(ratio)).sum())

flip = lambda d: np.array([[1 - d, d], [d, 1 - d]])   # p(output | input)
p_xyz = 0.5 * flip(0.1)[:, :, None] * flip(0.2)[None, :, :]   # X -> Y -> Z
I_xy, I_xz = mutual_information(p_xyz.sum(2)), mutual_information(p_xyz.sum(1))
I_xz_y, I_xy_z = cond_mi(p_xyz.transpose(0, 2, 1)), cond_mi(p_xyz)
print(f'I(X;Y) = {I_xy:.4f}, I(X;Z) = {I_xz:.4f}, '
      f'I(X;Z|Y) = {I_xz_y:.4f}, I(X;Y|Z) = {I_xy_z:.4f}')
print(f'I(X;Y,Z) by the chain rule: {I_xz + I_xy_z:.4f} = {I_xy + I_xz_y:.4f}')
p_xor = np.zeros((2, 2, 2))
for a, b in np.ndindex(2, 2):
    p_xor[a, b, a ^ b] = 0.25          # independent fair bits, Z = X xor Y
print(f'XOR: I(X;Y) = {mutual_information(p_xor.sum(2)):.4f}, '
      f'I(X;Y|Z) = {cond_mi(p_xor):.4f} nats')
```

```
I(X;Y) = 0.3681, I(X;Z) = 0.1201, I(X;Z|Y) = 0.0000, I(X;Y|Z) = 0.2480
I(X;Y,Z) by the chain rule: 0.3681 = 0.3681
XOR: I(X;Y) = 0.0000, I(X;Y|Z) = 0.6931 nats
```

For the two flips, $I(X; Z \mid Y) = 0$ is the Markov property in numbers, and both expansions of $I(X; Y, Z)$ give 0.3681 nats. For the exclusive or, conditioning on Z raises the information between X and Y from 0 to the full $\ln 2 = 0.693$ nats of a bit.

When does processing lose nothing? By the proposition, only when $X \to Z \to Y$ is also a Markov chain: Z must hold everything that Y says about X. An invertible map $Z = g(Y)$ does, since Y can be recomputed from Z. A map that discards part of Y loses nothing either, provided that what it discards is unrelated to X. The number of heads in a sequence of coin flips keeps everything the sequence says about the bias of the coin, because the order of the flips says nothing about it. A summary $T(Y)$ of this kind, for which $p(x \mid y)$ depends on y only through $T(y)$ and therefore $I(X; T(Y)) = I(X; Y)$, is called a *sufficient statistic*. The sufficient statistics of an exponential family (Section 36.4) are summaries of this kind: there Y is the data and X the parameter θ, with a prior on θ so that $I(\theta; Y)$ is defined.

The inequality itself needs neither finite entropies nor densities. Mutual information is a KL divergence, and no map of the data, random or deterministic, increases a KL divergence. Section 37.3.3.3 proves this for countable sets, and it holds in general; Exercise 5 turns it into a proof of the inequality. Applied to an invertible map and then to its inverse, the inequality gives the invariance proposition of Section 37.6.1.5 without the smoothness assumed there.

Applied to a network, the inequality is easy to over-read. The layers form a Markov chain $X \to Z_1 \to Z_2 \to \cdots$, so $I(X; Z_\ell)$ cannot increase with ℓ. From here to the end of this part, Y no longer stands for the middle of a chain but for a label. Whatever the network knows of the label it knows through the input, so $Y \to X \to Z_1 \to Z_2 \to \cdots$ is a Markov chain too, and $I(Y; Z_\ell)$ cannot increase with ℓ either. No architecture can create label information that the input lacks, and no later layer can recover what an earlier layer discarded.

This does not make depth harmful. Layers do not add information about the label; they rearrange it so that a simple final layer can read it. Let the label be a fair bit, and let the input be uniform on the circle of radius 1 around the origin when $Y = 0$ and on the circle of radius 2 when $Y = 1$. The input determines the label, so $I(Y; X) = \ln 2$, yet no line separates a circle from the one inside it. The best linear classifier puts its boundary tangent to the inner circle. It gets the inner circle right and, of the outer circle, only the third that

lies beyond the line, so it is right two times in three. The layer $Z = \|X\|^2$ takes the values 1 and 4, and a threshold between them is always right. It carries $I(Y;Z) = \ln 2$ nats, no more than the input did. The layer added nothing and made everything readable, and Exercise 3 defines a kind of information that counts only what a given family of classifiers can read.

The chain that starts from the input, $X \to Z_1 \to Z_2 \to \cdots$, needs a caution. A deterministic layer has $H(Z_\ell \mid X) = 0$, so for a discrete or quantized representation $I(X;Z_\ell)$ is just the entropy $H(Z_\ell)$ of the layer, and for a continuous one it is typically infinite (Section 37.6.1.3). As a measure of what a layer keeps of its input, $I(X;Z_\ell)$ therefore says something only when the layer is noisy or its output is discrete.

37.6.2.3 Fano's Inequality

Accurate prediction needs mutual information. A classifier that reaches 5% error on $1{,}000$ classes must see an input that carries most of the label's information, and Fano's inequality says how much.

Proposition (Fano's inequality). *Let Y take values in a set of k elements, let $\hat{Y} = g(X)$ be any estimate of Y computed from X and taking values in the same set, and let $P_e = P(\hat{Y} \neq Y)$ be its error probability. Then*

$$H_b(P_e) + P_e \log(k-1) \geq H(Y \mid X) = H(Y) - I(X;Y). \tag{37.58}$$

The left side is the cost of repairing the estimate: $H_b(P_e)$ nats to say whether it is wrong and, when it is, at most $\log(k-1)$ nats to say which of the other labels is right. Whoever holds the estimate and the repair knows Y, so the repair cannot cost less than the uncertainty about Y that the estimate leaves, and an estimate computed from X leaves at least as much as X does.

The proof makes this argument exact with one device, an indicator of the error. The rest is the chain rule.

Proof. Let $W = \mathbf{1}\{\hat{Y} \neq Y\}$ indicate an error. We expand $H(W, Y \mid \hat{Y})$, the uncertainty about the error and the label that remains once the estimate is known, in two orders. First, W is a function of $(Y, \hat{Y})$ and adds nothing, so $H(W, Y \mid \hat{Y}) = H(Y \mid \hat{Y})$. Second, $H(W, Y \mid \hat{Y}) = H(W \mid \hat{Y}) + H(Y \mid W, \hat{Y})$, and we bound both terms.

- Conditioning cannot raise entropy, because $H(W) - H(W \mid \hat{Y}) = I(W;\hat{Y}) \geq 0$. So $H(W \mid \hat{Y}) \leq H(W) = H_b(P_e)$.
- Given $W = 0$ the label equals the estimate and has no uncertainty. Given $W = 1$, which has probability P_e, the label is one of the $k-1$ values other than $\hat{Y}$. So $H(Y \mid W, \hat{Y}) \leq P_e \log(k-1)$.

Together, $H(Y \mid \hat{Y}) \leq H_b(P_e) + P_e \log(k-1)$. Finally, $Y \to X \to \hat{Y}$ is a Markov chain because $\hat{Y}$ is computed from X alone, and the data-processing inequality gives $I(Y;\hat{Y}) \leq I(Y;X)$, that is, $H(Y \mid \hat{Y}) \geq H(Y \mid X)$. ∎

The count of $k-1$ other labels in the second bound requires the estimate to stay inside the label set. If it may answer "abstain", $\log k$ replaces $\log(k-1)$, and the inequality as stated fails: for a fair bit Y independent of X and an estimator that always abstains, $P_e = 1$ and the left side is $H_b(1) + \ln 1 = 0$, below $H(Y \mid X) = \ln 2$.

In the smallest case the bound is tight. For the flipped bit, estimate the label Y by the feature, $\hat{Y} = X$. Then $P_e = 0.1$ and $k = 2$, and the left side is $H_b(0.1) + 0.1 \ln 1 = 0.325$, which equals $H(Y \mid X)$. Exercise 6 shows when equality holds in general.

No classifier, however it is trained, beats the bound. On $1{,}000$ balanced classes, $H(Y) = \ln 1000 = 6.91$ nats, and solving Equation 37.58 for $I(X;Y)$ gives the information that each error rate needs.

```
H_b = lambda p: -p * np.log(p) - (1 - p) * np.log(1 - p)   # binary entropy
k = 1000
for p_e in (0.01, 0.05, 0.2, 0.5):
    need = np.log(k) - H_b(p_e) - p_e * np.log(k - 1)
    print(f'P_e = {p_e:4.2f}: I(X;Y) >= {need:.3f} of log k = {np.log(k):.3f} nats')
```

```
P_e = 0.01: I(X;Y) >= 6.783 of log k = 6.908 nats
P_e = 0.05: I(X;Y) >= 6.364 of log k = 6.908 nats
P_e = 0.20: I(X;Y) >= 5.026 of log k = 6.908 nats
P_e = 0.50: I(X;Y) >= 2.761 of log k = 6.908 nats
```

Five percent error requires at least 6.36 of the 6.91 nats of label entropy, 92% of it, and even a classifier that is wrong half the time needs 2.76 nats, two fifths of it.

37.6.2.4 The Loss Certificate

Fano turns an error rate into a floor on information. A language model is not judged by how often it guesses the next word, and like every model trained by maximum likelihood it reports a loss instead. The loss gives a floor too, and the proof takes three lines.

Proposition (a loss certifies information). *Let $q(y \mid x)$ be any model of the label given the input, with cross-entropy loss $L = \mathbb{E}[-\log q(Y \mid X)]$ under $P_{X,Y}$. Then*

$$I(X;Y) \;\geq\; H(Y) - L, \tag{37.59}$$

and for k balanced classes, $I(X;Y) \geq \log k - L$.

Proof. We have $I(X;Y) = H(Y) - H(Y \mid X)$. For each x, the cross-entropy of $q(\cdot \mid x)$ under $p(\cdot \mid x)$ exceeds the entropy of $p(\cdot \mid x)$ by $D_{\mathrm{KL}}(p(\cdot \mid x) \,\|\, q(\cdot \mid x)) \geq 0$ (Section 37.1). Averaging over x gives $H(Y \mid X) \leq L$. ■

The loss on held-out data estimates L, so the test loss of any classifier or language model is a statement about the information its input carries. A classifier of ten balanced classes with a test loss of 0.3 nats certifies that its input holds at least $\ln 10 - 0.3 = 2.0$ of the 2.30 nats of the label. The model need not be good, or even sensible, for the bound to be valid; a bad model only certifies less. This is the bound of Barber and Agakov (2003) with a classifier as the decoder, and Section 37.7.3 develops it into a family of estimators.

The certificate is tight when the model is the true conditional, and only then. On the flipped bit, a model that predicts $\hat{Y} = X$ with confidence c has loss $L = -0.9 \log c - 0.1 \log(1-c)$, which we evaluate at four confidences.

```
for c in (0.6, 0.8, 0.9, 0.99):
    L = -(0.9 * np.log(c) + 0.1 * np.log(1 - c))   # cross-entropy of q
```

```
    print(f'confidence {c:4.2f}: loss {L:.4f}, '
          f'certifies I(X;Y) >= {np.log(2) - L:.4f} nats')
```

```
confidence 0.60: loss 0.5514, certifies I(X;Y) >= 0.1418 nats
confidence 0.80: loss 0.3618, certifies I(X;Y) >= 0.3314 nats
confidence 0.90: loss 0.3251, certifies I(X;Y) >= 0.3681 nats
confidence 0.99: loss 0.4696, certifies I(X;Y) >= 0.2236 nats
```

Only the calibrated model, with $c = 0.9$, certifies the full 0.368 nats. An overconfident model ($c = 0.99$) incurs a large loss on its errors and certifies only 0.22; a hesitant one ($c = 0.6$) certifies 0.14. Yet the four models make the same decisions and are wrong 10% of the time, so Fano's inequality, which sees only the error rate, certifies the full 0.368 nats for every one of them. The loss also sees the confidences, and it penalizes wrong ones. Both results bound the one quantity $H(Y \mid X)$ from above, Fano through an error rate and the certificate through a loss, and each bound becomes a lower bound on $I(X;Y)$. The two are closer than they look: Exercise 8 shows that Fano's bound is the certificate of one particular model, and that a classifier which knows where it is unsure certifies more.

37.6.3 Communication and Compression

Shannon asked two engineering questions, and their answers are a maximum and a minimum of mutual information: how much a noisy channel can carry, and how little a compressor may keep. Channel capacity can be skimmed on a first reading. Rate–distortion cannot, because the last part of the section is built on it.

37.6.3.1 Channel Capacity

How much information survives a line that flips each bit with probability δ? A memoryless *channel* is a conditional distribution $p(y \mid x)$ applied independently to each input symbol, and each symbol sent is one use of the channel. The flipped bit is a channel, with input X and output Y. The channel is fixed, but the sender chooses how often to send each input, and the *capacity* is the largest mutual information any choice achieves, in nats per use:

$$C = \max_{p(x)} I(X;Y). \tag{37.60}$$

A sender who transmits mostly zeros wastes the channel, since such an input has little entropy to convey. Shannon's channel coding theorem says what the capacity means. The *rate* of a code is the number of nats of message that it delivers per use of the channel. By coding long blocks of symbols, one can transmit at any rate below C with an error probability as small as desired, and at no rate above C (Shannon 1948; Cover and Thomas 1999). We state the theorem without proof.

For the flip channel, the *binary symmetric channel*, the capacity takes one line. Write $I(X;Y) = H(Y) - H(Y \mid X)$. Every row of $p(y \mid x)$ is $(1-\delta, \delta)$ up to order, so $H(Y \mid X) = H_b(\delta)$ whatever the input distribution, and $H(Y) \le \ln 2$ with equality if and only if Y is uniform. Since $P(Y=1) = \delta + P(X=1)(1-2\delta)$, for $\delta \neq \frac{1}{2}$ the output is uniform if and only if the input is. Hence

$$C = \ln 2 - H_b(\delta) \quad \text{nats per use}, \tag{37.61}$$

attained by the uniform input. At $\delta = 0.1$ this is the 0.368 nats of the flipped bit, whose input was uniform, and the two lines in series, $\delta = 0.26$, carry 0.120. At $\delta = \frac{1}{2}$ the output is independent of the input and the capacity is zero. At $\delta = 1$ it is $\ln 2$ again, because a receiver who knows that every bit is flipped flips it back. Flips destroy information only to the extent that the receiver cannot predict them. The cell sweeps the input distribution for the 0.1 channel, then the flip probability at the uniform input.

```
inputs = (0.1, 0.3, 0.5, 0.7, 0.9)
I_in = [mutual_information(np.array([[1 - a], [a]]) * flip(0.1)) for a in inputs]
print('P(X=1):', '  '.join(f'{a:6.2f}' for a in inputs))
print('I(X;Y):', '  '.join(f'{v:6.4f}' for v in I_in))
deltas = (0.0, 0.1, 0.26, 0.5, 1.0)
C = [np.log(2) - entropy(np.array([d, 1 - d])) for d in deltas]
print('delta: ', '  '.join(f'{d:6.2f}' for d in deltas))
print('C:     ', '  '.join(f'{c:6.4f}' for c in C))
```

```
P(X=1):   0.10    0.30    0.50    0.70    0.90
I(X;Y): 0.1463  0.3160  0.3681  0.3160  0.1463
delta:    0.00    0.10    0.26    0.50    1.00
C:      0.6931  0.3681  0.1201  0.0000  0.6931
```

An input that sends a 1 only 10% of the time has $H_b(0.1) = 0.325$ nats to offer and gets 0.146 of them through, two fifths of what the uniform input achieves. The left panel of Figure 37.16 draws the capacity as a function of δ. Networks contain channels of their own: dropout erases a feature with probability α and passes it on otherwise, and Exercise 9 finds that a binary feature gets at most $(1 - \alpha) \ln 2$ nats through it.

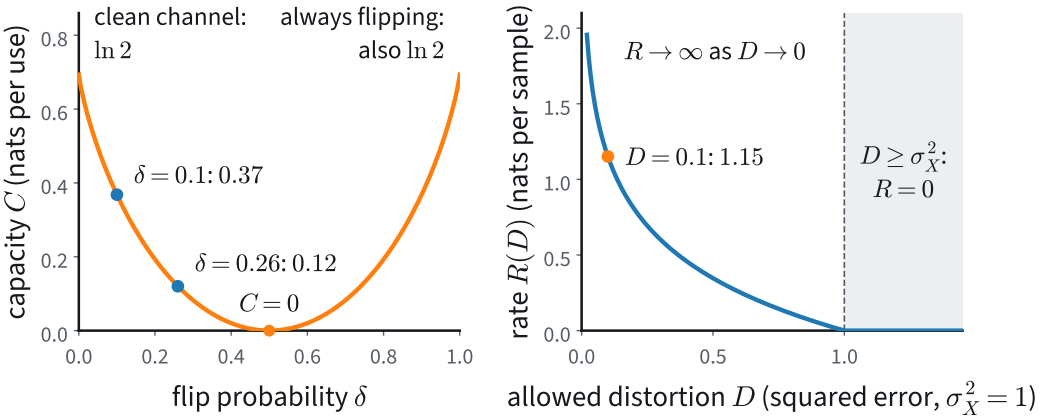


Figure 37.16: Shannon's two operational curves, in nats. Left: the capacity $C = \ln 2 - H_b(\delta)$ of the binary symmetric channel, with the flipped bit ($\delta = 0.1$) and the two flips in series ($\delta = 0.26$) marked; a channel that flips with probability $1/2$ carries nothing, and one that always flips carries a full $\ln 2$. Right: the Gaussian rate–distortion function for a source of variance $\sigma_X^2 = 1$; the rate is unbounded as $D \to 0$ and zero once D reaches the source variance.

37.6.3.2 Rate–Distortion

Lossless coding needs $H(X)$ nats per symbol (Section 37.2), and a continuous source would need infinitely many. Often an approximate reconstruction is enough: an image codec may change pixels slightly, and a learned representation may discard detail on purpose. Choose a *distortion* $d(x, \hat{x}) \geq 0$ that scores a reconstruction $\hat{x}$ of x, and allow an average distortion of at most D. How few nats per sample suffice? The *rate–distortion function* is the least mutual information an encoder $p(\hat{x} \mid x)$ must keep to meet the budget:

$$R(D) = \inf_{p(\hat{x}|x):\ \mathbb{E}[d(X,\hat{X})] \leq D} I(X; \hat{X}). \tag{37.62}$$

A stricter budget costs more information. The operational theorem mirrors the channel coding theorem, with the rate now counting the nats that a code spends per sample of the source. Block codes at any rate above $R(D)$ achieve average distortion D, and codes below it cannot (Shannon 1959).

In practice the constraint is usually replaced by a penalty: minimize $I(X; \hat{X}) + \lambda\, \mathbb{E}[d(X, \hat{X})]$ over all encoders, with a multiplier $\lambda > 0$. Every encoder is a point in the plane of distortion and rate, on or above the curve $R(D)$, and the penalized objective is constant along lines of slope $-\lambda$. The curve is convex, so the minimum lies where such a line touches it from below. A large λ selects a low distortion at a high rate. The information bottleneck of Section 37.6.4 is stated in this Lagrangian form.

Proposition (Gaussian rate–distortion). *For a source $X \sim \mathcal{N}(0, \sigma_X^2)$ under squared error $d(x, \hat{x}) = (x - \hat{x})^2$,*

$$R(D) = \frac{1}{2} \log \frac{\sigma_X^2}{D} \quad (0 < D < \sigma_X^2), \qquad R(D) = 0 \quad (D \geq \sigma_X^2). \tag{37.63}$$

Solved for the distortion, the formula reads $D = \sigma_X^2 e^{-2R}$: every additional bit of rate, $\ln 2$ nats, divides the squared error by four.

Proof. The lower bound is the harder half. It needs one fact from Section 37.1.4.3: among densities with variance s^2, the Gaussian has the largest differential entropy, $\frac{1}{2}\log(2\pi e s^2)$, and this bound grows with s^2. Take any encoder with $\mathbb{E}[(X - \hat{X})^2] \leq D$. Then

$$\begin{aligned} I(X; \hat{X}) &= h(X) - h(X \mid \hat{X}) = h(X) - h(X - \hat{X} \mid \hat{X}) \\ &\geq h(X) - h(X - \hat{X}) \\ &\geq \tfrac{1}{2}\log(2\pi e \sigma_X^2) - \tfrac{1}{2}\log(2\pi e D) = \tfrac{1}{2}\log\frac{\sigma_X^2}{D}. \end{aligned}$$

The second equality holds because subtracting the known $\hat{X}$ shifts the conditional density without changing its entropy. The first inequality holds because conditioning cannot raise entropy. The second is the maximum-entropy fact, since the error $X - \hat{X}$ has variance at most $\mathbb{E}[(X - \hat{X})^2] \leq D$. For $D \geq \sigma_X^2$ the encoder that sends nothing and reconstructs the mean 0 already meets the budget, so $R(D) = 0$.

It remains to exhibit an encoder that attains the bound when $D < \sigma_X^2$. Take $\sigma_X^2 = 1$ and add noise, $Z = X + \sigma\epsilon$ with $\epsilon \sim \mathcal{N}(0,1)$ independent of X and $\sigma > 0$. Then $\mathrm{Cov}(X,Z) = 1$ and $\mathrm{Var}(Z) = 1+\sigma^2$, so the squared correlation is $\rho_{XZ}^2 = 1/(1+\sigma^2)$, and Equation 37.55 gives the information this encoder keeps:

$$I(X; X + \sigma\epsilon) = \tfrac{1}{2}\log\left(1 + \sigma^{-2}\right). \tag{37.64}$$

Reconstruct with the conditional mean, $\hat{X} = \mathbb{E}[X \mid Z] = Z/(1+\sigma^2)$, the regression of X on Z with slope $\mathrm{Cov}(X,Z)/\mathrm{Var}(Z)$. Its mean squared error is the conditional variance, $D = 1 - 1/(1+\sigma^2) = \sigma^2/(1+\sigma^2)$, which takes every value in $(0,1)$ as σ varies. Since $\hat{X}$ is an invertible function of Z, $I(X;\hat{X}) = \frac{1}{2}\log(1+\sigma^{-2}) = \frac{1}{2}\log(1/D)$. For a general σ_X^2, rescale. ■

The encoder of the proof, a signal plus independent Gaussian noise, is a *Gaussian channel* with signal-to-noise ratio σ^{-2}, and Equation 37.64 is its capacity (Exercise 10). The noising step of a diffusion model (Chapter 17) is such a channel. Whatever the data, a coordinate noised to signal-to-noise ratio s therefore carries at most $\frac{1}{2}\log(1+s)$ nats about its clean value. The cell simulates the encoder of the proof on a million samples at $\sigma^2 = 1/9$, 1 and 3, for which $D = 0.1$, 0.5 and 0.75, and measures its squared error.

```
rng = np.random.default_rng()
x = rng.standard_normal(1_000_000)
for s2 in (1 / 9, 1.0, 3.0):                              # sigma^2
    z = x + np.sqrt(s2) * rng.standard_normal(x.size)     # encoder
    x_hat = z / (1 + s2)                                  # E[X | Z]
    D = s2 / (1 + s2)
    mse = np.mean((x - x_hat)**2)
    print(f'sigma^2 = {s2:5.3f}: D = {D:.3f}, MSE {mse:.3f}, '
          f'I(X;Z) = R(D) = {0.5 * np.log(1 + 1 / s2):.3f} nats')
```

```
sigma^2 = 0.111: D = 0.100, MSE 0.100, I(X;Z) = R(D) = 1.151 nats
sigma^2 = 1.000: D = 0.500, MSE 0.500, I(X;Z) = R(D) = 0.347 nats
sigma^2 = 3.000: D = 0.750, MSE 0.750, I(X;Z) = R(D) = 0.144 nats
```

The measured squared error agrees with D to within a few thousandths. A distortion of $D = 0.1$ costs 1.15 nats per sample, $D = 0.5$ costs 0.35, half a bit, and $D = 0.75$ costs 0.14.

The noisy encoder attains $R(D)$ as a mutual information, which is not yet a number of nats in a file. A code must map samples to definite strings, and it approaches the curve only by coding long blocks of samples jointly. A code that treats one sample at a time stays above it. The crudest one keeps only the sign of X. It spends $\ln 2 = 0.693$ nats and reconstructs $\pm\sqrt{2/\pi}$, the mean of $|X|$, with squared error $1 - 2/\pi = 0.363$, a distortion for which the curve asks only 0.506 nats. Exercise 11 measures the gap for finer quantizers.

The right panel of Figure 37.16 draws the curve. The rate grows without bound as $D \to 0$, because an exact copy of a continuous variable has infinite information, and it reaches 0 at $D = \sigma_X^2$. The steep end explains why image generators compress before they model: the last, imperceptible details of an image cost most of the nats, and Section 17.6 leaves them to an autoencoder.

37.6.4 The Information Bottleneck

A representation should keep what predicts the label and drop the rest. Dropping needs a reason, since the data-processing inequality says that discarding information never improves the best possible prediction. The reason is that the rest has a cost. A photograph of a dog also records the lighting, the lawn and where in the frame the dog stands. A representation that carries all of this takes more nats to store, and a classifier trained on finitely many photographs can mistake any of it for a sign of the label. The information bottleneck states the goal with two mutual informations. It is a rate–distortion problem in which the label supplies the distortion.

37.6.4.1 The Bottleneck Objective

Let Z be a representation of an input X produced by a stochastic encoder $p(z \mid x)$, and let Y be the label. The encoder has to be stochastic if its output is continuous: a deterministic one would typically have $I(X;Z) = \infty$ (Section 37.6.1.3). The *information bottleneck* (Tishby et al. 1999) asks for a Z that keeps little of X and much of Y:

$$\min_{p(z|x)} \; I(X;Z) - \beta\, I(Y;Z). \tag{37.65}$$

The first term is the rate, the information the representation keeps about the input; the second, weighted by $\beta > 0$, is the information it keeps about the label. The encoder sees only X, so $Y \to X \to Z$ is a Markov chain and the data-processing inequality gives $I(Y;Z) \le \min\{I(X;Y),\, I(X;Z)\}$. A representation cannot know more about the label than the input does, and it cannot predict well without keeping input information.

The multiplier sets the trade-off, and its two extremes are simple. For $\beta \le 1$, keeping nothing is optimal: by the inequality above, $I(X;Z) - \beta I(Y;Z) \ge (1-\beta)\, I(X;Z) \ge 0$, and the constant encoder, which ignores its input and has $I(X;Z) = I(Y;Z) = 0$, attains 0. A bottleneck whose optimum is the constant encoder is said to *collapse*. For large β the label term dominates, and the optimum keeps whatever predicts Y. As β grows, $I(Y;Z)$ approaches its ceiling $I(X;Y)$, and the encoder approaches a minimal sufficient statistic: the coarsest function of X that still predicts Y as well as X does. An encoder restricted to a family, as every network is, may stop short of the ceiling.

Proposition (the bottleneck as a rate–distortion problem). *If $Y \to X \to Z$ is a Markov chain, then*

$$I(Y;Z) = I(X;Y) - \mathbb{E}_{(x,z)\sim P_{X,Z}}\Big[D_{\mathrm{KL}}\big(p(y \mid x) \,\|\, p(y \mid z)\big)\Big]. \tag{37.66}$$

Proof. Expand $I(Y;X,Z)$ by the chain rule in both orders: $I(Y;Z) + I(Y;X \mid Z) = I(Y;X) + I(Y;Z \mid X)$. The Markov property makes $I(Y;Z \mid X) = 0$. In its divergence form, $I(Y;X \mid Z) = \mathbb{E}_{(x,z)}\big[D_{\mathrm{KL}}(p(y \mid x,z) \,\|\, p(y \mid z))\big]$, and the Markov property gives $p(y \mid x,z) = p(y \mid x)$. ■

Substituting this identity into the objective gives $I(X;Z) + \beta\, \mathbb{E}[d(X,Z)] - \beta I(X;Y)$ with the distortion $d(x,z) = D_{\mathrm{KL}}(p(y \mid x) \,\|\, p(y \mid z))$. The last term does not depend on the encoder, so the bottleneck is the Lagrangian form of a rate–distortion problem. Its distortion is supplied by the label rather than chosen by a designer: a representation z is a good reconstruction of x when it predicts the label as well as x does. Unlike a fixed distortion d in

ordinary rate–distortion, this one depends on the encoder through $p(y \mid z)$, which makes the problem harder.

37.6.4.2 The Gaussian Information Bottleneck

In one model every quantity of the bottleneck has a closed form (Chechik et al. 2005). Let (X, Y) be the Gaussian pair at $\rho = 0.9$, so $I(X;Y) = 0.830$ nats, and let the encoder be the rate–distortion encoder of Section 37.6.3.2,

$$Z = X + \sigma\,\epsilon, \qquad \epsilon \sim \mathcal{N}(0,1),$$

with the noise scale σ as its only parameter. The information kept about the input is Equation 37.64. For the label, $\mathrm{Cov}(Y,Z) = \mathrm{Cov}(Y,X) = \rho$ and $\mathrm{Var}(Z) = 1+\sigma^2$, so $\rho_{YZ}^2 = \rho^2/(1+\sigma^2)$, and Equation 37.55 gives both informations:

$$I(X;Z) = \tfrac{1}{2}\log\left(1+\sigma^{-2}\right), \qquad I(Y;Z) = -\tfrac{1}{2}\log\left(1-\frac{\rho^2}{1+\sigma^2}\right). \tag{37.67}$$

As σ falls from ∞ to 0, the representation moves from knowing nothing to knowing X exactly, and its label information rises from 0 to the ceiling 0.830.

To see the shape of the trade-off, write $u = I(X;Z)$ for the rate; the letter R stays with the function $R(D)$. Then $\sigma^{-2} = e^{2u} - 1$, so $1/(1+\sigma^2) = 1 - e^{-2u}$, and the label information as a function of the rate is

$$I(Y;Z) = -\tfrac{1}{2}\log\left(1-\rho^2(1-e^{-2u})\right), \qquad \frac{d\,I(Y;Z)}{du} = \frac{\rho^2}{\rho^2 + (1-\rho^2)\,e^{2u}}.$$

The derivative falls from ρ^2 at $u = 0$ toward 0, so the curve is concave. It leaves the origin with slope $\rho^2 = 0.81$: the first nat of input information yields at most 0.81 nats of label information, and later nats yield less. It approaches the ceiling $I(X;Y)$ as $u \to \infty$. Chechik et al. (2005) prove that for a Gaussian pair no encoder does better, so this curve is the frontier of the bottleneck; we use that result without proof. The multiplier β selects one point of the frontier, the minimizer of the Lagrangian, which we call the *operating point*.

Proposition (operating point and collapse). *In the Gaussian bottleneck with $\rho^2 < 1$, the Lagrangian $u - \beta\, I(Y;Z)$ is convex in u, with derivative $1-\beta\rho^2$ at $u = 0$. For $\beta \le 1/\rho^2$ its minimum is $u = 0$, the constant encoder. For $\beta > 1/\rho^2$ the minimizer and its label information are*

$$u^* = \frac{1}{2}\log\frac{\rho^2(\beta-1)}{1-\rho^2}, \qquad I(Y;Z)^* = \frac{1}{2}\log\frac{\beta-1}{\beta(1-\rho^2)}. \tag{37.68}$$

Proof. The label information is concave in u, so the Lagrangian is convex, and its derivative is $1 - \beta\rho^2/(\rho^2 + (1-\rho^2)e^{2u})$. At $u = 0$ this is $1-\beta\rho^2$. If $\beta\rho^2 \le 1$, the derivative is nonnegative at 0 and increases, so the minimum over $u \ge 0$ is at $u = 0$. Otherwise set it to zero: $\rho^2 + (1-\rho^2)e^{2u} = \beta\rho^2$, so $e^{2u} = \rho^2(\beta-1)/(1-\rho^2)$. Substituting $\rho^2 e^{-2u} = (1-\rho^2)/(\beta-1)$ into $I(Y;Z) = -\tfrac{1}{2}\log(1-\rho^2+\rho^2 e^{-2u})$ gives the second formula. ■

The general argument gave collapse for $\beta \le 1$. For this pair, at $\rho = 0.9$, it extends to $\beta \le 1/\rho^2 = 1.235$. Below the threshold, a nat of rate adds 1 to the objective but removes at most $\beta\rho^2 \le 1$, because the slope of the frontier never exceeds its value ρ^2 at the origin. Above it, rising β moves the operating point up the curve toward the ceiling: $\beta = 2$ gives $u^* = \frac{1}{2}\ln(0.81/0.19) = 0.725$ nats of rate and 0.484 nats of label information, and $\beta = 16$ gives 2.079 and 0.798. Figure 37.17 draws the frontier and the operating points in the *information plane*, which has the rate $I(X;Z)$ on its horizontal axis and the label information $I(Y;Z)$ on its vertical one.

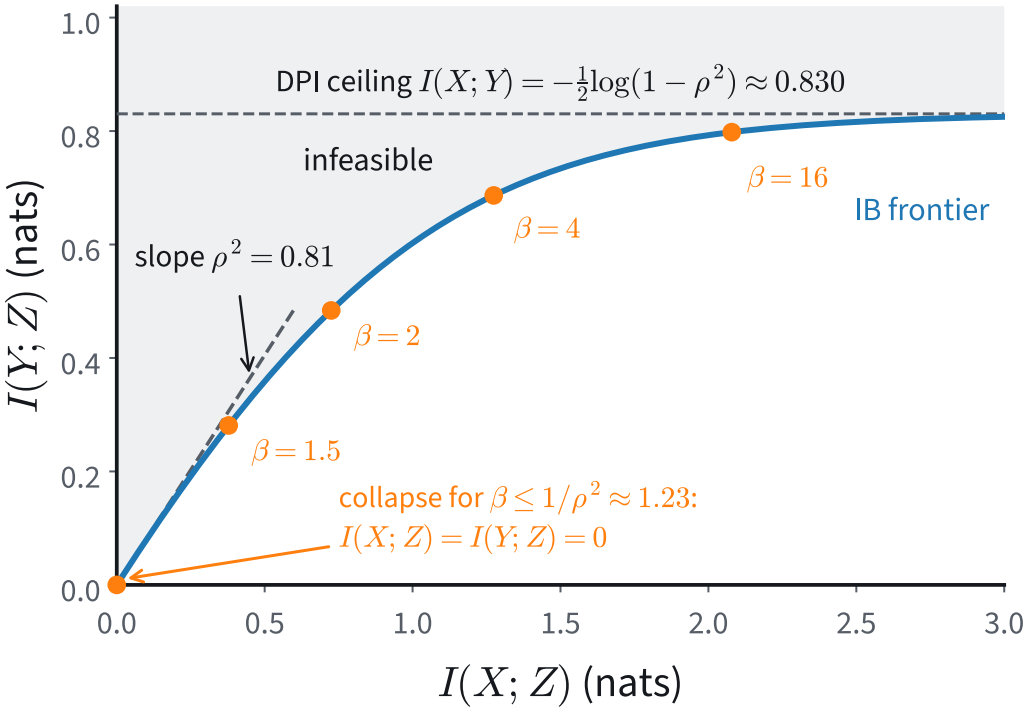


Figure 37.17: The information plane of the Gaussian bottleneck with $\rho = 0.9$. The concave frontier traces the largest attainable $I(Y;Z)$ at each rate $I(X;Z)$; the shaded region above it is infeasible, and the dashed ceiling is the data-processing limit $I(X;Y) = -\frac{1}{2}\log(1-\rho^2) \approx 0.830$ nats. The frontier leaves the origin with slope ρ^2, and the multiplier selects the point where the slope is $1/\beta$; for $\beta \le 1/\rho^2 \approx 1.23$ that point is the origin.

A threshold this sharp invites a test: the proposition predicts that the encoder keeps nothing at $\beta = 1.2$ and something at $\beta = 1.25$. We minimize the Lagrangian directly on a grid of $4{,}001$ noise scales, without using the closed form.

```
rho = 0.9
sigma = np.logspace(-2, 2, 4001)                  # encoder Z = X + sigma eps
I_xz = 0.5 * np.log(1 + sigma**-2)
I_yz = -0.5 * np.log(1 - rho**2 / (1 + sigma**2))
for beta in (1.2, 1.25, 1.5, 2, 4, 16):
    i = np.argmin(I_xz - beta * I_yz)             # Lagrangian on the grid
```

```
    u = (0.5 * np.log(rho**2 * (beta - 1) / (1 - rho**2))
         if beta * rho**2 > 1 else 0.0)          # closed-form optimum
    print(f'beta = {beta:5.2f}: I(X;Z) grid {I_xz[i]:.3f}, closed form {u:.3f};'
          f' I(Y;Z) = {I_yz[i]:.3f}')
```

```
beta =  1.20: I(X;Z) grid 0.000, closed form 0.000; I(Y;Z) = 0.000
beta =  1.25: I(X;Z) grid 0.032, closed form 0.032; I(Y;Z) = 0.026
beta =  1.50: I(X;Z) grid 0.379, closed form 0.378; I(Y;Z) = 0.281
beta =  2.00: I(X;Z) grid 0.725, closed form 0.725; I(Y;Z) = 0.484
beta =  4.00: I(X;Z) grid 1.275, closed form 1.274; I(Y;Z) = 0.687
beta = 16.00: I(X;Z) grid 2.078, closed form 2.079; I(Y;Z) = 0.798
```

The grid and the closed form agree to a thousandth of a nat, and the collapse is where the proposition puts it: at $\beta = 1.2$ the optimum is the constant encoder, and at $\beta = 1.25$ the encoder already keeps 0.032 nats. The same encoder with the same σ attains $R(D)$ for squared error with $D = \sigma^2/(1+\sigma^2)$. Rate–distortion scores it by how well it reconstructs X; the bottleneck scores it by how well it predicts Y.

37.6.4.3 The Variational Information Bottleneck

For the Gaussian pair we could evaluate both mutual informations of the objective. On real data we can evaluate neither, because the distributions are unknown: all we have is samples (x, y) and an encoder $p(z \mid x)$ of our own design. The *variational information bottleneck* (Alemi et al. 2017) replaces each term by a bound that samples can estimate.

For the label term the bound is already at hand. A decoder $q(y \mid z)$, a classifier that reads the representation, has a cross-entropy loss $L_Z = \mathbb{E}[-\log q(Y \mid Z)]$, and the loss certificate Equation 37.59 for the pair (Z, Y) reads $I(Y;Z) \geq H(Y) - L_Z$.

The rate is to be made small, so it needs a bound from above. Write it in the row form Equation 37.54, $I(X;Z) = \mathbb{E}_x\big[D_{\mathrm{KL}}(p(z \mid x) \,\|\, p(z))\big]$. The encoder $p(z \mid x)$ is known. The marginal $p(z)$ is not, because it is the average of the encoder over all inputs. Replace it by a fixed density $r(z)$ of our choice, such as a standard normal. Since $\log \frac{p(z|x)}{r(z)} = \log \frac{p(z|x)}{p(z)} + \log \frac{p(z)}{r(z)}$, taking expectations over x and z gives

$$\mathbb{E}_x\Big[D_{\mathrm{KL}}(p(z \mid x) \,\|\, r(z))\Big] = I(X;Z) + D_{\mathrm{KL}}(p(z) \,\|\, r(z)) \;\geq\; I(X;Z). \tag{37.69}$$

A wrong guess r for the marginal can only overstate the rate, and Exercise 12 computes by how much for a Gaussian encoder. Put together, the two bounds say that the KL term on the left plus βL_Z is an upper bound on the objective Equation 37.65, up to the constant $\beta H(Y)$. A network can minimize this bound, because both of its terms are averages over samples.

Anyone who has trained a variational autoencoder has met the left side of Equation 37.69 as a term of the loss. A variational autoencoder (Section 17.6) is this construction with the input as its own label: an encoder $p(z \mid x)$, a fixed prior $r(z)$, and in place of the classifier a decoder that reconstructs x from z. Its loss is the reconstruction error plus a weight times the KL term. That KL penalty is therefore an upper bound on a rate. It limits the nats that the latent code keeps about its input, and the reconstruction error is the distortion. Only the placement of the weight differs: in Equation 37.65 the multiplier sits on the term to be kept and in an autoencoder on the rate, so a large weight there corresponds to a small β here.

The decoder's loss and the KL term to a prior are the only quantities of this section that can be computed from samples, a lower bound on one mutual information and an upper bound on another. Every other number used the joint density, which nobody has for images or text. Section 37.7 asks how much samples alone can show.

37.6.5 Summary

Mutual information is the KL divergence of a joint distribution from the product of its marginals, $I(X;Y) = D_{\mathrm{KL}}(P_{X,Y} \,\|\, P_X \otimes P_Y)$; it is zero only at independence and unchanged by invertible maps of either variable. Unlike correlation, it sees nonlinear dependence: 1.14 nats between X and $Y = X^2 + 0.3\epsilon$, where the correlation is zero. Processing cannot increase it, although a layer can make the information easier to read. An error rate requires a minimum of it (Fano's inequality), and the cross-entropy loss L of any model certifies $I(X;Y) \geq H(Y) - L$. Channel capacity and the rate–distortion function are its maximum over inputs and its minimum over encoders. The information bottleneck $\min I(X;Z) - \beta I(Y;Z)$ is a rate–distortion problem whose distortion is supplied by the label: for a Gaussian pair it keeps nothing until β exceeds $1/\rho^2$, and in its computable form the rate is bounded by the KL penalty of a variational autoencoder.

37.6.6 Exercises

1. **Gaussian pointwise mutual information.** Let (X, Y) be the Gaussian pair of Section 37.6.1.3 with standard normal marginals and correlation ρ.
   1. Show that $\mathrm{pmi}(x, y) = -\frac{1}{2}\log(1-\rho^2) - \frac{\rho^2x^2 - 2\rho xy + \rho^2y^2}{2(1-\rho^2)}$.
   2. Average it over $P_{X,Y}$ and recover Equation 37.55 without the entropy identities. Use $\mathbb{E}[X^2] = \mathbb{E}[Y^2] = 1$ and $\mathbb{E}[XY] = \rho$.
   3. Average it over $P_X \otimes P_Y$ instead, where $\mathbb{E}[XY] = 0$. Show that the result is $-\frac{1}{2}\log(1-\rho^2) - \rho^2/(1-\rho^2) = -D_{\mathrm{KL}}(P_X \otimes P_Y \,\|\, P_{X,Y})$, and evaluate the reverse divergence at $\rho = 0.99$. Compare it with $I(X;Y)$.
   4. Let (X_i, Y_i), $i = 1, \ldots, m$, be independent copies of the pair. Show from the KL form that $I(\mathbf{X};\mathbf{Y}) = \sum_i I(X_i;Y_i) = -\frac{m}{2}\log(1-\rho^2)$, and evaluate it for $\rho = 0.9$ and $m = 16$.
2. code **Rare pairs and PMI.** Pointwise mutual information Equation 37.56 corrects co-occurrence counts for the frequencies of the two words (Section 37.6.1.4).
   1. Show that $\mathrm{pmi}(x, y) \leq \min\{-\log p_X(x), -\log p_Y(y)\}$. When is each of the two bounds attained?
   2. Extend `counts` by a fourth first word *sri* and a fourth second word *lanka* that occur only in the bigram "sri lanka", with count 2. Recompute the pmi of "new york", "machine learning", "the day", and "sri lanka", and repeat with count 1. Zero counts give pmi $= -\infty$ and make `(p * pmi).sum()` return `nan`, so restrict sums to the nonzero cells, as `mutual_information` does.
   3. Compare the contributions $p_{X,Y}(x, y)\,\mathrm{pmi}(x, y)$ of "new york" and "sri lanka" to $I(X;Y)$. Then add 1 to every count and recompute both pmi values. Which of these scores would you use to rank collocations, and why?
3. **Information versus usable information.** A label Y is a fair bit. Given $Y = 0$ the input $X \in \mathbb{R}^2$ is uniform on the circle of radius 1, and given $Y = 1$ it is uniform on the circle of radius 2. A representation computes $Z = \|X\|^2$.

1. Show that $I(Y;Z) = I(Y;X) = \ln 2$, although $x \mapsto \|x\|^2$ is not invertible. Which condition for equality in the data-processing inequality of Section 37.6.2.2 holds?
2. Show that no linear classifier $\hat{y} = \mathbf{1}(\mathbf{w}^\top \mathbf{x} + b > 0)$ classifies every input correctly, whereas a threshold on Z does.
3. The layers of a network form a Markov chain $X \to Z_1 \to Z_2 \to \cdots$. Show that the error of the best classifier of any kind that reads Z_ℓ cannot fall as ℓ grows. The accuracy of a linear classifier fit to Z_ℓ often rises with ℓ all the same. Why is there no contradiction?
4. Xu et al. (2020) measure the information that a family $\mathcal{V}$ of predictors $q(y \mid x)$ can extract. Let $H_{\mathcal{V}}(Y \mid X) = \inf_{q \in \mathcal{V}} \mathbb{E}[-\log q(Y \mid X)]$, and let $H_{\mathcal{V}}(Y)$ be the same infimum over the predictors in $\mathcal{V}$ that ignore their input. The *usable information* is

$$I_{\mathcal{V}}(X \to Y) = H_{\mathcal{V}}(Y) - H_{\mathcal{V}}(Y \mid X).$$

   Let $\mathcal{V}$ be logistic regression on the given input. Show that $I_{\mathcal{V}}(X \to Y) = 0$ and $I_{\mathcal{V}}(Z \to Y) = \ln 2$. For X, use that the expected logistic loss is convex in $(\mathbf{w}, b)$ and unchanged when $\mathbf{w}$ is rotated. Which inequality of Section 37.6.2.2 does $I_{\mathcal{V}}$ violate?

4. **Synergy and the Venn diagram.** Let X and Y be independent fair bits and $Z = X \oplus Y$ their exclusive or, for which the section computed $I(X;Y) = 0$ and $I(X;Y \mid Z) = \ln 2$.
   1. Which of the Markov chains $X \to Z \to Y$, $X \to Y \to Z$, and $Y \to X \to Z$ hold? What does the data-processing inequality therefore say about this triple?
   2. In a three-variable version of Figure 37.13, the area common to all three disks would be the *interaction information* $I(X;Y;Z) = I(X;Y) - I(X;Y \mid Z)$. Prove that $I(X;Y;Z) = H(X) + H(Y) + H(Z) - H(X,Y) - H(X,Z) - H(Y,Z) + H(X,Y,Z)$, and conclude that $I(X;Y;Z)$ is symmetric in its three arguments.
   3. Evaluate $I(X;Y;Z)$ for the triple of (a) and for three copies $X = Y = Z$ of one fair bit. What do the two values imply for a three-disk diagram of entropies?

   Adapted from Thomas M. Cover and Joy A. Thomas, Elements of Information Theory, 2nd ed., Exercise 2.25, and David J. C. MacKay, Information Theory, Inference, and Learning Algorithms, Exercise 8.8.

5. code **Data processing from f-divergences.** For any channel κ that maps an input to a random output, and any two distributions P and Q of the input, the output distributions satisfy $D_f(P\kappa \,\|\, Q\kappa) \le D_f(P \,\|\, Q)$ for every f-divergence (Section 37.3.3.3). The KL divergence and $\chi^2(P\|Q) = \sum_x p(x)^2/q(x) - 1$ are two of them.
   1. Let $X \to Y \to Z$ be a Markov chain. Take $P = P_{X,Y}$, $Q = P_X \otimes P_Y$, and the channel κ that maps (x,y) to (x,z) with $z \sim p(z \mid y)$. Show that $P\kappa = P_{X,Z}$ and $Q\kappa = P_X \otimes P_Z$, and conclude that $I(X;Z) \le I(X;Y)$.
   2. Which part of the section's data-processing proposition does this route not deliver?

3. For the two flips in series, with probabilities 0.1 and 0.2, compute the KL divergence and the χ^2 divergence of each joint table from the product of its marginals, before and after the second flip.
4. Show that for a fair input through the flip channel with probability δ, $\chi^2(P_{X,Y} \,\|\, P_X \otimes P_Y) = (1-2\delta)^2$, and check your numbers from (c). Conclude that a further flip with probability δ' multiplies this divergence by $(1-2\delta')^2$, here by 0.36. Is the ratio $I(X;Z)/I(X;Y)$ from
(c) the same factor?

6. **Fano's tightness.** ⊠ The proof of Equation 37.58 uses three inequalities.

1. For the flipped bit with $\hat{Y} = X$, verify that all three are equalities: the error event W is independent of $\hat{Y}$ (the error probability is δ for both values of X), given an error the label is uniform over the $k-1=1$ wrong values, and $Y \to \hat{Y} \to X$ is a Markov chain.
2. Show that Fano's inequality is an equality if and only if all three conditions hold: the error event is independent of the estimate, given an error and the estimate the label is uniform over the $k-1$ wrong values, and $\hat{Y}$ is a sufficient statistic of X for Y.
3. Let Y be uniform on $\{0,1,2\}$, let $X = Y$, and let $g(0)=0$, $g(1)=1$, $g(2)=0$. Compute both sides of Equation 37.58 and say which of the three conditions fail.

7. **Fano and a converse.** ⊠ Fano's inequality Equation 37.58 bounds from below the error of any estimate $\hat{Y} = g(X)$ of a label Y with k values.

1. Let $\hat{Y}(x) = \arg\max_y p(y \mid x)$ be the most probable label. Show that its error satisfies $P_e \le 1 - e^{-H(Y|X)}$. Bound $\max_y p(y \mid x)$ from below by $\sum_y p(y \mid x)^2 = \mathbb{E}[p(Y \mid X) \mid X = x]$, then apply Jensen's inequality once, to $\mathbb{E}[p(Y \mid X)] = \mathbb{E}\big[e^{\log p(Y|X)}\big]$.
2. An observation X carries $I(X;Y) = 9$ nats about a balanced $10{,}000$-class label Y. Use Equation 37.58 and (a) to bracket the error of the best classifier. The inequality Equation 37.58 has no closed-form solution for P_e, so solve it numerically. What does the width of this interval say about predicting a downstream error rate from a mutual information?

8. **The loss certificate and Fano.** A classifier of $k = 10$ balanced classes has test error $P_e = 0.08$ and test loss $L = 0.30$ nats.

1. How many nats of $I(X;Y)$ does Fano's inequality Equation 37.58 certify, and how many does the loss certificate Equation 37.59?
2. For any estimate $\hat{Y} = g(X)$ with error probability P_e, let the model $q(y \mid x)$ put probability $1-P_e$ on $y = g(x)$ and spread P_e evenly over the other $k-1$ labels. Show that its cross-entropy loss is $H_b(P_e) + P_e \log(k-1)$, and conclude that Fano's inequality is the loss certificate of this model. Which of the four models evaluated in Section 37.6.2.4 is it?
3. The classifier of (a) reports a smaller loss than the model that (b) builds from its decisions. What must its confidences do for that to happen? Give a pair (X,Y) with a binary label and a model with error rate P_e that certifies more than Fano's inequality does.

9. **Binary channels.** ⊠ Work in nats with the binary entropy H_b.

1. A *Z-channel* passes a 0 unchanged and turns a 1 into a 0 with probability $\frac{1}{2}$. Compute $I(X;Y) = H(Y) - H(Y \mid X)$ as a function of $a = P(X=1)$, and show that it is largest at $a = \frac{2}{5}$, where it equals $\ln \frac{5}{4}$ nats. Why does the best input send the unreliable symbol less than half the time, and why not never?
2. A feature takes the values ± 1 and passes through dropout with rate α, which replaces it by 0 with probability α and rescales it otherwise. Show that this *binary erasure channel* has capacity $(1-\alpha)\ln 2$ nats.
3. Find the flip probability δ at which the binary symmetric channel has the same capacity as dropout with $\alpha = 0.5$. Why does a flipped bit cost more capacity than an erased one?

10. **The Gaussian channel.** Let $Y = X + N$ with noise $N \sim \mathcal{N}(0, \sigma_N^2)$ independent of X, and let the input obey the power constraint $\mathbb{E}[X^2] \le P$.
    1. Show that $I(X;Y) = h(Y) - h(N)$ for any input with a density.
    2. Among densities with variance s^2, the Gaussian has the largest differential entropy, $\frac{1}{2}\log(2\pi e s^2)$. Use this to show $h(Y) \le \frac{1}{2}\log\big(2\pi e(P+\sigma_N^2)\big)$, and conclude that the capacity is $C = \frac{1}{2}\log(1 + P/\sigma_N^2)$, attained by a Gaussian input.
    3. Evaluate C at signal-to-noise ratios $P/\sigma_N^2 = 1, 10, 100$, in nats and in bits. Show that at high signal-to-noise ratio each doubling of P adds about $\frac{1}{2}\ln 2$ nats, half a bit.
    4. A diffusion model replaces a coordinate X of the data, scaled so that $\mathbb{E}[X^2] = 1$, by $\sqrt{a}\,X + \sqrt{1-a}\,\epsilon$ with $0 < a < 1$ and independent $\epsilon \sim \mathcal{N}(0,1)$. Show that the noised coordinate carries at most $\frac{1}{2}\log\frac{1}{1-a}$ nats about X, whatever the distribution of X, and evaluate the bound at $a = 0.5$ and $a = 0.01$. For which data is it attained?

11. code **Scalar quantization and rate–distortion.** ⊠ A uniform scalar quantizer with step Δ replaces X by the nearest multiple of Δ, the center of its bin, and an entropy coder then spends about $H(X_\Delta)$ nats per sample on the index X_Δ of the bin. For a continuous density, $H(X_\Delta) + \log\Delta \to h(X)$ as $\Delta \to 0$ (Section 37.1); use this fact.
    1. Let $X \sim \mathcal{N}(0, \sigma_X^2)$. For small Δ the mean squared error is approximately $\Delta^2/12$. Combine this approximation with the limit above to show that the rate exceeds the Gaussian rate–distortion function Equation 37.63 at the same distortion by $\frac{1}{2}\ln(2\pi e/12)$ nats (≈ 0.255 bits) per sample, independently of σ_X and Δ.
    2. Check the gap numerically for $\sigma_X = 1$ and $\Delta = 2, 1, 0.5, 0.25, 0.1$, computing the bin probabilities and the distortion by numerical integration. Part (a) uses two approximations, $D \approx \Delta^2/12$ and $H(X_\Delta) \approx h(X) - \log\Delta$. Which of them accounts for the deviation of the gap from its limit at $\Delta = 1$ and $\Delta = 0.5$?
    3. A learned codec rounds each coordinate of its latent vector to the nearest integer and entropy-codes the result. What does the gap of
       (a) imply for such a codec at high rate? Which kind of quantizer can close the gap?

12. **The rate bound of a variational autoencoder.** Let $X \sim \mathcal{N}(0,1)$, let the encoder be $Z = X + \sigma\epsilon$ as in Section 37.6.4.2, so that $p(z \mid x) = \mathcal{N}(z; x, \sigma^2)$, and let the prior be $r = \mathcal{N}(0,1)$. Use $D_{\mathrm{KL}}\big(\mathcal{N}(\mu, s^2) \,\|\, \mathcal{N}(0,1)\big) = \frac{1}{2}(\mu^2 + s^2 - 1 - \log s^2)$.
    1. Compute the three terms of Equation 37.69 and check the identity. By how many nats does the KL penalty overstate the rate at $\sigma = 1$?

2. Rescale the code to $Z' = Z/\sqrt{1+\sigma^2}$. Show that $I(X;Z') = I(X;Z)$ and that the bound is now an equality.
3. The rate does not depend on the units of the code, and the bound does. Which prior r makes the bound an equality for a given encoder? What does minimizing the KL penalty therefore do to the code, besides lowering its rate?

37.7 Estimating Mutual Information

Every mutual information computed so far came out of a formula for the joint density. Real pairs do not bring one along. We are handed a list, perhaps of spoken sentences with their transcripts or of queries with the documents that answered them, and from the list alone we want to say how much one member of a pair tells us about the other. Representation learning goes a step further and trains models to make that amount large, so the estimate becomes the training objective.

A widely used loss makes the question concrete. A contrastive model such as SimCLR or CLIP is trained on pairs, two crops of one photograph or an image and its caption, which are called two *views* of one example. It learns to pick the true partner of a view out of a batch of candidates, say 128 of them, and reports a cross-entropy loss, the InfoNCE loss. Papers often subtract that loss from $\ln 128 = 4.85$ and quote the difference as the information that the two views share. That number can never exceed 4.85 nats, however much the views share, because a model that always picks the right partner has shown only that it can tell 128 candidates apart. Since models are trained on such numbers and papers read them as measurements, we need to tell a measurement from a bound, and the slack of a bound from the error of a model.

There are two routes. The direct one counts samples, in the cells of a histogram or among the neighbors of each point. It works for a small amount of information between two numbers and fails for vectors. The other replaces the unknown densities by a model, built so that the result is a lower bound on the information, and maximizes the bound. Two results matter most: the limit on what any estimator can guarantee without assumptions (Section 37.7.2.4), and the bound behind the InfoNCE loss, with its ceiling (Section 37.7.4).

```
from scipy.spatial.distance import cdist
from scipy.special import digamma, logsumexp
```

37.7.1 A Pair with Known Information

An estimator can be judged only where the answer is known. We test every one on the standard Gaussian pair of Section 37.6.1.3 with correlation ρ, which shares

$$I(X;Y) = -\tfrac{1}{2}\log\left(1-\rho^2\right) \tag{37.70}$$

nats: 0.83 at $\rho = 0.9$ and 1.96 at $\rho = 0.99$. Logarithms are natural throughout.

The bounds of this section are built from the *pointwise mutual information* (Section 37.6.1.4), $\operatorname{pmi}(x,y) = \log \frac{p(x,y)}{p(x)\,p(y)}$, which says how much more likely a pair is under the joint distribution than under independence. For the Gaussian pair it is a quadratic:

$$\operatorname{pmi}(x,y) = -\tfrac{1}{2}\log(1-\rho^2) - \frac{x^2 - 2\rho xy + y^2}{2(1-\rho^2)} + \frac{x^2+y^2}{2}. \tag{37.71}$$

Its average over pairs drawn from the joint distribution is $I(X;Y)$. The average of its exponential over *independent* x and y is 1, because $p(x)\,p(y)\,e^{\operatorname{pmi}(x,y)} = p(x,y)$ integrates to one. The cell checks both facts on samples. For the second it pairs each x with the y of the neighboring sample, which destroys the dependence. We call these *mismatched pairs*. Each of them is a draw of independent x and y, and they will stand in for such draws wherever a bound needs them.

```
def gauss_pair(rng, rho, n):
    """Samples of the standard Gaussian pair; n is a count or a shape."""
    x = rng.standard_normal(n)
    y = rho * x + np.sqrt(1 - rho**2) * rng.standard_normal(n)
    return x, y

def mi_gauss(rho):
    """I(X;Y) of the pair, in nats."""
    return -0.5 * np.log(1 - rho**2)

def pmi_gauss(x, y, rho):
    """Pointwise mutual information pmi(x, y) of the pair, in nats."""
    c = 1 - rho**2
    return (-0.5 * np.log(c) - (x**2 - 2 * rho * x * y + y**2) / (2 * c)
            + (x**2 + y**2) / 2)

rng = np.random.default_rng()
x, y = gauss_pair(rng, 0.9, 200_000)
print(f'I(X;Y) = {mi_gauss(0.9):.3f} nats, mean pmi over pairs = '
      f'{pmi_gauss(x, y, 0.9).mean():.3f}')
print(f'mean exp(pmi) over independent pairs = '
      f'{np.exp(pmi_gauss(x, np.roll(y, 1), 0.9)).mean():.3f}')
```

```
I(X;Y) = 0.830 nats, mean pmi over pairs = 0.833
mean exp(pmi) over independent pairs = 0.994
```

Both averages agree with their targets to about a hundredth. The bounds of this section rest on these two identities, and the second will cause most of the trouble. At $\rho = 0.9$ it is an average of tame numbers: the variance of e^{pmi} over independent pairs is $e^{2I} - 1$ (Exercise 2), which is 4.3 here. At $I = 6$ the variance is $160{,}000$, and the same $200{,}000$ samples would leave the average uncertain by about 0.9.

37.7.2 Nonparametric Estimation

The obvious way to estimate a probability is to count the samples that fall into a region. What remains to choose is the regions, and we begin with a grid fixed in advance.

37.7.2.1 Histograms

The simplest estimator cuts the range of each variable into b intervals of equal width, counts the samples in each of the $b \times b$ cells, and applies the discrete definition of mutual informa-

tion to the table of frequencies. It makes two errors, and they have opposite signs.

The first is a loss. The bin of a value is a function of the value, and by the data-processing inequality (Section 37.6.2.2) a function of X shares with Y at most what X shares. Binning can only remove information, and it removes a lot when the dependence is finer than a bin. At $\rho = 0.99$ the standard deviation of y given x is $\sqrt{1-\rho^2} = 0.14$, while ten bins across a sample that reaches from about -4.5 to 4.5 are 0.9 wide, six times as much.

The second is a gain that is not there. The estimate is the KL divergence between the table of frequencies and the product of its row and column sums. It is therefore never negative, and it is zero only if the table factorizes exactly, which the fluctuating counts of an independent pair never do. Sampling noise reads as dependence, the more so the fewer samples a cell holds.

Besides its two errors the histogram has a ceiling. Its estimate cannot exceed the entropy of the binned x, and b bins hold at most $\log b$ nats: 2.30 for ten bins, whatever the pair shares. We will meet such a ceiling in every estimator of the section. None of them can report more than the logarithm of the number of things it can tell apart.

The cell implements the estimator as `binned_mi` and applies it to $200{,}000$ samples of four pairs with $b = 10$, 40 and 160: an independent pair, the Gaussian pair at $\rho = 0.9$ and at $\rho = 0.99$, and the pair $Y = X^2 + 0.3\,\epsilon$ of Section 37.6.1.5, which has zero correlation and shares 1.141 nats.

```
def binned_mi(x, y, bins):
    """Plug-in MI of the bins x bins histogram of (x, y), in nats."""
    p = np.histogram2d(x, y, bins=bins)[0] / len(x)       # cell frequencies
    q = p.sum(axis=1, keepdims=True) * p.sum(axis=0, keepdims=True)
    nz = p > 0                                            # occupied cells
    return float((p[nz] * np.log(p[nz] / q[nz])).sum())

mi_parabola = 1.141  # by quadrature, sec_mdl-mi-nonlinear-correlation
n = 200_000
x = rng.standard_normal(n)
pairs = {'rho = 0': (gauss_pair(rng, 0.0, n), 0.0),
         'rho = 0.9': (gauss_pair(rng, 0.9, n), mi_gauss(0.9)),
         'rho = 0.99': (gauss_pair(rng, 0.99, n), mi_gauss(0.99)),
         'parabola': ((x, x**2 + 0.3 * rng.standard_normal(n)), mi_parabola)}
print('pair        truth  b = 10  b = 40  b = 160')
for name, ((u, v), truth) in pairs.items():
    print(f'{name:10}  {truth:5.3f}'
          + ''.join(f'  {binned_mi(u, v, b):6.3f}' for b in (10, 40, 160)))
```

```
pair        truth  b = 10  b = 40  b = 160
rho = 0     0.000   0.000   0.003   0.035
rho = 0.9   0.830   0.629   0.813   0.845
rho = 0.99  1.959   1.090   1.800   1.955
parabola    1.141   0.551   0.989   1.136
```

Both errors are in the table. With $b = 10$ every dependent pair reads low: about 0.6 for 0.83 nats at $\rho = 0.9$, and roughly half of the truth at $\rho = 0.99$ and for the pair on the parabola. With $b = 160$ the independent pair is credited with a few hundredths of a nat, and the pair at $\rho = 0.9$ reads above its true value, which binning alone could never bring about. The entry for $\rho = 0.99$ at $b = 160$ lands within about 0.01 of the truth, but only because a loss

and a gain partly cancel there; the row above shows the same bins overshooting. With the truth unknown, nothing in the sample says which column to believe.

A third weakness concerns the coordinates. An invertible map of either variable leaves the mutual information unchanged (Section 37.6.1.5), and it moves the bins. Replace x by x^3 in the pair at $\rho = 0.9$. The largest of $200{,}000$ standard normal values is about 4.5 in magnitude, so x^3 ranges over about ± 90 and each of 40 bins is about 4.5 wide: roughly nine samples in ten land in the two or three bins around zero. A remedy is to bin the *ranks*, the positions of the values in the sorted sample. An increasing map does not change them, and every bin of ranks holds the same number of samples.

```
x, y = gauss_pair(rng, 0.9, n)
rank = lambda a: np.argsort(np.argsort(a))      # position in sorted order
print(f'true I = {mi_gauss(0.9):.3f} nats; histograms with b = 40:')
for name, u in (('x', x), ('x**3', x**3)):
    print(f'{name:4}: correlation {np.corrcoef(u, y)[0, 1]:.3f}, '
          f'raw {binned_mi(u, y, 40):.3f}, '
          f'ranks {binned_mi(rank(u), rank(y), 40):.3f} nats')
```

```
true I = 0.830 nats; histograms with b = 40:
x   : correlation 0.901, raw 0.819, ranks 0.819 nats
x**3: correlation 0.699, raw 0.439, ranks 0.819 nats
```

The information is still 0.830 nats, yet the cube lowers the correlation from 0.90 to about 0.7 and cuts the raw estimate to between 0.3 and 0.5 nats. Where in that range a sample lands depends on its few largest values, which set the width of the bins. The estimate from ranks is the same for x and for x^3, digit for digit, and within 0.02 of the truth.

Ranks repair the coordinates. Nothing repairs the dimension. A pair of five-dimensional vectors needs a histogram with ten axes, and ten bins on each give 10^{10} cells, nearly all of them empty for any sample we can collect. An estimator for vectors must let the data set the resolution, with small cells where samples are dense and large cells where they are sparse.

37.7.2.2 The KSG Estimator

The estimator of Kraskov et al. (2004), KSG for short, replaces fixed cells by neighborhoods of the sample points. It is the standard tool for a pair of low-dimensional variables. This subsection and the next can be skipped on a first reading.

Distances are measured by the largest coordinate difference (the Chebyshev distance), so that a ball is a square in the plane and a box in higher dimension. Around sample i draw the smallest box in the joint (x, y) space that reaches its m-th nearest neighbor, and call its half-width r_i. The box holds the fraction m/n of the n samples, an estimate of the joint probability near the point.

The two marginal probabilities are counted at the same scale. The strip $|x - x_i| < r_i$ holds $n_x^{(i)}$ other samples and the strip $|y - y_i| < r_i$ holds $n_y^{(i)}$. The shared half-width is what lets counts stand for densities. If the densities are nearly constant over the box, the box has probability about $p(x_i, y_i)\,(2r_i)^2$ and the strips about $p(x_i)\,2r_i$ and $p(y_i)\,2r_i$, so the widths cancel when the fraction in the box is divided by the product of the fractions in the strips. The logarithm of that ratio is a pointwise mutual information computed by counting. Figure 37.18 shows the three counts at one point.

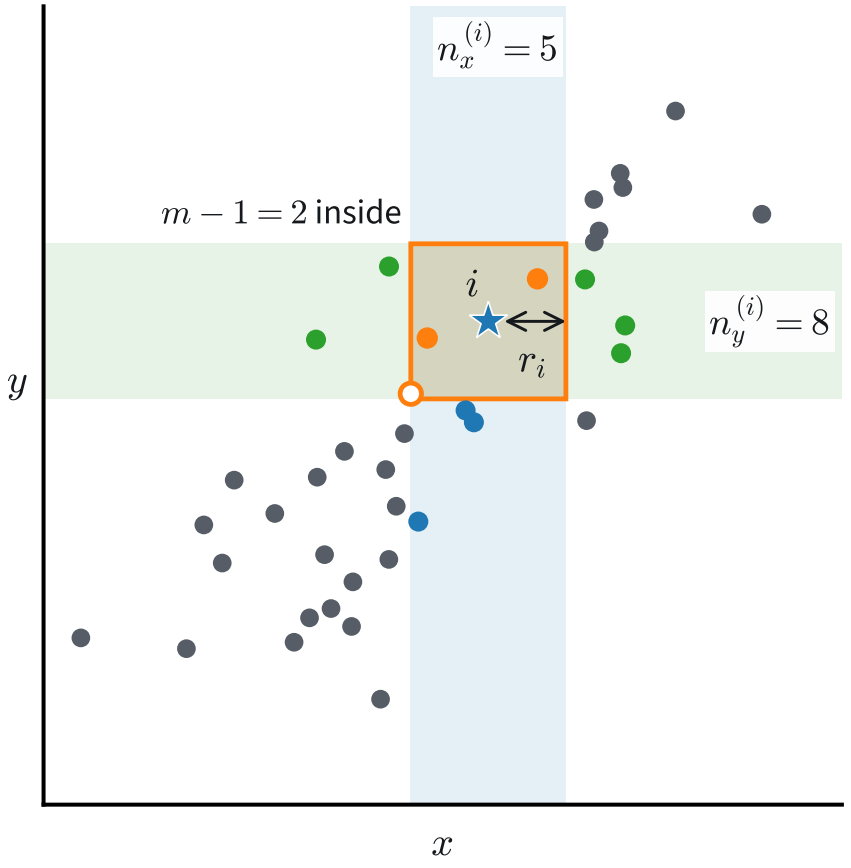


Figure 37.18: The KSG count at one sample point i (the star), with $m = 3$. The box of half-width r_i reaches the third nearest neighbor in the joint plane (the ring on its left edge; it lies in the horizontal strip and counts toward $n_y^{(i)}$ only); two neighbors lie strictly inside. The vertical strip of the same half-width holds $n_x^{(i)}$ other samples and the horizontal strip holds $n_y^{(i)}$ (blue and green dots lie in one strip only). The summand of the estimator is approximately $\log \frac{m/n}{(n_x/n)(n_y/n)}$, a pointwise mutual information from counts.

Averaging these local log ratios over the sample estimates $I(X;Y)$. One correction is needed, because the logarithm of a fraction of a handful of points is a biased estimate of a log probability. The probability mass p of the box that reaches the m-th of $n-1$ neighbors has $\mathbb{E}[\log p] = \psi(m) - \psi(n)$, not $\log(m/n)$, where $\psi = (\log\Gamma)'$ is the digamma function. We do not derive this. With ψ in place of the logarithm the estimator reads

$$\hat{I}_{\text{KSG}} = \psi(m) + \psi(n) - \frac{1}{n}\sum_{i=1}^{n}\left(\psi\left(n_x^{(i)}+1\right) + \psi\left(n_y^{(i)}+1\right)\right). \tag{37.72}$$

Since $\psi(j) \approx \log j$ for large j, each summand is approximately $\log \frac{m\,n}{n_x^{(i)}\,n_y^{(i)}}$, the local log ratio of the box fraction to the product of the strip fractions. The $+1$ puts the strips on the footing of the box: $\psi(m)$ belongs to a box whose boundary passes through its m-th neighbor, and a strip with $n_x^{(i)}$ samples strictly inside is treated as if its boundary passed through the next one. Both strip counts use the strict inequality and exclude the point itself, so a neighbor on the boundary of a strip is not counted. The radii r_i adapt to the data, and there is no bin width or bandwidth to choose.

The implementation counts the neighbors of all points at once from the $n \times n$ matrices of Chebyshev distances in each space (`cdist`). The joint distance is their elementwise maximum.

```
def ksg_counts(x, y, m=5):
    """Strip counts n_x, n_y of every sample point; x, y are (n, d)."""
    dx, dy = cdist(x, x, 'chebyshev'), cdist(y, y, 'chebyshev')
    d = np.maximum(dx, dy)                       # joint Chebyshev distance
    np.fill_diagonal(d, np.inf)
    r = np.partition(d, m - 1, axis=1)[:, m - 1]  # half-width: m-th neighbor
    n_x = (dx < r[:, None]).sum(axis=1) - 1      # strict, self excluded
    n_y = (dy < r[:, None]).sum(axis=1) - 1
    return n_x, n_y

def ksg_mi(x, y, m=5):
    """KSG nearest-neighbor estimate of I(X;Y) in nats; x, y are (n, d)."""
    n_x, n_y = ksg_counts(x, y, m)
    return float(digamma(m) + digamma(len(x))
                 - (digamma(n_x + 1) + digamma(n_y + 1)).mean())
```

Each summand of Equation 37.72 involves only counts, and counting alone bounds how large the estimate can be.

Proposition (sample-size ceiling of KSG). *For any sample of n points in which no two joint distances from the same point are equal,*

$$\hat{I}_{\text{KSG}} \le \psi(n) - \psi(m), \tag{37.73}$$

with equality when $Y = X$.

Proof. Without ties, exactly $m-1$ samples lie strictly closer to point i than its m-th neighbor. Each of them satisfies $|x_j - x_i| < r_i$ and $|y_j - y_i| < r_i$, because the joint distance is the larger of the two coordinate distances, so it lies in both strips: $n_x^{(i)} \ge m-1$ and $n_y^{(i)} \ge m-1$.

The digamma function is increasing, so each summand is at most $\psi(m)+\psi(n)-2\psi(m) = \psi(n)-\psi(m)$. For $Y = X$ the three distances coincide, both strips hold exactly the $m-1$ points inside the box, and every summand attains the bound. ■

At $n = 2{,}000$ and $m = 5$ the ceiling is $\psi(2000) - \psi(5) = 6.09$ nats, close to $\log(n/m) = 5.99$. A pair with infinite information, $Y = X$ for continuous X, is reported as 6.1 nats, and doubling the sample raises the ceiling by only about $\log 2$. This is the histogram's $\log b$ again, with neighborhoods in place of bins: counting neighbors can report only as much information as the sample has distinguishable neighborhoods, about n/m of them.

37.7.2.3 KSG in High Dimension

In more dimensions the estimator fails well below its ceiling, and for a different reason. Take d independent copies of the pair, each with $\rho = 0.9$, so that X and Y are d-dimensional. Information adds over independent coordinates, because the log ratio is a sum of d terms, so the truth is $d \times 0.83$ nats.

The estimator rests on densities that are nearly constant over each box, and in high dimension the boxes are too large for that. With $n = 2{,}000$ and $m = 5$ a box holds the fraction $m/n = 0.0025$ of the sample. Were the sample spread evenly over its $2d$ dimensions, such a box would have sides of $(m/n)^{1/(2d)}$ times the width of the data: 0.05 at $d = 1$, but 0.47 at $d = 4$ and 0.83 at $d = 16$. The dependence lives in a much narrower band along $y = \rho x$. Given x, each coordinate of y has standard deviation $\sqrt{1-\rho^2} = 0.44$, about 0.07 of the width of a sample that reaches from about -3.3 to 3.3. A box several times wider than the band fails as a coarse histogram cell does, and the estimate should fall behind the truth as d grows. The cell runs KSG on $2{,}000$ samples for d from 1 to 16 and measures by how much. It prints the average strip count of each row, and it runs $Y = X$, which attains the ceiling.

```
n, rho = 2000, 0.9
print(' d  true I    KSG  mean strip count')
for dim in (1, 2, 4, 8, 16):
    x, y = gauss_pair(rng, rho, (n, dim))        # dim independent pairs
    n_x, n_y = ksg_counts(x, y)
    print(f'{dim:2d}  {dim * mi_gauss(rho):6.2f}  {ksg_mi(x, y):5.2f}  '
          f'{(n_x.mean() + n_y.mean()) / 2:16.1f}')
x = rng.standard_normal((n, 1))
print(f'Y = X:  KSG = {ksg_mi(x, x):.2f}, '
      f'ceiling psi(n) - psi(m) = {digamma(n) - digamma(5):.2f} nats')
```

```
 d  true I    KSG  mean strip count
 1    0.83   0.83              71.3
 2    1.66   1.60              50.0
 4    3.32   2.51              31.3
 8    6.64   2.85              26.0
16   13.29   2.75              26.8
Y = X:  KSG = 6.09, ceiling psi(n) - psi(m) = 6.09 nats
```

In one and two dimensions KSG lands within about 0.1 nat of the truth. At $d = 4$ it reports about 2.5 for 3.3, almost a nat low. Then it stops: at $d = 8$ and at $d = 16$ it reports a little under 3 nats, although the truth doubles between these rows, to 13.3. The ceiling is not to blame, since that is less than half of the 6.1 nats printed for $Y = X$.

The strip counts show the saturation directly. Each summand is about $\log \frac{m\,n}{n_x n_y}$, so the estimate grows only if the strips empty out. They hold about 70 of the $2{,}000$ points at $d = 1$ and about 31 at $d = 4$, and then stay at about 26. A box as local as the one at $d = 1$ would need $m/n = 0.05^{32}$ at $d = 16$, a sample of some 10^{42} points. The sample that keeps neighborhoods small grows exponentially with the dimension.

37.7.2.4 Limits of Distribution-Free Estimation

Perhaps a cleverer estimator does better. Ask only for a lower bound: a function B of n samples such that $B \le I(X;Y)$ with probability at least 0.95 *for every distribution* of the pair. (Any level above one half would serve.) Lower bounds are the useful direction, because a representation learner wants to be sure that two views share a lot.

The smallest case shows the obstacle. Let $Y = X$ be a copy of a discrete variable, so that $I(X;Y) = H(X)$. An estimator that treats the symbols as labels without meaning cannot use which values it saw. It can use only the pattern of repetitions: how many distinct values occur once in the sample, how many occur twice, and so on. This pattern is the *type* of the sample, and we suppose that B is a function of it. Four steps give the limit, and the third is the one to watch.

1. Let X be uniform on M symbols and draw n samples. Two given samples coincide with probability $1/M$, so by the union bound some two of the $n(n-1)/2$ pairs coincide with probability at most $n(n-1)/(2M)$.
2. Apply the guarantee to the uniform distribution on $M = n^2$ symbols, whose entropy is $2\log n$: on a sample from *that* distribution, $B \le 2\log n$ with probability at least 0.95.
3. By the first step, a sample from that distribution consists of n distinct values with probability more than $\frac{1}{2}$. Two events whose probabilities add to more than 1 overlap, so $B \le 2\log n$ on some all-distinct sample.
4. All samples of n distinct values have the same type, so B reports at most $2\log n$ on every one of them, whatever source produced it.

Now take $M = 2^{256}$ symbols, entropy $256 \ln 2 = 177$ nats. At $n = 10^6$ the sample is all distinct with probability at least $1 - 4.3 \times 10^{-66}$, and on that sample B reports at most $2\log 10^6 = 27.6$ nats. A million samples cannot guarantee 177 nats, and they cannot guarantee 30. Exercise 6 sharpens the constant and asks what sample would suffice.

McAllester and Stratos (2020) proved the general statement, without the assumption on types:

> **Theorem (McAllester–Stratos, informal).** Any distribution-free lower bound on $I(X;Y)$ that holds with fixed confidence from n samples is $O(\log n)$.

The invariance of mutual information is part of the reason. An invertible relabelling of either variable leaves the information unchanged (Section 37.6.1.5), so an estimator that assumes nothing about the coordinates must cope with every relabelling of the sample at once, and all it can rely on is the pattern of repetitions. Whoever wants more must assume something. KSG assumes that distances in the given coordinates mean something and that the density is smooth in them, and what it offers in return is an estimate without a guarantee. The bounds of Section 37.7.3 can report more than the logarithm of their sample size, and Section 37.7.4.4 shows what such a report is worth.

37.7.2.5 The Compression-Phase Debate

Both limits met so far, a histogram that depends on its bins and a sample that cannot show more than the logarithm of its size, bear on a well-known experiment. Shwartz-Ziv and Tishby (2017) took a small classifier whose inputs are the $4{,}096$ binary patterns of 12 bits, binned the activations of each layer Z during training, and plotted the estimates of $I(X;Z)$ and $I(Y;Z)$, with input X and label Y. They reported a short *fitting* phase, in which both rise, and a long *compression* phase, in which $I(X;Z)$ falls while $I(Y;Z)$ holds, and they proposed compression as the reason why deep networks generalize. Saxe et al. (2018) found the compression phase with saturating activations such as tanh and not with ReLU, whose networks generalize as well.

What had been measured? In a deterministic network the binned layer Z_{bin} is a function of the input, so $H(Z_{\text{bin}} \mid X) = 0$ and the estimate is $I(X;Z_{\text{bin}}) = H(Z_{\text{bin}})$, the entropy of the binned activations. It is at most the logarithm of the number of inputs, $\ln 4096 = 8.3$ nats in the original experiment, and it falls whenever inputs that occupied different cells come to share one, as they do when training pushes saturating units into the two edge bins. The information itself does not move. As long as different inputs give different activations, $I(X;Z) = H(X)$ for a finite set of inputs, and for continuous inputs it is infinite (Section 37.6.1.3). The curves track how the activations spread over the bins, which is a property of the estimator (Exercise 10).

37.7.3 Variational Lower Bounds

Counting cannot measure a large mutual information, so the second route asks for less. The densities in $\log \frac{p(x,y)}{p(x)\,p(y)}$ are unknown, and a variational method replaces them by an auxiliary model, chosen so that the expectation it leads to is a *lower bound* on the information whatever the model is. Training the model to make that expectation large tightens the bound and can never carry it past $I(X;Y)$. The estimation problem has become an optimization problem. Its answer carries two errors: that of the model, which may be far from the best one, and that of the sample, since a batch average has to stand in for the expectation.

Four bounds of this kind follow. The first comes from a decoder (Barber–Agakov). The next two come from a function that scores pairs and needs no normalization: the Donsker–Varadhan bound and the bound of Nguyen, Wainwright and Jordan, NWJ for short. The fourth, InfoNCE, has a section of its own. Poole et al. (2019) treat the four in one framework. All the proofs use a single tool, Gibbs' inequality (Section 37.1): a KL divergence is never negative.

37.7.3.1 The Barber–Agakov Bound

The first bound replaces the unknown posterior $p(x \mid y)$ by a model of it.

Proposition (Barber–Agakov bound). *For any conditional density $q(x \mid y)$,*

$$I(X;Y) \geq H(X) + \mathbb{E}_{(x,y)\sim P_{X,Y}} \left[\log q(x \mid y)\right], \tag{37.74}$$

with equality if and only if $q(x \mid y) = p(x \mid y)$. For continuous X the differential entropy $h(X)$ takes the place of $H(X)$.

Proof. Write $I(X;Y) = H(X) - H(X \mid Y)$ and insert the model:

$$I(X;Y) - H(X) - \mathbb{E}\left[\log q(x \mid y)\right] = \mathbb{E}\left[\log \frac{p(x \mid y)}{q(x \mid y)}\right] = \mathbb{E}_y\left[D_{\mathrm{KL}}(p(\cdot \mid y) \,\|\, q(\cdot \mid y))\right] \geq 0$$

by Gibbs' inequality, applied for each value of y and averaged. Equality holds if and only if every divergence vanishes, that is, when q is the posterior. ■

The bound (Barber and Agakov 2003) is the information-theoretic reading of an encoder–decoder: if a decoder reconstructs X from Y with low cross-entropy, the pair must share a lot of information. Its weak point is $H(X)$, which is unknown in the situations where we need the bound. It is a constant, however, so *maximizing* the bound over q never needs it. Substituting $H(X) = -\mathbb{E}\log p(x)$ gives the equivalent ratio form $I(X;Y) \geq \mathbb{E}\left[\log\left(q(x \mid y)/p(x)\right)\right]$, which the proof of the InfoNCE bound will use.

We say that a model *certifies* the value of its bound. The word refers to the inequality between expectations. An average over a batch only estimates the certified value and can land on either side of it.

A classifier is a decoder of the label. To see what it certifies, exchange the roles of the variables, which the symmetry of I allows, and take the classifier as the model $q(y \mid x)$. The bound is then the loss certificate of Section 37.6.2.4: for k balanced classes and held-out cross-entropy L, $I(X;Y) \geq \log k - L$. Since $L \geq 0$, no 10-class classifier certifies more than $\ln 10 = 2.30$ nats. The InfoNCE bound will have this shape.

The Gaussian pair shows what a poor decoder loses. A decoder trained by squared error alone is a Gaussian $q(x \mid y) = \mathcal{N}(x; ay, 1)$ with unit variance. With $h(X) = \frac{1}{2}\log(2\pi e)$ the bound becomes $\frac{1}{2} - \frac{1}{2}\mathbb{E}(X - aY)^2 = \frac{1}{2} - \frac{1}{2}(1 - 2a\rho + a^2)$, which is largest at $a = \rho$, where it equals $\rho^2/2$. At $\rho = 0.9$ this decoder certifies 0.41 nats of the true 0.83, and it never certifies more than $\frac{1}{2}$ however strong the dependence. The shortfall is the error of a decoder whose variance is fixed at 1 instead of the true conditional variance $1-\rho^2$ (Exercise 4).

37.7.3.2 The Donsker–Varadhan Bound

The second bound needs neither a normalized model nor $H(X)$. Mutual information is itself a KL divergence, from the joint distribution to the product of the marginals (Section 37.6), so it suffices to bound a KL divergence from below. The bound does so through an arbitrary function T that scores a point, a *critic* as in Section 37.3.4, and nothing in it has to integrate to one.

Proposition (Donsker–Varadhan bound). *For distributions P and Q and any function T for which $\mathbb{E}_P[T]$ is defined and $\mathbb{E}_Q[e^T] < \infty$,*

$$D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \log \mathbb{E}_Q\left[e^T\right], \tag{37.75}$$

with equality if and only if $T = \log \frac{dP}{dQ} + c$ for a constant c. When $P \ll Q$ and the divergence is finite, the supremum over T therefore attains $D_{\mathrm{KL}}(P\|Q)$, which is the Donsker–Varadhan representation of the divergence (Donsker and Varadhan 1983).

Proof. If P is not absolutely continuous with respect to Q, the left side is infinite. Otherwise, given T, tilt Q by the critic: let $\tilde{Q}$ be the *tilted distribution* with density $\tilde{q}(x) = q(x)\, e^{T(x)}/Z$ and normalizer $Z = \mathbb{E}_Q[e^T]$. Then

$$D_{\mathrm{KL}}(P\|Q) - (\mathbb{E}_P[T] - \log Z) = \mathbb{E}_P\left[\log \frac{p}{q} - T + \log Z\right] = \mathbb{E}_P\left[\log \frac{p}{\tilde{q}}\right] = D_{\mathrm{KL}}(P\|\tilde{Q}) \geq 0,$$

by Gibbs' inequality. Equality holds if and only if $P = \tilde{Q}$, which means $p/q = e^T/Z$, or $T = \log(p/q) + \log Z$. ■

The gap of the bound is the divergence from P to the tilted distribution, so a critic is good when tilting Q by e^T reproduces P. A constant added to T cancels between the two terms and changes nothing. Specializing to $P = P_{X,Y}$ and $Q = P_X \otimes P_Y$ bounds mutual information: for any critic $T(x, y)$,

$$I(X;Y) \geq \mathbb{E}_{P_{X,Y}}[T] - \log \mathbb{E}_{P_X \otimes P_Y}\left[e^T\right],$$

and the best critic is the pmi plus any constant. The first expectation is an average over the sample pairs. The second needs independent pairs, and the mismatched pairs of a batch supply them. With a neural network as the critic, this estimator is known as *MINE*, for mutual information neural estimation (Belghazi et al. 2018).

37.7.3.3 The NWJ Bound

The logarithm in front of $\mathbb{E}_Q[e^T]$ makes trouble on a minibatch. By Jensen's inequality the logarithm of a batch average is on average below the logarithm of the population average, so the batch value $\overline{T} - \log \overline{e^T}$ of the DV bound is on average too high. An estimate of a lower bound should not err upward.

The tangent-line inequality $\log z \leq z/e$, with equality at $z = e$, removes the logarithm: applied to $Z = \mathbb{E}_Q[e^T]$ in Equation 37.75, it gives, for every critic T,

$$I(X;Y) \geq \mathbb{E}_{P_{X,Y}}[T] - e^{-1}\, \mathbb{E}_{P_X \otimes P_Y}\left[e^T\right]. \tag{37.76}$$

This is the *NWJ bound* (Nguyen et al. 2010). For the same critic it is never larger than the DV bound, and the two agree only when $Z = e$. It is tight at $T^* = 1 + \mathrm{pmi}$: where DV accepts the pmi plus any constant, NWJ insists on the constant 1. Since $e^{-1}\, \mathbb{E}_Q[e^T] = \mathbb{E}_Q[e^{T-1}]$, it is the critic bound that Section 37.3.4 gives for every f-divergence, written out for the KL divergence: $D_{\mathrm{KL}}(P\|Q) \geq \mathbb{E}_P[T] - \mathbb{E}_Q[e^{T-1}]$, obtained there from the convex conjugate of $u \log u$. For this one divergence, then, DV sharpens the general bound.

No logarithm of an average appears, so the batch value of Equation 37.76 is an unbiased estimate of the bound. Unbiased is not the same as reliable. For the optimal critic, $e^{T^*-1} = e^{\mathrm{pmi}}$ is the density ratio. Its expectation over independent pairs is 1, but when X and Y are strongly dependent that expectation is carried by a few enormous values among many that are close to zero, and a batch without an enormous value reports too much. Section 37.7.4.4 works out how much, and measures it.

37.7.4 InfoNCE

37.7.4.1 The Contrastive Loss

The e^T of the DV and NWJ bounds is unnormalized, so a single pair can dominate a batch. InfoNCE normalizes it within the batch: it makes the candidates compete. Draw one *positive* pair $(x, y^+) \sim P_{X,Y}$, whose first member x we call the *anchor*, and $N-1$ *negatives* $y_2, \dots, y_N$ drawn independently from P_Y. Score every candidate with the critic $T(x, y)$, and ask a softmax over the N scores to point at the positive. The *InfoNCE loss* (Oord et al. 2018) is the resulting cross-entropy,

$$\mathcal{L}_{\text{NCE}} = -\mathbb{E}\left[\log \frac{e^{T(x,y^+)}}{e^{T(x,y^+)} + \sum_{j=2}^{N} e^{T(x,y_j)}}\right], \tag{37.77}$$

where the expectation is over the positive pair and the negatives. The associated estimate of mutual information is the logarithm of the number of candidates minus the loss,

$$\hat{I}_{\text{NCE}} = \log N - \mathcal{L}_{\text{NCE}}. \tag{37.78}$$

Some papers write the denominator as the average $\frac{1}{N}\sum_j e^{T(x,y_j)}$; the expected logarithm of the ratio is then $\hat{I}_{\text{NCE}}$ itself. The loss is an N-class cross-entropy whose classes are the other examples in the batch rather than fixed labels. It is the loss of CPC, SimCLR and CLIP (Oord et al. 2018; Chen et al. 2020; Radford et al. 2021), where x and y are two views or two modalities of one example and the batch supplies the negatives.

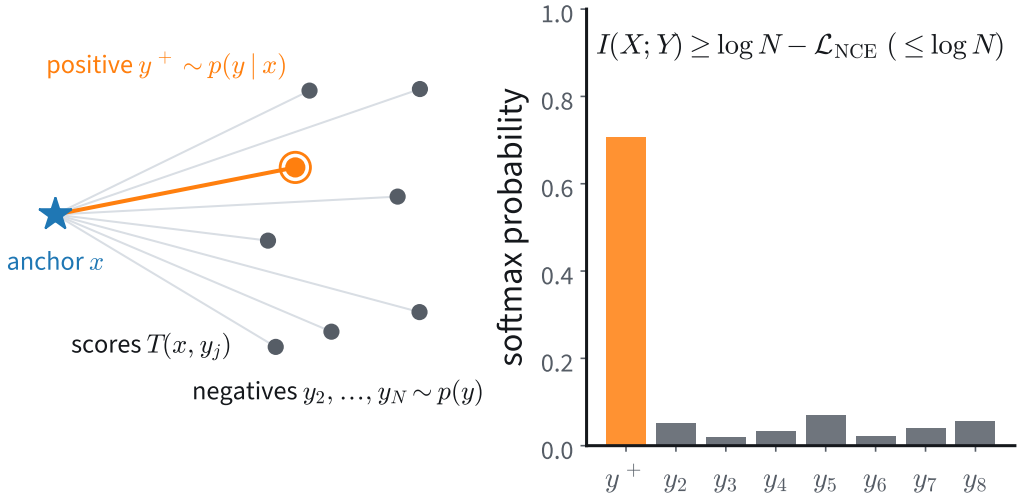


Figure 37.19: The InfoNCE task. Left: an anchor x is scored by the critic against the positive y^+, drawn jointly with it, and $N-1$ negatives drawn independently from the marginal; the rays carry the scores $T(x, y_j)$. Right: a softmax over the N scores must point at the positive. The resulting cross-entropy certifies $I(X;Y) \geq \log N - \mathcal{L}_{\text{NCE}}$, a bound that can never exceed $\log N$.

Figure 37.19 shows the task. Its smallest case is $N = 2$: with a single negative, the loss is the binary cross-entropy of telling the true partner from the other candidate. Even a perfect answer shows only that one of two candidates was right, and the estimate is at most $\ln 2 = 0.69$ nats.

37.7.4.2 The InfoNCE Bound

The proposition says that a small loss certifies information, and that the certificate has a ceiling (Poole et al. 2019).

Proposition (InfoNCE bound). *For any measurable critic T and any $N \geq 1$,*

$$I(X;Y) \geq \log N - \mathcal{L}_{\text{NCE}}. \tag{37.79}$$

The right-hand side is at most $\log N$.

Proof. Number the candidates $y_1 = y^+, y_2, \dots, y_N$ and write $V = (Y_1, \dots, Y_N)$ for the whole bag. The negatives $Y_{2:N}$ are independent of the pair (X, Y_1). The proof replaces Y by the bag, finds a normalized model of the bag given X, and applies Barber–Agakov.

Step 1: independent negatives add no information. The chain rule for mutual information (Section 37.6) splits the information in the bag, $I(X; Y_1, Y_{2:N}) = I(X;Y_1) + I(X; Y_{2:N} \mid Y_1)$. Given Y_1, the negatives are still independent of X, because they are independent of (X, Y_1); the second term is therefore 0, and $I(X;V) = I(X;Y)$.

Step 2: the softmax ratio is a normalized model. This is the step that is not routine. Define

$$\gamma(x,v) = \frac{e^{T(x,y_1)}}{\frac{1}{N}\sum_{j=1}^{N} e^{T(x,y_j)}},$$

which lies between 0 and N. Let $p(v)$ be the distribution of a bag of N independent draws from P_Y, the positive slot included. The N candidates of such a bag are independent and identically distributed, so permuting them does not change the distribution of the bag. The N ratios $e^{T(x,y_i)}/\frac{1}{N}\sum_j e^{T(x,y_j)}$ therefore have the same expectation, and for every fixed x

$$\mathbb{E}_{p(v)}\left[\gamma(x,V)\right] = \frac{1}{N}\sum_{i=1}^{N} \mathbb{E}_{p(v)}\left[\frac{e^{T(x,Y_i)}}{\frac{1}{N}\sum_j e^{T(x,Y_j)}}\right] = \mathbb{E}_{p(v)}\left[\frac{\frac{1}{N}\sum_i e^{T(x,Y_i)}}{\frac{1}{N}\sum_j e^{T(x,Y_j)}}\right] = 1.$$

Hence $q(v \mid x) = p(v)\,\gamma(x,v)$ is a conditional distribution over bags.

Step 3: Barber–Agakov on the bag. The marginal distribution of the bag V is $p(v)$, because the positive is marginally a draw from P_Y and is independent of the negatives. Apply the ratio form of Equation 37.74 to the pair (X, V) with the model $q(v \mid x)$, the two variables' roles exchanged, which the symmetry of I allows:

$$I(X;V) \geq \mathbb{E}_{p(x,v)}\left[\log \frac{q(v \mid x)}{p(v)}\right] = \mathbb{E}\left[\log \gamma(X,V)\right] = \log N - \mathcal{L}_{\text{NCE}}.$$

The last equality holds because $\log \gamma$ is $\log N$ plus the log-softmax probability of the positive. With Step 1 this proves the bound. The softmax probability of the positive is at most 1, so $\mathcal{L}_{\text{NCE}} \geq 0$ and the right-hand side is at most $\log N$. ■

Naming the right partner out of N candidates, even with certainty, demonstrates $\log N$ nats, the information in one choice among N; no critic can make that choice show more. The ten-class classifier that could not certify more than $\ln 10$ faced the same task with the labels as candidates. A larger N raises the ceiling, which is the reason usually given for the large batches of CPC, SimCLR and CLIP. The loss is smallest for the critics $T^*(x, y) = \text{pmi}(x, y) + c(x)$, with any function c of the anchor alone (Exercise 7).

37.7.4.3 The Ceiling with an Exact Critic

One might guess that with the exact critic $T = \text{pmi}$ the estimate follows the truth up to $\log N$ and stops there. Inserting $T = \text{pmi}$ into the loss shows what happens instead:

$$\hat{I}_{\text{NCE}} = I(X;Y) - \mathbb{E}\left[\log \frac{1}{N} \sum_{j=1}^{N} e^{\text{pmi}(x,y_j)}\right]. \tag{37.80}$$

The estimate is the truth minus a correction. The correction is the bound's own slack, so it is nonnegative by the proposition, and we can say roughly how large it is. The average inside the logarithm has two parts. The $N - 1$ negatives are independent of x, so each of their terms has mean 1 (the second of the two identities we checked at the start), and together they contribute about 1. The positive contributes $e^{\text{pmi}(x,y^+)}/N$, which is about e^I/N for a typical pair. Replacing both parts by these values gives

$$\hat{I}_{\text{NCE}} \approx I - \log\left(1 + \frac{e^I}{N}\right) = -\log\left(e^{-I} + \frac{1}{N}\right). \tag{37.81}$$

The second form is a smooth minimum of I and $\log N$. It is close to I while e^I is much smaller than N, close to $\log N$ when e^I is much larger, and below both in between: at $I = \log N$ it reads $\log N - \log 2$, which is 0.69 nats short of either. The guess overlooked that the positive sits in its own denominator, next to negatives that average 1, and it dominates the sum only when e^I far exceeds N.

Replacing a heavy-tailed sum by its mean is a rough step, so the approximation needs a test. For $N = 128$ and $I = 4$ it predicts 3.6 nats, although the ceiling $\ln 128 = 4.85$ leaves room for the truth. The cell scores all $N \times N$ pairs of a batch at once: row i holds anchor x_i against every y_j, its diagonal entry is the positive, and the other $N - 1$ entries are its negatives. The rows share their candidates, which the proposition permits (Exercise 8). The cell sweeps the true information for $N = 16$, 128 and 1024, whose ceilings $\ln N$ are 2.77, 4.85 and 6.93 nats, and prints the estimates at three levels.

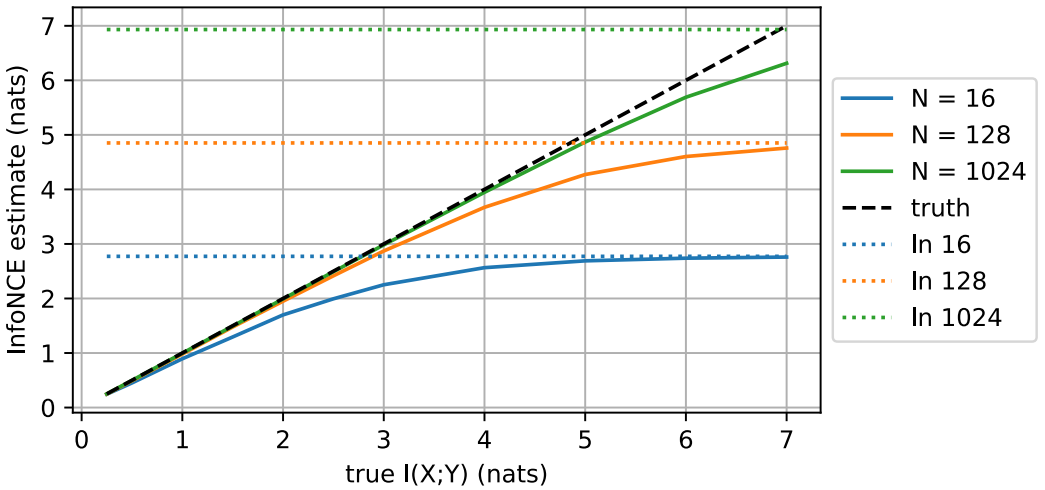
```
def infonce_bound(f):
    """log N minus the InfoNCE loss of an N x N score matrix f[i, j]."""
    return np.log(len(f)) + np.mean(np.diag(f) - logsumexp(f, axis=1))

def exact_scores(x, y, rho):
    """The exact critic T = pmi on all N x N pairs (x_i, y_j) of a batch."""
```

```
    return pmi_gauss(x[:, None], y[None, :], rho)

true_mi = np.array([0.25, 0.5, 1, 1.5, 2, 2.5, 3, 4, 5, 6, 7])
rhos = np.sqrt(1 - np.exp(-2 * true_mi))                # invert mi_gauss
Ns = (16, 128, 1024)
ests = np.array([[np.mean([infonce_bound(exact_scores(*gauss_pair(rng, r, N), r))
                           for _ in range(200 if N <= 128 else 50)])
                  for r in rhos] for N in Ns])              # (N, level)
ceilings = [[np.log(N)] * len(true_mi) for N in Ns]
d2l.plot(true_mi, [*ests, true_mi, *ceilings], 'true I(X;Y) (nats)',
         'InfoNCE estimate (nats)',
         legend=[f'N = {N}' for N in Ns] + ['truth'] + [f'ln {N}' for N in Ns],
         fmts=('C0-', 'C1-', 'C2-', 'k--', 'C0:', 'C1:', 'C2:'),
         figsize=(5.5, 3), legend_loc='outside')
for I in (1, 4, 7):
    k = list(true_mi).index(I)
    print(f'I = {I}:  ' + '   '.join(f'N = {N}: {e:.2f}'
                                      for N, e in zip(Ns, ests[:, k])))
```

```
I = 1:  N = 16: 0.89   N = 128: 0.98   N = 1024: 1.00
I = 4:  N = 16: 2.56   N = 128: 3.67   N = 1024: 3.95
I = 7:  N = 16: 2.76   N = 128: 4.76   N = 1024: 6.31
```



No curve has a corner. Each leaves the diagonal early and bends smoothly toward its ceiling, and all nine printed values lie within 0.1 nat of Equation 37.81. At $I = 1$ every batch size is within about 0.1 nat of the truth. At $I = 4$ a batch of 128 reports 3.7, and even $N = 1024$ falls a little short. At $I = 7$, just above its ceiling of 6.93, $N = 1024$ reports 6.3.

Two exact statements stand behind the approximation. The average inside the logarithm is at least the positive's own term $e^{\mathrm{pmi}(x,y^+)}/N$, so the correction is at least $\mathbb{E}[\mathrm{pmi}(x, y^+)] - \log N = I - \log N$, which is the ceiling again. And the correction vanishes only as $N \to \infty$ (Exercise 7). The ceiling is the crudest consequence of this slack.

The smooth minimum also makes the estimate hard to invert. With $N = 128$ a reading of 3.7 stands for 4 nats and a reading near 4.75 for 7, so the top 1.2 nats of the scale hold every

truth from 4 nats to infinity.

Classification gives these numbers a scale. If each of two views of an image determines its label among $1{,}000$ balanced classes, the label is a function of either view, and by the data-processing inequality (Section 37.6.2.2) the views share at least its entropy, $\ln 1000 = 6.91$ nats. A loss with 128 candidates cannot show that much, and with $N = 1024$ the estimate is still 6.3 when the views share 7. Whether two views carry enough for classification cannot be read off such a number.

37.7.4.4 Exact Critics Compared

The three bounds can now be compared on equal terms: $N = 128$, and the exact critic $T^* = 1 + \text{pmi}$ for each (adding a constant does not change InfoNCE). DV and NWJ take their independent pairs from the N mismatched pairs of the batch, and the enormous values of the NWJ subsection let us predict the outcome. A pair that is typical under the joint distribution has pmi near I. Such pairs have probability about e^{-I} under independence, because there $p(x)\,p(y) = p(x,y)\,e^{-\text{pmi}}$. So about one mismatched pair in e^I could pass for a true pair. Each of these *rare pairs* has e^{pmi} near e^I, and together they carry $\mathbb{E}[e^{\text{pmi}}] = 1$. A batch holds about Ne^{-I} of them: six at $I = 3$, two at $I = 4$, and 0.3 at $I = 6$.

While a batch holds several rare pairs, its average of e^{pmi} is near 1 and the three bounds should agree. When the count falls below one, most batches hold none. Call such a batch *starved*. Its average of e^{pmi} is near zero. On it NWJ reports the batch average of T^* minus almost nothing, about $1 + I$ and one nat high. DV reports the average pmi minus the logarithm of a near-zero average, many nats high.

NWJ is unbiased, so something must make up for the starved batches. It is the batch that is *hit*, one that holds a mismatched pair with $e^{\text{pmi}} \geq N$. That one pair adds at least 1 to the batch average of e^{pmi} and takes at least a nat off NWJ, which then reads about I or less, now and then far less. At $I = 6$ a rare pair has e^{pmi} near $e^6 = 403$, more than N, so a batch that holds one is hit. With a mean count of 0.3 that is roughly one batch in four, and most of the others should be starved.

InfoNCE normalizes within the batch, so a negative that passes for a partner spoils at most one row of the softmax. It should be steady, and it should sit where Equation 37.81 puts it, at 4.6 nats for $I = 6$. The dividing line $Ne^{-I} \approx 1$ is $I \approx \ln N$. The logarithm of the batch size limits DV and NWJ as well, only their values do not show it.

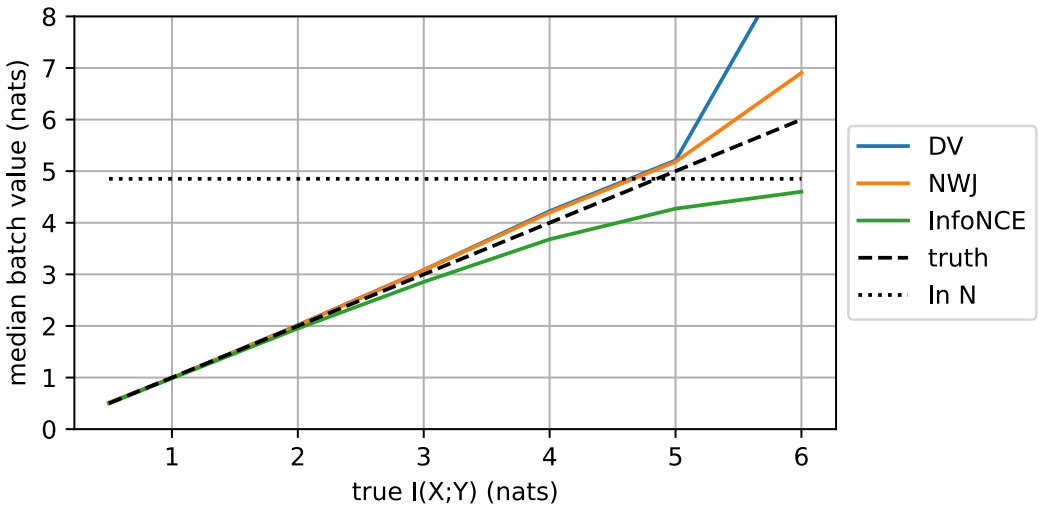
The cell draws 200 batches at each level and prints the median and the quartiles of the batch values, which describe heavy tails better than a standard deviation does, and the mean for NWJ. Its last two columns give the share of starved batches, counted as those whose average of e^{pmi} over the mismatched pairs is below 0.1, and the share of hit batches. The plot shows the three medians.

```
def quartiles(a):
    """Median and quartiles of a list of batch values, as text."""
    q1, med, q3 = np.percentile(a, [25, 50, 75])
    return f'{med:5.2f} [{q1:5.2f}, {q3:5.2f}]'

N, true_mi = 128, np.array([0.5, 1, 2, 3, 4, 5, 6])
medians = []
# Last two columns: share of batches whose mean e^pmi over mismatched pairs
```

```
# is below 0.1, and share holding a mismatched pair with e^pmi >= N.
print(f'{"I":>3}  {"DV median [IQR]":20}  {"NWJ median [IQR]":20}  '
      f'{"NWJ mean":>8}  {"InfoNCE median [IQR]":20}  {"<0.1":>4}  {">=N":>4}')
for I in true_mi:
    rho, batches = np.sqrt(1 - np.exp(-2 * I)), []
    for _ in range(200):
        x, y = gauss_pair(rng, rho, N)
        T_joint = 1 + pmi_gauss(x, y, rho)              # T* on true pairs
        T_prod = 1 + pmi_gauss(x, np.roll(y, 1), rho)   # T* on mismatched
        e_pmi = np.exp(T_prod - 1)                      # e^pmi, mismatched
        batches.append((T_joint.mean() - (logsumexp(T_prod) - np.log(N)),  # DV
                        T_joint.mean() - e_pmi.mean(),                     # NWJ
                        infonce_bound(exact_scores(x, y, rho)),
                        e_pmi.mean() < 0.1, e_pmi.max() >= N))
    dv, nwj, nce, low, rare = np.array(batches).T
    medians.append([np.median(a) for a in (dv, nwj, nce)])
    print(f'{I:3.1f}  {quartiles(dv)}  {quartiles(nwj)}  {nwj.mean():8.2f}'
          f'  {quartiles(nce)}  {low.mean():4.0%}  {rare.mean():4.0%}')
d2l.plot(true_mi, [*np.array(medians).T, true_mi, [np.log(N)] * len(true_mi)],
         'true I(X;Y) (nats)', 'median batch value (nats)',
         legend=['DV', 'NWJ', 'InfoNCE', 'truth', 'ln N'], ylim=[0, 8],
         fmts=('C0-', 'C1-', 'C2-', 'k--', 'k:'), figsize=(5.5, 3),
         legend_loc='outside')
```

```
  I  DV median [IQR]       NWJ median [IQR]      NWJ mean  InfoNCE median [IQR]  <0.1   >=N
0.5   0.51 [ 0.43,  0.58]   0.51 [ 0.43,  0.58]      0.50   0.50 [ 0.45,  0.54]    0%    0%
1.0   0.99 [ 0.88,  1.13]   0.99 [ 0.88,  1.12]      0.99   0.99 [ 0.92,  1.04]    0%    0%
2.0   2.02 [ 1.81,  2.17]   2.01 [ 1.80,  2.16]      1.97   1.95 [ 1.90,  2.01]    0%    1%
3.0   3.08 [ 2.75,  3.47]   3.08 [ 2.70,  3.36]      3.02   2.85 [ 2.79,  2.92]    0%    2%
4.0   4.23 [ 3.74,  4.95]   4.20 [ 3.71,  4.57]      3.76   3.68 [ 3.62,  3.74]    8%    6%
5.0   5.21 [ 4.54,  8.28]   5.18 [ 4.44,  5.94]      4.83   4.27 [ 4.22,  4.31]   32%   36%
6.0   9.47 [ 5.33, 29.04]   6.91 [ 5.10,  7.00]      5.93   4.60 [ 4.58,  4.64]   57%   28%
```



Up to $I = 3$ the three medians are within about 0.2 nat of the truth, hardly any batch is starved or hit, and InfoNCE has the narrowest spread. Starved batches appear where the

count put them: no more than about one in ten at $I = 4$, about a third at $I = 5$, and more than half at $I = 6$.

In that last row NWJ's median is about $7 = I + 1$ with its upper quartile on top of it, so half of the batches overshoot by about a nat. The quarter or so of the batches that are hit pull the mean back to within about half a nat of 6. DV's median lies several nats above the truth, with an upper quartile in the tens, so its curve leaves the plot. InfoNCE reads 4.6 with an interquartile range below 0.1.

The spread of NWJ can be computed. Its mismatched-pair term is a batch average of e^{pmi}, with variance $(e^{2I} - 1)/N$: a standard deviation of 36 nats at $I = 6$, carried by batches so rare that 200 draws seldom include one (Exercise 2).

None of this contradicts the bounds, which hold for expectations over the population. This is how the variational bounds live with the limit of Section 37.7.2.4. InfoNCE reports at most $\log N$ with N at most the sample size, so it respects the limit by construction. DV and NWJ can report more, and when they do, their batch values are no longer guarantees.

37.7.4.5 Training a Critic

In practice the critic is learned. When a contrastive loss reports less than the truth, is the critic at fault, or the bound? We train a critic of the dual-encoder form that CLIP and SimCLR use: two small networks $g, h : \mathbb{R} \to \mathbb{R}^{16}$ embed x and y, and the critic is the inner product $T(x, y) = g(x)^\top h(y)$. The $N \times N$ score matrix of a batch is then one matrix product of the two embedding matrices, so N anchors against N candidates cost $2N$ network evaluations instead of N^2. The data are two disjoint pools of $4{,}096$ pairs at $\rho = 0.99$, one for training and one for evaluation; the true information is 1.96 nats.

```
rho, n_pool = 0.99, 4096
x_tr, y_tr = (a.astype(np.float32) for a in gauss_pair(rng, rho, n_pool))
x_ev, y_ev = (a.astype(np.float32) for a in gauss_pair(rng, rho, n_pool))
```

Each encoder is a multilayer perceptron with two hidden layers of 64 ReLU units and Glorot-uniform initialization. The loss is the InfoNCE loss of the score matrix. It is the ordinary classification cross-entropy, in which the label of row i is i, the index of its positive.

```
def encoder(d_out=16, hidden=64):
    """An MLP that embeds a scalar in d_out dimensions."""
    net = torch.nn.Sequential(torch.nn.Linear(1, hidden), torch.nn.ReLU(),
                              torch.nn.Linear(hidden, hidden), torch.nn.ReLU(),
                              torch.nn.Linear(hidden, d_out))
    for layer in net[::2]:                          # the Linear layers
        torch.nn.init.xavier_uniform_(layer.weight)
        torch.nn.init.zeros_(layer.bias)
    return net

def scores(g, h, x, y):
    """All N x N critic values T(x_i, y_j) = g(x_i) . h(y_j)."""
    return g(x[:, None]) @ h(y[:, None]).T

def infonce_loss(f):
    """Cross-entropy of each row's softmax at its positive, the diagonal."""
```

```
        return torch.nn.functional.cross_entropy(f, torch.arange(len(f)))

    g, h = encoder(), encoder()
```

Training runs $1{,}000$ Adam steps with learning rate 10^{-3} on batches of $N = 128$ pairs drawn from the training pool. Afterwards, `critic_np` evaluates the trained critic on NumPy arrays, the form in which the exact critic `exact_scores` takes its inputs.

```
    opt = torch.optim.Adam([*g.parameters(), *h.parameters()], lr=1e-3)
    N = 128
    for step in range(1000):
        idx = rng.choice(n_pool, N, replace=False)
        loss = infonce_loss(scores(g, h, torch.tensor(x_tr[idx]),
                                   torch.tensor(y_tr[idx])))
        opt.zero_grad()
        loss.backward()
        opt.step()

    def critic_np(a, b):
        """The trained critic on NumPy inputs, as an N x N NumPy array."""
        with torch.inference_mode():
            return scores(g, h, torch.tensor(a), torch.tensor(b)).numpy()
```

The comparison that answers the question puts both critics on the *same* batches. The evaluation cell cuts the evaluation pool into disjoint batches of N pairs, for N from 2 to 128, and computes the InfoNCE bound of the learned critic and of the exact critic on each. A critic trained with 128 candidates may be tested with fewer, because the best critic, $\mathrm{pmi}(x, y) + c(x)$, does not depend on N.

```
    def infonce_eval(score, x, y, N):
        """Mean InfoNCE bound over the disjoint batches of N pairs in (x, y)."""
        return np.mean([infonce_bound(score(x[i:i + N], y[i:i + N]))
                        for i in range(0, len(x), N)])

    exact_np = lambda a, b: exact_scores(a, b, rho)   # the exact critic T = pmi

    print('   N   ln N   learned   exact')
    for N_ev in (2, 8, 32, 128):
        print(f'{N_ev:4d}  {np.log(N_ev):5.2f}  '
              f'{infonce_eval(critic_np, x_ev, y_ev, N_ev):7.2f}  '
              f'{infonce_eval(exact_np, x_ev, y_ev, N_ev):6.2f}')
    print(f'true I(X;Y) = {mi_gauss(rho):.2f} nats')
```

```
   N   ln N   learned   exact
   2   0.69     0.58    0.58
   8   2.08     1.42    1.42
  32   3.47     1.78    1.80
 128   4.85     1.89    1.91
true I(X;Y) = 1.96 nats
```

The learned critic matches the exact one to within a few hundredths of a nat at every N. The gap to the true 1.96 nats belongs to the bound, and it closes only as N grows: both critics

reach about 1.9 nats at $N = 128$. At $N = 8$ the ceiling $\ln 8 = 2.08$ already exceeds the truth, so the shortfall of about 0.55 nat there is not the ceiling at work. It is the slack of the bound, which Equation 37.81 puts at about 0.6. A scalar pair gives the critic an easy job. In high dimension its error is no longer negligible, and without a known truth the two errors cannot be separated.

SimCLR-style systems use a different critic: they normalize the two embeddings z_x and z_y and divide their cosine similarity by a *temperature* τ, $T(x, y) = \mathrm{sim}(z_x, z_y)/\tau$ (Chen et al. 2020). The temperature sets how sharply the softmax separates the candidates, and it adds a ceiling of its own. A cosine lies in $[-1, 1]$, so no two scores differ by more than $2/\tau$, and a cosine critic certifies less than $2/\tau$ nats whatever N is (Exercise 5): 4 nats at $\tau = 0.5$ and 20 at $\tau = 0.1$. It is a large temperature that limits the bound.

What should one use, then? For two variables of one or two dimensions that share at most a few nats, count neighbors: KSG was within about 0.1 nat there. For vectors, no estimator of this section measures a large mutual information, and the question changes from *how much* to *at least how much*. InfoNCE answers that one steadily. Train with it on as many candidates as fit in a batch and read $\log N - \mathcal{L}_{\mathrm{NCE}}$ as a lower bound. A reading within a nat or so of $\log N$ says only that the views share at least that much, and a low reading may be the critic's fault. DV and NWJ have no range of their own. A couple of nats below $\log N$, where their batch values can be trusted, InfoNCE is nearly as close to the truth and steadier; higher up, their typical batch overshoots.

A tighter bound does not mean a better representation, either. Tschannen et al. (2020) found that downstream quality correlates poorly with the tightness of the bound being maximized. Looser bounds sometimes give better representations, and the encoder architecture and the form of the critic matter as much as the information. Two questions remain open: how to estimate a large mutual information in high dimension at all, and why loose bounds still train good representations.

37.7.5 Summary

Samples can measure mutual information when it is small and the variables have few dimensions. Beyond that, no estimator reports more than the logarithm of the number of things it can tell apart: $\log b$ for a histogram with b bins, about $\log(n/m)$ for nearest-neighbor counts, which saturate far below that in high dimension, and the order of $\log n$ for any lower bound that holds for every distribution. Variational methods therefore settle for a lower bound, computed from a decoder or a critic and maximized. InfoNCE, the loss of contrastive learning, is steady, and with an exact critic it reports about $-\log(e^{-I} + 1/N)$, never more than $\log N$ for N candidates. The Donsker–Varadhan and NWJ bounds can report more, but their batch values can be trusted only a couple of nats below $\log N$. On the Gaussian pair a trained critic matched the exact one, so the shortfall of the contrastive estimate belonged to the bound. None of these numbers says how good the trained representation is.

37.7.6 Exercises

1. code **Bias of the histogram estimator.** For an independent pair, `binned_mi` of Section 37.7.2.1 reports about 0.003 nats at $b = 40$ and a few hundredths of a nat at $b = 160$, from $n = 200{,}000$ samples. This problem predicts such numbers.

1. For an independent pair, $2n$ times the estimate of `binned_mi` is the G-statistic of the test for independence. When every cell's expected count is large, this statistic is approximately χ^2-distributed with $(b-1)^2$ degrees of freedom, and so has mean $(b-1)^2$. Predict the mean estimate at $b = 40$ and at $b = 160$, and compare with the first row of the table. Why are the printed values smaller, and why is the gap wider at $b = 160$? *Hint:* count the cells that hold any sample.
2. Replace each coordinate by its rank divided by n before binning, so that every bin of each axis holds the same number of samples. Average the estimate over 20 independent pairs with $\rho = 0$ for $n \in \{2000, 20{,}000\}$ and $b \in \{10, 40\}$, and compare with the prediction of (a). For which setting does the prediction fail, and why?
3. Bin the ranks of the Gaussian pair at $\rho = 0.99$ with $n = 200{,}000$ and $b \in \{40, 100, 400\}$. Which of the two errors of the histogram dominates at each b? Together with (b), what does this suggest about how b should grow with n?

2. **Heavy tails of the exponentiated pmi.** ⊠ Let (X, Y) be the Gaussian pair with correlation ρ, $I = -\frac{1}{2}\log(1-\rho^2)$, and $Q = P_X \otimes P_Y$. Under Q the mean of e^{pmi} is 1 for every ρ, although a typical independent pair has e^{pmi} near 0 when ρ is close to 1.

   1. Show that $\mathbb{E}_Q[e^{2\,\text{pmi}}] = \int p^2/q = 1/(1-\rho^2) = e^{2I}$, so that e^{pmi} has variance $e^{2I} - 1$ under Q. *Hint:* the integrand is proportional to a Gaussian density with precision matrix $2\Sigma^{-1} - \mathbf{I}$, where Σ is the covariance of the pair, and $\det(2\Sigma^{-1} - \mathbf{I}) = 1$.
   2. A random permutation of a batch leaves on average one y paired with its own x. Show that such a fixed point adds e^{pmi}/N to the batch mean of e^{pmi} over the permuted pairs, with expectation e^{2I}/N: 0.43, 23 and 1270 at $I = 2, 4, 6$ with $N = 128$. Why does `np.roll` avoid this?
   3. For the NWJ critic $T^* = 1 + \text{pmi}$, show that the batch estimate has mean I when the mismatched pairs are independent draws, and that its mismatched-pair term then has variance $(e^{2I} - 1)/N$: standard deviations of 0.65, 4.8 and 36 nats at $I = 2, 4, 6$ with $N = 128$. Show that the rolled pairs (x_j, y_{j+1}) of the cell give the same variance for $N \geq 3$, although neighboring terms share a variable. *Hint:* condition on the shared pair.

3. code **Limits of KSG.** The function `ksg_counts` counts neighbors with strict inequalities and excludes each point from its own counts, and `ksg_mi` turns the counts into an estimate.

   1. The table of Section 37.7.2.3 prints the mean strip count $\bar{n}$ of each row. Using $\psi(j) \approx \log j$ and $n_x \approx n_y \approx \bar{n}$, show that the estimate is about $\log \frac{mn}{\bar{n}^2}$, and reproduce the five printed estimates from the five counts to within a few tenths of a nat. Why does the formula come out a little low in every row?
   2. Run `ksg_counts` and `ksg_mi` at $d = 16$ with $\rho = 0.5$ and with $\rho = 0.99$, where the truth is 2.3 and 31 nats. Does the estimate still increase with the dependence? Could it rank the dependence of several pairs, and could it measure it?
   3. Let Y be X plus Gaussian noise of standard deviation 10^{-6}, with X standard normal and one-dimensional. What is the true $I(X;Y)$, and what does KSG report at $n = 2000$? Which ceiling binds, and how many samples would raise it to the truth?

4. **Barber–Agakov with a Gaussian decoder.** Take the standard bivariate Gaussian pair with correlation ρ, whose information is $-\frac{1}{2}\log(1-\rho^2)$, and the decoders $q(x \mid y) =$

$\mathcal{N}(x; ay, s^2)$ with parameters $a \in \mathbb{R}$ and $s^2 > 0$. In the bound Equation 37.74, use the differential entropy $h(X) = \frac{1}{2}\log(2\pi e)$.

1. Evaluate the right-hand side of Equation 37.74 as a function of a and s^2. Maximize it over both parameters and show that the maximum is the true information.
2. The text fixes $s^2 = 1$, which is what training with the squared error alone amounts to, and finds the best bound $\rho^2/2$. Show that its gap to the true information is $D_{\mathrm{KL}}(\mathcal{N}(0, 1-\rho^2) \,\|\, \mathcal{N}(0,1))$, as the proof of the proposition says it must be. How does the gap behave as $\rho \to 1$?
3. Let $\hat{x}(y)$ be any predictor of X with mean squared error D. Use the decoder $\mathcal{N}(x; \hat{x}(y), D)$ to show that $I(X;Y) \geq -\frac{1}{2}\log D$. What is the corresponding bound when $X \sim \mathcal{N}(0, v)$? Compare it with the Gaussian rate–distortion function $R(D) = \frac{1}{2}\log(v/D)$ for $D < v$ of Section 37.6.3.2, the least information any description of X needs to reach squared error D.

5. **Temperature of a cosine critic.** SimCLR-style systems score candidates with $T(x,y) = s(x,y)/\tau$, where $s(x,y) = \mathrm{sim}(z_x, z_y) \in [-1,1]$ is a cosine similarity and $\tau > 0$ is the temperature. For one anchor x, write $s_j = s(x, y_j)$, with $y_1 = y^+$ the positive, and let ℓ be the loss inside the expectation Equation 37.77.

   1. Show that $\mathcal{L}_{\mathrm{NCE}} \geq \log\left(1 + (N-1)e^{-2/\tau}\right)$, and hence $\hat{I}_{\mathrm{NCE}} < 2/\tau$ for every N. Evaluate the ceiling $\log N - \log\left(1 + (N-1)e^{-2/\tau}\right)$ for $N = 4096$ and $\tau \in \{1, 0.5, 0.1\}$ (you should find about 2.0, 4.0 and 8.3 nats). When does the temperature rather than the batch size limit the bound?
   2. Show that $\partial \ell / \partial s_j = p_j/\tau$ for every negative $j \geq 2$, where p_j is the softmax probability of candidate j. Express $\partial \ell / \partial s_1$ through the p_j.
   3. Show that $\tau\ell \to \max\{0, \max_{j\geq 2} s_j - s_1\}$ as $\tau \to 0$, and that

$$\ell = \log N - \frac{1}{\tau}\Big(s_1 - \frac{1}{N}\sum_{j=1}^{N} s_j\Big) + O(\tau^{-2}) \quad \text{as } \tau \to \infty.$$

      Which negatives drive the gradient in each limit (Wang and Liu 2021)?

6. **Entropy bounds from samples.** ⊠ Section 37.7.2.4 states the theorem of McAllester and Stratos (2020) informally and proves a version of it for entropy; this problem sharpens and applies that proof. Draw a sample S of n i.i.d. values from a discrete distribution p. The *type* of S records, for each $i \geq 1$, how many distinct values occur exactly i times in S; it forgets which values these are. Let B be a function of the type that is a lower confidence bound for every discrete distribution: $P_{S\sim p^n}(B \leq H(p)) \geq 0.95$ for every p. Let $\mathcal{T}_0$ be the type of a sample of n distinct values.

   1. The text applies the guarantee to the uniform distribution on n^2 values. Show that any integer $M \geq n^2/1.9$ serves as well, so that $B(\mathcal{T}_0) \leq \log M$. By how much does this improve on $2\log n$?
   2. Let p be uniform on $M = 2^{256}$ values. How large must n be before $2\log n$ reaches $H(p)$? What does this say about measuring the entropy of a cryptographic key from samples?

7. **Optimal InfoNCE critic.** ▯ For one anchor x and one bag $v = (y_1, \dots, y_N)$ with $y_1 = y^+$, write $\ell(x, v) = -\log\left(e^{T(x,y_1)} / \sum_{j=1}^{N} e^{T(x,y_j)}\right)$, so that $\mathcal{L}_{\text{NCE}}$ of Equation 37.77 is its expectation.

   1. Show that replacing $T(x, y)$ by $T(x, y) + c(x)$, for any function c, leaves $\mathcal{L}_{\text{NCE}}$ unchanged.
   2. Shuffle the bag so that the positive sits at a uniformly random position J and the other $N - 1$ positions hold the negatives. Show that

$$P(J = i \mid x, y_1, \dots, y_N) = \frac{e^{\text{pmi}(x,y_i)}}{\sum_{j=1}^{N} e^{\text{pmi}(x,y_j)}}. \tag{37.82}$$

      Use Gibbs' inequality to conclude that every critic $T(x, y) = \text{pmi}(x, y) + c(x)$ minimizes $\mathcal{L}_{\text{NCE}}$.
   3. For this critic, show that

$$\hat{I}_{\text{NCE}} = I(X;Y) - \mathbb{E}\left[\log \frac{1}{N} \sum_{j=1}^{N} e^{\text{pmi}(x,y_j)}\right]. \tag{37.83}$$

      Use the law of large numbers to argue that the correction tends to zero as $N \to \infty$ when $I(X;Y)$ is finite. Evaluate the correction by Monte Carlo at $\rho = 0.99$ and $N = 8$, and compare it with the gap between the exact-critic value (about 1.4) and the truth (1.96) in Section 37.7.4.5.
   4. The approximation Equation 37.81 replaces the negatives' share of the average by 1 and $\text{pmi}(x, y^+)$ by I. Use Jensen's inequality to show that the first replacement lowers the estimate and the second raises it. Which of the two wins in the table of Section 37.7.4.3?

8. **In-batch negatives and the symmetric loss.** The cells of Section 37.7.4.5 score all $N \times N$ pairs of a batch of N independent pairs (x_i, y_i) and use row i's off-diagonal entries as the negatives of anchor x_i.

   1. Show that each row satisfies the hypotheses of the InfoNCE proposition: its candidates y_j, $j \neq i$, are draws from P_Y that are independent of (x_i, y_i). Conclude that the mean row loss $\mathcal{L}$ certifies $I(X;Y) \geq \log N - \mathcal{L}$ in expectation.
   2. The rows share candidates, so their losses are dependent. Does this affect the validity of the bound, the expectation of the mean row loss, or its variance?
   3. CLIP uses the symmetric loss, the mean of the row-wise and the column-wise cross-entropies. Show that it bounds $I(X;Y)$ in the same way, with the same ceiling $\log N$.
   4. A joint critic $T(x, y)$, a network applied to the concatenated pair, needs N^2 network evaluations for the scores of a batch, where the separable critic $g(x)^\top h(y)$ needs $2N$. Evaluate both counts for CLIP's batches of $N = 32{,}768$. A separable critic cannot represent every function of the pair. Show that the pmi of the Gaussian pair is separable with embeddings of three dimensions, and that two suffice for InfoNCE.

9. **Negative sampling as shifted PMI.** ▯ Skip-gram with negative sampling, the training method of word2vec, scores a center word w and a context word o by the inner product $s(w,o) = \mathbf{u}_o^\top \mathbf{v}_w$ of their vectors. It learns to tell the pairs that occur in a corpus from pairs whose context word is one of K noise words drawn from a distribution P. Let $p(w,o)$ be the frequency of the pair among all (center, context) pairs of the corpus, with marginals $p(w)$ and $p(o)$. Averaged over the corpus, the loss is

$$\ell = -\sum_{w,o} p(w,o)\log\sigma(s(w,o)) - K\sum_{w} p(w)\sum_{o} P(o)\log\sigma(-s(w,o)), \quad (37.84)$$

where $\sigma(s) = 1/(1+e^{-s})$ is the sigmoid. Assume that the embeddings have enough dimensions for every score $s(w,o)$ to be chosen independently.

   1. Show that ℓ is minimized at $s^*(w,o) = \log\frac{p(w,o)}{p(w)\,P(o)} - \log K$, which equals $\mathrm{pmi}(w,o) - \log K$ when P is the unigram distribution $p(o)$ (Levy and Goldberg 2014).
   2. word2vec draws noise words with probability proportional to their count raised to the power 0.75. Show that s^* then differs from $\mathrm{pmi}(w,o)$—$\log K$ by a term that depends on o alone. For which context words is this term negative? How does it act on pairs of rare words, such as the "sri lanka" of Exercise 2 of Section 37.6?
   3. Compare s^* with the optimal InfoNCE critic of Exercise 7b. Why does the binary classification of negative sampling fix the additive term that InfoNCE leaves free?

10. [extended] **Compression phase at small scale.** ▯▯ Shwartz-Ziv and Tishby (2017) studied fully connected tanh networks of widths 12-10-7-5-4-3-2 on the 4096 binary input patterns of 12 bits. For each hidden layer Z they binned every unit's activation into 30 equal intervals of $[-1,1]$ and computed $I(X;Z)$ and $I(Y;Z)$ from the binned values. Section 37.7.2.5 describes the debate about the compression phase they reported.

   1. Let X be uniform on the 4096 patterns and let Z_{bin} be the binned layer of a deterministic network. Show that the binned estimate of $I(X;Z)$ equals $H(Z_{\text{bin}})$ and never exceeds $\ln 4096 \approx 8.32$ nats. When does it decrease during training?
   2. Fix a random vector $\mathbf{w} \in \mathbb{R}^{12}$ and let $Y = 1$ when $\mathbf{w}^\top\mathbf{x}$ exceeds its median over the 4096 patterns. Train the tanh network, with a softmax over its two output units, by SGD with the cross-entropy loss on a random 85% of the patterns. Plot the trajectory of every hidden layer in the information plane $(I(X;Z), I(Y;Z))$ over training.
   3. Repeat (b) with ReLU hidden units, binning each unit into 30 equal intervals between 0 and its largest activation over the run.
   4. Record the fraction of tanh units with $|z| > 0.99$ during training. Does the decrease of the binned $I(X;Z)$ in (b) track this fraction? Which conclusions about compression survive the change of activation in (c) (Saxe et al. 2018)?

37.8 Further Reading

The chapter kept to what can be proved in a page and checked on a small example, and each entry below starts where a section stopped. Books and papers for what the chapter teaches are listed in its introduction (Chapter 37); these entries are for what it does not. Read first the three that bear on practice: calibration, the constant of a scaling law, and the bias and

variance of mutual-information estimators. Bits-back coding is worth reading for its own sake. The entries on capacity and on information geometry can wait. No other page of the book builds on this one.

37.8.1 Entropy and Losses

37.8.1.1 Calibration after Training

Over the population the log-loss is smallest at the true conditional probabilities (Section 37.1.3.1), which are calibrated. A trained network is somewhere else, and C. Guo et al. (2017) find that modern networks miss on one side: they are overconfident. One-hot targets are a reason. They ask for an infinite gap between the logits (Section 37.1), so once the training set is classified without error, the loss can fall only by pushing the logits apart. Late in training the same authors see the held-out log-loss rise again, although the held-out error does not: the network overfits its probabilities and not its decisions. The temperature scaling of Section 3.5 repairs this with one constant, fitted on held-out data. It is the temperature of distillation, used for a different purpose.

How much can such a repair achieve? Let $\bar{\mathbf{p}}(\mathbf{q})$ be the frequencies of the classes among the inputs that receive the prediction $\mathbf{q}$. A cross-entropy is an entropy plus a KL divergence, and this holds for each prediction, so the held-out log-loss is

$$\mathbb{E}\big[-\log q_y\big] = \mathbb{E}_{\mathbf{q}}\big[D_{\mathrm{KL}}(\bar{\mathbf{p}}(\mathbf{q}) \,\|\, \mathbf{q})\big] + \mathbb{E}_{\mathbf{q}}\big[H(\bar{\mathbf{p}}(\mathbf{q}))\big].$$

The first term is the miscalibration, in nats: it is zero only when the classes occur as often as the predictions say. The second is the uncertainty that remains about the label once the prediction is known. A map from predictions to new predictions can remove the first term and cannot lower the second. Temperature scaling is such a map with one parameter, and one is too few for a model that is overconfident on some classes and underconfident on others. Every proper scoring rule splits in the same way, with an entropy and a divergence of its own (Gneiting and Raftery 2007).

37.8.1.2 Entropy Relative to a Reference

Section 37.1.4.1 found that differential entropy grows by $\ln 100$ when metres become centimetres, while a KL divergence stays where it was, because both of its densities change by the same factor. Entropy measured against a reference density m, as $-\int p \log(p/m)\, dx$, has the form of $-D_{\mathrm{KL}}(p\|m)$ and is protected in the same way. The ordinary h is this quantity with $m = 1$ in the chosen units, and one per metre is not one per centimetre, so the $\ln 100$ was a change of reference. Jaynes (1968) obtains the form as the limit of discrete entropies on ever finer grids, and with it the maximum-entropy principle reads "stay as close to the reference as the constraints allow". The KL penalty of a variational autoencoder (Section 37.6.4.3) is, up to its sign, such a relative entropy, with the prior as the reference, and so it does not depend on the coordinates of the latent variable.

37.8.2 Codes and Compression

37.8.2.1 Compression without a Model

Every coder of Section 37.2 is driven by a model that predicts the next symbol. The Lempel–Ziv algorithms need none. LZ77 (Ziv and Lempel 1977) replaces a repeated substring by a

pointer to its previous occurrence, and LZ78 (Ziv and Lempel 1978) builds a dictionary of phrases as it parses. Both are *universal*: on any stationary source whose statistics one long text reveals (an ergodic source), their rate approaches the entropy rate as the text grows, although nobody tells them what the source is. The approach to that rate is slow. On *The Time Machine*, `zlib`, which combines LZ77 with a Huffman code, needs 2.84 bits per character, and the counting model of Section 37.2.2.2, which predicts each character from the three before it, needs 2.55.

37.8.2.2 Bits-Back Coding

A model with a latent variable z, such as a variational autoencoder, cannot drive the arithmetic coder of Section 37.2.2.1. The coder needs $p(x)$, and the model offers only the ELBO of Section 36.7, a lower bound on $\log p(x)$ computed with an approximate posterior, which that section writes $q(z \mid x)$. Surprisingly, the bound is a file length all the same, because decoding is sampling. A decoder for $q(z \mid x)$ that is fed random bits returns a z drawn from q and uses up $\log_2 1/q(z \mid x)$ of them. In *bits-back coding* the sender takes these bits from the end of the file that already holds the earlier data points and that, being compressed, is as good as random. The sender then appends x, coded with $p(x \mid z)$, and z, coded with the prior $p(z)$. The receiver decodes z and x and then *encodes* z with $q(z \mid x)$, which puts the borrowed bits back. The file has grown by $\log_2 1/p(x \mid z) + \log_2 1/p(z)$ bits and shrunk by $\log_2 1/q(z \mid x)$, and on average over q the net growth is the negative ELBO. The receiver undoes the sender's steps last to first, so the coder must work as a stack. The arithmetic coder on which Frey and Hinton (1997) built the scheme is not one. Asymmetric numeral systems (Duda 2013) are, and with them Townsend et al. (2019) compress MNIST with a small variational autoencoder at a rate close to its negative ELBO.

37.8.2.3 Description Length beyond Two-Part Codes

The two-part code of Section 37.2.2.3 spends bits on the parameter at a chosen precision, so its length depends on the grid. A *prequential* code sends no parameter. It codes each symbol with the model fitted to the symbols before it, as the adaptive counting model of Section 37.2.2.2 did, and the cost of learning appears inside the data term. Blier and Ollivier (2018) code the training labels of deep networks in this way and find the code far shorter than two-part codes and than variational codes, which apply bits-back coding to the weights. A large network, for all its parameters, does compress its labels. Grünwald (2007) develops these codes together with normalized maximum likelihood, the code whose worst-case excess over the best model of the class, chosen in hindsight, is smallest. Kolmogorov complexity, the length of the shortest program that prints the data, is the idealization of all of them, and it is not computable (Li and Vitányi 2019).

37.8.2.4 The Constant of a Scaling Law

Section 11.7 fits the held-out loss of language models of many sizes by a constant E plus two power laws, one falling in the number of parameters and one in the number of training tokens (Hoffmann et al. 2022), and reads E as the entropy of the text. At best E is the entropy rate per token (Section 37.2.3.1), below which no expected loss falls. The fitted value is a regression constant, extrapolated from losses that all lie above it, and not a measurement of that rate. Kaplan et al. (2020) fit similar runs with no constant at all, which must fail somewhere beyond their range, because a loss that tends to zero ends below the floor. And E is counted per token, so constants fitted with different tokenizers cannot be compared

until both are converted to bits per byte (Section 8.3.2.4).

37.8.3 Comparing Distributions

37.8.3.1 Rényi Divergences and Privacy

How much does a randomized algorithm reveal about one record when it is run many times? Differential privacy measures the leak by a divergence between the distributions of the output on two data sets that differ in that record. Differentially private SGD adds noise to every gradient step and takes thousands of steps, so the divergence must be one that adds up over them. The Rényi divergence of order α (Rényi 1961),

$$D_\alpha(P\|Q) = \frac{1}{\alpha - 1} \log \sum_x p(x)^\alpha q(x)^{1-\alpha},$$

does. Its logarithm stands *outside* the sum, where an f-divergence has its function inside. The sum factorizes over independent components, so D_α adds over them, as KL does and χ^2 and total variation do not. Rényi differential privacy (Mironov 2017) bounds D_α for one step, and the bound for a training run is the sum over its steps. KL adds too, but it bounds the log-ratio $\log(p(x)/q(x))$ only on average. D_α is nondecreasing in α, from KL as $\alpha \to 1$ to the largest value of the log-ratio as $\alpha \to \infty$, so a large order also protects the record against the worst output. Erven and Harremoës (2014) collect the limits and inequalities, and Exercise 1d of Section 37.3 relates D_α to the α-divergences.

37.8.3.2 Information Geometry

To second order, every smooth f-divergence between $q_{\theta+\mathbf{h}}$ and q_θ is the quadratic $\frac{1}{2} f''(1)\, \mathbf{h}^\top \mathbf{I}(\theta)\, \mathbf{h}$ in the Fisher information (Section 37.3). That section found this by a Taylor expansion. Chentsov's theorem says that it had to come out so. Such a quadratic gives every small step $\mathbf{h}$ a length, which makes it a Riemannian metric (it measures steps, where the metrics of Section 37.3.1.1 measure the distance between two distributions). For distributions on finite sets the Fisher information is, up to scale, the only Riemannian metric that is unchanged when the data are replaced by a sufficient statistic (Čencov 1982). No f-divergence changes under that replacement: the data can be drawn again from the statistic without knowing θ, so the data-processing inequality of Section 37.3.3.3 holds in both directions. The quadratic of an f-divergence is therefore a multiple of the Fisher information. Gradient descent with its steps measured in this metric is the natural gradient, which K-FAC approximates with one Kronecker product per layer (Section 35.5.5), and the KL trust region of Section 15.2 is, to second order, a ball in it. Amari (2016) develops the geometry, in which maximum likelihood and both steps of the EM algorithm (Section 36.7) become projections.

37.8.3.3 Sinkhorn Divergences and Sliced Wasserstein

How is a transport distance used as a loss on minibatches in many dimensions? Entropic regularization (Section 37.4.2.4) is one answer, and it has a flaw: the regularized cost from a distribution to itself is not zero, and a model that minimizes this cost ends up more concentrated than its data. The Sinkhorn divergence of Exercise 7 in Section 37.4 subtracts the mean of the two self-costs. Beyond that exercise lies the theorem that makes the divergence a loss. On a bounded set, with the Euclidean distance or its square as the cost, it is

nonnegative, zero only between equal distributions, and it tends to zero if and only if the model converges in distribution to the data (Feydy et al. 2019). The *sliced* Wasserstein distance is the other answer. Projected onto one direction, two samples lie on a line, where transport is solved by sorting (Section 37.4.2), and the sliced distance averages this cost over random directions (Bonneel et al. 2015). In many dimensions a random direction seldom shows where two distributions differ, and the image generators of Deshpande et al. (2018) therefore train a discriminator and project its features in place of the pixels.

37.8.3.4 Lipschitz Critics in Wasserstein GANs

The Kantorovich–Rubinstein dual of Section 37.4.2 needs a 1-Lipschitz function, and a network is not one by construction. Section 16.8 compares the three ways of making it one, weight clipping, spectral normalization and a gradient penalty, and Section 16.4 treats the penalties. The penalty of Gulrajani et al. (2017) asks for a gradient of norm one at points between a real and a generated sample, because an optimal critic has unit slope along the lines on which transport moves mass (Exercise 5d of Section 37.4).

37.8.3.5 Stein Variational Gradient Descent

Sampling a model that is known only through its score s_Q takes a long random walk (Section 37.5). The kernel Stein discrepancy of Section 37.5.3 measures how far a sample from P is from such a model Q. Stein variational gradient descent (Liu and Wang 2016) moves the sample there instead. Starting from any set of points, with distribution P, each point y takes small steps along the velocity

$$v(y) = \mathbb{E}_{x\sim P}\big[k(x,y)\, s_Q(x) + \nabla_x k(x,y)\big].$$

The first term is the kernel-weighted average of the scores near y, and it carries y toward higher density. The second pushes y away from its neighbours and does the work of the noise in the random walk: without it all points would gather at a mode. Scaled to the unit ball of the feature space of Section 37.5.3, this velocity decreases $D_{\mathrm{KL}}(P\|Q)$ faster than any other in the ball, at the rate $\|v\| = \mathrm{KSD}(P,Q)$. The same Stein operator serves as a test statistic and as a velocity field. The points still take many small steps, but the steps are deterministic and the points arrive together, which suits a posterior that a few dozen points must represent, as for the weights of a small network in the original paper.

37.8.4 Mutual Information

37.8.4.1 Codes That Approach Capacity

Section 37.6.3.1 states that every rate below capacity can be reached, but Shannon's random codes take exponential time to decode. Low-density parity-check codes, decoded by passing messages on the graph of the code, come close to capacity in practice (Richardson and Urbanke 2008), and polar codes (Arıkan 2009) provably achieve it on symmetric binary-input channels, with encoding and decoding in $O(n \log n)$ operations for block length n. A 5G phone uses both.

37.8.4.2 Learned Lossy Compression

Section 37.6.3.2 computes the rate–distortion function of a Gaussian source. Nobody knows that function for photographs, and a learned image codec is a point somewhere above it.

Ballé et al. (2017) train three parts together: an encoder, a decoder, and an *entropy model*, a learned distribution over the rounded latent. Rounding has no gradient, so uniform noise stands in for it during training. The rate R in their loss is the arithmetic-coding length of the latent under the entropy model (Section 37.2), and D is the distortion of the decoded image. Minimizing $R + \lambda D$ picks the point of the codec's curve at which its slope is $-\lambda$. Ballé et al. (2018) make the entropy model conditional on a second latent, a *hyperprior*, which lowers the rate at equal distortion.

37.8.5 Estimating Mutual Information

37.8.5.1 Upper Bounds

Every variational estimator of Section 37.7 is a lower bound, so none can certify that a representation keeps *little* information, for instance about a protected attribute that it should have forgotten. An upper bound needs the conditional density $p(z \mid x)$, which a stochastic encoder supplies. The variational bottleneck of Section 37.6.4.3 is one such bound, with a fixed density $r(z)$ in place of the marginal. The contrastive log-ratio upper bound, CLUB (Cheng et al. 2020), needs no r: it averages $\log p(z \mid x)$ over the matched pairs of a batch and subtracts its average over the mismatched pairs. It is an upper bound in expectation when the density is the encoder's own. Between a representation and an attribute there is no such density, and a network is fitted in its place. The guarantee then holds on one condition: the fitted density must lie closer in KL divergence to the joint distribution than to the product of the marginals. That cannot be checked without the density that was missing.

37.8.5.2 Bias and Variance of the Bounds

With an exact critic NWJ is right on average but noisy, and InfoNCE is steady but capped at $\log N$ (Section 37.7.4.4). Poole et al. (2019) interpolate between the two with one mixing parameter, which selects a point of the trade-off. J. Song and Ermon (2020) prove that the noise is no accident of one estimator: for the NWJ and DV bounds the variance grows exponentially with the true information, even with an exact critic. Their repair clips the critic's density ratio, which lowers the variance and introduces a bias. Every repair on offer moves error from variance to bias, and none removes it.

The same authors propose tests that need no known answer: an estimate must not change when a function of one variable is appended to that variable, and it must double when the pair is replaced by two independent copies of itself. On images the estimators they try fail these tests. Check a reported mutual information of several nats against the logarithm of the batch size, and trust an estimator on your own data only after it has passed such a test there.

37.8.5.3 Noisy Networks and the Bottleneck View

The debate of Section 37.7.2.5 began with the proposal of Tishby and Zaslavsky (2015) to judge each layer against the bottleneck frontier of Section 37.6.4, the largest $I(Y;Z)$ that can be had at each $I(X;Z)$. That subsection left the compression phase with nothing to measure: in a deterministic network on continuous inputs, $I(X;Z)$ is infinite. Goldfeld et al. (2019) make the question well posed by adding Gaussian noise to the activations, which makes $I(X;Z)$ finite. It can then fall during training, and they trace the fall to clustering: the representations of each class draw together. This, they argue, is also what the binned estimates on deterministic networks had measured.

Chapter 38

Dynamics: Differential Equations and Generative Flows

Every network in this book was trained by taking small steps downhill, and a residual network adds a small update to its input at every layer. Both are step-by-step solutions of a differential equation, which says how fast a state changes from where it is now. A diffusion model turns Gaussian noise into an image the same way: each call to a network returns a direction, and the sample moves a short step along it. How few steps can such a model take, and where does the direction come from when all we have are example images?

A cloud of noisy particles is unpredictable one particle at a time, yet its density changes deterministically, and running that change backward turns noise into data. The tools of the Basics part stop short of this. Backpropagation differentiates a fixed computation, not the solution of an equation. Probability describes a random variable at one time, not a density that moves. And noise whose size grows like the square root of time breaks the ordinary chain rule.

Three results carry the chapter. The eigenvalues of an equation limit the step that can follow it, learning rates included (Section 38.1.3.3). The density of a noisy cloud can be run backward once one field, the score, is known (Section 38.4). Least squares on noisy samples learns that field, or a velocity in its place (Section 38.5, Section 38.6). Beside these, a linear equation with an input solves to the recurrence of a state space model (Section 38.1.2.3, Section 12.2), Section 38.2 makes the equation itself the model, the limit of a residual network (Section 7.4) as its steps shrink, and Section 38.3 supplies the calculus of noise. A reader headed for diffusion models can skip the adjoint method and optimal transport.

This chapter holds the proofs for Chapter 17, which points here instead of deriving its results. Small results, such as the regression target, appear in both. The long derivations (time reversal, the probability-flow ODE, the theorems behind score and flow matching) are only here, and that chapter can be read without them.

We assume eigendecompositions (Section 33.3), Jacobians (Section 34.2), Gaussians and conditional expectation, and no measure theory.

Resources and Further Reading

The following references accompany the material this chapter teaches; topics it does not teach are collected in Section 38.7.

Books and lecture notes

- Nonlinear Dynamics and Chaos — Steven Strogatz: a geometric introduction to flows, fixed points and their stability, the material of Section 38.1.
- On Neural Differential Equations — Patrick Kidger (2022): the book-length treatment of neural ODEs, controlled and stochastic differential equations, with the solver and adjoint machinery of Section 38.2 laid out in full.
- Applied Stochastic Differential Equations — Särkkä & Solin: an application-minded treatment of Itô calculus, SDEs and the Fokker–Planck equation for Section 38.3 and Section 38.4; the authors provide the full PDF.
- Stochastic Differential Equations: An Introduction with Applications — Bernt Øksendal: a rigorous reference for Brownian motion, the Itô integral and existence theory, behind Section 38.3.
- Numerical Solution of Stochastic Differential Equations — Kloeden & Platen: the reference on Euler–Maruyama, Milstein and higher-order schemes and their strong and weak orders, for the simulation part of Section 38.3.
- MIT 6.S184 Lecture Notes: Flow Matching and Diffusion Models — Holderrieth & Erives: self-contained notes that derive diffusion and flow matching from SDEs and the continuity equation, companions to Section 38.5 and Section 38.6.

Courses and tutorials

- MIT 6.S184: Generative AI with Stochastic Differential Equations: lectures, slides and labs that build a diffusion model from first principles, following the route of Section 38.4 to Section 38.6; openly licensed.
- Understanding Diffusion Models: A Unified Perspective — Calvin Luo (2022): derives the variational and score-based views of diffusion side by side, for Section 38.5.
- Flow Matching Guide and Code — Lipman et al. (2024): a detailed reference on flow matching for Section 38.6, paired with the `facebookresearch/flow_matching` PyTorch library. It runs time from noise at 0 to data at 1, as do the MIT notes and the flow-matching papers below; this chapter runs it the other way (Figure 38.25).

Foundational papers

- Neural Ordinary Differential Equations — Chen et al. (NeurIPS 2018): parameterizes a network's dynamics as an ODE and trains it with the adjoint method, as in Section 38.2.
- Generative Modeling by Estimating Gradients of the Data Distribution — Song & Ermon (NeurIPS 2019): score-based generation with denoising score matching and annealed Langevin sampling, for Section 38.5.
- Denoising Diffusion Probabilistic Models — Ho, Jain & Abbeel (2020): the discrete-time DDPM formulation that Section 38.5 reads as a discretized SDE.
- Score-Based Generative Modeling through SDEs — Song et al. (ICLR 2021): unifies score matching and diffusion as forward and reverse SDEs and introduces the probability-flow ODE of Section 38.4.
- Denoising Diffusion Implicit Models — Song, Meng & Ermon (ICLR 2021): the deterministic sampler of Section 38.5, which reuses a trained DDPM with fewer steps.
- Flow Matching for Generative Modeling — Lipman et al. (2022): simulation-free training of continuous flows by regressing conditional vector fields, the method of Sec-

tion 38.6.
- Flow Straight and Fast: Rectified Flow — Liu, Gong & Liu (2022): straight interpolation paths and reflow for few-step sampling, for Section 38.6.

Blogs and visual explainers

- Generative Modeling by Estimating Gradients of the Data Distribution — Yang Song: the author's walkthrough connecting score matching, Langevin dynamics and the SDE picture of Section 38.5, with figures.
- What are Diffusion Models? — Lilian Weng: a regularly updated survey of diffusion from DDPM through score SDEs and guidance, for Section 38.5.
- Perspectives on Diffusion — Sander Dieleman: shows how a diffusion model is at once a denoiser, a score estimator, a reverse-SDE solver and a flow, the dictionary of Section 38.6.
- The Annotated Diffusion Model — Hugging Face: a line-by-line PyTorch implementation of DDPM that connects the mathematics of Section 38.5 to runnable code.

38.1 Ordinary Differential Equations and Numerical Solvers

Anyone who has trained a network has seen a run blow up because the learning rate was too large. It is tempting to blame the loss surface, with its sharp valleys and its plateaus. Yet the same thing happens on the simplest loss there is. Gradient descent on the bowl $\mathcal{L}(x) = 25x^2$ takes the step $x \leftarrow x - \eta \cdot 50x$, and for every learning rate above $\eta = 0.04$ the iterates grow without bound, although the bowl has a single minimum and nothing about it is difficult. If the loss is not the problem, what is?

The answer comes from an older subject. Gradient descent is a step-by-step solution of the differential equation $\dot{x} = -50x$, which says: move downhill at a speed proportional to the slope. Its exact solution $x(t) = e^{-50t}x_0$ decays to zero from every start. Euler proposed in 1768 to solve such an equation by following the arrow for a short time h and then reading it again, and the learning rate plays the role of h. One step multiplies x by $1 - 50h$. With $h = 0.01$ the factor is 0.5, and the computed solution decays like the true one. With $h = 0.05$ it is -1.5, and the computed solution flips sign and grows. The equation is stable; the steps are not. The idea of this section is that the eigenvalues of a differential equation decide both how its exact motion behaves and how large a step can follow it.

Deep learning meets this gap between an equation and its steps in three places. Gradient descent (Section 9.2) is the first. A residual network (Section 7.4) adds a small update to its input at every layer, which is one such step. Diffusion and flow-matching models (Section 17.4) generate an image by following a learned velocity field from noise to data, one network evaluation per step, so the number of steps is the cost of a sample.

The text has two parts. The first asks when a differential equation has exactly one solution and solves the linear case completely. The second (Section 38.1.3) asks how accurate and how stable a step-by-step solution is. Three results carry the rest of the chapter: a Lipschitz field has exactly one trajectory through each point, so its flow is invertible; the real parts of the eigenvalues of a linear field decide whether its motion decays; and Euler's method is stable only while $h\lambda$ stays in a disc. The pendulum and transient growth can be skipped on a first reading. The chapter uses the symbols below throughout.

Table 38.1: Notation for the dynamics chapter. The later rows belong to later sections, which define them where they are first used.

Symbol	Meaning	First used in
$\mathbf{x}(t) \in \mathbb{R}^d$, $\dot{\mathbf{x}}$	state, its time derivative	ODEs and solvers
$\mathbf{f}(\mathbf{x}, t)$	vector field of an ODE; drift of an SDE	ODEs and solvers; SDEs
Φ_t	flow map, $\Phi_t(\mathbf{x}_0) = \mathbf{x}(t)$	ODEs and solvers
L	Lipschitz constant of $\mathbf{f}$	ODEs and solvers
$\mathbf{A}$; $\lambda_j = a_j + \mathrm{i}\, b_j$; $\mathbf{V}, \Lambda$	linear field; its eigenvalues; eigenvectors and their diagonal	ODEs and solvers
$\mathbf{K}$	rotation generator $\left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$	ODEs and solvers
$\mathbf{J}$	Jacobian of $\mathbf{f}$	ODEs and solvers
h, $t_n = nh$, $\mathbf{x}_n$	step size, time grid, numerical iterate	ODEs and solvers
$R(z)$, $z = h\lambda$	stability function of a solver	ODEs and solvers
ε_n	global error $\lVert \mathbf{x}_n - \mathbf{x}(t_n) \rVert$	ODEs and solvers
$\mathcal{L}$, η, $\mathbf{H}$	loss, learning rate, Hessian	ODEs and solvers
c; T	damping coefficient; time horizon	ODEs and solvers
θ	network parameters	Neural ODEs
$\mathbf{a}(t)$	adjoint state	Neural ODEs
p_t	density of the state at time t	Neural ODEs
$\mathbf{W}_t$	Brownian motion	SDEs
$g(t)$; $\mathbf{G}$	diffusion coefficient; noise matrix	SDEs
θ, g	OU rate and noise amplitude, $dX = -\theta X\, dt + g\, dW$	SDEs
α_t, σ_t	mean coefficient and standard deviation of $\mathbf{X}_t$ given $\mathbf{X}_0$	SDEs
$\mathbf{x}_0$; ϵ	data endpoint at $t = 0$; noise endpoint at the final time, $t = 1$ from flow matching on	SDEs
$\mathbf{s}_t = \nabla \log p_t$; $\mathbf{s}_\theta$	score; learned score	Fokker–Planck
$\mathbf{u}_t$; $\mathbf{u}_\theta$	true (probability-flow) velocity; learned velocity	Fokker–Planck
$\mathbf{j}$	probability current	Fokker–Planck
$\bar{t} = T - t$	reversed clock	Fokker–Planck
w_k, $r_k(\mathbf{x})$, v_k	mixture weights, responsibilities, component variances	Fokker–Planck
β_t, $\bar{\alpha}_t$	DDPM noise schedule	Score matching
$w(t)$; γ	loss weighting over noise levels; guidance scale	Score matching
π	coupling	Flow matching

Symbol	Meaning	First used in
i, $\mathbf{I}$, D_{KL}	imaginary unit (upright), identity, KL divergence	throughout

38.1.1 Vector Fields and Flows

A differential equation describes a motion by prescribing a velocity at every point. Two motions can be solved by hand, and we begin with them.

38.1.1.1 Decay, Rotation, and the Spiral

The first is *decay*. The rule "move toward zero at a speed proportional to the distance" is the equation $\dot{x} = -x$, where the dot is the derivative in time. Its solution from $x(0) = x_0$ is

$$x(t) = e^{-t} x_0,$$

as differentiating shows: $\dot{x} = -e^{-t} x_0 = -x$. Each unit of time multiplies the distance to the origin by $e^{-1} \approx 0.37$.

The second is *rotation*. In the plane, let the velocity be the position turned by a right angle,

$$\dot{\mathbf{x}} = \mathbf{K}\mathbf{x}, \qquad \mathbf{K} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix},$$

that is $\dot{x}_1 = -x_2$ and $\dot{x}_2 = x_1$. The velocity is perpendicular to the position, because $\mathbf{x}^{\top}\mathbf{K}\mathbf{x} = -x_1 x_2 + x_2 x_1 = 0$, so the length of $\mathbf{x}$ never changes:

$$\frac{d}{dt}\|\mathbf{x}\|^2 = 2\,\mathbf{x}^{\top}\dot{\mathbf{x}} = 2\,\mathbf{x}^{\top}\mathbf{K}\mathbf{x} = 0.$$

A point moves on a circle, and since its speed $\|\mathbf{K}\mathbf{x}\|$ equals its radius $\|\mathbf{x}\|$ it turns at one radian per unit of time: one revolution takes 2π.

Adding the two motions gives the field we use as the running example of the chapter,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \qquad \mathbf{A} = -\tfrac{1}{2}\mathbf{I} + \mathbf{K} = \begin{pmatrix} -0.5 & -1 \\ 1 & -0.5 \end{pmatrix}.$$

The rotation part does not change the length and the decay part shrinks it: $\frac{d}{dt}\|\mathbf{x}\|^2 = 2\mathbf{x}^{\top}\mathbf{A}\mathbf{x} = -\|\mathbf{x}\|^2$, so $\|\mathbf{x}(t)\| = e^{-t/2}\|\mathbf{x}_0\|$. A point decays at rate $\frac{1}{2}$ while it turns at unit speed: it spirals into the origin (Figure 38.1). The eigenvalues of $\mathbf{A}$ are $-0.5 \pm \mathrm{i}$. Their real part gives the decay rate and their imaginary part the angular speed. Section 38.1.2 shows that this holds in general: a linear field is a sum of decays, growths and rotations, one for each eigenvalue.

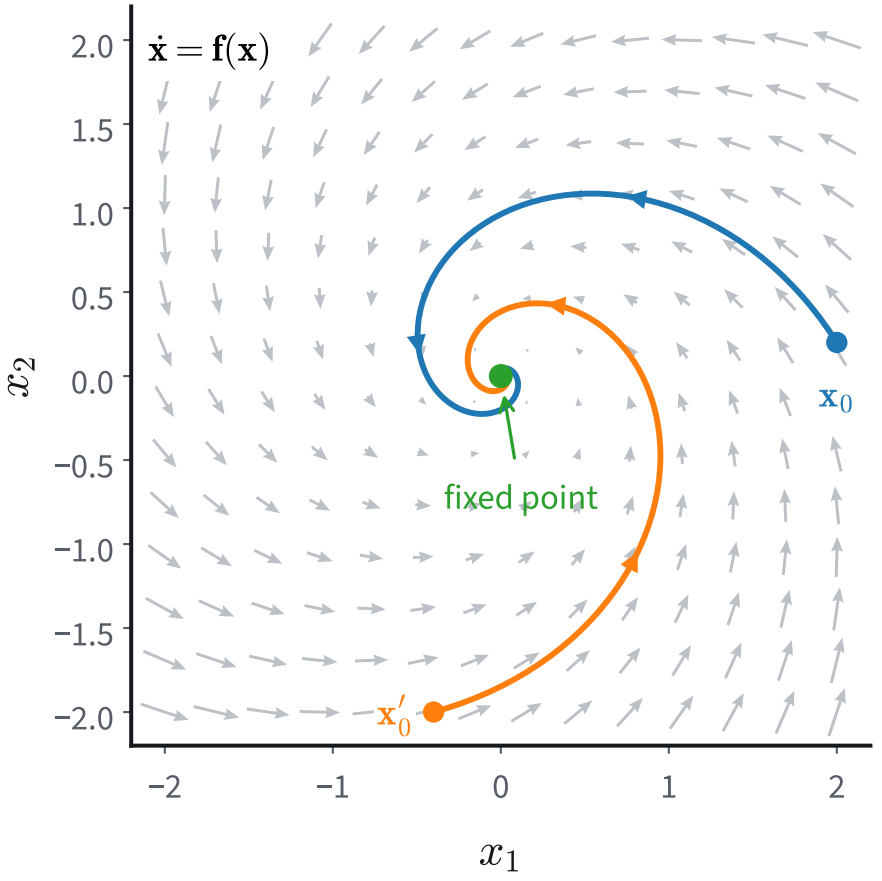


Figure 38.1: An ODE as a velocity field. The faint grid arrows are the field $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$ with $\mathbf{A} = \left(\begin{smallmatrix} -0.5 & -1 \\ 1 & -0.5 \end{smallmatrix}\right)$; the two coloured curves are trajectories that always follow the local arrow, spiralling into the stable fixed point at the origin (eigenvalues $-0.5 \pm \mathrm{i}$).

38.1.1.2 The Initial-Value Problem

Most fields are not linear, so we need a form that covers every rule of motion. The general problem gives a velocity rule and a starting point:

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), t), \qquad \mathbf{x}(0) = \mathbf{x}_0. \tag{38.1}$$

Here $\mathbf{x}(t) \in \mathbb{R}^d$ is the state. The function $\mathbf{f} : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$ is a **vector field**: it attaches an arrow to every point at every time. A **trajectory** is a differentiable path whose tangent at each instant is the arrow at its current position.

The vector $\mathbf{x}_0$ is the **initial value**, and Equation 38.1 is called an **initial-value problem**. When $\mathbf{f}$ does not depend on t, the system is **autonomous**: the arrows stay fixed and only the state moves. Decay, rotation and the spiral are the autonomous fields $\mathbf{f}(\mathbf{x}) = -\mathbf{x}$, $\mathbf{Kx}$ and $\mathbf{Ax}$.

Integrating both sides from 0 to t (Section 34.6) gives the equivalent **integral form**

$$\mathbf{x}(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\mathbf{x}(s), s)\, ds. \tag{38.2}$$

The solution is the start plus the accumulated velocity. The existence proof below works with the integral form, and Euler's method (Section 38.1.3) is the left-endpoint Riemann sum of this integral.

38.1.1.3 Existence and Uniqueness

Does an initial-value problem have a solution, and only one? For a model defined by a field, uniqueness is what makes the output a function of the input, and it can fail.

Consider $\dot{x} = \sqrt{|x|}$ from $x_0 = 0$. The constant path $x \equiv 0$ solves it. So does, for every waiting time $c \geq 0$,

$$x_c(t) = \begin{cases} 0, & t \leq c, \\ (t-c)^2/4, & t > c, \end{cases}$$

since $\dot{x}_c = (t-c)/2 = \sqrt{x_c}$ for $t > c$. A particle at rest may leave at any time (Figure 38.2), so a model defined by this field does not assign one output to its input. The cause is the slope of $\sqrt{|x|}$, which is unbounded at 0: a small displacement x produces the much larger velocity $\sqrt{|x|}$.

For a well-behaved field a solution can be built by iterating the integral form Equation 38.2: guess a path, compute the path the field would produce from it, and repeat. This is **Picard iteration**. For $\dot{x} = x$ with $x_0 = 1$, start from the constant path $\varphi_0 \equiv 1$ and set $\varphi_{k+1}(t) = 1 + \int_0^t \varphi_k(s)\, ds$:

$$\varphi_1(t) = 1 + t, \qquad \varphi_2(t) = 1 + t + \tfrac{t^2}{2}, \qquad \varphi_3(t) = 1 + t + \tfrac{t^2}{2} + \tfrac{t^3}{6}.$$

The iterates converge to the solution e^t. Here each iteration adds one Taylor term of e^t; for other fields the iterates look different, but they still converge, provided the field does

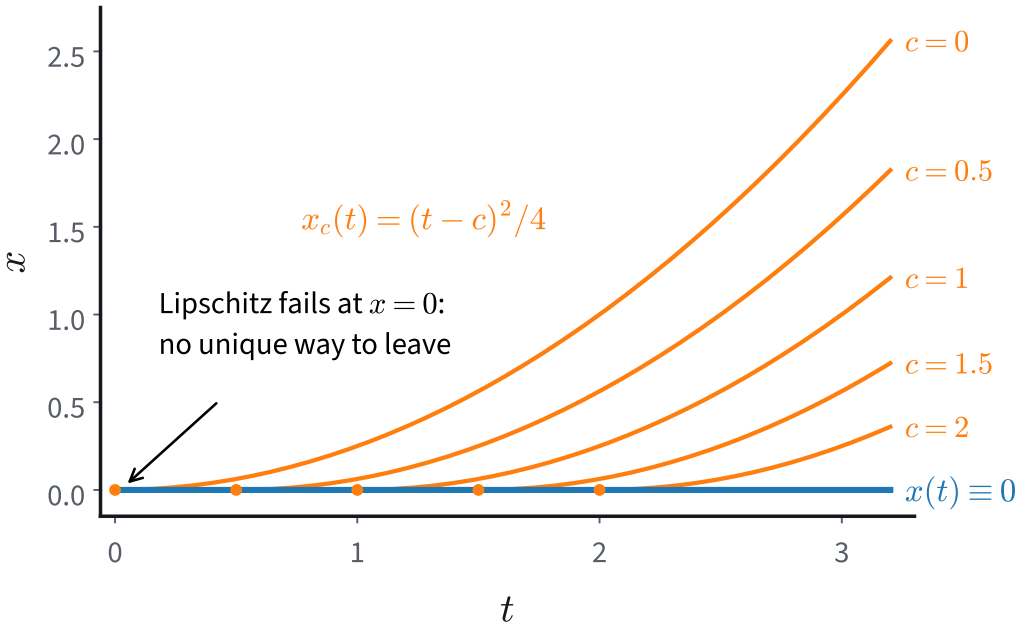


Figure 38.2: Solutions of $\dot{x} = \sqrt{|x|}$ through $x(0) = 0$: the constant solution $x \equiv 0$ and a parabolic solution $x_c(t) = (t-c)^2/4$ for every departure time c. Because the slope of $\sqrt{|x|}$ is unbounded at the origin, the Lipschitz condition fails and the solution is not unique.

not change too abruptly from point to point. The next proposition makes "too abruptly" precise.

Proposition (Picard–Lindelöf). *Let $\mathbf{f}(\mathbf{x}, t)$ be continuous in t and globally Lipschitz in $\mathbf{x}$: there is an L with*

$$\|\mathbf{f}(\mathbf{x}, t) - \mathbf{f}(\mathbf{y}, t)\| \le L \, \|\mathbf{x} - \mathbf{y}\| \qquad \text{for all } \mathbf{x}, \mathbf{y}, t.$$

Then the initial-value problem Equation 38.1 has exactly one solution on $[0, T]$, for every $\mathbf{x}_0$ and every $T > 0$ (Coddington and Levinson 1955).

This is the result of the first part that everything later leans on. The fan is what happens when no such L exists near 0. The proof shows that the Picard iterates converge, and that they converge to the only solution.

Proof. A continuous path φ on $[0, \delta]$ solves the problem if and only if it is a fixed point of the **Picard operator**

$$(P\varphi)(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\varphi(s), s) \, ds.$$

In words, P replaces a guessed path by the path the field would produce from it. (A solution satisfies the integral form; conversely, a continuous fixed point is differentiable by the fundamental theorem of calculus, with the right derivative and the right start.)

The step that is not routine pulls the Lipschitz bound through the integral. For two paths on $[0,\delta]$, write $d(\varphi,\psi)=\max_{t\le\delta}\|\varphi(t)-\psi(t)\|$. Then for $t\le\delta$,

$$\|(P\varphi)(t)-(P\psi)(t)\|\le\int_0^t L\,\|\varphi(s)-\psi(s)\|\,ds\le L\,\delta\;d(\varphi,\psi).$$

With $\delta=1/(2L)$, every application of P halves the distance between two candidate paths.

Start from the constant path $\varphi_0\equiv\mathbf{x}_0$ and set $\varphi_{k+1}=P\varphi_k$. Consecutive iterates then differ by at most $2^{-k}d(\varphi_1,\varphi_0)$, and these differences add up like a geometric series. So the iterates converge uniformly on $[0,\delta]$, to a path φ. We take one fact from analysis on trust here: a uniform limit of continuous paths is continuous. Since $d(P\varphi_k,P\varphi)\le\frac{1}{2}d(\varphi_k,\varphi)\to 0$, the limit satisfies $\varphi=P\varphi$: it is a solution. Two solutions φ,ψ would satisfy $d(\varphi,\psi)=d(P\varphi,P\psi)\le\frac{1}{2}d(\varphi,\psi)$, so they coincide.

Finally restart at $t=\delta$ from $\varphi(\delta)$. The step length $1/(2L)$ does not depend on where the solution is, because L is the same everywhere, so $\lceil 2LT\rceil$ restarts cover $[0,T]$, with one solution on each piece. ■

The theorem asks for a Lipschitz bound and for that bound to hold globally, and each requirement rules out a different failure. The fan shows what happens without a Lipschitz bound. Without the global bound, the solution can blow up. The field $\dot{x}=x^2$ is smooth, and from $x_0>0$ separation of variables gives

$$x(t)=\frac{x_0}{1-x_0t},$$

which reaches infinity at the finite time $t=1/x_0$. The proof shows why the theorem does not cover it. On the interval $|x|\le\rho$ the field is Lipschitz with $L=2\rho$, since $|x^2-y^2|=|x+y|\,|x-y|$. The restart length $1/(2L)=1/(4\rho)$ therefore shrinks as the solution grows, and the restarts can add up to a finite time. A field that is Lipschitz only on bounded sets guarantees a solution only for a while.

38.1.1.4 The Flow Map

A generative model moves a whole cloud of starting points at once, so we now fix the field and vary the start. For an autonomous, globally Lipschitz field, write

$$\Phi_t(\mathbf{x}_0)=\mathbf{x}(t)$$

for the state at time t of the one trajectory started at $\mathbf{x}_0$. This **flow map** $\Phi_t:\mathbb{R}^d\to\mathbb{R}^d$ moves all of space forward by t along the arrows.

For decay the flow map is $\Phi_t(x)=e^{-t}x$. Flowing for s and then for t gives $\Phi_t(\Phi_s(x))=e^{-t}e^{-s}x=e^{-(t+s)}x=\Phi_{t+s}(x)$, and $\Phi_{-t}(x)=e^{t}x$ undoes Φ_t. The same law holds for every autonomous field:

$$\Phi_0=\mathrm{id},\qquad \Phi_{t+s}=\Phi_t\circ\Phi_s. \tag{38.3}$$

The reason is that the arrows do not change with time: the path $\tau \mapsto \mathbf{x}(s+\tau)$ starts at $\Phi_s(\mathbf{x}_0)$ and follows the same arrows, so by uniqueness it is the trajectory from that point. The decay example suggests that the law also holds for negative times. It does, because running a field backward is the same theorem applied to a second field.

Corollary (the flow map is a bijection). *Let $\mathbf{f}(\mathbf{x})$ be autonomous and globally Lipschitz. Then Φ_t exists for every $t \in \mathbb{R}$, and each Φ_t is a bijection of $\mathbb{R}^d$ with inverse Φ_{-t}.*

Proof. The field $\mathbf{g}(\mathbf{y}) = -\mathbf{f}(\mathbf{y})$ is Lipschitz with the same L. A path $\mathbf{x}(t)$ solves $\dot{\mathbf{x}} = \mathbf{f}$ on $[-T, 0]$ if and only if $\mathbf{y}(s) = \mathbf{x}(-s)$ solves $\dot{\mathbf{y}} = \mathbf{g}$ on $[0, T]$, since $\dot{\mathbf{y}}(s) = -\dot{\mathbf{x}}(-s) = -\mathbf{f}(\mathbf{y}(s))$. The theorem applied to $\mathbf{g}$ gives a unique solution backward in time for every start, so Φ_t is defined for all real t. The composition law Equation 38.3, whose argument did not use the sign of s or t, then gives $\Phi_{-t} \circ \Phi_t = \Phi_t \circ \Phi_{-t} = \Phi_0 = \mathrm{id}$. ∎

One consequence is that two different trajectories never meet: a meeting point would have two different backward solutions. (For a time-dependent field the same argument applies to $\mathbf{g}(\mathbf{y}, s) = -\mathbf{f}(\mathbf{y}, -s)$.)

An invertible map from noise to data whose inverse can be computed is what a generative flow needs, and the fields of many neural networks qualify. A field built from linear maps and Lipschitz activations, such as $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$, is globally Lipschitz, with L at most the product of the weight norms and the activations' constants ($\|\mathbf{W}_2\|\,\|\mathbf{W}_1\|$ here). The model of Section 38.2 is therefore well-posed, and by the corollary its time-T map is a bijection.

38.1.2 Linear ODEs and Stability

Linear fields are the one family we can solve completely, and they describe most other fields near a point where the field vanishes.

38.1.2.1 The Matrix Exponential

The scalar equation $\dot{x} = ax$ is solved by $x(t) = e^{at} x_0$. For a linear system

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}, \qquad \mathbf{x}(0) = \mathbf{x}_0, \qquad \mathbf{A} \in \mathbb{R}^{d \times d},$$

we look for the same form with a matrix in the exponent. For any square matrix $\mathbf{C}$, define the **matrix exponential** by the power series that defines e^z (stated without proof in Section 33.3):

$$e^{\mathbf{C}} = \sum_{k=0}^{\infty} \frac{\mathbf{C}^k}{k!} = \mathbf{I} + \mathbf{C} + \frac{\mathbf{C}^2}{2} + \frac{\mathbf{C}^3}{6} + \cdots. \tag{38.4}$$

The series converges for every matrix: term by term, $\|\mathbf{C}^k\| \le \|\mathbf{C}\|^k$ (the operator norm is submultiplicative), so it is dominated by the scalar series for $e^{\|\mathbf{C}\|}$, which converges.

Proposition (the matrix exponential solves linear ODEs). $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$ *is the unique solution of* $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$, $\mathbf{x}(0) = \mathbf{x}_0$.

Proof. Differentiate the series Equation 38.4 with $\mathbf{C} = \mathbf{A}t$ term by term (legitimate: on any bounded t-interval the differentiated series still converges uniformly, by the same domination):

$$\frac{d}{dt}e^{\mathbf{A}t} = \sum_{k=1}^{\infty} \frac{\mathbf{A}^k\, t^{k-1}}{(k-1)!} = \mathbf{A}\sum_{j=0}^{\infty} \frac{(\mathbf{A}t)^j}{j!} = \mathbf{A}\, e^{\mathbf{A}t}.$$

So $\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0$ satisfies the ODE, and at $t = 0$ the series collapses to $\mathbf{I}$, so $\mathbf{x}(0) = \mathbf{x}_0$. Uniqueness is Picard–Lindelöf: the field $\mathbf{x} \mapsto \mathbf{A}\mathbf{x}$ is Lipschitz with $L = \|\mathbf{A}\|$. $\blacksquare$

An infinite series of matrix powers is not yet a formula we can read. The eigendecomposition turns it into d scalar exponentials. Recall that if $\mathbf{A}$ has d independent eigenvectors $\mathbf{v}_j$, $\mathbf{A}\mathbf{v}_j = \lambda_j \mathbf{v}_j$, then with $\mathbf{V} = (\mathbf{v}_1, \dots, \mathbf{v}_d)$ and $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_d)$ we have $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$ (Section 33.3).

Powers then telescope, $(\mathbf{A}t)^k = \mathbf{V}(\Lambda t)^k\mathbf{V}^{-1}$, because each adjacent $\mathbf{V}^{-1}\mathbf{V}$ cancels, and summing the series Equation 38.4 gives

$$e^{\mathbf{A}t} = \mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}, \qquad e^{\Lambda t} = \operatorname{diag}\!\left(e^{\lambda_1 t}, \dots, e^{\lambda_d t}\right). \tag{38.5}$$

The formula is a decoupling statement. Expand the start in the eigenbasis, $\mathbf{x}_0 = \sum_j c_j \mathbf{v}_j$; then

$$\mathbf{x}(t) = \sum_{j=1}^{d} c_j\, e^{\lambda_j t}\, \mathbf{v}_j.$$

Along each eigendirection the system is the scalar equation $\dot{x} = \lambda_j x$, evolving independently of the others. Each term $c_j e^{\lambda_j t}\mathbf{v}_j$ is called a **mode** of the system.

For the spiral we can do better than the general formula. Because $\mathbf{A} = -\tfrac{1}{2}\mathbf{I} + \mathbf{K}$ and $\mathbf{I}$ commutes with $\mathbf{K}$, the exponential splits, $e^{\mathbf{A}t} = e^{-t/2}e^{\mathbf{K}t}$. Because $\mathbf{K}^2 = -\mathbf{I}$, the powers of $\mathbf{K}$ cycle through $\mathbf{I}, \mathbf{K}, -\mathbf{I}, -\mathbf{K}$, and the series for $e^{\mathbf{K}t}$ splits into even and odd terms, the series of $\cos t$ and $\sin t$:

$$e^{\mathbf{A}t} = e^{-t/2}\left(\cos t\, \mathbf{I} + \sin t\, \mathbf{K}\right) = e^{-t/2}\begin{pmatrix} \cos t & -\sin t \\ \sin t & \cos t \end{pmatrix}.$$

The flow shrinks by $e^{-t/2}$ and turns by t radians, as we found from the field itself. From $\mathbf{x}_0 = (2, 0)$ at $t = 2$ the state is $e^{-1}(2\cos 2,\ 2\sin 2) = (-0.306,\ 0.669)$, with length $2e^{-1} = 0.736$. For this $\mathbf{A}$ the eigenvectors and eigenvalues $-0.5 \pm \mathrm{i}$ are complex, but the product $\mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}$ is real.

A word of caution about the split of the exponential. The rule $e^{\mathbf{P}+\mathbf{Q}} = e^{\mathbf{P}}e^{\mathbf{Q}}$ holds when $\mathbf{P}\mathbf{Q} = \mathbf{Q}\mathbf{P}$, since then $(\mathbf{P}+\mathbf{Q})^k$ expands by the binomial theorem. For matrices that do not commute it fails (6).

There is a third route to $e^{\mathbf{A}t}$: the compound-interest limit $(\mathbf{I} + t\mathbf{A}/n)^n$, which needs no series and no eigenvectors. The cell computes all three for the spiral at $t = 2$. Before running it, guess how fast the compounding converges in n.

```
A = np.array([[-0.5, -1.0], [1.0, -0.5]])      # spiral: eigenvalues -0.5 +- i
t = 2.0
lam, V = np.linalg.eig(A)                      # A = V diag(lam) V^{-1}
expAt_eig = (V @ np.diag(np.exp(lam * t)) @ np.linalg.inv(V)).real
term, expAt_series = np.eye(2), np.eye(2)      # identity plus 29 terms (At)^k / k!
for k in range(1, 30):
    term = term @ (A * t) / k
    expAt_series += term
x0 = np.array([2.0, 0.0])
print('eigenvalues of A:', lam.round(4))
print(f'|series - eigen formula| = {np.abs(expAt_series - expAt_eig).max():.1e}')
for n in [2 ** 10, 2 ** 16, 2 ** 24]:          # compounding (I + tA/n)^n
    expAt_n = np.linalg.matrix_power(np.eye(2) + (t / n) * A, n)
    err = np.abs(expAt_n - expAt_eig).max()
    print(f'n = 2^{int(np.log2(n))}: |(I+tA/n)^n - eigen| = {err:.1e}, '
          f'n * error = {n * err:.2f}')
print('x(2) =', (expAt_eig @ x0).round(4))
print(f'||x(2)|| = {np.linalg.norm(expAt_eig @ x0):.6f},  '
      f'e^(-t/2) ||x0|| = {np.exp(-0.5 * t) * np.linalg.norm(x0):.6f}')
```

```
eigenvalues of A: [-0.5+1.j -0.5-1.j]
|series - eigen formula| = 5.6e-17
n = 2^10: |(I+tA/n)^n - eigen| = 8.8e-04, n * error = 0.90
n = 2^16: |(I+tA/n)^n - eigen| = 1.4e-05, n * error = 0.90
n = 2^24: |(I+tA/n)^n - eigen| = 5.4e-08, n * error = 0.90
x(2) = [-0.3062  0.669 ]
||x(2)|| = 0.735759,  e^(-t/2) ||x0|| = 0.735759
```

The series and the eigendecomposition agree to rounding, and the state $(-0.3062,\ 0.6690)$ and its length 0.735759 are the values computed by hand. The compounding is far slower: its error times n stays at 0.90 as n grows from 2^{10} to 2^{24}, so the error falls only like $1/n$. This is compound interest with a matrix interest rate, and it is also n steps of Euler's method, whose error of order $1/n$ is proved in Section 38.1.3.

38.1.2.2 Stability from Eigenvalues

The decoupled form tells us at a glance whether a linear system settles down. Write each eigenvalue as $\lambda_j = a_j + \mathrm{i}\, b_j$. Its mode has magnitude $|e^{\lambda_j t}| = e^{a_j t}$, since $e^{\mathrm{i}b_j t} = \cos b_j t + \mathrm{i} \sin b_j t$ has modulus one. The real part sets growth or decay; the imaginary part sets rotation. For a diagonalizable $\mathbf{A}$ this gives the following classification.

- If $\operatorname{Re}\lambda_j < 0$ for all j, every mode decays, and $\mathbf{x}(t) \to \mathbf{0}$ from every start: the origin is **asymptotically stable**. The slowest mode sets the decay rate $-\max_j \operatorname{Re}\lambda_j$.
- If $\operatorname{Re}\lambda_j > 0$ for some j, that mode grows exponentially, and every start with a component along $\mathbf{v}_j$ escapes: the origin is **unstable**.

A complex pair $a \pm \mathrm{i}b$ is the spiral with general rates: a rotation at angular speed b that decays or grows with the sign of a. To see this, write the eigenvector of $\lambda = a + \mathrm{i}b$ as $\mathbf{v} = \mathbf{p} + \mathrm{i}\mathbf{q}$ with real $\mathbf{p}, \mathbf{q}$ (independent, since $b \neq 0$). The real and imaginary parts of $\mathbf{A}\mathbf{v} = (a + \mathrm{i}b)\mathbf{v}$ are

$$\mathbf{A}\mathbf{p} = a\mathbf{p} - b\mathbf{q}, \qquad \mathbf{A}\mathbf{q} = b\mathbf{p} + a\mathbf{q}.$$

So $\mathbf{A}$ maps the plane of $\mathbf{p}$ and $\mathbf{q}$ to itself, and in the basis $(\mathbf{q}, \mathbf{p})$ it is the matrix $a\mathbf{I} + b\mathbf{K}$. The computation for the spiral then gives

$$e^{(a\mathbf{I}+b\mathbf{K})t} = e^{at}\,(\cos bt\,\mathbf{I} + \sin bt\,\mathbf{K})\,:$$

scaling by e^{at} composed with rotation by the angle bt. The same block, iterated in discrete time instead of flowed, appears in Section 33.3.1.6.

In two dimensions, and for $\det \mathbf{A} \neq 0$, the signs of the eigenvalues give six standard pictures.

eigenvalues of $\mathbf{A}$ (2×2)	portrait	behavior
real, both negative	stable node	monotone decay
real, both positive	unstable node	monotone growth
real, opposite signs	saddle	decay in, escape out
complex pair, $\operatorname{Re}\lambda < 0$	stable spiral	decaying oscillation
complex pair, $\operatorname{Re}\lambda > 0$	unstable spiral	growing oscillation
purely imaginary pair	center	closed orbits

Figure 38.3 draws one field of each kind; the stable spiral is the field of Figure 38.1.

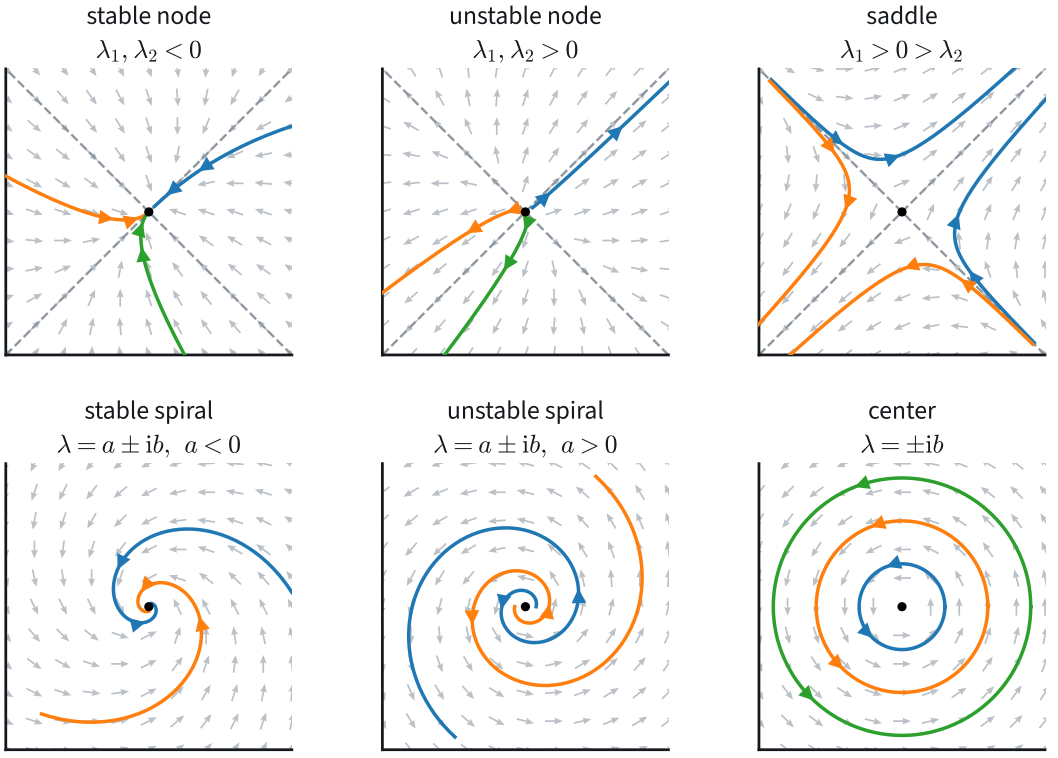


Figure 38.3: Six planar linear systems, one per row of the table; dashed lines in the node and saddle panels are the eigendirections, along which trajectories enter or leave.

When $\mathbf{A}$ is not diagonalizable, the modes acquire polynomial factors $t^k e^{\lambda t}$, which change the verdict only for eigenvalues on the imaginary axis.

38.1.2.3 Linear Systems with an Input

So far nothing pushed the system from outside. The state of a state space model that reads a sequence (Section 12.2) obeys a linear equation with an input $\mathbf{u}(t)$,

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}(t), \qquad \mathbf{x}(0) = \mathbf{x}_0.$$

The matrix exponential solves this equation as well. Since $\frac{d}{dt} e^{-\mathbf{A}t} = -e^{-\mathbf{A}t}\mathbf{A}$, the product rule gives $\frac{d}{dt}\big(e^{-\mathbf{A}t}\mathbf{x}\big) = e^{-\mathbf{A}t}(\dot{\mathbf{x}} - \mathbf{A}\mathbf{x}) = e^{-\mathbf{A}t}\mathbf{B}\mathbf{u}$, and the right-hand side no longer contains $\mathbf{x}$. Integrating from 0 to t and multiplying by $e^{\mathbf{A}t}$ yields the **variation of constants** formula

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}_0 + \int_0^t e^{\mathbf{A}(t-s)}\,\mathbf{B}\mathbf{u}(s)\,ds. \tag{38.6}$$

The state is the decayed start plus every past input, each decayed for the time since it arrived. Now suppose that the input is constant during a step, $\mathbf{u}(s) = \mathbf{u}_n$ for $s \in [t_n, t_n + h)$, which is called a *zero-order hold*. The integral over one step can then be done once and for all, and the states at the grid times obey a linear recurrence without any approximation:

$$\mathbf{x}_{n+1} = \bar{\mathbf{A}}\,\mathbf{x}_n + \bar{\mathbf{B}}\,\mathbf{u}_n, \qquad \bar{\mathbf{A}} = e^{\mathbf{A}h}, \qquad \bar{\mathbf{B}} = \int_0^h e^{\mathbf{A}\tau}\,d\tau\;\mathbf{B} = \mathbf{A}^{-1}\big(e^{\mathbf{A}h} - \mathbf{I}\big)\mathbf{B}, \tag{38.7}$$

where the last form needs $\mathbf{A}$ to be invertible. In one dimension these are $\bar{a} = e^{h\lambda}$ and $\bar{b} = (e^{h\lambda} - 1)\,b/\lambda$, the coefficients on which the state space models of Section 12.2.3.2 are built (there the step is Δ and the state is h). The point to remember is that $|e^{h\lambda}| = e^{h\,\mathrm{Re}\,\lambda} < 1$ for every step $h > 0$ whenever $\mathrm{Re}\,\lambda < 0$: the exact recurrence is stable whenever the equation is. On $\dot{x} = -50x$ with $h = 0.05$ the exact factor is $e^{-2.5} = 0.08$, where the opening's Euler step had -1.5.

38.1.2.4 Transient Growth

When every mode decays, the state goes to zero, but its length $\|\mathbf{x}(t)\|$ need not decrease all the way. A triangular matrix (called `B` in the code below) shows it:

$$\dot{\mathbf{x}} = \begin{pmatrix} -1 & 10 \\ 0 & -2 \end{pmatrix}\mathbf{x}, \qquad \mathbf{x}_0 = (0, 1),$$

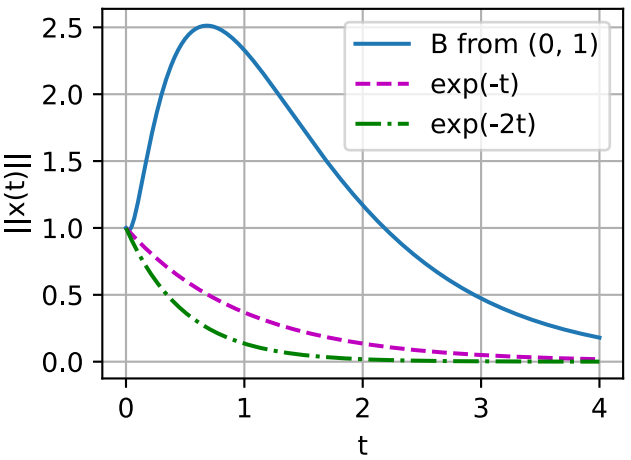
with eigenvalues -1 and -2, both negative. The second row involves only x_2, so $x_2(t) = e^{-2t}$. The first row is then $\dot{x}_1 = -x_1 + 10e^{-2t}$, a decay equation with an input, and Equation 38.6 gives

$$x_1(t) = 10\int_0^t e^{-(t-s)}e^{-2s}\,ds = 10\left(e^{-t} - e^{-2t}\right).$$

The first coordinate starts at 0, rises to $10(\tfrac{1}{2} - \tfrac{1}{4}) = 2.5$ at $t = \log 2 \approx 0.69$, and only then decays. The reason is the one of Section 33.4.2.3, which treats the discrete-time case in full: the eigenvectors $\mathbf{v}_1 = (1, 0)$ and $\mathbf{v}_2 = (0.995, -0.0995)$ span lines only 5.7° apart, so the short vector $\mathbf{x}_0 = 10.0\,\mathbf{v}_1 - 10.05\,\mathbf{v}_2$ is the difference of two long ones. They decay at different rates, and the cancellation stops before both have died away. The cell evaluates the closed form and plots the length of the state against the two modes.

```
B = np.array([[-1.0, 10.0], [0.0, -2.0]])     # both eigenvalues negative
ts = np.linspace(0, 4, 401)
x1 = 10 * (np.exp(-ts) - np.exp(-2 * ts))      # closed form from x0 = (0, 1)
x2 = np.exp(-2 * ts)
lam_B, V_B = np.linalg.eig(B)
x_eig = np.array([(V_B @ np.diag(np.exp(lam_B * s)) @ np.linalg.inv(V_B))[:, 1]
                  for s in ts])                 # e^{Bt} x0 = second column
norm = np.hypot(x1, x2)
print('eigenvalues of B:', lam_B)
print('|coefficients of x0|:', np.abs(np.linalg.solve(V_B, [0.0, 1.0])).round(2))
print(f'|closed form - eigen| = {np.abs(x_eig - np.c_[x1, x2]).max():.1e}')
print(f'peak {norm.max():.3f} at t = {ts[norm.argmax()]:.2f}')
d2l.plot(ts, [norm, np.exp(-ts), np.exp(-2 * ts)], 't', '||x(t)||',
         legend=['B from (0, 1)', 'exp(-t)', 'exp(-2t)'])
```

```
eigenvalues of B: [-1. -2.]
|coefficients of x0|: [10.   10.05]
|closed form - eigen| = 1.6e-15
peak 2.513 at t = 0.68
```



The length climbs from 1 to 2.51 at $t \approx 0.68$ and afterwards decays like $10e^{-t}$, at the rate of the slower mode. Eigenvalues give the asymptotic rate of decay, not the size of the transient. This matters when reading a training curve: a momentum run that converges can grow for its first few steps (7).

38.1.2.5 Linearization at Fixed Points

Nonlinear fields have no closed-form solution, but near a point where the field vanishes they behave like a linear one. A **fixed point** of an autonomous field is a state $\mathbf{x}^\star$ with $\mathbf{f}(\mathbf{x}^\star) = \mathbf{0}$: a particle placed there never moves. For a continuously differentiable $\mathbf{f}$ and a small displacement $\delta(t) = \mathbf{x}(t) - \mathbf{x}^\star$, Taylor's theorem (Section 34.2) gives

$$\dot{\delta} = \mathbf{f}(\mathbf{x}^\star + \delta) = \mathbf{J}\,\delta + o(\|\delta\|), \qquad \mathbf{J} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}^\star).$$

Near the fixed point the field is the linear field of its Jacobian $\mathbf{J}$. A fixed point is **hyperbolic** if no eigenvalue of $\mathbf{J}$ has zero real part. For a hyperbolic fixed point, the Hartman–Grobman theorem (Hartman 1960; Grobman 1959) says that near $\mathbf{x}^\star$ the trajectories of the nonlinear field are those of $\dot{\delta} = \mathbf{J}\delta$, up to a continuous change of coordinates. The eigenvalues of the Jacobian then decide local stability by the classification above.

The damped pendulum is the one nonlinear field we work out. Its angle ϕ from the bottom obeys $\ddot{\phi} = -\sin\phi - c\dot{\phi}$, with damping coefficient $c \geq 0$. With the angular velocity $\omega = \dot{\phi}$ it becomes the first-order system

$$\dot{\phi} = \omega, \qquad \dot{\omega} = -\sin\phi - c\,\omega, \qquad \mathbf{J}(\phi, \omega) = \begin{pmatrix} 0 & 1 \\ -\cos\phi & -c \end{pmatrix}.$$

The fixed points have $\omega = 0$ and $\sin\phi = 0$: the pendulum hanging at $\phi = 0$ and balanced upright at $\phi = \pi$.

- At the bottom, $\mathbf{J}$ has characteristic polynomial $\lambda^2 + c\lambda + 1$, with roots $\lambda = -c/2 \pm \mathrm{i}\sqrt{1 - c^2/4}$ for $c < 2$. At $c = 0.5$ they are $-0.25 \pm 0.97\,\mathrm{i}$: a stable spiral, the ring-down of a released pendulum, which swings with period about $2\pi/0.97$ while its amplitude decays like $e^{-t/4}$.
- At the top, the polynomial is $\lambda^2 + c\lambda - 1$, whose roots $(-c \pm \sqrt{c^2 + 4})/2$ have product -1, so one is positive and one negative for every c. At $c = 0.5$ they are 0.78 and -1.28: a saddle. Only the starts on one curve approach the top. They are the pendulum swung up with just the right speed, which comes to rest balanced like a pencil on its tip.

The same computation, a Jacobian and its eigenvalues, classifies the fixed points of a trained field, with one exception. When an eigenvalue lies on the imaginary axis, the linearization decides nothing. The fields $\dot{x} = -x^3$ and $\dot{x} = x^3$ both have Jacobian 0 at the origin. The first is stable, with $x(t) = x_0/\sqrt{1 + 2x_0^2 t} \to 0$; the second escapes to infinity at $t = 1/(2x_0^2)$.

38.1.3 Numerical Solvers

Few differential equations have a closed-form solution, so in practice we compute: follow the field for a short time h, read it again, and repeat. How far does such a step-by-step solution drift from the true one, how can it drift less for the same work, and when do the steps become unstable although the equation is not? The answers are an error law $O(h^p)$ and a region of stable steps, both read from one polynomial. At the end, gradient descent turns out to be the simplest of these solvers.

38.1.3.1 Forward Euler

Over one short step, the integral form Equation 38.2 can be approximated by holding the velocity at its value at the start of the step. Let us first see what this does on an equation whose answer we know.

Take $\dot{x} = x$ from $x_0 = 1$ to $T = 1$, where the answer is $e = 2.718$. Divide the interval into n steps of length $h = 1/n$, and in each step move with the velocity at its start: $x \leftarrow x + hx = (1 + 1/n)x$. After n steps, $x_n = (1 + 1/n)^n$. This is a dollar at 100% annual interest compounded n times a year:

n	1	2	4	8	16	32	365
$(1 + 1/n)^n$	2.000	2.250	2.441	2.566	2.638	2.677	2.7146
error	0.718	0.468	0.277	0.153	0.080	0.041	0.0037

The ratios of consecutive errors are $1.53, 1.69, 1.82, 1.90, 1.95$: doubling the number of steps comes closer and closer to halving the error. For large n the error is about $e/(2n) = 1.36\,h$, because $(1 + 1/n)^n = \exp(n \log(1 + 1/n)) = \exp(1 - \frac{1}{2n} + \cdots) \approx e\,(1 - \frac{1}{2n})$. Error proportional to h is what we will call order one.

For a general field, with step size h and grid $t_n = nh$, **forward Euler** is

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n, t_n). \tag{38.8}$$

The method follows the arrow at the current point for time h, then reads the field again. Equivalently, it is the Taylor expansion of the true solution cut after the linear term, and the error of one step is the first term dropped. If $\|\ddot{\mathbf{x}}(t)\| \le M$ along the solution, then

$$\mathbf{x}(t_{n+1}) = \mathbf{x}(t_n) + h\,\mathbf{f}(\mathbf{x}(t_n), t_n) + \mathbf{r}_n, \qquad \|\mathbf{r}_n\| \le \tfrac{Mh^2}{2}.$$

A per-step error of order h^2 does not give a final error of order h^2: reaching a fixed horizon T takes T/h steps, the per-step errors accumulate, and each step may amplify the error already made. One power of h is lost, and the proof is short.

Proposition (Euler converges at order 1). *Let* $\mathbf{f}$ *be* L*-Lipschitz in* $\mathbf{x}$*, and let the solution of Equation 38.1 on* $[0, T]$ *satisfy* $\|\ddot{\mathbf{x}}(t)\| \le M$*. Then the Euler iterates Equation 38.8 obey*

$$\max_{n \le T/h} \|\mathbf{x}_n - \mathbf{x}(t_n)\| \;\le\; \frac{Mh}{2L}\,(e^{LT} - 1) \;=\; O(h).$$

Proof. Let $\varepsilon_n = \|\mathbf{x}_n - \mathbf{x}(t_n)\|$, and subtract the Taylor identity above from the update Equation 38.8:

$$\varepsilon_{n+1} \le \underbrace{\|\mathbf{x}_n - \mathbf{x}(t_n)\| + h\,\|\mathbf{f}(\mathbf{x}_n, t_n) - \mathbf{f}(\mathbf{x}(t_n), t_n)\|}_{\le (1+hL)\,\varepsilon_n \text{ by Lipschitz}} + \underbrace{\|\mathbf{r}_n\|}_{\le Mh^2/2}.$$

Each step multiplies the existing error by at most $1 + hL$ and adds at most $Mh^2/2$. Unrolling the recursion from $\varepsilon_0 = 0$ sums a geometric series:

$$\varepsilon_n \le \frac{Mh^2}{2}\sum_{j=0}^{n-1}(1+hL)^j = \frac{Mh^2}{2}\cdot\frac{(1+hL)^n-1}{hL} \le \frac{Mh}{2L}\left(e^{LT}-1\right),$$

using $(1+hL)^n \le e^{nhL} \le e^{LT}$. ■

For $\dot{x} = x$ on $[0,1]$ we have $M = \max|\ddot{x}| = e$ and $L = 1$, so the bound is $\frac{e}{2}(e-1)h = 2.3\,h$, against the true error $1.36\,h$: the right order, with a pessimistic constant. The structure of the bound recurs for every solver. A local error of order h^{p+1} per step becomes a global error of order h^p after T/h steps, inflated by the factor e^{LT} that measures how strongly the dynamics amplify old mistakes. A method with global error of order h^p has **order** p; Euler has order 1.

38.1.3.2 Runge–Kutta Methods

Euler commits to the slope at the start of the step while the trajectory curves away from it. Runge–Kutta methods measure the slope at points inside the step and combine the measurements. Figure 38.4 shows one step of size $h = 1$ on $\dot{x} = x$ from $x = 1$.

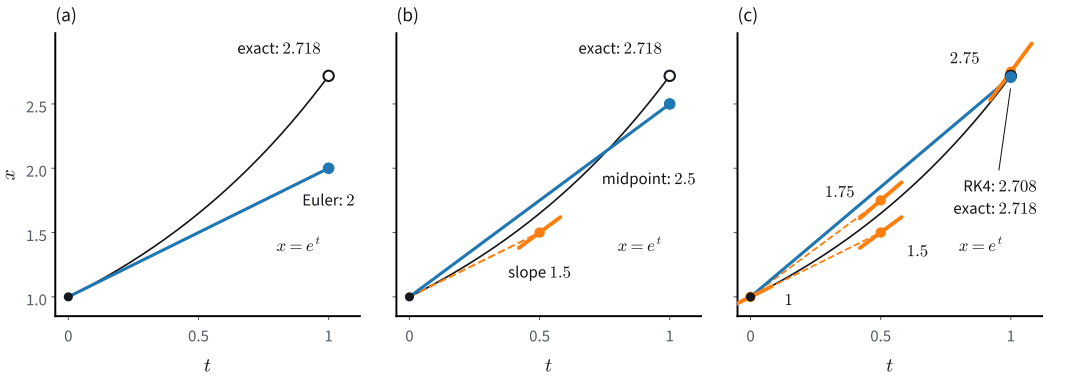


Figure 38.4: One step of size $h = 1$ on $\dot{x} = x$ from $x = 1$. (a) Euler follows the starting slope and lands at 2. (b) The midpoint method takes a half step only to measure the slope there, 1.5, and lands at 2.5. (c) RK4 measures four slopes, 1, 1.5, 1.75 and 2.75, averages them with weights $\frac{1}{6}(1,2,2,1)$ and lands at 2.708; the exact value is $e = 2.718$.

The simplest improvement, the **midpoint method**, takes half an Euler step only to measure the slope there, and then takes the real step with that slope: $\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}\big(\mathbf{x}_n + \frac{h}{2}\mathbf{f}(\mathbf{x}_n)\big)$ (autonomous notation). One Taylor line in one dimension shows why it helps:

$$x + hf\big(x + \tfrac{h}{2}f(x)\big) = x + hf + \tfrac{h^2}{2}f'f + O(h^3).$$

The true expansion is $x(t+h) = x + h\dot{x} + \frac{h^2}{2}\ddot{x} + O(h^3)$ with $\ddot{x} = \frac{d}{dt}f(x(t)) = f'f$, so the h^2 term is now right instead of missing. The local error is $O(h^3)$ and the global order 2, for two field evaluations per step. **Heun's method** takes a full trial Euler step and averages the slopes at its two ends, $\mathbf{x}_{n+1} = \mathbf{x}_n + \frac{h}{2}\left[\mathbf{f}(\mathbf{x}_n) + \mathbf{f}(\mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n))\right]$, with the same order and the same two evaluations. Many diffusion samplers use this two-stage method (Section 17.4).

Four slope measurements give the standard general-purpose method of scientific computing, **classical Runge–Kutta (RK4)**:

$$\begin{aligned}
\mathbf{k}_1 &= \mathbf{f}(\mathbf{x}_n,\, t_n),\\
\mathbf{k}_2 &= \mathbf{f}(\mathbf{x}_n + \tfrac{h}{2}\mathbf{k}_1,\, t_n + \tfrac{h}{2}),\\
\mathbf{k}_3 &= \mathbf{f}(\mathbf{x}_n + \tfrac{h}{2}\mathbf{k}_2,\, t_n + \tfrac{h}{2}),\\
\mathbf{k}_4 &= \mathbf{f}(\mathbf{x}_n + h\,\mathbf{k}_3,\, t_n + h),\\
\mathbf{x}_{n+1} &= \mathbf{x}_n + \tfrac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right).
\end{aligned} \tag{38.9}$$

The update averages four slope measurements, weighting the two at the midpoint twice. These weights are Simpson's rule, $\int_a^b g \approx \frac{b-a}{6}(g(a) + 4g(\frac{a+b}{2}) + g(b))$, which integrates cubics exactly, with the midpoint value split between two measurements.

Why order 4? On a linear field the answer takes one computation. Each method applied to the scalar test equation $\dot{y} = \lambda y$ multiplies y by a fixed function of $z = h\lambda$ per step, and we can compare that function with the exact factor e^z.

Proposition (stability functions). *On $\dot{y} = \lambda y$, with λ real or complex and $z = h\lambda$, one step of each method multiplies y by its* **stability function** *$R(z)$:*

$$R_{\text{Euler}}(z) = 1 + z, \qquad R_{\text{mid}}(z) = R_{\text{Heun}}(z) = 1 + z + \frac{z^2}{2}, \qquad R_{\text{RK4}}(z) = 1 + z + \frac{z^2}{2} + \frac{z^3}{6} + \frac{z^4}{24}. \tag{38.10}$$

Proof. For RK4 the four measurements are $\mathbf{k}_1 = \lambda y$, $\mathbf{k}_2 = \lambda y(1 + \frac{z}{2})$, $\mathbf{k}_3 = \lambda y(1 + \frac{z}{2} + \frac{z^2}{4})$ and $\mathbf{k}_4 = \lambda y(1 + z + \frac{z^2}{2} + \frac{z^3}{4})$, each obtained by substituting the previous one. The update is $y + \frac{z}{6}\left(6 + 3z + z^2 + \frac{z^3}{4}\right) y$, which is the stated polynomial times y. For Euler, $y + zy$. For the midpoint method, $y + h\lambda(y + \frac{z}{2}y)$, and for Heun, $y + \frac{z}{2}(y + (1 + z)y)$; both are $(1 + z + \frac{z^2}{2})y$. ■

Each R is the Taylor polynomial of the exact factor e^z, cut at the method's order. For RK4, $e^z - R_{\text{RK4}}(z) = z^5/120 + O(z^6)$, so on this equation one step errs by $O(h^5)$ and T/h steps by $O(h^4)$. For nonlinear fields that are four times continuously differentiable and Lipschitz in $\mathbf{x}$, a longer Taylor match gives the same orders: midpoint and Heun have order 2, RK4 order 4 (Hairer et al. 1993); we do not prove this. The stability function returns when we ask which steps are stable.

Practical solvers also choose the step size as they go. An *embedded pair* such as Dormand–Prince computes solutions of orders 4 and 5 from the same evaluations and uses their difference as an estimate of the local error (Dormand and Prince 1980); 4 builds the simplest such controller.

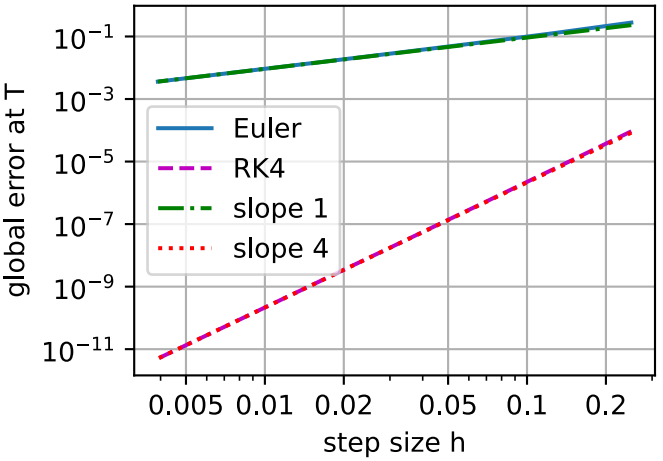
On log–log axes, error $\approx Ch^p$ is a line of slope p, so the measured slopes should be 1 and 4. We test this on the spiral $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ of Figure 38.1, whose eigenvalues are $-0.5 \pm \mathrm{i}$ and whose exact endpoint $e^{\mathbf{A}T}\mathbf{x}_0$ at $T = 2$ the matrix-exponential cell computed, halving the step six times.

```
def euler(f, x0, T, n):
    """Forward Euler: n steps of size h = T/n."""
    x, h = np.asarray(x0, dtype=float), T / n
    for k in range(n):
        x = x + h * f(x)
    return x

def rk4(f, x0, T, n):
    """Classical Runge-Kutta: four slope measurements per step."""
    x, h = np.asarray(x0, dtype=float), T / n
    for k in range(n):
        k1 = f(x)
        k2 = f(x + 0.5 * h * k1)
        k3 = f(x + 0.5 * h * k2)
        k4 = f(x + h * k3)
        x = x + (h / 6) * (k1 + 2 * k2 + 2 * k3 + k4)
    return x

f = lambda x: A @ x                          # the spiral field from above
T = 2.0
x_true = expAt_eig @ x0                      # exact endpoint via e^{At}
ns = 2 ** np.arange(3, 10)                   # 8, 16, ..., 512 steps
hs = T / ns
err_eu = np.array([np.linalg.norm(euler(f, x0, T, n) - x_true) for n in ns])
err_rk = np.array([np.linalg.norm(rk4(f, x0, T, n) - x_true) for n in ns])
print('measured slope, Euler:', f'{np.polyfit(np.log(hs), np.log(err_eu), 1)[0]:.3f}')
print('measured slope, RK4  :', f'{np.polyfit(np.log(hs), np.log(err_rk), 1)[0]:.3f}')
print(f'error at 512 steps: Euler {err_eu[-1]:.1e}, RK4 {err_rk[-1]:.1e}')
print('RK4 with 128 steps (the same 512 evaluations):',
      f'{np.linalg.norm(rk4(f, x0, T, 128) - x_true):.1e}')
d2l.plot(hs, [err_eu, err_rk, err_eu[-1] * (hs / hs[-1]),
              err_rk[-1] * (hs / hs[-1]) ** 4],
         'step size h', 'global error at T', xscale='log', yscale='log',
         legend=['Euler', 'RK4', 'slope 1', 'slope 4'])
```

```
measured slope, Euler: 1.040
measured slope, RK4  : 4.022
error at 512 steps: Euler 3.6e-03, RK4 5.0e-12
RK4 with 128 steps (the same 512 evaluations): 1.3e-09
```



The fitted slopes are 1.04 and 4.02. What the slopes do not show is the cost. A method of order p reaches error ε with a number of steps proportional to $\varepsilon^{-1/p}$, so a high order pays even at four evaluations per step: at 512 steps Euler is off by 3.6×10^{-3}, and with the same 512 field evaluations, that is 128 steps, RK4 reaches 1.3×10^{-9}, six orders of magnitude below.

38.1.3.3 Stiffness and Implicit Methods

Order measures how accurate a step is when it is small. Stability asks a different question: whether the errors of many steps die out or compound. For explicit methods such as Euler and RK4, stability caps the step size, however little accuracy we need.

The smallest system that shows the problem has two decoupled modes, $\dot{\mathbf{x}} = \operatorname{diag}(-50, -1)\,\mathbf{x}$ from $(1, 1)$, followed to $T = 1$. The fast mode has decayed to $e^{-50} \approx 10^{-22}$; the slow one is at $e^{-1} = 0.368$. After $t \approx 0.1$ only the slow mode is left, and steps of $h = 0.1$ would follow it well. Yet the fast coordinate is the opening's equation $\dot{x} = -50x$, and Euler multiplies it by $1 - 50h$ at every step for as long as we integrate. A system is **stiff** when some Jacobian eigenvalues have large negative real parts compared with the rate at which the solution of interest varies: a mode that is long gone still sets the step.

To see which steps are allowed, and what a remedy looks like, we use the test equation $\dot{y} = \lambda y$ with complex λ, whose solution decays if and only if $\operatorname{Re}\lambda < 0$.

Proposition (Euler and backward Euler on the test equation). *Let $z = h\lambda$. Forward Euler gives $y_n = (1 + z)^n y_0$, which decays if and only if $|1 + z| < 1$; for real $\lambda < 0$ this is*

$$h < \frac{2}{|\lambda|}.$$

Backward Euler, $\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_{n+1}, t_{n+1})$, gives $y_n = (1 - z)^{-n} y_0$, which decays for every $h > 0$ whenever $\operatorname{Re}\lambda < 0$.

Proof. Forward Euler: $y_{n+1} = y_n + h\lambda y_n = (1+z)y_n$; iterate. For real $z < 0$ the condition $|1+z| < 1$ reads $-2 < z < 0$. Backward Euler: $y_{n+1} = y_n + h\lambda y_{n+1}$, so $y_{n+1} = y_n/(1-z)$, and $|1 - z| \geq \operatorname{Re}(1 - z) = 1 - \operatorname{Re} z > 1$ when $\operatorname{Re} z < 0$. ■

Backward Euler evaluates the field at the end of the step, which is not yet known, so each step is an equation to solve. That is the meaning of *implicit*. The price is a linear solve per step when the field is linear and a few Newton iterations otherwise (10), and it is worth paying when the stability limit, not accuracy, would set the step of an explicit method.

A plot of the z plane shows all methods at once. The **stability region** of a method is the set $\{z : |R(z)| < 1\}$ where its update decays; for backward Euler $R(z) = 1/(1 - z)$. A method is **A-stable** if its region contains the whole open left half-plane $\mathrm{Re}\, z < 0$: it makes every decaying mode decay, whatever the step (Dahlquist 1963). Backward Euler is A-stable, since its region is everything outside the closed unit disc around $+1$. No explicit Runge–Kutta method is A-stable, because its R is a polynomial (2).

Figure 38.5 marks the examples. The opening's step $h = 0.05$ on $\dot{x} = -50x$ is the point $z = -2.5$, outside Euler's disc, and $h = 0.01$ is $z = -0.5$, inside. The spiral, with $\lambda = -0.5 \pm \mathrm{i}$, gives $|1 + z|^2 = (1 - h/2)^2 + h^2$, which is below 1 if and only if $h < 0.8$; at $h = 0.8$ its two points $-0.4 \pm 0.8\,\mathrm{i}$ lie on the circle. The exact step e^z of Section 38.1.2.3 has modulus below one on the whole left half-plane, which is the behaviour backward Euler imitates and forward Euler does not.

We now run both methods on the two-mode system at five step sizes, backward Euler with one linear solve per step. The reader can predict the first row: at $h = 0.1$ forward Euler multiplies the fast coordinate by $1 - 5 = -4$ ten times.

```
def backward_euler_linear(A, x0, T, n):
    """Backward Euler on dx/dt = A x: solve (I - h A) x_new = x at each step."""
    x, h = np.asarray(x0, dtype=float), T / n
    for _ in range(n):
        x = np.linalg.solve(np.eye(len(x)) - h * A, x)
    return x

A_stiff = np.diag([-50.0, -1.0])              # one fast mode, one slow mode
x0_stiff = np.array([1.0, 1.0])
T = 1.0
print(f'exact at T = 1: fast {np.exp(-50):.1e}, slow {np.exp(-1):.4f}')
print(f'{"h":>5} {"fwd fast":>10} {"fwd slow":>9} {"bwd fast":>10} {"bwd slow":>9}')
for n in [10, 20, 25, 50, 100]:
    fwd = euler(lambda x: A_stiff @ x, x0_stiff, T, n)
    bwd = backward_euler_linear(A_stiff, x0_stiff, T, n)
    print(f'{T / n:5.2f} {fwd[0]:10.1e} {fwd[1]:9.4f} {bwd[0]:10.1e} {bwd[1]:9.4f}')
```

```
exact at T = 1: fast 1.9e-22, slow 0.3679
    h   fwd fast  fwd slow   bwd fast  bwd slow
 0.10    1.0e+06    0.3487    1.7e-08    0.3855
 0.05    3.3e+03    0.3585    1.3e-11    0.3769
 0.04   -1.0e+00    0.3604    1.2e-12    0.3751
 0.02    0.0e+00    0.3642    8.9e-16    0.3715
 0.01    7.9e-31    0.3660    2.5e-18    0.3697
```

At $h = 0.1$ both methods have the slow mode within about 5% of the truth (0.349 and 0.386 against 0.368), accurate enough for many purposes. Forward Euler nonetheless returns $4^{10} \approx 10^6$ for the fast component, whose true value is 10^{-22}. At $h = 0.04$ the factor is -1, and the fast component flips sign at every step and never shrinks. Forward Euler needs $h < 0.04$, and at $h = 0.02$ its slow mode is five times more accurate than the 5% we were

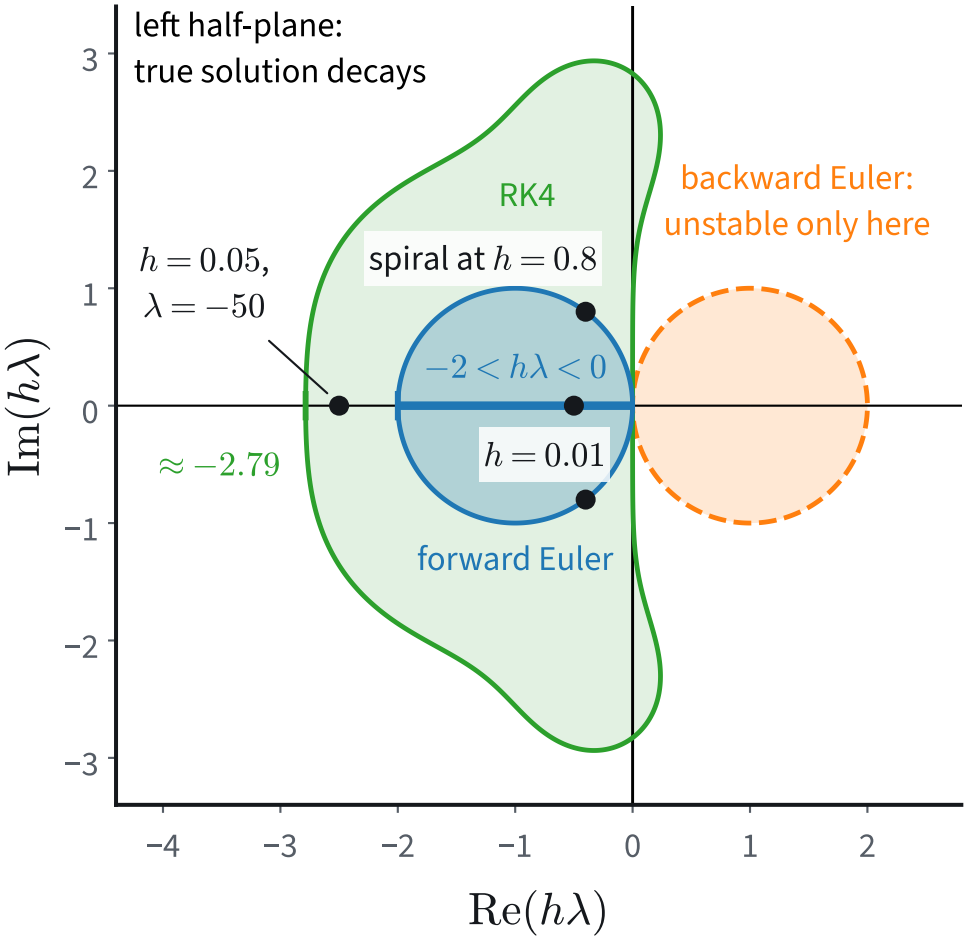


Figure 38.5: Stability regions in the complex plane of $z = h\lambda$ for the test equation $\dot{y} = \lambda y$; a decaying mode sits in the left half-plane. Forward Euler decays only inside the disc $|1 + z| < 1$, whose real slice is $-2 < h\lambda < 0$; RK4 enlarges it to a lobed region reaching ≈ -2.79 on the real axis; backward Euler is stable everywhere except the disc around $+1$, in particular on the entire left half-plane, which is A-stability. The marked points are the opening's $h = 0.05$ step, its $h = 0.01$ step, and the spiral's largest stable Euler step.

content with.

Backward Euler is stable at every step size. Its slow-mode error is the $O(h)$ error of a first-order method, so stability does not bring accuracy. In practice, explicit adaptive solvers are the default, and implicit methods are used when the field is stiff, which a learned reverse-time field can be.

38.1.3.4 Gradient Descent as a Solver

The opening treated one coordinate. With several, the same factor appears once per eigenvalue of the Hessian. Infinitesimal descent on a loss $\mathcal{L}$ is the **gradient flow**

$$\dot{\mathbf{x}} = -\nabla \mathcal{L}(\mathbf{x}),$$

a field whose fixed points are the critical points of $\mathcal{L}$ and along whose trajectories the loss only falls: $\frac{d}{dt}\mathcal{L}(\mathbf{x}(t)) = \nabla\mathcal{L}^\top \dot{\mathbf{x}} = -\|\nabla\mathcal{L}\|^2 \le 0$. Gradient descent with learning rate η, $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\,\nabla\mathcal{L}(\mathbf{x}_k)$, is forward Euler Equation 38.8 on this flow with $h = \eta$.

Section 35.1 derives the learning-rate ceiling $\eta < 2/\lambda_{\max}(\mathbf{H})$ on a quadratic with Hessian $\mathbf{H}$ and studies the speed below it. We do not repeat that analysis. What the solver view adds is where the ceiling comes from. Near a minimum the flow linearizes, eigenvector by eigenvector, onto the test equation:

$$\dot{\mathbf{x}} = -\nabla\mathcal{L}(\mathbf{x}) \xrightarrow{\text{linearize}} \dot{\delta} = -\mathbf{H}\delta \xrightarrow{\text{Euler, } h=\eta} \delta_{k+1} = (\mathbf{I} - \eta\mathbf{H})\,\delta_k \xrightarrow{\text{stability}} \eta < \frac{2}{\lambda_{\max}(\mathbf{H})}.$$

The learning-rate ceiling is thus the boundary of Euler's stability disc, reached by the stiffest mode at $z = -\eta\lambda_{\max} = -2$. The cell reruns the sweep of Section 35.1 on $\mathbf{H} = \mathrm{diag}(10, 1)$ from $(1, 1)$ for 100 steps, and each row can be read in Figure 38.5.

```
H = np.diag([10.0, 1.0])                      # lambda_max = 10, lambda_min = 1
print('GD on this quadratic is stable iff eta < 2/lambda_max =', 2 / H[0, 0])
print(f'{"eta":>6} {"|1-eta*lam_max|":>16} {"|1-eta*lam_min|":>16} {"||x_100||":>10}')
for eta in [0.05, 0.15, 0.19, 0.201, 0.21, 0.25]:
    x = np.array([1.0, 1.0])
    for _ in range(100):
        x = x - eta * (H @ x)                 # gradient descent = forward Euler
    print(f'{eta:6.3f} {abs(1 - eta * H[0, 0]):16.2f} '
          f'{abs(1 - eta * H[1, 1]):16.3f} {np.linalg.norm(x):10.2e}')
```

```
GD on this quadratic is stable iff eta < 2/lambda_max = 0.2
   eta  |1-eta*lam_max|  |1-eta*lam_min|  ||x_100||
 0.050            0.50           0.950   5.92e-03
 0.150            0.50           0.850   8.75e-08
 0.190            0.90           0.810   2.66e-05
 0.201            1.01           0.799   2.70e+00
 0.210            1.10           0.790   1.38e+04
 0.250            1.50           0.750   4.07e+17
```

This is the table of the stiff system with η in place of h. At $\eta = 0.201$ the steep mode sits at $z = -2.01$, just outside the disc: its factor is -1.01, and after 100 steps the iterate has grown past its start while the flat coordinate is about 10^{-10}. Stability fixes η through $\lambda_{\max}$, and $\lambda_{\min}$ then sets the speed, so the condition number of the Hessian is the stiffness of the gradient flow. A diverging learning rate is a solver instability.

Two more readings are left to the exercises. The $O(\eta^2)$ error of each explicit step has a meaning too: to that order gradient descent follows the flow of a modified loss that penalizes large gradients (5). Polyak's heavy-ball momentum discretizes the second-order equation $m\ddot{\mathbf{x}} + c\dot{\mathbf{x}} = -\nabla\mathcal{L}(\mathbf{x})$ of a ball with mass and friction (Polyak 1964) (3).

Every field in this section was given to us. The next section, Section 38.2, defines a model whose layers are Euler steps of a field that is *learned*, and asks how such a model is trained and what it does to a probability density.

38.1.4 Summary

Ordinary differential equations. An ordinary differential equation prescribes a velocity at every point. When the field is globally Lipschitz, exactly one trajectory passes through each point, and the flow map is a bijection. Linear systems are solved by $e^{\mathbf{A}t} = \mathbf{V}e^{\Lambda t}\mathbf{V}^{-1}$, so the real parts of the eigenvalues decide decay and the imaginary parts rotation, and with an input held constant over a step the exact solution is a linear recurrence that is stable at every step size. Nearly parallel eigenvectors allow transient growth, and at a hyperbolic fixed point the Jacobian classifies a nonlinear field as its eigenvalues would a linear one.

Numerical solvers. A numerical solver turns a local error $O(h^{p+1})$ into a global error $O(h^p)$, with $p = 1$ for Euler and $p = 4$ for RK4. Forward Euler is stable only for $|1 + h\lambda| < 1$ and backward Euler on the whole left half-plane, and a system is stiff when a decayed fast mode still sets the step. Gradient descent is forward Euler on $\dot{\mathbf{x}} = -\nabla\mathcal{L}$, so $\eta < 2/\lambda_{\max}(\mathbf{H})$ is a solver stability bound and ill-conditioning is stiffness.

38.1.5 Exercises

1. **Separation of trajectories.** Consider a field of the form $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2$, the Neural ODE field of the next section.

   1. Show that $\mathbf{f}$ is Lipschitz with constant $L = \|\mathbf{W}_2\|\,\|\mathbf{W}_1\|$ in the operator norm. Which property of tanh does the proof use? Does the bound change when tanh is replaced by ReLU?

   2. Let $\mathbf{x}(t)$ and $\mathbf{y}(t)$ be two trajectories of an autonomous field with Lipschitz constant L. Differentiate $\|\mathbf{x}(t) - \mathbf{y}(t)\|^2$ and show that

   $$e^{-Lt}\|\mathbf{x}(0) - \mathbf{y}(0)\| \le \|\mathbf{x}(t) - \mathbf{y}(t)\| \le e^{Lt}\|\mathbf{x}(0) - \mathbf{y}(0)\|.$$

   What does the lower bound imply for two trajectories that start at distinct points? What do the two bounds imply for the Lipschitz constants of Φ_T and of its inverse?

2. **Rotating modes.** The stiffness proposition treats a single decaying mode. A complex eigenvalue λ enters through $z = h\lambda$, the coordinate of Figure 38.5.

1. For the spiral field of Figure 38.1, with eigenvalues $-0.5 \pm \mathrm{i}$, find the largest step size for which forward Euler decays.
2. For the rotational field $\dot{\mathbf{x}} = \left(\begin{smallmatrix} 0 & -b \\ b & 0 \end{smallmatrix}\right)\mathbf{x}$, show that forward Euler multiplies $\|\mathbf{x}\|$ by $\sqrt{1+h^2b^2}$ per step and that backward Euler divides it by the same factor. By which factor does each method change $\|\mathbf{x}\|$ over one revolution, that is $2\pi/(hb)$ steps, when $hb = 0.1$?
3. The stability function of RK4 is the last formula in Equation 38.10, and its square modulus on the imaginary axis decides rotation. Show that $|R(\mathrm{i}y)|^2 = 1 - y^6/72 + y^8/576$ for real y, and find the range of hb for which RK4 does not amplify the rotation.
4. An explicit Runge–Kutta method with s stages computes $\mathbf{k}_i = \mathbf{f}(\mathbf{x}_n + h\sum_{j<i} a_{ij}\mathbf{k}_j)$ for $i = 1, \dots, s$ and $\mathbf{x}_{n+1} = \mathbf{x}_n + h\sum_i b_i\mathbf{k}_i$, as RK4 does in Equation 38.9. Show that its stability function is a polynomial in z of degree at most s. Why does this rule out A-stability for explicit methods?

3. **Heavy ball as a discretized ODE.** Heavy-ball momentum $\mathbf{x}_{k+1} = \mathbf{x}_k - \eta\nabla\mathcal{L}(\mathbf{x}_k) + \beta(\mathbf{x}_k - \mathbf{x}_{k-1})$ discretizes the equation $m\ddot{\mathbf{x}} + c\dot{\mathbf{x}} = -\nabla\mathcal{L}(\mathbf{x})$ of a ball with mass m and friction c.
   1. Replace $\ddot{\mathbf{x}}(t_k)$ by the central difference $(\mathbf{x}_{k+1} - 2\mathbf{x}_k + \mathbf{x}_{k-1})/h^2$ and $\dot{\mathbf{x}}(t_k)$ by the backward difference $(\mathbf{x}_k - \mathbf{x}_{k-1})/h$. Show that the result is the heavy-ball update, and express η and β through m, c, and h.
   2. For $\mathcal{L}(x) = \frac{\lambda}{2}x^2$ the ODE is a damped oscillator, critically damped at $c = 2\sqrt{m\lambda}$. Express this condition through η, β, and λ. The heavy-ball recurrence is critically damped when its characteristic polynomial $z^2 - (1 + \beta - \eta\lambda)z + \beta$ has a double root. Find the corresponding β and compare the two conditions for small $\eta\lambda$.
   3. The heavy-ball recurrence on this quadratic converges if and only if $0 < \eta\lambda < 2(1+\beta)$ for $0 \le \beta < 1$ (exercise 9a of Section 35.1 asks for the proof). Translate the condition into one on h, m, c, and λ. The ODE decays for every $\lambda > 0$, but its discretization does not. What is the largest stable step h as $c \to 0$, in terms of the angular frequency $\sqrt{\lambda/m}$ of the oscillator? Which eigenvalue of the Hessian of an ill-conditioned quadratic sets the step limit?

4. code **Adaptive step size.** ⊠ The embedded pairs of Section 38.1.3 estimate the local error from two methods of different order. The simplest pair uses the stages $\mathbf{k}_1 = \mathbf{f}(\mathbf{x})$ and $\mathbf{k}_2 = \mathbf{f}(\mathbf{x} + h\mathbf{k}_1)$ of Heun's method: the Euler value $\mathbf{x} + h\mathbf{k}_1$ and the Heun value $\mathbf{x} + \frac{h}{2}(\mathbf{k}_1 + \mathbf{k}_2)$ differ by $\Delta = \frac{h}{2}\|\mathbf{k}_2 - \mathbf{k}_1\|$. A controller accepts the step, keeping the Heun value, when $\Delta \le \texttt{tol}$, and in either case continues with the step size

$$h \leftarrow h\,\min\Big(2,\ \max\big(0.2,\ 0.9\sqrt{\texttt{tol}/\Delta}\,\big)\Big).$$

   1. Which order of the local error does the exponent $1/2$ assume?
   2. Implement the controller, starting from $h = 0.1$ and shortening the last step so that it ends at T. Integrate the spiral field `f` from `x0` to $T = 2$ for `tol` $= 10^{-2}, 10^{-3}, \dots, 10^{-6}$. Count the field evaluations, including those of rejected steps, and compare the error with the exact endpoint `x_true`. How does the number of evaluations scale with `tol`?
   3. Integrate the field of `A_stiff` from $(1, 1)$ to $T = 10$ with `tol` $= 10^{-3}$ and plot the accepted step sizes against time. At which value do they settle after the fast mode has decayed? Explain this value with Heun's stability function $1 + z + z^2/2$ from

Equation 38.10.

5. code **Implicit gradient regularization.** ⊠⊠ Gradient descent is forward Euler on the gradient flow (Section 38.1.3.4), so each step leaves the flow of $\mathcal{L}$ by $O(\eta^2)$. Backward error analysis looks for a modified field $\tilde{\mathbf{f}} = \mathbf{f} + \eta \mathbf{f}_1$ whose exact time-η flow agrees with one Euler step up to $O(\eta^3)$. Barrett and Dherin (2021) find that for gradient descent this field is again a gradient field, of the modified loss

$$\tilde{\mathcal{L}}(\mathbf{x}) = \mathcal{L}(\mathbf{x}) + \frac{\eta}{4}\|\nabla \mathcal{L}(\mathbf{x})\|^2.$$

   1. Expand the exact flow of $\tilde{\mathbf{f}}$ over time η to second order, and show that $\mathbf{f}_1 = -\frac{1}{2}(\partial \mathbf{f}/\partial \mathbf{x})\,\mathbf{f}$. Derive $\tilde{\mathcal{L}}$ for $\mathbf{f} = -\nabla \mathcal{L}$. Which regions of parameter space does the added term penalize?
   2. For $\mathcal{L}(x) = \frac{\lambda}{2}x^2$ with $0 < \eta\lambda < 1$, show that the iterates of gradient descent lie exactly on the gradient flow of $\frac{\tilde{\lambda}}{2}x^2$ with $\tilde{\lambda} = -\log(1-\eta\lambda)/\eta$. Expand $\tilde{\lambda}$ in η and compare it with the curvature of $\tilde{\mathcal{L}}$.
   3. Gradient descent on this quadratic still converges for $1 < \eta\lambda < 2$, with iterates that alternate in sign. Trajectories of a scalar autonomous field with a Lipschitz constant cannot cross, so the time-η map of such a field preserves order. Conclude that no such field takes the values of the iterates at the times $k\eta$. For a quadratic with Hessian $\mathbf{H}$ in d dimensions, show that for every starting point with a component along the top eigenvector, the iterates lie on the flow of no gradient field $-\nabla(\frac{1}{2}\mathbf{x}^\top \tilde{\mathbf{H}}\mathbf{x})$ once $\eta \geq 1/\lambda_{\max}(\mathbf{H})$.
   4. For $\mathcal{L}(x) = x^4/4 + x^2/2$ and $x_0 = 1.5$, run gradient descent to $T = 1$ with $\eta = 0.1, 0.05, 0.025, 0.0125$. Compute the gradient flows of $\mathcal{L}$ and of $\tilde{\mathcal{L}}$ to $T = 1$ with `rk4` and many steps, the latter separately for each η. Fit the slopes of the two endpoint errors of gradient descent against η.

6. code **Symplectic Euler and splitting.** The harmonic oscillator $\dot{q} = p$, $\dot{p} = -q$ conserves $q^2 + p^2$. Forward Euler spirals out of its circular orbits and backward Euler spirals in. The *symplectic Euler* update advances q with the old p and then p with the new q: $q_{n+1} = q_n + hp_n$, $p_{n+1} = p_n - hq_{n+1}$.

   1. Write the update as a matrix $\mathbf{S}_h$ acting on (q, p) and show that $\det \mathbf{S}_h = 1$ for every h, so the update preserves area as the exact flow does, while forward Euler's matrix has determinant $1 + h^2$.
   2. Show that the eigenvalues of $\mathbf{S}_h$ are distinct and lie on the unit circle if and only if $0 < h < 2$. What happens at $h = 2$ and for $h > 2$?
   3. Show that $q^2 + p^2 + hqp$ is conserved exactly by the update. What does its level set through $(1, 0)$ look like, and how far from the origin can the computed orbit get at $h = 0.5$ and at $h = 1$? (Hint: the axes of the quadratic form have lengths in the ratio $\sqrt{(2+h)/(2-h)}$.)
   4. Integrate from $(1, 0)$ to $T = 200$ with forward Euler, backward Euler and symplectic Euler at $h = 0.1$ and $h = 1$, and report the largest and smallest $\|(q, p)\|$ along each run.
   5. The update is the exact flow of $\dot{q} = p$ with p frozen, for time h, followed by the exact flow of $\dot{p} = -q$ with q frozen. Write the two generators $\mathbf{P}$ and $\mathbf{Q}$, whose sum is the oscillator's matrix, compute $e^{h\mathbf{P}}$, $e^{h\mathbf{Q}}$ and $e^{h(\mathbf{P}+\mathbf{Q})}$, and show that $e^{h\mathbf{Q}}e^{h\mathbf{P}} = \mathbf{S}_h$ differs from $e^{h(\mathbf{P}+\mathbf{Q})}$ by terms of order h^2. Which property used for the spiral's

closed form fails here? The symmetric splitting $e^{h\mathbf{Q}/2}e^{h\mathbf{P}}e^{h\mathbf{Q}/2}$ is the *leapfrog* integrator of Hamiltonian Monte Carlo (Neal 2011); what is its order?

7. code **Transient growth and nonnormality.** The text shows one matrix with transient growth. This exercise finds how the growth depends on the angle between the eigenvectors, and meets it again in momentum.
   1. For $\mathbf{B}_\nu = \left(\begin{smallmatrix} -1 & \nu \\ 0 & -2 \end{smallmatrix}\right)$ and $\mathbf{x}_0 = (0, 1)$, show that $x_1(t) = \nu(e^{-t} - e^{-2t})$ and find the peak of x_1 and the time at which it occurs. Find numerically the smallest ν for which $\|\mathbf{x}(t)\|$ ever exceeds $\|\mathbf{x}_0\|$.
   2. Compute the angle between the two eigenvectors of $\mathbf{B}_\nu$ as a function of ν, express $\mathbf{x}_0$ in the eigenbasis, and explain the growth as the cancellation of two large components.
   3. Compute $\max_t \|e^{\mathbf{B}_\nu t}\|_2$ for $\nu = 1, 5, 10, 20$ numerically and compare it with the peak of the trajectory from $(0, 1)$. Why can the first never be smaller than the second?
   4. The heavy-ball recurrence of 3 on $\mathcal{L}(x) = \frac{\lambda}{2}x^2$ is the linear map $(x_{k+1}, x_k) = \mathbf{M}(x_k, x_{k-1})$ with $\mathbf{M} = \left(\begin{smallmatrix} 1+\beta-\eta\lambda & -\beta \\ 1 & 0 \end{smallmatrix}\right)$. Show that $\mathbf{M}$ is not normal, and compute its eigenvalues for $\beta = 0.9$, $\eta\lambda = 0.1$. Run it from $(x_0, x_{-1}) = (1, 0)$: for how many steps does $\|(x_{k+1}, x_k)\|$ grow before it decays, and to what peak? What does this say about reading the first iterations of a momentum run as divergence?

8. **Limits of linearization.**
   1. Classify the origin of $\dot{x} = -x^3$, $\dot{x} = x^3$ and $\dot{x} = x^2$ (stable, unstable, or stable from one side only) by solving each equation. All three have Jacobian 0 there. What does Hartman–Grobman say about them?
   2. For the damped pendulum, find the damping c at which the bottom rest point changes from a spiral to a node, and show that the top is a saddle for every $c \geq 0$.
   3. Sketch the portrait near the bottom for $c = 0.5$, $c = 2$ and $c = 4$, and describe how a released pendulum moves in each case.

9. code **Order on a nonlinear field.** The text measured the orders on a linear field, where they follow from the stability function. Here only the Taylor argument explains them (the exercise is adapted from Särkkä and Solin (2019)). Implement Heun's method beside the text's `euler` and `rk4`. Integrate the damped pendulum with $c = 0.5$ from $(\phi, \omega) = (2.5, 0)$ to $T = 5$ with $n = 16, 32, \dots, 1024$ steps, using `rk4` with 2^{16} steps as the reference. Plot the endpoint errors against h on log–log axes and fit the slopes. Which method reaches an error of 10^{-6} with the fewest field evaluations? Then set $c = 0$ and integrate to $T = 50$ at $h = 0.1$, plotting the energy $\frac{1}{2}\omega^2 + (1 - \cos\phi)$ of each method against time. Which methods drift, and in which direction?

10. code **Implicit steps on a stiff nonlinear field.** ⊠ The text's stiff system is linear and diagonal, so each backward-Euler step was one linear solve. Take instead the heavily damped pendulum, $c = 50$, from $(\phi, \omega) = (1, 0)$ to $T = 2$.
    1. Compute the Jacobian eigenvalues at the bottom and the step-size limits of forward Euler and RK4 that they imply.
    2. Show that after the fast transient the motion nearly satisfies $\dot{\phi} = -\sin\phi/c$, and compute $\phi(2)$ from this one-dimensional equation.
    3. One backward-Euler step solves $\mathbf{g}(\mathbf{y}) = \mathbf{y} - \mathbf{x}_n - h\mathbf{f}(\mathbf{y}) = \mathbf{0}$ for $\mathbf{y} = \mathbf{x}_{n+1}$. Write Newton's iteration $\mathbf{y} \leftarrow \mathbf{y} - (\mathbf{I} - h\mathbf{J}(\mathbf{y}))^{-1}\mathbf{g}(\mathbf{y})$ for it, and implement backward Euler with three Newton iterations per step, started from $\mathbf{x}_n$.

4. Run forward Euler, RK4 and backward Euler at $h = 0.1$, 0.06, 0.055 and 0.02, against a reference computed with `rk4` and 2^{16} steps. Count the field and Jacobian evaluations each method needs for an endpoint error below 10^{-4}. Which method wins, and why?

38.2 Neural ODEs, Adjoints, and Continuous Flows

A residual block adds a correction to its input, and so does a step of Euler's method (Section 38.1.3). If a residual network (Section 7.4) is a numerical solution of a differential equation, what do we gain by taking the equation itself as the model, and what does it cost?

Generative modelling gives one answer. A normalizing flow is a network that can be inverted, so that every output comes from exactly one input. The density of its outputs then follows from the change-of-variables formula, which asks for the determinant of the network's Jacobian at every sample, a matrix with 3072 rows for a small colour image of 32×32 pixels. Discrete flows avoid this cost by restricting the form of their layers. A flow in continuous time avoids it differently: the determinant turns into a trace, and the trace can be estimated with one backward pass.

The model is a differential equation $\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, t)$ whose solution at a final time is the output, a *Neural ODE* (Chen et al. 2018). Training it and computing its density each come down to integrating one more equation alongside the state. For training, that equation is the *adjoint* equation, which optimal control developed in the 1950s to ask how the cost of a trajectory changes with its controls (Pontryagin et al. 1962). For the density, it is an integral of the trace of the Jacobian, and a model that computes it is a *continuous normalizing flow* (Grathwohl et al. 2019). Compared with a fixed residual stack, the model gains an invertible map, a computable density, and a step count that is no longer part of the model. It loses the ability to reverse order on the line, and each gain has a catch that this section measures: a trained field is tied to the solver it was trained with, the memory-free gradient drifts, and the cheap trace is noisy.

We train such a model through a solver, derive the adjoint and compare it with backpropagation, and then follow a density along a flow. Three results are worth remembering: the adjoint equations, the fact that backpropagation through a solver discretizes them, and the instantaneous change of variables Equation 38.18. A reader who wants only the last, on which the likelihoods of diffusion models rest, can go straight to Section 38.2.3. Order preservation and the subsection on memory and drift can be skipped on a first reading. Notation: the field $\mathbf{f}_\theta$ has parameters θ and Jacobian $\mathbf{J} = \partial \mathbf{f}/\partial \mathbf{x}$, Φ_t is its flow map, a solver takes N steps of size h to the horizon $T = Nh$, $\mathcal{L}$ is a loss, $\mathbf{a}(t)$ the adjoint and p_t the density.

```
import numpy as np
import torch
```

The adjoint and density checks reuse the spiral field $\mathbf{A}$ and the starting point $\mathbf{x}_0$ of Section 38.1, together with two tools from there: the matrix exponential $e^{\mathbf{B}s} = \mathbf{V} e^{\Lambda s} \mathbf{V}^{-1}$ from the eigendecomposition $\mathbf{B} = \mathbf{V} \Lambda \mathbf{V}^{-1}$, and the classical fourth-order Runge–Kutta solver (RK4).

```
A = np.array([[-0.5, -1.0], [1.0, -0.5]])      # spiral sink: eigenvalues -0.5 +- i
x0 = np.array([2.0, 0.0])

def expm_eig(B, s):
    """e^{Bs} via the eigendecomposition B = V diag(mu) V^{-1}."""
    mu, V = np.linalg.eig(B)
    return (V @ np.diag(np.exp(mu * s)) @ np.linalg.inv(V)).real

def rk4(f, x, T, n):
    """n classical RK4 steps of dx/dt = f(x) from time 0 to time T."""
    h = T / n
    for _ in range(n):
        k1 = f(x)
        k2 = f(x + h / 2 * k1)
        k3 = f(x + h / 2 * k2)
        k4 = f(x + h * k3)
        x = x + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4)
    return x
```

38.2.1 Neural ODEs

A Neural ODE is what a residual network becomes when its depth turns into time.

38.2.1.1 From Residual Blocks to a Continuous Model

A residual blockWe start from the network we already have, a stack of residual blocks, and ask which continuous model it approximates. A residual block (He et al. 2016) and a forward-Euler step of the ODE $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ read

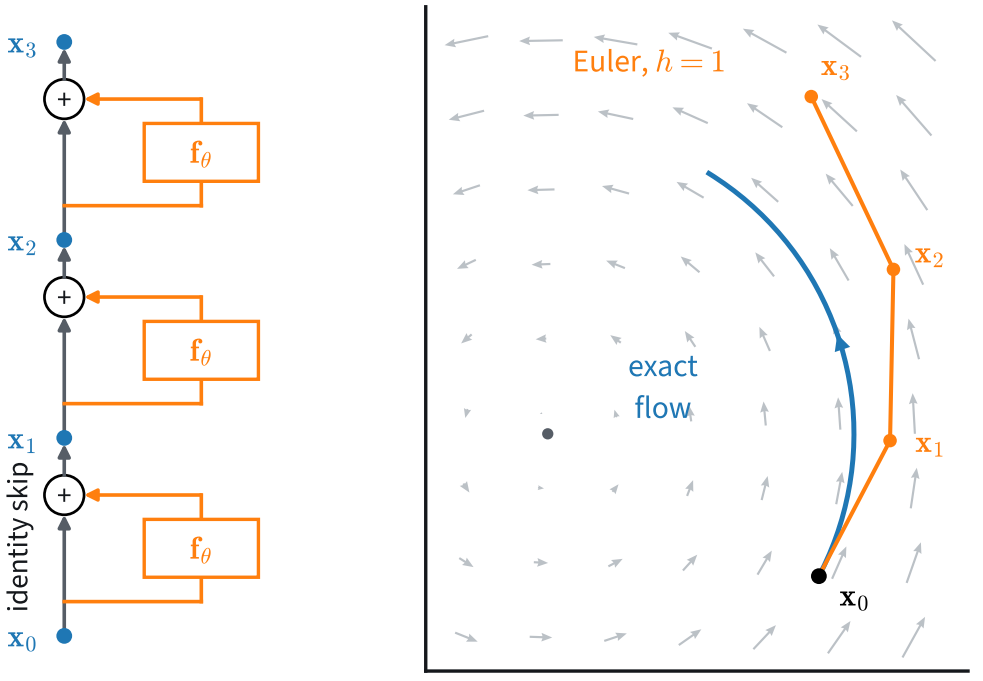
$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathbf{f}_{\theta_l}(\mathbf{x}_l) \qquad \text{and} \qquad \mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n, t_n). \tag{38.11}$$

With $h = 1$ and $\mathbf{f}(\cdot, t_l) = \mathbf{f}_{\theta_l}$ the two updates are the same formula. A ResNet whose blocks have different weights is therefore a time-dependent field sampled at the layer times and integrated by Euler's method with unit steps (Figure 38.6 draws the case in which all blocks share one field).

To take the limit, keep one parameter vector θ for all times and let the steps shrink. (The field may still depend on t; a ResNet with distinct blocks is a field that varies in time.) With N steps of size $h = T/N$ at the times $t_n = nh$,

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}_\theta(\mathbf{x}_n, t_n), \qquad \mathbf{x}_N \xrightarrow{N\to\infty} \mathbf{x}(T) = \mathbf{x}_0 + \int_0^T \mathbf{f}_\theta(\mathbf{x}(t), t)\,dt. \tag{38.12}$$

The convergence follows from Euler's global error bound in Section 38.1.3: for a smooth field that is Lipschitz in $\mathbf{x}$, the Euler iterate after N steps is within $O(h)$ of the exact solution. The **Neural ODE** is the limit object: the model whose output is the solution $\mathbf{x}(T)$ of $\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, t)$ with $\mathbf{x}(0) = \mathbf{x}_0$. The parameters define the field. The solver (Euler, RK4, an adaptive method) is a choice made afterwards, which changes the computed map, its cost and its error, but not the model.



$$\mathbf{x}_{l+1} = \mathbf{x}_l + \mathbf{f}_\theta(\mathbf{x}_l) \quad \Leftrightarrow \quad \mathbf{x}_{n+1} = \mathbf{x}_n + h\,\mathbf{f}(\mathbf{x}_n) \quad \text{with } h = 1$$

Figure 38.6: Left: a stack of residual blocks that share one field, each adding the learned branch to an identity skip. Right: the same updates read as forward-Euler steps of size $h = 1$, beside the exact flow of the same field, from which they drift.

The exact map and the computed map can differ in kind, not only in accuracy. The field $\dot{x} = -x$ has the exact flow $\Phi_t(x) = e^{-t}x$, a bijection of $\mathbb{R}$ for every t. One Euler step with $h = 1$ computes $x + 1 \cdot (-x) = 0$ and maps every input to zero. The exact map, by contrast, is always invertible when the field is continuous in t and globally Lipschitz in $\mathbf{x}$: the reversed path $\mathbf{y}(s) = \mathbf{x}(T - s)$ solves $\dot{\mathbf{y}} = -\mathbf{f}(\mathbf{y}, T - s)$, whose field is again Lipschitz, so it is the only solution from $\mathbf{x}(T)$ (Section 38.1.1.3) and it recovers the start.

38.2.1.2 Order Preservation on the Line

Invertibility is a gain, but it is also a constraint. Is a Neural ODE at least as expressive as the residual stack it came from? On the real line it is not, and a picture shows why. In the (t, x) plane, trajectories of a Lipschitz field never cross. Two trajectories that swapped -1 and 1 would have to meet on the way (panel (a) of Figure 38.7).

Proposition (flows on $\mathbb{R}$ preserve order). *Let $f(x, t)$ be continuous in t and globally Lipschitz in x, and write $\Phi_t(x)$ for the solution at time t started from x at time 0. Then for every $T > 0$ the map Φ_T is strictly increasing.*

Proof. Let $x < y$ and suppose $\Phi_T(x) \geq \Phi_T(y)$. The function $g(t) = \Phi_t(y) - \Phi_t(x)$ is continuous with $g(0) > 0$ and $g(T) \leq 0$, so by the intermediate value theorem $g(s) = 0$ for some $s \in (0, T]$: the two trajectories meet at time s at a point z. Now run time backward from s. Both reversed trajectories $\tau \mapsto \Phi_{s-\tau}(x)$ and $\tau \mapsto \Phi_{s-\tau}(y)$ solve the same initial-value problem $\dot{w} = -f(w, s - \tau)$, $w(0) = z$, whose field is again Lipschitz. By Picard–Lindelöf that problem has only one solution, so the two reversed trajectories coincide, and at $\tau = s$ this says $x = y$, a contradiction. ■

The proof has one idea, to apply uniqueness backward from the meeting time. As a consequence, no Neural ODE on $\mathbb{R}$ realizes $x \mapsto -x$: that map sends -1 to 1 and 1 to -1, which reverses their order.

The residual block $x + f(x)$ with $f(x) = -2x$ does realize it, in one step (panel (b) of Figure 38.7). That block is the Euler step with $h = 1$ of $\dot{x} = -2x$, whose exact flow $e^{-2t}x$ never reverses order. The two statements agree because one Euler step $x + hf(x)$ of a field with Lipschitz constant L is guaranteed to be increasing when $hL < 1$: for $x > y$,

$$(x + hf(x)) - (y + hf(y)) \;\geq\; (x - y) - hL\,(x - y) \;=\; (1 - hL)(x - y),$$

which is positive when $hL < 1$. Here $hL = 2$, so the step may reverse order. A finite step is not a flow, and the proposition does not constrain it.

One more dimension removes the obstruction. Let $\mathbf{K} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$, which satisfies $\mathbf{K}^2 = -\mathbf{I}$, so that the exponential series splits into $e^{s\mathbf{K}} = \cos s\,\mathbf{I} + \sin s\,\mathbf{K}$, the rotation by the angle s. The field $\dot{\mathbf{x}} = \pi\mathbf{K}\mathbf{x}$ has the time-one map

$$\Phi_1 = e^{\pi\mathbf{K}} = \cos\pi\,\mathbf{I} + \sin\pi\,\mathbf{K} = -\mathbf{I}.$$

It carries $(x, 0)$ to $(-x, 0)$ along half circles, and trajectories pass each other in the plane without meeting (panel (c) of Figure 38.7). Adding dimensions in this way is called *augmentation* (Dupont et al. 2019): the input x is embedded as $(x, 0)$, the flow runs in the plane, and the first coordinate is read out. A flow is a continuous rearrangement of space, and on a line there is no room for two points to pass.

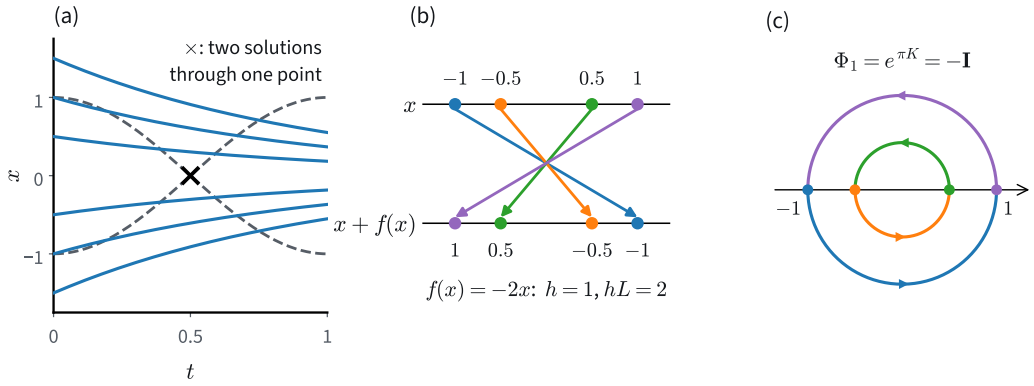


Figure 38.7: (a) On the line, trajectories of a Lipschitz field never cross (here $\dot{x} = -x$), so the time-T map preserves order; the dashed pair that swaps -1 and 1 would need two solutions through one point. (b) The residual step $x + f(x)$ with $f(x) = -2x$ swaps the points in one step: it is an Euler step with $hL = 2$, not a flow. (c) One extra dimension removes the obstruction: the rotation by π carries $(x, 0)$ to $(-x, 0)$ along half circles that pass without meeting (colours as in (b)).

38.2.1.3 Training Through the Solver

The forward pass of a Neural ODE is a loop of N solver steps. The direct way to train it is to *discretize, then differentiate*: write the solver into the computation graph, compute the loss on $\mathbf{x}_N$, and backpropagate through the unrolled steps. With Euler as the solver the program is a ResNet of N blocks that share one set of weights, so ordinary automatic differentiation trains it.

The task is deliberately easy: carry 64 points of the unit circle to the matching points of the ellipse centred at $(1.5, 0)$ with semi-axes 1.2 and 0.4, a stretch and a shift. The exact flow of the affine field $\dot{\mathbf{x}} = \operatorname{diag}(\ln 1.2, \ln 0.4)\,\mathbf{x} + \mathbf{c}$ does this without error for a suitable $\mathbf{c}$. The question is whether training finds a flow or only a ten-step program. The field has one hidden layer of 32 tanh units, the loss is the mean squared error at the endpoints, the solver takes $N = 10$ Euler steps of size $h = 0.1$, and Adam (Section 35.5) runs 401 full-batch iterations at learning rate 0.05, lowered to 0.005 from iteration 300.

```
angle = np.linspace(0, 2 * np.pi, 64, endpoint=False)
X0_ring = np.stack([np.cos(angle), np.sin(angle)], axis=1)           # circle
X1_ring = np.stack([1.5 + 1.2 * np.cos(angle), 0.4 * np.sin(angle)], axis=1)
X, Y = torch.tensor(X0_ring).float(), torch.tensor(X1_ring).float()
net = torch.nn.Sequential(torch.nn.Linear(2, 32), torch.nn.Tanh(),
                          torch.nn.Linear(32, 2))
for layer in net[::2]:                          # Glorot-uniform weights, zero biases
    torch.nn.init.xavier_uniform_(layer.weight)
    torch.nn.init.zeros_(layer.bias)

def flow(X, N=10, h=0.1, sign=1.0):
    """N Euler steps of size h; sign=-1 integrates the field backward."""
    for _ in range(N):                          # one Euler step = one residual block
```

```
            X = X + sign * h * net(X)
        return X

    opt = torch.optim.Adam(net.parameters(), lr=0.05)
    sched = torch.optim.lr_scheduler.MultiStepLR(opt, [300], 0.1)  # 0.005 from 300
    for it in range(401):
        loss = ((flow(X) - Y) ** 2).mean()
        opt.zero_grad()
        loss.backward()
        opt.step()
        sched.step()
        if it % 200 == 0:
            print(f'iter {it:3d}  loss {loss.item():.1e}')
```

```
iter   0  loss 1.4e+00
iter 200  loss 5.5e-06
iter 400  loss 1.2e-06
```

The loss falls from order one to well below 10^{-4}, typically a few times 10^{-6}.

Two tests tell a flow from a ten-step program. First, integrate the trained field with $N = 100$ and $N = 1000$ steps ($Nh = 1$ throughout); the larger N, the closer the computed map is to the exact time-one flow of the field. Before running the cell, guess what this does to the endpoint error. Second, integrate forward N steps and then backward N steps with $\mathbf{x} \leftarrow \mathbf{x} - h\,\mathbf{f}_\theta(\mathbf{x})$, and measure how far the round trip lands from where it started.

```
    with torch.inference_mode():
        for n in (10, 100, 1000):
            fwd = flow(X, n, 1 / n)
            back = flow(fwd, n, 1 / n, sign=-1.0)
            err = torch.linalg.vector_norm(fwd - Y, dim=1).mean()
            trip = torch.linalg.vector_norm(back - X, dim=1).mean()
            print(f'N = {n:4d}: endpoint error {err:.1e}, '
                  f'forward-backward round trip {trip:.1e}')
```

```
N =   10: endpoint error 1.4e-03, forward-backward round trip 5.5e-02
N =  100: endpoint error 1.7e-02, forward-backward round trip 5.6e-03
N = 1000: endpoint error 1.9e-02, forward-backward round trip 5.6e-04
```

At the training step count the endpoints lie within 0.01 of their targets, typically a few thousandths. With finer steps the endpoint error *rises* to about 0.02 and then stops changing. Training fitted the ten-step program. The exact time-one flow of the same field, which $N = 1000$ approximates, lands about 0.02 from the targets, under 2% of the ellipse's long semi-axis. The field defines a flow close to its ten-step program, not equal to it. For the step count to be free after training, the field must be fitted to the flow and not to one program: by training through a finer or higher-order solver, or by a regression that involves no solver at all, which is how Section 38.5 and Section 38.6 learn their fields.

The round trip misses its start by about 0.055 at $N = 10$, 0.0056 at $N = 100$ and 0.00056 at $N = 1000$, tenfold less per tenfold more steps. Each pair of forward and backward Euler steps leaves a residual of order h^2, and N of them add up to $O(h)$, Euler's order. The exact flow is invertible, whereas the numerical map becomes invertible only in the limit $h \to 0$.

38.2.2 The Adjoint Method

Backpropagation through N solver steps gives the gradient of the N-step program. The adjoint method differentiates the exact flow instead, by integrating one linear ODE backward in time. It is first of all what backpropagation through a solver converges to as the steps shrink. It is also an alternative implementation: backpropagation stores the N intermediate states, whereas the adjoint ODE can recompute them on the way back, at a loss of stability that the last subsection quantifies. Its central object is the *adjoint state* $\mathbf{a}(t) = \partial\mathcal{L}/\partial\mathbf{x}(t)$, the gradient of the loss with respect to the state at time t. This is the adjoint $\bar{a}$ of Equation 34.30, the total derivative of the output with respect to a node, with one node $\mathbf{x}(t)$ for every instant: reverse mode carries it from layer to layer (Section 34.5), and here it is carried through time.

38.2.2.1 The Scalar Case

Before the general statement we work one example to the end, an example in which every quantity has a closed form. Take $\dot{x} = \theta x$ with $x(0) = x_0$ and the loss $\mathcal{L} = \frac{1}{2}x(T)^2$ on the endpoint. Directly, $x(T) = e^{\theta T}x_0$, so $\partial x(T)/\partial\theta = T\,x(T)$ and

$$\frac{\partial\mathcal{L}}{\partial\theta} = x(T)\,\frac{\partial x(T)}{\partial\theta} = T\,x(T)^2.$$

With $\theta = T = x_0 = 1$ this is $e^2 = 7.389$.

The adjoint reaches the same number by another route. A nudge δ to the state at time t grows to $e^{\theta(T-t)}\delta$ by time T and changes the loss by $x(T)\,e^{\theta(T-t)}\delta$, so $a(t) = e^{\theta(T-t)}x(T)$. This adjoint solves the ODE $\dot{a} = -\theta a$ backward from the final value $a(T) = x(T)$, which is the derivative of the loss $\ell(x) = \frac{1}{2}x^2$ at $x(T)$. A change of θ pushes the trajectory at the rate $\partial f/\partial\theta = x$, and the adjoint converts each push into a change of the loss. Summing over time,

$$\int_0^T a(t)\,\frac{\partial f}{\partial\theta}\,dt = \int_0^T e^{\theta(T-t)}x(T)\;e^{\theta t}x_0\,dt = \int_0^T x(T)^2\,dt = T\,x(T)^2. \tag{38.13}$$

The integrand is constant: x grows forward, a shrinks by the same factor when read forward, and their product stays at $x(T)^2$ (Figure 38.8). The gradient is the area under it.

38.2.2.2 The Adjoint Equations

The general case keeps the shape of the scalar one: an adjoint that runs backward from the gradient of the loss, and a parameter gradient that integrates the adjoint against $\partial\mathbf{f}/\partial\theta$.

Proposition (the adjoint equations). *Let* $\mathbf{f}$ *be continuously differentiable in* $\mathbf{x}$ *and* θ*, let* $\mathbf{x}(t)$ *solve* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\theta,t)$ *(the field* $\mathbf{f}_\theta$*, with its parameters written as an argument) on* $[0,T]$ *from a fixed* $\mathbf{x}(0)$*, and let* $\mathcal{L} = \ell(\mathbf{x}(T))$ *for a differentiable* ℓ*. Then the adjoint state satisfies the linear ODE*

$$\dot{\mathbf{a}}(t) = -\left(\frac{\partial\mathbf{f}}{\partial\mathbf{x}}\right)^{\top}\mathbf{a}(t), \qquad \mathbf{a}(T) = \nabla\ell(\mathbf{x}(T)), \tag{38.14}$$

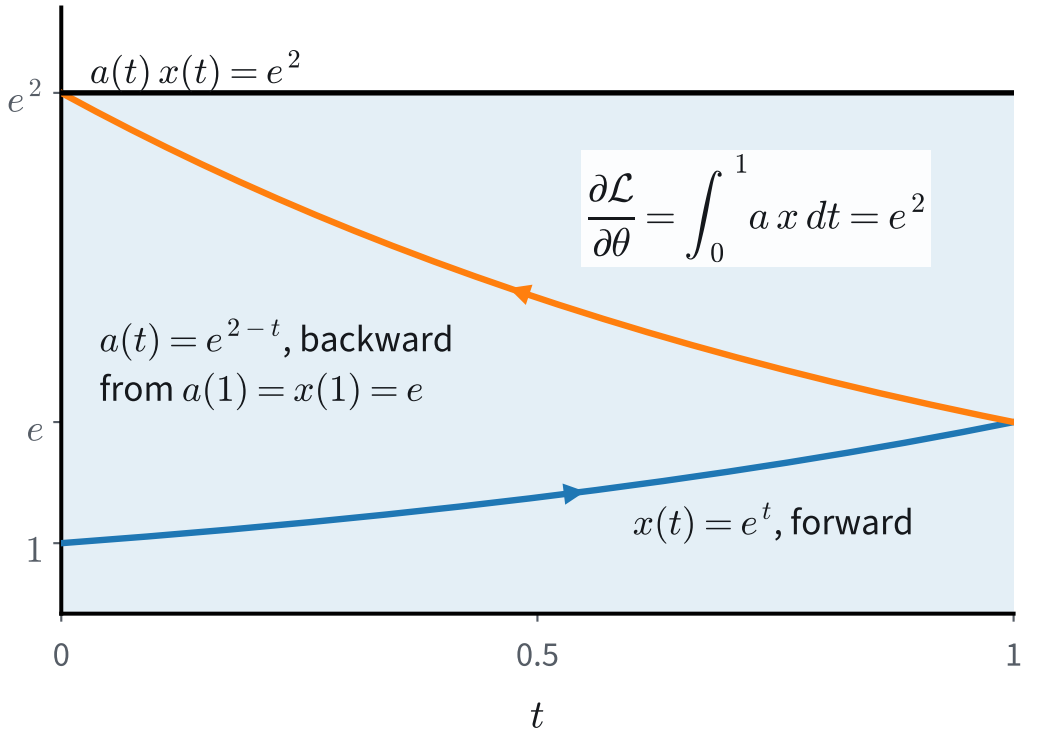


Figure 38.8: The scalar adjoint for $\dot{x} = \theta x$ at $\theta = 1, x_0 = 1, T = 1$: the state grows forward, the adjoint is integrated backward from the loss gradient $a(1) = x(1)$, and their product is constant; the parameter gradient is the shaded area, $Tx(T)^2 = e^2$.

integrated backward from T, and the parameter gradient is the integral

$$\frac{\partial \mathcal{L}}{\partial \theta} = \int_0^T \left(\frac{\partial \mathbf{f}}{\partial \theta}\right)^\top \mathbf{a}(t)\, dt, \tag{38.15}$$

with both partial derivatives evaluated along the trajectory $(\mathbf{x}(t), \theta, t)$ (Chen et al. 2018).

Proof (variational). Perturb the parameters by an infinitesimal $\delta\theta$. Because $\mathbf{f}$ is C^1, the solution is differentiable in θ, and the first-order perturbation $\delta\mathbf{x}(t)$ of the trajectory obeys the *linearization* of the dynamics (differentiate the ODE in θ and use the chain rule):

$$\delta\dot{\mathbf{x}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta, \qquad \delta\mathbf{x}(0) = \mathbf{0}.$$

Define $\mathbf{a}(t)$ by the backward ODE Equation 38.14 and differentiate the pairing $\mathbf{a}^\top \delta\mathbf{x}$:

$$\frac{d}{dt}\left(\mathbf{a}^\top \delta\mathbf{x}\right) = \dot{\mathbf{a}}^\top \delta\mathbf{x} + \mathbf{a}^\top \delta\dot{\mathbf{x}} = -\mathbf{a}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \mathbf{a}^\top \frac{\partial \mathbf{f}}{\partial \mathbf{x}}\,\delta\mathbf{x} + \mathbf{a}^\top \frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta = \mathbf{a}^\top \frac{\partial \mathbf{f}}{\partial \theta}\,\delta\theta.$$

The state-coupling terms cancel, and the adjoint ODE was defined so that they would. Integrate from 0 to T, using $\delta\mathbf{x}(0) = \mathbf{0}$ at the left endpoint:

$$\mathbf{a}(T)^\top \delta\mathbf{x}(T) = \int_0^T \mathbf{a}(t)^\top \frac{\partial \mathbf{f}}{\partial \theta}\, dt\ \delta\theta.$$

The left side is $\nabla \ell(\mathbf{x}(T))^\top \delta\mathbf{x}(T) = \delta\mathcal{L}$, the first-order change of the loss; the right side is linear in $\delta\theta$ with coefficient Equation 38.15. It remains to identify $\mathbf{a}$ with the adjoint state. Run the same argument with $\delta\theta = \mathbf{0}$ and a nudge $\delta\mathbf{x}(s)$ injected at time s: the pairing $\mathbf{a}^\top \delta\mathbf{x}$ is then constant on $[s, T]$, so $\delta\mathcal{L} = \mathbf{a}(T)^\top \delta\mathbf{x}(T) = \mathbf{a}(s)^\top \delta\mathbf{x}(s)$, and the solution of Equation 38.14 is $\partial\mathcal{L}/\partial\mathbf{x}(s)$. ■

The backward ODE is linear in $\mathbf{a}$, with the transposed Jacobian as its matrix. Each of its steps computes $\mathbf{a}^\top \partial\mathbf{f}/\partial\mathbf{x}$, one vector–Jacobian product, which costs one reverse-mode pass through $\mathbf{f}$ (Section 34.5). (A ReLU field is only piecewise C^1, so the proposition does not cover it, although autograd still differentiates the unrolled solver piece by piece. Smooth activations such as tanh or GELU satisfy the hypothesis.)

38.2.2.3 Backpropagation as the Discrete Adjoint

Backpropagation through the Euler program can now be set beside the adjoint ODE, line by line. Differentiate the program $\mathbf{x}_{k+1} = \mathbf{x}_k + h\,\mathbf{f}(\mathbf{x}_k, \theta)$ by the chain rule, with $\mathbf{a}_k = \partial\mathcal{L}/\partial\mathbf{x}_k$. The Jacobian of one step is $\mathbf{I} + h\mathbf{J}(\mathbf{x}_k)$, so

$$\mathbf{a}_k = \left(\mathbf{I} + h\mathbf{J}(\mathbf{x}_k)\right)^\top \mathbf{a}_{k+1}, \qquad \mathbf{a}_N = \nabla \ell(\mathbf{x}_N), \qquad \frac{\partial \mathcal{L}}{\partial \theta} = \sum_{k=0}^{N-1} h \left(\frac{\partial \mathbf{f}}{\partial \theta}(\mathbf{x}_k)\right)^\top \mathbf{a}_{k+1}. \tag{38.16}$$

Compare this with one explicit Euler step of the adjoint ODE Equation 38.14 from t_{k+1} back to t_k:

$$\mathbf{a}_k = \mathbf{a}_{k+1} + h\,\mathbf{J}(\mathbf{x}_{k+1})^\top \mathbf{a}_{k+1}.$$

The two updates differ only in where the Jacobian is evaluated: at the start $\mathbf{x}_k$ of the forward step or at its end $\mathbf{x}_{k+1}$. The sum in Equation 38.16 is a rectangle rule for the integral Equation 38.15. So backpropagation through a solver is a *discrete adjoint*, and a solver applied to the adjoint ODE is the *continuous adjoint*, discretized. The literature calls the two routes discretize-then-optimize and optimize-then-discretize (Kidger 2022). The first is the exact gradient of the program, the second approximates the gradient of the flow, and for Euler they differ at first order in h. Of everything in this subsection, this is the statement to keep.

The scalar example shows both at once. For $\dot{x} = \theta x$ the Euler program is $x_N = (1+h\theta)^N x_0$, so $\partial x_N/\partial\theta = Nh(1+h\theta)^{N-1}x_0 = T\,x_N/(1+h\theta)$ and the discrete adjoint gives $\partial\mathcal{L}/\partial\theta = T\,x_N^2/(1+h\theta)$. With $\theta = T = x_0 = 1$ this is 6.116 at $N = 10$, 7.244 at $N = 100$ and 7.374 at $N = 1000$, against the exact $e^2 = 7.389$. The gaps 1.27, 0.145 and 0.0148 shrink tenfold per tenfold N.

The spiral field of Section 38.1, $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x}$ with $\mathbf{A} = \left(\begin{smallmatrix} -0.5 & -1 \\ 1 & -0.5 \end{smallmatrix}\right)$, gives the same check in two dimensions with every ingredient in closed form. Take θ to be the entries of $\mathbf{A}$ and $\mathcal{L} = \frac{1}{2}\|\mathbf{x}(T)\|^2$. The trajectory is $e^{\mathbf{A}t}\mathbf{x}_0$, the adjoint is $\mathbf{a}(t) = e^{\mathbf{A}^\top(T-t)}\mathbf{x}(T)$ (the solution of $\dot{\mathbf{a}} = -\mathbf{A}^\top\mathbf{a}$), and since $\partial\mathbf{f}/\partial A_{ij} = x_j\mathbf{e}_i$, the gradient Equation 38.15 becomes $\partial\mathcal{L}/\partial\mathbf{A} = \int_0^T \mathbf{a}(t)\,\mathbf{x}(t)^\top dt$. The cell writes the backward sweep Equation 38.16 by hand and tests it against central finite differences on the same program, for $T = 2$ and $N = 1000$ steps.

```
def loss_and_grad_unrolled(A, x0, T, n):
    """L = ||x_n||^2 / 2 through n unrolled Euler steps; dL/dA by hand-written
    reverse mode, the discrete adjoint."""
    h, xs = T / n, [x0]
    for k in range(n):
        xs.append(xs[-1] + h * (A @ xs[-1]))   # forward sweep, tape the states
    a, gA = xs[-1].copy(), np.zeros_like(A)    # a_n = dL/dx_n = x_n
    for k in range(n - 1, -1, -1):             # backward sweep: one VJP per step
        gA += h * np.outer(a, xs[k])           # dL/dA += h a_{k+1} x_k^T
        a = a + h * (A.T @ a)                  # a_k = (I + hA)^T a_{k+1}
    return 0.5 * xs[-1] @ xs[-1], gA

T, n = 2.0, 1000
_, g_disc = loss_and_grad_unrolled(A, x0, T, n)
g_fd = np.zeros_like(A)                        # central finite differences
for i in range(2):
    for j in range(2):
        E = np.zeros_like(A); E[i, j] = 1e-5
        g_fd[i, j] = (loss_and_grad_unrolled(A + E, x0, T, n)[0]
                      - loss_and_grad_unrolled(A - E, x0, T, n)[0]) / 2e-5
print(f'discrete adjoint vs finite differences: {np.abs(g_disc - g_fd).max():.1e}')
```

```
discrete adjoint vs finite differences: 1.2e-10
```

The two agree to about 10^{-10}, the accuracy of the finite differences themselves, so the sweep is coded correctly. The distance that matters is the one to the gradient of the exact flow. For it we evaluate the continuous adjoint integral by the trapezoid rule on 2000 intervals, with

a and **x** from the matrix exponential, and compare it with the discrete adjoint for three step counts.

```
n_quad = 2000                                  # continuous adjoint, trapezoid rule
ts = np.linspace(0.0, T, n_quad + 1)
xT = expm_eig(A, T) @ x0
F = np.stack([np.outer(expm_eig(A.T, T - s) @ xT, expm_eig(A, s) @ x0)
              for s in ts])                    # a(s) x(s)^T
g_cont = (T / n_quad) * (F[1:-1].sum(axis=0) + 0.5 * (F[0] + F[-1]))
for n_steps in (100, 1000, 10000):
    gap = np.abs(loss_and_grad_unrolled(A, x0, T, n_steps)[1] - g_cont).max()
    print(f'n = {n_steps:5d} Euler steps: |discrete - continuous adjoint| = {gap:.1e}')
```

```
n =   100 Euler steps: |discrete - continuous adjoint| = 2.9e-02
n =  1000 Euler steps: |discrete - continuous adjoint| = 2.8e-03
n = 10000 Euler steps: |discrete - continuous adjoint| = 2.8e-04
```

The gap falls from 2.9×10^{-2} to 2.8×10^{-3} to 2.8×10^{-4}, tenfold per tenfold N, as in the scalar case.

38.2.2.4 Memory and Drift

The adjoint ODE needs $\mathbf{x}(t)$ at every backward step, because the Jacobian is evaluated there. There are two ways to supply it. Store the forward states, which costs memory proportional to N, or re-integrate $\dot{\mathbf{x}} = \mathbf{f}$ backward from $\mathbf{x}(T)$ alongside $\mathbf{a}$, which keeps the memory constant in N at the cost of one more field evaluation per step. The smallest case shows that the second option amplifies errors.

For $\dot{x} = -\lambda x$ with $\lambda > 0$, the backward integration solves $\dot{x} = +\lambda x$. An absolute error δ in the stored $x(T)$ therefore becomes $e^{\lambda T}\delta$ by the time it reaches $t = 0$. Take $\lambda = 50$, the fast decay rate of the stiff example of Section 38.1.3.3, as one direction of a state whose coordinates are of size one, and not along a coordinate axis. Each stored coordinate is rounded at about 10^{-16} in double precision, and the rounding has a component of that size along the fast direction, so $\delta = 10^{-16}$. The reconstructed start is then off by 2.7×10^{-11} at $T = 0.25$, by 7×10^{-6} at $T = 0.5$ and by 5×10^{5} at $T = 1$, larger than the state itself (4 measures this growth).

For the spiral, whose eigenvalues have real part -0.5, the factor over $T = 2$ is only e. A strongly contracting forward flow is a strongly expanding backward one, so re-integration saves memory at the cost of drift. For stiff fields *checkpointing* is the sturdier choice: it stores the state only at a few checkpoints and recomputes the states in between.

38.2.3 Continuous Normalizing Flows

A Neural ODE moves points. A generative model must also say how probable each output is, so we now follow a density along the flow.

38.2.3.1 Density Along a Flow

Pushing samples through a flow turns it into a generator: draw $\mathbf{x}(0)$ from a simple density p_0 and integrate. (The generators of Section 38.5 and Section 38.6 run such a flow backward in time, from noise at the final time to data at $t = 0$; the density formulas below hold

in either direction.) Training such a generator by maximum likelihood needs the density of what comes out. A *normalizing flow* is an invertible network trained in this way, and its density follows from the change-of-variables formula of Section 36.2: for an invertible differentiable map $\mathbf{g}$ with Jacobian $\mathbf{J}_{\mathbf{g}}$,

$$\log p_{\text{out}}(\mathbf{g}(\mathbf{x})) = \log p_{\text{in}}(\mathbf{x}) - \log \left|\det \mathbf{J}_{\mathbf{g}}(\mathbf{x})\right| . \tag{38.17}$$

The log-density drops by the logarithm of the factor by which $\mathbf{g}$ stretches volume near $\mathbf{x}$. A flow built from many layers needs one log-determinant per layer and sample. A general determinant costs $O(d^3)$ operations, about 2×10^{10} for the $d = 3072$ of a 32×32 colour image. Discrete flows therefore restrict their layers to triangular Jacobians, whose determinant is the product of the diagonal, or to identity-plus-low-rank Jacobians, such as $\det(\mathbf{I} + \mathbf{u}\mathbf{v}^\top) = 1 + \mathbf{v}^\top\mathbf{u}$. Both determinants cost $O(d)$.

In continuous time the formula simplifies. Start with one dimension. The field $\dot{x} = \theta x$ on $\mathbb{R}$ has the flow $\Phi_t(x) = e^{\theta t}x$, which stretches every interval by $e^{\theta t}$. A density carried along is divided by the stretch, so $\log p_t(x(t)) = \log p_0(x_0) - \theta t$ and $\frac{d}{dt}\log p_t(x(t)) = -\theta$. The number θ is the 1×1 Jacobian of the field, and therefore also its trace.

The spiral is the running example in two dimensions. Since $\mathbf{A} = -\frac{1}{2}\mathbf{I} + \mathbf{K}$ and $\mathbf{I}$ commutes with $\mathbf{K}$, $e^{\mathbf{A}t} = e^{-t/2}e^{t\mathbf{K}}$, a rotation followed by a shrink by $e^{-t/2}$. A disc is carried to a disc of $e^{-t/2}$ times its radius, so its area is multiplied by e^{-t}, and the density along each trajectory rises by e^{t}. Here $\operatorname{tr}\mathbf{A} = -1$, so the rate of change of the log-density is again minus the trace. Figure 38.9 draws this beside a field that preserves area and one that expands it.

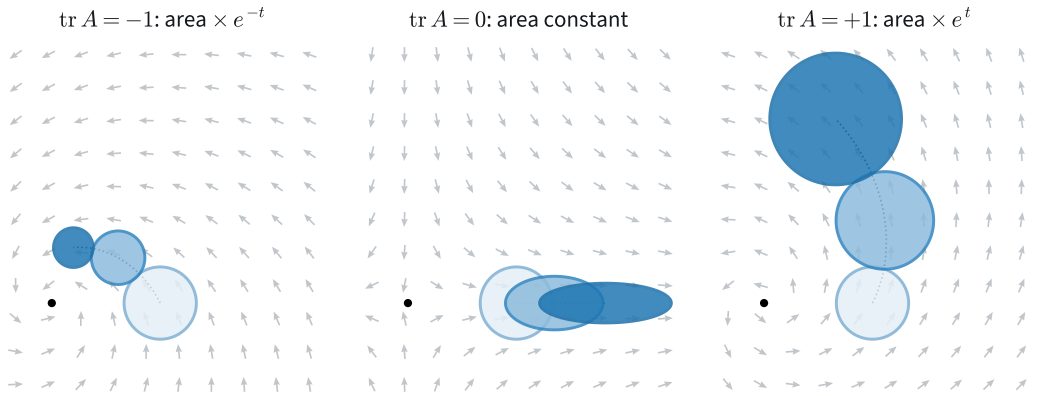


Figure 38.9: A disc of samples carried by the exact flows of three linear fields, shown at $t = 0$, 0.6 and 1.2 (darker is later); the dotted line is the path of the centre. The density along a trajectory changes by the factor $e^{-t\operatorname{tr}\mathbf{A}}$: it rises where the flow contracts (left, the spiral), stays constant under the area-preserving saddle although the shape changes (middle), and falls where the flow expands (right).

The general case needs one fact about determinants: over a short step the flow map is close to the identity, and the determinant of a matrix close to the identity is governed by its trace.

Lemma. *For any square matrix* $\mathbf{J}$, $\det(\mathbf{I} + h\mathbf{J}) = 1 + h\operatorname{tr}\mathbf{J} + O(h^2)$.

Proof. In the Leibniz expansion of the determinant, any permutation other than the identity

uses at least two off-diagonal entries, each of size $O(h)$, so it contributes $O(h^2)$. The identity permutation contributes $\prod_i (1 + hJ_{ii}) = 1 + h\sum_i J_{ii} + O(h^2)$. ■

Proposition (instantaneous change of variables). *Let $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t)$ with $\mathbf{f}$ continuously differentiable in $(\mathbf{x}, t)$ and globally Lipschitz in $\mathbf{x}$, so that each flow map Φ_t is a C^1 bijection with a C^1 inverse (the solution of a C^1 field is C^1 in its initial value (Hairer et al. 1993), and the inverse runs the field backward). Let $\mathbf{x}(0) \sim p_0$ for a positive differentiable density p_0. Then the density p_t of $\mathbf{x}(t)$ exists and is positive, and along each trajectory* (Chen et al. 2018)

$$\frac{d}{dt} \log p_t(\mathbf{x}(t)) = -\operatorname{tr}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)(\mathbf{x}(t), t). \tag{38.18}$$

Proof. The density exists by Equation 38.17 applied to the bijection Φ_t, and it is positive because p_0 is and the Jacobian of a map with a differentiable inverse is nonsingular. Over a short interval the flow advances by the map $\Phi_{t\to t+h}(\mathbf{x}) = \mathbf{x} + \int_t^{t+h} \mathbf{f}(\mathbf{x}(s), s)\, ds$. Differentiating in $\mathbf{x}$ and using the continuity of $\mathbf{J}$, its Jacobian is $\mathbf{I} + h\mathbf{J} + o(h)$, where $\mathbf{J} = \partial \mathbf{f}/\partial \mathbf{x}$ is evaluated at $(\mathbf{x}(t), t)$ and $o(h)$ stands for terms that vanish faster than h. Apply Equation 38.17 to this one map and expand with the Lemma, which tolerates the perturbation: replacing $h\mathbf{J}$ by $h\mathbf{J} + o(h)$ changes the determinant only by $o(h)$. For small h the determinant is close to 1, hence positive, so the absolute value can be dropped:

$$\log p_{t+h}(\mathbf{x}(t+h)) = \log p_t(\mathbf{x}(t)) - \log\det(\mathbf{I} + h\mathbf{J} + o(h)) = \log p_t(\mathbf{x}(t)) - h \operatorname{tr}\mathbf{J} + o(h),$$

using $\log(1+u) = u + O(u^2)$. Subtract $\log p_t(\mathbf{x}(t))$, divide by h, and let $h \to 0$. ■

The trace of the Jacobian is the *divergence* of the field, the local rate at which the flow expands volume. Where the field spreads samples apart ($\operatorname{tr}\mathbf{J} > 0$) the log-density along each trajectory falls, and where it compresses them, it rises. Integrating Equation 38.18 along a trajectory gives

$$\log p_T(\mathbf{x}(T)) = \log p_0(\mathbf{x}(0)) - \int_0^T \operatorname{tr}\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right) dt. \tag{38.19}$$

The per-layer sum of log-determinants of a discrete flow has become an integral of traces. A model that computes it, by integrating the augmented state $(\mathbf{x}, \log p)$ with any ODE solver, is a **continuous normalizing flow** (CNF). It needs no restriction on the architecture of $\mathbf{f}$. To evaluate the density of a given point $\mathbf{x}(T)$, integrate the field backward from it to $\mathbf{x}(0)$, accumulate the trace on the way, and read off p_0 there. Likelihoods of diffusion models are computed in this way (Section 17.5).

For a linear field the trace integral is $t \operatorname{tr}\mathbf{A}$, and Equation 38.19 compared with Equation 38.17 for the map $e^{\mathbf{A}t}$ says $\log\det e^{\mathbf{A}t} = t \operatorname{tr}\mathbf{A}$. This identity also follows in one line from the eigendecomposition $\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{-1}$:

$$\det e^{\mathbf{A}t} = \det\left(\mathbf{V} e^{\Lambda t}\mathbf{V}^{-1}\right) = \prod_j e^{\Lambda_{jj} t} = e^{t \operatorname{tr}\mathbf{A}} \tag{38.20}$$

for diagonalizable $\mathbf{A}$, and for every $\mathbf{A}$ by continuity, since diagonalizable matrices are dense. For the spiral both sides equal $e^{-1.5} = 0.2231$ at $t = 1.5$. The proposition follows individual trajectories. Stated for the whole density at once instead of along one trajectory, the same law is the continuity equation of fluid mechanics, which Section 38.4 derives together with the term that noise adds.

38.2.3.2 Checking the Trace Integral

For the spiral the proposition can be compared with a closed form. Start from $p_0 = \mathcal{N}(\mathbf{0}, \mathbf{I})$; a linear map sends it to $p_t = \mathcal{N}(\mathbf{0}, e^{\mathbf{A}t} e^{\mathbf{A}^\top t})$, which for the spiral is $\mathcal{N}(\mathbf{0}, e^{-t}\mathbf{I})$ because $e^{t\mathbf{K}}$ is orthogonal. The cell evaluates this Gaussian at the transported points of four random samples and compares with $\log p_0(\mathbf{x}_0) - t \operatorname{tr} \mathbf{A}$.

```
t = 1.5
X0s = np.random.randn(4, 2)                    # four samples from p_0 = N(0, I)
Et = expm_eig(A, t)
Xts = X0s @ Et.T                               # flow each sample to time t
Sig = Et @ Et.T                                # x(t) ~ N(0, E_t E_t^T)
logp0 = -0.5 * (X0s ** 2).sum(axis=1) - np.log(2 * np.pi)
logpt = (-0.5 * np.einsum('ki,ij,kj->k', Xts, np.linalg.inv(Sig), Xts)
         - 0.5 * np.log(np.linalg.det(Sig)) - np.log(2 * np.pi))
print('log p_t(x(t)), Gaussian formula :', logpt.round(6))
print('log p_0(x_0) - t tr(A)          :', (logp0 - t * np.trace(A)).round(6))
```

```
log p_t(x(t)), Gaussian formula : [-0.366901 -1.895157 -1.329256 -0.498344]
log p_0(x_0) - t tr(A)          : [-0.366901 -1.895157 -1.329256 -0.498344]
```

The two rows agree in every printed digit, as Equation 38.20 says they must; with a constant trace there was little to go wrong. A nonlinear field is the real test, because its trace changes along the path. Take the planar field $\mathbf{f}(\mathbf{x}) = \mathbf{W}_2 \tanh(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$ with 8 hidden units and random weights. Its Jacobian and trace are explicit:

$$\mathbf{J} = \mathbf{W}_2 \operatorname{diag}(\mathbf{s})\, \mathbf{W}_1, \qquad \mathbf{s} = 1 - \tanh^2(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1), \qquad \operatorname{tr} \mathbf{J} = \sum_i s_i \, (\mathbf{W}_1 \mathbf{W}_2)_{ii},$$

using the cyclic property $\operatorname{tr}\left(\mathbf{W}_2 \operatorname{diag}(\mathbf{s})\, \mathbf{W}_1\right) = \operatorname{tr}\left(\operatorname{diag}(\mathbf{s})\, \mathbf{W}_1 \mathbf{W}_2\right)$. The cell integrates the augmented state $(\mathbf{x}, \int \operatorname{tr} \mathbf{J}\, dt)$ with RK4 to $T = 1$ and compares the integral with $\log |\det D\Phi_T|$, computed from central finite differences of the flow map itself.

```
H = 8
W1, b1 = np.random.randn(H, 2), np.random.randn(H)
W2 = np.random.randn(2, H) / np.sqrt(H)
field = lambda x: W2 @ np.tanh(W1 @ x + b1)
trJ = lambda x: np.sum((1 - np.tanh(W1 @ x + b1) ** 2)
                       * np.einsum('ij,ji->i', W1, W2))
aug = lambda z: np.append(field(z[:2]), trJ(z[:2]))  # d/dt (x, int tr J)
xa, n = np.random.randn(2), 200
zT = rk4(aug, np.append(xa, 0.0), 1.0, n)
D = np.zeros((2, 2))                           # D Phi_1 by central differences
for j in range(2):
```

```
    e = np.zeros(2); e[j] = 1e-5
    D[:, j] = (rk4(field, xa + e, 1.0, n) - rk4(field, xa - e, 1.0, n)) / 2e-5
logdet = np.log(abs(np.linalg.det(D)))
print(f'tr J at x(0) {trJ(xa):+.3f}, at x(1) {trJ(zT[:2]):+.3f}')
print(f'trace integral {zT[2]:+.6f}  log|det D Phi| {logdet:+.6f}  '
      f'gap {abs(zT[2] - logdet):.1e}')
```

```
tr J at x(0) +0.640, at x(1) +0.378
trace integral +0.513576  log|det D Phi| +0.513576  gap 6.6e-11
```

The trace generally takes different values at the two ends of the path, yet the integral and the log-determinant agree to about 10^{-10}, the accuracy of the solver and of the finite differences. Nothing in the proposition used linearity.

38.2.3.3 The Hutchinson Trace Estimator

The formula for $\operatorname{tr}\mathbf{J}$ above used the structure of a one-layer field. For a general network the trace needs the d diagonal entries of $\mathbf{J}$, which cost d reverse-mode passes, one per coordinate. The trace has already replaced an $O(d^3)$ determinant by d passes. A random projection replaces the d passes by one, and returns an unbiased but noisy estimate (Hutchinson 1989).

Proposition (Hutchinson estimator). *For any matrix $\mathbf{M} \in \mathbb{R}^{d\times d}$ and any random vector $\epsilon \in \mathbb{R}^d$ with $\mathbb{E}[\epsilon\epsilon^\top] = \mathbf{I}$, such as a standard Gaussian or a Rademacher vector with independent entries ± 1 of equal probability,*

$$\operatorname{tr}(\mathbf{M}) = \mathbb{E}\left[\epsilon^\top \mathbf{M}\epsilon\right]. \tag{38.21}$$

Proof. $\mathbb{E}[\epsilon^\top\mathbf{M}\epsilon] = \mathbb{E}[\operatorname{tr}(\mathbf{M}\epsilon\epsilon^\top)] = \operatorname{tr}(\mathbf{M}\,\mathbb{E}[\epsilon\epsilon^\top]) = \operatorname{tr}(\mathbf{M})$, by the cyclic property of the trace and its linearity, which lets it commute with the expectation. ■

The estimator touches $\mathbf{J}$ only through $\epsilon^\top\mathbf{J}$, one vector–Jacobian product, which is one reverse-mode pass through $\mathbf{f}$ with ϵ as the seed; no Jacobian is formed. For the one-hidden-layer field the product is explicit,

$$\epsilon^\top\mathbf{J} = \left((\epsilon^\top\mathbf{W}_2)\odot\mathbf{s}\right)\mathbf{W}_1,$$

and it costs about as much as evaluating $\mathbf{f}$. A CNF evaluates one such product with every evaluation of the field, so the likelihood costs one additional backward pass per field evaluation (FFJORD, (Grathwohl et al. 2019)).

The proposition says that the average over probes converges to the trace. It does not say how fast, and that is what we measure. We take a one-hidden-layer field in $d = 50$ with 64 hidden units, compute its exact trace from the formula above, and draw $m = 10^5$ Rademacher probes.

```
d, H, m = 50, 64, 100_000
W1, b1 = np.random.randn(H, d) / np.sqrt(d), 0.1 * np.random.randn(H)
W2 = np.random.randn(d, H) / np.sqrt(H)
x = np.random.randn(d)
```

```
s = 1 - np.tanh(W1 @ x + b1) ** 2
tr_exact = np.sum(s * np.einsum('ij,ji->i', W1, W2))
E = np.random.choice([-1.0, 1.0], size=(m, d))  # Rademacher probes, one per row
print(f'exact trace {tr_exact:.4f}')
```

```
exact trace -1.1227
```

```
W1t, b1t, W2t = (torch.tensor(w) for w in (W1, b1, W2))
field = lambda x: W2t @ torch.tanh(W1t @ x + b1t)
_, vjp = torch.func.vjp(field, torch.tensor(x))  # reverse mode at x
rows = torch.func.vmap(vjp)(torch.tensor(E))[0]  # row k: eps_k^T J
est = (rows * torch.tensor(E)).sum(dim=1)        # eps_k^T J eps_k
print(f'Hutchinson {est.mean():.4f} +- {est.std() / m ** 0.5:.4f} '
      f'({m} probes, one VJP each)')
```

```
Hutchinson -1.1443 +- 0.0142 (100000 probes, one VJP each)
```

The average lies within a few standard errors of the exact trace. But read the output again: 10^5 probes locate the trace only to about 0.015, when 50 passes would have given it exactly. The standard error implies a standard deviation of about 4 to 5 for one probe, many times the trace, which is typically a few tenths in size. For Rademacher probes one probe has variance $2\sum_{i\neq j} S_{ij}^2$ with $\mathbf{S} = (\mathbf{J} + \mathbf{J}^\top)/2$: the off-diagonal entries set the noise, and the trace sees only the diagonal. Our random field is a hard case, because its diagonal entries have random signs and nearly cancel. Subtracting $\mathbf{x}$ from the field, a pull toward the origin, would move the trace to about -50 and leave the noise unchanged, a relative error of a tenth per probe.

Why, then, does a CNF train with one probe per step? The estimate is unbiased and enters the log-likelihood linearly, so the estimated log-likelihood and its gradient are unbiased as well. A stochastic gradient asks for nothing more (Section 35.4): the noise of the probes is averaged over minibatches and iterations, as the noise of the minibatch itself is. What one probe cannot give is the likelihood of a single point. For that, average many probes or compute the exact trace.

The dynamics of this section were deterministic throughout. A diffusion model adds fresh noise at every step, and increments that scale like $\sqrt{h}$ break the Taylor expansions this section relied on. Section 38.3 asks how to integrate such an equation.

38.2.4 Summary

A residual block is an Euler step, and one field with shrinking steps gives a Neural ODE. Its exact flow is invertible and, on $\mathbb{R}$, order-preserving, so the map $x \mapsto -x$ needs a second dimension. A field trained through ten Euler steps fits the ten-step program, not the flow. The gradient of a loss on $\mathbf{x}(T)$ is the integral of $(\partial \mathbf{f}/\partial\theta)^\top \mathbf{a}$ along the trajectory, where the adjoint $\mathbf{a}$ solves a linear ODE backward from $\nabla \ell(\mathbf{x}(T))$. Backpropagation through Euler is exact for the program and within $O(h)$ of the gradient of the flow. Re-integrating the state backward saves memory but amplifies a stored error by $e^{\lambda T}$ for a mode that decays at rate λ. Along a trajectory $\frac{d}{dt}\log p_t = -\operatorname{tr}\mathbf{J}$. Hutchinson's estimator obtains the trace from one vector–Jacobian product per probe, without bias but with noise that training averages out and a single likelihood does not.

38.2.5 Exercises

1. code **The learned field at other step counts.** The `#odes-solvers-neural-ode-train` cell trains the field with $N = 10$ Euler steps of size $h = 0.1$, and the text found that the endpoint error rises from a few thousandths to about 0.02 when the same field is integrated with $N = 100$ or $N = 1000$ steps.
    1. Evaluate the trained `flow` with $N = 1$, 2 and 5 steps ($Nh = 1$). Why is the error at $N = 1$ large although ten steps of the same field are accurate to a few thousandths? Which map does one Euler step of the trained field compute?
    2. What does the error at $N = 1000$ measure? Name the two maps whose distance it is, and explain why it does not shrink as N grows further.
    3. Integrate the trained field backward from the targets `Y` with $N = 10$, 100 and 1000 steps of $\mathbf{x} \leftarrow \mathbf{x} - h\,\mathbf{f}_\theta(\mathbf{x})$ and compare the result with `X`. The forward–backward round trip of the text went to zero like $1/N$; this error settles at a few hundredths. Why? *Hint: at large N the backward integration applies the exact inverse flow to the ellipse, not to the ten-step image of the circle.*
    4. Retrain with $N = 1$ and with $N = 2$, and repeat (a). Which of the three trained fields comes closest to its own time-one flow?

2. code **What a flow cannot represent.** ▣ Two trajectories of a field with Lipschitz constant L satisfy $e^{-Lt}\|\mathbf{x}-\mathbf{y}\| \le \|\Phi_t(\mathbf{x}) - \Phi_t(\mathbf{y})\| \le e^{Lt}\|\mathbf{x}-\mathbf{y}\|$ (1), and Section 38.2.1.2 showed that no flow on $\mathbb{R}$ reverses order (Dupont et al. 2019).
    1. Let $d = 1$. Use the lower bound to give a second proof that Φ_T is strictly increasing, and show that $\Phi_T(1) - \Phi_T(-1) \ge 2e^{-LT}$. Which of the two statements, order or distance, does the proposition of the text give, and which needs the bound?
    2. The training example carries the circle points $(0, \pm 1)$, which are 2 apart, to $(1.5, \pm 0.4)$, which are 0.8 apart. Show that the exact time-T flow of a field with Lipschitz constant L can do this only if $LT \ge \ln 2.5 \approx 0.92$, whereas the long axis, stretched from 2 to 2.4, gives only $LT \ge \ln 1.2$. A one-hidden-layer tanh field has $L \le \|\mathbf{W}_2\|_2\|\mathbf{W}_1\|_2$ because tanh is 1-Lipschitz. Compute this bound for the trained field and check the inequality.
    3. Embed $\mathbb{R}$ in $\mathbb{R}^2$ as $x \mapsto (x, 0)$. Show that every linear field $\dot{\mathbf{x}} = \mathbf{B}\mathbf{x}$ whose time-one map sends $(x, 0)$ to $(-x, 0)$ for every x has $e^{\mathbf{B}} = -\mathbf{I}$. *Hint: an eigenvalue -1 of $e^{\mathbf{B}}$ comes from an eigenvalue μ of $\mathbf{B}$ with $e^{\mu} = -1$, which is not real.* Find such a field whose trajectories are ellipses rather than circles.
    4. Change the training cell to fit the map $x \mapsto -x$ on 64 evenly spaced points of $[-1, 1]$ with a field of input and output dimension 1 and $N = 10$ steps. What does the trained model output, and why does the loss settle at a value you can predict before training? Repeat with a field of dimension 2, inputs $(x, 0)$, and the loss on the first coordinate.
    5. Train the one-dimensional model once more with $N = 1$ and $h = 1$, and evaluate it with $N = 1$ and with $N = 100$ steps of size $h = 0.01$. Relate the two results to the proposition and to the condition $hL < 1$.

3. **Adjoint with a running loss.** ▣ The adjoint equations Equation 38.14 and Equation 38.15 treat a loss on the endpoint. Regularizers such as the kinetic energy $\int_0^T \|\mathbf{f}(\mathbf{x}(t), \theta, t)\|^2\, dt$ of Finlay et al. (2020) accumulate a cost along the trajectory. Let

$$\mathcal{L} = \ell_T(\mathbf{x}(T)) + \int_0^T \ell(\mathbf{x}(t), \theta, t)\, dt.$$

1. Extend the variational proof of Equation 38.14 to show that the adjoint acquires a source term, $\dot{\mathbf{a}} = -(\partial \mathbf{f}/\partial \mathbf{x})^\top \mathbf{a} - \nabla_{\mathbf{x}} \ell$ with $\mathbf{a}(T) = \nabla \ell_T(\mathbf{x}(T))$, and that $\partial \mathcal{L}/\partial \theta = \int_0^T \left((\partial \mathbf{f}/\partial \theta)^\top \mathbf{a} + \nabla_\theta \ell\right) dt$.
2. Finlay et al. (2020) instead append a coordinate E with $\dot{E} = \ell$ and $E(0) = 0$ to the state, and add $E(T)$ to the endpoint loss. Apply Equation 38.14 to the augmented state and show that it reproduces (a).
3. For the kinetic energy, $\ell = \|\mathbf{f}\|^2$. Write $\nabla_{\mathbf{x}} \ell$ as a vector–Jacobian product. How many reverse-mode passes through $\mathbf{f}$ does one step of the adjoint ODE then require?

4. code **Reconstructing the trajectory backward.** ⊠ The continuous adjoint can avoid storing the forward states by integrating $\mathbf{x}(t)$ backward from $\mathbf{x}(T)$ alongside $\mathbf{a}(t)$, and Section 38.2.2 quoted the resulting drift $e^{\lambda T}\delta$.
   1. For $\dot{x} = -\lambda x$ with $\lambda > 0$, an error δ in the stored $x(T)$ causes an error in the reconstructed $x(0)$. Derive it.
   2. Let $\mathbf{R}$ be the rotation by 0.6 radians and $\mathbf{B} = \mathbf{R}\,\mathrm{diag}(-50, -1)\,\mathbf{R}^\top$. Integrate $\dot{\mathbf{x}} = \mathbf{B}\mathbf{x}$ from $(1, 1)$ to time T with `rk4` and 2000 steps, then integrate the field $-\mathbf{B}\mathbf{x}$ from the result for the same time. Report the reconstruction error for $T = 0.25,\ 0.5,\ 1$ and 2, and compare its growth with (a), taking δ to be the double-precision rounding error $\approx 10^{-16}$. Repeat with $\mathrm{diag}(-50, -1)$ itself. Why does the diagonal matrix reconstruct far more accurately?
   3. Activation checkpointing (Section 5.7.3.3) stores the state at the start of each segment of the trajectory. A hybrid scheme reconstructs the states within a segment by integrating backward from the checkpoint at its end. For the matrix $\mathbf{B}$ of (b), what is the longest segment duration for which the error of (a) stays below 10^{-8}?

5. **Log-determinant of a residual step.** ⊠ The instantaneous change of variables Equation 38.18 is a limit of infinitesimal steps. A finite residual step $F(\mathbf{x}) = \mathbf{x} + \mathbf{g}(\mathbf{x})$ needs its exact log-determinant, as in the invertible residual networks of Behrmann et al. (2019) .
   1. Let $\mathbf{g}$ be Lipschitz with constant $c < 1$. For $d = 1$ the one-line argument of Section 38.2.1.2 (an Euler step is increasing when $hL < 1$) shows that F is strictly increasing, hence injective. For general d, write $F(\mathbf{x}) = \mathbf{y}$ as a fixed-point equation for $\mathbf{x}$ and use the Banach fixed-point theorem, as in the proof of Picard–Lindelöf, to show that F is a bijection of $\mathbb{R}^d$. How fast does the fixed-point iteration converge?
   2. Show that $\|\mathbf{J}_{\mathbf{g}}\| \le c$, where $\mathbf{J}_{\mathbf{g}}$ is the Jacobian of $\mathbf{g}$, and use the eigenvalues of $\mathbf{J}_{\mathbf{g}}$ to show that
$$\log\det(\mathbf{I} + \mathbf{J}_{\mathbf{g}}) = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \operatorname{tr}\left(\mathbf{J}_{\mathbf{g}}^k\right).$$
   3. An Euler step has $\mathbf{g} = h\mathbf{f}$ and $\mathbf{J}_{\mathbf{g}} = h\mathbf{J}$. Compare $\log\det(\mathbf{I} + h\mathbf{J})$ with the term $h \operatorname{tr}\mathbf{J}$ that Equation 38.19 accumulates per step. How large is the difference per step, and how large is it in total over T/h steps?

4. The Hutchinson estimator Equation 38.21 estimates $\mathrm{tr}(\mathbf{J}_{\mathbf{g}}^{k})$ by $\epsilon^{\top}\mathbf{J}_{\mathbf{g}}^{k}\epsilon$. How many vector–Jacobian products per probe does the series truncated after n terms cost?

6. code **Variance of the Hutchinson estimator.** ▯ The estimator Equation 38.21 estimates $\mathrm{tr}\,\mathbf{M}$ by $\epsilon^{\top}\mathbf{M}\epsilon$ for a single probe ϵ. Let $\mathbf{S} = (\mathbf{M} + \mathbf{M}^{\top})/2$, so that $\epsilon^{\top}\mathbf{M}\epsilon = \epsilon^{\top}\mathbf{S}\epsilon$.
   1. Show that the variance of $\epsilon^{\top}\mathbf{M}\epsilon$ is $2\|\mathbf{S}\|_F^2$ for Gaussian probes and $2\sum_{i\neq j} S_{ij}^2$ for Rademacher probes. For which matrices is a single Rademacher probe exact?
   2. Form the Jacobian $\mathbf{J} = \mathbf{W}_2\,\mathrm{diag}(\mathbf{s})\,\mathbf{W}_1$ of the field in the `#mdl-neural-odes-the-hutchinson-trace-estimator-1` cell explicitly (it is 50×50), evaluate both variances, and compare the Rademacher value with m times the square of the standard error that the next cell printed. How many probes does each distribution need for a standard error of 0.01?
   3. In a CNF, $\mathbf{M}$ is the Jacobian $\partial\mathbf{f}/\partial\mathbf{x}$ in dimension d. Suppose that all entries have comparable magnitude. How does the relative standard error $\sqrt{\mathrm{Var}}/|\,\mathrm{tr}\,\mathbf{M}|$ of one probe scale with d when the diagonal entries share one sign, and when their signs are random?

7. code **Discrete and continuous adjoints.** ▯▯ Backpropagation through a solver differentiates the executed program, whereas a solver applied to the adjoint ODE Equation 38.14 approximates the gradient of the exact flow (Kidger 2022). Compare the two recursions step by step for an autonomous field with Jacobian $\mathbf{J}(\mathbf{x}) = \partial\mathbf{f}/\partial\mathbf{x}$ and forward states $\mathbf{x}_k$.
   1. For forward Euler, the discrete adjoint Equation 38.16 computes $\mathbf{a}_k = (\mathbf{I} + h\mathbf{J}(\mathbf{x}_k))^{\top}\mathbf{a}_{k+1}$, whereas an explicit Euler step of the adjoint ODE from t_{k+1} back to t_k computes $\mathbf{a}_k = \mathbf{a}_{k+1} + h\mathbf{J}(\mathbf{x}_{k+1})^{\top}\mathbf{a}_{k+1}$. Bound the difference of the two updates for a smooth $\mathbf{f}$. For a linear field the two coincide; what then produces the gap that the `#mdl-neural-odes-backpropagation-as-the-discrete-adjoint-1` cell prints? For the scalar program $x_N = (1 + h\theta)^N x_0$ the text found the gradient $T\,x_N^2/(1 + h\theta)$; show that this is what the recursion computes.
   2. Differentiate one Heun step $\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{h}{2}\big[\mathbf{f}(\mathbf{x}_k) + \mathbf{f}(\mathbf{u}_k)\big]$ with $\mathbf{u}_k = \mathbf{x}_k + h\mathbf{f}(\mathbf{x}_k)$ to obtain its discrete adjoint recursion. Show that this recursion is a Heun step of the adjoint ODE from t_{k+1} to t_k in which $\mathbf{J}(\mathbf{x}_{k+1})$ is replaced by $\mathbf{J}(\mathbf{u}_k)$. At which order in h do the two updates differ?
   3. For a linear field $\mathbf{f}(\mathbf{x}) = \mathbf{A}\mathbf{x}$, one RK4 step multiplies the state by $R(h\mathbf{A})$, where $R(z) = 1 + z + z^2/2 + z^3/6 + z^4/24$ is the RK4 stability polynomial (2). Show that the discrete adjoint of RK4 is an RK4 step of the adjoint ODE.
   4. Replace the Euler step of `loss_and_grad_unrolled` by an RK4 step, and compute the gradient of the unrolled loss by central finite differences instead of a hand-written backward sweep. Measure its gap to the continuous-adjoint gradient `g_cont` for $n = 5$, 10, 20, 40 and 80 steps. The trapezoid rule that computes `g_cont` uses `n_quad = 2000` intervals and is accurate only to about 10^{-7}, so refine it first. Which slope do you measure? Which order does the gradient computed through a solver of order p inherit?

8. **Checkpointing arithmetic.** ▯ A forward pass of N Euler steps is differentiated by the discrete adjoint Equation 38.16, whose backward sweep needs $\mathbf{x}_k$ at step k.
   1. Storing every state holds N states and recomputes nothing. Suppose instead that only every k-th state is stored as a checkpoint, and that when the backward sweep enters a segment of k steps, the segment is recomputed forward from its check-

point and its states are held until the sweep leaves it. How many states are held at the peak, and how many extra forward steps are computed in total?

2. Minimize the peak over k. Show that the minimum is near $k = \sqrt{N}$, and that the extra work is about one forward pass whatever k is. For $N = 1000$, compare the peak with the 1000 states of full storage.
3. Recursive checkpointing applies the same idea inside each segment and holds $O(\log N)$ states at the cost of $O(\log N)$ forward passes (Griewank 1992). Compare it with the continuous adjoint, whose memory is constant but whose backward re-integration amplifies an error by $e^{\lambda T}$ on $\dot{x} = -\lambda x$. For which λT does re-integration from a state stored in single precision ($\delta \approx 6 \times 10^{-8}$) lose all accuracy?

9. code **Liouville's formula.** ⊠⊠ The instantaneous change of variables followed a density along a trajectory. This exercise proves it a second way, through the Jacobian $\mathbf{M}(t) = D\Phi_t(\mathbf{x}_0)$ of the flow map itself.

   1. Differentiate $\frac{d}{dt}\Phi_t(\mathbf{x}_0) = \mathbf{f}(\Phi_t(\mathbf{x}_0), t)$ with respect to $\mathbf{x}_0$ to obtain the variational equation $\dot{\mathbf{M}} = \mathbf{J}(\mathbf{x}(t), t)\,\mathbf{M}$ with $\mathbf{M}(0) = \mathbf{I}$.
   2. Prove Jacobi's formula $\frac{d}{dt}\det\mathbf{M} = \operatorname{tr}(\mathbf{M}^{-1}\dot{\mathbf{M}})\det\mathbf{M}$ for invertible $\mathbf{M}(t)$. *Hint: write* $\det\mathbf{M}(t+h) = \det\mathbf{M}(t)\,\det\left(\mathbf{I} + h\mathbf{M}^{-1}\dot{\mathbf{M}} + O(h^2)\right)$ *and use the Lemma.*
   3. Conclude that $\log|\det D\Phi_T(\mathbf{x}_0)| = \int_0^T \operatorname{tr}\mathbf{J}(\mathbf{x}(t), t)\,dt$, and derive Equation 38.19 from the discrete change of variables Equation 38.17 applied to the bijection Φ_T. For a linear field the result is the formula Equation 38.20.
   4. Integrate $\dot{\mathbf{M}} = \mathbf{J}\mathbf{M}$ with RK4 alongside the 8-unit tanh field of the `#mdl-neural-odes-checking-the-trace-integral-1` cell (re-create its weights, since the Hutchinson cells reuse the names) and compare $\log|\det\mathbf{M}(1)|$ with the two numbers that cell prints.

38.3 Stochastic Differential Equations

The two previous sections integrated deterministic vector fields. From a given start the trajectory is fixed, and a flow never forgets where it came from: two different starting points always reach two different endpoints. A continuous normalizing flow can still carry a data distribution to a Gaussian, but only after it has been trained on that particular data. The diffusion models of the main text work the other way round. Their forward process adds fresh noise at every step until nothing of the starting point remains, and this is what makes generation possible, because the reverse process then has a known place to begin. What equation does such a process solve, and how do we calculate with it when the noise is new at every instant?

A small example shows what is involved. Take one number, $x_0 = 2$. At each step keep 99% of it and add an independent Gaussian draw with standard deviation $\sqrt{0.02} \approx 0.14$:

$$x_{k+1} = 0.99\,x_k + \sqrt{0.02}\,\xi_k, \qquad \xi_k \sim \mathcal{N}(0,1).$$

After 400 steps the mean is $2 \cdot 0.99^{400} \approx 0.036$ and the variance is close to 1. Any other start ends in the same place, as a draw from a distribution very close to $\mathcal{N}(0,1)$. One feature of the recurrence looks odd. If it describes a process in continuous time with time step $h = 0.01$, then the decay $0.99 = 1 - h$ changes x by an amount of order h, but the noise

has size $\sqrt{h}$, not h. This choice is forced. Noise of size h would vanish as $h \to 0$, while noise of size $\sqrt{h}$ survives, and its square is of order h, the same order as the drift. Ordinary calculus discards squares of small changes, so this one fact requires a new calculus.

The continuous-time limit of such noise is Brownian motion, named after the botanist Robert Brown, who watched pollen grains jitter in water in 1827. Norbert Wiener constructed it as a mathematical object in 1923, and Kiyoshi Itô built the calculus for equations driven by it in the 1940s (Itô 1944). The central idea of Itô's calculus is that along a Brownian path the square of a short increment equals the elapsed time on average, so the chain rule gains a second-derivative term.

The forward process of a diffusion model is such an equation, and Section 17.2 is where the book uses it: the variance-preserving path there is the process behind our recurrence. Its law at each time, given the start, is Gaussian, which gives denoising score matching its regression targets in closed form. We take the limit of random walks, build the calculus, solve the process in closed form, and then simulate general equations and measure the error.

Three results carry the section, and the rest is preparation or consequence: the quadratic variation $(dW)^2 = dt$, Itô's lemma, and the Gaussian kernel of the Ornstein–Uhlenbeck process. Only the first is hard to believe, and its proof takes six lines. A reader in a hurry can take the two properties of the Itô integral (Section 38.3.2.2) on trust and leave the orders of convergence (Section 38.3.4.2) for a second reading. Throughout, f is the drift, g the noise amplitude, W Brownian motion, θ the decay rate of the running example, and α_t, σ_t the mean factor and the standard deviation of its Gaussian marginals.

38.3.1 Brownian Motion

38.3.1.1 Noise as a Forward Process

Before building any machinery we check what a forward process needs, and why a deterministic flow cannot provide it. Starting from data distributed as $p_0 = p_{\text{data}}$, the process must end at one known distribution, whatever the data looked like,

$$p_0 = p_{\text{data}} \;\longrightarrow\; p_T \approx \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

so that at generation time we always know where to start. A deterministic flow cannot do this for every dataset at once. The flows of Section 38.1 are invertible maps Φ, and an invertible map sends distinct laws to distinct laws: if Φ carried p and q to the same distribution, applying Φ^{-1} would show $p = q$. So no single fixed flow carries two different datasets to the same endpoint.

Let us try three processes on two "datasets" that are as simple as datasets can be: point masses at $x_0 = 2$ and at $x_0 = -2$.

- **Drift alone**, $\dot{x} = -x$. The solution is $x(t) = x_0 e^{-t}$, so at $t = 3$ the two starts sit at $\pm 2e^{-3} = \pm 0.0996$. Both datasets shrink toward the point mass at 0, which is not a Gaussian, and at every finite time they are still two distinct points.
- **Noise alone**, $dX = \sqrt{2}\, dW$. Here W is a noise whose variance grows linearly in time, constructed in the next subsection, and the equation means $X_t = x_0 + \sqrt{2}\, W_t$. The law at time t is $\mathcal{N}(x_0, 2t)$: each cloud keeps its center at ± 2 while its standard

deviation $\sqrt{2t}$ grows without bound (2.45 at $t = 3$). The start is forgotten only relative to this growing width, and there is no limiting law.

- **Both together**, $dX = -X\,dt + \sqrt{2}\,dW$. From either start the law at $t = 3$ is $\mathcal{N}(\pm 0.0996,\ 0.9975)$, a formula proved later in this section. The two laws are nearly indistinguishable from each other and from $\mathcal{N}(0, 1)$.

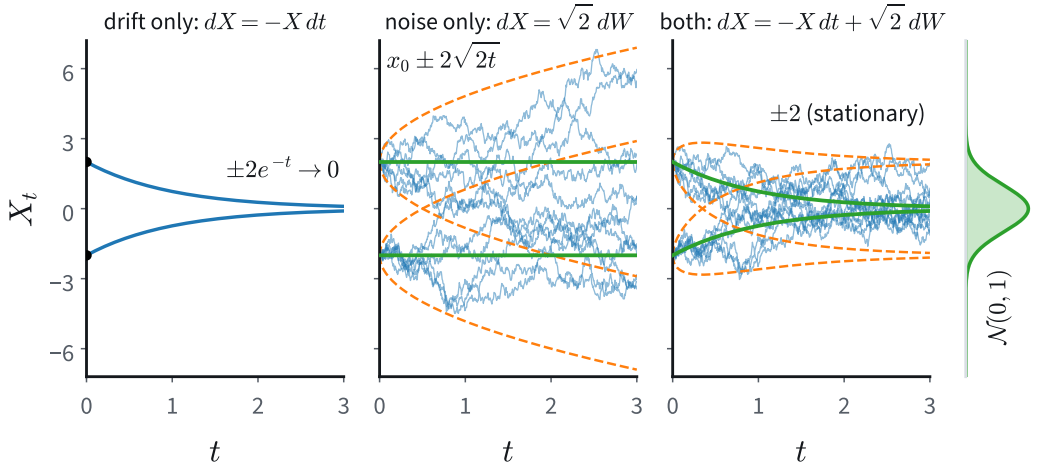


Figure 38.10: Two failures and the remedy, from the two starts $x_0 = \pm 2$. Drift alone ($\dot{x} = -x$) sends every start to the point 0. Noise alone ($X_t = x_0 + \sqrt{2}\,W_t$) keeps each start as the center of a cloud whose two-standard-deviation envelope $x_0 \pm 2\sqrt{2t}$ grows without bound. Both together ($dX = -X\,dt + \sqrt{2}\,dW$) carry every start into the same stationary law $\mathcal{N}(0, 1)$, drawn sideways at the right. Green lines are the means; dashed lines are the analytic two-standard-deviation envelopes.

Figure 38.10 draws the three processes. Only the first fails outright. The second is usable, and is in use, if it runs until its variance $2t$ is far larger than that of the data; this family is called *variance-exploding*, and its endpoint variance must be made huge because it has no stationary law. The third is the running example of this section, and the recurrence of the opening simulates it. Its drift pulls each start toward zero while the noise refills the variance that the drift removes, and the two balance at variance 1. Section 38.3.3 proves that this happens from every start.

In general, a forward process of this kind is a deterministic vector field plus a noise term. For data in $\mathbb{R}^d$ we write

$$d\mathbf{X} \;=\; \underbrace{\mathbf{f}(\mathbf{X}, t)\,dt}_{\text{drift}} \;+\; \underbrace{g(t)\,d\mathbf{W}}_{\text{diffusion}}, \tag{38.22}$$

which reads: over a short time dt, the state moves by the velocity $\mathbf{f}$ times dt and receives a random kick whose size is set by the noise amplitude g. The drift steers the mean, and the diffusion injects variance. Such a process can also be run backward in distribution, which is how a diffusion model generates; the next section shows how.

38.3.1.2 From Random Walks to the Wiener Process

The noise term $d\mathbf{W}$ needs a precise object behind it. We build it as the limit of the simplest random process, a coin-flip walk, and the limit explains the square root in our recurrence.

Fix a step duration Δt and let a walker take independent steps of size $\pm\sqrt{\Delta t}$, each sign chosen by a fair coin. After k steps, at time $t = k\,\Delta t$, the position is a sum of k independent mean-zero steps of variance Δt apiece. Variances of independent terms add (Section 36.2), so

$$\mathbb{E}\big[W_t^{(\Delta t)}\big] = 0, \qquad \mathrm{Var}\big(W_t^{(\Delta t)}\big) = k\,\Delta t = t,$$

whatever the step size. With $\Delta t = 0.01$, for instance, 100 steps of ± 0.1 give variance $100 \cdot 0.01 = 1$ at time 1. Steps of ± 0.01 would give variance 0.01, and steps of ± 1 would give 100.

More generally, steps of size $c\,\Delta t^a$ give variance $(t/\Delta t)\,c^2\Delta t^{2a} = c^2\,t\,\Delta t^{2a-1}$ at time t, which tends to 0 for $a > \frac{1}{2}$ and to ∞ for $a < \frac{1}{2}$. Among such step sizes only $a = \frac{1}{2}$ gives a finite nonzero variance in the limit, and this square root returns in the noise term of every numerical scheme below. As $\Delta t \to 0$ the number of steps before time t grows without bound, and the central limit theorem (a sum of many independent mean-zero terms with total variance t is approximately $\mathcal{N}(0, t)$) makes the limiting position Gaussian.

The limit process is called **Brownian motion** or the **Wiener process** (Wiener 1923). It is defined by four properties:

1. $W_0 = 0$;
2. **independent increments**: for $s < t$, the increment $W_t - W_s$ is independent of the entire path up to time s;
3. **Gaussian increments**: $W_t - W_s \sim \mathcal{N}(0,\, t - s)$;
4. continuous sample paths.

That a process with these properties exists is Wiener's theorem, and that the rescaled walks converge to it is Donsker's theorem; we take both as given. In particular $\mathbb{E}[W_t] = 0$ and $\mathrm{Var}(W_t) = t$: the process spreads with standard deviation $\sqrt{t}$ forever (Figure 38.11). Simulation uses the increment form

$$\Delta W \;=\; W_{t+\Delta t} - W_t \;=\; \sqrt{\Delta t}\,\xi, \qquad \xi \sim \mathcal{N}(0,1), \tag{38.23}$$

which says that a Brownian increment over a step of length Δt is a standard normal draw scaled by $\sqrt{\Delta t}$.

The multivariate Brownian motion $\mathbf{W}_t$ runs one independent scalar Brownian motion per coordinate. For the diagonal, state-independent noise $g(t)\,d\mathbf{W}$ of the diffusion-model SDEs in this chapter, every calculation below therefore applies coordinate by coordinate.

Independent increments determine how the values of W at two times are related. For $s \le t$ split $W_t = W_s + (W_t - W_s)$; the increment is independent of W_s and both have mean zero, so

$$\mathrm{Cov}(W_s, W_t) = \mathbb{E}\big[W_s^2\big] + \mathbb{E}[W_s]\,\mathbb{E}[W_t - W_s] = s = \min(s, t).$$

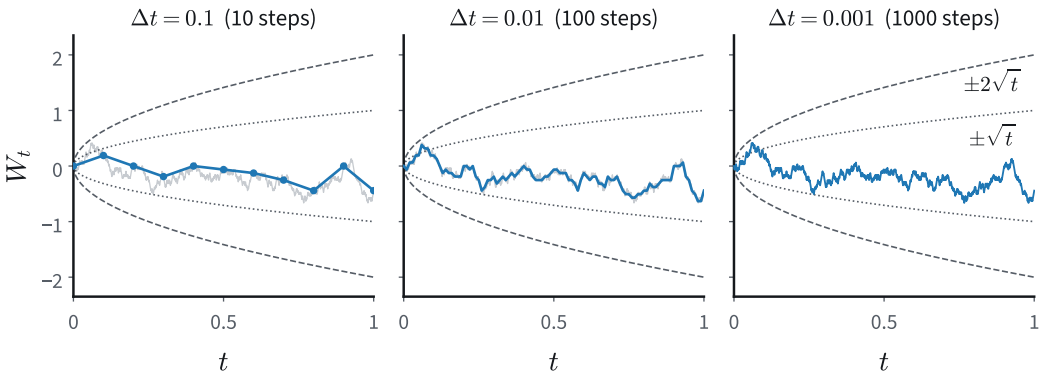


Figure 38.11: One coin-flip walk of 1000 steps $\pm\sqrt{0.001}$, viewed on meshes $\Delta t = 0.1, 0.01$ and 0.001 (the same randomness; faint gray: the full path). Each view is itself a random walk with step variance Δt; refining the mesh fills in detail without changing the large-scale path. At each fixed time the spread follows the $\pm\sqrt{t}$ and $\pm 2\sqrt{t}$ envelopes.

The walk carries its position at time s forward unchanged and adds fresh noise that is uncorrelated with it. The formula is worth more than its one-line proof suggests: a Gaussian process is determined by its mean and covariance, so mean zero, covariance $\min(s, t)$ and continuous paths characterize Brownian motion.

The same increments make the path rough. Brownian paths are continuous but nowhere differentiable, a theorem of Paley, Wiener, and Zygmund (Paley et al. 1933). The increment scaling Equation 38.23 gives the heuristic reason:

$$\left|\frac{\Delta W}{\Delta t}\right| = \frac{|\xi|}{\sqrt{\Delta t}} \longrightarrow \infty \quad \text{in probability as } \Delta t \to 0,$$

so a difference quotient grows without bound as the step shrinks. This is not a proof (it rules out a derivative at each fixed time, while "nowhere" concerns all times at once), but it shows the obstruction.

Rescaling time by a factor c and space by $\sqrt{c}$ returns a Brownian motion (problem 12), so zooming in on a path shows the same roughness at every scale, where a differentiable curve would flatten into its tangent. Hence dW/dt does not exist, and dW has to be given a meaning inside integrals. The Itô calculus supplies that meaning.

38.3.1.3 Simulation of Brownian Paths

The increment form Equation 38.23 is also a simulation recipe: take cumulative sums of $\sqrt{\Delta t}\,\xi$ steps. We generate 20,000 paths on $[0, 1]$ this way, and with them the random generator `rng` that every later cell draws from.

```
rng = np.random.default_rng()
T_bm, n_bm, n_paths = 1.0, 500, 20000
dt_bm = T_bm / n_bm
dW_bm = np.sqrt(dt_bm) * rng.standard_normal((n_paths, n_bm))
```

```
W_bm = np.concatenate([np.zeros((n_paths, 1)), np.cumsum(dW_bm, axis=1)],
                      axis=1)
for t in [0.25, 0.5, 1.0]:
    k = round(t / dt_bm)
    print(f'Var(W_t) at t={t:.2f}: {W_bm[:, k].var():.4f}   (theory: {t:.2f})')
inside = np.abs(W_bm[:, -1]) < 2 * np.sqrt(T_bm)
print(f'fraction inside +-2 sqrt(t) at t=1: {inside.mean():.3f}'
      '   (Gaussian: 0.954)')
```

```
Var(W_t) at t=0.25: 0.2472   (theory: 0.25)
Var(W_t) at t=0.50: 0.4983   (theory: 0.50)
Var(W_t) at t=1.00: 1.0045   (theory: 1.00)
fraction inside +-2 sqrt(t) at t=1: 0.956   (Gaussian: 0.954)
```

The variances equal t up to the Monte Carlo standard error $t\sqrt{2/20000}$, or 1% of t, as they must for sums of such steps: the printout tests the code, not the theory. About 95% of the paths lie inside $\pm 2\sqrt{t}$ at $t = 1$. This is a statement about one fixed time. Far fewer paths, about half on this grid, stay inside the envelope over the whole interval. The same ensemble gives the covariance, as the average of $W_s W_t$ over a grid of time pairs.

```
ts = np.array([0.1, 0.25, 0.5, 0.75, 1.0])
idx = np.round(ts / dt_bm).astype(int)
emp = W_bm[:, idx].T @ W_bm[:, idx] / n_paths   # E[W_s W_t]; means are zero
print('empirical E[W_s W_t]:')
print(emp.round(3))
print('max |empirical - min(s,t)| = '
      f'{np.abs(emp - np.minimum.outer(ts, ts)).max():.4f}')
```

```
empirical E[W_s W_t]:
[[0.099 0.098 0.098 0.097 0.098]
 [0.098 0.247 0.248 0.247 0.249]
 [0.098 0.248 0.498 0.495 0.5  ]
 [0.097 0.247 0.495 0.738 0.745]
 [0.098 0.249 0.5   0.745 1.005]]
max |empirical - min(s,t)| = 0.0117
```

Each row is constant to the right of the diagonal, up to a sampling error of about 0.01: what happens after time s adds nothing to the covariance with W_s.

38.3.2 Itô Calculus

38.3.2.1 Quadratic Variation

Ordinary calculus stops its Taylor expansions at first order because the square of a small change is negligible. For Brownian motion this fails, and the result of this subsection says by how much. It is the one on which the rest of the section rests.

Take a partition $0 = t_0 < t_1 < \cdots < t_n = t$ with increments $\Delta t_i = t_{i+1} - t_i$ and mesh $h_{\max} = \max_i \Delta t_i$, and add up the squared increments of a path along it. For a differentiable path x with $|x'| \le M$, the mean value theorem gives $|\Delta x_i| \le M\, \Delta t_i$, so

$$\sum_i (\Delta x_i)^2 \;\le\; M^2 \sum_i \Delta t_i^2 \;\le\; M^2\, h_{\max}\, t \;\longrightarrow\; 0$$

as the mesh shrinks. This is why $(dx)^2$ is negligible against dx in ordinary calculus: a smooth increment squared is of order Δt^2. A Brownian increment squared is of order Δt, since $\mathbb{E}[(\Delta W_i)^2] = \Delta t_i$, and these add up to t no matter how fine the mesh (Figure 38.12). The sum is random, but its fluctuations vanish in the limit.

Proposition (quadratic variation of Brownian motion). *As the mesh $h_{\max} \to 0$,*

$$Q_n \;=\; \sum_{i=0}^{n-1} \big(W_{t_{i+1}} - W_{t_i}\big)^2 \;\longrightarrow\; t \quad \text{in mean square:} \quad \mathbb{E}\big[(Q_n - t)^2\big] \to 0. \tag{38.24}$$

Proof. Each increment is $\mathcal{N}(0, \Delta t_i)$, so $\mathbb{E}[(\Delta W_i)^2] = \Delta t_i$ and $\mathbb{E}[Q_n] = \sum_i \Delta t_i = t$ for every partition. For the spread, a Gaussian has fourth moment $3\,\Delta t_i^2$, so $\mathrm{Var}\big((\Delta W_i)^2\big) = 3\Delta t_i^2 - \Delta t_i^2 = 2\Delta t_i^2$; the increments are independent, so their variances add:

$$\mathbb{E}\big[(Q_n - t)^2\big] = \mathrm{Var}(Q_n) = \sum_i 2\,\Delta t_i^2 \;\le\; 2\,h_{\max} \sum_i \Delta t_i = 2\,h_{\max}\, t \;\longrightarrow\; 0,$$

and mean-square convergence follows. ∎

The sum of squared Brownian increments converges to the elapsed time, a deterministic number, although every increment is random. On a uniform grid with n intervals of $[0, 1]$ the proof gives the standard deviation $\sqrt{2/n}$: 0.35 for $n = 16$ and 0.003 for $n = 2^{18}$.

The proposition is usually written as the **Itô multiplication table**, the working rules for products of differentials:

$$(dW)^2 = dt, \qquad dW\,dt = 0, \qquad (dt)^2 = 0. \tag{38.25}$$

The first rule is Equation 38.24 in shorthand: a squared Brownian increment contributes at order dt. The other two record that $dW\,dt$, of size $\Delta t^{3/2}$, and $(dt)^2 = \Delta t^2$ vanish faster than Δt and so contribute nothing in the limit.

At fine meshes the fluctuations are so small that a single path shows the limit. We sample a path on 2^{18} intervals of $[0, 1]$, view it on coarser subgrids, and print its sum of squares next to the predicted standard deviation, together with the same sum for the smooth path $\sin 2\pi t$ and its predicted value $2\pi^2/n$.

```
n_fine = 2**18
dW_qv = np.sqrt(1.0 / n_fine) * rng.standard_normal(n_fine)
W_qv = np.concatenate([[0.0], np.cumsum(dW_qv)])
print(f'{"n":>7} {"QV of W":>8} {"sqrt(2/n)":>10} {"QV of sin":>10}'
      f' {"2 pi^2/n":>9}')
for k in range(4, 19, 2):
    n = 2**k
    W_n = W_qv[::n_fine // n]                # the same path on a coarser grid
```

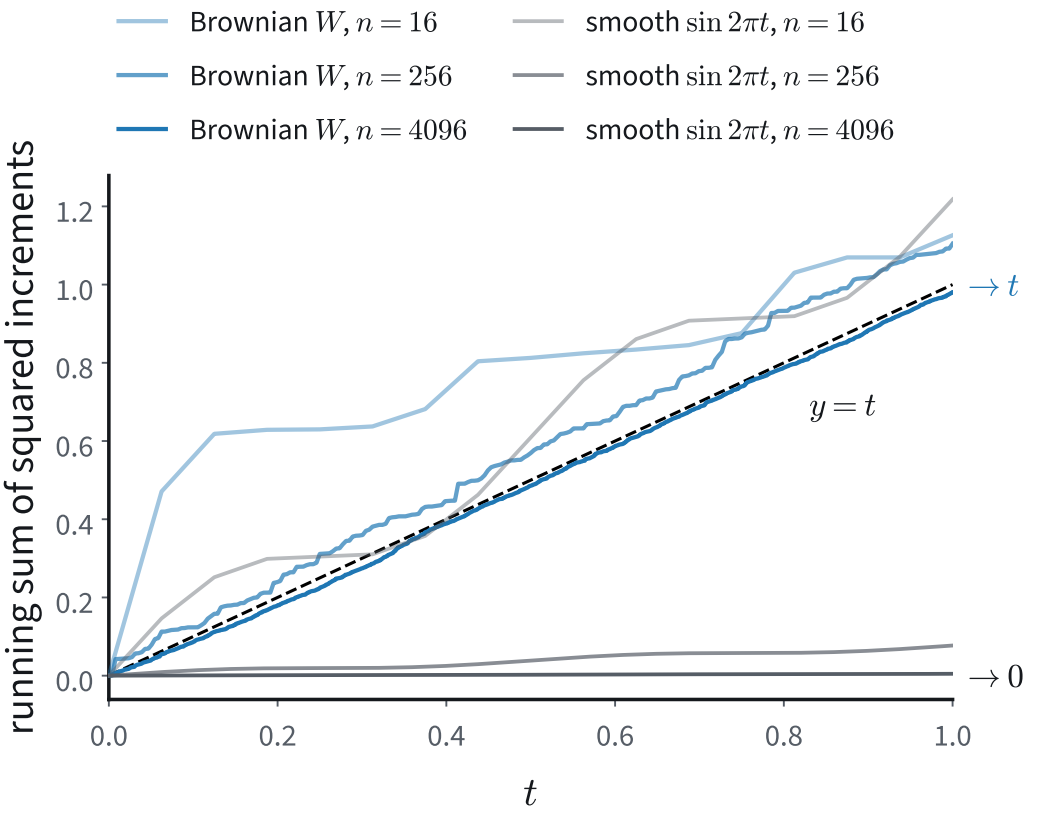


Figure 38.12: Running sums of squared increments along one Brownian path for meshes of $n = 16$, 256, and 4096 intervals (darker shades are finer meshes). The sums approach $y = t$, whereas the corresponding sums for the smooth path $\sin 2\pi t$ (gray) approach zero. The smooth sums shrink like $2\pi^2/n$, which is still 1.2 at $n = 16$, 0.077 at 256 and 0.005 at 4096.

```
    s_n = np.sin(2 * np.pi * np.linspace(0, 1, n + 1))
    print(f'{n:7d} {np.sum(np.diff(W_n)**2):8.4f} {np.sqrt(2 / n):10.4f} '
          f'{np.sum(np.diff(s_n)**2):10.6f} {2 * np.pi**2 / n:9.6f}')
```

```
      n  QV of W  sqrt(2/n)  QV of sin  2 pi^2/n
     16   0.8010     0.3536   1.217927  1.233701
     64   1.1435     0.1768   0.308177  0.308425
    256   0.9221     0.0884   0.077102  0.077106
   1024   0.9833     0.0442   0.019277  0.019277
   4096   0.9863     0.0221   0.004819  0.004819
  16384   1.0009     0.0110   0.001205  0.001205
  65536   0.9975     0.0055   0.000301  0.000301
 262144   0.9978     0.0028   0.000075  0.000075
```

The Brownian column scatters around 1 with the standard deviation $\sqrt{2/n}$ that the proof predicts. On 16 intervals it typically lands between about 0.5 and 1.6, and it settles to $t = 1$ within a few thousandths at $n = 2^{18}$. The smooth column is large on coarse grids, 1.22 at $n = 16$, but it shrinks like $2\pi^2/n$ and agrees with that prediction to three digits from $n = 256$ on.

38.3.2.2 The Itô Integral

The rough path forces us to define integrals against dW directly. We start with the simplest one, $\int_0^t W\, dW$, and find that the usual Riemann-sum definition does not single out an answer.

For a smooth integrator, Riemann sums with the integrand evaluated at the left endpoint, the right endpoint, or the midpoint of each interval all converge to the same limit. For the integrand $H = W$, the right-endpoint sum exceeds the left-endpoint sum by

$$\sum_i W_{t_{i+1}}\,\Delta W_i - \sum_i W_{t_i}\,\Delta W_i = \sum_i \left(W_{t_{i+1}} - W_{t_i}\right)\Delta W_i = \sum_i (\Delta W_i)^2 \;\longrightarrow\; t,$$

by the proposition just proved. The two discretizations of "the same" integral disagree by a finite, deterministic amount, here on one path with 1000 steps.

```
n_lr = 1000
dt_lr = 1.0 / n_lr
t_lr = np.linspace(0, 1, n_lr + 1)
dW_lr = np.sqrt(dt_lr) * rng.standard_normal(n_lr)
W_lr = np.concatenate([[0.0], np.cumsum(dW_lr)])
left = np.sum(W_lr[:-1] * dW_lr)              # integrand at the left endpoint
right = np.sum(W_lr[1:] * dW_lr)              # integrand at the right endpoint
print(f'left sum {left:.4f}   right sum {right:.4f}')
print(f'right - left {right - left:.4f}   sum of (dW)^2 {np.sum(dW_lr**2):.4f}')
```

```
left sum -0.3784   right sum 0.6005
right - left 0.9788   sum of (dW)^2 0.9788
```

The left sum is a random number that changes from run to run. The right sum exceeds it by exactly the printed sum of squared increments, which usually lies within about 0.1 of 1

(its standard deviation is $\sqrt{2/1000} \approx 0.045$). We must choose an evaluation point, and the choice changes the answer.

Itô's choice is the left endpoint, so that the integrand never sees the coming increment. A fair game explains why. Think of ΔW_i as the outcome of the i-th fair bet and of H_i as the stake placed on it. A gambler must choose the stake from what is known at time t_i, before the bet is settled, and the total winnings are $\sum_i H_i\,\Delta W_i$. Because the increment ΔW_i is independent of everything known at t_i, including H_i, each bet has expected winnings $\mathbb{E}[H_i\,\Delta W_i] = \mathbb{E}[H_i]\,\mathbb{E}[\Delta W_i] = 0$, so no betting strategy has positive expected winnings. A stake that could peek at the outcome, as the right-endpoint sum does, would make money on average.

The definition writes this rule down. An integrand is *predictable* if its value on each interval is decided before the increment over that interval is drawn. A **simple predictable process** H is constant on each interval, $H_s = H_i$ for $s \in (t_i, t_{i+1}]$, where H_i is determined by the path up to time t_i and $\mathbb{E}[H_i^2] < \infty$. Its **Itô integral** is the sum of the winnings,

$$\int_0^t H_s\,dW_s \;=\; \sum_i H_i \left(W_{t_{i+1}} - W_{t_i}\right). \tag{38.26}$$

The fair-game argument shows that it has mean zero, $\mathbb{E}\left[\int_0^t H\,dW\right] = 0$. Its variance is computable as well.

Proposition (Itô isometry). *For a simple predictable H,*

$$\mathbb{E}\left[\left(\int_0^t H_s\,dW_s\right)^2\right] \;=\; \int_0^t \mathbb{E}\big[H_s^2\big]\,ds. \tag{38.27}$$

Proof. Expand the square of the sum in Equation 38.26. The step that needs care is the cross terms. A cross term with $i < j$ contains the factor ΔW_j, which is independent of everything known at time t_j, including the product $H_i\,\Delta W_i\,H_j$. So the expectation factorizes: $\mathbb{E}\big[H_i\,\Delta W_i\,H_j\,\Delta W_j\big] = \mathbb{E}\big[H_i\,\Delta W_i\,H_j\big]\,\mathbb{E}[\Delta W_j] = 0$. A diagonal term factorizes the same way into $\mathbb{E}\big[H_i^2\big]\,\mathbb{E}\big[(\Delta W_i)^2\big] = \mathbb{E}\big[H_i^2\big]\,\Delta t_i$. Summing the diagonal gives $\sum_i \mathbb{E}[H_i^2]\,\Delta t_i = \int_0^t \mathbb{E}[H_s^2]\,ds$. ∎

The isometry is also what carries the definition beyond step processes, to integrands such as $H = W$ itself. It says that $H \mapsto \int H\,dW$ preserves the norm $\|H\|^2 = \int_0^t \mathbb{E}[H_s^2]\,ds$. So if simple predictable processes converge to H in this norm, their integrals converge in mean square, and the limit is taken as the definition of $\int_0^t H\,dW$. The zero mean and the isometry survive the limit and hold for every predictable H of finite norm. In particular, the left-endpoint sums of a continuous integrand of this kind converge to its integral. We do not prove these statements; see Øksendal (2003), chapter 3.

The isometry turns the variance of a noise integral into an ordinary integral of a second moment. For the integral we started from, $\mathbb{E}[W_s^2] = s$, so

$$\operatorname{Var}\left(\int_0^1 W_s\, dW_s\right) = \int_0^1 s\, ds = \tfrac{1}{2}.$$

A sample of 20,000 left-endpoint sums on the same 1000-step grid shows the size of the fluctuations.

```
dW_iso = np.sqrt(dt_lr) * rng.standard_normal((20000, n_lr))
W_iso = np.cumsum(dW_iso, axis=1) - dW_iso    # W at the left end of each step
I_iso = np.sum(W_iso * dW_iso, axis=1)        # 20,000 draws of int_0^1 W dW
print(f'mean {I_iso.mean():.4f}   (theory 0)')
print(f'variance {I_iso.var():.4f}   (theory 1/2)')
```

```
mean -0.0010   (theory 0)
variance 0.4988   (theory 1/2)
```

Mean and variance agree with 0 and $\frac{1}{2}$ within their standard errors, 0.005 and 0.013; the second is larger than for a Gaussian because the integral has heavy tails. Itô's lemma below gives the same $\frac{1}{2}$ without sampling.

The other evaluation points define other integrals. The midpoint choice, or the average of the left and right sums, defines the *Stratonovich integral* (Stratonovich 1966). It obeys the ordinary chain rule but gives up the zero mean, because its integrand sees half of the coming increment. This book uses Itô's convention, for two reasons: the zero mean makes expectations easy, and the simplest simulation step, which evaluates the coefficients at the start of the step, is an Itô sum (problem 9).

38.3.2.3 Drift, Diffusion, and Solutions

With the integral in hand, the forward-process template Equation 38.22 acquires a meaning. A **stochastic differential equation** is written

$$dX = f(X,t)\, dt + g(X,t)\, dW, \tag{38.28}$$

and the differential form is shorthand for the integral equation

$$X_t = X_0 + \int_0^t f(X_s, s)\, ds + \int_0^t g(X_s, s)\, dW_s,$$

an ordinary integral plus an Itô integral. The **drift** f steers the average motion and the **diffusion** coefficient g sets the size of the noise. Over a short step, with the coefficients frozen at the start of the step, $\mathbb{E}[dX] = f\, dt$ and $\operatorname{Var}(dX) = g^2\, dt$. So f is the conditional mean velocity and g^2 the rate at which variance is pumped in. With $g \equiv 0$ the equation is the ODE of Section 38.1.

For example, $dX = g\, dW$ with a constant g has no drift integral, and the Itô integral of a constant is $g\, W_t$, so $X_t = X_0 + g\, W_t$. This is the noise-only panel of Figure 38.10 with $g = \sqrt{2}$.

As for ODEs, Lipschitz coefficients guarantee a solution. Suppose that f and g are Lipschitz in x and grow at most linearly, $|f(x,t)| + |g(x,t)| \le C(1 + |x|)$, and that X_0 is independent

of W with $\mathbb{E}[X_0^2] < \infty$. Then the equation has a unique solution (Øksendal (2003), Theorem 5.2.1). The proof is a stochastic version of the Picard iteration of Section 38.1.1.3, with the isometry in place of the integral bound.

We distinguish two kinds of noise. In **additive** noise the amplitude depends on time only, $g = g(t)$, as in the template Equation 38.22; in **multiplicative** noise it depends on the state, $g = g(X, t)$. The difference looks cosmetic, but it decides how accurately a simulated path follows the true one (Section 38.3.4.2). The diffusion-model SDEs of this chapter have additive noise, in the two families of Figure 38.10: noise alone (variance-exploding) and noise balanced by a restoring drift.

Unlike an ODE solution, a solution of an SDE is not one curve but a distribution over paths. Running the equation many times from the same start with fresh noise gives a cloud of paths like those of Figure 38.10, and what a generative model cares about is the law of the cloud at each time.

38.3.2.4 Itô's Lemma

To compute with an SDE we need the chain rule: if X solves an SDE, what equation does a function $\phi(X_t, t)$ solve? The ordinary chain rule gives $d(W^2) = 2W\,dW$, an Itô integral with mean zero, but $\mathbb{E}[W_t^2] = t$. Something is missing.

One step of a walk shows the missing term. Take $\phi(x) = x^2$ and one step $\pm\sqrt{\Delta t}$ from x, each sign with probability $\frac{1}{2}$. The average of ϕ after the step is

$$\tfrac{1}{2}\left[(x + \sqrt{\Delta t})^2 + (x - \sqrt{\Delta t})^2\right] = x^2 + \Delta t = \phi(x) + \tfrac{1}{2}\phi''(x)\,\Delta t,$$

since $\phi'' = 2$. The rise comes from the square of the step, $(\Delta W)^2 = \Delta t$, which the ordinary chain rule throws away.

With $x = 0.7$ and $\Delta t = 0.25$ the step lands at 0.2 or 1.2, where ϕ is 0.04 or 1.44. The average 0.74 lies above $\phi(0.7) = 0.49$ by $0.25 = \Delta t$ (Figure 38.13). A convex function gains from symmetric noise, and because a squared Brownian increment is of order dt, the gain per unit time, $\frac{1}{2}\phi'' g^2$, does not vanish in the limit.

The general formula comes from the same computation. Let X solve $dX = f\,dt + g\,dW$ and let $\phi(x, t)$ be once continuously differentiable in time and twice in the state. A small change of ϕ, expanded to second order because first order is no longer enough, is

$$d\phi = \phi_t\,dt + \phi_x\,dX + \tfrac{1}{2}\phi_{xx}\,(dX)^2 + \cdots.$$

Substitute $dX = f\,dt + g\,dW$ and multiply out $(dX)^2$ with the table Equation 38.25:

$$(dX)^2 = f^2(dt)^2 + 2fg\,dt\,dW + g^2(dW)^2 = g^2\,dt.$$

The drift and cross terms vanish, and the noise term survives at first order. Collecting terms gives the chain rule of stochastic calculus. What we have given is a heuristic derivation, so we state the result as a theorem without proof.

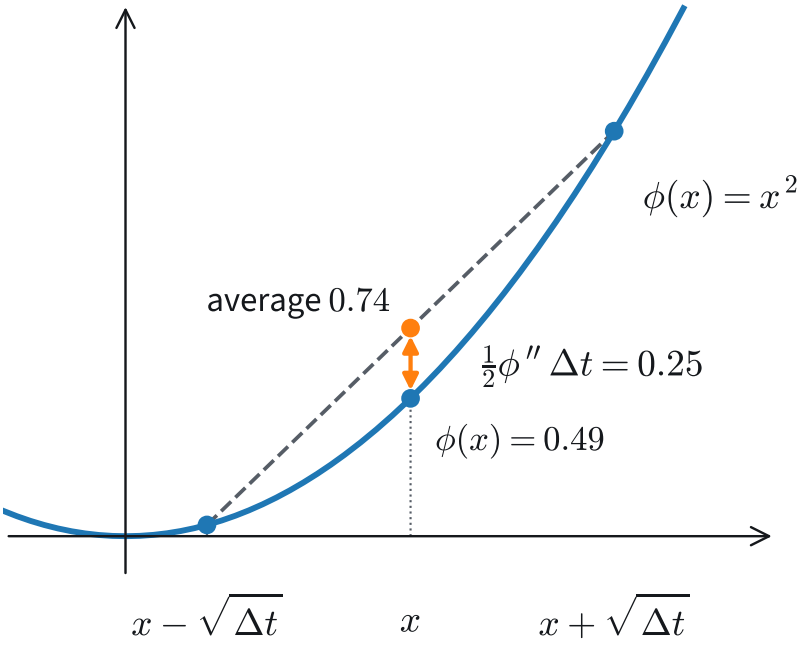


Figure 38.13: The Itô correction as Jensen's gap. One step of $\pm\sqrt{\Delta t}$ from x moves a convex ϕ to an average that lies above $\phi(x)$ by $\frac{1}{2}\phi''(x)\,\Delta t$; here $\phi = x^2$, $x = 0.7$, $\Delta t = 0.25$, and the gap is 0.25.

Theorem (Itô's lemma). *Let X solve $dX = f\,dt + g\,dW$ under the conditions of the previous subsection, and let $\phi \in C^{1,2}$, once continuously differentiable in time and twice in the state. Then $Y_t = \phi(X_t, t)$ satisfies*

$$d\phi \;=\; \left(\phi_t + f\,\phi_x + \tfrac{1}{2} g^2\,\phi_{xx}\right) dt \;+\; g\,\phi_x\,dW. \tag{38.29}$$

Relative to the ordinary chain rule, the new term is $\frac{1}{2}g^2\phi_{xx}\,dt$, the **Itô correction**. The proof has to control the remainder terms of the Taylor expansion in mean square (Øksendal (2003), chapter 4).

Example 1: W^2. Take $X = W$ (so $f = 0$, $g = 1$) and $\phi(x) = x^2$. The lemma gives $d(W^2) = 2W\,dW + dt$, the dt that the ordinary chain rule dropped. In integrated form,

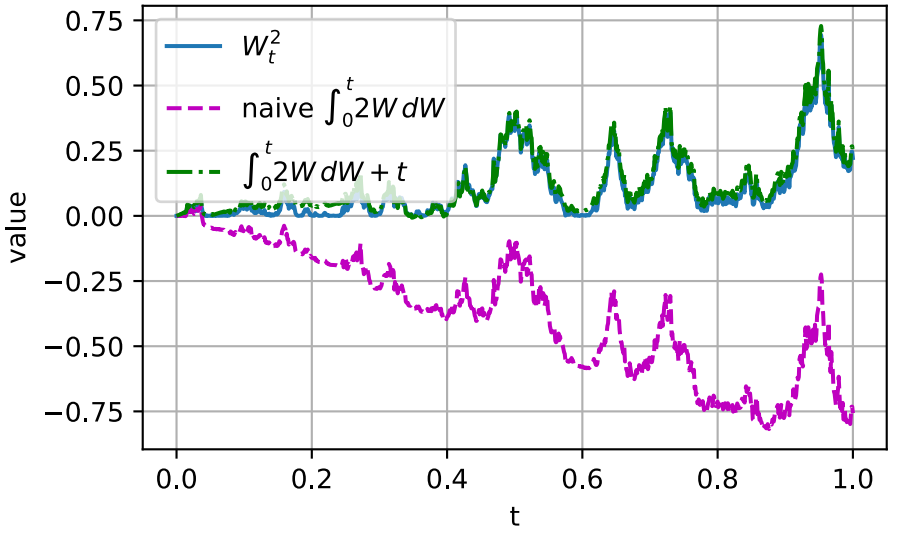
$$W_t^2 = \int_0^t 2W_s\,dW_s + t, \qquad \text{that is,} \qquad \int_0^t W_s\,dW_s = \tfrac{1}{2}\left(W_t^2 - t\right). \tag{38.30}$$

The $-t$ restores the zero mean of the Itô integral, since $\mathbb{E}[W_t^2] = t$. The formula also gives the variance of the integral a second way: W_1^2 is a squared standard normal, with variance 2, so $\mathrm{Var}\left(\frac{1}{2}(W_1^2 - 1)\right) = \frac{1}{4}\cdot 2 = \frac{1}{2}$, the number the isometry gave.

The correction can be watched as it accumulates. On any grid the gap between W_t^2 and the left-endpoint sum for $\int_0^t 2W\,dW$ telescopes, $W_t^2 - \sum_{t_i<t} 2W_{t_i}\Delta W_i = \sum_{t_i<t}(\Delta W_i)^2$, so it is the running quadratic variation and should follow the line t. We reuse the path of `#sdes-left-right-sums`.

```
ito = np.concatenate([[0.0], np.cumsum(2 * W_lr[:-1] * dW_lr)])  # int 2W dW
gap = W_lr**2 - ito
print(f'final gap W_1^2 - int 2W dW = {gap[-1]:.4f}   (Ito correction: t = 1)')
print(f'max |gap_t - t| along the path = {np.abs(gap - t_lr).max():.4f}')
d2l.plot(t_lr, [W_lr**2, ito, ito + t_lr], 't', 'value',
         legend=['$W_t^2$', 'naive $\\int_0^t 2W\\,dW$',
                 '$\\int_0^t 2W\\,dW + t$'], figsize=(5, 3))
```

```
final gap W_1^2 - int 2W dW = 0.9788   (Ito correction: t = 1)
max |gap_t - t| along the path = 0.0479
```



The final gap equals the sum of squared increments printed by the earlier cell, so it usually lies within about 0.1 of 1. Along the whole path the gap follows t closely, typically within about 0.05 and rarely by more than 0.1: by the proof of Equation 38.24 its standard deviation at time t is $\sqrt{2t\,\Delta t}$, which is 0.045 at $t = 1$ for 1000 steps and shrinks as the grid is refined. The naive curve sags below W_t^2, and adding t puts it back on top.

Example 2: geometric Brownian motion. The SDE $dX = \mu X\,dt + \nu X\,dW$ with $X_0 > 0$ models a quantity that grows at mean rate μ with noise proportional to its size, such as a stock price (Black and Scholes 1973). Its noise is multiplicative. Apply the lemma to $\phi(x) = \log x$, with $\phi' = 1/x$, $\phi'' = -1/x^2$, and $g^2 = \nu^2 x^2$. The logarithm is smooth only for $x > 0$, so assume for the moment that X stays positive:

$$d\log X = \big(\mu - \tfrac{1}{2}\nu^2\big)\,dt + \nu\,dW, \qquad \text{so} \qquad X_t = X_0 \exp\!\big(\big(\mu - \tfrac{1}{2}\nu^2\big)\,t + \nu W_t\big). \tag{38.31}$$

The formula we obtain is positive, and the lemma applied to it as a function of W_t and t confirms that it solves the equation, which removes the assumption. The $-\frac{1}{2}\nu^2$ is the correction with $\phi'' < 0$: a concave function loses from symmetric noise. Noise therefore lowers the typical growth rate $\mu - \frac{1}{2}\nu^2$ below the mean growth rate μ, and problem 4 shows how far the mean and the median of X_t drift apart. We will measure simulation errors against this exact solution.

The multivariate form follows the same pattern, and we take it as given (Øksendal 2003). For $d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}\,d\mathbf{W}$ with $\mathbf{X} \in \mathbb{R}^d$, a matrix $\mathbf{G}$, and multivariate Brownian motion $\mathbf{W}$, the multiplication table reads $dW_i\,dW_j = \delta_{ij}\,dt$, and a twice continuously differentiable ϕ satisfies

$$d\phi(\mathbf{X}) \;=\; \nabla\phi \cdot d\mathbf{X} \;+\; \tfrac{1}{2}\sum_{i,j} \big(\mathbf{G}\mathbf{G}^{\top}\big)_{ij}\,\frac{\partial^2 \phi}{\partial x_i\,\partial x_j}\,dt. \tag{38.32}$$

The correction weights each second derivative by the covariance rate $\mathbf{G}\mathbf{G}^{\top}$ of the noise. For

the noise $\mathbf{G} = g(t)\,\mathbf{I}$ of the diffusion-model SDEs it reduces to $\frac{1}{2}g(t)^2 \sum_i \partial^2\phi/\partial x_i^2 \; dt$, a multiple of the Laplacian of ϕ. This term becomes the diffusion term of the Fokker–Planck equation in the next section.

The lemma needs two continuous derivatives in the state. It does not apply as stated to a function with a kink, such as $\phi(x) = |x|$.

38.3.3 The Ornstein–Uhlenbeck Process

The running example now gets its name and its solution. The Ornstein–Uhlenbeck (OU) process (Uhlenbeck and Ornstein 1930) is

$$dX = -\theta X\, dt + g\, dW, \qquad \theta > 0, \tag{38.33}$$

the decay ODE $\dot{x} = -\theta x$ of Section 38.1.2 with noise of constant amplitude g added. The drift pulls excursions back toward zero at rate θ, and the noise keeps producing new ones (Figure 38.14). The recurrence of the opening is this process with $\theta = 1$ and $g = \sqrt{2}$, advanced on a grid with step $h = 0.01$ by freezing the drift over each step: one step multiplies by $1 - \theta h = 0.99$ and adds noise of standard deviation $g\sqrt{h} = \sqrt{0.02}$.

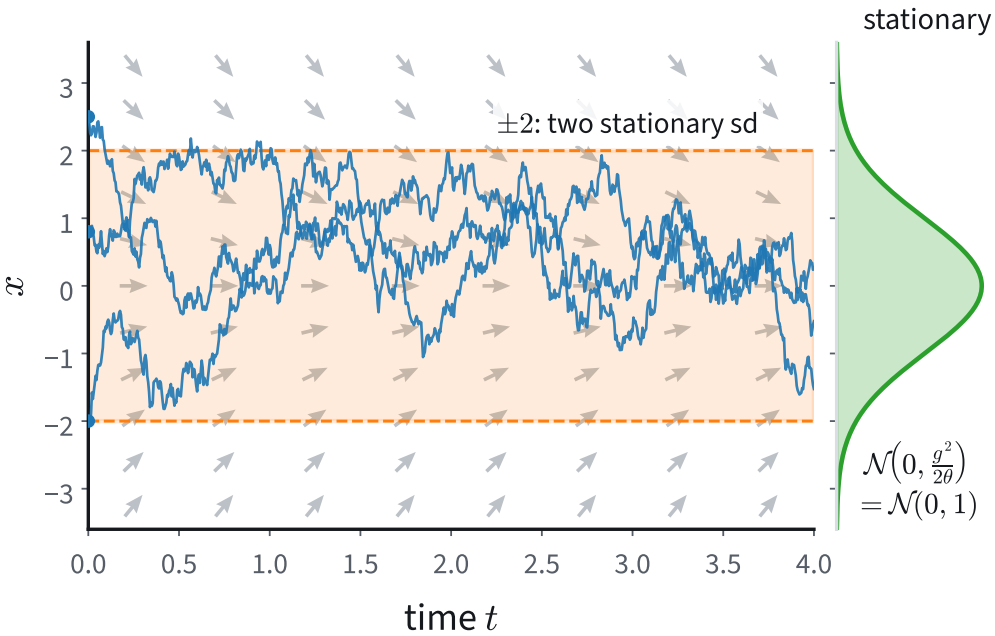


Figure 38.14: Mean reversion. The drift $-\theta x$ (arrows) pulls every path toward 0 at rate $\theta = 1$ while noise of amplitude $g = \sqrt{2}$ keeps it moving; paths from three starts forget them within a few multiples of $1/\theta$ and then fluctuate inside the stationary band of two standard deviations, $\pm 2\sqrt{g^2/2\theta} = \pm 2$. Right: the stationary density.

38.3.3.1 The Transition Kernel

The OU process is one of the few SDEs whose law is known in closed form at every time, and this is why diffusion models are built on it. The law given the start is Gaussian. Of all the results of this section, this is the one the rest of the book uses most.

Proposition (OU transition kernel). *For the OU process Equation 38.33,*

$$X_t \mid X_0 = x_0 \;\sim\; \mathcal{N}\!\left(x_0\, e^{-\theta t},\; \frac{g^2}{2\theta}\,(1 - e^{-2\theta t})\right). \tag{38.34}$$

Proof. We use the integrating factor that solves the decay ODE. Apply Itô's lemma Equation 38.29 to $\phi(x,t) = e^{\theta t}x$, for which $\phi_t = \theta e^{\theta t}x$, $\phi_x = e^{\theta t}$, and $\phi_{xx} = 0$, so there is no correction:

$$d(e^{\theta t}X_t) = (\theta e^{\theta t}X_t - \theta e^{\theta t}X_t)\,dt + e^{\theta t}g\,dW_t = g\,e^{\theta t}\,dW_t.$$

The drift cancels, and integrating from 0 to t and multiplying by $e^{-\theta t}$ gives

$$X_t = X_0\, e^{-\theta t} + g\int_0^t e^{-\theta(t-s)}\,dW_s. \tag{38.35}$$

The remaining integral has a deterministic integrand, so it is a limit of weighted sums of independent Gaussian increments and is Gaussian with mean zero (we grant that mean-square limits of Gaussians are Gaussian). The Itô isometry Equation 38.27 gives its variance:

$$\operatorname{Var}(X_t \mid X_0) = g^2\int_0^t e^{-2\theta(t-s)}\,ds = \frac{g^2}{2\theta}\,(1 - e^{-2\theta t}). \qquad \blacksquare$$

The solution Equation 38.35 reads as a fading memory. The start decays at rate θ, and a noise kick from time s survives with weight $e^{-\theta(t-s)}$: recent noise counts, old noise is forgotten. In the recurrence the same thing happens step by step, since a kick from k steps ago survives with weight $0.99^k \approx e^{-0.01k}$. Every later section, and Section 17.2 in the main text, writes the kernel with two coefficients, the mean factor and the standard deviation,

$$\alpha_t = e^{-\theta t}, \qquad \sigma_t^2 = \frac{g^2}{2\theta}\,(1 - e^{-2\theta t}), \qquad \text{so that} \qquad X_t \mid X_0 = x_0 \sim \mathcal{N}(\alpha_t x_0,\, \sigma_t^2). \tag{38.36}$$

The data enter only through the mean $\alpha_t x_0$, and the noise only through the variance σ_t^2. For the running example ($\theta = 1$, $g = \sqrt{2}$, $\sigma_t^2 = 1 - e^{-2t}$) started at $x_0 = 2$, the kernel is $\mathcal{N}(1.213,\, 0.795^2)$ at $t = 0.5$, $\mathcal{N}(0.736,\, 0.930^2)$ at $t = 1$, and $\mathcal{N}(0.037,\, 1.000^2)$ at $t = 4$. Because the kernel is Gaussian at every time and for every start, everything about the noised state given its start is known in closed form, including the gradient of its log-density, $-(x - \alpha_t x_0)/\sigma_t^2$.

The recurrence is only a discretization of the process, so its cloud should follow the kernel closely but need not follow it exactly. We run it as a plain loop on $10{,}000$ copies of $x_0 = 2$,

save the cloud at five times, and print its mean and standard deviation next to $2\alpha_t$ and σ_t, together with the fraction of the cloud inside $2\alpha_t \pm 2\sigma_t$.

```
theta_ou, g_ou, h_ou = 1.0, np.sqrt(2.0), 0.01
X_ou = np.full(10000, 2.0)                    # 10,000 copies of x_0 = 2
snap_steps = {round(t / h_ou): t for t in (0.5, 1.0, 1.5, 2.0, 4.0)}
X_ou_snap = {}
for k in range(1, 401):
    xi = rng.standard_normal(X_ou.size)
    X_ou = (1 - theta_ou * h_ou) * X_ou + g_ou * np.sqrt(h_ou) * xi
    if k in snap_steps:
        X_ou_snap[snap_steps[k]] = X_ou.copy()
print(f'{"t":>4} {"mean":>6} {"2alpha_t":>8} {"std":>6} {"sigma_t":>7}'
      f' {"inside":>7}')
for t, X in X_ou_snap.items():
    alpha_t = np.exp(-theta_ou * t)
    sigma_t = np.sqrt(g_ou**2 / (2 * theta_ou) * (1 - alpha_t**2))
    inside = np.mean(np.abs(X - 2 * alpha_t) < 2 * sigma_t)
    print(f'{t:4.1f} {X.mean():6.3f} {2 * alpha_t:8.3f} {X.std():6.3f} '
          f'{sigma_t:7.3f} {inside:7.3f}')
```

```
   t   mean 2alpha_t    std sigma_t  inside
 0.5  1.211    1.213  0.801   0.795   0.954
 1.0  0.736    0.736  0.925   0.930   0.956
 1.5  0.455    0.446  0.973   0.975   0.955
 2.0  0.277    0.271  0.987   0.991   0.957
 4.0  0.041    0.037  1.008   1.000   0.951
```

Means and standard deviations agree with the kernel within two or three standard errors (about 0.01 for a mean and 0.007 for a standard deviation from $10{,}000$ samples), and about 95% of the cloud lies inside the band at every time, as for any Gaussian. At this sample size the discretization cannot be seen. The cloud from the single point 2 widens quickly and then stops widening.

38.3.3.2 The Stationary Distribution

A forward process must forget its start. Letting $t \to \infty$ in the kernel Equation 38.34 shows that the OU process does: the mean $x_0 e^{-\theta t}$ decays to zero and the variance approaches $g^2/(2\theta)$, so

$$X_t \mid X_0 = x_0 \;\longrightarrow\; \mathcal{N}\!\left(0,\, \frac{g^2}{2\theta}\right) \quad \text{in distribution, for every } x_0.$$

This is the distribution-independent endpoint that Section 38.3.1.1 asked for. The limit is also invariant: a process started in it stays in it. Indeed, if $X_0 \sim \mathcal{N}(0, g^2/2\theta)$ is independent of W, then by Equation 38.35 X_t is the sum of $X_0 e^{-\theta t}$ and an independent Gaussian with mean zero and variance σ_t^2, hence Gaussian with mean zero and variance

$$e^{-2\theta t}\,\frac{g^2}{2\theta} + \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right) = \frac{g^2}{2\theta}.$$

The decay of the start removes as much variance as the noise adds. The approach to the stationary law happens on the time scale $1/\theta$, the relaxation time: after a few multiples of it, $e^{-\theta t}$ is negligible. For the running example $g^2/2\theta = 1$, so the stationary law is $\mathcal{N}(0,1)$.

The recurrence has a stationary law of its own, and it is slightly different. One step maps a variance v to $(1-\theta h)^2 v + g^2 h$. From $v = 1$ a step of the running example gives $0.9801 + 0.02 = 1.0001$, slightly more than 1, and the fixed point of the map is

$$v_\infty = \frac{g^2 h}{1-(1-\theta h)^2} = \frac{0.02}{0.0199} = 1.005 \qquad (h = 0.01).$$

The recurrence therefore settles half a percent above the exact variance 1. This is the first sign of a discretization error, and Section 38.3.4.2 shows that it shrinks in proportion to h.

The cloud saved at $t = 4$, after four relaxation times, can be compared with the kernel quantile by quantile.

```
X_4 = X_ou_snap[4.0]
z = np.array([-1.96, -0.674, 0.0, 0.674, 1.96])  # N(0, 1) quantiles
q_sim = np.percentile(X_4, [2.5, 25, 50, 75, 97.5])
print('simulated quantiles at t=4:', q_sim.round(2))
print('N(2 e^-4, 1) quantiles:    ', (2 * np.exp(-4) + z).round(2))
a = 1 - theta_ou * h_ou                          # the recurrence's decay factor
print(f'variance {X_4.var():.4f}   (stationary 1, '
      f'recurrence fixed point {g_ou**2 * h_ou / (1 - a**2):.4f})')
```

```
simulated quantiles at t=4: [-1.94 -0.63  0.04  0.69  2.03]
N(2 e^-4, 1) quantiles:     [-1.92 -0.64  0.04  0.71  2.  ]
variance 1.0156   (stationary 1, recurrence fixed point 1.0050)
```

The five quantiles agree with those of the Gaussian within a few hundredths, and the residual mean $2e^{-4} = 0.037$ is still visible in the median. The half percent, however, is out of reach: the sample variance has a standard error of 0.014 and cannot separate 1 from 1.005.

38.3.3.3 The Variance-Preserving Normalization

The stationary variance $g^2/2\theta$ is a free parameter, and diffusion models fix it to 1 by choosing

$$g^2 = 2\theta \qquad \Longrightarrow \qquad X_t \mid X_0 = x_0 \;\longrightarrow\; \mathcal{N}(0,1).$$

The running example makes this choice. With it, $\sigma_t^2 = 1 - \alpha_t^2$, and the coefficient $\bar{\alpha}_t = \alpha_t^2 = e^{-2\theta t}$ of DDPM (Ho et al. 2020) (our α_t is that paper's $\sqrt{\bar{\alpha}_t}$) lets us write the solution Equation 38.35 in distribution as

$$X_t \;=\; \sqrt{\bar{\alpha}_t}\, X_0 \;+\; \sqrt{1-\bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(0,1) \text{ independent of } X_0. \tag{38.37}$$

The marginal is a weighted mix of data and fresh noise, with weights whose squares add to one. Suppose the data have zero mean and unit variance. Taking variances of the two independent terms gives

$$\mathrm{Var}(X_t) = \bar{\alpha}_t \mathrm{Var}(X_0) + (1 - \bar{\alpha}_t) = \bar{\alpha}_t + (1 - \bar{\alpha}_t) = 1 \qquad \text{for all } t.$$

This is what **variance-preserving** (VP) means: for unit-variance data the marginal variance equals 1 at every time, not only in the limit. The process moves variance from the data to the noise, a fraction $\bar{\alpha}_t$ still carried by the data and $1-\bar{\alpha}_t$ already replaced by noise, without changing the total. The name describes unit-variance data only. For data of variance s^2 the marginal variance drifts from s^2 to 1 (problem 8), so the scale of the data matters. Images scaled to $[-1, 1]$, as in Section 17.1, have variance below one, while latent diffusion models rescale their latents to about unit variance (Rombach et al. 2022).

The variance is preserved; the shape of the distribution is not. A convenient measure is the fourth moment, which is 3 for a standard Gaussian. Expanding Equation 38.37 gives $\mathbb{E}[X_t^4] = 3 + \bar{\alpha}_t^2 \left(\mathbb{E}[X_0^4] - 3\right)$ (same problem). For data that are ± 1 with probability $\frac{1}{2}$ each (the Rademacher distribution), $\mathbb{E}[X_0^4] = 1$, the smallest fourth moment a unit-variance law can have, and the prediction is $3 - 2\bar{\alpha}_t^2$: 2.264 at $t = 0.25$, 2.963 at $t = 1$, and 3.000 at $t = 3$. The cell below runs the recurrence from $50{,}000$ such points. The recurrence adds a Gaussian $S \sim \mathcal{N}(0, s_n^2)$ to $a^n X_0$, where $a = 1 - \theta h$ and $s_n^2 = g^2 h(1 - a^{2n})/(1 - a^2)$, so its own moments are known as well (columns `rec`).

```
X_vp = rng.choice([-1.0, 1.0], size=50000)  # unit variance, far from Gaussian
X_vp_snap = {0: X_vp}
for k in range(1, 301):
    xi = rng.standard_normal(X_vp.size)
    X_vp = a * X_vp + g_ou * np.sqrt(h_ou) * xi
    if k in (25, 100, 300):
        X_vp_snap[k] = X_vp
print(f'{"t":>4} {"Var":>6} {"rec":>6} {"E X^4":>6} {"rec":>6}'
      f' {"3-2abar^2":>9}')
for k, X in X_vp_snap.items():
    s2 = g_ou**2 * h_ou * (1 - a**(2 * k)) / (1 - a**2)  # noise variance so far
    m4_rec = a**(4 * k) + 6 * a**(2 * k) * s2 + 3 * s2**2
    abar = np.exp(-2 * theta_ou * k * h_ou)
    print(f'{k * h_ou:4.2f} {X.var():6.3f} {a**(2 * k) + s2:6.3f} '
          f'{np.mean(X**4):6.3f} {m4_rec:6.3f} {3 - 2 * abar**2:9.3f}')
```

```
   t    Var    rec  E X^4    rec 3-2abar^2
0.00  1.000  1.000  1.000  1.000     1.000
0.25  1.013  1.002  2.341  2.280     2.264
1.00  1.012  1.004  3.043  2.990     2.963
3.00  1.010  1.005  3.043  3.030     3.000
```

The variance stays at 1 within sampling error while the fourth moment climbs from 1 to about 3. The recurrence's own values sit slightly above the continuous-time ones (2.280 against 2.264 at $t = 0.25$, 3.030 against 3.000 at $t = 3$), the same discretization error as before.

Diffusion models usually let the rate vary with time. The **VP-SDE** of score-based diffusion (Y. Song, Sohl-Dickstein, et al. 2021),

$$dX = -\tfrac{1}{2}\beta(t)\, X\, dt + \sqrt{\beta(t)}\, dW, \tag{38.38}$$

is an OU process with time-dependent rate $\theta(t) = \beta(t)/2$ and noise amplitude $g(t)^2 = \beta(t)$. It keeps $g^2 = 2\theta$ at every instant, so the variance stays at 1 along the whole schedule, and its marginal is Equation 38.37 with $\bar{\alpha}_t = e^{-\int_0^t \beta(s)\,ds}$ (problem 10 proves that its transitions are exact Gaussians). This is the forward process that score matching learns to reverse, and the variance-preserving path of Section 17.2.

38.3.4 Simulation

38.3.4.1 The Euler–Maruyama Scheme

Most SDEs have no closed-form solution, so we need to simulate them. Our recurrence already shows how. To simulate Equation 38.28 on a grid $t_n = n\,\Delta t$ we freeze the coefficients over each step, as forward Euler does for ODEs (Section 38.1.3), and add a Gaussian kick whose size the increment form Equation 38.23 dictates. The **Euler–Maruyama** update (Maruyama 1955) is

$$\hat{X}_{n+1} = \hat{X}_n + f(\hat{X}_n, t_n)\,\Delta t + g(\hat{X}_n, t_n)\,\sqrt{\Delta t}\,\xi_n, \qquad \xi_n \sim \mathcal{N}(0,1) \text{ i.i.d.}, \tag{38.39}$$

where $\hat{X}_n$ approximates X_{t_n} and $\hat{X}_0 = X_0$. The scheme is a drift step plus a Brownian increment over the step. The kick scales as $\sqrt{\Delta t}$, since any other exponent makes the noise vanish or explode in the limit (Section 38.3.1.2), and with $g = 0$ the scheme is forward Euler.

For the OU process the step reads

$$\hat{X}_{n+1} = (1 - \theta\,\Delta t)\,\hat{X}_n + g\sqrt{\Delta t}\,\xi_n = 0.99\,\hat{X}_n + 0.1414\,\xi_n, \tag{38.40}$$

where the second form is the running example at $\Delta t = 0.01$, the recurrence we have been running all along.

For this process we could have done better. With $g^2 = 2\theta$ the kernel Equation 38.34 over one interval is $\mathcal{N}(e^{-\theta\Delta t}x, 1 - e^{-2\theta\Delta t})$, and sampling from it is the forward step of DDPM (Section 17.2), $x_k = \sqrt{1-\beta}\,x_{k-1} + \sqrt{\beta}\,\epsilon$ with $1 - \beta = e^{-2\theta\Delta t}$. DDPM's step is therefore exact at every step size, and Euler–Maruyama is its first-order approximation: at $\theta = 1$ and $\Delta t = 0.01$ the exact step multiplies by $e^{-0.01} = 0.99005$ and adds noise of variance $1 - e^{-0.02} = 0.0198$, against 0.99 and 0.02. Such exact steps need a linear equation (problem 10); for a general drift there is only the approximate scheme.

The helper below implements Equation 38.39 for a batch of paths at once. It takes the increments ΔW as an argument, so that two simulations can be driven by the same noise.

```
def euler_maruyama(f, g, x0, t, dW):
    """Simulate dX = f dt + g dW on the grid t for a batch of paths."""
    X = np.empty((len(x0), len(t)))
    X[:, 0] = x0
    for k in range(len(t) - 1):
        h = t[k + 1] - t[k]
        X[:, k + 1] = (X[:, k] + f(X[:, k], t[k]) * h
                       + g(X[:, k], t[k]) * dW[:, k])
    return X
```

A quick test of the helper: with the noise switched off it must reproduce forward Euler on $\dot{x} = -x$.

```
t_ode = np.linspace(0, 2.0, 201)
X_ode = euler_maruyama(lambda x, s: -x, lambda x, s: 0.0,
                       np.ones(1), t_ode, np.zeros((1, 200)))
print('g = 0 recovers Euler on dx/dt = -x: '
      f'max |X_t - e^(-t)| = {np.abs(X_ode[0] - np.exp(-t_ode)).max():.5f}')
```

```
g = 0 recovers Euler on dx/dt = -x: max |X_t - e^(-t)| = 0.00185
```

The error is the Euler error at $h = 10^{-2}$, which is largest near $t = 1$, where it equals $e^{-1} - 0.99^{100}$. With the noise switched on, "error" itself needs a definition, and there are two.

38.3.4.2 Strong and Weak Convergence

A simulated path is random, so we must say what it means for it to be close to the true solution. There are two answers, depending on whether we care about paths or about laws.

Let $N = T/\Delta t$ be the number of steps, so that $\hat{X}_N$ approximates X_T. Drive the scheme and the exact solution with the same Brownian increments. The **strong error** $\mathbb{E}\,|\hat{X}_N - X_T|$ measures pathwise tracking: does the simulated trajectory follow the true trajectory driven by that noise? The **weak error** $|\mathbb{E}\,\varphi(\hat{X}_N) - \mathbb{E}\,\varphi(X_T)|$, for smooth test functions φ, measures only whether the law at time T comes out right. A generative model needs weak accuracy, since it only uses the samples at the final time. Strong accuracy matters when paths are coupled across resolutions, as in multilevel Monte Carlo, which uses cheap coarse paths to correct expensive fine ones (problem 11). Figure 38.15 shows the two notions side by side.

The convergence rates are classical theorems, and their proofs are beyond this section; they are in Kloeden and Platen (1992), together with the precise smoothness and growth conditions on f, g, and φ, which we only indicate. A scheme has strong order p if its strong error is at most $C\,\Delta t^{p}$, and weak order p likewise.

Theorem (orders of Euler–Maruyama; stated without proof). *Let $\mathbb{E}[X_0^2] < \infty$ and let f and g be Lipschitz in x with linear growth and sufficiently regular in t.*

1. *Euler–Maruyama has strong order $\frac{1}{2}$: $\mathbb{E}\,|\hat{X}_N - X_T| \le C\,\Delta t^{1/2}$.*
2. *If the noise is additive, $g = g(t)$, and the coefficients are smooth with bounded derivatives, the strong order is 1: $\mathbb{E}\,|\hat{X}_N - X_T| \le C\,\Delta t$.*
3. *If f, g, and φ are smooth with derivatives of polynomial growth and X_0 has finite moments of all orders, the weak order is 1, for additive and multiplicative noise alike: $|\mathbb{E}\,\varphi(\hat{X}_N) - \mathbb{E}\,\varphi(X_T)| \le C\,\Delta t$.*

A count of single-step errors explains the exponents. It is not a proof: it adds the errors of the steps as if each stayed where it was made, and the proofs must also control how the later dynamics carry an early error forward. The Euler–Maruyama step is the stochastic Taylor expansion of the solution over one step, truncated after the $g\,\Delta W$ term. For multiplicative noise the first omitted term is Milstein's $\frac{1}{2} g\,\partial_x g\,((\Delta W)^2 - \Delta t)$ (Milstein 1975). It has mean zero, and its typical size is $\frac{1}{2}|g\,\partial_x g|\,\Delta t\;\mathbb{E}|\xi^2 - 1| \approx 0.97 \cdot \frac{1}{2}|g\,\partial_x g|\,\Delta t$. The $N = T/\Delta t$ steps

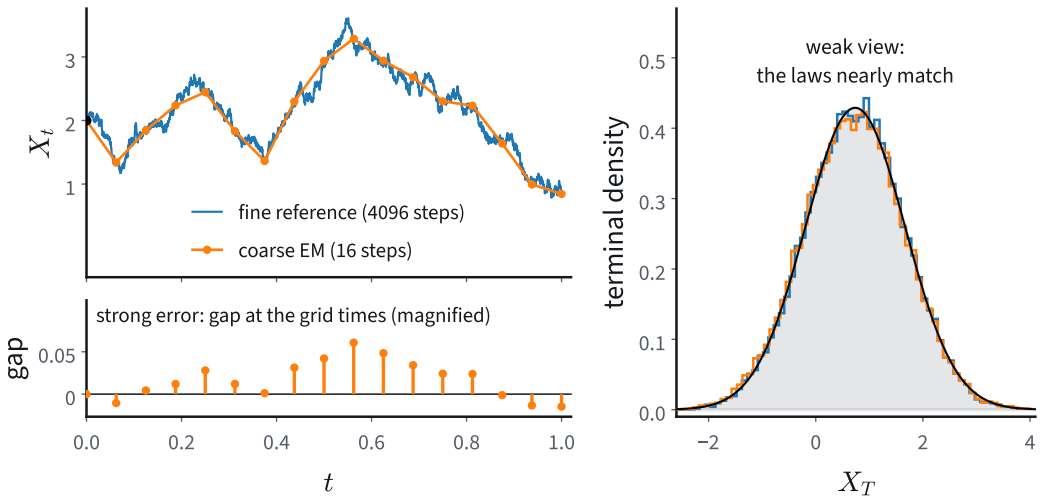


Figure 38.15: Strong versus weak convergence of Euler–Maruyama on the running example $dX = -X\,dt + \sqrt{2}\,dW$ from $X_0 = 2$ on $[0, 1]$. Left: one coarse path (16 steps, orange) against a fine reference (4096 steps, blue) driven by the same Brownian increments; the strip below magnifies their gap at the coarse grid times, whose size is what strong error measures. Right: terminal histograms of 20,000 paths at the two step sizes nearly match each other and the analytic marginal (black), illustrating weak accuracy; at 16 steps the terminal variance is about 4% too large.

contribute independent mean-zero terms of this size, and such terms add like a random walk, to $\sqrt{N}\,\Delta t = \sqrt{T\,\Delta t}$: order $\frac{1}{2}$.

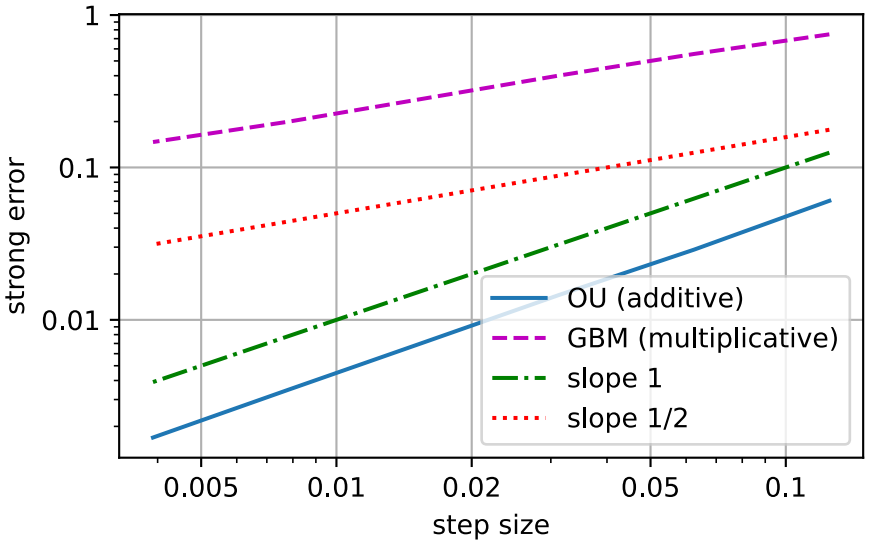
For additive noise $\partial_x g = 0$ removes this term. The next omitted stochastic terms have per-step standard deviation of order $\Delta t^{3/2}$ and add to $\sqrt{N}\,\Delta t^{3/2} = O(\Delta t)$, while the deterministic remainders of order Δt^2 add to $N\,\Delta t^2 = O(\Delta t)$: order 1. For the weak error only means matter. Every omitted stochastic term has mean zero, the bias of one step is of order Δt^2, and N steps give $O(\Delta t)$, whatever the noise.

We measure both strong orders on the same noise: the running example from $X_0 = 2$ against a reference on a 4096-step grid, and geometric Brownian motion ($\mu = \nu = 1, X_0 = 1.5$) against its exact solution Equation 38.31. Coarse increments are sums of consecutive fine ones, so every resolution sees the same Brownian path.

```
T, n_ref, n_em = 1.0, 2**12, 2000
mu_g, nu_g = 1.0, 1.0                       # GBM: multiplicative noise
ou_f, ou_g = lambda x, s: -theta_ou * x, lambda x, s: g_ou
gbm_f, gbm_g = lambda x, s: mu_g * x, lambda x, s: nu_g * x
x0_ou, x0_gbm = np.full(n_em, 2.0), np.full(n_em, 1.5)
dW_f = np.sqrt(T / n_ref) * rng.standard_normal((n_em, n_ref))
t_f = np.linspace(0, T, n_ref + 1)
ou_ref = euler_maruyama(ou_f, ou_g, x0_ou, t_f, dW_f)[:, -1]
gbm_exact = 1.5 * np.exp((mu_g - nu_g**2 / 2) * T + nu_g * dW_f.sum(axis=1))
dts, errs_ou, errs_gbm = [], [], []
for n in [2**k for k in range(3, 9)]:        # 8 to 256 steps
```

```
    dW = dW_f.reshape(n_em, n, n_ref // n).sum(axis=2)
    t = np.linspace(0, T, n + 1)
    ou = euler_maruyama(ou_f, ou_g, x0_ou, t, dW)[:, -1]
    gbm = euler_maruyama(gbm_f, gbm_g, x0_gbm, t, dW)[:, -1]
    dts.append(T / n)
    errs_ou.append(np.abs(ou - ou_ref).mean())
    errs_gbm.append(np.abs(gbm - gbm_exact).mean())
print('strong-order slopes:  OU (additive) '
      f'{np.polyfit(np.log(dts), np.log(errs_ou), 1)[0]:.2f},'
      '  GBM (multiplicative) '
      f'{np.polyfit(np.log(dts), np.log(errs_gbm), 1)[0]:.2f}')
d2l.plot(dts, [errs_ou, errs_gbm, np.array(dts), 0.5 * np.sqrt(dts)],
         'step size', 'strong error', xscale='log', yscale='log',
         legend=['OU (additive)', 'GBM (multiplicative)',
                 'slope 1', 'slope 1/2'], figsize=(5, 3))
```

```
strong-order slopes:  OU (additive) 1.03,  GBM (multiplicative) 0.48
```



The measured slopes are close to 1 for the additive-noise OU process and close to $\frac{1}{2}$ for the multiplicative-noise geometric Brownian motion, the two strong orders of the theorem. The two curves have different heights because the processes have different scales. What can be compared is how they fall: each halving of the step halves the OU error and divides the error of geometric Brownian motion by about $\sqrt{2}$.

For the weak order no sampling is needed. Euler–Maruyama applied to the OU process is a linear Gaussian recursion: if $\hat{X}_n$ is Gaussian, so is $\hat{X}_{n+1}$. So the simulated marginal is Gaussian at every step, with mean and variance

$$m_{n+1} = (1 - \theta\,\Delta t)\, m_n, \qquad v_{n+1} = (1 - \theta\,\Delta t)^2\, v_n + g^2\,\Delta t.$$

The mean is multiplied by $1 - \theta\Delta t$ per step where the exact solution multiplies by $e^{-\theta\Delta t}$ (0.900 against 0.905 at $\Delta t = 0.1$). The variance recursion sums a geometric series, $v_n =$

$g^2\Delta t\,(1-a^{2n})/(1-a^2)$ with $a = 1-\theta\Delta t$ and $v_0 = 0$, whose limit is the fixed point v_∞ found earlier. Evaluating these closed forms at $T = 1$ from $X_0 = 2$ for six step counts gives the errors of the simulated law with no sampling error to blur their slopes.

```
mean_T, var_T = 2 * np.exp(-1), 1 - np.exp(-2)  # exact OU kernel at T = 1
print(f'{"n":>4} {"EM mean":>8} {"EM var":>7}'
      f'   exact {mean_T:.4f}, {var_T:.4f}')
dts, errs = [], []
for n in [10, 20, 40, 80, 160, 320]:
    h = 1 / n
    a_h = 1 - theta_ou * h                  # one-step mean multiplier
    m = 2 * a_h**n                          # EM mean after n steps
    v = g_ou**2 * h * (1 - a_h**(2 * n)) / (1 - a_h**2)   # geometric series
    print(f'{n:4d} {m:8.4f} {v:7.4f}')
    dts.append(h)
    errs.append([abs(m - mean_T), abs(v - var_T)])
s_m, s_v = np.polyfit(np.log(dts), np.log(np.array(errs)), 1)[0]
print(f'weak-order slopes:  mean {s_m:.2f},  variance {s_v:.2f}')
```

```
   n  EM mean  EM var   exact 0.7358, 0.8647
  10   0.6974  0.9247
  20   0.7170  0.8938
  40   0.7265  0.8791
  80   0.7311  0.8718
 160   0.7335  0.8682
 320   0.7346  0.8664
weak-order slopes:  mean 1.01,  variance 1.01
```

At 10 steps the simulated law is $\mathcal{N}(0.697, 0.925)$ against the exact $\mathcal{N}(0.736, 0.865)$; at 320 steps it is $\mathcal{N}(0.735, 0.866)$. Both fitted slopes are 1.01: halving the step halves the error of the marginal, which is weak order 1.

Multiplicative noise halves the strong order but leaves the weak order at 1 (problem 6). A scheme that looks more accurate can also converge to the wrong equation: Heun's method, which averages the noise coefficient over the step, converges to the Stratonovich solution instead of the Itô one (problem 9).

Everything in closed form so far was conditional on the start x_0. For non-Gaussian data the marginal p_t of the whole cloud has no closed form, and generation needs it: to run the process backward, one must know where the density is. The next section derives the equation that p_t obeys (Section 38.4).

38.3.5 Summary

Brownian motion is the limit of the $\pm\sqrt{\Delta t}$ coin-flip walk. It has independent $\mathcal{N}(0, t-s)$ increments, covariance $\min(s,t)$, and quadratic variation $\sum_i (\Delta W_i)^2 \to t$, and its paths are nowhere differentiable. Because $(dW)^2 = dt$, the stochastic integral must fix its evaluation point. Itô's left endpoint gives the zero mean and the isometry $\mathbb{E}[(\int H\,dW)^2] = \int \mathbb{E}[H^2]\,ds$, and the chain rule gains the correction $\frac{1}{2}g^2\phi_{xx}\,dt$. The Ornstein–Uhlenbeck process $dX = -\theta X\,dt + g\,dW$ has the Gaussian kernel $\mathcal{N}(x_0 e^{-\theta t}, \frac{g^2}{2\theta}(1-e^{-2\theta t}))$ and the stationary law $\mathcal{N}(0, g^2/2\theta)$, reached from every start; with $g^2 = 2\theta$, unit-variance data

keep variance 1 at every time. This is the forward process of diffusion models, and the forward step of DDPM is its exact transition. Euler–Maruyama is Euler plus a $\sqrt{\Delta t}$ kick, with strong order 1 for additive noise, $\frac{1}{2}$ for multiplicative noise, and weak order 1 for both.

38.3.6 Exercises

1. **Biased random walk.** A walker takes a step of size $\pm c\sqrt{\Delta t}$ every Δt time units, as in Section 38.3.1.2, but its coin is biased: the step is positive with probability $p = \frac{1}{2}\left(1 + b\,\Delta t^{\delta}\right)$ for constants $b, c, \delta > 0$ and Δt small enough that $p \le 1$. Let $X_t^{(\Delta t)}$ be the position after $k = t/\Delta t$ steps, starting from 0.

    1. Compute $\mathbb{E}\big[X_t^{(\Delta t)}\big]$ and $\mathrm{Var}\big(X_t^{(\Delta t)}\big)$. For which exponent δ does the mean have a finite nonzero limit as $\Delta t \to 0$? What happens to the mean for smaller and for larger δ?
    2. For that exponent, find the limits of the mean and the variance. Which constant drift and diffusion coefficients give an SDE Equation 38.28 whose solution from $X_0 = 0$ has the same mean and variance at every t?
    3. Take $c = 1$ and $\Delta t = 10^{-3}$, and choose b so that the drift in (b) equals 1. Compute p. Compare the drift displacement with the standard deviation of the noise displacement, first over one step and then over the interval $[0, 1]$.

2. **Infinite total variation.** The total variation of a path x along a partition $0 = t_0 < t_1 < \cdots < t_n = t$ is $V_n = \sum_i |x(t_{i+1}) - x(t_i)|$. Pathwise integrals $\int H\,dx$, defined as limits of sums $\sum_i H(s_i)\,\Delta x_i$ with arbitrary evaluation points $s_i \in [t_i, t_{i+1}]$, exist for every continuous H when V_n stays bounded as the mesh shrinks.

    1. Let $x = W$ and use the uniform partition with $\Delta t_i = t/n$. Compute $\mathbb{E}[V_n]$ and $\mathrm{Var}(V_n)$, using $\mathbb{E}|\xi| = \sqrt{2/\pi}$ for $\xi \sim \mathcal{N}(0,1)$, and conclude that $V_n \to \infty$ in probability as $n \to \infty$.
    2. Let x be continuous with $V_n \le V$ for every partition. Show that $\max_i |\Delta x_i| \to 0$ as the mesh shrinks, and deduce $\sum_i (\Delta x_i)^2 \le \max_i |\Delta x_i| \cdot V \to 0$: such a path has quadratic variation zero. Why does Equation 38.24 then give a second proof that Brownian paths have unbounded total variation?
    3. Let x be continuous with bounded total variation. Show that the left-endpoint sums $\sum_i x(t_i)\,\Delta x_i$ and the right-endpoint sums $\sum_i x(t_{i+1})\,\Delta x_i$ both converge to $\frac{1}{2}\left(x(t)^2 - x(0)^2\right)$. Apply the same algebra to $x = W$ to derive Equation 38.30 without Itô's lemma.

3. code **Quadratic variation of an Itô process.** Let X solve Equation 38.28, and let $[X]_t$ be the limit in probability of $\sum_i (X_{t_{i+1}} - X_{t_i})^2$ over partitions of $[0, t]$ whose mesh tends to zero. The computation of $(dX)^2$ in Section 38.3.2.4 suggests $[X]_t = \int_0^t g(X_s, s)^2\,ds$.

    1. Write $X_t = X_0 + A_t + M_t$ with $A_t = \int_0^t f(X_s, s)\,ds$ and $M_t = \int_0^t g(X_s, s)\,dW_s$, and assume that $f(X_s, s)$ is bounded on $[0, t]$. Show that $\sum_i (\Delta A_i)^2$ and $\sum_i \Delta A_i\,\Delta M_i$ tend to zero along every path, so that $[X]_t = [M]_t$. Problem 2b supplies the first limit; the second needs only the continuity of M.
    2. Reshape the increments `dW_qv` of `#sdes-quadratic-variation` to shape $(1, 2^{18})$ and pass them to `euler_maruyama` to simulate the OU process $dX = -X\,dt + \sqrt{2}\,dW$

from $X_0 = 2$ and the geometric Brownian motion $dX = X\,dt + X\,dW$ from $X_0 = 1.5$ on $[0, 1]$. For each path, print the sum of squared increments on the subsampled grids of that cell next to $\int_0^1 g(X_s, s)^2\,ds$ evaluated along the same path. For which of the two processes is $[X]_1$ random?

3. Let $X_t = \mu t + gW_t$ with constants μ and g be observed at $n + 1$ equally spaced times on $[0, T]$. Show that $\frac{1}{T}\sum_i (\Delta X_i)^2 \to g^2$ in mean square as $n \to \infty$ for every fixed T, whereas the estimator X_T/T of μ has variance g^2/T for every n. What does this imply for estimating the diffusion coefficient and the drift from one finely sampled trajectory?

4. **Geometric Brownian motion and the $-\nu^2/2$ correction.** Consider $dX = \mu X\,dt + \nu X\,dW$ with $X_0 > 0$ and its solution, given by Equation 38.31.

   1. Using $\mathbb{E}[e^{\lambda Z}] = e^{\lambda^2 v/2}$ for $Z \sim \mathcal{N}(0, v)$, show that $\mathbb{E}[X_t] = X_0 e^{\mu t}$ and $\mathbb{E}[X_t^2] = X_0^2 e^{(2\mu+\nu^2)t}$. Find the median of X_t.
   2. Take $\mu = 0.1$ and $\nu = 0.6$. What happens to the mean, to the median, and to a typical path as $t \to \infty$? How can these statements hold at the same time?
   3. For which ν, at a fixed $\mu > 0$, does the median stop growing? Check the condition for the parameters of (b) on 100 paths simulated with `euler_maruyama` to $t = 20$ at $\Delta t = 10^{-2}$: what fraction of the paths ends below X_0, and what fraction does the exact solution Equation 38.31 predict?

   Adapted from Simo Särkkä and Arno Solin, Applied Stochastic Differential Equations, Exercise 4.3.

5. code **Strong order and Milstein's correction.** As Section 38.3.4.2 explains, Euler–Maruyama omits the Milstein term $\frac{1}{2} g\,\partial_x g\,((\Delta W)^2 - \Delta t)$, and its strong order is $\frac{1}{2}$ unless that term vanishes.

   1. By Equation 38.31, $Y = \log X$ satisfies an SDE with additive noise. Apply `euler_maruyama` to that SDE with the increments `dW_f` and the step sizes of `#sdes-em-strong-order`, exponentiate the endpoints, and measure the strong error against `gbm_exact`. Explain the result.
   2. Use Itô's lemma to show that for $dX = f(X)\,dt + g(X)\,dW$ with $g > 0$, the variable $Y = \psi(X)$ with $\psi' = 1/g$ has unit additive noise, and find its drift.
   3. Milstein's scheme (Milstein 1975) adds the first omitted term of the stochastic Taylor expansion to the Euler–Maruyama step,

$$\hat{X}_{n+1} = \hat{X}_n + f\,\Delta t + g\,\Delta W_n + \tfrac{1}{2}\,g\,\partial_x g\,((\Delta W_n)^2 - \Delta t)\,, \tag{38.41}$$

   with f, g, and $\partial_x g$ evaluated at $(\hat{X}_n, t_n)$. Implement it for the geometric Brownian motion of `#sdes-em-strong-order`, and measure its strong-order slope on the same increments and step sizes. By what factor does it reduce the strong error at $\Delta t = 2^{-8}$?

6. code **Weak order under multiplicative noise.** For geometric Brownian motion, the Euler–Maruyama step multiplies $\hat{X}_n$ by $1 + \mu\,\Delta t + \nu\,\Delta W_n$. Its strong order is $\frac{1}{2}$, as `#sdes-em-strong-order` measures.

1. Derive recursions for the mean and the second moment of the simulated $\hat{X}_n$, and solve them in closed form.
2. As `#sdes-em-weak-order` does for the OU process, compute the errors of both moments at $T = 1$ against the exact moments of problem 4a, for $\mu = \nu = 1$, $X_0 = 1.5$, and $n \in \{20, 40, 80, 160, 320, 640\}$ steps, and fit the slopes. How do they compare with the strong-order slope of the same scheme?

7. code **Covariance of the OU process.** Let X be the OU process Equation 38.33.
   1. For $s \le t$, split the integral in Equation 38.35 at time s and show that $\mathrm{Cov}(X_s, X_t \mid X_0) = e^{-\theta(t-s)}\mathrm{Var}(X_s \mid X_0)$. Write the result in closed form with Equation 38.34.
   2. Let $X_0 \sim \mathcal{N}(0, g^2/2\theta)$ be independent of W. Show that $\mathrm{Cov}(X_s, X_t) = \frac{g^2}{2\theta}\, e^{-\theta|t-s|}$ for all $s, t \ge 0$.
   3. Estimate $\mathrm{Cov}(X_1, X_t)$ for $t \in \{1, 1.5, 2, 4\}$ from the saved clouds `X_ou_snap` of `#sdes-ou-cloud`, and compare the estimates with the formulas of (a) and (b). Which formula does the cloud follow, and why?
   4. A sampler stores the state of a stationary OU process every Δ time units. How large must Δ be for the correlation between consecutive stored states to fall below 0.01?

8. **Variance preservation, exactly.** Let X_t follow Equation 38.37 with zero-mean data X_0.
   1. Assume unit-variance data and show that $\mathbb{E}[X_t^4] = 3 + \bar{\alpha}_t^2\,(\mathbb{E}[X_0^4] - 3)$. Specialize to Rademacher data $X_0 = \pm 1$ to obtain the prediction $3 - 2\bar{\alpha}_t^2$ that `#sdes-vp-normalization` prints.
   2. The excess kurtosis $\mathbb{E}[X^4]/\mathrm{Var}(X)^2 - 3$ is zero for every Gaussian. At what rate in t does it decay for X_t? Compare this rate with the rate at which the conditional mean $\mathbb{E}[X_t \mid X_0]$ forgets the data.
   3. Suppose instead that the data have variance s^2. Find $\mathrm{Var}(X_t)$. For $s = 2$ and $\theta = 1$, how long must the process run before $\mathrm{Var}(X_T)$ is within 1% of the variance of the reference $\mathcal{N}(0, 1)$?

9. code **Itô versus Stratonovich.** ⊠ By problem 2c, the average of the left- and right-endpoint sums for $\int_0^t W\, dW$ equals $\frac{1}{2}W_t^2$ for every partition. Sums of this kind define the Stratonovich integral (Stratonovich 1966), which obeys the ordinary chain rule.
   1. Heun's method of Section 38.1.3 carries over to SDEs if the noise term is treated like a second slope. The predictor is the Euler–Maruyama step $\tilde{X} = \hat{X}_n + f(\hat{X}_n)\,\Delta t + g(\hat{X}_n)\,\Delta W_n$, and the corrector averages both coefficients:

$$\hat{X}_{n+1} = \hat{X}_n + \tfrac{1}{2}\left[f(\hat{X}_n) + f(\tilde{X})\right]\Delta t + \tfrac{1}{2}\left[g(\hat{X}_n) + g(\tilde{X})\right]\Delta W_n. \qquad (38.42)$$

      Implement it for the geometric Brownian motion of `#sdes-em-strong-order`, on that cell's increments `dW_f` and step sizes. Compare its endpoints with the exact Itô solution Equation 38.31 and with $X_0 \exp(\mu T + \nu W_T)$. To which of the two does the scheme converge?

   2. Expand the corrector's noise term to first order in $\tilde{X} - \hat{X}_n$, and use $(\Delta W_n)^2 \approx$

Δt to show that the scheme behaves like Euler–Maruyama for an Itô SDE with a modified drift. Find that drift and check that it explains the limit in (a).

3. Why do the two conventions give the same solution for every SDE of the form Equation 38.22?

10. **Exact transitions of linear SDEs.** ▣ Consider the linear SDE $dX = -\theta(t)X\,dt + g(t)\,dW$ with continuous θ and g, and let $\Theta(t) = \int_0^t \theta(u)\,du$.

    1. Apply Itô's lemma to $\phi(x,t) = e^{\Theta(t)}x$ and show that, given X_s, the state X_t at a later time t is Gaussian with mean $e^{-(\Theta(t)-\Theta(s))}X_s$ and variance $\int_s^t e^{-2(\Theta(t)-\Theta(u))}\,g(u)^2\,du$.
    2. Specialize (a) to the VP-SDE Equation 38.38 and show that a transition of any length maps a unit-variance state to a unit-variance state. Conversely, assume for the linear SDE of (a) that $\mathrm{Var}(X_t) = 1$ for every $t \geq 0$ whenever $\mathrm{Var}(X_0) = 1$, and show that $g(t)^2 = 2\,\theta(t)$ at every t.
    3. Consider the semilinear SDE $dX = (-\theta X + q(X))\,dt + g\,dW$ with constants $\theta, g > 0$ and a smooth function q. The exponential Euler–Maruyama step holds q at the value $q(\hat{X}_n)$ over a step of length Δt and solves the remaining linear SDE exactly, by the method of (a). Write the step. Show that it agrees with the Euler–Maruyama step Equation 38.39 to first order in Δt and that it is exact when q is constant. For $q \equiv 0$, Euler–Maruyama damps the mean only for $\Delta t < 2/\theta$ (Section 38.1.3.3). What limits the step size of the exponential step?
    4. Exact steps have zero weak error at every step size. Which property of the drift and the diffusion of the VP-SDE makes them available? The generative sampler of a diffusion model simulates, backward in time, an SDE whose drift is $-\frac{1}{2}\beta(t)\,x - \beta(t)\,\nabla \log p_t(x)$: the VP drift plus a multiple of the score of the forward marginals, which is a nonlinear function of x for non-Gaussian data. Why can its steps not be made exact? Exponential integrators for diffusion samplers (Zhang and Chen 2023) apply the step of (c) to this drift. Which part of the drift does that step treat exactly, and which part does it approximate?

11. [extended] **Multilevel Monte Carlo.** ▣▣ Estimating $\mathbb{E}[\varphi(X_T)]$ by averaging over Euler–Maruyama paths has two sources of error, the weak bias of the scheme and the sampling noise. Multilevel Monte Carlo (Michael B. Giles 2008) reduces the cost by coupling paths across step sizes, which is one of the uses of strong accuracy named in Section 38.3.4.2. Throughout, the mean-square strong error $\mathbb{E}\big[(\hat{X}_N - X_T)^2\big]^{1/2}$ has the same orders as the strong error of that section.

    1. Plain Monte Carlo averages $\varphi(\hat{X}_N)$ over M independent paths at a cost of $MT/\Delta t$ steps. Assume a weak bias $c_1\,\Delta t$ and a variance v of $\varphi(\hat{X}_N)$. Show that a mean squared error below ε^2 costs order ε^{-3} steps at the best choice of M and Δt.
    2. Let P_l be φ of the Euler–Maruyama endpoint with step $h_l = T\,2^{-l}$, and write $\mathbb{E}[P_L] = \mathbb{E}[P_0] + \sum_{l=1}^{L} \mathbb{E}[P_l - P_{l-1}]$. Each term is estimated from M_l independent pairs in which the coarse path uses the sums of consecutive pairs of fine increments, as in `#sdes-em-strong-order`. For Lipschitz φ, bound $V_l = \mathrm{Var}(P_l - P_{l-1})$ by the mean-square strong error. Take $\varphi(x) = x$, and predict how V_l scales with h_l for the OU process and the geometric Brownian motion of `#sdes-em-strong-order`, with that cell's parameters. Measure V_l for $l = 1, \dots, 8$

with `euler_maruyama`, and fit the slopes on the finer levels.

3. Minimize the total cost $\sum_l M_l/h_l$ subject to $\sum_l V_l/M_l \le \varepsilon^2/2$ with a Lagrange multiplier. Choose the finest level L so that the bias stays below $\varepsilon/\sqrt{2}$. Assuming $V_l \propto h_l^r$, show that the cost is of order ε^{-2} for $r = 2$ and of order $\varepsilon^{-2}(\log \varepsilon)^2$ for $r = 1$.
4. Implement both estimators for $\mathbb{E}[X_T] = X_0 e^{\mu T}$ of the geometric Brownian motion of `#sdes-em-strong-order`, with M_l from
   (c) and your measured V_l. For $\varepsilon \in \{0.2, 0.1, 0.05, 0.02\}$, record the total number of steps each estimator uses and check its error over a few repetitions. Plot ε^2 times the cost against ε. Repeat with the Milstein scheme of problem 5c in place of Euler–Maruyama. `euler_maruyama` stores every path in full, so simulate large M_l in batches.
5. Repeat the measurement of (b) for the discontinuous $\varphi(x) = \mathbf{1}(x > X_0)$. Why does V_l decay more slowly, and what does the slower decay do to the cost?

12. code **Brownian scaling and exit times.**

1. For $c > 0$ let $\tilde{W}_t = W_{ct}/\sqrt{c}$. Check properties 1–4 of Section 38.3.1.2 and conclude that $\tilde{W}$ is a Brownian motion. Which property of the $\pm\sqrt{\Delta t}$ walk does this make exact?
2. Let τ_a be the first time $|W_t|$ reaches a level $a > 0$. Use (a) to show that τ_a has the same distribution as $a^2\tau_1$, so that $\mathbb{E}[\tau_a] = a^2\,\mathbb{E}[\tau_1]$.
3. Granting $\mathbb{E}[\tau_1] = 1$ (it follows from $\mathbb{E}[W_\tau^2] = \mathbb{E}[\tau]$, a consequence of Equation 38.30 and the zero mean of the Itô integral up to a stopping time), simulate $4{,}000$ walks with Gaussian steps of variance $\Delta t = 10^{-3}$ and estimate $\mathbb{E}[\tau_a]$ for $a = 0.5, 1, 2$, with Monte Carlo standard errors. The estimates tend to run slightly above a^2, most clearly for small a. Explain the sign of the bias from the way a discrete walk crosses a level, and say how it scales with a and Δt.

38.4 The Fokker–Planck Equation and Probability Flow

The previous section simulated the Ornstein–Uhlenbeck process, the stochastic differential equation $dX = -X\,dt + \sqrt{2}\,dW$ that pulls a particle toward the origin while kicking it with noise. Every run drew different paths, and no single path can be predicted. Yet something repeats from run to run. Start a hundred thousand particles at $x = 2$ and look at their histogram at time $t = 0.5$: on every run it is the same bell curve, centered near 1.21 with a standard deviation near 0.80. The randomness of the paths averages out, and the density of the cloud moves deterministically. Which equation moves it?

Adriaan Fokker and Max Planck wrote that equation for Brownian particles pushed by a force (Fokker 1914; Planck 1917), and Andrey Kolmogorov showed that it governs the density of every diffusion (Kolmogorov 1931). It has two terms: the drift carries the density along, and the noise smooths it. The smoothing can also be written as transport, a flow that moves probability from where the density is high toward where it is low. Seen at the level of the density, noise acts like a deterministic velocity.

This matters for generative models. A diffusion model generates data by running a noising process backward. No map takes the endpoint of a noisy path back to its start, since many starts lead to the same endpoint. But we know how the density of the whole cloud changed, and we can build processes whose density changes the other way. One is an ordinary dif-

ferential equation, obtained by replacing the noise with the outward flow; its particles have the same histogram as the stochastic process at every time. The other is again a stochastic process, and it turns noise into data. We choose the drift and the noise level ourselves. Both constructions need one more quantity: the direction in which the log-density increases, at every point and time.

We check each step on the Ornstein–Uhlenbeck process and on a two-mode "data" density whose log-density gradient has a closed form. For real data that gradient is unknown, and this is the question we leave open. Three results carry the section: the Fokker–Planck equation Equation 38.49, the identity Equation 38.60 that turns its smoothing term into transport, and the family of reverse processes Equation 38.68. A reader at home with the divergence theorem can skip Section 38.4.2.1; the likelihood formula and the bias computation at the end can wait for a second reading.

```
from scipy.special import erf
```

38.4.1 From Paths to Densities

38.4.1.1 Pathwise and Ensemble Views

Every SDE in this section is

$$d\mathbf{X} = \mathbf{f}(\mathbf{X}, t)\, dt + g(t)\, d\mathbf{W},$$

with a drift $\mathbf{f} : \mathbb{R}^d \times [0, T] \to \mathbb{R}^d$, a scalar noise level $g(t)$, a d-dimensional Brownian motion $\mathbf{W}$, and a random start $\mathbf{X}_0 \sim p_0$. Over a short step the particle moves by $\mathbf{f}\, dt$ plus an independent Gaussian kick of variance $g^2\, dt$ in each coordinate.

One realization of the Brownian motion produces one path $\mathbf{X}_t$, different on every run. The object of this section is instead the density $p_t(\mathbf{x})$ of the random variable $\mathbf{X}_t$, called the **time-marginal**. It averages over both sources of randomness, the draw of $\mathbf{X}_0$ and the noise along the way, and this averaging makes it deterministic.

Our running process is the Ornstein–Uhlenbeck (OU) process of Section 38.3.3, $dX = -\theta X\, dt + g\, dW$. Started from a point $X_0 = x_0$, its marginal is the Gaussian kernel Equation 38.34, which we write as

$$X_t \sim \mathcal{N}(\alpha_t x_0,\ \sigma_t^2), \qquad \alpha_t = e^{-\theta t}, \quad \sigma_t^2 = \frac{g^2}{2\theta}\left(1 - e^{-2\theta t}\right). \tag{38.43}$$

We take $\theta = 1$ and $g = \sqrt{2}$ throughout, so the variance saturates at $g^2/(2\theta) = 1$ and the stationary law is $\mathcal{N}(0, 1)$. The drift moves the center of the cloud and the noise widens it; the Fokker–Planck equation will turn both actions into operators on p_t.

38.4.1.2 Simulated Marginals

The SDE section compared the moments of such a cloud with the kernel. Here we look at the whole histogram: $100{,}000$ particles start at $x_0 = 2$ and move by Euler–Maruyama steps of size $\Delta t = 0.002$ (Section 38.3.4.1), and at $t = 0.1$, 0.5 and 2 the cloud is drawn against the Gaussian Equation 38.43.

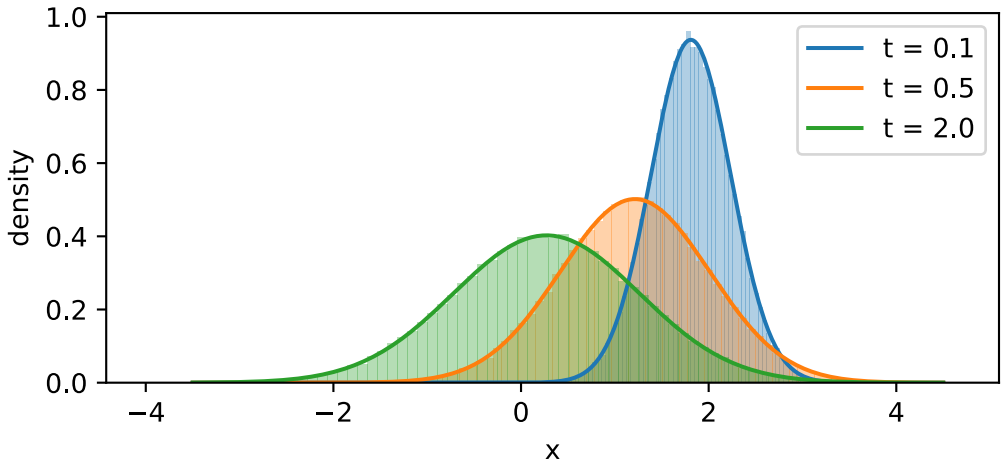
```
theta, g = 1.0, np.sqrt(2.0)            # OU: dX = -theta X dt + g dW

def ou_mean_var(x0, t, theta=theta, g=g):
    """Mean and variance of X_t started at x0: N(alpha_t x0, sigma_t^2)."""
    alpha = np.exp(-theta * t)
    return alpha * x0, g**2 / (2 * theta) * (1 - alpha**2)

rng = np.random.default_rng()
n, dt, snap_at = 100_000, 0.002, {50: 0.1, 250: 0.5, 1000: 2.0}  # step -> t
x, snaps = np.full(n, 2.0), {}          # every particle starts at x0 = 2
for k in range(1, max(snap_at) + 1):    # Euler--Maruyama
    x += -theta * x * dt + g * np.sqrt(dt) * rng.standard_normal(n)
    if k in snap_at:
        snaps[snap_at[k]] = x.copy()

grid = np.linspace(-3.5, 4.5, 400)
d2l.set_figsize((6, 2.5))
for (t, cloud), c in zip(snaps.items(), ('C0', 'C1', 'C2')):
    m, v = ou_mean_var(2.0, t)
    print(f't={t}: mean {cloud.mean():.3f} (exact {m:.3f}), '
          f'std {cloud.std():.3f} (exact {np.sqrt(v):.3f})')
    d2l.plt.hist(cloud, bins=80, density=True, alpha=0.35, color=c)
    d2l.plt.plot(grid, np.exp(-(grid - m)**2 / (2 * v)) / np.sqrt(2 * np.pi * v),
                 color=c, label=f't = {t}')
d2l.plt.xlabel('x'); d2l.plt.ylabel('density'); d2l.plt.legend();
```

```
t=0.1: mean 1.810 (exact 1.810), std 0.425 (exact 0.426)
t=0.5: mean 1.211 (exact 1.213), std 0.794 (exact 0.795)
t=2.0: mean 0.270 (exact 0.271), std 0.990 (exact 0.991)
```



At every snapshot the cloud's mean and standard deviation agree with $2e^{-t}$ and $\sqrt{1-e^{-2t}}$ to a few thousandths, the size of the Monte Carlo error (for the mean at $t = 0.5$ it is $0.80/\sqrt{10^5} \approx 0.0025$). At $t = 2$ the mean is still $2e^{-2} \approx 0.27$, about a quarter of a standard deviation away from the stationary $\mathcal{N}(0, 1)$. This is one run, but the three curves it matches contain no randomness: a second run would differ in every path and match the

same curves. We now look for the equation that moves them.

38.4.2 The Fokker–Planck Equation

A path cannot be differentiated in time, but the average of a smooth function of $\mathbf{X}_t$ over all paths can. Turning a statement about averages into a statement about the density requires moving derivatives from the function onto the density, and that needs three facts about fields in $\mathbb{R}^d$.

38.4.2.1 Divergence, Laplacian, and Integration by Parts

The derivation uses only the gradient of Section 34.2 and three tools built from it, each of which also has a physical meaning we will need later. Let $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$ be a smooth vector field, which we picture as the velocity of a fluid, and $h : \mathbb{R}^d \to \mathbb{R}$ a smooth scalar field.

Divergence. We want to know how much fluid leaves a small box per unit time. The **divergence** of $\mathbf{u}$ is the scalar field

$$\nabla \cdot \mathbf{u} = \sum_{i=1}^{d} \frac{\partial u_i}{\partial x_i}. \tag{38.44}$$

It is the net rate at which fluid leaves an infinitesimal box around $\mathbf{x}$, divided by the volume of the box: positive at a source, negative at a sink. Two plane fields show the range. The expanding field $\mathbf{u} = (x_1, x_2)$ has $\nabla \cdot \mathbf{u} = 1 + 1 = 2$, so every box of area A loses fluid at the rate $2A$, wherever it sits. The rotation $\mathbf{u} = (-x_2, x_1)$ has $\nabla \cdot \mathbf{u} = 0 + 0 = 0$: fluid circulates, and no box gains or loses any. A zero divergence therefore does not mean a zero field. Figure 38.16 checks both statements on one box.

Summing the outflow of many small boxes cancels every interior face, which leaves the flux through the outer boundary. This is the **divergence theorem**: for a bounded region Ω with outward unit normal $\mathbf{n}$,

$$\int_{\Omega} \nabla \cdot \mathbf{u} \, d\mathbf{x} = \oint_{\partial\Omega} \mathbf{u} \cdot \mathbf{n} \, dS. \tag{38.45}$$

The total divergence inside a region equals the net flow out through its boundary. In one dimension it is the fundamental theorem of calculus, $\int_a^b u'(x)\,dx = u(b) - u(a)$. We use the theorem without proof; it is proved in any text on multivariable calculus.

Laplacian. We want a measure of how a function compares with its surroundings. The **Laplacian** is the divergence of the gradient (in front of a function, Δ is this operator and not an increment as in Δt),

$$\Delta h = \nabla \cdot (\nabla h) = \sum_{i=1}^{d} \frac{\partial^2 h}{\partial x_i^2}. \tag{38.46}$$

It compares h at a point with its average over a small sphere of radius r around the point: that average is $h(\mathbf{x}) + \frac{r^2}{2d}\Delta h(\mathbf{x}) + O(r^4)$, which Exercise 1 proves. In one dimension, with

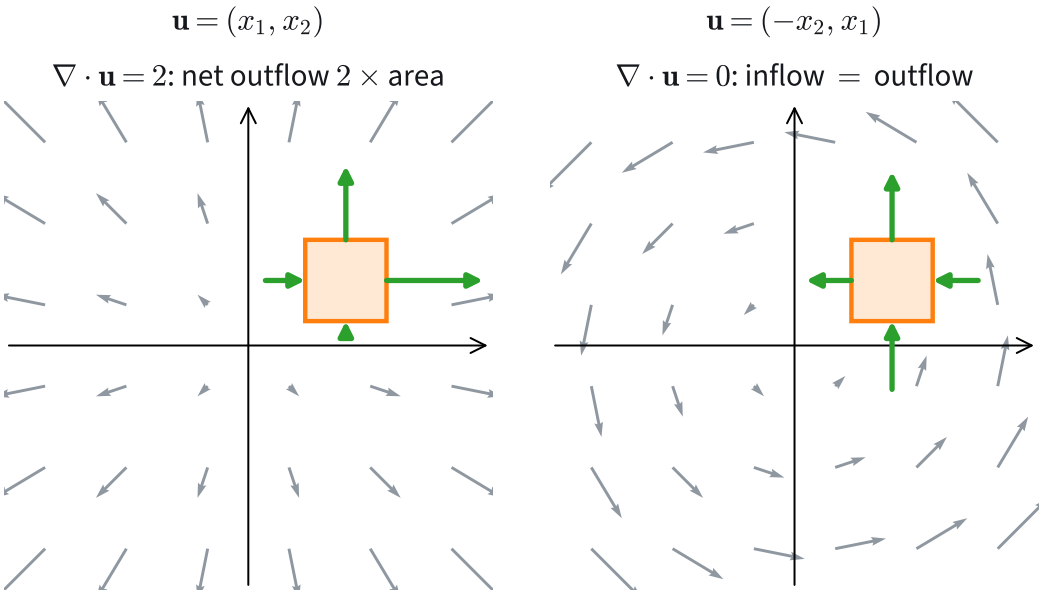


Figure 38.16: The divergence as net outflow, on the box of side 0.5 centered at $(0.6, 0.4)$. Each green arrow is the normal component of the field at the midpoint of a side, drawn away from the side for outflow and onto it for inflow. (a) For $\mathbf{u} = (x_1, x_2)$ fluid enters through the left and bottom sides and leaves through the right and top; the outflow exceeds the inflow by $2\times$ the area. (b) For the rotation $\mathbf{u} = (-x_2, x_1)$ the inflow through the right and bottom sides equals the outflow through the other two.

$h(x) = x^2$, the "sphere" is the pair of points $x \pm r$, the average is $\frac{1}{2}\big((x-r)^2 + (x+r)^2\big) = x^2 + r^2$, and $\frac{r^2}{2} h'' = r^2$, so the formula holds with no remainder (at $x = 1$, $r = 0.3$ the average is 1.09). So $\Delta h > 0$ means that h lies below its neighborhood average. A zero Laplacian does not mean a constant function: $h = x_1^2 - x_2^2$ has $\Delta h = 2 - 2 = 0$ and equals its average over every circle.

Integration by parts. This is the d-dimensional form of the one-dimensional rule of Section 34.6.2.3, and it is the tool that moves a derivative from one factor onto the other. If h and $\mathbf{u}$ are smooth and $h\,\mathbf{u}$ decays fast enough at infinity (in this section, because h vanishes outside a bounded set or because both factors have Gaussian tails), then

$$\int h\,(\nabla \cdot \mathbf{u})\, d\mathbf{x} = -\int \nabla h \cdot \mathbf{u}\, d\mathbf{x}. \tag{38.47}$$

The derivative moves from $\mathbf{u}$ onto h, and the sign flips.

Proof. The product rule gives $\nabla \cdot (h\mathbf{u}) = \nabla h \cdot \mathbf{u} + h\,\nabla \cdot \mathbf{u}$. Integrate both sides over a ball of radius R and let $R \to \infty$. By the divergence theorem the left side is the flux of $h\mathbf{u}$ through the sphere of radius R, which tends to zero under the decay assumption. ■

Applied twice, first with $\mathbf{u} = \nabla p$ and then with the roles exchanged, it moves a whole Laplacian across the integral sign, and the two signs cancel: $\int h\,\Delta p = -\int \nabla h \cdot \nabla p = \int p\,\Delta h$.

38.4.2.2 From Itô's Lemma to the PDE

We now derive the equation for p_t. Throughout we assume that $\mathbf{f}$ is continuously differentiable in $\mathbf{x}$ and continuous in t, that g is continuous, that p_t is smooth and positive, and that derivatives in t may be taken under the integral sign.

The idea is to look at the cloud through averages. If ϕ is a smooth bump equal to 1 on a small box and 0 outside a slightly larger one, then $\mathbb{E}[\phi(\mathbf{X}_t)]$ is roughly the probability of the box. Knowing how every such average changes over time determines how the density changes. So take any smooth **test function** $\phi : \mathbb{R}^d \to \mathbb{R}$ that vanishes outside a bounded set.

Itô's lemma (Section 38.3.2.4) is the chain rule for SDE paths. For $d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}\,d\mathbf{W}$ it adds to the ordinary chain rule the second-order term $\frac{1}{2}\sum_{ij} (\mathbf{G}\mathbf{G}^{\top})_{ij}\,\partial_i \partial_j \phi$, because the squared Brownian increment $(dW)^2$ is of size dt rather than negligible. With $\mathbf{G} = g(t)\,\mathbf{I}$ that term is $\frac{1}{2} g^2$ times the Laplacian of ϕ:

$$d\phi(\mathbf{X}_t) = \big(\nabla\phi \cdot \mathbf{f} + \tfrac{1}{2} g^2 \Delta\phi\big)\, dt + g\,\nabla\phi \cdot d\mathbf{W}.$$

The function changes by a drift part, the chain rule plus the Laplacian correction, and a noise part. Now take expectations. The Itô integral of $g\nabla\phi$ has mean zero, because each Brownian increment is independent of the integrand at the start of that increment (Section 38.3.2.2). The noise part drops out:

$$\frac{d}{dt}\,\mathbb{E}[\phi(\mathbf{X}_t)] = \mathbb{E}\big[\nabla\phi \cdot \mathbf{f} + \tfrac{1}{2} g^2 \Delta\phi\big]\,. \tag{38.48}$$

The average of ϕ changes at the average rate of its drift-plus-diffusion derivative. Both sides are integrals against the marginal density. Writing $\mathbb{E}[\,\cdot\,] = \int (\cdot)\, p_t(\mathbf{x})\, d\mathbf{x}$ and moving d/dt under the integral,

$$\int \phi\ \partial_t p_t\ d\mathbf{x} = \int (\nabla\phi \cdot \mathbf{f})\, p_t\ d\mathbf{x} + \tfrac{1}{2} g^2 \int (\Delta\phi)\, p_t\ d\mathbf{x}.$$

The derivatives on the right still act on ϕ; we move them onto p_t with integration by parts Equation 38.47. Because ϕ vanishes outside a bounded set, no boundary terms appear and no decay of p_t is needed. The drift term needs one move, with $h = \phi$ and $\mathbf{u} = \mathbf{f} p_t$; the diffusion term needs two, and its signs cancel:

$$\int (\nabla\phi \cdot \mathbf{f})\, p_t\, d\mathbf{x} = -\int \phi\, \nabla \cdot (\mathbf{f} p_t)\, d\mathbf{x}, \qquad \int (\Delta\phi)\, p_t\, d\mathbf{x} = \int \phi\, \Delta p_t\, d\mathbf{x}.$$

Substituting both gives one integral against ϕ:

$$\int \phi \left(\partial_t p_t + \nabla \cdot (\mathbf{f} p_t) - \tfrac{1}{2} g^2 \Delta p_t \right) d\mathbf{x} = 0.$$

This holds for every test function ϕ. The bracket is continuous, and a continuous function that integrates to zero against every bump is zero (a bump placed where it is positive would give a positive integral). So the bracket vanishes pointwise, and p_t solves the **Fokker–Planck equation**, also called the *Kolmogorov forward equation*:

$$\partial_t p_t(\mathbf{x}) = -\nabla \cdot \big(\mathbf{f}(\mathbf{x}, t)\, p_t(\mathbf{x})\big) + \tfrac{1}{2} g(t)^2\, \Delta p_t(\mathbf{x}). \tag{38.49}$$

The first term is the drift's contribution, the second the noise's. For a matrix noise $\mathbf{G}\, d\mathbf{W}$ the second term becomes $\frac{1}{2} \sum_{ij} \partial_i \partial_j (a_{ij} p_t)$ with $\mathbf{a} = \mathbf{G}\mathbf{G}^\top$; Exercise 7 works out what changes. Risken (1996) collects the many variants of the equation used in physics.

The derivation gives one direction only: the marginal of the SDE solves the equation. Later we need the converse, that a solution which starts from p_0 *is* the marginal, and for this the equation must have only one solution. It does under mild conditions, which every drift we use satisfies ($g > 0$ and a drift that is Lipschitz in $\mathbf{x}$ uniformly in t): among probability densities with a given initial law the solution is then unique (Bogachev et al. (2015) give precise statements). We use this uniqueness without proof, and say so where we do.

38.4.2.3 The Heat Equation and the Two Terms

Switch the drift off. The Fokker–Planck equation is then the **heat equation**,

$$\mathbf{f} = \mathbf{0}: \qquad \partial_t p_t = \tfrac{1}{2} g^2\, \Delta p_t. \tag{38.50}$$

The density rises wherever it lies below its neighborhood average. Before any general statement, we check it at one point. Take $g = 1$, $d = 1$ and $p_t = \mathcal{N}(0, t)$, the law of Brownian motion started at 0. With $p = e^{-x^2/(2t)}/\sqrt{2\pi t}$, direct differentiation gives

$\partial_t p = p\,(\frac{x^2}{2t^2} - \frac{1}{2t})$ and $\partial_{xx} p = p\,(\frac{x^2}{t^2} - \frac{1}{t})$. At $x = 0$, $t = 1$: $p = 0.399$, $\partial_t p = -0.1995$ and $\frac{1}{2}\partial_{xx} p = -0.1995$. The peak sinks at the rate the Laplacian predicts.

More generally, a Gaussian $\mathcal{N}(\mathbf{m}, v_t \mathbf{I})$ solves the heat equation when its variance grows as $v_t = v_0 + g^2 t$ for constant g, or as $v_t = v_0 + \int_0^t g(\tau)^2\, d\tau$ when g changes with time. This is the $\mathrm{Var}(W_t) = t$ law of Brownian motion. The "variance-exploding" forward process of diffusion models is this equation with a growing $g(t)$.

Now read the two terms of Equation 38.49. The first, $-\nabla \cdot (\mathbf{f} p)$, is *advection*: it removes density wherever the drift carries more mass out of a small box than in. The second, $\frac{1}{2} g^2 \Delta p$, is *diffusion*: it raises the density where it lies below its neighborhood average and lowers it where it lies above. It is Itô's $(dW)^2 = dt$ correction, seen at the level of the density.

38.4.2.4 Gaussian Solutions of the OU Equation

For the OU process we know the answer in closed form, which lets us test the equation on a whole family of solutions. With $f(x) = -\theta x$ in one dimension, Equation 38.49 reads

$$\partial_t p = \theta\, \partial_x (x\, p) + \tfrac{1}{2} g^2\, \partial_{xx} p.$$

Substituting a general Gaussian reduces this PDE to two ODEs.

Proposition (Gaussian solutions of the OU equation). *Let $p(x,t) = \mathcal{N}(x; m(t), v(t))$ be a Gaussian density with differentiable mean $m(t)$ and variance $v(t) > 0$. Then p solves the OU Fokker–Planck equation if and only if*

$$\dot m = -\theta m, \qquad \dot v = g^2 - 2\theta v. \tag{38.51}$$

Proof. For a Gaussian, $\partial_x p = -\frac{x-m}{v}\, p$. Direct differentiation turns each side of the PDE into p times a quadratic polynomial in $(x - m)$:

$$\partial_t p = p\left[\frac{\dot m}{v}(x-m) + \frac{\dot v}{2v^2}\big((x-m)^2 - v\big)\right],$$

while, writing $x = (x-m) + m$ in the drift term,

$$\theta\, \partial_x (xp) + \tfrac{g^2}{2}\partial_{xx} p = p\left[-\frac{\theta m}{v}(x-m) + \Big(\frac{g^2}{2v^2} - \frac{\theta}{v}\Big)\big((x-m)^2 - v\big)\right].$$

The functions $1, (x-m), (x-m)^2$ are linearly independent, so the two sides agree for all x only when the coefficients match: $\dot m = -\theta m$ from the linear term and $\dot v = g^2 - 2\theta v$ from the quadratic term (the constant term repeats the latter). ■

The mean decays and the variance relaxes toward $g^2/(2\theta)$. Solving the two ODEs from $m(0) = m_0$, $v(0) = v_0$ gives

$$m(t) = m_0 e^{-\theta t}, \qquad v(t) = v_0 e^{-2\theta t} + \frac{g^2}{2\theta}(1 - e^{-2\theta t}).$$

For a point start, $v_0 = 0$, this is the kernel Equation 38.43 at every $t > 0$, so the transition density of Section 38.3.3 solves the Fokker–Planck equation.

The fixed point $\dot{m} = \dot{v} = 0$ is $m = 0$, $v = g^2/(2\theta)$: the stationary Gaussian is the steady state $\partial_t p = 0$ of the PDE (Exercise 3 derives it without assuming a Gaussian form). Setting $\theta = 0$ recovers the heat equation's $\dot{v} = g^2$.

38.4.2.5 Linearity and Mixtures

Real data are not Gaussian, so we need a density with more structure that we can still follow in closed form. A Gaussian mixture is one. Our running "data" density is

$$p_0 = \tfrac{1}{2}\,\mathcal{N}(-2,\,0.25) + \tfrac{1}{2}\,\mathcal{N}(2,\,0.25),$$

two well-separated modes of standard deviation 0.5. A draw from p_0 picks a component $\mathcal{N}(\mu_k, v_k)$ and then a point from it. By the kernel Equation 38.43, X_t is $\alpha_t X_0$ plus an independent $\mathcal{N}(0, \sigma_t^2)$ variable. Given the component, it is therefore a sum of two independent Gaussians, with mean $\alpha_t\mu_k$ and variance $\alpha_t^2 v_k + \sigma_t^2$, and averaging over the component gives the marginal: the mixture of the evolved components. With $\theta = 1$, $g = \sqrt{2}$ it is the mixture of $\mathcal{N}\big(\pm 2e^{-t},\, 1 - 0.75\,e^{-2t}\big)$, in closed form at every time.

The same fact can be read off the Fokker–Planck equation, which is linear in p. A mixture $w_1 p_t^{(1)} + w_2 p_t^{(2)}$ of solutions is therefore a solution, and by the uniqueness stated after Equation 38.49 it is the marginal for the mixed start. Under OU each Gaussian component follows the moment ODEs Equation 38.51 on its own.

Every experiment from here on uses this marginal, so we define it once, together with its CDF and a sampler. The parameters are explicit arguments, so the exercises can change them.

```
ws, mus, v0s = np.array([0.5, 0.5]), np.array([-2.0, 2.0]), np.array([0.25, 0.25])

def mixture_params(t, mus=mus, v0s=v0s, theta=theta, g=g):
    """Means and variances of the OU-evolved components (moment ODEs)."""
    alpha = np.exp(-theta * t)
    return alpha * mus, alpha**2 * v0s + g**2 / (2 * theta) * (1 - alpha**2)

def gauss(x, m, v):
    return np.exp(-(x - m)**2 / (2 * v)) / np.sqrt(2 * np.pi * v)

def p_t(x, t, ws=ws, mus=mus, v0s=v0s):
    """Exact marginal density of the mixture at time t."""
    m, v = mixture_params(t, mus, v0s)
    return sum(w * gauss(x, mk, vk) for w, mk, vk in zip(ws, m, v))

def mixture_cdf(x, t, ws=ws, mus=mus, v0s=v0s):
    m, v = mixture_params(t, mus, v0s)
    return sum(w * 0.5 * (1 + erf((x - mk) / np.sqrt(2 * vk)))
               for w, mk, vk in zip(ws, m, v))

def sample_p0(n, rng, ws=ws, mus=mus, v0s=v0s):
    k = rng.choice(len(ws), size=n, p=ws)   # component of each draw
```

```
    return mus[k] + np.sqrt(v0s[k]) * rng.standard_normal(n)
```

With the closed form in code, we can put it into the PDE directly. Both sides are evaluated on a grid at $t = 0.25$, 1 and 3, with ∂_t by a centered difference in time and the x-derivatives by `np.gradient`.

```
xs = np.linspace(-5.0, 5.0, 4001)
dx, eps = xs[1] - xs[0], 1e-5
for t in (0.25, 1.0, 3.0):
    dpdt = (p_t(xs, t + eps) - p_t(xs, t - eps)) / (2 * eps)
    p = p_t(xs, t)
    rhs = (np.gradient(theta * xs * p, dx)                # -d/dx(f p)
           + 0.5 * g**2 * np.gradient(np.gradient(p, dx), dx))
    res = (dpdt - rhs)[5:-5]            # trim the one-sided edge stencils
    print(f't={t}: max|residual| = {np.abs(res).max():.1e}, '
          f'max|dp/dt| = {np.abs(dpdt).max():.1e}')
```

```
t=0.25: max|residual| = 4.4e-06, max|dp/dt| = 4.4e-01
t=1.0: max|residual| = 6.3e-07, max|dp/dt| = 1.7e-01
t=3.0: max|residual| = 1.2e-06, max|dp/dt| = 3.2e-03
```

The residual is three to five orders of magnitude below $\partial_t p$ at each time (4×10^{-6} against 0.44 at $t = 0.25$; at $t = 3$ the density barely moves, and 10^{-6} stands against 3×10^{-3}). It is the truncation error of the second-order stencils: halving the grid spacing divides it by four at all three times. The argument did not need this computation; the cell guards against a slip in the code on which all later experiments rely.

38.4.3 The Score Function

The rest of the section rewrites the diffusion term with the gradient of $\log p_t$, and every construction that follows needs this gradient. We meet it first, with the mixture as the main example.

38.4.3.1 Definition and Geometry

The question is which direction, from a point $\mathbf{x}$, leads most quickly toward more probable points. The **score** of a density p_t answers it:

$$\mathbf{s}_t(\mathbf{x}) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}). \tag{38.52}$$

The score is a vector field over data space that points uphill on the log-density. The gradient is with respect to the input $\mathbf{x}$, not the parameters of a model; Section 37.5.1 introduced this data score and contrasted it with the parameter score of maximum likelihood.

The length of $\mathbf{s}_t$ is the rate at which $\log p_t$ increases, $\|\nabla p_t\|/p_t$. It is therefore large where the density is small but changing, in the tails and in the valleys between modes, and not where the density is large. The score vanishes at the critical points of p_t: modes, minima and saddles, which its derivative tells apart.

Take a Gaussian first. For $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu, v\mathbf{I})$ the log-density is $-\|\mathbf{x} - \mu\|^2/(2v) + \text{const}$, so

$$\nabla \log p(\mathbf{x}) = -\frac{\mathbf{x} - \mu}{v}. \tag{38.53}$$

The score of a Gaussian is a linear spring pulling toward the mean, stiffer for smaller variance. With $\mu = 0$, $v = 0.25$ and $x = 1$ it is -4, four times the pull at unit variance. For a general covariance the spring is $-\Sigma^{-1}(\mathbf{x} - \mu)$.

One more property follows from the definition. A constant factor in p drops out of $\nabla \log p$, so the score of $\tilde{p}/Z$ is the score of $\tilde{p}$ for any constant Z. The score can therefore be learned without ever computing a normalizing constant. Figure 38.17 shows the score field of a two-mode density in two dimensions.

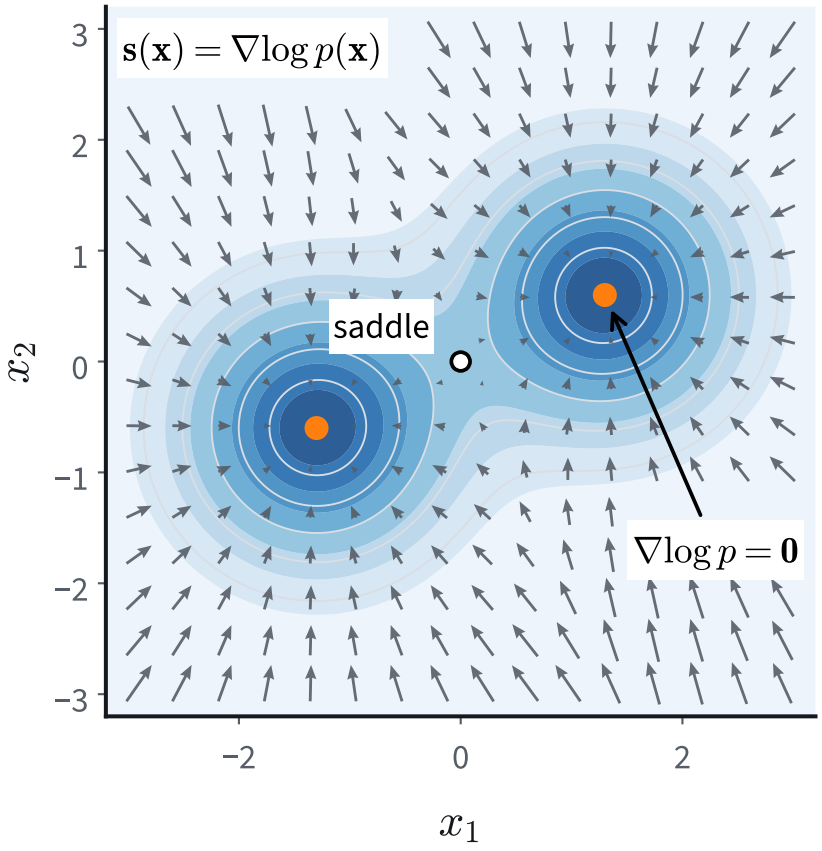


Figure 38.17: The score $\mathbf{s}(\mathbf{x}) = \nabla \log p(\mathbf{x})$ of a two-component Gaussian mixture, drawn over density contours. The arrows point uphill and vanish at critical points; the orange points mark the two modes, and the symmetry point between them is another zero, with different curvature. Arrows have the true length $\|\nabla p\|/p$, capped at 4 in the far tails, where it grows without bound; they are short near the modes and the saddle.

38.4.3.2 The Score of a Gaussian Mixture

Our data density is a mixture, so we need its score in a form we can evaluate and read. For $p(x) = \sum_k w_k N_k(x)$ with weights w_k and components $N_k(x) = \mathcal{N}(x; \mu_k, v_k)$, differenti-

ate $\log p$ directly:

$$\nabla \log p(x) = \frac{\sum_k w_k \nabla N_k(x)}{\sum_j w_j N_j(x)} = \sum_k r_k(x) \left(-\frac{x - \mu_k}{v_k} \right), \qquad r_k(x) = \frac{w_k N_k(x)}{\sum_j w_j N_j(x)}. \tag{38.54}$$

The mixture's score is a weighted average of the component springs. The weight $r_k(x)$ is the *responsibility* of component k, the posterior probability that a point at x came from it. For two components with equal weights, means $\pm\mu$ and a common variance v_0, the responsibilities reduce to a logistic function of x and the score has the closed form

$$s_0(x) = \frac{\mu \tanh(\mu x / v_0) - x}{v_0}. \tag{38.55}$$

Near each mode $\tanh \to \pm 1$ and the score is the local spring $-(x \mp \mu)/v_0$. For the running mixture, $\mu = 2$ and $v_0 = 0.25$, it is $s_0(x) = 8 \tanh(8x) - 4x$. At $x = 0$ the score vanishes, the third critical point, and it grows from zero with slope

$$s_0'(0) = \frac{\mu^2 - v_0}{v_0^2} = 60.$$

A positive slope means that $x = 0$ is a minimum of the density, a repeller of the uphill flow $\dot{x} = s_0(x)$. The mixture has two modes exactly when $\mu^2 > v_0$; for $\mu^2 \le v_0$ they merge into one (Exercise 6).

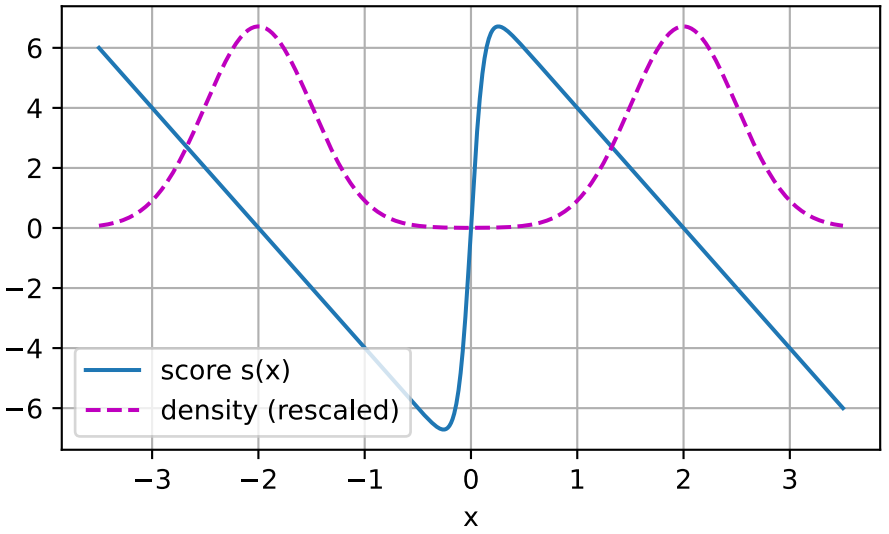
In code we use the general formula Equation 38.54, with a numerically stable softmax for the responsibilities, because the later experiments need the score of p_t at every time. The tanh form, the zeros and the slope 60 serve as checks on the implementation.

```
def score_t(x, t, ws=ws, mus=mus, v0s=v0s):
    """Exact score of p_t: responsibility-weighted component springs."""
    m, v = mixture_params(t, mus, v0s)
    logw = np.stack([np.log(w) - 0.5 * np.log(2 * np.pi * vk)
                     - (x - mk)**2 / (2 * vk) for w, mk, vk in zip(ws, m, v)])
    r = np.exp(logw - logw.max(axis=0))
    r /= r.sum(axis=0)                  # responsibilities, a stable softmax
    return sum(rk * -(x - mk) / vk for rk, mk, vk in zip(r, m, v))

gs = np.linspace(-3.5, 3.5, 1400)
mu, v0 = mus[1], v0s[1]
s = score_t(gs, 0.0)
s_tanh = (mu * np.tanh(mu * gs / v0) - gs) / v0
print(f'max|responsibility form - tanh form| = {np.abs(s - s_tanh).max():.1e}')
mid = (gs[:-1] + gs[1:]) / 2
print('score changes sign near', np.round(mid[s[:-1] * s[1:] < 0], 2) + 0.0)
delta = 1e-5                            # centered difference at x = 0
print(f'slope at 0: {(score_t(delta, 0.0) - score_t(-delta, 0.0)) / (2 * delta):.2f}, '
      f'formula (mu^2 - v0) / v0^2 = {(mu**2 - v0) / v0**2:.2f}')
dens = p_t(gs, 0.0)
d2l.plot(gs, [s, dens * np.abs(s).max() / dens.max()], 'x', '',
```

```
            legend=['score s(x)', 'density (rescaled)'], figsize=(5.5, 3))
```

```
max|responsibility form - tanh form| = 7.3e-15
score changes sign near [-2.  0.  2.]
slope at 0: 60.00, formula (mu^2 - v0) / v0^2 = 60.00
```



The checks pass: the two formulas agree to machine precision, and the zeros and the slope are where we computed them. What the plot adds is where the score is large: in the valley between the modes, where the density is nearly zero but its logarithm climbs steeply toward either mode, and in the tails.

38.4.4 The Continuity Equation and the Probability-Flow ODE

The Fokker–Planck equation describes particles that move at random. We now ask which law governs particles that move deterministically, and find that the two laws coincide once the noise is written as a velocity.

38.4.4.1 Conservation of Probability

Forget noise for a moment. Suppose every particle follows a velocity field, $\dot{\mathbf{x}} = \mathbf{u}(\mathbf{x}, t)$. What PDE does their density q_t obey? Probability is neither created nor destroyed, so the mass inside a fixed region Ω can change only by flowing across its boundary. Particles cross a small patch of boundary at the rate density times velocity. We call this flux density the **probability current**, $\mathbf{j} = q\,\mathbf{u}$: at each point it gives the rate, per unit area, at which probability crosses a surface placed there. The mass balance of the region is

$$\frac{d}{dt}\int_{\Omega} q_t \, d\mathbf{x} = -\oint_{\partial\Omega} \mathbf{j}\cdot\mathbf{n}\, dS = -\int_{\Omega} \nabla\cdot(q_t\mathbf{u})\, d\mathbf{x}, \tag{38.56}$$

where the divergence theorem Equation 38.45 turned the boundary flux into a volume integral. The mass inside changes at minus the net outflow. Since Ω is arbitrary, the integrands

agree pointwise, which gives the **continuity equation**:

$$\partial_t q_t + \nabla \cdot (q_t\, \mathbf{u}) = 0. \tag{38.57}$$

The density at a point falls at the rate at which the current diverges there. In one dimension the balance reads $\partial_t \int_a^b q\, dx = j(a) - j(b)$: what flows in at a minus what flows out at b. Figure 38.18 draws it.

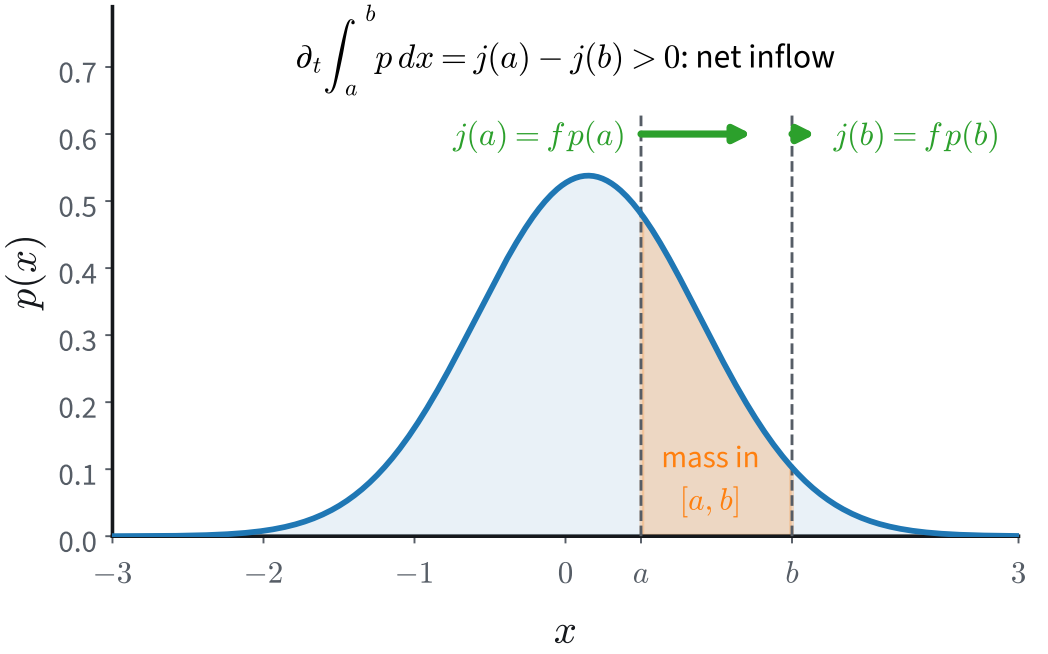


Figure 38.18: The continuity equation as a flux balance. A density $p(x)$ (blue) is carried by a constant drift f, giving the probability current $j(x) = f\,p(x)$. On the window from a to b on the right flank of the bump, the mass inside (orange) grows at the rate $j(a) - j(b)$, the net inflow: more current enters at a than leaves at b because the density is higher at a.

The balance also says what happens at the edges of the domain. On all of $\mathbb{R}^d$ we assume that the current decays at infinity. For the Fokker–Planck equation this means that $\mathbf{f}p_t$ and ∇p_t decay, which Gaussian tails ensure even though the OU drift grows linearly. Integrating Equation 38.57 over $\mathbb{R}^d$ then gives $\frac{d}{dt}\int q_t = 0$: total probability is conserved. On a bounded domain a *reflecting* boundary imposes $\mathbf{j} \cdot \mathbf{n} = 0$, so mass is conserved there too, while an *absorbing* boundary imposes $q = 0$, and mass can leave.

Two small cases show the equation at work in one dimension. For a translation, $u = c$, the equation is $\partial_t q + c\,\partial_x q = 0$, solved by $q_t(x) = q_0(x - ct)$: the profile slides without changing shape. For a contraction, $u = -x$, every particle moves as $x(t) = x(0)e^{-t}$, the cloud shrinks by the factor e^{-t}, and since the mass of each interval is preserved, the density carried along each trajectory rises by e^{t}.

The contraction suggests a general rule for the density seen by a moving particle. Expanding the divergence with the product rule,

$$\nabla \cdot (q\,\mathbf{u}) = \mathbf{u} \cdot \nabla q + q\,\nabla \cdot \mathbf{u},$$

the continuity equation becomes $\partial_t q + \mathbf{u} \cdot \nabla q = -q\,\nabla \cdot \mathbf{u}$. The left side is the rate of change of q along a trajectory $\mathbf{x}(t)$ of $\dot{\mathbf{x}} = \mathbf{u}$, by the chain rule. Dividing by q,

$$\frac{d}{dt} \log q_t(\mathbf{x}(t)) = -\nabla \cdot \mathbf{u}(\mathbf{x}(t), t). \tag{38.58}$$

Along a trajectory the log-density falls at the rate of the local expansion. For the contraction, $\nabla \cdot u = -1$ and the log-density rises at rate 1, as computed above. The trajectories of $\dot{\mathbf{x}} = \mathbf{u}$ are called the **characteristics** of the continuity equation. The rule is the instantaneous change of variables of Section 38.2.3, proved there along trajectories.

38.4.4.2 Diffusion as Transport

The Fokker–Planck equation Equation 38.49 is not yet a continuity equation: next to the transport term $-\nabla \cdot (\mathbf{f}p)$ it has a Laplacian. The chain rule turns the Laplacian into a transport term. Wherever $p > 0$,

$$\nabla p = p\,\nabla \log p, \tag{38.59}$$

which is the derivative of $\log p$ written backward, and therefore

$$\tfrac{1}{2}g^2\,\Delta p = \tfrac{1}{2}g^2\,\nabla \cdot (\nabla p) = \nabla \cdot (p \cdot \tfrac{1}{2}g^2 \nabla \log p). \tag{38.60}$$

The diffusion term is the divergence of a current, the density times $+\frac{1}{2}g^2\nabla \log p$. Moved to the left side of a continuity equation it appears with a minus sign, as a velocity $-\frac{1}{2}g^2\nabla \log p$.

Proposition (Fokker–Planck is a continuity equation). *Wherever $p_t > 0$, the Fokker–Planck equation Equation 38.49 is equivalent to*

$$\partial_t p_t + \nabla \cdot (p_t\,\mathbf{u}_t) = 0, \qquad \mathbf{u}_t(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t) - \tfrac{1}{2}g(t)^2\,\nabla \log p_t(\mathbf{x}). \tag{38.61}$$

Proof. Substitute Equation 38.60 into Equation 38.49 and merge the two divergences:

$$\partial_t p = -\nabla \cdot (\mathbf{f}p) + \nabla \cdot (p \cdot \tfrac{1}{2}g^2 \nabla \log p) = -\nabla \cdot (p\,[\mathbf{f} - \tfrac{1}{2}g^2 \nabla \log p]). \qquad \blacksquare$$

At the level of densities, the SDE moves probability as if each particle followed the velocity $\mathbf{u}_t$: the drift plus a push down the log-density gradient. The corresponding current is

$$\mathbf{j} = \mathbf{f}\,p_t - \tfrac{1}{2}g^2\,\nabla p_t = p_t\,\mathbf{u}_t. \tag{38.62}$$

The heat equation shows what this velocity looks like. Take $\mathbf{f} = 0$ and $p_t = \mathcal{N}(0, v_t)$ in one dimension with $v_t = v_0 + g^2 t$. The score is $-x/v_t$, so

$$u_t(x) = -\tfrac{1}{2}g^2\Big(-\frac{x}{v_t}\Big) = \frac{g^2}{2v_t}\,x.$$

The velocity points away from the center and is proportional to the distance from it, and it weakens as the density widens. The ODE $\dot{x} = g^2 x/(2v_t)$ separates: $\frac{d}{dt}\log x = \frac{g^2}{2v_t} = \frac{1}{2}\frac{d}{dt}\log v_t$, so

$$x(t) = x(0)\sqrt{v_t/v_0}. \tag{38.63}$$

Every particle is stretched by the same factor, and a Gaussian stretched by $\sqrt{v_t/v_0}$ has variance v_t, which is the variance that Brownian motion produces. With $v_0 = g = 1$ and $t = 3$ every particle doubles its distance from the center. A Brownian particle started at $x = 1$ instead lands at 1 plus a draw of standard deviation $\sqrt{3} \approx 1.73$. The two processes agree on the histogram at every time but move individual particles differently (Figure 38.19).

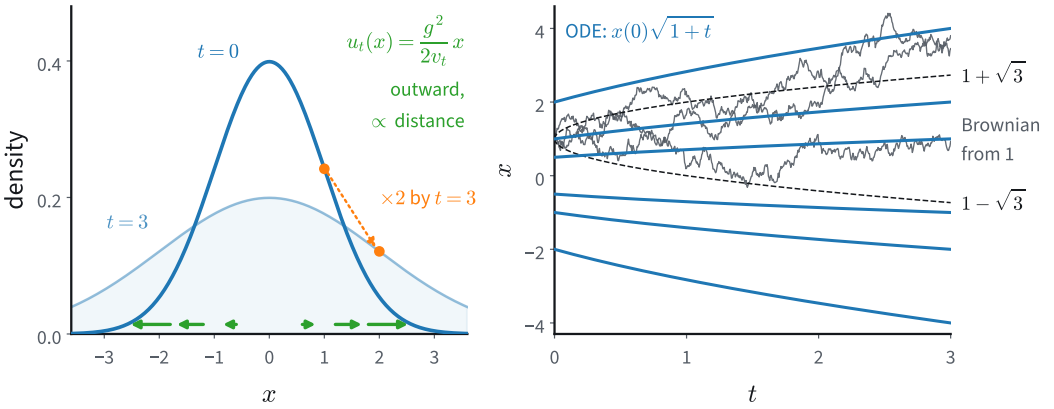


Figure 38.19: The heat equation $\partial_t p = \frac{1}{2}\partial_{xx} p$ from $p_0 = \mathcal{N}(0,1)$, solved as a flow. (a) The density at $t = 0$ and at $t = 3$, with the velocity $u_0(x) = x/2$ drawn as arrows on the axis: outward and proportional to distance. The particle at $x = 1$ moves to $x = 2$ by $t = 3$, and the density it carries halves. (b) Flow trajectories $x(0)\sqrt{1+t}$, smooth and never crossing, against three Brownian paths from $x = 1$ and their one-standard-deviation envelope $1 \pm \sqrt{t}$.

There is a limit to this construction. As $v_0 \to 0$ the velocity at small times is $g^2 x/(2g^2 t) = x/(2t)$, which is unbounded as $t \to 0$. A deterministic flow can spread a point mass into a Gaussian only through a field that blows up at the start, because a Lipschitz flow maps one point to one point (Exercise 4).

The next cell runs this example. From one draw of 20,000 points of $\mathcal{N}(0,1)$ it forms a Brownian cloud at $t = 3$ (adding an independent $\mathcal{N}(0,3)$ increment) and a flow cloud by integrating $\dot{x} = x/(2(1+t))$ with Heun steps (Section 38.1). It also evaluates both sides of Equation 38.60 by finite differences.

```
p = gauss(xs, 0.0, 2.0)                # a heat-equation Gaussian, g = 1
lhs = 0.5 * np.gradient(np.gradient(p, dx), dx)
rhs = np.gradient(p * 0.5 * np.gradient(np.log(p), dx), dx)
print(f'identity: max|lhs - rhs| = {np.abs(lhs - rhs)[5:-5].max():.1e} '
      f'(max|lhs| = {np.abs(lhs).max():.2f})')
```

```python
def heat_velocity(x, t, v0=1.0, g=1.0):  # u_t = -(1/2) g^2 score = g^2 x / (2 v_t)
    return g**2 * x / (2 * (v0 + g**2 * t))

n, T, steps = 20_000, 3.0, 600
dt = T / steps
z = rng.standard_normal(n)              # p_0 = N(0, 1)
x_bm = z + np.sqrt(T) * rng.standard_normal(n)   # Brownian motion to time T
x_ode = z.copy()
for k in range(steps):                  # Heun steps for the probability flow
    k1 = heat_velocity(x_ode, k * dt)
    k2 = heat_velocity(x_ode + dt * k1, (k + 1) * dt)
    x_ode += 0.5 * dt * (k1 + k2)
print(f'std at t={T}: Brownian {x_bm.std():.3f}, flow {x_ode.std():.3f}, '
      f'exact {np.sqrt(1 + T):.3f}')
print(f'max|x_ode - 2 z| = {np.abs(x_ode - 2 * z).max():.1e}')
```

```
identity: max|lhs - rhs| = 1.1e-07 (max|lhs| = 0.07)
std at t=3.0: Brownian 1.992, flow 1.984, exact 2.000
max|x_ode - 2 z| = 5.7e-06
```

Both clouds have standard deviation 2.00 up to a sampling error of order 0.01, and the flow map equals $x \mapsto 2x$ up to the Heun error of about 10^{-5}. The identity holds to finite-difference precision, as a consequence of the chain rule must. Random kicks and a deterministic stretch have produced the same histogram.

38.8.4.3 The Probability-Flow ODE

The heat equation was one example. The proposition above says that the same construction works for every SDE of our form: following the velocity $\mathbf{u}_t$ gives an ODE,

$$\frac{d\mathbf{x}}{dt} = \mathbf{u}_t(\mathbf{x}) = \mathbf{f}(\mathbf{x}, t) - \tfrac{1}{2} g(t)^2 \, \nabla \log p_t(\mathbf{x}), \tag{38.64}$$

the **probability-flow ODE** (Y. Song, Sohl-Dickstein, et al. 2021). The next proposition is the first of the two results that generative models take from this section.

Proposition (the deterministic twin). *Let p_t be the marginal density of $d\mathbf{X} = \mathbf{f}\,dt + g\,d\mathbf{W}$, $\mathbf{X}_0 \sim p_0$, assumed smooth and positive, so that it solves the Fokker–Planck equation. Suppose that $\mathbf{u}_t$ is continuous in t, continuously differentiable in $\mathbf{x}$, and Lipschitz in $\mathbf{x}$ with a constant that does not depend on t. Let $\mathbf{x}(t)$ solve Equation 38.64 with $\mathbf{x}(0) \sim p_0$. Then $\mathbf{x}(t) \sim p_t$ for every t: the ODE and the SDE share all time-marginals.*

The Lipschitz hypothesis holds for the OU drift and for the score of a Gaussian or a Gaussian mixture with positive component variances, whose slope is bounded for t in $[0, T]$.

Proof. By the existence and uniqueness theorem of Section 38.1.1.3, a field that is Lipschitz with a constant independent of t has exactly one trajectory through every point, forward and backward, so its time-t map is a bijection of $\mathbb{R}^d$. Let q_t be the density of the ODE's state. The field $\mathbf{u}_t$ is fixed and known (it is built from p, not from q), so q_t obeys the continuity equation $\partial_t q + \nabla \cdot (q\mathbf{u}_t) = 0$, which is linear in q. By the previous proposition p_t solves the same equation, and both start from $q_0 = p_0$. The derivation of the log-density rule Equation 38.58 used only this PDE, so along each characteristic $\log q_t$ and $\log p_t$ change at

the same rate $-\nabla \cdot \mathbf{u}_t$, and they start equal. Since the characteristics reach every point, $q_t = p_t$ everywhere. ■

The marginals agree; the paths do not. An SDE path is jagged and crosses other paths. An ODE trajectory is smooth, and two trajectories never meet at the same point at the same time, by uniqueness (panel (b) of Figure 38.19).

The ODE also gives a likelihood. Integrating the log-density rule Equation 38.58 with $q_t = p_t$ along a trajectory from 0 to T gives

$$\log p_0(\mathbf{x}(0)) = \log p_T(\mathbf{x}(T)) + \int_0^T \nabla \cdot \mathbf{u}_t(\mathbf{x}(t))\, dt. \tag{38.65}$$

The log-density of a data point is the log-density of its image at time T, which is nearly Gaussian and easy to evaluate, plus the accumulated expansion along the way. With a learned score in place of the exact one this is, up to solver error, the log-density of the *model* defined by that field, which is how likelihoods of diffusion models are reported (Y. Song, Sohl-Dickstein, et al. 2021).

38.4.4.4 SDE and ODE Marginals Compared

The heat flow was a stretch of one Gaussian. On the mixture the proposition has more to do: the density changes shape, from two modes to one, and the jagged and the smooth process must still agree on it at every time. From one sample of p_0 we launch two clouds of $20{,}000$ particles: one follows the SDE with Euler–Maruyama steps, the other the probability-flow ODE with Heun steps and the exact score. As a yardstick we compute the one-sample Kolmogorov–Smirnov distance of each cloud to the exact CDF of p_t, the largest gap between the empirical and the exact CDF. For an exact sample of size n this distance exceeds $1.36/\sqrt{n}$ only 5% of the time. Before looking at the output, guess which cloud's distance changes more from one snapshot to the next.

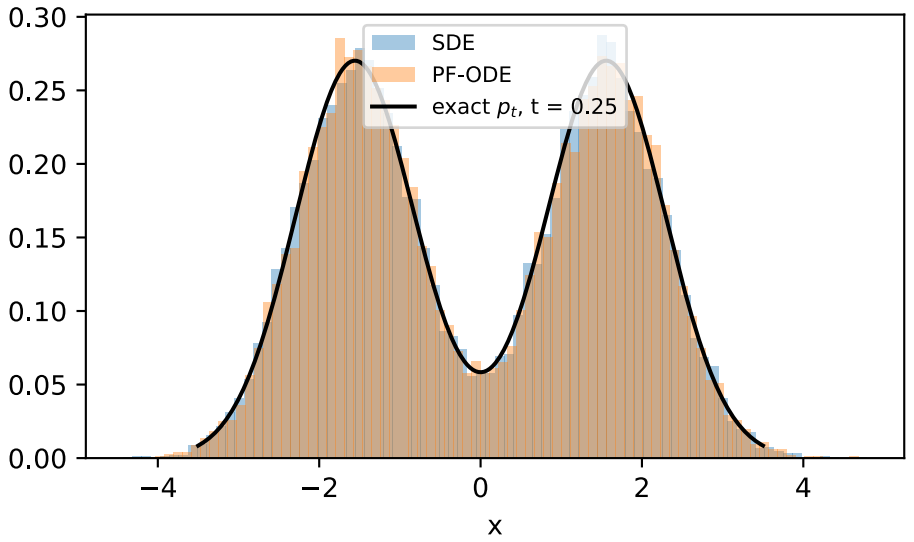
```
def pf_velocity(x, t, score=score_t, theta=theta, g=g):
    return -theta * x - 0.5 * g**2 * score(x, t)   # u_t = f - (1/2) g^2 s_t

def ks_exact(sample, t, **mixture):
    """One-sample Kolmogorov--Smirnov distance to the exact p_t."""
    xs_sorted, n = np.sort(sample), len(sample)
    F = mixture_cdf(xs_sorted, t, **mixture)
    return max((np.arange(1, n + 1) / n - F).max(), (F - np.arange(n) / n).max())

n, T, steps = 20_000, 3.0, 600
dt = T / steps
x_sde = sample_p0(n, rng)
x_ode = x_sde.copy()                    # both clouds start from one sample
snaps = {}
for k in range(steps):
    t = k * dt
    x_sde += -theta * x_sde * dt + g * np.sqrt(dt) * rng.standard_normal(n)
    k1 = pf_velocity(x_ode, t)          # Heun steps for the ODE
    k2 = pf_velocity(x_ode + dt * k1, t + dt)
    x_ode += 0.5 * dt * (k1 + k2)
```

```
    if k + 1 in (50, 200, 600):          # t = 0.25, 1, 3
        snaps[(k + 1) * dt] = (x_sde.copy(), x_ode.copy())
print(f'95% band for n = {n}: {1.36 / np.sqrt(n):.4f}')
for t, (a, b) in snaps.items():
    print(f't={t:.2f}: KS(SDE) = {ks_exact(a, t):.4f}, KS(ODE) = {ks_exact(b, t):.4f}')
gx = np.linspace(-3.5, 3.5, 400)
d2l.set_figsize((5.5, 3))
d2l.plt.hist(snaps[0.25][0], bins=80, density=True, alpha=0.4, label='SDE')
d2l.plt.hist(snaps[0.25][1], bins=80, density=True, alpha=0.4, label='PF-ODE')
d2l.plt.plot(gx, p_t(gx, 0.25), 'k', label='exact $p_t$, t = 0.25')
d2l.plt.xlabel('x'); d2l.plt.legend(loc='upper center', fontsize=8);
```

```
95% band for n = 20000: 0.0096
t=0.25: KS(SDE) = 0.0038, KS(ODE) = 0.0053
t=1.00: KS(SDE) = 0.0044, KS(ODE) = 0.0053
t=3.00: KS(SDE) = 0.0070, KS(ODE) = 0.0053
```



In most runs both clouds stay within the 0.0096 band at all three times, with distances of order 0.003 to 0.009. A value just above the band occurs about as often as it would for an exact sample. At $t = 0.25$ the marginal is still bimodal, and both histograms reproduce its two modes; by $t = 1$ it has a single mode (the merging time is 0.78, Exercise 6). The ODE's distance hardly changes with time. A strictly increasing map that carries p_0 to p_t carries the sample's empirical CDF along with it, so the ODE cloud keeps the distance its starting sample had and adds only solver error. The SDE cloud is redrawn by the noise and its distance fluctuates.

38.4.5 Time Reversal

The forward SDE turns data into noise. Generation needs the process played backward, from noise to data. This may sound impossible: the heat equation cannot be solved backward from an arbitrary final density, because smoothing destroys detail that nothing can restore. We do not solve it backward. The forward family p_t is given, and we ask only for

a process that visits it in reverse order; its drift will contain $\nabla \log p_t$, which carries the destroyed detail. The ODE of the previous subsection is one such process, integrated backward. For the SDE the answer is less obvious: the backward process is again a diffusion, and its drift carries twice the score correction of the ODE.

38.4.5.1 Bayes' Rule on an Infinitesimal Step

To reverse one small step we use Bayes' rule to ask where a particle came from. Over a short interval τ the forward step is the Euler–Maruyama Gaussian (Section 38.3.4.1):

$$p(\mathbf{x}_{t+\tau} \mid \mathbf{x}_t) \approx \mathcal{N}(\mathbf{x}_{t+\tau};\ \mathbf{x}_t + \mathbf{f}(\mathbf{x}_t, t)\,\tau,\ g^2\tau\,\mathbf{I}).$$

Now condition on the endpoint $\mathbf{x}_{t+\tau} = \mathbf{y}$. Bayes' rule gives the probable starting points as the forward kernel reweighted by the marginal at time t:

$$p(\mathbf{x}_t = \mathbf{x} \mid \mathbf{x}_{t+\tau} = \mathbf{y}) \propto p(\mathbf{y} \mid \mathbf{x})\ p_t(\mathbf{x}).$$

A starting point is likely if it could have reached $\mathbf{y}$ and if it was likely to begin with. Take logarithms and expand $\log p_t(\mathbf{x})$ around the endpoint: $\log p_t(\mathbf{x}) \approx \log p_t(\mathbf{y}) + (\mathbf{x}-\mathbf{y}) \cdot \nabla \log p_t(\mathbf{y})$. The step $\mathbf{x}-\mathbf{y}$ is of size $\sqrt{\tau}$, so the neglected quadratic term is of order τ and changes the posterior only at higher order. To the same order we may evaluate $\mathbf{f}$ at $\mathbf{y}$ instead of $\mathbf{x}$, since $\mathbf{f}(\mathbf{x},t) - \mathbf{f}(\mathbf{y},t)$ is of size $\sqrt{\tau}$ and is multiplied by τ. Adding the linear term to the Gaussian exponent $-\|\mathbf{x}-(\mathbf{y}-\mathbf{f}(\mathbf{y},t)\,\tau)\|^2/(2g^2\tau)$ and completing the square shifts the mean by $g^2\tau\,\nabla \log p_t(\mathbf{y})$:

$$p(\mathbf{x} \mid \mathbf{y}) \approx \mathcal{N}(\mathbf{x};\ \mathbf{y} - [\mathbf{f}(\mathbf{y},t) - g^2 \nabla \log p_t(\mathbf{y})]\,\tau,\ g^2\tau\,\mathbf{I}). \tag{38.66}$$

A backward step is a Gaussian kick of the same variance $g^2\tau$, whose drift is the forward drift corrected by the full g^2 times the score. The marginal p_t acts as a prior that bends the reversed step toward where the data density was.

38.4.5.2 The Reverse-Time SDE

The one-step calculation is a heuristic: it expands to first order and starts from an approximate kernel. Anderson (1982) proved that it gives the right answer for the whole process. We state his theorem without proof: the time reversal of the diffusion $d\mathbf{X} = \mathbf{f}\,dt + g\,d\mathbf{W}$ is the **reverse-time SDE**

$$d\mathbf{X} = \big[\mathbf{f}(\mathbf{X},t) - g(t)^2\,\nabla_{\mathbf{x}} \log p_t(\mathbf{X})\big]\,dt + g(t)\,d\bar{\mathbf{W}}, \tag{38.67}$$

integrated from $t = T$ down to $t = 0$. Here $\bar{\mathbf{W}}$ is a Brownian motion that runs backward in time; in a simulation it means a fresh Gaussian increment at each backward step. Started from $\mathbf{X}_T \sim p_T$, the process has marginal p_t at every intermediate time, and at $t = 0$ it delivers the data density. The theorem identifies the reversed process as a whole; below we prove only the statement about marginals, which is the part that sampling needs.

The smallest case can be checked by hand. Take OU at stationarity, $p_t = \mathcal{N}(0,1)$ for all t, with $\theta = 1$, $g = \sqrt{2}$ and score $-x$. The probability-flow velocity is $-x - \frac{1}{2}\cdot 2 \cdot (-x) = 0$:

the ODE particles do not move, so the cloud stays $\mathcal{N}(0,1)$. In the reversed clock $\bar{t} = T - t$ the reverse SDE has drift $-\mathbf{f} + g^2\mathbf{s}_t = x - 2x = -x$ and noise $\sqrt{2}$, which is the same OU process. Stationary OU is its own reverse.

With $\frac{1}{2}g^2$ in place of g^2 the reverse drift would be $x - x = 0$, and the cloud would spread like Brownian motion instead of staying put. So the reverse SDE needs the full factor g^2, twice the correction of the ODE. A family of samplers explains the factor.

Proposition (one family, marginals fixed). *Let p_t be the marginal density of the forward SDE, under the hypotheses of the deterministic-twin proposition and with $\mathbf{f}$ Lipschitz in $\mathbf{x}$. For any $\lambda \geq 0$, the reverse-time SDE with noise level λg and drift*

$$\mathbf{b}_\lambda = \mathbf{f} - \tfrac{1+\lambda^2}{2}\, g^2\, \nabla \log p_t \tag{38.68}$$

has marginals p_t when started from p_T. The choice $\lambda = 1$ is Anderson's reverse SDE Equation 38.67; the choice $\lambda = 0$ is the probability-flow ODE Equation 38.64.

The more noise a sampler injects, the larger its score correction.

Proof. Run the clock backward: let $\bar{t} = T - t$ and $q_{\bar{t}} = p_{T-\bar{t}}$, the density evolution played in reverse, so that $\partial_{\bar{t}} q = -\partial_t p = \nabla \cdot (\mathbf{f} q) - \frac{1}{2}g^2 \Delta q$, with $\mathbf{f}$ and g evaluated at forward time $T - \bar{t}$. A process $d\mathbf{Y} = \mathbf{b}'\, d\bar{t} + \lambda g\, d\mathbf{W}$ in the reversed clock has its own Fokker–Planck equation $\partial_{\bar{t}} q = -\nabla \cdot (q\, \mathbf{b}') + \frac{\lambda^2}{2} g^2 \Delta q$. For the two to agree we need

$$-\nabla \cdot (q\, \mathbf{b}') = \nabla \cdot (\mathbf{f} q) - \tfrac{1+\lambda^2}{2} g^2 \Delta q = \nabla \cdot \Big(q \Big[\mathbf{f} - \tfrac{1+\lambda^2}{2} g^2 \nabla \log q \Big] \Big),$$

where *both* Laplacians were converted to transport form with Equation 38.60. The equation holds for $\mathbf{b}' = -\mathbf{f} + \frac{1+\lambda^2}{2} g^2 \nabla \log q$. A drift $\mathbf{b}'$ in the reversed clock is a drift $-\mathbf{b}'$ in forward time, which gives Equation 38.68. It remains to pass from equal evolution equations and the equal start $q_0 = p_T$ to equal marginals, and this is the step that needs uniqueness. For $\lambda = 0$ it is the argument along characteristics of the deterministic twin. For $\lambda > 0$ it is the uniqueness stated after Equation 38.49, which we did not prove; it applies because $\mathbf{b}'$ is Lipschitz when $\mathbf{f}$ and $\mathbf{u}_t$ are. ■

The proof also explains the factor of two. A reverse process with noise λg smooths its own density, so its drift must undo two smoothings. One $\frac{1}{2}g^2 \nabla \log p_t$ cancels the forward smoothing; that is the whole correction of the probability-flow ODE, which injects no noise. A further $\frac{\lambda^2}{2} g^2 \nabla \log p_t$ compensates the smoothing by the reverse process's own noise. Anderson's $\lambda = 1$ contains both halves. The marginals do not determine the drift, though: any other drift with the same marginals differs from $\mathbf{b}'$ by a field $\mathbf{w}$ with $\nabla \cdot (q\mathbf{w}) = 0$, and Exercise 9 builds such fields.

Intermediate values of λ trade the randomness of the sampler against the size of the correction at fixed marginals, and diffusion-model samplers use this freedom (Section 17.4). Figure 38.20 shows three values of λ on the running mixture. The deterministic paths split toward the two modes near $\bar{t} \approx 2.2$, that is, near forward time 0.78, where the marginal stops being bimodal.

In Equation 38.67 the forward drift $\mathbf{f}$ and the noise level g are chosen by us, and the terminal density p_T is close to a known Gaussian by design. The score $\nabla \log p_t$ is the only unknown,

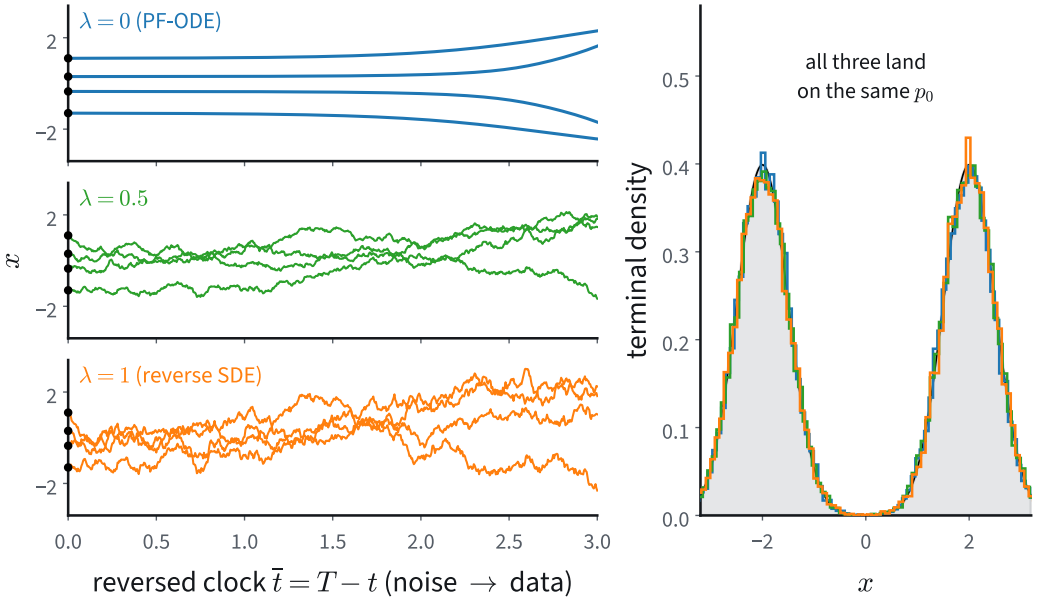


Figure 38.20: Reverse-time trajectories from the same four terminal points, using the exact score of the running mixture with drift $\mathbf{f} - \frac{1+\lambda^2}{2} g^2 \nabla \log p_t$ and noise λg. The probability-flow ODE ($\lambda = 0$) is deterministic; $\lambda = 0.5$ and Anderson's reverse SDE ($\lambda = 1$) add noise and a larger score correction. The three terminal histograms of $20{,}000$ particles each (right) agree with the data density p_0.

and it is the same field that the probability-flow ODE needs, so one estimate of it serves both samplers. Figure 38.21 shows the pair of processes as evolving densities.

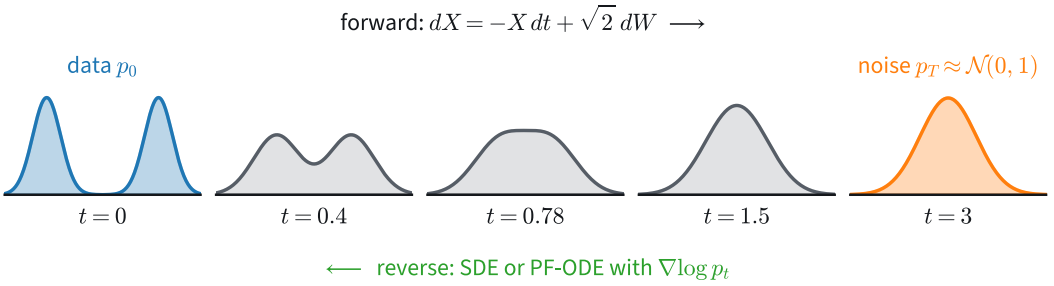


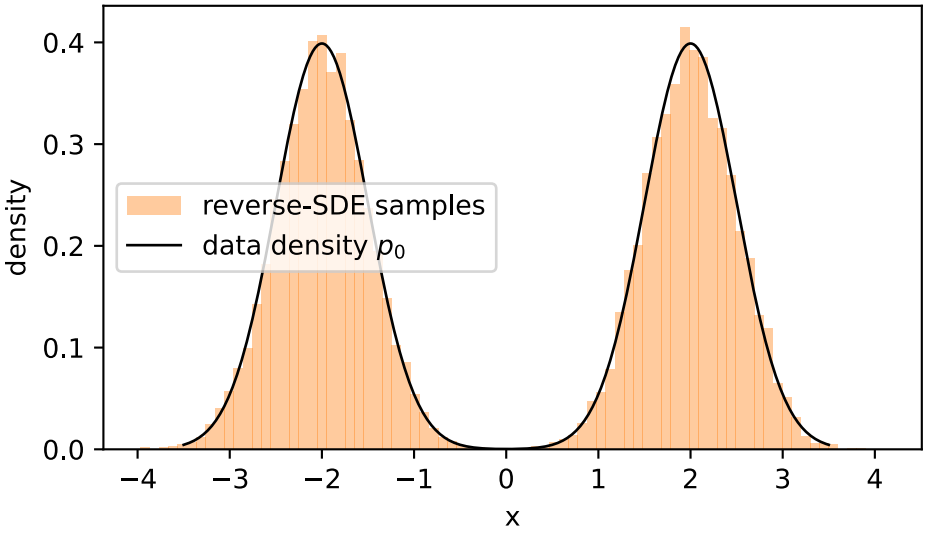
Figure 38.21: The forward marginals of the running mixture $p_0 = \frac{1}{2}\mathcal{N}(-2, 0.25) + \frac{1}{2}\mathcal{N}(2, 0.25)$ under OU with $\theta = 1$, $g = \sqrt{2}$, exact at every time. The reverse processes of this section run the same marginals from right to left. The flat top at $t \approx 0.78$ is where the two modes merge.

38.4.5.3 Reverse-Time Sampling

We now generate data with the exact score in place of a trained network. We draw $20{,}000$ points from $\mathcal{N}(0, 1)$, which approximates p_T at $T = 3$, and integrate Anderson's reverse SDE down to $t = 0$. In the reversed clock $\bar{t} = T - t$, Equation 38.67 reads $d\mathbf{Y} = [-\mathbf{f} + g^2 \nabla \log p_{T-\bar{t}}]\, d\bar{t} + g\, d\mathbf{W}$, an ordinary forward SDE that Euler–Maruyama can integrate. The cell first measures how far p_T is from $\mathcal{N}(0, 1)$, then compares the generated modes with the targets ± 2 and width 0.5.

```
print(f'max|p_T - N(0,1)| at T=3: {np.abs(p_t(gs, 3.0) - gauss(gs, 0.0, 1.0)).max():.4f}'
      f' (peak {p_t(gs, 3.0).max():.2f})')
x = rng.standard_normal(n)              # N(0, 1) stands in for p_T
for k in range(steps):                  # reversed clock: t_bar = k dt
    t = T - k * dt                      # forward time
    x += (theta * x + g**2 * score_t(x, t)) * dt \
         + g * np.sqrt(dt) * rng.standard_normal(n)
left, right = x[x < 0], x[x >= 0]
print(f'mass on the left: {len(left) / n:.3f} (target 0.5)')
print(f'means {left.mean():+.3f}, {right.mean():+.3f} (targets -2, +2)')
print(f'widths {left.std():.3f}, {right.std():.3f} (target 0.5)')
print(f'KS to the exact p_0: {ks_exact(x, 0.0):.4f}')
d2l.set_figsize((5.5, 3))
d2l.plt.hist(x, bins=80, density=True, alpha=0.4, color='C1',
             label='reverse-SDE samples')
d2l.plt.plot(gs, p_t(gs, 0.0), 'k', lw=1, label='data density $p_0$')
d2l.plt.xlabel('x'); d2l.plt.ylabel('density'); d2l.plt.legend();
```

```
max|p_T - N(0,1)| at T=3: 0.0016 (peak 0.40)
mass on the left: 0.499 (target 0.5)
means -1.997, +2.000 (targets -2, +2)
widths 0.507, 0.507 (target 0.5)
KS to the exact p_0: 0.0044
```



The terminal mismatch $\max_x |p_T - \mathcal{N}(0,1)|$ is 0.0016 against a peak of 0.40, so starting from Gaussian noise is justified. The reverse SDE recovers modes at ± 2 to within about 0.02. Its widths scatter around 0.505 by a few thousandths, slightly above the target 0.5, and the KS distance to the exact p_0 is usually below 0.01, close to the 0.0096 band of an exact sample.

The fraction on the left tells us less than its place in the output suggests. Start, score and noise are symmetric about 0, so the law of the output is symmetric whatever the sampler's errors, and the fraction is one half up to the binomial standard deviation 0.0035. It can show sampling noise and nothing else. Whether a reverse sampler restores *unequal* mode weights this run cannot say; the exercise "Prior mismatch, ODE versus SDE" takes that up with weights 0.25 and 0.75.

The width excess is the bias of Euler–Maruyama, which has weak order one (Section 38.3.4.1). The Monte Carlo error of a width from $10{,}000$ samples, $0.5/\sqrt{2 \cdot 10^4} \approx 0.0035$, hides it, but we can isolate it without sampling. For Gaussian data $\mathcal{N}(2, 0.25)$, the right mode alone, every reverse step is affine in x. The mean m and the variance V of the sampler's state then follow an exact recursion. With $\kappa_t = \theta - g^2/v_t$ the slope of the reverse drift, it reads $m \leftarrow m + \Delta t\,(\kappa_t m + g^2 m_t/v_t)$ and $V \leftarrow (1 + \kappa_t \Delta t)^2 V + g^2 \Delta t$. Starting from the exact p_T, the only error left is the discretization.

```
def em_reverse_width(steps, T=3.0, mu=2.0, v0=0.25):
    """Exact output width of reverse Euler--Maruyama on data N(mu, v0)."""
    dt, mu, v0 = T / steps, np.array([mu]), np.array([v0])
    (m,), (V,) = mixture_params(T, mu, v0)          # start from the exact p_T
    for k in range(steps):
        (mt,), (vt,) = mixture_params(T - k * dt, mu, v0)
        kappa = theta - g**2 / vt       # slope of the reverse drift in x
        m, V = (m + dt * (kappa * m + g**2 * mt / vt),
                (1 + kappa * dt)**2 * V + g**2 * dt)
    return np.sqrt(V)
```

```
for steps in (300, 600, 1200, 2400):
    print(f'{steps:4d} steps: width {em_reverse_width(steps):.4f} (target 0.5)')
```

```
 300 steps: width 0.5096 (target 0.5)
 600 steps: width 0.5047 (target 0.5)
1200 steps: width 0.5024 (target 0.5)
2400 steps: width 0.5012 (target 0.5)
```

The width excess over 0.5 is 0.0096, 0.0047, 0.0024 and 0.0012: it halves when the step halves, the first-order bias of Euler–Maruyama. Most of it arises near $t = 0$, where the reverse drift is stiff: its slope there is $\theta - g^2/v_0 = 1 - 8 = -7$, against -1 near $t = T$ (Section 38.1.3.3). Making the steps ten times finer on $t \le 0.5$ alone, $1{,}500$ steps in all, removes nearly nine tenths of the bias; $1{,}500$ uniform steps remove six tenths. At 600 steps the bias, 0.005, accounts for the excess of the sampling run and is only slightly larger than the Monte Carlo error of a width. One sampling run therefore cannot resolve it, while the recursion can.

In this section the data were a two-component mixture whose score has a closed form, and the only errors were Euler–Maruyama's bias and the small gap between p_T and $\mathcal{N}(0, 1)$. For images we have samples of p_0 and nothing else. The score of p_t is unknown, and solving the Fokker–Planck equation for it is out of the question: a grid with 100 cells per axis has 100^d unknowns. Section 38.5 shows that a regression on noised samples recovers the score.

38.4.6 Summary

The density of an SDE $d\mathbf{X} = \mathbf{f}\,dt + g\,d\mathbf{W}$ obeys the Fokker–Planck equation $\partial_t p = -\nabla \cdot (\mathbf{f}p) + \frac{1}{2}g^2\Delta p$, in which the drift transports the density and the noise smooths it. For OU a Gaussian solves it if and only if its moments obey $\dot m = -\theta m$ and $\dot v = g^2 - 2\theta v$. Because $\nabla p = p\nabla \log p$, the smoothing term is transport with velocity $-\frac{1}{2}g^2\nabla \log p$, so the Fokker–Planck equation is a continuity equation with velocity $\mathbf{u}_t = \mathbf{f} - \frac{1}{2}g^2\nabla \log p_t$. The probability-flow ODE $\dot{\mathbf{x}} = \mathbf{u}_t(\mathbf{x})$ has the SDE's marginals at every time, with smooth, non-crossing trajectories along which the likelihood can be computed. The time reversal of the SDE has drift $\mathbf{f} - g^2\nabla \log p_t$: one half of the correction undoes the forward smoothing, and the other compensates the noise of the reverse process. A parameter λ interpolates between the ODE and the SDE at fixed marginals (proved here for the marginals, given uniqueness for the Fokker–Planck equation; Anderson's theorem for the whole process is cited). In all of these samplers the only unknown is the score $\nabla \log p_t$.

38.4.7 Exercises

1. **Laplacian as averaging.** Averaging a smooth h over the sphere of radius r centered at $\mathbf{x}$ gives $h(\mathbf{x}) + \frac{r^2}{2d}\Delta h(\mathbf{x}) + O(r^4)$, as stated in Section 38.4.2.1.
    1. Let $\mathbf{e}$ be uniformly distributed on the unit sphere of $\mathbb{R}^d$. Show by symmetry that $\mathbb{E}[e_i] = 0$, $\mathbb{E}[e_i e_j] = \delta_{ij}/d$ and $\mathbb{E}[e_i e_j e_k] = 0$. Use these moments in a Taylor expansion of $h(\mathbf{x} + r\mathbf{e})$ to prove the averaging formula. Why does no r^3 term appear?
    2. In one dimension the sphere is the pair of points $x \pm \delta$. Show that one forward-Euler step of the heat equation $\partial_t p = D\,\partial_{xx} p$, with $\partial_{xx} p$ replaced by the centered difference $(p_{i+1} - 2p_i + p_{i-1})/\delta^2$, sets each new p_i to a weighted combi-

nation of p_{i-1}, p_i and p_{i+1} whose weights sum to one. Find the largest step Δt for which all three weights are nonnegative. Why can the iteration then never create a new maximum?

2. **Moments from the weak form.** The evolution law Equation 38.48 was derived for test functions that vanish outside a bounded set. Assume that it also holds for polynomial ϕ, which the Gaussian tails of this section's densities justify.
    1. For the one-dimensional OU process, insert $\phi(x) = x$ and $\phi(x) = x^2$ and derive ODEs for the mean $m(t)$ and the variance $v(t)$ of X_t. Show that they are the moment ODEs Equation 38.51 for every initial law p_0, Gaussian or not. Check this against `mixture_params` by computing the mean and the variance of the two-mode mixture at time t.
    2. For the linear SDE $d\mathbf{X} = \mathbf{A}\mathbf{X}\,dt + g\,d\mathbf{W}$ in $\mathbb{R}^d$ with a constant matrix $\mathbf{A}$, use $\phi(\mathbf{x}) = x_i$ and $\phi(\mathbf{x}) = x_i x_j$ to show that the mean and the covariance satisfy $\dot{\mathbf{m}} = \mathbf{A}\mathbf{m}$ and $\dot{\Sigma} = \mathbf{A}\Sigma + \Sigma\mathbf{A}^\top + g^2\mathbf{I}$. Which matrix equation must a stationary covariance solve?

3. **Stationary density from zero current.**
    1. Set $\partial_t p = 0$ in the OU Fokker–Planck equation, $0 = \theta\,\partial_x(xp) + \frac{1}{2}g^2\partial_{xx}p$. Integrate once in x and argue from decay at infinity that the constant of integration, which is minus the probability current Equation 38.62, vanishes. Solve the resulting first-order ODE and show that $p_\infty = \mathcal{N}(0, g^2/(2\theta))$ without assuming a Gaussian form.
    2. Repeat the argument for a general drift $f(x)$ with constant g, and express p_∞ through a potential U with $f = -U'$. Conversely, given a density $p \propto e^{-E(x)}$, find the drift that makes p stationary and write it through the score of p.
    3. Apply the result to the double-well drift $f(x) = x - x^3$ with $g = \sqrt{2}$ and find the modes of p_∞. What goes wrong for the unstable drift $f(x) = \theta x$ with $\theta > 0$?

4. **Affine Gaussian flow.** Start the OU process from $X_0 \sim \mathcal{N}(x_0, v_0)$ with $v_0 > 0$. Then $p_t = \mathcal{N}(m_t, v_t)$, where m_t and v_t solve the moment ODEs Equation 38.51. This is the heat-equation example of Section 38.4.4 with a drift and a moving center.
    1. Write the probability-flow velocity Equation 38.61 for this p_t and show that it is affine in x. Show that the standardized coordinate $z = (x - m_t)/\sqrt{v_t}$ is constant along every trajectory, and write the trajectory $x(t)$ in closed form. Compare it with Equation 38.63.
    2. Compute $\partial_x u_t$, evaluate the time integral in the likelihood identity Equation 38.65 along a trajectory, and verify the identity by comparing the two Gaussian log-densities.
    3. Let $v_0 \to 0$, so that the process starts from the point mass at x_0 of Equation 38.43. What happens to the velocity field as $t \to 0$? Use Section 38.1.1.3 to explain why a deterministic flow can spread a point mass into a Gaussian only through a field of this kind.

5. code **Same marginals, different couplings.** The `#fokker-planck-pf-ode-overlay` cell launches the SDE cloud and the ODE cloud from one shared sample of p_0. Store a copy of that sample before the loop.
    1. At each snapshot time, compute for both clouds the fraction of particles whose sign differs from the sign of their starting point, that is, the fraction that has

switched modes. The order-preservation proposition of Section 38.2.1.2 says that the time-T map of a field on $\mathbb{R}$ that is continuous in t and globally Lipschitz in x is strictly increasing. Use this and the symmetry of p_0 to explain why the fraction is zero for the exact flow. Does Heun's method keep it at zero? (Hint: the slope of u_t at 0 near $t = 0$ is $-\theta - \frac{1}{2}g^2 s_0'(0) = -61$.)

2. Compute the correlation between starting and current positions for both clouds. Predict the value for the SDE cloud from the OU solution $X_t = e^{-\theta t}X_0 + g\int_0^t e^{-\theta(t-s)}\,dW_s$ of Section 38.3.3 and the variance 4.25 of the mixture p_0, and compare.
3. Which uses of a diffusion model need the deterministic coupling of the probability-flow ODE, and not only its marginals?

6. **Mode merging.** Consider the symmetric two-component mixture with the score Equation 38.55.

1. Derive Equation 38.55 from the responsibility formula Equation 38.54.
2. Show that the score has three zeros when $\mu^2 > v_0$ and only the zero $x = 0$ when $\mu^2 \le v_0$. Classify each zero as a local maximum or a local minimum of the density. (Hint: the slope at 0 is $(\mu^2 - v_0)/v_0^2$.)
3. Under the OU process the mixture stays symmetric, with component means $\pm\mu e^{-\theta t}$ and the component variance computed by `mixture_params`. Find the time t^* at which p_t stops being bimodal for the section's parameters, $\mu = 2$, $v_0 = 0.25$, $\theta = 1$ and $g = \sqrt{2}$, and find the panel of Figure 38.21 that shows it.
4. What does the change at t^* mean for a reverse-time sampler that starts from noise? (Biroli et al. (2024) analyze the high-dimensional counterpart of this symmetry breaking and call it speciation.)

7. code **State-dependent noise.** ⊠ For an SDE $d\mathbf{X} = \mathbf{f}\,dt + \mathbf{G}(\mathbf{X}, t)\,d\mathbf{W}$ with a matrix noise, write $\mathbf{a} = \mathbf{G}\mathbf{G}^\top$. Itô's correction is then $\frac{1}{2}\sum_{ij} a_{ij}\,\partial_i\partial_j\phi$, and two integrations by parts, $\int p\,a_{ij}\,\partial_i\partial_j\phi = \int \phi\,\partial_i\partial_j(a_{ij}p)$, give $\partial_t p = -\nabla\cdot\mathbf{j}$ with the current

$$j_i = f_i\,p - \tfrac{1}{2}\sum_k \partial_k(a_{ik}\,p)$$

and, dividing by p, the probability-flow velocity

$$\mathbf{u} = \mathbf{f} - \tfrac{1}{2}\,\mathbf{a}\,\nabla\log p - \tfrac{1}{2}\,\nabla\cdot\mathbf{a}, \qquad (\nabla\cdot\mathbf{a})_i = \sum_k \partial_k a_{ik}.$$

A constant matrix only replaces g^2 by $\mathbf{a}$ in the score term; a state-dependent $\mathbf{a}$ adds the last term. Now let $dX = g(X)\,dW$ with no drift and a smooth positive g, on the circle $[0, 1)$ with periodic boundaries.

1. Specialize the matrix current and the general velocity above to this SDE. Show that a stationary density has constant current, that periodicity forces the constant to vanish, and find p_∞.
2. Replacing the constant g^2 of Equation 38.49 by $g(x)^2$ gives the PDE $\partial_t p = \frac{1}{2}g(x)^2\,\partial_{xx}p$. Which stationary density does this PDE predict?
3. Simulate $50{,}000$ particles started from the uniform distribution with Euler–Maruyama, $g(x) = 1 + \frac{1}{2}\sin(2\pi x)$ and $\Delta t = 10^{-3}$ up to $t = 2$, wrapping

positions modulo 1. Compare the histogram with both predictions. Where do the particles accumulate, and why?

8. code **Fokker–Planck on a grid.** ⊠ Divide $[-6, 6]$ into N cells of width $\delta = 12/N$ with centers x_i, and let $p_i(t)$ approximate the density in cell i. A conservative discretization of the OU equation assigns to each interior face $x_{i+1/2}$ the current

$$j_{i+1/2} = f(x_{i+1/2}) \frac{p_i + p_{i+1}}{2} - \frac{g^2}{2} \frac{p_{i+1} - p_i}{\delta}, \tag{38.69}$$

sets the current through the two outer faces to zero, and evolves $\dot{p}_i = -(j_{i+1/2} - j_{i-1/2})/\delta$. This is a linear system $\dot{\mathbf{p}} = \mathbf{F}\mathbf{p}$.

   1. Show that $\sum_i p_i\, \delta$ is conserved for every $\mathbf{p}$, the discrete counterpart of the divergence theorem Equation 38.45, and say which boundary condition of Section 38.4.4 the zero outer currents impose.
   2. Build $\mathbf{F}$ for $\theta = 1$, $g = \sqrt{2}$ and $\delta = 0.05$. Start from the two-mode mixture `p_t(x, 0.0)` on the cell centers and integrate to $t = 0.5$ with forward Euler at step $\Delta t = 10^{-3}$ and with backward Euler at step $\Delta t = 10^{-2}$ (one `np.linalg.solve` per step). Compare both results with `p_t(x, 0.5)`.
   3. Increase the forward-Euler step from 10^{-3} and find the largest step that stays stable on a run to $t = 5$, long enough for an unstable mode seeded by rounding error to grow. Relate this step to the most negative eigenvalue $-\nu$ of $\mathbf{F}$ through the bound $\Delta t < 2/\nu$ of Section 38.1.3.3. How does the limit change when δ is halved? Compare it with the condition of Exercise 1b.
   4. A grid with 100 cells per axis in d dimensions has 100^d unknowns. Evaluate this count for $d = 2$ and for $d = 784$, the pixel count of a 28×28 image. What does it imply for how diffusion models use the Fokker–Planck equation?

   Adapted from Särkkä and Solin (2019), Exercise 5.3.

9. **Divergence-free velocity fields.** ⊠ Let $\mathbf{R}$ be a constant antisymmetric $d \times d$ matrix.

   1. Show that $\nabla \cdot (p\, \mathbf{R} \nabla \log p) = 0$ for every smooth positive density p. Conclude that for every $\beta \in \mathbb{R}$ the ODE $\dot{\mathbf{x}} = \mathbf{u}_t(\mathbf{x}) + \beta \mathbf{R} \nabla \log p_t(\mathbf{x})$, with $\mathbf{u}_t$ from Equation 38.61, has the same marginals as the probability-flow ODE.
   2. Take the two-dimensional OU process at stationarity, $p_t = \mathcal{N}(\mathbf{0}, \frac{g^2}{2\theta}\mathbf{I})$ for all t, and $\mathbf{R} = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$. Show that the probability-flow velocity vanishes, and describe the trajectories of the modified ODE.
   3. Now put the rotation into the SDE itself: $d\mathbf{X} = (-\theta\mathbf{I} + \beta\mathbf{R})\mathbf{X}\, dt + g\, d\mathbf{W}$ in $\mathbb{R}^2$. Use the covariance equation of Exercise 2b to show that $\frac{g^2}{2\theta}\mathbf{I}$ is a stationary covariance for every β, so that the SDE has the same stationary density as in (b). Compute its stationary current, the current Equation 38.62 of this SDE. Why does the argument of Exercise 3a, which forces a stationary current to vanish, fail in two dimensions?
   4. Suppose that $\mathbf{f} = -\nabla U$, so that $\mathbf{u}_t$ is a gradient field. Among all velocity fields $\mathbf{w}$ whose continuity equation Equation 38.57 is solved by p_t, show that $\mathbf{u}_t$ has the smallest kinetic energy $\mathbb{E}_{p_t}\|\mathbf{w}\|^2$. The Benamou–Brenier formulation (Benamou and Brenier 2000) minimizes the same energy over the path p_t as well.

10. **Exact Gaussian reverse step.** ⊠ Let $p_t = \mathcal{N}(m_t, v_t)$ be a Gaussian OU marginal. Over

a step τ, the exact forward transition is $X_{t+\tau} \mid X_t = x \sim \mathcal{N}(\alpha_\tau x, \sigma_\tau^2)$ with $\alpha_\tau = e^{-\theta\tau}$ and $\sigma_\tau^2 = \frac{g^2}{2\theta}(1 - e^{-2\theta\tau})$, by the kernel Equation 38.43.

1. Use Gaussian conditioning to show that

$$X_t \mid X_{t+\tau} = y \;\sim\; \mathcal{N}\left(\frac{\sigma_\tau^2\, m_t + \alpha_\tau v_t\, y}{v_{t+\tau}},\; \frac{\sigma_\tau^2\, v_t}{v_{t+\tau}}\right). \tag{38.70}$$

   Here $v_{t+\tau} = \alpha_\tau^2 v_t + \sigma_\tau^2$ is the marginal variance at time $t + \tau$.

2. Expand the mean and the variance to first order in τ, and recover the reverse step Equation 38.66, including the full factor g^2 in front of the score.

11. code **Mismatched drift and noise.** ⊠ Consider a reverse-time sampler in the reversed clock $\bar{t} = T - t$ with drift $-\mathbf{f} + c\, g^2 \nabla \log p_t$ and noise λg. The family Equation 38.68 uses $c = (1 + \lambda^2)/2$.

    1. Take Gaussian data $p_0 = \mathcal{N}(0, v_0)$ and the OU forward process, so that $p_t = \mathcal{N}(0, v_t)$. Derive the ODE for the variance $V(\bar{t})$ of the sampler's state, and show that $V(\bar{t}) = v_{T-\bar{t}}$ solves it for every v_0 only when $c = (1 + \lambda^2)/2$.
    2. From the sign of the mismatch term in your ODE, predict whether the generated distribution is too wide or too narrow for $c = \frac{1}{2}$, $\lambda = 1$ (the ODE's correction with the SDE's noise) and for $c = 1$, $\lambda = 0$ (the SDE's correction without noise).
    3. Modify the `#fokker-planck-reverse-sde` cell to run both settings on the two-mode mixture. Report the recovered mode means and widths against the targets ± 2 and 0.5.

12. [extended] **Prior mismatch, ODE versus SDE.** ⊠⊠ Samplers start from the reference $q_T = \mathcal{N}(0, 1)$ instead of the exact p_T. This project compares how the reverse SDE and the reverse probability-flow ODE carry that mismatch to $t = 0$. Use unequal weights $w = (0.25, 0.75)$, which make p_T differ visibly from q_T for small T: pass `ws=np.array([0.25, 0.75])` to `p_t`, `score_t`, `mixture_cdf` and `sample_p0`, and `score=lambda x, t: score_t(x, t, ws=np.array([0.25, 0.75]))` to `pf_velocity`.

    1. Run Anderson's reverse SDE as in `#fokker-planck-reverse-sde` and the reverse-time probability-flow ODE with Heun steps, both from q_T, for $T \in \{0.25, 0.5, 1, 2, 3\}$ with step $\Delta t = 0.005$. Record the fraction of samples with $x < 0$, whose target is 0.25.
    2. Anderson's theorem holds for the whole process: started at $X_T = y$, the reverse SDE produces $X_0 \sim p(x_0 \mid x_T = y)$. Use this to predict the SDE's left-mode mass as an expectation over $y \sim q_T$ of a responsibility of p_T (Equation 38.54). For the ODE, use that its map from x_T to x_0 is increasing and carries p_T to p_0, and predict the left-mode mass from the standard normal CDF and the CDFs of p_T and p_0. Evaluate both predictions by quadrature and compare them with the measurements.
    3. Let q_0 be the law of a sampler's output. Use the data-processing inequality of Section 37.3.3 to prove that the ODE output satisfies $D_{\mathrm{KL}}(q_0 \| p_0) = D_{\mathrm{KL}}(q_T \| p_T)$, while the SDE output satisfies $D_{\mathrm{KL}}(q_0 \| p_0) \le D_{\mathrm{KL}}(q_T \| p_T)$.
    4. Compute $D_{\mathrm{KL}}(q_T \| p_T)$ and the SDE's $D_{\mathrm{KL}}(q_0 \| p_0)$ by quadrature for each T. The posterior $p(x_0 \mid x_T = y)$ is a two-component Gaussian mixture whose components follow from Exercise 10a, so q_0 is a one-dimensional integral over y. By what factor does the SDE shrink the mismatch at $T = 1$?

5. Repeat the measurement of (a) at $T = 1$ for the samplers of the family Equation 38.68 with $\lambda \in \{0, 0.5, 1, 2\}$. Which sampler would you use when the forward horizon T must be short? What limits how large λ can be at a fixed step size?

38.5 Score Matching and Diffusion Models

The previous section showed how to run noise back into data: integrate the reverse-time SDE, or the probability-flow ODE, from a Gaussian at the end of the forward process down to time zero. Both equations need the score $\nabla \log p_t$ of the noised data, the gradient of the log-density with respect to the point rather than a parameter. For the two-mode mixture of that section we could write the score down. In practice we have samples and no density. In this section the samples are 4,096 draws from the same mixture, standing in for the few million images of Chapter 17. How do we learn a field that we cannot evaluate at even a single point?

Fitting a density by maximum likelihood and differentiating it does not help, because a density built from a neural network needs a normalizing constant, an integral over all of space. Aapo Hyvärinen observed in 2005 that the score needs no normalizer, and that an integration by parts removes the unknown data score from the objective (Hyvärinen 2005). Pascal Vincent noticed something simpler still (Vincent 2011). The score of noised data is the average, over the clean points that could have produced a noisy point, of the direction back to the clean point. That direction is known for every training pair, so ordinary least squares learns the score from samples alone.

Diffusion models are trained this way. The standard diffusion loss, "predict the noise you added", is this regression with a particular weighting. A one-dimensional Gaussian turns each claim into a number we can check by hand before the mixture tests it.

No network is trained in this section. We fit a score once, by linear least squares on 17 fixed features at a single noise level, to see the regression work. Every sampler then runs on the exact score of the mixture, so that each error we measure is the sampler's own. The networks, with their errors, are the business of Chapter 17.

We first learn the score at one noise level, sample with it by the oldest score-based method, and see why one level is not enough. We then pass to all noise levels at once. Three results carry the section, and Chapter 17 relies on each: Vincent's theorem, the identification of the DDPM chain with the variance-preserving SDE of Section 38.3, and the identification of the DDIM sampler with Euler's method. Nothing later depends on the proof of Hyvärinen's identity, and the subsection on guidance can be skipped on a first reading. The next section learns a velocity instead of a score.

38.5.1 Learning the Score

We want a vector field $\mathbf{s}_\theta$, in practice a network with d inputs and d outputs, that approximates the score of the data, and we can only draw samples. The first task is an objective that compares the model with the true field and that we can evaluate from samples.

38.5.1.1 A Normalizer-Free Objective

A neural network defines a density most flexibly through an energy: an *energy-based model* is $p_\theta(\mathbf{x}) = e^{-E_\theta(\mathbf{x})}/Z_\theta$ with a network E_θ and the normalizer $Z_\theta = \int e^{-E_\theta(\mathbf{x})}\, d\mathbf{x}$ (LeCun et al. 2006). Maximum likelihood needs $\nabla_\theta \log Z_\theta$, a d-dimensional integral that changes with every update of θ. The score in $\mathbf{x}$ does not see it: Z_θ is a constant in $\mathbf{x}$, so $\nabla_{\mathbf{x}} \log p_\theta = -\nabla_{\mathbf{x}} E_\theta$.

So we compare fields instead of densities. With $\nabla \log p$ the score of the data, the **explicit score-matching** objective is the mean squared distance between the two fields, weighted by where the data are:

$$J_{\text{ESM}}(\theta) = \tfrac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim p}\left[\left\| \mathbf{s}_\theta(\mathbf{x}) - \nabla \log p(\mathbf{x}) \right\|^2 \right]. \tag{38.71}$$

It is zero exactly when the model field equals the data score wherever the data have mass. When $\mathbf{s}_\theta$ is the score of a density q, this is the Fisher divergence of Section 37.5.1, which also discusses what such a divergence can and cannot see; here $\mathbf{s}_\theta$ may be any vector field, since a network with d outputs need not be the gradient of anything. The objective has removed the normalizer, but it still contains $\nabla \log p$, the one field we do not know. Hyvärinen's integration by parts removes it.

Proposition (Hyvärinen's identity). *Let p be a smooth positive density and $\mathbf{s}_\theta$ a smooth vector field, with $p(\mathbf{x})\, \mathbf{s}_\theta(\mathbf{x}) \to \mathbf{0}$ as $\|\mathbf{x}\| \to \infty$ and all the expectations below finite. Then*

$$J_{\text{ESM}}(\theta) = \mathbb{E}_{\mathbf{x}\sim p}\left[\tfrac{1}{2} \|\mathbf{s}_\theta(\mathbf{x})\|^2 + \nabla \cdot \mathbf{s}_\theta(\mathbf{x}) \right] + C, \tag{38.72}$$

where $\nabla \cdot \mathbf{s}_\theta = \sum_i \partial s_i / \partial x_i$ is the divergence and $C = \frac{1}{2}\mathbb{E}_p \|\nabla \log p\|^2$ does not depend on θ.

Proof. Expand the square in Equation 38.71. The term $\frac{1}{2}\mathbb{E}\|\nabla \log p\|^2$ is the constant C, the term $\frac{1}{2}\mathbb{E}\|\mathbf{s}_\theta\|^2$ appears unchanged in Equation 38.72, and it remains to transform the cross term. In one dimension,

$$-\mathbb{E}_{x\sim p}\left[s_\theta(x)\, (\log p)'(x) \right] = -\int s_\theta(x)\, \frac{p'(x)}{p(x)}\, p(x)\, dx = -\int s_\theta(x)\, p'(x)\, dx,$$

and integrating by parts with the boundary term $[s_\theta\, p]_{-\infty}^{\infty} = 0$ leaves $+\int s'_\theta(x)\, p(x)\, dx = \mathbb{E}_p[s'_\theta]$. This is Stein's identity Equation 37.45 with $g = s_\theta$: the derivative has moved from the unknown density onto the model. In d dimensions, apply the same step to each coordinate i, with s_i in place of s_θ and $\partial_i p$ in place of p', integrating over x_i first; Fubini's theorem allows this under the stated integrability. Summed over i, the cross term becomes $\mathbb{E}_p[\sum_i \partial_i s_i] = \mathbb{E}_p[\nabla \cdot \mathbf{s}_\theta]$. ■

Every term on the right of Equation 38.72 is an expectation under p of something we can evaluate for a given model, so a sample average estimates it, with no Z_θ and no $\nabla \log p$. The right-hand side is called **implicit score matching**.

A Gaussian shows that the identity works. Let the data be $x \sim \mathcal{N}(0, v)$, whose score is $-x/v$, and take the one-parameter model $s_w(x) = -x/w$. Then $\frac{1}{2}s_w^2 + s'_w = x^2/(2w^2) - 1/w$, so the implicit objective is $J(w) = v/(2w^2) - 1/w$. Its derivative $-v/w^3 + 1/w^2$ vanishes at $w = v$: the objective never saw the data score and still returns the variance. The constant

agrees as well. At the minimum $J(v) = -1/(2v)$, and $C = \frac{1}{2}\mathbb{E}[x^2/v^2] = 1/(2v)$, so the explicit objective is $J(v) + C = 0$ there, as it must be when the model matches the data. In practice the expectation is a sample average. The cell replaces v in J by the mean of x^2 over 10,000 draws from $\mathcal{N}(0, 2)$ and minimizes over a grid of w.

```
rng = np.random.default_rng()
v = 2.0
x = rng.normal(0.0, np.sqrt(v), 10000)        # data x ~ N(0, v)
w = np.linspace(0.5, 4.0, 701)                # model s_w(x) = -x / w
J_ism = (x**2).mean() / (2 * w**2) - 1 / w    # E[s_w^2 / 2 + s_w']
print(f'implicit score matching picks w = {w[J_ism.argmin()]:.2f}  (v = {v})')
```

```
implicit score matching picks w = 2.02  (v = 2.0)
```

The sample objective is minimized at the sample mean of x^2, which lies within a few hundredths of $v = 2$ for 10,000 draws. The two terms of Equation 38.72 pull against each other, and the Gaussian shows how. The divergence term $-1/w$ rewards a field that points inward at the samples, a sink where the data are, and rewards it more the steeper the sink. The squared norm $x^2/(2w^2)$ stops the field from growing without bound. The balance sits at the true variance.

Implicit score matching has two limits. The proof needs the boundary term to vanish. For data on a bounded interval it does not, and the implicit objective then picks the wrong field (exercise 1). The second limit is cost. The divergence is the trace of the Jacobian, $\nabla \cdot \mathbf{s}_\theta = \operatorname{tr}(\partial \mathbf{s}_\theta / \partial \mathbf{x})$. Computing it exactly takes one reverse-mode pass per row of the Jacobian, d passes in all, and d is in the millions for images. Hutchinson's estimator (Section 38.2.3) replaces the trace by one vector-Jacobian product per random direction, but it adds variance. Denoising score matching avoids the trace altogether.

38.5.1.2 Denoising Score Matching

We want an objective without the trace, and we want the score where a sampler will need it, which includes points near but not on the data. Matching the score of *noised* data meets both requirements. Perturb each sample with Gaussian noise of scale σ:

$$\tilde{\mathbf{x}} = \mathbf{x} + \sigma\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \qquad \text{so} \quad p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}};\, \mathbf{x},\, \sigma^2\mathbf{I}).$$

The noised marginal $p_\sigma(\tilde{\mathbf{x}}) = \int p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\, p(\mathbf{x})\, d\mathbf{x}$ is the data density blurred by a Gaussian, and it approaches p as σ shrinks. Its score is as unknown as the data score. The score of the *conditional*, however, follows from the logarithm of a Gaussian density:

$$\nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \frac{\mathbf{x} - \tilde{\mathbf{x}}}{\sigma^2} = -\frac{\epsilon}{\sigma}. \tag{38.73}$$

This vector points from the noisy observation back to the clean sample that produced it, and we know it for every pair we generate. **Denoising score matching** (DSM) uses it as a regression target:

$$J_{\text{DSM}}(\theta) = \tfrac{1}{2}\, \mathbb{E}_{\mathbf{x}\sim p,\, \tilde{\mathbf{x}}\sim p_\sigma(\cdot\mid\mathbf{x})} \left[\left\| \mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \right\|^2 \right]. \tag{38.74}$$

The loss asks the network to predict, from the noisy point alone, the direction back to a clean point that it cannot see. The target depends on the clean point, while the score we want depends only on the noisy one. Least squares connects the two, because it fits the target's conditional mean. The following lemma states that fact; the next section uses it again with a velocity in place of the score.

Lemma (regression to the conditional mean). *Let (X, Y) be jointly distributed with $\mathbb{E}\|Y\|^2 < \infty$, and let $\mathbf{m}(X) = \mathbb{E}[Y \mid X]$. Then for every measurable $\mathbf{v}$ with $\mathbb{E}\|\mathbf{v}(X)\|^2 < \infty$,*

$$\mathbb{E}\,\|\mathbf{v}(X) - Y\|^2 = \mathbb{E}\,\|\mathbf{v}(X) - \mathbf{m}(X)\|^2 + \mathbb{E}\,\|Y - \mathbf{m}(X)\|^2 . \tag{38.75}$$

The second term does not involve $\mathbf{v}$: minimizing a least-squares loss against the noisy target Y is the same problem as minimizing it against the conditional mean $\mathbf{m}(X)$, up to an additive constant.

Proof. Insert $\pm\mathbf{m}(X)$ and expand. The cross term is $2\,\mathbb{E}\left[(\mathbf{v}(X) - \mathbf{m}(X))^\top(\mathbf{m}(X) - Y)\right]$. By the tower rule $\mathbb{E}[Z] = \mathbb{E}[\mathbb{E}[Z \mid X]]$ (Section 36.2) we may first average over Y with X fixed, and $\mathbb{E}[\mathbf{m}(X) - Y \mid X] = \mathbf{0}$ makes the cross term vanish. ■

Proposition (Vincent's theorem). *Let the noising kernel $p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$ be differentiable in $\tilde{\mathbf{x}}$, with gradient and integral over $\mathbf{x}$ interchangeable (both hold for the Gaussian kernel), and let the DSM target and $\mathbf{s}_\theta(\tilde{\mathbf{x}})$ have finite second moments. Then*

$$J_{\mathrm{DSM}}(\theta) = \tfrac{1}{2}\,\mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma}\left[\,\|\mathbf{s}_\theta(\tilde{\mathbf{x}}) - \nabla \log p_\sigma(\tilde{\mathbf{x}})\|^2\right] + C,$$

with C independent of θ. The right-hand side is the explicit objective Equation 38.71 for p_σ, so denoising and explicit score matching on the noised data have the same minimizers and the same gradients.

Proof. Apply the lemma with $X = \tilde{\mathbf{x}}$ and $Y = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})$. It remains to identify the conditional mean of Y, and the identification is the content of the theorem: *the marginal score is the posterior mean of the conditional scores*. Writing the posterior $p(\mathbf{x} \mid \tilde{\mathbf{x}})$ by Bayes' rule and then exchanging gradient and integral,

$$\mathbb{E}\,[Y \mid \tilde{\mathbf{x}}] = \int \frac{\nabla_{\tilde{\mathbf{x}}}\, p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})}{p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})}\,\frac{p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\,p(\mathbf{x})}{p_\sigma(\tilde{\mathbf{x}})}\,d\mathbf{x} = \frac{\nabla_{\tilde{\mathbf{x}}} \int p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x})\,p(\mathbf{x})\,d\mathbf{x}}{p_\sigma(\tilde{\mathbf{x}})} = \nabla \log p_\sigma(\tilde{\mathbf{x}}).$$

■

The proof used no property of the Gaussian beyond differentiability. Gaussian noise is chosen because its conditional score Equation 38.73 is the noise itself, scaled by $-1/\sigma$. Predicting the score and predicting the noise ϵ are therefore the same task, and a network ϵ_θ that predicts the noise defines the score model $\mathbf{s}_\theta = -\epsilon_\theta/\sigma$. This *noise-prediction* form is the one diffusion models use.

Example: the Gaussian. Let the data be $x \sim \mathcal{N}(0, v)$ again, noised at level σ. The noised marginal is $p_\sigma = \mathcal{N}(0, v + \sigma^2)$, with score $-\tilde{x}/(v + \sigma^2)$. Bayes' rule for two Gaussians gives the posterior of the clean point,

$$x \mid \tilde{x} \;\sim\; \mathcal{N}\Big(\frac{v}{v + \sigma^2}\,\tilde{x},\; \frac{v\sigma^2}{v + \sigma^2}\Big),$$

a shrinkage of the noisy point toward the data mean, with a spread that does not depend on $\tilde{x}$. Every claim of this subsection becomes a formula. The conditional mean of the target $Y = (x - \tilde{x})/\sigma^2$ is $(\frac{v}{v+\sigma^2} - 1)\tilde{x}/\sigma^2 = -\tilde{x}/(v + \sigma^2)$, the marginal score, as Vincent's theorem says. The best possible mean squared error, the second term of Equation 38.75, is the posterior variance of the target, $\mathrm{Var}(x \mid \tilde{x})/\sigma^4 = \frac{1}{\sigma^2} - \frac{1}{v+\sigma^2}$. With $v = 1$ and $\sigma = 0.5$ the floor is $4 - 0.8 = 3.2$, and the posterior mean of the clean point is $0.8\,\tilde{x}$. The cell checks both numbers by Monte Carlo on 100,000 pairs.

```
x, eps = rng.standard_normal((2, 100000))            # v = 1
xt, y = x + 0.5 * eps, -eps / 0.5                     # sigma = 0.5; DSM target y
print(f'mean squared error of the conditional mean -xt/1.25: {((y + xt /
↪  1.25)**2).mean():.3f}'
      '  (floor 3.2)')
print(f'least-squares slope of x on xt: {(xt * x).mean() / (xt**2).mean():.3f}  (0.8)')
```

```
mean squared error of the conditional mean -xt/1.25: 3.196  (floor 3.2)
least-squares slope of x on xt: 0.799  (0.8)
```

The best regression function has mean squared error 3.2 on the targets, not 0, because many clean points explain one noisy point and the regression can only return their average. The slope 0.80 is the shrinkage factor $v/(v + \sigma^2)$. Both observations hold for any data distribution and give two corollaries.

- **Tweedie's formula** (Efron 2011). Rearranging $\mathbb{E}[(\mathbf{x} - \tilde{\mathbf{x}})/\sigma^2 \mid \tilde{\mathbf{x}}] = \nabla \log p_\sigma(\tilde{\mathbf{x}})$ gives $\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] = \tilde{\mathbf{x}} + \sigma^2 \nabla \log p_\sigma(\tilde{\mathbf{x}})$. The best denoiser under squared error moves the noisy point one step of size σ^2 up the score of the noised data (Figure 38.22). For the Gaussian it reads $\tilde{x} - \sigma^2\tilde{x}/(v + \sigma^2) = 0.8\,\tilde{x}$, the slope just measured.
- **The floor.** At the optimum the mean squared error $2J_{\mathrm{DSM}}$ equals $\mathbb{E}\|Y - \mathbf{m}(X)\|^2$, the average posterior variance of the target, which is 3.2 for the Gaussian above. An observed loss also contains the fit's own error, so it should be compared with this floor, not with zero.

The floor also says what goes wrong at a very small noise level. As $\sigma \to 0$ it grows like $1/\sigma^2$. The target $-\epsilon/\sigma$ becomes large, while the score it averages to keeps a finite mean square. A single small noise level therefore gives a poorly conditioned regression, whose targets are mostly noise.

Differentiating Tweedie's formula once more limits how sharply a noised density can curve, a bound that guidance will break at the end of the section.

Proposition (curvature of a noised density). *For any data distribution and any $\sigma > 0$,*

$$\nabla^2 \log p_\sigma(\tilde{\mathbf{x}}) = -\frac{\mathbf{I}}{\sigma^2} + \frac{1}{\sigma^4}\,\mathrm{Cov}\,[\mathbf{x} \mid \tilde{\mathbf{x}}]\,, \tag{38.76}$$

so no eigenvalue of $-\nabla^2 \log p_\sigma$ exceeds $1/\sigma^2$.

Proof. By Tweedie's formula $\nabla \log p_\sigma(\tilde{\mathbf{x}}) = (\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] - \tilde{\mathbf{x}})/\sigma^2$, so it remains to differentiate the posterior mean. Bayes' rule gives $\nabla_{\tilde{\mathbf{x}}} \log p(\mathbf{x} \mid \tilde{\mathbf{x}}) = \nabla_{\tilde{\mathbf{x}}} \log p_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) - \nabla \log p_\sigma(\tilde{\mathbf{x}}) = (\mathbf{x} - \mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}])/\sigma^2$, hence the Jacobian of $\mathbb{E}[\mathbf{x} \mid \tilde{\mathbf{x}}] = \int \mathbf{x}\, p(\mathbf{x} \mid \tilde{\mathbf{x}})\, d\mathbf{x}$ is $\mathrm{Cov}[\mathbf{x} \mid \tilde{\mathbf{x}}]/\sigma^2$. A covariance is positive semidefinite. ■

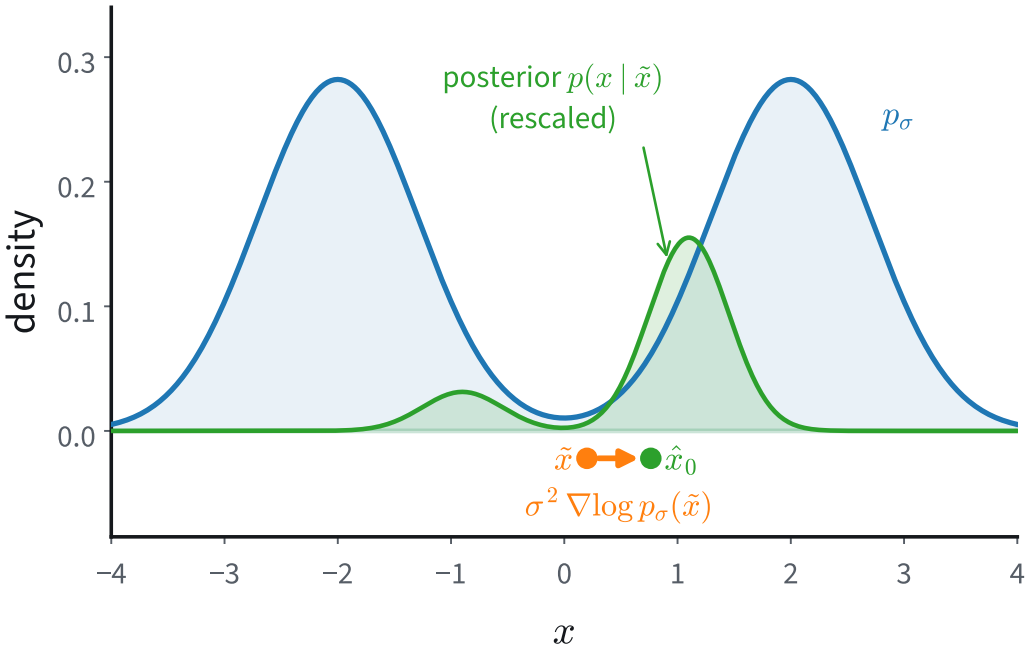


Figure 38.22: Tweedie's formula. A noisy observation $\tilde{x}$ lies in the low-density valley of the smoothed mixture p_σ. Its posterior clean-data distribution is bimodal but asymmetric, and the step $\tilde{x} + \sigma^2 \nabla \log p_\sigma(\tilde{x})$ lands on the posterior mean $\hat{x}_0$, the optimal denoiser.

For the Gaussian the right-hand side is $-\frac{1}{\sigma^2} + \frac{v}{\sigma^2(v+\sigma^2)} = -\frac{1}{v+\sigma^2}$, the curvature of $\log \mathcal{N}(0, v+\sigma^2)$.

38.5.1.3 A Least-Squares Score in One Dimension

The Gaussian has a linear score, which a one-parameter model can represent. Does least squares on noisy pairs also recover a curved score? The data are the mixture of the previous section, $p = \frac{1}{2}\mathcal{N}(-2, 0.5^2) + \frac{1}{2}\mathcal{N}(2, 0.5^2)$, noised at $\sigma = 0.5$. Noising adds σ^2 to each component's variance, so p_σ is again a mixture, with component variance $0.25+0.25 = 0.5$. By Equation 38.55, a symmetric mixture with means $\pm\mu$ and common component variance v has the score $(\mu \tanh(\mu x/v) - x)/v$. With $\mu = 2$ and $v = 0.5$ the target curve is

$$\nabla \log p_\sigma(\tilde{x}) = 4 \tanh(4\tilde{x}) - 2\tilde{x}.$$

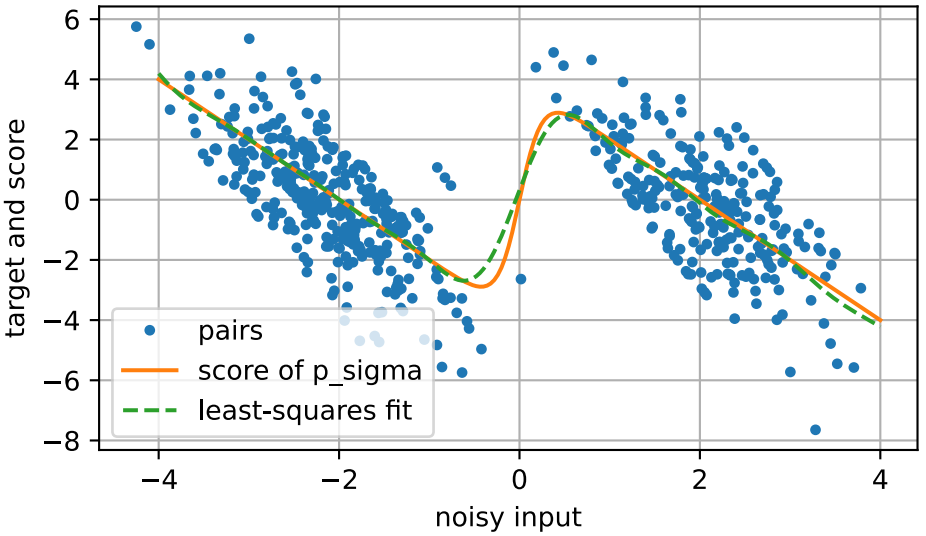
Near each mode it is close to $-(\tilde{x} \mp 2)/0.5$, the linear score of a single component. Between the modes it rises from about -2.9 at $\tilde{x} = -0.45$ to $+2.9$ at $\tilde{x} = 0.45$, pushing points out of the valley.

The model is not a network. It is a linear combination of 17 fixed features, sixteen Gaussian bumps of width 0.6 centered on $[-4.5, 4.5]$ plus the identity x. We choose it because fitting the DSM loss Equation 38.74 is then ordinary least squares on the pairs $(\tilde{x}, -\epsilon/\sigma)$, which `np.linalg.lstsq` solves exactly: any gap between the fit and the score is sampling error, with no optimizer to blame. Noise costs nothing to draw, so each of the 4,096 clean points is noised four times. The cell fits the model on these 16,384 pairs, compares it with the analytic score, and evaluates both far outside the data at $x = 8$. The mean squared errors it prints are $2J_{\text{DSM}}$, the quantity the regression lemma splits.

```
n, sigma = 4096, 0.5
x = rng.normal(4.0 * rng.integers(0, 2, n) - 2.0, 0.5)         # x ~ p, the mixture
def mixture_score(q, var, means=(-2.0, 2.0)):                   # analytic score, any level
    logw = np.stack([-(q - m)**2 / (2 * var) for m in means])
    w = np.exp(logw - logw.max(0))                               # responsibilities, stable
    return (w * np.stack([(m - q) / var for m in means])).sum(0) / w.sum(0)
xx = np.repeat(x, 4)                                             # four noisy copies per point
eps = rng.standard_normal(xx.size)
xt, y = xx + sigma * eps, -eps / sigma                           # the (input, target) pairs
centers = np.linspace(-4.5, 4.5, 16)
def features(q):                                                # identity + 16 bumps, width 0.6
    return np.column_stack([q, np.exp(-(q[:, None] - centers)**2 / 0.72)])
w_ls = np.linalg.lstsq(features(xt), y, rcond=None)[0]          # least squares = the
↪ regression
s_hat = lambda q: features(q) @ w_ls
g = np.linspace(-8, 8, 16001); dg = g[1] - g[0]
p_sigma = 0.5 * (np.exp(-(g + 2)**2) + np.exp(-(g - 2)**2)) / np.sqrt(np.pi)  # var 0.5 each
floor = 1 / sigma**2 - (p_sigma * mixture_score(g, 0.5)**2).sum() * dg
excess = (p_sigma * (s_hat(g) - mixture_score(g, 0.5))**2).sum() * dg
print(f'mean squared error: fit {((s_hat(xt) - y)**2).mean():.2f}, exact score '
      f'{((mixture_score(xt, 0.5) - y)**2).mean():.2f}, population floor {floor:.2f}')
print(f'excess risk {excess:.3f}; fit at x = 8: {s_hat(np.array([8.0]))[0]:+.1f}, '
      f'exact score {4 * np.tanh(32) - 16:.0f} (no pairs there)')
grid = np.linspace(-4, 4, 201)
```

```
d2l.plot([xt[:600], grid, grid], [y[:600], 4 * np.tanh(4 * grid) - 2 * grid, s_hat(grid)],
         'noisy input', 'target and score', fmts=('.', '-', '--'),
         legend=['pairs', 'score of p_sigma', 'least-squares fit'], figsize=(5.5, 3))
```

```
mean squared error: fit 2.14, exact score 2.15, population floor 2.11
excess risk 0.012; fit at x = 8: -7.1, exact score -12 (no pairs there)
```



The plot shows all that the regression sees: a cloud of targets $-\epsilon/\sigma$ that spreads over several units at every $\tilde{x}$. Through the cloud runs its conditional mean, the score of p_σ, and the least-squares fit lies on it. The printout follows the regression lemma. The fit and the exact score have mean squared errors of about 2.1 to 2.2 on these pairs, and they agree with each other to within a few hundredths. The population floor $\frac{1}{\sigma^2} - \mathbb{E}_{p_\sigma}(\nabla \log p_\sigma)^2$, computed by quadrature, is 2.11; it is the mean squared vertical spread of the cloud around the curve, and the few hundredths by which the printed averages miss it are the sampling error of an average over 16,384 pairs (exercise 4). The *excess risk*, the first term of the lemma, measures how far the fit lies from the score. It is around 0.01, under one percent of the floor.

The last number shows the limit of the method. No noisy pair lands beyond $|\tilde{x}| \approx 5$, so at $x = 8$ the fit is whatever the basis extrapolates to. The printed value need not be near the exact -12. Across fresh draws it moves by many units, and it can even be positive, pointing away from the data. Least squares constrains the fit only where pairs fall, so a sampler that starts far from the data meets an arbitrary field there.

38.5.2 Sampling with a Score

The previous section's samplers need the score at every noise level of a forward process. With the score at one level only, or the exact score of p itself, we can still draw samples, by a method that needs no forward process at all.

38.5.2.1 Langevin Dynamics

Picture a particle that drifts uphill on the log-density and is jostled by noise. The drift pulls it toward high density, and the noise keeps it from collapsing onto a mode. **Langevin dynamics** is the SDE

$$d\mathbf{X} = \tfrac{1}{2}\nabla \log p(\mathbf{X})\, dt + d\mathbf{W}, \tag{38.77}$$

whose drift is half the score and whose noise is a standard Brownian motion $\mathbf{W}$. The claim is that p is its stationary law.

For the Gaussian $p = \mathcal{N}(0, v)$, with score $-x/v$, we already know this. Then Equation 38.77 reads $dX = -\frac{X}{2v}\, dt + dW$, an Ornstein–Uhlenbeck process. An OU process $dX = -\theta X\, dt + g\, dW$ has the stationary law $\mathcal{N}(0, g^2/(2\theta))$ (Section 38.3.3), and here $\theta = 1/(2v)$ and $g = 1$ give the variance v. The proposition extends this to smooth targets whose score grows at most linearly.

Proposition (stationarity). *Let p be a smooth positive density whose score $\nabla \log p$ is locally Lipschitz and grows at most linearly, so that Equation 38.77 has a unique solution that does not explode and its Fokker–Planck equation has a unique solution for each initial density. If $\mathbf{X}_0 \sim p$, then $\mathbf{X}_t \sim p$ for all $t \geq 0$.*

Proof. For an SDE $d\mathbf{X} = \mathbf{b}\, dt + d\mathbf{W}$ with unit diffusion, the density q_t of $\mathbf{X}_t$ solves the Fokker–Planck equation $\partial_t q = -\nabla \cdot (q\, \mathbf{b}) + \frac{1}{2}\Delta q$ (Section 38.4.2). Substitute $\mathbf{b} = \frac{1}{2}\nabla \log p$ and $q = p$, and use $p\, \nabla \log p = \nabla p$:

$$-\nabla \cdot \left(\tfrac{1}{2}\, p\, \nabla \log p\right) + \tfrac{1}{2}\Delta p = -\tfrac{1}{2}\nabla \cdot (\nabla p) + \tfrac{1}{2}\Delta p = 0.$$

So $q_t \equiv p$ solves the equation with the initial density p. Uniqueness turns "p solves the equation" into "the law of $\mathbf{X}_t$ is p". ■

The drift transports mass up the log-density at exactly the rate at which the noise spreads it out. Stationarity says nothing yet about a chain started elsewhere, and a sampler always starts elsewhere. Under conditions of this kind the law of $\mathbf{X}_t$ does converge to p from every starting point (Roberts and Tweedie 1996). We do not prove it; how long the convergence takes is what the experiment below measures.

To sample, we discretize. The Euler–Maruyama step of Equation 38.77 with step h is the **Langevin sampler**

$$\mathbf{x} \leftarrow \mathbf{x} + \tfrac{h}{2}\, \mathbf{s}(\mathbf{x}) + \sqrt{h}\, \xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

which needs only a score $\mathbf{s}$, exact or learned. The discretized chain does not inherit the stationarity of the SDE. For the Gaussian the step is the linear recursion $x \leftarrow (1 - \frac{h}{2v})\, x + \sqrt{h}\, \xi$, whose stationary variance V solves $V = (1 - \frac{h}{2v})^2 V + h$. Hence $V = v/(1 - h/(4v))$, and the sampler overstates the variance by about $h/4$ (exercise 2 measures this).

The cell runs the sampler on the mixture $\frac{1}{2}\mathcal{N}(\pm 2, 0.5^2)$ with its exact score, $h = 0.01$ and 2,000 steps, twice: from a broad start $\mathcal{N}(0, 3^2)$, and with every chain started at the left mean. Before reading the output, guess how many of the 10,000 chains of the second run reach the right mode.

```
def langevin(q, h, steps, var=0.25):          # var: component variance of p
    for _ in range(steps):
        xi = rng.standard_normal(q.shape)
        q = q + 0.5 * h * mixture_score(q, var) + np.sqrt(h) * xi
    return q
warm = langevin(rng.normal(0.0, 3.0, 10000), 0.01, 2000)
print(f'spread-out start: P(X > 0) = {(warm > 0).mean():.3f}, '
      f'E[X^2] = {(warm**2).mean():.2f}  (truth 0.5, 4.25)')
cold = langevin(np.full(10000, -2.0), 0.01, 2000)
print(f'one-mode start:   P(X > 0) = {(cold > 0).mean():.3f}  (crossed)')
```

```
spread-out start: P(X > 0) = 0.497, E[X^2] = 4.21  (truth 0.5, 4.25)
one-mode start:   P(X > 0) = 0.012  (crossed)
```

About one chain in a hundred has crossed. In the valley the density is tiny and the drift points back to the mode the chain came from. Crossing needs a rare run of noise in one direction, and the waiting time grows exponentially with the height of the barrier in $-\log p$.

This changes how the first line must be read. From the broad start the second moment matches the true $\mathbb{E}X^2 = 4 + 0.25 = 4.25$ to within a few hundredths, the sampling error of 10,000 chains (the step-size bias $h/4 \approx 0.0025$ is far below it): each chain has equilibrated inside the mode it fell into. The even split between the modes, however, is no evidence for the sampler. Start and target are both symmetric, so half the chains end on each side whatever happens in between. The proportions are those of the start, not of p: with mode weights 0.8 and 0.2 the same broad start would still split about evenly. Our symmetric mixture cannot show this failure; Section 17.1 measures it.

The time a chain needs to forget where it started is its *mixing time*. Slow mixing is a property of the SDE itself, so a smaller step does not cure it. A smaller step removes only the discretization bias.

38.5.2.2 Limits of a Single Noise Level

A score at one noise level has now failed in three ways. The learned score is arbitrary where no noisy pair landed, so a sampler that starts far from the data meets a field that may push it away. Langevin dynamics with the correct score crosses the valley between the modes only rarely, so the mode weights of its samples are those of its start. And the score itself hardly records the weights: two mixtures of well-separated modes that differ only in their weights have almost the same score wherever they have mass (Section 37.5.1).

Learning the score at a ladder of noise levels addresses all three. At a large σ the noised density has no valley, and the noisy pairs cover the whole region a sampler visits. For the mixture, p_σ has component variance $0.25 + \sigma^2$ and is unimodal once its component standard deviation reaches the distance 2 from the center to each mean, at $\sigma \geq 1.94$. As σ shrinks the detail returns, and a sampler that has already spread its mass across both modes only needs to refine it.

Annealed Langevin dynamics (Song and Ermon 2019) runs the sampler down such a ladder, each level with its own score. Exercise 3 moves chains out of a single mode this way, and Section 17.2 shows that the ladder recovers unequal weights. A ladder of separately chosen levels leaves open how to space the levels and how to move between them. A forward process settles both: it supplies a continuum of noise levels, and its reverse is the sampler.

38.5.3 Scores Across Noise Levels

From here on the model is one field $\mathbf{s}_\theta(\mathbf{x}, t)$ that takes the noise level as a second input and is trained at every level at once, by the loss of the next subsection. The samplers are those of the previous section.

38.5.3.1 Forward Processes and the Noise-Conditional Loss

DSM needs the conditional law of the noised point given the clean one, and it needs that law to be Gaussian so that the target is explicit. Every SDE whose drift is linear in $\mathbf{x}$ and whose noise amplitude does not depend on $\mathbf{x}$ has Gaussian transition kernels, as Equation 38.34 showed for the OU process. The two standard forward processes of Section 38.3 are of this kind, and their kernels can be written once for both as

$$p_t(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\alpha_t \mathbf{x}_0,\, \sigma_t^2 \mathbf{I}), \qquad \text{i.e.} \qquad \mathbf{x}_t = \alpha_t \mathbf{x}_0 + \sigma_t \epsilon.$$

The coefficient α_t shrinks the signal and σ_t scales the noise. The two processes differ only in these two numbers.

- The **variance-exploding (VE)** SDE $d\mathbf{X} = \sqrt{\frac{d}{dt}\sigma^2(t)}\; d\mathbf{W}$, with $\sigma^2(t)$ increasing and $\sigma(0) = 0$, adds noise without shrinking the data: $\alpha_t = 1$ and $\sigma_t = \sigma(t)$. It is the continuous version of the ladder of the previous subsection.
- The **variance-preserving (VP)** SDE is an Ornstein–Uhlenbeck process with a time-dependent rate,

$$d\mathbf{X} = -\tfrac{1}{2}\beta(t)\,\mathbf{X}\,dt + \sqrt{\beta(t)}\; d\mathbf{W},$$

  with $\alpha_t = e^{-\frac{1}{2}\int_0^t \beta(r)\,dr}$ and $\sigma_t^2 = 1 - \alpha_t^2$, so that unit-variance data keep unit variance at every t (Section 38.3).

For standard Gaussian data, $\mathcal{N}(\mathbf{0}, \mathbf{I})$, the VP process does nothing visible: $\mathbf{x}_t \sim \mathcal{N}(\mathbf{0}, (\alpha_t^2 + \sigma_t^2)\mathbf{I}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$ at every t, the score is $-\mathbf{x}$, and the probability-flow drift (displayed below) is $-\frac{1}{2}\beta\mathbf{x} - \frac{1}{2}\beta(-\mathbf{x}) = \mathbf{0}$. Every sample of the deterministic flow sits still. This case returns when we sample with large steps.

With a Gaussian kernel at every t, the conditional score is $-\epsilon/\sigma_t$, and DSM applies at each level. Averaging over levels with a weight $w(t) > 0$ gives the **noise-conditional DSM loss**

$$\mathcal{L}(\theta) = \tfrac{1}{2}\,\mathbb{E}_t\, w(t)\, \mathbb{E}_{\mathbf{x}_0,\,\epsilon}\left[\left\| \mathbf{s}_\theta(\alpha_t \mathbf{x}_0 + \sigma_t \epsilon, t) + \frac{\epsilon}{\sigma_t} \right\|^2\right], \tag{38.78}$$

with t drawn uniformly from the training range. Each draw of $(t, \mathbf{x}_0, \epsilon)$ is one regression pair.

The weight decides which levels the network serves best, not what it learns. For any positive w the population minimizer is $\mathbf{s}(\cdot, t) = \nabla \log p_t$ at every t, by Vincent's theorem at each level, provided the network can represent it. The floor at level t, by the same argument as for the Gaussian, is the mean squared error $d/\sigma_t^2 - \mathbb{E}\|\nabla \log p_t\|^2$ (exercise 4), which grows like d/σ_t^2 at small σ_t. An unweighted loss is therefore dominated by its least noisy levels.

Two weights are standard. The discrete-time model of the next subsection uses $w(t) = \sigma_t^2$, which removes the $1/\sigma_t^2$ growth. The *likelihood weighting* $w = g^2$, the squared diffusion coefficient of the forward SDE, ties the score error to the model's likelihood. With this weight, T times the excess of the loss over its floor, plus the KL divergence from p_T to the Gaussian the sampler starts from, bounds the KL divergence from the data to the reverse-SDE model (exercise 9). Here T is the length of the time range. The floor must be subtracted: for the VP process under this weight it grows like d/t as $t \to 0$ and is not integrable, which is also why training starts at a small $\varepsilon > 0$ (exercise 5).

Generation follows the previous section, with the learned field in place of the exact score. For a forward SDE $d\mathbf{X} = \mathbf{f}(\mathbf{X}, t)\, dt + g(t)\, d\mathbf{W}$ on $[0, T]$, we start from the Gaussian that approximates p_T and integrate either the reverse-time SDE Equation 38.67 or the probability-flow ODE Equation 38.64 from $t = T$ down to 0:

$$d\mathbf{X} = \big[\mathbf{f} - g^2\, \mathbf{s}_\theta(\mathbf{X}, t)\big]\, dt + g\, d\bar{\mathbf{W}}, \qquad \frac{d\mathbf{x}}{dt} = \mathbf{f} - \tfrac{1}{2} g^2\, \mathbf{s}_\theta(\mathbf{x}, t), \tag{38.79}$$

where $\bar{\mathbf{W}}$ is a Brownian motion running backward in time. For VP, $\mathbf{f} = -\frac{1}{2}\beta\mathbf{x}$ and $g^2 = \beta$. A generative model is thus a triple: a forward process, a learned score, and a sampler. Figure 38.23 shows the triple at work in two dimensions, the companion of the forward and reverse densities of Figure 38.21.

38.5.3.2 DDPM and the Variance-Preserving SDE

The Denoising Diffusion Probabilistic Model (Ho et al. 2020) is written in discrete time and looks at first like a separate construction. It is a Markov chain of $N = 1000$ noising steps with a schedule $\beta_1, \dots, \beta_N \in (0, 1)$,

$$\mathbf{x}_t = \sqrt{1 - \beta_t}\, \mathbf{x}_{t-1} + \sqrt{\beta_t}\, \epsilon_t, \qquad \epsilon_t \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \text{ i.i.d.}, \tag{38.80}$$

which shrinks the state slightly and adds a little noise at each step. In this subsection and the next, t counts levels, $t = 0, 1, \dots, N$, and level t stands for the time $t\,\Delta$ of the SDE, with $\Delta = T/N$. With $\beta_t = \beta((t-1)\Delta)\,\Delta$ the step agrees to first order in β_t with the Euler–Maruyama step $(1 - \frac{1}{2}\beta_t)\,\mathbf{x}_{t-1} + \sqrt{\beta_t}\,\epsilon_t$ of the VP-SDE (Section 38.3.4.1). More is true: it is an *exact* transition of the VP process, over a slightly longer interval. Apply the VP kernel of the previous subsection not to $[0, t]$ but to an interval on which $\int \beta(r)\, dr = -\log(1 - \beta_t)$. Its mean coefficient is $e^{-\frac{1}{2}\int \beta} = \sqrt{1 - \beta_t}$ and its variance is $1 - (1 - \beta_t) = \beta_t$, the law of Equation 38.80 given $\mathbf{x}_{t-1}$. For the largest step of the standard schedule, $\beta_t = 0.02$, the interval has $\int \beta = 0.0202$ instead of 0.02. The DDPM chain is therefore the VP process observed at discrete times, and its marginal at every level has a closed form. This is the result of the subsection to remember; the DDPM training loss follows from it in two lines.

Proposition (the $\bar{\alpha}$-marginal). *Let $\bar{\alpha}_t = \prod_{s=1}^{t}(1 - \beta_s)$. Then conditionally on $\mathbf{x}_0$,*

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\, \epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{38.81}$$

Proof. Induction on t; the case $t = 0$ holds with $\bar{\alpha}_0 = 1$. Assume Equation 38.81 at $t - 1$ and substitute it into Equation 38.80:

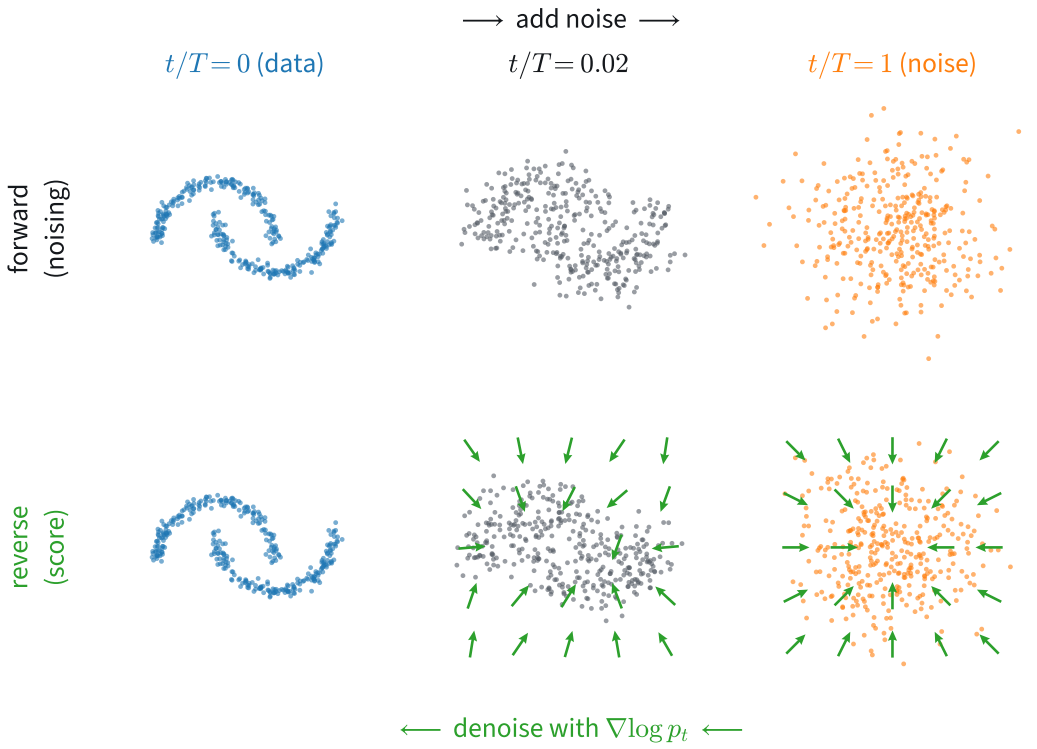


Figure 38.23: A two-moons distribution under the VP forward process at $t/T = 0, 0.02$, and 1 (top row, left to right), approaching an isotropic Gaussian. The bottom row traverses the same marginals in reverse, right to left; short arrows show the direction of the exact score $\nabla \log p_t$ of the noised data. At high noise the score is the pull toward the origin; at low noise it points toward the data.

$$\mathbf{x}_t = \sqrt{(1-\beta_t)\bar{\alpha}_{t-1}}\,\mathbf{x}_0 + \sqrt{(1-\beta_t)(1-\bar{\alpha}_{t-1})}\,\bar{\epsilon} + \sqrt{\beta_t}\,\epsilon_t,$$

with $\bar{\epsilon}$ and ϵ_t independent standard Gaussians. A sum of independent Gaussians is Gaussian with the sum of the variances, $(1-\beta_t)(1-\bar{\alpha}_{t-1}) + \beta_t = 1-(1-\beta_t)\bar{\alpha}_{t-1} = 1-\bar{\alpha}_t$, and $(1-\beta_t)\bar{\alpha}_{t-1} = \bar{\alpha}_t$. ■

In the kernel notation of the previous subsection, DDPM has $\alpha_t = \sqrt{\bar{\alpha}_t}$ and $\sigma_t = \sqrt{1-\bar{\alpha}_t}$. (A warning for readers of (Ho et al. 2020): the paper writes α_t for $1-\beta_t$, where our α_t is the signal coefficient $\sqrt{\bar{\alpha}_t}$, and its simple loss lacks our factor $\frac{1}{2}$.) Training therefore never simulates the chain. It jumps to any level in one step by Equation 38.81, and because the kernel is Gaussian, DSM applies directly: the conditional score is $-\epsilon/\sigma_t$. Parameterize the score model through a noise-prediction network, $\mathbf{s}_\theta(\mathbf{x}_t, t) = -\epsilon_\theta(\mathbf{x}_t, t)/\sigma_t$. Then

$$\left\| \mathbf{s}_\theta(\mathbf{x}_t, t) + \frac{\epsilon}{\sigma_t} \right\|^2 = \frac{1}{\sigma_t^2} \left\| \epsilon - \epsilon_\theta(\mathbf{x}_t, t) \right\|^2,$$

so the DDPM *simple loss* $\frac{1}{2}\mathbb{E}_{t,\mathbf{x}_0,\epsilon}\|\epsilon - \epsilon_\theta(\mathbf{x}_t, t)\|^2$ is the noise-conditional DSM loss Equation 38.78 with weight $w(t) = \sigma_t^2 = 1-\bar{\alpha}_t$. "Predict the noise you added" is denoising score matching with a weight that cancels the $1/\sigma_t^2$ growth of the floor: every level contributes a loss of order d.

The third part of DDPM is its sampler. *Ancestral sampling* steps through the levels from N down to 1,

$$\mathbf{x}_{t-1} = \frac{\mathbf{x}_t + \beta_t\,\mathbf{s}_\theta(\mathbf{x}_t, t)}{\sqrt{1-\beta_t}} + \sqrt{\beta_t}\,\xi, \qquad \xi \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

without the noise in the last step. To first order in β_t this is $\mathbf{x}_t + \frac{1}{2}\beta_t\mathbf{x}_t + \beta_t\mathbf{s}_\theta + \sqrt{\beta_t}\,\xi$, the Euler–Maruyama step of the reverse SDE in Equation 38.79 for the VP process. The model was first derived by a different route (Sohl-Dickstein et al. 2015; Ho et al. 2020), as a bound on the log-likelihood of a latent-variable model, the route that Section 17.5 follows; it yields the same regression with a different weight on each level.

The cell sets up what the samplers below use: the standard linear schedule β_t from 10^{-4} to 0.02 over $N = 1000$ levels, and our mixture samples rescaled to unit variance, so that the modes sit at $\pm 2/\sqrt{4.25} = \pm 0.970$ with component standard deviation 0.243. It also runs the chain once and compares it at three levels with one-shot samples from Equation 38.81, by the two-sample Kolmogorov–Smirnov (KS) statistic, the largest gap between their empirical distribution functions.

```
N, n = 1000, len(x)
beta = np.linspace(1e-4, 0.02, N)                        # the DDPM schedule
alpha_bar = np.cumprod(1.0 - beta)
def ks(a, b):                                          # two-sample Kolmogorov-Smirnov statistic
    q = np.sort(np.concatenate([a, b]))
    return np.abs(np.searchsorted(np.sort(a), q, 'right') / len(a)
                  - np.searchsorted(np.sort(b), q, 'right') / len(b)).max()
x0 = x / x.std()                                           # standardized mixture: modes +-0.970
```

```
xt = x0.copy()
for t in range(1, N + 1):                              # the forward chain, one level at a time
    xt = np.sqrt(1 - beta[t-1]) * xt + np.sqrt(beta[t-1]) * rng.standard_normal(n)
    if t in (10, 100, 1000):
        one_shot = np.sqrt(alpha_bar[t-1]) * x0 + np.sqrt(1 - alpha_bar[t-1]) *
↪  rng.standard_normal(n)
        print(f't = {t:4d}: alpha_bar {alpha_bar[t-1]:.3f}, Var(x_t) {xt.var():.3f} '
              f'(formula 1.000), KS {ks(xt, one_shot):.3f}')
```

```
t =   10: alpha_bar 0.998, Var(x_t) 1.000 (formula 1.000), KS 0.008
t =  100: alpha_bar 0.897, Var(x_t) 0.989 (formula 1.000), KS 0.015
t = 1000: alpha_bar 0.000, Var(x_t) 0.996 (formula 1.000), KS 0.022
```

The induction left this output nothing to decide: chain and formula agree to within the sampling error of 4,096 points. The rows are more useful as a picture of the levels. At $t = 100$, $\bar{\alpha} = 0.897$ and the noised law is still bimodal, with modes at ± 0.92 and component standard deviation 0.40. At $t = N$, $\bar{\alpha}_N \approx 4 \times 10^{-5}$, and both samples are standard Gaussians whatever the data were.

38.5.4 Deterministic Sampling and Guidance

Ancestral sampling calls the network once per level, $N = 1000$ times per sample. Can the same network produce samples with far fewer calls? The answer is a statement about ODE solvers and the third main result of the section. The second subsection asks what a sampler follows when it is steered toward a class.

38.5.4.1 Denoising Diffusion Implicit Models

The probability-flow ODE is deterministic and smooth, so a coarse grid of levels should suffice to integrate it. DDIM (J. Song et al. 2021) realizes this with an update that reuses the trained noise-prediction network and skips levels. At level t, the network's noise estimate gives a guess of the clean sample by inverting the marginal Equation 38.81,

$$\hat{\mathbf{x}}_0 = \frac{\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\,\epsilon_\theta(\mathbf{x}_t, t)}{\sqrt{\bar{\alpha}_t}}.$$

To move to any earlier level $t' < t$, DDIM rebuilds the sample from this guess and the *same* noise estimate, with the coefficients of level t':

$$\mathbf{x}_{t'} = \sqrt{\bar{\alpha}_{t'}}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_{t'}}\,\epsilon_\theta(\mathbf{x}_t, t). \tag{38.82}$$

The update is deterministic: no fresh noise enters. Read literally, it moves a sample along the forward curve $\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\epsilon$ of the pair $(\hat{\mathbf{x}}_0, \epsilon_\theta)$. But a network trained by least squares returns the conditional mean $\mathbb{E}[\epsilon \mid \mathbf{x}_t]$, not the noise that produced this sample, and the estimate changes from level to level. What, then, does DDIM integrate, and how large an error does a long stride make?

A change of coordinates answers both questions. Divide by $\sqrt{\bar{\alpha}_t}$. The rescaled point $\bar{\mathbf{x}}_t = \mathbf{x}_t/\sqrt{\bar{\alpha}_t}$ equals $\mathbf{x}_0 + \bar{\sigma}_t\epsilon$ with $\bar{\sigma}_t = \sqrt{(1-\bar{\alpha}_t)/\bar{\alpha}_t}$. In these coordinates the VP process is a VE process, clean data plus noise of scale $\bar{\sigma}$, which grows from 0.01 at the first level to about

157 at the last under the standard schedule. The schedule only decides which values of $\bar{\sigma}$ the grid visits. On $\bar{\mathbf{x}}_t$ and $\bar{\sigma}_t$ the bar marks the rescaled point and its noise level; the bar on $\bar{\alpha}_t$ is DDPM's and means something else.

Proposition (DDIM is Euler's method). *In the coordinates $(\bar{\sigma}, \bar{\mathbf{x}})$ the DDIM update Equation 38.82 reads*

$$\bar{\mathbf{x}}_{t'} = \bar{\mathbf{x}}_t + (\bar{\sigma}_{t'} - \bar{\sigma}_t)\, \epsilon_\theta(\mathbf{x}_t, t), \tag{38.83}$$

one Euler step of the ODE $d\bar{\mathbf{x}}/d\bar{\sigma} = \epsilon_\theta$. If ϵ_θ is the exact noise prediction $\hat{\epsilon}(\mathbf{x}_t, t) = \mathbb{E}[\epsilon \mid \mathbf{x}_t]$, this ODE is the probability-flow ODE of the VP process, written in the coordinates $(\bar{\sigma}, \bar{\mathbf{x}})$.

Proof. Step (i) is algebra; the substance is in (ii) and (iii), which show that the probability-flow ODE takes this form in the new coordinates. In (ii), t is the continuous time of the SDE.

(i) Divide Equation 38.82 by $\sqrt{\bar{\alpha}_{t'}}$ and substitute $\hat{\mathbf{x}}_0 = \bar{\mathbf{x}}_t - \bar{\sigma}_t \epsilon_\theta$: $\bar{\mathbf{x}}_{t'} = \hat{\mathbf{x}}_0 + \bar{\sigma}_{t'} \epsilon_\theta = \bar{\mathbf{x}}_t + (\bar{\sigma}_{t'} - \bar{\sigma}_t)\epsilon_\theta$.

(ii) Change variables in the VP probability-flow ODE $d\mathbf{x}/dt = -\frac{1}{2}\beta\mathbf{x} - \frac{1}{2}\beta\nabla_{\mathbf{x}} \log p_t$ of Equation 38.79, with $\alpha_t = \sqrt{\bar{\alpha}(t)}$ and $\bar{\alpha}(t) = e^{-\int_0^t \beta}$. Let $\bar{p}_t$ be the density of $\bar{\mathbf{x}}_t$; then $\nabla_{\mathbf{x}} \log p_t = \nabla_{\bar{\mathbf{x}}} \log \bar{p}_t / \sqrt{\bar{\alpha}}$. Since $d\sqrt{\bar{\alpha}}/dt = -\frac{1}{2}\beta\sqrt{\bar{\alpha}}$, the shrinkage terms cancel and $d\bar{\mathbf{x}}/dt = -\frac{\beta}{2\bar{\alpha}} \nabla \log \bar{p}_t$. Since $\bar{\sigma}^2 = 1/\bar{\alpha} - 1$, we have $d\bar{\sigma}/dt = \beta/(2\bar{\alpha}\bar{\sigma})$, and dividing gives $d\bar{\mathbf{x}}/d\bar{\sigma} = -\bar{\sigma}\, \nabla \log \bar{p}(\bar{\mathbf{x}})$.

(iii) Tweedie's formula for the kernel $\mathcal{N}(\mathbf{x}_0, \bar{\sigma}^2 \mathbf{I})$ reads $\mathbb{E}[\mathbf{x}_0 \mid \bar{\mathbf{x}}] = \bar{\mathbf{x}} + \bar{\sigma}^2 \nabla \log \bar{p}$, so $\mathbb{E}[\epsilon \mid \bar{\mathbf{x}}] = (\bar{\mathbf{x}} - \mathbb{E}[\mathbf{x}_0 \mid \bar{\mathbf{x}}])/\bar{\sigma} = -\bar{\sigma} \nabla \log \bar{p}$, the right-hand side of (ii). Conditioning on $\bar{\mathbf{x}}_t$ is the same as conditioning on $\mathbf{x}_t$. ■

DDIM is therefore a first-order solver for the probability-flow ODE, in a clock in which the noise level is the time. Its error is Euler's error: a stride holds ϵ_θ fixed while the exact $\hat{\epsilon}$ changes along the trajectory. Two corollaries mark the extremes of this error.

Corollary (a point mass is integrated without error). *If the data are a single point $\mathbf{x}_0$, every DDIM stride with the exact $\hat{\epsilon}$ lands on the probability-flow trajectory, whatever its length.* Here $\hat{\epsilon} = (\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$, and along a solution of $d\bar{\mathbf{x}}/d\bar{\sigma} = (\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$ the ratio $(\bar{\mathbf{x}} - \mathbf{x}_0)/\bar{\sigma}$ is constant. An Euler step for a constant right-hand side is exact.

Corollary (Gaussian data shrink). *For data $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ the exact flow is the identity, but one DDIM stride with the exact noise prediction multiplies every sample by*

$$\sqrt{\bar{\alpha}_t \bar{\alpha}_{t'}} + \sqrt{(1 - \bar{\alpha}_t)(1 - \bar{\alpha}_{t'})} = \cos(\phi_t - \phi_{t'}), \qquad \bar{\alpha}_t = \cos^2 \phi_t,$$

which is below 1 for every stride of positive length. The marginal is $\mathcal{N}(\mathbf{0}, \mathbf{I})$ at every level, so $\hat{\epsilon}(\mathbf{x}_t, t) = \sqrt{1 - \bar{\alpha}_t}\, \mathbf{x}_t$ by Tweedie, and substituting into Equation 38.82 gives the factor. Writing $\bar{\alpha}_t = \cos^2 \phi_t$ turns it into the cosine of the angle advanced. In the bar coordinates the exact solution is $\bar{\mathbf{x}} \propto \sqrt{1 + \bar{\sigma}^2}$, so $\hat{\epsilon} = \bar{\sigma}\bar{\mathbf{x}}/(1 + \bar{\sigma}^2)$ changes along it. Holding it fixed across a stride removes too much signal, by the factor above.

The corollary gives the factor of one stride; over a sampling run the factors multiply. The cell computes the product for 10, 50 and 1,000 equally spaced strides of the DDPM schedule. It then runs DDIM on the standardized mixture with its exact noise prediction

$\hat{\epsilon} = -\sqrt{1-\bar{\alpha}_t}\,\nabla \log p_t$, where the noised law at level t has modes $\pm\sqrt{\bar{\alpha}_t}\,m$ and component variance $\bar{\alpha}_t v_0 + 1 - \bar{\alpha}_t$, with $m = 0.970$ and $v_0 = 0.243^2$ the clean values, and compares 10 and 50 strides with 1,000 from the same initial draws.

```
m, v0 = 2 / np.sqrt(4.25), 0.25 / 4.25                # standardized mixture: modes +-m,
↪  variance v0
def strides(K):                                      # K equal strides from level N to level 1
    return np.linspace(N - 1, 0, K + 1).round().astype(int)
for K in (10, 50, 1000):                              # the Gaussian corollary
    a = alpha_bar[strides(K)]
    print(f'N(0,1) data, {K:4d} strides: every sample times '
          f'{np.prod(np.sqrt(a[:-1] * a[1:]) + np.sqrt((1 - a[:-1]) * (1 - a[1:]))):.3f}')
def eps_hat(xt, a):                                   # exact noise prediction of the mixture
    return -np.sqrt(1 - a) * mixture_score(xt, a * v0 + 1 - a, (-np.sqrt(a) * m, np.sqrt(a) *
    ↪  m))
def ddim(z, K, eps_fn):                               # K DDIM strides from noise z
    xt, idx = z.copy(), strides(K)
    for t1, t2 in zip(idx[:-1], idx[1:]):
        a1, a2 = alpha_bar[t1], alpha_bar[t2]
        e = eps_fn(xt, a1)
        xt = np.sqrt(a2) * (xt - np.sqrt(1 - a1) * e) / np.sqrt(a1) + np.sqrt(1 - a2) * e
    return xt
z = rng.standard_normal(8000)                         # one draw, shared by every stride count
x_ref = ddim(z, 1000, eps_hat)
for K in (10, 50):
    x_K = ddim(z, K, eps_hat)
    print(f'mixture, {K:3d} strides vs 1000: mean |gap| {np.abs(x_K - x_ref).mean():.3f}, '
          f'KS {ks(x_K, x_ref):.3f}')
```

```
N(0,1) data,   10 strides: every sample times 0.832
N(0,1) data,   50 strides: every sample times 0.964
N(0,1) data, 1000 strides: every sample times 0.998
mixture,  10 strides vs 1000: mean |gap| 0.084, KS 0.083
mixture,  50 strides vs 1000: mean |gap| 0.015, KS 0.018
```

Ten strides shrink unit-variance Gaussian data by the factor 0.832, fifty by 0.964, and a thousand by 0.998. These factors are deterministic and measure the Euler error alone, since the noise prediction is exact. The mixture behaves the same way. Ten strides land on average 0.08 from the thousand-stride result, on a scale where the modes sit at ± 0.97. Their empirical distribution differs from it by a KS statistic of 0.08, and fifty strides reduce both numbers to about 0.02. Figure 38.24 shows both experiments. A higher-order solver in the same coordinates spends two network calls per stride and wins only once the strides are short (exercise 6).

Panel (a) answers the literal reading of the update. The sample travels along the probability-flow trajectory through its noise end, not along the forward curve that produced it. DDIM is the deterministic end of the family of reverse processes Equation 38.68, whose members reinject a chosen amount of noise and share the single-time marginals of the forward process, so one trained network serves them all. Section 17.4 chooses among them.

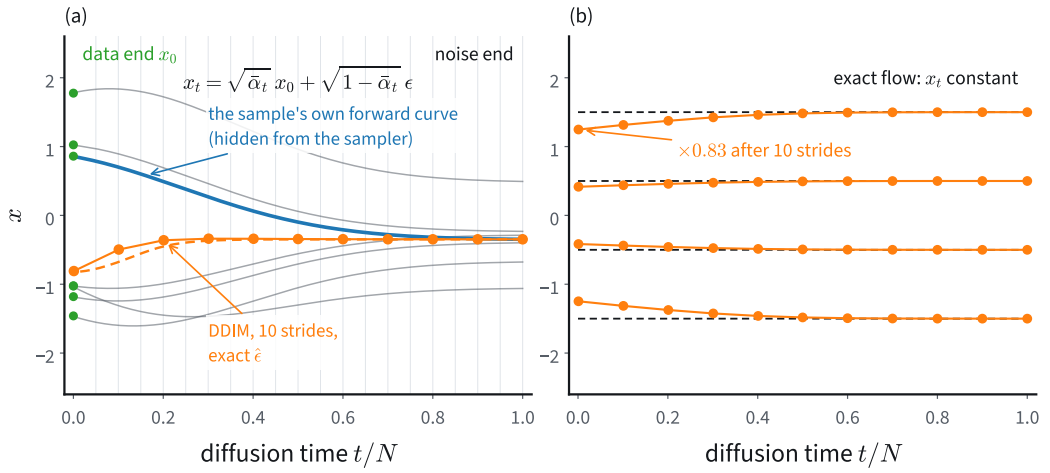


Figure 38.24: (a) Each forward pair (x_0, ϵ) of the standardized mixture owns one curve; the blue pair crossed the valley. From the same noise end, DDIM with the exact noise prediction follows the probability-flow ODE (10 strides, dots; 1,000 strides, dashed) into the other mode: it never sees the sample's own curve. (b) For unit-variance Gaussian data the exact flow is the identity (dashed); ten DDIM strides with the exact noise prediction shrink every sample by 0.83, the Euler error of the Gaussian corollary.

38.5.4.2 Guidance with Bayes' Rule

Section 17.3 teaches guidance and measures it on images. We keep two facts: guidance is Bayes' rule on scores, and with a scale above one the sampler follows densities that no noised data could have.

Suppose we want samples of one class y only. On the mixture, let y be "the right mode", so that we want samples from $\mathcal{N}(0.970, 0.243^2)$ alone. Bayes' rule at noise level t, $p_t(\mathbf{x} \mid y) \propto p_t(\mathbf{x})\, p_t(y \mid \mathbf{x})$, becomes additive for scores, because the evidence $p(y)$ does not depend on $\mathbf{x}$:

$$\nabla_{\mathbf{x}} \log p_t(\mathbf{x} \mid y) = \nabla_{\mathbf{x}} \log p_t(\mathbf{x}) + \nabla_{\mathbf{x}} \log p_t(y \mid \mathbf{x}). \tag{38.84}$$

The conditional score is the unconditional score plus the gradient of a classifier's log-probability, and every sampler of this section runs unchanged with it. **Classifier guidance** (Dhariwal and Nichol 2021) takes the second term from a classifier $p_\phi(y \mid \mathbf{x}, t)$ of noisy inputs and scales it by a **guidance scale** γ:

$$\tilde{\mathbf{s}}(\mathbf{x}, t) = \nabla \log p_t(\mathbf{x}) + \gamma\, \nabla \log p_\phi(y \mid \mathbf{x}, t), \qquad \tilde{\mathbf{s}} = \nabla \log \tilde{p}_t, \quad \tilde{p}_t(\mathbf{x}) \propto p_t(\mathbf{x})\, p_\phi(y \mid \mathbf{x}, t)^\gamma.$$

With $\gamma = 1$ this is Bayes' rule. With $\gamma > 1$ the guided field is the score of a *tilted* density that raises the classifier's probability to a power. **Classifier-free guidance (CFG)** (Ho and Salimans 2022) reads Equation 38.84 from right to left, $\nabla \log p_t(y \mid \mathbf{x}) = \nabla \log p_t(\mathbf{x} \mid y) - \nabla \log p_t(\mathbf{x})$, and so needs no classifier, only a conditional and an unconditional score model $\mathbf{s}_\theta(\mathbf{x}, t, y)$ and $\mathbf{s}_\theta(\mathbf{x}, t, \emptyset)$, which one network can supply (Section 17.3.4.2):

$$\tilde{\mathbf{s}}(\mathbf{x}, t) = \mathbf{s}_\theta(\mathbf{x}, t, \emptyset) + \gamma \left[\mathbf{s}_\theta(\mathbf{x}, t, y) - \mathbf{s}_\theta(\mathbf{x}, t, \emptyset)\right], \tag{38.85}$$

equivalently $(1-\gamma)\,\mathbf{s}_\emptyset + \gamma\,\mathbf{s}_y$, which for $\gamma > 1$ extrapolates past the conditional score.

One fact limits the interpretation of $\gamma > 1$. The tilted density $\tilde{p}_t$ is in general not the noised marginal of *any* clean distribution. The marginal p_t is the law of $\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0$ noised at scale σ_t, so by Equation 38.76 no eigenvalue of $-\nabla^2 \log p_t$ exceeds $1/\sigma_t^2$: a noised marginal cannot curve more sharply than its own noise. Raising the classifier to the power γ adds curvature and breaks this bound, for the mixture as soon as γ exceeds 1.0625 at the noisiest levels and 1.09 at $t = 100$ (exercise 8). At $\gamma = 3$ the bound holds only at about ten of the thousand levels, the least noisy ones, and at $\gamma = 10$ at fewer still.

What does the guided sampler produce, if not samples of a noised class? The cell runs the ancestral chain on the standardized mixture with the exact unconditional score and the exact class-conditional score, the score of the single Gaussian $\mathcal{N}(\sqrt{\bar{\alpha}_t}\,m,\ \bar{\alpha}_t v_0 + 1 - \bar{\alpha}_t)$, combined by Equation 38.85 at $\gamma \in \{1, 3, 10\}$.

```
def cfg_sample(gamma, n=8000):                        # ancestral chain with the guided score
    xt = rng.standard_normal(n)
    for t in range(N - 1, -1, -1):
        a, b = alpha_bar[t], beta[t]
        v = a * v0 + 1 - a                              # noised component variance
        s_uncond = mixture_score(xt, v, (-np.sqrt(a) * m, np.sqrt(a) * m))
        s_cond = (np.sqrt(a) * m - xt) / v              # class y = the right mode
        xt = (xt + b * ((1 - gamma) * s_uncond + gamma * s_cond)) / np.sqrt(1 - b)
        if t > 0:
            xt = xt + np.sqrt(b) * rng.standard_normal(n)
    return xt
for gamma in (1.0, 3.0, 10.0):
    xs = cfg_sample(gamma)
    print(f'gamma = {gamma:4.1f}: mass right of 0 {(xs > 0).mean():.3f}, '
          f'mean {xs.mean():.3f}, std {xs.std():.3f}')
print(f'exact conditional: mean {m:.3f}, std {np.sqrt(v0):.3f}')
```

```
gamma =  1.0: mass right of 0 1.000, mean 0.972, std 0.246
gamma =  3.0: mass right of 0 1.000, mean 1.028, std 0.234
gamma = 10.0: mass right of 0 1.000, mean 1.074, std 0.233
exact conditional: mean 0.970, std 0.243
```

At $\gamma = 1$ the chain reproduces the class: all the mass lies right of zero, where the unconditional mixture puts only half of it, and the mean and standard deviation are within about 0.01 of the exact 0.970 and 0.243. At $\gamma = 3$ and 10 there is no mass left to move between modes, so the guidance moves the mode itself, to about 1.03 and 1.08, away from the class boundary. The width stays within a few percent of 0.24.

The shift is not produced at the clean level, where the classes are eight standard deviations apart and the classifier is certain, but at intermediate levels, where the noised classes overlap (exercise 7). Guidance with $\gamma > 1$ thus moves samples away from the other class, beyond where the class itself sits. A class with a single Gaussian mode has little diversity to lose; on images the loss is measurable (Section 17.3.4.2).

Guided or not, every sampler of this section calls the network many times, and the number of calls is set by how curved its paths are. The forward process fixes those paths. We chose it

because its kernels are Gaussian, not because its trajectories are easy to integrate, and the Gaussian corollary showed that even the simplest data give curved paths in DDIM's clock. Which noise-to-data paths are straight? The next section regresses on a velocity instead of a score and finds that the question is one of optimal transport.

38.5.5 Summary

A score needs no normalizing constant. For noised data it is the conditional mean of a target we know for every training pair, $-\epsilon/\sigma$, so least squares on noisy pairs learns it from samples alone, and the loss ends at the posterior-variance floor, not at zero. Hyvärinen's identity does the same for clean data at the cost of a Jacobian trace. A single noise level leaves the field arbitrary far from the data and Langevin dynamics confined to the modes it starts in, so the score is learned at every level of a forward process. DDPM is the variance-preserving SDE observed at discrete times: its $\bar{\alpha}$-marginal is exact, and its noise-prediction loss is DSM weighted by σ_t^2. DDIM is Euler's method for the probability-flow ODE in the coordinates $(\bar{\sigma}, \bar{\mathbf{x}})$, exact for a point mass and too contractive for Gaussian data. Guidance is Bayes' rule on scores, and for $\gamma > 1$ it leaves the family of noised densities, whose curvature is at most $1/\sigma_t^2$. Every sampler here ran on an exact score.

38.5.6 Exercises

1. **Boundary terms.** Hyvärinen's identity Equation 38.72 assumes that $p(x)\, s_\theta(x)$ vanishes at the ends of the support. Let the data be uniform on $[0, 1]$, so that $\nabla \log p = 0$ inside the interval, and fit the linear model $s_{a,b}(x) = a + bx$.
    1. Minimize the explicit objective Equation 38.71 and the implicit objective, the right-hand side of Equation 38.72, over (a, b). Which density has the score that the implicit objective selects?
    2. Repeat the one-dimensional integration by parts of the proof on $[0, 1]$, keep the boundary term, and show that it accounts for the difference between the two objectives.
    3. Pixel intensities are bounded in the same way. Why does denoising score matching at a noise level $\sigma > 0$ not inherit the problem?
2. code **Langevin discretization bias.** For the Gaussian target $p = \mathcal{N}(0, v)$, the Langevin sampler of Equation 38.77 with step h is the linear recursion $x \leftarrow (1 - \frac{h}{2v})\, x + \sqrt{h}\, \xi$.
    1. Find the step sizes for which the recursion has a stationary distribution, compute its variance, and expand the bias to first order in h.
    2. Treat each mode of the mixture in the cell that defines `langevin` as a separate Gaussian with $v = 0.25$, and predict the stationary $\mathbb{E}[X^2]$ at $h \in \{0.01, 0.1, 0.25, 0.5\}$. Compare each predicted bias with the Monte Carlo standard error of an estimate from 10000 chains. Then run `langevin` for 2000 steps at each of these step sizes from the initial points `rng.normal(0.0, 3.0, 10000)` of `warm`. From which step size on does the experiment resolve the bias?
    3. The Metropolis-adjusted Langevin algorithm (Roberts and Tweedie 1996) removes the bias with an accept–reject test that needs the density ratio $p(\mathbf{x}')/p(\mathbf{x})$. Why can a sampler that has only a trained score network $\mathbf{s}_\theta$ not apply this test directly?

3. code **Annealing out of one mode.** In the cell that defines `langevin`, chains started in the left mode stay there. Annealed Langevin dynamics first runs the sampler on smoothed densities p_σ, here the mixture with component variance $0.25+\sigma^2$, for the decreasing ladder of noise levels $\sigma = 2, 1, 0.5, 0.25, 0.1, 0$.
    1. By exercise 6 of Section 38.4, the mixture $\frac{1}{2}\mathcal{N}(-\mu, v_0) + \frac{1}{2}\mathcal{N}(\mu, v_0)$ is unimodal if and only if $\mu^2 \le v_0$. The means of `mixture_score` are ± 2. For which σ is p_σ unimodal? Which levels of the ladder lie in this range?
    2. Start 10000 chains at -2 and run 300 steps of `langevin` at each level of the ladder, passing the component variance $0.25 + \sigma^2$ as `var`. Do this once with the fixed step $h = 0.01$ and once with $h = 0.04\,(0.25 + \sigma^2)$, which equals 0.01 at $\sigma = 0$. Report the fraction of chains with $X > 0$ at the end of each run.
    3. Song and Ermon (2019) scale the step size of each level with σ^2. Use the contraction factor of the linear recursion in exercise 2 to explain why the fixed step fails at the large noise levels.
4. code **The Bayes-risk floor.** ⊠ The cell that fits `w_ls` prints the mean squared error of the fit and of the exact score, both averages over the same 16384 noisy pairs, and the population floor by quadrature.
    1. Let $Y = -\epsilon/\sigma$ be the DSM target Equation 38.73 for data in $\mathbb{R}^d$. Use the regression lemma Equation 38.75 and Vincent's theorem to show that the population Bayes risk is
    $$\mathbb{E}\,\|Y - \nabla \log p_\sigma(\tilde{\mathbf{x}})\|^2 = \frac{d}{\sigma^2} - \mathbb{E}_{\tilde{\mathbf{x}}\sim p_\sigma}\,\|\nabla \log p_\sigma(\tilde{\mathbf{x}})\|^2 .$$
    2. Compute the standard error of the printed exact-score average from its 16384 per-pair terms. Is the printed value consistent with the population floor?
    3. The two printed averages differ by much less than that standard error. Explain why. Write their difference as a sample average of $(s_\theta - s_\sigma)(s_\theta + s_\sigma - 2y)$, with $s_\sigma = \nabla \log p_\sigma$, and say what makes the variance of this average small and why its mean can be slightly negative even though the excess risk is positive.
    4. Estimate the excess risk $\mathbb{E}_{p_\sigma}(s_\theta - s_\sigma)^2$ at the noisy inputs `xt` in two ways: as the difference of the two printed squared errors, and as the mean of $(s_\theta - s_\sigma)^2$ at these points. Compare both with the printed quadrature value. Which is more precise, and why?
5. **Weightings across noise levels.** ⊠ Consider the VP-SDE with the linear schedule $\beta(t) = \beta_{\min} + t\,(\beta_{\max} - \beta_{\min})$ on $[0, 1]$, with $\beta_{\min} = 0.1$ and $\beta_{\max} = 20$, so that $\bar{\alpha}(t) = \exp\big(-\beta_{\min} t - \frac{1}{2}(\beta_{\max} - \beta_{\min})t^2\big)$ and $\sigma_t^2 = 1 - \bar{\alpha}(t)$. Let the data in $\mathbb{R}^d$ have a smooth density, and assume that $\mathbb{E}\|\nabla \log p_t\|^2$ stays bounded as $t \to 0$. Write $R(t)$ for the Bayes risk of the noise-conditional DSM loss Equation 38.78 at level t, which exercise 4 gives with σ_t in place of σ.
    1. Show that $R(t) \approx d/(\beta_{\min} t)$ as $t \to 0$.
    2. Find the behavior of $w(t)\,R(t)$ as $t \to 0$ for $w = 1$, for the DDPM weighting $w = \sigma_t^2$, and for the likelihood weighting $w = g^2 = \beta(t)$. For which weightings is $\int_0^1 w(t)\,R(t)\,dt$ finite, and which noise levels does each weighting emphasize?

3. Training draws t uniformly from $[\varepsilon, 1]$. For the likelihood weighting, show that the second moment of $w(t)\,R(t)$ under this draw grows like $1/\varepsilon$. Y. Song, Durkan, et al. (2021) (Section 5.1) instead draw t from the density proportional to $\beta(t)/\sigma_t^2$ on $[\varepsilon, 1]$ and multiply the DSM term at the drawn level by $c\,\sigma_t^2$, where c normalizes that density. Show that this estimate is unbiased for the likelihood-weighted loss and that its value at the optimum, $c\,\sigma_t^2\,R(t)$, stays bounded as $t \to 0$.

6. code **DDIM against a second-order stride.** ⊠ By the proposition Equation 38.83, DDIM is Euler's method for $d\bar{\mathbf{x}}/d\bar{\sigma} = \hat{\epsilon}$ on the grid `strides(K)` of the cell that defines `ddim`.
   1. Implement Heun's method in the coordinates $(\bar{\sigma}, \bar{x})$ on the same grid: evaluate $\hat{\epsilon}$ at the start of a stride and at the Euler predictor, and step with their average, two evaluations per stride. Compare its endpoint error against `x_ref` with that of DDIM at equal numbers of evaluations: Heun with $5, 10, 20$ strides against DDIM with $10, 20, 40$. Which method wins at each budget?
   2. Show that for data concentrated at a single point x_0 every DDIM stride lands on the probability-flow trajectory, and verify it by passing to `ddim` the point-mass prediction $(\bar{x} - x_0)/\bar{\sigma}$, written in terms of x_t and $\bar{\alpha}_t$. Which quantity does DDIM hold fixed across a stride, and which quantity does Euler's method in the original clock t hold fixed?
   3. Repeat (a) with the component variance `v0` replaced by 0.01 and by 10^{-4}, and recompute `x_ref`. How does the ranking change? Explain it with the slope $\partial\hat{\epsilon}/\partial\bar{x} = \bar{\sigma}/(v_0 + \bar{\sigma}^2)$ of the noise prediction near one narrow component, which peaks at $1/(2\sqrt{v_0})$ near $\bar{\sigma} = \sqrt{v_0}$, and with the stability region of Heun's method for the last stride.
   4. Karras et al. (2022) (Algorithm 1) skip the second evaluation on the stride that ends at the lowest noise level. Make this change and repeat (c). Why does the skip cost nothing for a point mass?

   The deliverable is a table of errors against evaluations for the three component variances.

7. code **Guidance by noise level.** ⊠ In the cell that defines `cfg_sample`, raising γ from 1 to 10 moves the mean of the guided samples from about 0.97 to about 1.08.
   1. At the last level of the chain, $t = 0$ in `cfg_sample`, the class posterior of the right mode is a logistic function of x. Write it, and compute by quadrature the mean of the tilted density proportional to $p_t(x)\,p_t(y \mid x)^\gamma$ at that level for $\gamma \in \{1, 3, 10\}$. Can this tilt explain the printed shift? Where must the shift come from?
   2. Modify `cfg_sample` so that it uses $\gamma = 10$ only on the steps $t \geq 500$, only on $200 \leq t < 500$, or only on $t < 200$, and $\gamma = 1$ on the others. Which band of noise levels produces the shift? Relate the answer to the separation $\sqrt{\bar{\alpha}_t}\,m/\sqrt{v_t}$ of the two noised classes, where m and v_0 are the variables `m` and `v0`, and $v_t = \bar{\alpha}_t\,v_0 + 1 - \bar{\alpha}_t$ is the noised component variance, the variable `v` of `cfg_sample`.
   3. Kynkäänniemi et al. (2024) find on image models that guidance is harmful at high noise levels, largely unnecessary at low noise levels, and beneficial in between. Which parts of this pattern does the one-dimensional example reproduce, and which can it not exhibit?

8. **Guided densities and noised marginals.** ⊠ The guided field Equation 38.85 with $\gamma > 1$ is the score of the tilted density $\tilde{p}_t(\mathbf{x}) \propto p_t(\mathbf{x})\,p_t(y \mid \mathbf{x})^\gamma$. Call a density a noised marginal at level t if it is the law of $\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sigma_t\epsilon$, with $\sigma_t^2 = 1 - \bar{\alpha}_t$, for some clean

distribution of $\mathbf{x}_0$.

1. Apply Equation 38.76 to the clean variable $\sqrt{\bar{\alpha}_t}\,\mathbf{x}_0$ to show that a noised marginal p_t has

$$\nabla^2 \log p_t(\mathbf{x}) = -\frac{\mathbf{I}}{\sigma_t^2} + \frac{\bar{\alpha}_t}{\sigma_t^4}\mathrm{Cov}\left[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}\right]. \tag{38.86}$$

   Conclude that no eigenvalue of $-\nabla^2 \log p_t$ exceeds $1/\sigma_t^2$, whatever the clean distribution.

2. In `cfg_sample`, the two noised classes are $\mathcal{N}(\pm\sqrt{\bar{\alpha}_t}\,m, v_t)$, where m and v_0 are the variables `m` and `v0`, and $v_t = \bar{\alpha}_t\, v_0 + \sigma_t^2$ is the noised component variance. The label y names the right class. Show that $-(\log \tilde{p}_t)''(x) = 1/v_t + (\gamma - 1)\, a_t^2\, \varsigma(a_t x)\,(1 - \varsigma(a_t x))$, where $a_t = 2\sqrt{\bar{\alpha}_t}\, m / v_t$ and $\varsigma(u) = 1/(1 + e^{-u})$ is the logistic function.

3. Find the smallest γ for which $\tilde{p}_t$ violates the bound of

   (a) at some x. Evaluate it at the levels $t = 0$, 100, and 500 of `alpha_bar`, and find its limit as $t \to N$. What do the values say about the densities that the chain of `cfg_sample` follows at $\gamma = 3$?

9. **Score error and likelihood.** ▯▯ With the likelihood weighting $w = g^2$, the population score loss bounds the Kullback–Leibler divergence from the data to the model, by Theorem 1 of Y. Song, Durkan, et al. (2021) . The paper proves the bound with path measures; this problem derives it from the Fokker–Planck equation of Section 38.4.2, following Lu, Zheng, et al. (2022), who take this route for the SDE (their Proposition C.1) and for the probability-flow ODE (their Theorem 3.1). Let the forward SDE $d\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\, dt + g(t)\, d\mathbf{W}$ have marginals p_t on $[0, T]$. In reverse time $\bar{t} = T - t$, the exact reversal and the model are

$$d\mathbf{x} = \left[-\mathbf{f} + g^2\, \nabla \log p_t\right] d\bar{t} + g\, d\mathbf{B}_{\bar{t}}, \qquad d\mathbf{x} = \left[-\mathbf{f} + g^2\, \mathbf{s}_\theta(\mathbf{x}, t)\right] d\bar{t} + g\, d\mathbf{B}_{\bar{t}}, \tag{38.87}$$

evaluated at $t = T - \bar{t}$. The first has marginals p_t; the second, started from a prior π at $\bar{t} = 0$, has marginals q_t. Assume that all densities are smooth and positive and decay fast enough for every integration by parts.

1. Write both Fokker–Planck equations in $\bar{t}$ in the form $\partial_{\bar{t}} \rho = -\nabla \cdot (\rho\, \mathbf{u})$, and identify the velocities $\mathbf{u}_p$ and $\mathbf{u}_q$.

2. Show that $\frac{d}{d\bar{t}} D_{\mathrm{KL}}(p_t \,\|\, q_t) = \mathbb{E}_{p_t}\left[(\mathbf{u}_p - \mathbf{u}_q)^\top \nabla \log(p_t / q_t)\right]$.

3. Bound the right-hand side by $\frac{1}{2} g^2\, \mathbb{E}_{p_t} \|\mathbf{s}_\theta - \nabla \log p_t\|^2$, integrate over $\bar{t}$, and obtain

$$D_{\mathrm{KL}}(p_0 \,\|\, q_0) \le D_{\mathrm{KL}}(p_T \,\|\, \pi) + \frac{1}{2} \int_0^T g(t)^2\, \mathbb{E}_{p_t} \left\|\mathbf{s}_\theta(\mathbf{x}, t) - \nabla \log p_t(\mathbf{x})\right\|^2 dt. \tag{38.88}$$

   Under which condition on q_t is the bound of the derivative tight?

4. Repeat (a) and (b) for the probability-flow ODE with $\mathbf{s}_\theta$ in place of the score. Which term that made (c) possible is missing, and why does the score loss alone no longer bound $D_{\mathrm{KL}}(p_0 \,\|\, q_0)$?

10. [extended] **Learned noise prediction.** ▯▯ The samplers `langevin`, `ddim`, and `cfg_sample` all use exact scores. Replace the exact score by a trained network for the standardized mixture `x / x.std()` under the DDPM schedule `beta` and `alpha_bar`.

    1. In the framework of your tab, train a noise-prediction network $\epsilon_\theta(x_t, t)$ with input $(x_t, t/1000)$ and two hidden layers of 64 tanh units, with Adam at learning rate 3×10^{-3} for 4000 steps on batches of 256, on the simple loss with t uniform over the 1000 levels. Compute its root-mean-square error against the exact noise prediction `eps_hat` on samples of p_t at several levels t. Where is the error largest?
    2. Sample with `ddim` at $K = 10$ and $K = 50$, and with the ancestral chain of `cfg_sample` in which the network's score replaces the guided score. Compare each set of samples with exact draws of the mixture by the fraction in the right mode, the standard deviation, and `ks`. Which sampler is most sensitive to the error of (a), and at which step of that sampler does the error enter?
    3. Retrain with the per-level weights that correspond to $w = 1$ and to the likelihood weighting $w = \beta$, that is, with the noise loss at level t multiplied by $w(t)/(1-\bar{\alpha}_t)$, and repeat (a) and (b). Relate the results to the exercise on weightings, 5.

    The deliverable is a table of the statistics of (b) for each weighting and sampler, together with a plot of the error of (a) against t.

38.6 Flow Matching and Optimal Transport

The previous section learned the score of noised data by least squares and turned it into samples, with a reverse chain of a thousand small steps or a few dozen deterministic strides, each of which evaluates the score once. So many steps are needed because the paths from noise to data curve, and a solver can follow a curved path only in short pieces. Those paths were fixed by the noising process, which we chose for its Gaussian kernels, not for easy integration. Could we choose the path from noise to data ourselves, and choose one along which a solver can take long steps?

Picture two data points at -2 and $+2$ and a cloud of Gaussian noise that must end on them. If every particle moved in a straight line at constant speed toward its own point, one Euler step of length one would carry it there exactly. But a velocity field is one function of position and time. Particles bound for different points pass through the same place, and the field there can point only one way.

Flow matching (Lipman et al. 2023) and rectified flow (Liu et al. 2023) prescribe the path and learn only the velocity that moves mass along it. The argument is the one we used for the score: the velocity of the cloud at a point is the average of the velocities of the per-sample paths through that point, and least squares learns averages.

Which paths are cheapest is a much older question. In 1781 Gaspard Monge asked how to move a pile of earth into a hole with the least total effort, and Benamou and Brenier (2000) showed that under squared distance the cheapest motion carries every particle along a straight line at constant speed. A sample costs one network call per solver step, so these are the paths we want.

Three results carry the section. The velocity of the cloud is a posterior mean, so one regression trains it (Section 38.6.1). Re-pairing noise with the endpoints of a flow never raises the cost of transport (Section 38.6.2.3). And the least kinetic energy of any flow between two laws is their squared Wasserstein distance (Section 38.6.3). The dictionary between score, noise and velocity (Section 38.6.1.4) and the solver comparison (Section 38.6.4) can be skipped on a first reading.

! One clock

Time runs as in the previous section, from the data at $t = 0$ to the noise, now at $t = 1$. This holds for scores and velocities alike: training noises the data forward in time, and every sampler integrates from $t = 1$ down to $t = 0$.
The flow-matching papers cited in this section, and their code, run time the other way, from noise at 0 to data at 1 (Figure 38.25). To compare a formula with theirs, substitute $1 - t$ for their t, read their noise $\mathbf{x}_0$ as our ϵ and their data $\mathbf{x}_1$ as our $\mathbf{x}_0$, and flip the sign of every velocity.

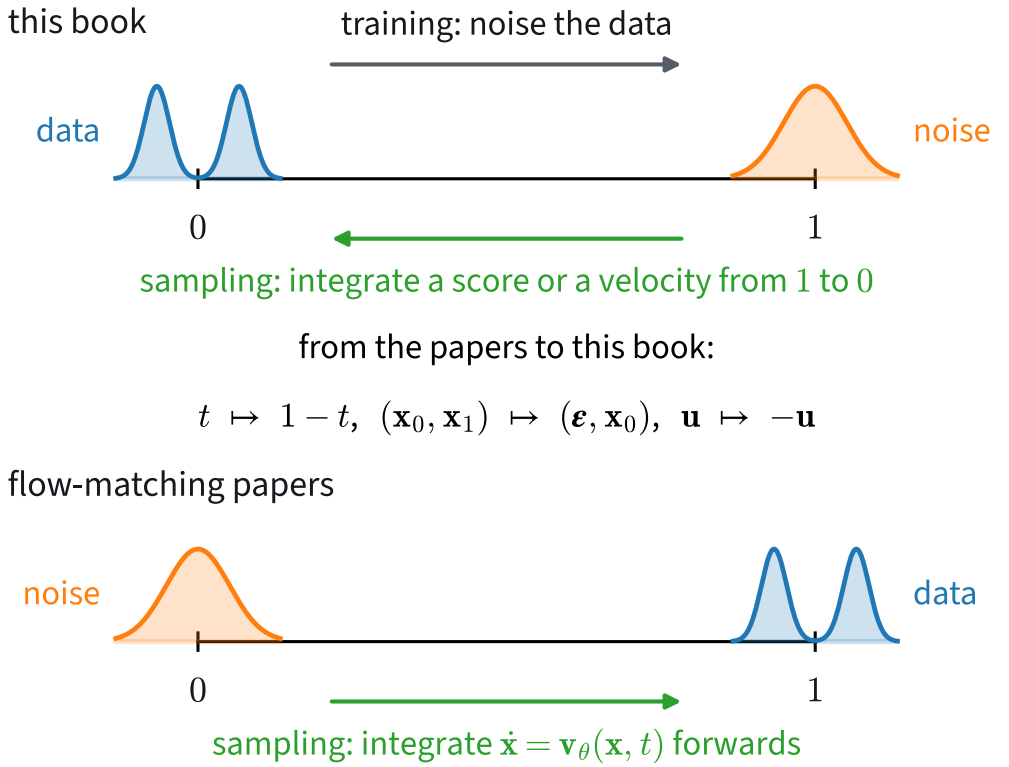


Figure 38.25: One clock and the papers' clock. This book (top) puts the data at $t = 0$ and the noise at $t = 1$, noises the data forward in time, and samples by integrating a score or a velocity from 1 to 0. The flow-matching papers (bottom) put the noise at 0 and the data at 1 and sample forward. The endpoint densities are the same; the substitution $t \mapsto 1 - t$ converts one clock into the other and flips the sign of every velocity.

The data are $\mathbf{x}_0 \sim p_0$, the noise is $\epsilon \sim p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$, and p_t is a path of densities between them. Statements about couplings hold for any two endpoint laws, so there we write the pair as $(\mathbf{x}_0, \mathbf{x}_1)$. A conditioning variable $\mathbf{z}$ labels per-sample paths; the velocity of one of them is $\mathbf{u}_t(\mathbf{x} \mid \mathbf{z})$, the velocity of the whole cloud is $\mathbf{u}_t(\mathbf{x})$, and a network $\mathbf{u}_\theta(\mathbf{x}, t)$ learns it. The running examples are the two points at ± 2, where everything has a closed form, and the two-moons data set in the plane, where we train a network.

38.6.1 Flow Matching

38.6.1.1 Probability Paths and Velocity Fields

Diffusion got its path from a noising process and then had to learn the score to run that process backward. Here the path is ours to choose. A **probability path** is a family of densities $(p_t)_{t\in[0,1]}$ with p_0 the data and p_1 the noise. A velocity field $\mathbf{u}_t$ *realizes* the path if the pair satisfies the continuity equation of Section 38.4.4, which says that a density moved by a velocity field changes by the net inflow of mass,

$$\partial_t p_t + \nabla \cdot (p_t\, \mathbf{u}_t) = 0.$$

Then, for a Lipschitz field, integrating $\dot{\mathbf{x}} = \mathbf{u}_t(\mathbf{x})$ from a sample of p_1 at $t = 1$ down to $t = 0$ yields a sample of p_t at every time, and of the data at $t = 0$. If a network $\mathbf{u}_\theta$ approximates $\mathbf{u}_t$, generation is an ODE solve from noise. Such a model is a continuous normalizing flow (Section 38.2.3) run backward in time: its flow map from $t = 1$ to $t = 0$ is the generator. Flow matching trains it without simulating the ODE. The natural objective is the **flow-matching loss**

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{t\sim\mathcal{U}[0,1],\ \mathbf{x}\sim p_t}\left[\,\left\|\mathbf{u}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x})\right\|^2\right]. \tag{38.89}$$

It is the mean squared error of the network against the true velocity, averaged over time and over the cloud. It regresses on a field we cannot evaluate, because $\mathbf{u}_t$ depends on the whole data distribution.

We therefore build the path out of pieces we can write down. A conditioning variable $\mathbf{z}$ is a data point, or a pair of a noise point and a data point. For each of its values, pick a small moving blob $p_t(\mathbf{x} \mid \mathbf{z})$ that sits on $\mathbf{z}$ at $t = 0$ and has spread into noise by $t = 1$, with a velocity $\mathbf{u}_t(\mathbf{x} \mid \mathbf{z})$ that we know in closed form. With $\mathbf{z} \sim q$, the path is the mixture $p_t(\mathbf{x}) = \int p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z}$. Which velocity moves the mixture?

Proposition (the marginal velocity is a posterior mean). *Suppose that for each* $\mathbf{z}$ *the density* $p_t(\cdot \mid \mathbf{z})$ *is positive and continuously differentiable in* $(\mathbf{x}, t)$*, that it satisfies the continuity equation* $\partial_t p_t(\mathbf{x} \mid \mathbf{z}) = -\nabla \cdot (p_t(\mathbf{x} \mid \mathbf{z})\, \mathbf{u}_t(\mathbf{x} \mid \mathbf{z}))$ *with a continuously differentiable velocity, and that the integrals below may be differentiated under the integral sign. Define on* $\{p_t > 0\}$ *the* **marginal velocity**

$$\mathbf{u}_t(\mathbf{x}) = \mathbb{E}\left[\mathbf{u}_t(\mathbf{x} \mid \mathbf{z}) \mid \mathbf{x}_t = \mathbf{x}\right] = \int \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})\, \frac{p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})}{p_t(\mathbf{x})}\, d\mathbf{z}. \tag{38.90}$$

Then $(p_t, \mathbf{u}_t)$ *satisfies the continuity equation.*

The velocity of the cloud at $\mathbf{x}$ is the average of the velocities of the blobs that could have put a particle at $\mathbf{x}$, each weighted by how likely it is to have done so.

Proof. Differentiate the mixture under the integral sign, substitute the conditional continuity equation, and pull the divergence out of the integral:

$$\partial_t p_t(\mathbf{x}) = \int \partial_t p_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z} = -\nabla \cdot \int p_t(\mathbf{x} \mid \mathbf{z})\, \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})\, q(\mathbf{z})\, d\mathbf{z} = -\nabla \cdot \left(p_t(\mathbf{x})\, \mathbf{u}_t(\mathbf{x})\right).$$

The last step multiplies and divides by $p_t(\mathbf{x})$ and recognizes Equation 38.90. ■

Where only one blob passes, the marginal velocity is that blob's velocity. Where several pass, it is their average, and this averaging is what will bend the flow.

38.6.1.2 The Conditional Flow Matching Theorem

The target of Equation 38.89 is the posterior mean Equation 38.90 of a quantity we can compute for every sample. By the regression lemma Equation 38.75, least squares against a per-sample target fits its conditional mean. We may therefore regress on the per-sample quantity, which is what the **conditional flow matching** (CFM) loss does:

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathbb{E}_{t,\, \mathbf{z}\sim q,\, \mathbf{x}\sim p_t(\cdot\mid\mathbf{z})} \left[\left\| \mathbf{u}_\theta(\mathbf{x}, t) - \mathbf{u}_t(\mathbf{x} \mid \mathbf{z}) \right\|^2 \right]. \tag{38.91}$$

It needs only a sample of $(t, \mathbf{z}, \mathbf{x})$ and the closed-form conditional velocity.

Theorem (CFM trains the marginal field). *Let $\mathbf{u}_t$ be the marginal velocity Equation 38.90 and suppose that $\mathbb{E}\|\mathbf{u}_t(\mathbf{x} \mid \mathbf{z})\|^2 < \infty$. Then*

$$\mathcal{L}_{\text{CFM}}(\theta) = \mathcal{L}_{\text{FM}}(\theta) + C$$

with C independent of θ. In particular the two objectives have the same gradients and the same minimizers (Lipman et al. 2023; Tong et al. 2024).

Proof. Apply the regression lemma with $X = (\mathbf{x}, t)$, where $\mathbf{x} \sim p_t(\cdot \mid \mathbf{z})$, and $Y = \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})$, which is square-integrable by hypothesis. The law of X is $t \sim \mathcal{U}[0,1]$ and $\mathbf{x} \sim p_t$, and the conditional mean of Y given X is the marginal velocity, by its definition Equation 38.90. The lemma splits $\mathcal{L}_{\text{CFM}}$ into $\mathbb{E}\|\mathbf{u}_\theta(X) - \mathbf{u}_t(\mathbf{x})\|^2 = \mathcal{L}_{\text{FM}}$ and $C = \mathbb{E}\|Y - \mathbf{u}_t(\mathbf{x})\|^2$, which does not involve θ. ■

The proof is one line because the lemma does the work, and the theorem needs no hypothesis on the path beyond a square-integrable target. It is the proposition that guarantees that the trained field carries p_1 to p_0 when integrated from $t = 1$ to $t = 0$. The constant C is the conditional variance of the target. It measures how much the per-sample velocities through a point disagree, so the CFM loss of a perfect field equals C, not zero. Denoising score matching has the same structure, with the score of the noising kernel as the per-sample quantity and the marginal score as its posterior mean.

38.6.1.3 Example: Two Points in One Dimension

Every object of the last two subsections has a closed form for the two-point data set of the opening, and the closed form shows the failure that Section 38.6.2 repairs. The data are $x_0 = \pm a$ with probability $\frac{1}{2}$ each, with $a = 2$: the two modes of the chapter's running mixture with their width removed. The noise is $\epsilon \sim \mathcal{N}(0, 1)$, drawn independently of

x_0. The simplest blob is a point sliding along a line, $x_t = (1-t)\,x_0 + t\,\epsilon$, with velocity $\dot{x}_t = \epsilon - x_0$.

Take the data point as the conditioning variable, $\mathbf{z} = x_0$. Given x_0, the position x_t is $\mathcal{N}((1-t)\,x_0, t^2)$, a positive density for $t > 0$, so the proposition applies. Eliminating $\epsilon = (x-(1-t)\,x_0)/t$ expresses the velocity as a function of the current position, $u_t(x \mid x_0) = (x-x_0)/t$. Run backward in time, the particle heads for its data point at the speed that arrives exactly at $t = 0$.

To average this velocity we need the posterior of x_0 at a point x. The posterior odds of $+a$ against $-a$ at x are the ratio of the two Gaussian densities. In the exponent the terms x^2 and $(1-t)^2a^2$ cancel and only the cross term survives, so the odds are $\exp\left(2(1-t)\,ax/t^2\right)$ and the posterior mean is $\mathbb{E}[x_0 \mid x_t = x] = a\tanh\left((1-t)\,ax/t^2\right)$. The marginal velocity is therefore

$$u_t(x) = \frac{x - a\tanh\left(\frac{(1-t)\,ax}{t^2}\right)}{t}. \tag{38.92}$$

Run backward in time, the cloud at x heads for the average of the two targets, weighted by how plausible each is from x, at the speed that arrives on time. At $(t, x) = (0.5, 0.5)$ the velocity is -2.86; at $(0.7, -1)$ it is 0.97. Three properties can be read off Equation 38.92, and Figure 38.26 draws them.

- **At $t = 1$ the field is $u_1(x) = x$.** The noise sample carries no information about its target, so the posterior mean is the data mean 0. One Euler step from $t = 1$ to $t = 0$, $x \mapsto x - u_1(x)$, sends *every* sample to 0. This holds for any data under independent pairing: $\mathbf{u}_1(\mathbf{x}) = \mathbb{E}[\epsilon - \mathbf{x}_0 \mid \epsilon = \mathbf{x}] = \mathbf{x} - \mathbb{E}[\mathbf{x}_0]$, so one step of the exact field lands on the data mean.
- **The origin first attracts, then repels.** As t falls, a sampler moves against u_t, so a positive slope at the origin pulls particles toward it and a negative slope pushes them away. The slope is $\partial_x u_t(0) = (1-(1-t)\,a^2/t^2)/t$: $+0.56$ at $t = 0.9$, zero at $t^* = 2\sqrt{2} - 2 \approx 0.83$ (the root of $(1-t)\,a^2 = t^2$), -0.31 at $t = 0.8$ and -14 at $t = 0.5$. Particles near 0 drift toward 0 until t falls to t^* and leave for a mode only later, so the flow chooses a mode late. Far from the origin the tanh saturates and $u_t \approx (x \mp a)/t$, the conditional velocity itself.
- **The floor C is finite although the conditional velocity is unbounded.** Here $C = \mathbb{E}_t\mathbb{E}_x\mathrm{Var}(x_0 \mid x_t)/t^2$ with $\mathrm{Var}(x_0 \mid x_t = x) = a^2\,\mathrm{sech}^2\left((1-t)\,ax/t^2\right)$, which is tiny except near the origin, where little mass remains as $t \to 0$ (3).

The cell compares the closed form with a direct posterior average at three points, takes the slope at the origin by a finite difference, and applies one Euler step of the exact field to $\epsilon = \pm 1$.

```
a = 2.0                                     # data at -a and +a, noise N(0, 1)
def u_two_point(x, t):                      # closed form of the marginal velocity
    return (x - a * np.tanh((1 - t) * a * x / t**2)) / t

def u_posterior(x, t):                      # posterior average of (x - x0)/t
    x0 = np.array([-a, a])
    w = np.exp(-(x - (1 - t) * x0)**2 / (2 * t**2))   # likelihood of x given x0
```

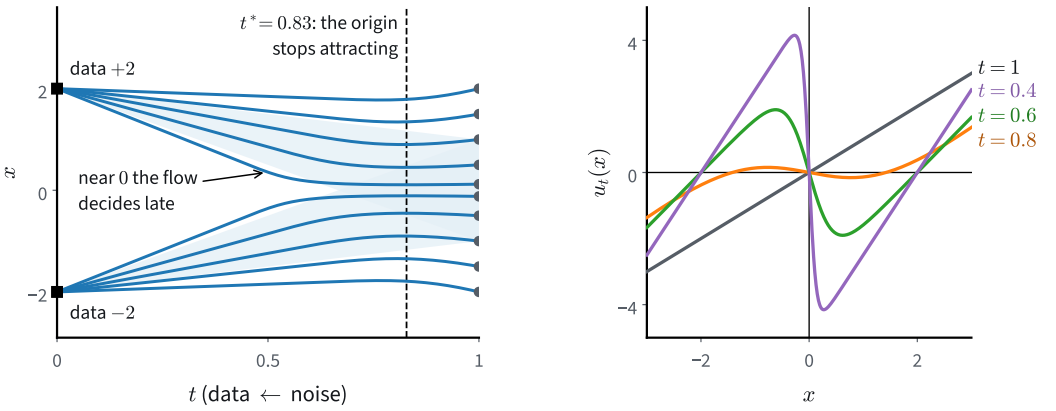


Figure 38.26: Two data points at ± 2 and standard Gaussian noise with independent pairing; sampling runs from the noise at $t = 1$ (right) to the data at $t = 0$ (left). (a) Trajectories of the exact marginal flow over the one-standard-deviation bands of the two conditional blobs: until t falls to $t^* = 0.83$ every trajectory moves toward 0, and those that start near 0 leave for a mode only below $t \approx 0.6$. (b) The velocity $u_t(x)$ is x at $t = 1$, its slope at the origin turns negative and steep as t falls, and it tends to $(x \mp 2)/t$ away from the origin.

```
    return (x - (w * x0).sum() / w.sum()) / t

points = [(0.5, 0.5), (0.7, -1.0), (0.2, 0.1)]
print('closed form - posterior average:',
      max(abs(u_two_point(x, t) - u_posterior(x, t)) for t, x in points))
ts = np.array([0.9, 0.8, 0.5])
print('slope at the origin at t = 0.9, 0.8, 0.5:',
      ((u_two_point(1e-6, ts) - u_two_point(-1e-6, ts)) / 2e-6).round(2),
      '   t* =', round(2 * np.sqrt(2) - 2, 4))
eps = np.array([-1.0, 1.0])
print('one Euler step from t = 1 to 0 from', eps, 'lands at', eps - u_two_point(eps, 1.0))
```

```
closed form - posterior average: 4.440892098500626e-16
slope at the origin at t = 0.9, 0.8, 0.5: [  0.56  -0.31 -14.  ]    t* = 0.8284
one Euler step from t = 1 to 0 from [-1.  1.] lands at [0. 0.]
```

The numbers are those of the formula, and one Euler step from either start lands on 0, the data mean. This is the limit of the construction. The conditional paths are straight, but the marginal flow is curved, and one step collapses every sample onto the mean. How to pair noise with data so that the flow comes out straight is the subject of Section 38.6.2. Before that, we settle how the velocity is related to the score of the previous section.

38.6.1.4 Score, Noise, and Velocity

Suppose a network has been trained to predict the noise, as diffusion models are. Do we need a second training run to obtain a velocity? We do not. On a Gaussian path the score, the noise, the clean point and the velocity are affine functions of one posterior mean, so each determines the others. Condition on a data point and take the path

$$\mathbf{x}_t = \alpha_t\,\mathbf{x}_0 + \sigma_t\,\epsilon, \qquad \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \tag{38.93}$$

with smooth coefficients (α_t, σ_t), $(\alpha_0, \sigma_0) = (1, 0)$ and $(\alpha_1, \sigma_1) = (0, 1)$. The blob around $\alpha_t \mathbf{x}_0$ has width σ_t: at $t = 0$ it is the data point, at $t = 1$ the noise. This is the noising kernel of the previous section, with the horizon set to 1.

The smallest case is Gaussian data $x_0 \sim \mathcal{N}(0, s_0^2)$ in one dimension on the cosine path $\alpha_t = \cos\frac{\pi t}{2}$, $\sigma_t = \sin\frac{\pi t}{2}$. The marginal is $\mathcal{N}(0, s_t^2)$ with $s_t^2 = \alpha_t^2 s_0^2 + \sigma_t^2$, the score is $-x/s_t^2$, and the velocity is $u_t(x) = (\dot{s}_t/s_t)\,x$: a Gaussian cloud only stretches or shrinks (5). With $s_0 = 2$ at $t = \frac{1}{2}$ we get $s_t^2 = 2.5$, and a particle at x moves with velocity $-0.94\,x$: run from $t = 1$ to 0, the cloud stretches from width 1 to width 2. With $s_0 = 1$ every p_t is the standard Gaussian and the cloud never moves: $u \equiv 0$.

Proposition (score–velocity identity). *For the Gaussian path Equation 38.93, at any t with $\alpha_t, \sigma_t > 0$ and on $\{p_t > 0\}$, the marginal velocity and the marginal score determine each other:*

$$\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\alpha}_t}{\alpha_t}\,\mathbf{x} - \left(\sigma_t\dot{\sigma}_t - \sigma_t^2\,\frac{\dot{\alpha}_t}{\alpha_t}\right)\nabla \log p_t(\mathbf{x}). \tag{38.94}$$

The velocity is a multiple of the position plus a multiple of the score, and the schedule alone fixes both multiples.

Proof. We show that both sides are affine in one posterior mean, $\hat{\mathbf{x}}_0 = \mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$. Differentiating the path gives the conditional velocity $\dot{\mathbf{x}}_t = \dot{\alpha}_t\mathbf{x}_0 + \dot{\sigma}_t\epsilon$, and $\epsilon = (\mathbf{x} - \alpha_t\mathbf{x}_0)/\sigma_t$. Taking the posterior mean as in Equation 38.90,

$$\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\sigma}_t}{\sigma_t}\,\mathbf{x} + \left(\dot{\alpha}_t - \alpha_t\,\frac{\dot{\sigma}_t}{\sigma_t}\right)\hat{\mathbf{x}}_0.$$

The marginal score is the posterior mean of the scores of the kernels, which is Tweedie's formula of the previous section. The kernel $\mathcal{N}(\alpha_t\mathbf{x}_0, \sigma_t^2\mathbf{I})$ has score $(\alpha_t\mathbf{x}_0 - \mathbf{x})/\sigma_t^2$, so

$$\nabla \log p_t(\mathbf{x}) = \mathbb{E}\left[\frac{\alpha_t\mathbf{x}_0 - \mathbf{x}}{\sigma_t^2}\,\middle|\,\mathbf{x}_t = \mathbf{x}\right] = \frac{\alpha_t\hat{\mathbf{x}}_0 - \mathbf{x}}{\sigma_t^2}, \qquad \text{i.e.} \quad \hat{\mathbf{x}}_0 = \frac{\mathbf{x} + \sigma_t^2\,\nabla \log p_t(\mathbf{x})}{\alpha_t}.$$

Substituting this $\hat{\mathbf{x}}_0$ into the velocity and collecting the $\mathbf{x}$ and $\nabla \log p_t$ terms gives Equation 38.94. ■

On a variance-preserving path, $\alpha_t^2 + \sigma_t^2 = 1$, so $\sigma_t\dot{\sigma}_t = -\alpha_t\dot{\alpha}_t$ and the coefficient of the score collapses to $\dot{\alpha}_t/\alpha_t$: $\mathbf{u}_t(\mathbf{x}) = \frac{\dot{\alpha}_t}{\alpha_t}(\mathbf{x} + \nabla \log p_t(\mathbf{x}))$. For the variance-preserving SDE of the previous section, $\alpha_t = \exp\big(-\frac{1}{2}\int_0^t \beta(s)\,ds\big)$, so $\dot{\alpha}_t/\alpha_t = -\frac{1}{2}\beta(t)$ and the velocity is $-\frac{1}{2}\beta(t)(\mathbf{x} + \nabla \log p_t(\mathbf{x}))$. This is the probability-flow velocity $\mathbf{f} - \frac{1}{2}g^2\nabla \log p_t$ of Equation 38.61 with $\mathbf{f} = -\frac{1}{2}\beta\,\mathbf{x}$ and $g^2 = \beta$: on this path the flow-matching velocity and the probability-flow ODE are the same field on the same clock. On the straight path $\alpha_t = 1 - t$, $\sigma_t = t$ the identity reads $\mathbf{u}_t(\mathbf{x}) = -(\mathbf{x} + t\,\nabla \log p_t(\mathbf{x}))/(1 - t)$. For the Gaussian example, $\nabla \log p_t = -x/s_t^2$ turns the variance-preserving form into $\frac{\dot{\alpha}_t}{\alpha_t}(1 - 1/s_t^2)\,x = \alpha_t\dot{\alpha}_t(s_0^2 - 1)\,x/s_t^2$, the stretching rate $\dot{s}_t/s_t$ found above.

The same posterior mean underlies the targets that networks are trained on in practice (Table 38.2).

Table 38.2: Network targets on a Gaussian path. Each row gives the quantity a network predicts, the posterior mean it estimates, and its conversion from the score. The name **v**-prediction comes from the literature; on the cosine path its target $\alpha_t\epsilon - \sigma_t\mathbf{x}_0$ is $2/\pi$ times the velocity target.

Network predicts	Posterior mean	From the score $\mathbf{s} = \nabla \log p_t$
score $\mathbf{s}_\theta$	$\nabla \log p_t(\mathbf{x})$	$\mathbf{s}$
noise $\hat{\epsilon}$	$\mathbb{E}[\epsilon \mid \mathbf{x}_t = \mathbf{x}]$	$-\sigma_t\,\mathbf{s}$
clean point $\hat{\mathbf{x}}_0$	$\mathbb{E}[\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$	$(\mathbf{x} + \sigma_t^2\,\mathbf{s})/\alpha_t$
velocity $\mathbf{u}_\theta$	$\mathbb{E}[\dot{\mathbf{x}}_t \mid \mathbf{x}_t = \mathbf{x}]$	the identity Equation 38.94
v-prediction $\hat{\mathbf{v}}$	$\mathbb{E}[\alpha_t\epsilon - \sigma_t\mathbf{x}_0 \mid \mathbf{x}_t = \mathbf{x}]$	$-\frac{\sigma_t}{\alpha_t}\,\mathbf{x} - \sigma_t\,\frac{\alpha_t^2+\sigma_t^2}{\alpha_t}\,\mathbf{s}$

Each row is a t-dependent affine function of the same posterior mean, so a network trained on any row defines all of them. Which row is easiest to learn is a separate question, about the variance of the training targets at the two ends of the path (8).

As a check of the algebra, the cell computes the velocity of the running mixture (components at ± 2 with variance 0.25) at $t = 0.4$ on the cosine path twice: from the posterior mean, and from Equation 38.94 with the analytic score.

```
t = 0.4                                      # one fixed time, cosine schedule
alpha, sigma = np.cos(np.pi * t / 2), np.sin(np.pi * t / 2)
dalpha, dsigma = -np.pi / 2 * sigma, np.pi / 2 * alpha
grid = np.linspace(-4.0, 4.0, 201)
means, var0 = np.array([-2.0, 2.0]), 0.25    # the running mixture
vart = alpha**2 * var0 + sigma**2            # noised component variance
w = np.stack([np.exp(-(grid - alpha * m)**2 / (2 * vart)) for m in means])
w /= w.sum(0)                                # posterior mode responsibilities
x0_hat = (w * np.stack([m + alpha * var0 / vart * (grid - alpha * m)
                        for m in means])).sum(0)          # E[x_0 | x_t]
u_posterior = dalpha * x0_hat + dsigma / sigma * (grid - alpha * x0_hat)
score = (w * np.stack([(alpha * m - grid) / vart for m in means])).sum(0)
u_dictionary = (dalpha / alpha) * grid \
    - (sigma * dsigma - sigma**2 * dalpha / alpha) * score
print(f'max |u_posterior - u_dictionary| on [-4, 4]: '
      f'{np.abs(u_posterior - u_dictionary).max():.1e}')
```

```
max |u_posterior - u_dictionary| on [-4, 4]: 3.1e-15
```

The two routes agree to rounding error. A noise predictor is therefore also a velocity field: the samplers of this section apply to it after the conversion of Table 38.2.

38.6.2 Rectified Flow

38.6.2.1 Straight Paths and Their Crossings

The two-point example used straight conditional paths. In d dimensions the same construction pairs a data sample $\mathbf{x}_0$ with a noise sample ϵ, drawn independently, and joins them by

a straight line traversed at constant speed:

$$\mathbf{x}_t = (1-t)\,\mathbf{x}_0 + t\,\epsilon, \qquad \mathbf{u}_t(\mathbf{x}_t \mid \mathbf{z}) = \dot{\mathbf{x}}_t = \epsilon - \mathbf{x}_0, \qquad \mathbf{z} = (\mathbf{x}_0, \epsilon). \tag{38.95}$$

The conditional velocity does not depend on t, and the CFM loss Equation 38.91 becomes the **rectified flow** objective (Liu et al. 2023; Lipman et al. 2023)

$$\mathcal{L}_{\mathrm{RF}}(\theta) = \mathbb{E}_{t,\,\mathbf{x}_0\sim p_0,\,\epsilon\sim p_1}\left[\left\|\mathbf{u}_\theta((1-t)\mathbf{x}_0 + t\epsilon,\ t) - (\epsilon - \mathbf{x}_0)\right\|^2\right]. \tag{38.96}$$

Training draws data and noise, interpolates between them, and regresses on the difference. With Gaussian noise we could condition on $\mathbf{x}_0$ alone, as in the two-point example, and the proposition would apply. Conditioning on the pair is what lets us change the pairing later, and it has a price: given $\mathbf{z}$, a point sliding along a line has no density, so the proposition does not apply as stated. Its conclusion still holds, by the same computation applied to the expectation $\mathbb{E}\,\varphi(\mathbf{x}_t)$ of a smooth test function φ (9).

Although every conditional path is straight, the marginal flow is not (Figure 38.27). Two segments that cross at $(\mathbf{x}, t)$ feed the posterior mean Equation 38.90 two different directions. A function has one value at each point, so the field averages them. An ODE whose field is Lipschitz in $\mathbf{x}$ has exactly one trajectory through each point (Section 38.1.1.3), so two trajectories of the learned flow can never meet. Where the segments cross, the trajectories bend away from each other instead.

The goal is a field whose trajectories are straight lines traversed at constant speed. Such a field is constant along each trajectory, so one Euler step of any length is exact. The independent pairing is far from this limit, as the two-point example already showed. The bend comes from the pairing and not from the loss, because every conditional path is already a straight line.

38.6.2.2 Example: Two Moons

The two moons are two interleaved crescents in the plane. One Euler step of the exact field would collapse the noise onto their mean, and we can count how many steps a trained field needs instead. We score samples by the **squared energy distance** (Székely and Rizzo 2013) between laws P and Q with finite first moments,

$$\mathcal{E}^2(P,Q) = 2\,\mathbb{E}\|X-Y\| - \mathbb{E}\|X-X'\| - \mathbb{E}\|Y-Y'\|,$$

where $X, X' \sim P$ and $Y, Y' \sim Q$ are independent. It compares the mean distance between a point of P and a point of Q with the mean distances within each law. It is the squared MMD of Section 37.4.1 with the kernel $k(\mathbf{x},\mathbf{y}) = \|\mathbf{x}\| + \|\mathbf{y}\| - \|\mathbf{x}-\mathbf{y}\|$, and it is zero only if $P = Q$. We estimate it from two samples by the unbiased U-statistic, and every score quoted below is this squared distance.

The cell builds the data, standardized to zero mean and unit variance per coordinate, a held-out sample of $2{,}048$ points, and one noise array `z_eval` that every later comparison reuses.

```
def two_moons(n, rng, noise=0.07):
    t = rng.uniform(0.0, np.pi, n)
```

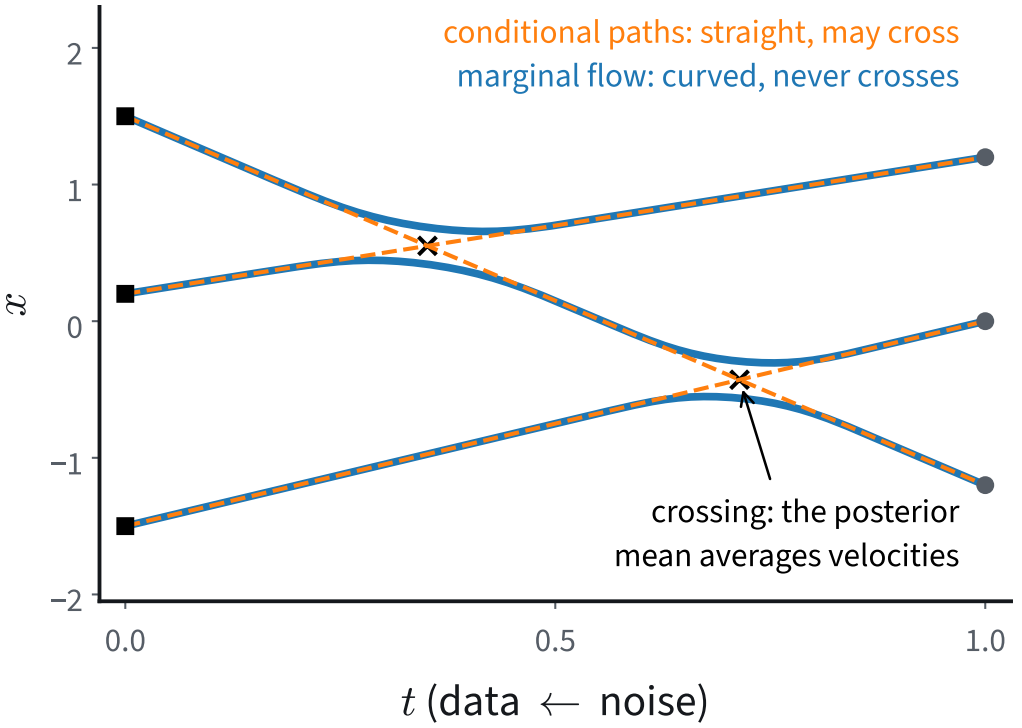


Figure 38.27: Three data–noise pairs in one dimension joined by straight segments (dashed), two pairs of which cross (×). Where segments cross, the marginal velocity averages their directions, so the trajectories of its flow (solid), run from the noise at $t = 1$ to the data at $t = 0$, bend and never meet. Each trajectory ends on a different data point than its own segment, in sorted order: in one dimension the flow's pairing is the sorted one, which is the optimal-transport coupling.

```
    m = rng.integers(0, 2, n)                    # which moon
    X = np.stack([np.cos(t) * (1 - 2 * m) + m,
                  np.sin(t) * (1 - 2 * m) + 0.5 * m], axis=1)
    return X + noise * rng.standard_normal((n, 2))

rng = np.random.default_rng()
raw = two_moons(8192, rng)
mu, sd = raw.mean(0), raw.std(0)
moons = (raw - mu) / sd                          # standardized training data
held_out = (two_moons(2048, rng) - mu) / sd
z_eval = rng.standard_normal((2048, 2))          # noise for every comparison

def energy_distance(a, b):                       # squared energy distance, U-statistic
    d = lambda u, v: np.linalg.norm(u[:, None, :] - v[None, :, :], axis=-1)
    n, m = len(a), len(b)
    return (2 * d(a, b).mean() - d(a, a).sum() / (n * (n - 1))
            - d(b, b).sum() / (m * (m - 1)))

print(f'fresh data sample vs held-out: {energy_distance(moons[:2048], held_out):+.4f}')
print(f'point mass at the data mean vs held-out: '
      f'{energy_distance(np.zeros((2048, 2)), held_out):.3f}')
```

```
fresh data sample vs held-out: +0.0005
point mass at the data mean vs held-out: 0.905
```

Two samples of the same law score within a few thousandths of zero, which is the resolution of the metric at this sample size. A point mass at the data mean, which is what one Euler step of the exact field produces, scores about 0.9.

The model is a small tanh network that takes $(\mathbf{x}, t)$ and returns a velocity; `field` evaluates it. The function `train_rf` takes a sampler of pairs, so that we can change the pairing later. It returns the network together with its loss averaged over the last 500 batches, because single batches scatter, and over the two coordinates, so the printed value is $\mathcal{L}_{\mathrm{RF}}/2$.

```
def make_net():                          # 3 -> 64 -> 64 -> 2, Glorot weights, zero biases
    net = nn.Sequential(nn.Linear(3, 64), nn.Tanh(), nn.Linear(64, 64), nn.Tanh(),
                        nn.Linear(64, 2))
    for layer in net[::2]:
        nn.init.xavier_uniform_(layer.weight)
        nn.init.zeros_(layer.bias)
    return net

def field(net, x, t):                    # the time enters centered, as 2t - 1
    return net(torch.cat([x, 2 * t - 1], 1))

def train_rf(sample_pairs, steps=4000, tail=500):
    """Regress v(x_t, t) on eps - x0 (papers: x1 - x0 at time 1 - t).
    Return the net and the mean tail loss."""
    net = make_net()
    opt = torch.optim.Adam(net.parameters(), lr=3e-3)
    sched = torch.optim.lr_scheduler.CosineAnnealingLR(opt, steps)   # 3e-3 -> 0
    tail_sum = torch.zeros(())
```

```
    for step in range(steps):
        x0, eps = sample_pairs(256)               # data, noise
        t = torch.rand(256, 1)
        xt = (1 - t) * x0 + t * eps               # a point on each segment
        loss = ((field(net, xt, t) - (eps - x0))**2).mean()
        opt.zero_grad()
        loss.backward()
        opt.step()
        sched.step()
        if step >= steps - tail:
            tail_sum += loss.detach()             # stays on the device
    return net, (tail_sum / tail).item()
```

The first run pairs noise and data independently. By the CFM theorem its loss cannot reach zero, and where it stops measures how much the segments cross.

```
data = torch.tensor(moons, dtype=torch.float32)
def independent_pairs(n):                         # random data rows, fresh noise
    return data[torch.randint(0, len(data), (n,))], torch.randn(n, 2)

net, plateau = train_rf(independent_pairs)
print(f'CFM loss, mean of the last 500 batches: {plateau:.2f}')
```

```
CFM loss, mean of the last 500 batches: 1.42
```

The loss settles at about 1.4 per coordinate and cannot go much lower. The floor C is the conditional variance of $\epsilon - \mathbf{x}_0$ given $\mathbf{x}_t$, averaged over t. For this data set it can be computed by weighting every training point by its posterior probability given $\mathbf{x}_t$; it is 1.38 per coordinate (1), and the network sits a few hundredths above it. A loss of 1.4 therefore does not mean a poor fit. Almost all of it is the disagreement of crossing segments, which no network can remove.

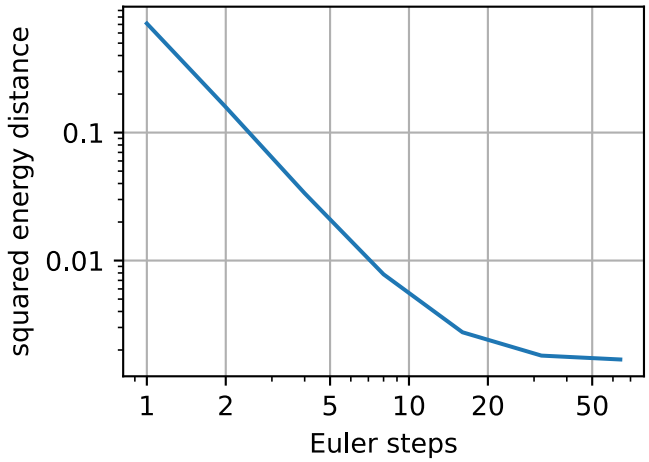
Sampling integrates $\dot{\mathbf{x}} = \mathbf{u}_\theta(\mathbf{x}, t)$ from $t = 1$ down to 0 with Euler's method (Section 38.1.3): K steps of length $1/K$, each of which subtracts $\mathbf{u}_\theta(\mathbf{x}, t)/K$ at $t = 1, 1 - 1/K, \dots, 1/K$. The function `euler_path` returns the whole trajectory, of shape `(steps + 1, n, 2)`; its last row is the sample.

```
@torch.inference_mode()
def euler_path(net, z, steps):
    q = torch.tensor(z, dtype=torch.float32)
    path = [q]
    for k in range(steps):                        # t = 1, 1 - 1/steps, ..., 1/steps
        t = torch.full((len(z), 1), 1 - k / steps)
        q = q - field(net, q, t) / steps
        path.append(q)
    return torch.stack(path).numpy()
```

One step of the exact field would give the point mass that scored 0.9. For the trained field, the cell integrates the noise array `z_eval` with $K = 1, 2, \dots, 64$ steps and plots the squared energy distance to the held-out data on log–log axes.

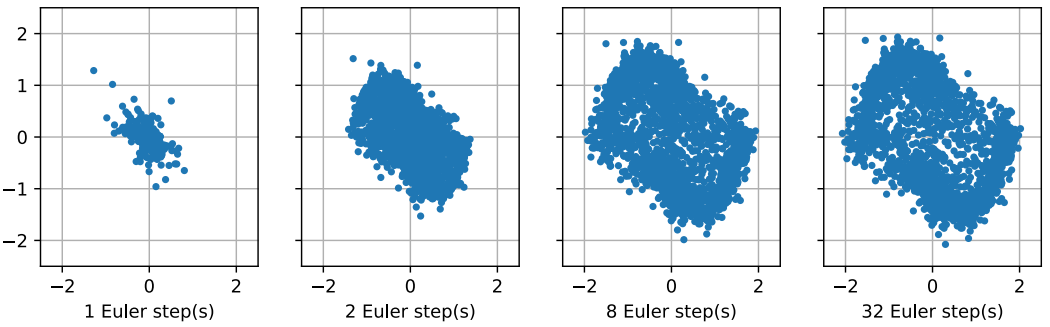
```
steps_list = [1, 2, 4, 8, 16, 32, 64]
eds = [energy_distance(euler_path(net, z_eval, K)[-1], held_out) for K in steps_list]
print('  '.join(f'{K}: {e:.3f}' for K, e in zip(steps_list, eds)))
d2l.plot(steps_list, eds, 'Euler steps', 'squared energy distance',
         xscale='log', yscale='log')
```

```
1: 0.712  2: 0.158  4: 0.034  8: 0.008  16: 0.003  32: 0.002  64: 0.002
```



One step scores about 0.7, the same order as the point mass at the mean (0.9): the network has learned, imperfectly, the collapse that the exact field performs at $t = 1$. Two steps score about 0.15, four about 0.04, eight about 0.01, and from sixteen steps on the score levels off, usually at a few thousandths and never much above 0.01. The decline is Euler's discretization error. The level that remains is the error of the trained field; it varies with the training run and is often as small as the metric can resolve with $2{,}048$ points. The samples show the same progression.

```
fig, axes = d2l.plt.subplots(1, 4, figsize=(10, 2.6), sharex=True, sharey=True)
for ax, K in zip(axes, (1, 2, 8, 32)):
    s = euler_path(net, z_eval, K)[-1]
    d2l.plot(s[:, 0], s[:, 1], f'{K} Euler step(s)', fmts=('.',),
             xlim=(-2.5, 2.5), ylim=(-2.5, 2.5), axes=ax)
```



One step gives a blob around the mean, two a bent ellipse, eight recognizable moons, and thirty-two sharp ones. The curvature that the independent pairing builds into the flow costs sixteen to thirty-two network calls per sample.

38.6.2.3 Reflow

The independent pairing is what bends the flow, and the trained model supplies a better one. Run its ODE from fresh noise and pair each noise point ϵ with the endpoint $\hat{\mathbf{x}}_0(\epsilon)$ at which it arrives. Trajectories of one flow never cross, so these pairs cross far less than independent ones. Retraining the same architecture on the new pairs is called **reflow** (Liu et al. 2023). It keeps both endpoint distributions and does not increase the cost of transport, for every convex cost function.

Proposition (reflow keeps the marginals and lowers the transport cost). *Let $(\mathbf{X}_0, \mathbf{X}_1) \sim \pi$ have finite second moments, let $\mathbf{X}_t = (1-t)\mathbf{X}_0 + t\mathbf{X}_1$, let $\mathbf{u}_t$ be the marginal velocity of this straight path, and let $\mathbf{Z}_t$ solve $\dot{\mathbf{Z}}_t = \mathbf{u}_t(\mathbf{Z}_t)$ backward in time from the noise end, $\mathbf{Z}_1 = \mathbf{X}_1$. Assume that $\mathbf{X}_t$ has a positive density for every $t > 0$ and that $\mathbf{u}_t$ is continuously differentiable and Lipschitz in $\mathbf{x}$ with a constant independent of t, so that this flow exists. Then (i) $\mathbf{Z}_t$ has the law of $\mathbf{X}_t$ for every t, in particular $\mathbf{Z}_0 \sim p_0$; (ii) for every convex function c,*

$$\mathbb{E}\, c(\mathbf{Z}_1 - \mathbf{Z}_0) \;\le\; \mathbb{E}\, c(\mathbf{X}_1 - \mathbf{X}_0). \tag{38.97}$$

Proof. (i) The law of $\mathbf{X}_t$ satisfies the continuity equation with $\mathbf{u}_t$, by the marginal-velocity proposition with $\mathbf{z} = (\mathbf{X}_0, \mathbf{X}_1)$ in the weak form that covers point masses (9). The law of $\mathbf{Z}_t$ satisfies it too, because $\mathbf{Z}_t$ moves with the field $\mathbf{u}_t$. Both equal p_1 at $t = 1$, and two solutions of the continuity equation with such a field that agree at one time agree at every time. This uniqueness is the one step we do not prove in full: Section 38.4.4.3 gave the argument for smooth positive densities, where along each trajectory both log-densities change at the rate $-\nabla \cdot \mathbf{u}_t$, and we use it here without redoing it in the weak form. (ii) Write the displacement as a time integral and apply Jensen's inequality twice:

$$\mathbb{E}\, c(\mathbf{Z}_1 - \mathbf{Z}_0) = \mathbb{E}\, c\Big(\int_0^1 \mathbf{u}_t(\mathbf{Z}_t)\, dt \Big) \le \int_0^1 \mathbb{E}\, c\big(\mathbf{u}_t(\mathbf{Z}_t)\big)\, dt = \int_0^1 \mathbb{E}\, c\big(\mathbb{E}[\mathbf{X}_1 - \mathbf{X}_0 \mid \mathbf{X}_t]\big)\, dt \le \mathbb{E}\, c(\mathbf{X}_1 - \mathbf{X}_0).$$

The first inequality is Jensen's inequality for the time average. The equality uses (i) to replace $\mathbf{Z}_t$ by $\mathbf{X}_t$ and then the definition of the marginal velocity. The last inequality is the conditional Jensen inequality. ■

With $c = \|\cdot\|^2$ the last step loses exactly the conditional variance, so the cost falls at least by the floor of the old pairing: $\mathbb{E}\|\mathbf{Z}_1 - \mathbf{Z}_0\|^2 \le \mathbb{E}\|\mathbf{X}_1 - \mathbf{X}_0\|^2 - C$. Re-pairing helps only where segments cross, because there the conditional velocity is not a function of $(\mathbf{x}_t, t)$.

The hypotheses are stronger than our examples allow. For the two points, and for any finite training set paired independently with Gaussian noise, the marginal field is Lipschitz on $[\delta, 1]$ for each $\delta > 0$ but not up to $t = 0$: the slope -14 of the two-point field at $t = 0.5$ grows without bound as $t \to 0$. The proof then gives (i) for every $t \ge \delta$ and (ii) for the displacement from $t = 1$ to $t = \delta$, divided by $1 - \delta$. Letting $\delta \to 0$ is a further step that we do not prove, and the examples that follow go beyond the proposition in this sense.

For the two points the conclusion can be verified directly. A flow in one dimension preserves order, so one round pairs each noise point ϵ with $a\,\mathrm{sign}(\epsilon)$ (6). This pairing costs $a^2 - 2a\sqrt{2/\pi} + 1 = 1.81$ against $a^2 + 1 = 5$ for the independent one, and its segments never cross, so the second round has $C = 0$ and its field is integrated exactly by one Euler step.

For the two moons, the cell draws $8{,}192$ fresh noise points, integrates the trained model for 32 steps to obtain their endpoints, and trains a new network with `train_rf` on these fixed pairs.

```
eps_pairs = np.random.standard_normal((8192, 2))     # noise endpoints, kept
x0_pairs = euler_path(net, eps_pairs, 32)[-1]         # the model's own endpoints
X0, Eps = (torch.tensor(p, dtype=torch.float32) for p in (x0_pairs, eps_pairs))
def model_pairs(n):                                   # the same pairs, never re-paired
    i = torch.randint(0, len(X0), (n,))
    return X0[i], Eps[i]

net2, plateau2 = train_rf(model_pairs)
print(f'reflow loss, mean of the last 500 batches: {plateau2:.4f}')
```

```
reflow loss, mean of the last 500 batches: 0.0032
```

The reflow loss settles at a few thousandths instead of 1.4, so the new segments almost never cross. A field without crossings is nearly constant along its trajectories, and then a few Euler steps should do. The next cell samples from both models with 1, 2 and 32 steps from the same noise array, and prints the mean squared length of the old and the new pairs, the cost that the proposition bounds.

```
for K in (1, 2, 32):
    e1 = energy_distance(euler_path(net, z_eval, K)[-1], held_out)
    e2 = energy_distance(euler_path(net2, z_eval, K)[-1], held_out)
    print(f'{K:2d} Euler step(s): energy distance  original {e1:.3f}  reflowed {e2:.3f}')
x0_indep = moons[np.random.randint(0, len(moons), len(eps_pairs))]
print(f'mean squared pair length: independent {np.mean(np.sum((eps_pairs - x0_indep)**2,
↪  1)):.2f}'
      f', reflowed {np.mean(np.sum((eps_pairs - x0_pairs)**2, 1)):.2f}')
```

```
 1 Euler step(s): energy distance  original 0.712  reflowed 0.002
 2 Euler step(s): energy distance  original 0.158  reflowed 0.002
32 Euler step(s): energy distance  original 0.002  reflowed 0.002
mean squared pair length: independent 4.01, reflowed 0.28
```

One Euler step of the reflowed model scores the same as thirty-two steps of the original, usually a few thousandths, and more than a hundred times below the original's single step. The reflowed network was trained to reproduce the original's 32-step endpoints, so matching their quality in one step is the best it can do. It inherits the first model's error.

The mean squared length of the pairs drops from 4 to about 0.3. For the exact marginal flow the proposition guarantees a drop at least to $4 - 2.76 = 1.24$, since the floor of the independent pairing is $2 \times 1.38 = 2.76$ summed over the two coordinates. The measured drop is far larger than the bound requires, because the first step of the proof, Jensen's inequality for the time average, is far from tight on the curved trajectories of the first flow. (The bound is stated for the exact marginal flow, which the trained network and its 32 Euler steps only approximate.)

Straightness can be measured as well, by the mean distance of each 32-step trajectory from the chord joining its two ends.

```
def chord_gap(P):                                    # P: (steps + 1, n, 2) trajectory
    chord = P[0] + (P[-1] - P[0]) * np.linspace(0, 1, len(P))[:, None, None]
    return np.linalg.norm(P - chord, axis=-1).mean()

print(f'mean distance from the chord: original {chord_gap(euler_path(net, z_eval, 32)):.3f}'
      f', reflowed {chord_gap(euler_path(net2, z_eval, 32)):.3f}')
```

```
mean distance from the chord: original 0.246, reflowed 0.013
```

The original trajectories wander about a quarter of a unit from their chords, first toward the mean as the two-point example predicts and then out to the moons. The reflowed ones stay about a hundredth from theirs, which is why one step suffices. Reflow straightens a flow, but it does not make it optimal. In more than one dimension a flow can be straight and crossing-free and still move mass farther than it must.

38.6.3 Optimal Transport and Straightness

38.6.3.1 Couplings and the Wasserstein Distance

Reflow lowered the transport cost. Which pairing is cheapest, and is its flow straight? We restate the vocabulary of Section 37.4.2. A **coupling** π of p_0 and p_1 is a joint law of pairs with these marginals, a plan that says how much mass travels from each source point to each target point. The **2-Wasserstein distance** is the cost of the cheapest plan under squared distance,

$$W_2^2(p_0, p_1) = \min_{\pi \in \Pi(p_0, p_1)} \mathbb{E}_{(\mathbf{x}_0, \mathbf{x}_1) \sim \pi} \left[\|\mathbf{x}_1 - \mathbf{x}_0\|^2 \right], \tag{38.98}$$

where $\Pi(p_0, p_1)$ is the set of couplings. The distance is the least mean squared distance by which the mass must move. When one of the two laws has a density and both have finite second moments, the optimal plan is unique and is a map from the law with the density. For us that law is the noise p_1, and the map is $\mathbf{x}_0 = \nabla\psi(\mathbf{x}_1)$ for a convex function ψ. This is Brenier's theorem (Brenier 1991). The data need no density, and the two points and the two moons have none.

The smallest case is a Gaussian in one dimension, with data $\mathcal{N}(m, s^2)$, $s > 0$, and noise $\mathcal{N}(0, 1)$. The sorted pairing sends each quantile to the same quantile, $x_0 = m + s\epsilon$, and costs $\mathbb{E}(m + s\epsilon - \epsilon)^2 = m^2 + (s-1)^2$. The independent pairing costs $\mathbb{E}(\epsilon - x_0)^2 = m^2 + 1 + s^2$. In one dimension the sorted pairing is optimal for every convex cost: if two pairs cross, swapping their targets does not increase the cost (4). Hence

$$W_2^2(\mathcal{N}(0,1), \mathcal{N}(m, s^2)) = m^2 + (s-1)^2. \tag{38.99}$$

The distance separates the cost of shifting the mean from the cost of stretching the width. Between $\mathcal{N}(0, 4)$ and $\mathcal{N}(0, 1)$ it is 1, against 5 for independent pairs: the independent pairing spends four fifths of its cost moving mass past other mass.

This explains where the trajectories of Figure 38.27 end. In one dimension the trajectories of a flow cannot cross, so a flow preserves order, and the order-preserving pairing is the optimal one. On a line, one round of reflow therefore reaches optimal transport (6).

In two dimensions, freedom from crossings is not enough. Rotate a standard Gaussian cloud about the origin by a quarter turn in unit time, with the field that assigns to the point (ξ_1, ξ_2) the velocity $\frac{\pi}{2}(-\xi_2, \xi_1)$. No two trajectories cross, and the cloud ends where it started, so $W_2^2(p_0, p_1) = 0$. Yet every particle moves, at speed $\frac{\pi}{2}\|\mathbf{x}\|$. The cost of a whole flow, not only of its endpoints, is its **kinetic energy** $\int_0^1 \mathbb{E}\|\mathbf{u}_t(\mathbf{x}_t)\|^2\, dt$, the mean squared speed of the particles integrated over time. For the rotation it is $(\pi/2)^2\, \mathbb{E}\|\mathbf{x}\|^2 = \pi^2/2 \approx 4.9$ (Figure 38.28).

Straightening the arcs does not cure this. Pair each $\mathbf{x}_0$ with its quarter turn $\mathbf{R}\mathbf{x}_0$ and join the two by a segment. The segments $((1-t)\mathbf{I} + t\mathbf{R})\mathbf{x}_0$ never cross, because that matrix is invertible for every t, and each is traversed at constant speed. So $C = 0$ and reflow leaves the pairing unchanged, yet it costs $\mathbb{E}\|\mathbf{R}\mathbf{x}_0 - \mathbf{x}_0\|^2 = 4$ against $W_2^2 = 0$. On a line, crossings are the only way to waste motion. In the plane a pairing can be straight, crossing-free and still wrong.

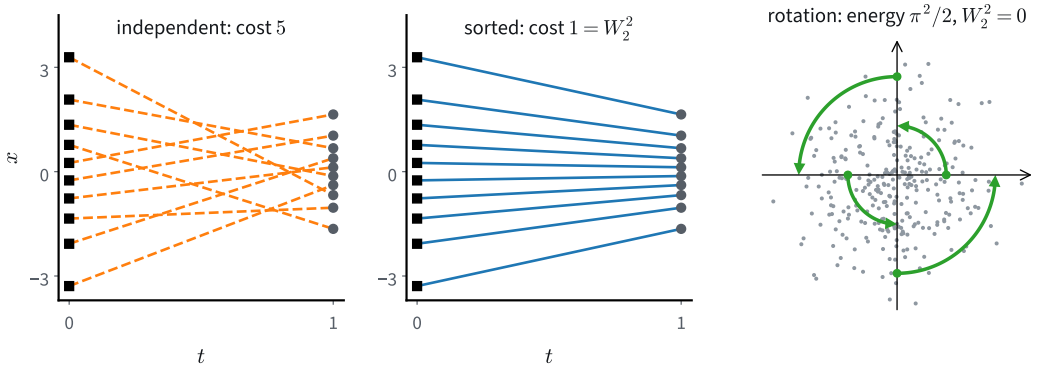


Figure 38.28: Three ways to move a Gaussian. (a) Independent pairing of data $\mathcal{N}(0, 4)$ at $t = 0$ with noise $\mathcal{N}(0, 1)$ at $t = 1$, drawn for the ten deciles: segments cross, and the mean squared displacement is 5. (b) The sorted pairing $x_0 = 2\epsilon$ never crosses and costs 1, the squared Wasserstein distance. (c) A quarter rotation of a standard Gaussian cloud in two dimensions ends where it started, so $W_2^2 = 0$, yet its kinetic energy is $\pi^2/2$: in two dimensions a crossing-free flow need not be optimal.

Kinetic energy also ranks flows with the same endpoints. Take three flows between data $\mathcal{N}(0, 4)$ and noise $\mathcal{N}(0, 1)$: the straight flow of the sorted pairing, the marginal flow of straight segments under the independent pairing, and the cosine path of Section 38.6.1.4. The first must be cheapest, but which of the other two comes second? All three have Gaussian marginals $\mathcal{N}(0, s_t^2)$, so their velocity is $(\dot{s}_t/s_t)\, x$ and the kinetic energy is $\int_0^1 \dot{s}_t^2\, dt$ (5). The cell also checks the two pairing costs by sorting samples.

```
n = 200_000
x0, eps = 2 * np.random.standard_normal(n), np.random.standard_normal(n)
print(f'data N(0,4), noise N(0,1): independent pairing {np.mean((eps - x0)**2):.2f}, '
      f'sorted pairing {np.mean((np.sort(eps) - np.sort(x0))**2):.3f}')
```

```
t = np.linspace(0, 1, 100_001)
def kinetic_energy(s):                          # marginals N(0, s_t^2): u_t(x) = (s'/s) x
    return np.sum(np.diff(s)**2 / np.diff(t))  # the integral of (ds/dt)^2

for name, s in [('sorted, straight', 2 - t),
                ('independent, straight', np.sqrt(4 * (1 - t)**2 + t**2)),
                ('cosine path', np.sqrt(1 + 3 * np.cos(np.pi * t / 2)**2))]:
    print(f'{name:22s} kinetic energy {kinetic_energy(s):.3f}')
```

```
data N(0,4), noise N(0,1): independent pairing 5.00, sorted pairing 0.996
sorted, straight       kinetic energy 1.000
independent, straight  kinetic energy 1.858
cosine path            kinetic energy 1.234
```

The sorted straight flow attains $W_2^2 = 1$. The straight segments of the independent pairing produce a marginal flow that costs 1.86, more than the cosine path's 1.23. Its width $s_t^2 = 4(1-t)^2 + t^2$ first shrinks, as t falls from 1 to 0.8, and then grows, so its particles move inward and back out. Straight conditional segments under the wrong pairing do not make a straight flow.

38.6.3.2 The Benamou–Brenier Theorem

The cheapest of the three flows reached W_2^2. This is a general fact, and it connects the static problem Equation 38.98, which pairs endpoints, to the objects that flow matching trains, which are flows.

Theorem (Benamou–Brenier). *Let one endpoint law, here the noise p_1, have a density, let p_0 and p_1 have finite second moments, and call a pair $(p_t, \mathbf{u}_t)$ admissible if $\mathbf{u}_t$ is continuously differentiable and Lipschitz in $\mathbf{x}$ with a constant independent of t, and the pair satisfies the continuity equation with the endpoints p_0 and p_1. (i) Every admissible pair has kinetic energy at least $W_2^2(p_0, p_1)$. (ii) If the marginal velocity of the straight interpolation of an optimal coupling is admissible (for example, when the Brenier map is a smooth diffeomorphism whose derivative and inverse derivative are bounded), that flow has kinetic energy W_2^2, and then, with the minimum over admissible pairs,* (Benamou and Brenier 2000)

$$W_2^2(p_0, p_1) = \min_{(p_t, \mathbf{u}_t)} \int_0^1 \int \|\mathbf{u}_t(\mathbf{x})\|^2 \, p_t(\mathbf{x}) \, d\mathbf{x} \, dt. \tag{38.100}$$

The squared Wasserstein distance is the least kinetic energy of any flow between p_0 and p_1. Without the condition in (ii) the equality still holds, with the minimum taken over weak solutions of the continuity equation with square-integrable velocity. We do not prove this general form, and our examples need it: the optimal field for the two points has slope $1/t$ and is not admissible.

Proof. The proof sandwiches the kinetic energy between W_2^2 from below and from above. Step II is the lower bound (i), and step III gives (ii).

I. *Every flow defines a coupling.* Let $\mathbf{X}_t$ solve $\dot{\mathbf{X}}_t = \mathbf{u}_t(\mathbf{X}_t)$ backward in time from $\mathbf{X}_1 \sim p_1$. By the uniqueness fact used in the reflow proposition, $\mathbf{X}_t \sim p_t$ for every t, so $(\mathbf{X}_0, \mathbf{X}_1)$ is a coupling of p_0 and p_1.

II. *Lower bound.* The cost of this coupling is at least W_2^2, and Jensen's inequality for the time average bounds it by the kinetic energy:

$$W_2^2(p_0,p_1) \le \mathbb{E}\,\|\mathbf{X}_1-\mathbf{X}_0\|^2 = \mathbb{E}\left\|\int_0^1 \mathbf{u}_t(\mathbf{X}_t)\,dt\right\|^2 \le \int_0^1 \mathbb{E}\,\|\mathbf{u}_t(\mathbf{X}_t)\|^2\,dt = \int_0^1\int \|\mathbf{u}_t\|^2\,p_t\,d\mathbf{x}\,dt.$$

III. *Upper bound.* Take an optimal coupling $\pi^\star$ and its straight interpolation $\mathbf{X}_t = (1-t)\mathbf{X}_0 + t\mathbf{X}_1$, with marginal velocity $\mathbf{u}_t(\mathbf{x}) = \mathbb{E}[\mathbf{X}_1-\mathbf{X}_0 \mid \mathbf{X}_t=\mathbf{x}]$. By the marginal-velocity proposition in its weak form this pair satisfies the continuity equation with the right endpoints, and by the conditional Jensen inequality its kinetic energy is

$$\int_0^1 \mathbb{E}\big\|\mathbb{E}[\mathbf{X}_1-\mathbf{X}_0 \mid \mathbf{X}_t]\big\|^2\,dt \le \mathbb{E}\|\mathbf{X}_1-\mathbf{X}_0\|^2 = W_2^2(p_0,p_1).$$

The two bounds meet. ■

When the field that flow matching would learn on an optimal coupling is admissible, it is the minimizer. Since the lower bound also applies to it, the inequality in III is an equality, so the conditional variance is zero: the segments of an optimal coupling do not cross. Jensen's inequality in II is tight only if each particle moves at constant velocity, so every minimizer moves its particles along straight lines at constant speed. A straight constant-speed flow attains W_2^2 only if, in addition, its pairing is optimal, which the quarter-turn pairing is not (9).

38.6.3.3 Approximate Optimal Couplings

The theorem names the target, an optimal coupling, and practice has two approximations to it. Reflow re-pairs by the model. A *minibatch optimal-transport* pairing (Tong et al. 2024; Pooladian et al. 2023) needs no trained model: it re-pairs each batch of 256 noise and data points by the assignment of least total squared distance before the training step. 7 trains the two moons with this pairing. The loss plateau falls from 1.4 to about 0.06 per coordinate, and one Euler step already scores a few thousandths. Neither method finds the optimal coupling of the marginals: a batch plan depends on the batch, and reflow inherits the errors of the first model.

38.6.4 Sampling with a Learned Field

A learned velocity is integrated as it is, and a learned score through the probability-flow ODE or the reverse SDE Equation 38.79. On a variance-preserving path the velocity and the probability-flow ODE are the same field, by the score–velocity identity. In every case a sample costs one field evaluation per solver stage, and the question is how to spend a budget of them. Section 17.4 answers it for image models. We keep one experiment, on a field with no learning error.

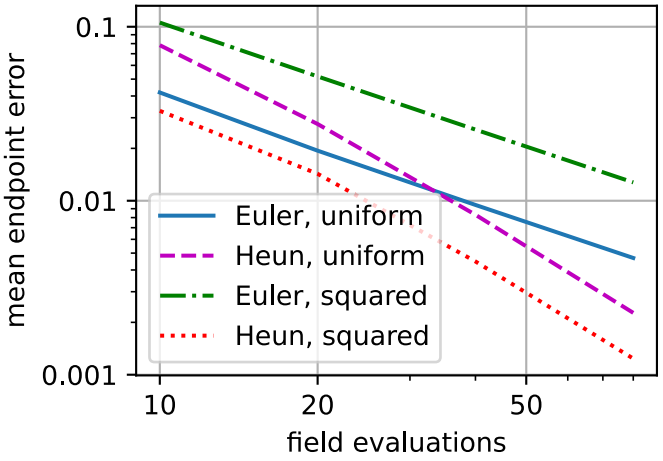
38.6.4.1 Solver Order at Equal Cost

Heun's method is second order but evaluates the field twice per step. Is it worth two evaluations? We take the probability-flow ODE with the *exact* score of the running mixture,

components at ± 2 with variance 0.25, under the variance-preserving schedule with $\beta(t)$ rising linearly from 0.1 to 20. The cell integrates from $t = 1$ to $t = 0$ with Euler and with Heun's method (Section 38.1.3), and compares the endpoints of $8{,}000$ starting points with a Heun solution of 800 steps. Both methods get the same budgets of 10 to 80 evaluations, spent on two grids of times: equal steps in t, and the *squared* grid $t_k = u_k^2$ with u_k equally spaced from 1 to 0, whose steps shrink toward the data end $t = 0$.

```
bmin, bmax = 0.1, 20.0
beta_fn = lambda t: bmin + t * (bmax - bmin)           # VP noise schedule
abar_fn = lambda t: np.exp(-(bmin * t + 0.5 * (bmax - bmin) * t**2))
def score_vp(q, t, means=(-2.0, 2.0), var=0.25):       # exact mixture score
    a = abar_fn(t)
    v = a * var + (1 - a)
    w = np.stack([np.exp(-(q - np.sqrt(a) * m)**2 / (2 * v)) for m in means])
    return (w * np.stack([(np.sqrt(a) * m - q) / v
                          for m in means])).sum(0) / w.sum(0)
def pf_ode(q, t):                                      # probability-flow ODE
    return -0.5 * beta_fn(t) * (q + score_vp(q, t))
def solve(q, ts, heun=False):                          # steps along grid ts
    for t1, t2 in zip(ts[:-1], ts[1:]):
        h, d1 = t2 - t1, pf_ode(q, t1)
        q = q + 0.5 * h * (d1 + pf_ode(q + h * d1, t2)) if heun else q + h * d1
    return q
grids = {'uniform': lambda K: np.linspace(1, 0, K + 1),     # equal steps in t
         'squared': lambda K: np.linspace(1, 0, K + 1)**2}  # dense near t = 0
z = np.random.default_rng(0).standard_normal(8000)
ref = solve(z, grids['uniform'](800), heun=True)       # fine reference
nfes = [10, 20, 40, 80]                                # Heun: n // 2 steps
errs = {f'{m}, {g}': [np.abs(solve(z, grid(n // k), k == 2) - ref).mean()
                      for n in nfes]
        for g, grid in grids.items() for m, k in (('Euler', 1), ('Heun', 2))}
print(f'{"NFE":16s}' + ''.join(f'{n:8d}' for n in nfes))
for name, e in errs.items():
    print(f'{name:16s}' + ''.join(f'{v:8.4f}' for v in e))
d2l.plot(nfes, list(errs.values()), 'field evaluations', 'mean endpoint error',
         legend=list(errs), xscale='log', yscale='log')
```

```
NFE                10      20      40      80
Euler, uniform   0.0419  0.0194  0.0095  0.0047
Heun, uniform    0.0784  0.0276  0.0083  0.0023
Euler, squared   0.1057  0.0518  0.0256  0.0127
Heun, squared    0.0329  0.0143  0.0045  0.0012
```



On the uniform grid the orders show in the rates. Each doubling of the budget divides Euler's error by about two and Heun's by a factor that grows toward four (2.8, 3.3, 3.6). At equal cost, though, Heun loses at 10 evaluations (about 0.08 against Euler's 0.04) and at 20 (0.028 against 0.02), and leads only from 40. On the squared grid Heun wins at every budget, and at 10 evaluations its error, about 0.03, is below Euler's on either grid. Euler moves the other way: the squared grid more than doubles its error. On this field, where the steps go matters more than the order of the solver.

Two properties of the field account for the table. By $t = 0.5$ the paths have covered less than a tenth of their length, because for most of that time the noised law is close to $\mathcal{N}(0, 1)$, whose score cancels the drift. Equal steps in t spend half the budget there. And the endpoint error is a sum of signed step errors, so a grid changes how they cancel as well as how large they are. On equal steps Euler falls short while the paths speed up and overshoots while they slow down, and the two cancel to about an eighth of their total size, whereas more than half of Heun's step errors survive. On the squared grid about a third of each survives.

Diffusion samplers crowd their steps toward the data in the same way, and choose how much by measurement (Karras et al. 2022). The grid and the solver are chosen together, and the best grid depends on the schedule and on the data (Section 17.4).

38.6.4.2 Comparison of Model Families

The methods of this section and the previous one differ in the path they fix, the quantity they regress on, and the solver they sample with; Table 17.1 sets them side by side. What they share is the regression. Each fits a conditional mean of a per-sample quantity that is known in closed form, and samples with a solver from this chapter.

What remains open is the pairing of noise with data. Independent pairs make the flow curve, and reflow and batch re-pairing straighten it on the two moons. No known method finds the optimal pairing cheaply in high dimension, and the number of steps a learned field needs is still found by trial. Chapter 17 takes up both questions for images.

38.6.5 Summary

A probability path from noise to data can be chosen freely. The velocity that realizes it is the posterior mean of the velocities of per-sample paths, so conditional flow matching trains it

by least squares, with the same gradients as the intractable marginal loss. On a Gaussian path the velocity and the score are affine in one posterior mean, so one network can be read as either. On straight conditional paths the marginal flow bends wherever segments cross. Reflow re-pairs noise with the model's own endpoints, does not increase the transport cost, and straightens the flow: one Euler step of the reflowed two-moons model matches thirty-two steps of the original. The squared Wasserstein distance is the least kinetic energy of any flow between the endpoints, attained by straight constant-speed motion under an optimal pairing; a pairing can be straight and crossing-free without being optimal. Sampling is numerical integration, and the time grid matters as much as the solver: on the exact field of the running mixture Heun's method loses to Euler below about 40 evaluations on a uniform grid, and wins at every budget on a grid crowded toward the data.

38.6.6 Exercises

1. **Crossing segments and the floor.** The constant C of the CFM theorem is the conditional variance $\mathbb{E}\|Y - \mathbf{u}_t(\mathbf{x})\|^2$ of the target $Y = \mathbf{u}_t(\mathbf{x} \mid \mathbf{z})$.
   1. Show that $C = 0$ if and only if, for almost every t, the conditional velocity is almost surely a function of $(\mathbf{x}_t, t)$. What does this require of the segments of the straight path Equation 38.95?
   2. For the independent pairing of $\mathcal{N}(\mathbf{0}, \mathbf{I})$ with the standardized `moons`, compute the conditional variance of $\epsilon - \mathbf{x}_0$ given $\mathbf{x}_t$, summed over coordinates, at $t = 0$ and at $t = 1$. The training cell prints the loss per coordinate; which number should it be compared with?
   3. Compute the floor numerically. Draw $(t, \mathbf{x}_0, \epsilon)$, form $\mathbf{x}_t$, weight every training point $\mathbf{x}_0'$ by $\exp\big(-\|\mathbf{x}_t - (1-t)\mathbf{x}_0'\|^2/2t^2\big)$, and average the posterior variance of $\mathbf{x}_0$, divided by t^2, over a few thousand draws. Compare the result with the printed plateau. At which t is the per-t floor largest, and why does it fall toward both ends?
   4. The reflow loss settles at a few thousandths. What does this say about crossings among the segments from $\hat{\mathbf{x}}_0(\epsilon)$ to ϵ, allowing for the approximation error of the network?
2. **Stochastic interpolants.** ⊠ Albergo et al. (2023) write the paths of this section and of the previous one as $\mathbf{x}_t = a_t\mathbf{x}_0 + b_t\mathbf{x}_1 + c_t\mathbf{w}$, where $(\mathbf{x}_0, \mathbf{x}_1)$ is drawn from a coupling of p_0 and p_1, $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ is independent of both, and the smooth schedules satisfy $a_0 = b_1 = 1$ and $a_1 = b_0 = c_0 = c_1 = 0$. With a Gaussian p_1 the diffusion and rectified-flow paths are the one-sided case $c \equiv 0$; the latent term $c_t\mathbf{w}$ serves two-sided paths between arbitrary p_0 and p_1.
   1. Take $\mathbf{z} = (\mathbf{x}_0, \mathbf{x}_1, \mathbf{w})$ in the marginal velocity Equation 38.90. Show that it is $\mathbb{E}[\dot{a}_t\mathbf{x}_0 + \dot{b}_t\mathbf{x}_1 + \dot{c}_t\mathbf{w} \mid \mathbf{x}_t = \mathbf{x}]$, and write the corresponding CFM loss Equation 38.91.
   2. Give schedules that recover rectified flow Equation 38.95 and the cosine path, whose data and noise coefficients are $\cos\frac{\pi t}{2}$ and $\sin\frac{\pi t}{2}$, both with independent endpoints and $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$.
   3. For $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$, independent endpoints, and $c_t > 0$, show that the interpolant has the same marginal velocity as the interpolant with $c \equiv 0$ and $\sqrt{b_t^2 + c_t^2}$ in place of b_t.
   4. For $c_t > 0$, show that $\nabla \log p_t(\mathbf{x}) = -\mathbb{E}[\mathbf{w} \mid \mathbf{x}_t = \mathbf{x}]/c_t$. What does interior noise provide when p_1 is an arbitrary distribution rather than a Gaussian?
3. **Late decisions.** Take the two-point example with data $\pm a$.

1. Derive Equation 38.92 from the posterior over x_0, and show that the terms in x^2 cancel in the log-odds.
2. Show that $\partial_x u_t(0) = (1-(1-t)\,a^2/t^2)/t$ and that, as t falls, the origin turns from attracting to repelling at $t^* = \tfrac{1}{2}(\sqrt{a^4+4a^2}-a^2)$, which is $2\sqrt{2}-2 \approx 0.83$ for $a=2$. How does t^* behave as $a \to \infty$ and as $a \to 0$? Interpret both limits.
3. Show that away from the origin the slope $\partial_x u_t$ tends to $1/t$, and that an Euler step from $t=h$ down to $t=0$ lands exactly on $\pm a$ from any point where the tanh is saturated. Which part of the flow, then, limits a fixed-step solver, and at which times?
4. Show that $C = \mathbb{E}_t \mathbb{E}_x \mathrm{Var}(x_0 \mid x_t)/t^2$ is finite although the conditional velocity $(x-x_0)/t$ is unbounded. Use $\mathrm{Var}(x_0 \mid x_t = x) = a^2\,\mathrm{sech}^2\left((1-t)\,ax/t^2\right)$ and the fact that the density of x_t near 0 is of order $e^{-(1-t)^2a^2/2t^2}/t$.

4. code **Optimal transport in one dimension.**
   1. For data $\mathcal{N}(m, s^2)$ with $s > 0$ and noise $\mathcal{N}(0,1)$, show that the sorted pairing is $x_0 = m + s\epsilon$ and costs $m^2 + (s-1)^2$, while the independent pairing costs $m^2 + 1 + s^2$.
   2. Prove that in one dimension the sorted pairing minimizes $\mathbb{E}\,c(x_1 - x_0)$ for every convex c. Suppose two pairs (x_0, x_1) and (x_0', x_1') cross: $x_0 < x_0'$ but $x_1 > x_1'$. Show that $c(x_1' - x_0) + c(x_1 - x_0') \le c(x_1 - x_0) + c(x_1' - x_0')$. (The two new differences lie between the two old ones and have the same sum.) Conclude Equation 38.99.
   3. Show that the flow map from $t=1$ to $t=0$ of a one-dimensional flow with a Lipschitz field is increasing, and hence that a flow carrying p_1 to p_0, with p_1 a density, induces the sorted pairing. Conclude that in one dimension every such flow has an optimal pairing.
   4. Check (a) by sorting 10^5 samples for $(m, s) = (0, 2)$ and $(1, 2)$.
5. code **Kinetic energy of Gaussian paths.**
   1. Show that if $p_t = \mathcal{N}(0, s_t^2)$ in one dimension, the velocity $u_t(x) = (\dot{s}_t/s_t)\,x$ satisfies the continuity equation, and that its kinetic energy is $\int_0^1 \dot{s}_t^2\,dt$. Why does a Gaussian cloud need no other velocity?
   2. For data $\mathcal{N}(0, 4)$ and noise $\mathcal{N}(0,1)$, find s_t for (i) the straight flow of the sorted pairing, (ii) the marginal flow of straight segments under the independent pairing, and (iii) the cosine path. Compute the three kinetic energies in closed form or by quadrature, and order them against $W_2^2 = 1$.
   3. Explain why (ii) costs more than (iii) although its conditional segments are straight. Sketch s_t for both.
   4. Repeat (ii) for data $\mathcal{N}(0,1)$. Show that the independent pairing has kinetic energy about 0.43 although $W_2^2 = 0$. Which pairing costs nothing, and what is its flow?
6. code **One reflow round on two points.** Take data $\pm a$, noise $\mathcal{N}(0,1)$, the independent pairing, and the marginal flow Equation 38.92 of the two-point example.
   1. Using part (c) of 4, show that the flow map from $t=1$ to $t=0$ is $\epsilon \mapsto a\,\mathrm{sign}(\epsilon)$, so one reflow round produces the sorted pairing.
   2. Show that its cost is $a^2 - 2a\sqrt{2/\pi} + 1$, about 1.81 for $a=2$, against $a^2+1=5$ for the independent pairing, and explain why this cost is W_2^2.
   3. Show that the reflowed segments $(1-t)\,a\,\mathrm{sign}(\epsilon) + t\,\epsilon$ never cross, so that the floor of the second round is $C=0$, and write down its field. Is one Euler step of it exact?

4. Integrate the closed-form field with RK4 from $20{,}000$ Gaussian draws at $t = 1$ down to $t = 10^{-3}$ (the velocity grows like $1/t$, so stop short of 0), and check the endpoints and the cost.
5. The floor of the independent pairing is $C \approx 2.03$ for $a = 2$ (compute it). Check the bound $\mathbb{E}\|\mathbf{Z}_1 - \mathbf{Z}_0\|^2 \le \mathbb{E}\|\mathbf{X}_1 - \mathbf{X}_0\|^2 - C$ of the reflow proposition. How much slack does it leave here?

7. code **Minibatch optimal-transport pairing.** Replace `independent_pairs` by a sampler that draws 256 noise points and 256 data points and re-pairs them by the assignment of least total squared distance (`scipy.optimize.linear_sum_assignment` on the 256×256 cost matrix), and train with `train_rf`.
   1. Report the loss plateau, the energy distance at $1, 2, 8$ and 32 steps, and the mean squared pair length before and after re-pairing. Compare with the independent pairing and with reflow.
   2. Show that a re-paired noise point is still a standard Gaussian draw and a re-paired data point is still a draw from the data, so the marginal field still carries p_1 to p_0.
   3. Explain why the batch pairing is not an optimal coupling of the two distributions, and what happens as the batch size grows. Which of the two re-pairing methods needs a trained model, and which inherits its errors?
8. **Conditioning of prediction targets.** Take the Gaussian path Equation 38.93 with one-dimensional data $x_0 \sim \mathcal{N}(0, 1)$ and the cosine path.
   1. Compute, as functions of t, the variances of the sampled training targets in Table 38.2: the conditional score $-\epsilon/\sigma_t$, the noise ϵ, the clean point x_0, the velocity $\dot\alpha_t x_0 + \dot\sigma_t \epsilon$, and the v-target $\alpha_t \epsilon - \sigma_t x_0$. Which blow up, and at which end of the path?
   2. A network trained on the noise outputs $\hat\epsilon$ with error δ. Show that the implied clean point $(x - \sigma_t \hat\epsilon)/\alpha_t$ carries the error $\sigma_t \delta/\alpha_t$, and that the noise implied by a clean-point prediction with error δ carries $\alpha_t \delta/\sigma_t$. At which end of the path is each amplified?
   3. Show that on the cosine path the velocity target equals $\pi/2$ times the **v**-target, so that **v**-prediction is velocity prediction, and that the velocity implied by a noise prediction carries the error $\pi\delta/(2\alpha_t)$. Which parameterizations keep the error bounded at both ends?
9. **Reflow preserves marginals and can stall.**
   1. Prove part (i) of the reflow proposition in detail for the straight path, whose conditional laws are point masses. For a smooth test function φ with compact support, show that $\frac{d}{dt}\mathbb{E}\,\varphi(\mathbf{X}_t) = \mathbb{E}\big[\nabla\varphi(\mathbf{X}_t)\cdot \mathbf{u}_t(\mathbf{X}_t)\big]$, which is the continuity equation in weak form. Where is the tower property used, and where the Lipschitz hypothesis?
   2. Let π_k be the pairing after k rounds. Show that $\mathbb{E}_{\pi_k}\|\mathbf{x}_1 - \mathbf{x}_0\|^2$ is non-increasing and bounded below by W_2^2, hence convergent.
   3. Show that if a round changes nothing, $\pi_{k+1} = \pi_k$, then $C = 0$ for π_k. Identify the fixed point in one dimension.
   4. In two dimensions pair each $\mathbf{x}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ with $\mathbf{R}\mathbf{x}_0$, where $\mathbf{R}$ is the quarter-turn rotation. Show that the segments $\big((1-t)\mathbf{I} + t\mathbf{R}\big)\mathbf{x}_0$ never cross (the matrix is invertible for every t, because its eigenvalues are $(1-t) \pm \mathrm{i}\,t$), so that $C = 0$ and reflow changes nothing, while the cost is $\mathbb{E}\|\mathbf{R}\mathbf{x}_0 - \mathbf{x}_0\|^2 = 4$ against $W_2^2 = 0$. Which property beyond being straight and crossing-free does optimality need?
10. code **A stochastic sampler for a flow-matching model.** ☒
    1. On the straight path $\alpha_t = 1 - t$, $\sigma_t = t$, invert Equation 38.94 to get $\nabla \log p_t(\mathbf{x}) =$

$-(\mathbf{x} + (1-t)\,\mathbf{u}_t(\mathbf{x}))/t$.

2. Show that for every $g \geq 0$ the SDE $d\mathbf{X} = \big[\mathbf{u}_t(\mathbf{X}) - \frac{1}{2}g^2 \nabla \log p_t(\mathbf{X})\big]dt + g\, d\bar{\mathbf{W}}$, integrated from $t = 1$ down to $t = 0$ with a Brownian motion $\bar{\mathbf{W}}$ that runs backward in time, has the marginals p_t. Substitute $\tau = 1 - t$, write the Fokker–Planck equation in τ, and use $\nabla \cdot (p \nabla \log p) = \Delta p$. Why does the score term enter with a minus sign here? A trained velocity field therefore defines a family of stochastic samplers.
3. Implement the sampler with Euler–Maruyama for the trained two-moons field, with $g \in \{0, \frac{1}{2}, 1\}$ and $K \in \{8, 32, 128\}$ steps. Stop at $t = 1/K$, because the score grows like $1/t$, and report the energy distance of each run. Which g needs the fewest steps for a given quality, and does noise help at $K = 8$?
4. Repeat for the reflowed field. Explain, from the straightness of its trajectories, why no one-step stochastic sample can match its one-step deterministic sample.

38.7 Further Reading

The list in the chapter introduction (Chapter 38) names books, courses and papers that teach again what the chapter taught. This section points past the chapter, and past Chapter 17 as well: where that chapter or its own further reading (Section 17.9) covers a topic, we give the pointer and no more. What remains is mathematics, and it is not of equal weight. We would read three entries first: exponential integrators, one reason a diffusion sampler gets by with few steps; Wasserstein gradient flows, the reason transport kept reappearing in the chapter; and Schrödinger bridges, the stochastic counterpart of an optimal pairing. The rest answer questions a reader may or may not have. Nothing else in the book depends on this section.

38.7.1 Solvers

38.7.1.1 Exponential Integrators

The field of a diffusion sampler is semilinear: a linear term known in closed form plus a learned nonlinear part. An explicit Runge–Kutta step must stay inside the stability region set by the linear part (Section 38.1.3.3), however smooth the learned part is. An exponential integrator solves the linear part exactly, through the variation-of-constants formula Equation 38.6, and approximates only the integral of the nonlinear remainder. Hochbruck and Ostermann (2010) survey the method. Its use on the probability-flow ODE, in DEIS (Zhang and Chen 2023) and DPM-Solver (Lu, Zhou, et al. 2022), is taken up in Section 17.4. For SDEs, the stochastic Taylor expansion behind Milstein's correction continues to higher orders, but the terms beyond Milstein's involve iterated stochastic integrals that are expensive to sample (Kloeden and Platen 1992).

38.7.1.2 Geometric Integration and Leapfrog

On a frictionless oscillator forward Euler gains energy at every step and backward Euler loses it, so neither can be run for long. A *symplectic* method, such as symplectic Euler or leapfrog, does not try to follow the flow more closely. It preserves phase-space volume instead, and backward error analysis then shows that it conserves a nearby modified energy up to an exponentially small error, so its energy error stays bounded over exponentially long times rather than growing with the horizon (6 works the smallest case). Hairer et al.

(2006) develop this theory, including the modified equations whose first correction for gradient descent is the implicit gradient regularizer of Barrett and Dherin (2021) . Leapfrog is the integrator inside Hamiltonian Monte Carlo (Duane et al. 1987; Neal 2011) because a reversible, volume-preserving proposal keeps the Metropolis acceptance rule simple.

38.7.2 Continuous-Time Models

38.7.2.1 Flows with Cheap Determinants

A discrete normalizing flow needs the log-determinant of every layer's Jacobian, which costs $O(d^3)$ in general, and Section 38.2.3 replaced it by an integrated trace. Discrete flows instead restrict the layer. Coupling layers (Dinh, Pascanu, et al. 2017) and autoregressive flows make the Jacobian triangular, so its determinant is the product of the diagonal. Invertible residual networks (Behrmann et al. 2019) keep the Lipschitz constant of each block below one and estimate the log-determinant by the power series of 5. Papamakarios et al. (2021) survey the families and what each gives up.

38.7.2.2 Reversible Solvers and Regularized Flows

The adjoint method of Section 38.2.2 rebuilds the forward states by integrating backward, and a strongly contracting flow amplifies the error of that reconstruction. A reversible solver recovers its own forward steps from the endpoint up to rounding, so the backward pass carries no drift; Kidger (2022) (ch. 5) compares it with checkpointing.

A flow trained by likelihood also has no reason to be cheap to integrate. Penalizing the kinetic energy $\int \|\mathbf{f}_\theta\|^2 \, dt$ during training (Finlay et al. 2020) pushes it toward straight paths, the goal that flow matching reaches by choosing its regression target (Section 38.6.3).

38.7.2.3 Latent and Controlled Differential Equations

The chapter's models map an initial state to a final one, but time series arrive at irregular times and must be read as they come. A latent ODE (Rubanova et al. 2019) encodes the observations into an initial state and decodes the trajectory at any set of times. A neural controlled differential equation (Kidger et al. 2020) drives the state by the interpolated data path itself, $d\mathbf{z} = \mathbf{f}_\theta(\mathbf{z}) \, d\mathbf{X}_t$, which is the continuous-time counterpart of a recurrent network.

38.7.2.4 Transformers as Interacting Particles

Geshkovski et al. (2023) read a deep stack of self-attention layers as Euler steps of an interacting-particle ODE for the token representations on the sphere, and study how the tokens cluster as depth grows. The analysis uses the chapter's language of fields, flows and the continuity equation on a model that does not generate data.

38.7.3 Noise

38.7.3.1 Wasserstein Gradient Flows

The Fokker–Planck equation has a variational reading. For a gradient drift $\mathbf{f} = -\nabla V$ and a constant g, it is the steepest descent of the free energy $\mathcal{F}(p) = \int V p \, d\mathbf{x} + \frac{g^2}{2} \int p \log p \, d\mathbf{x}$ in the geometry of the Wasserstein-2 distance. The JKO scheme (Jordan et al. 1998) discretizes

it as a sequence of proximal steps, $p_{k+1} = \arg\min_p \mathcal{F}(p) + \frac{1}{2h} W_2^2(p, p_k)$, the analogue for densities of the backward Euler step of Section 38.1.3.3. This is why transport, from the continuity equation to Benamou–Brenier (Benamou and Brenier 2000), keeps reappearing in the chapter.

38.7.3.2 Physical Noise and the Wong–Zakai Theorem

Section 38.3 defines the Itô integral with the left endpoint and shows that the midpoint rule, which defines the Stratonovich integral, keeps the ordinary chain rule. Which one describes a physical system driven by fast but smooth noise? If the Brownian path is replaced by smooth approximations and the resulting ODEs are solved classically, the solutions converge to the Stratonovich solution (Wong and Zakai 1965). In one dimension the two readings of $dX = f\,dt + g\,dW$ differ by the drift $\frac{1}{2} g\,\partial_x g$. They agree for the additive noise of diffusion models; for multiplicative noise the modeler has to choose.

38.7.3.3 Stochastic Gradient Descent as an SDE

Minibatch noise makes stochastic gradient descent a random recurrence. Reading a step with learning rate η as an Euler–Maruyama step of $d\theta = -\nabla \mathcal{L}\,dt + \sqrt{\eta}\,\Sigma^{1/2} d\mathbf{W}$, where Σ is the covariance of a minibatch gradient, turns the learning rate into a temperature (Q. Li et al. 2017). On a quadratic loss the iterates then settle into the stationary Gaussian of an OU process (Section 38.3.3), whose variance grows with the ratio of the learning rate to the batch size. Mandt et al. (2017) use this to run SGD as an approximate posterior sampler. The model needs small learning rates and nearly Gaussian gradient noise, and both assumptions can fail in practice.

38.7.3.4 Neural SDEs

When the learned model is itself an SDE, with a learned drift and a learned diffusion, the adjoint of Section 38.2.2 becomes a backward SDE driven by the same Brownian path, which must be stored or regenerated from its random seed (X. Li et al. 2020). The training signal is then a divergence between laws of whole paths, or a discriminator that compares them, rather than a loss on one endpoint (Kidger 2022). For two diffusions with the same noise, Girsanov's theorem gives that divergence as an integral of the squared difference of their drifts (Øksendal 2003).

38.7.4 Learned Generators

Most of what follows the chapter in this direction is taught in Chapter 17: prediction targets, loss weightings and schedules in Section 17.5, where the log-SNR coordinate of Kingma et al. (2021) orders the choices, and generation in one or a few steps in Section 17.7. Three topics lie beyond both chapters.

38.7.4.1 Schrödinger Bridges and Entropic Transport

The minibatch pairing of Section 38.6.3 is an exact assignment on each batch, and the generator it trains is an ODE. The Schrödinger bridge asks instead for the law of paths that is closest in relative entropy to a reference diffusion and has the two given marginals. Its pairing is the plan of entropy-regularized optimal transport, computed by the Sinkhorn iterations of Section 37.4.2.4 (Cuturi 2013; Peyré and Cuturi 2019) rather than an assignment,

and the generator is an SDE. De Bortoli et al. (2021) compute it by fitting forward and backward diffusions in alternation. Albergo et al. (2023) reach both kinds of generator from one construction: a path $a_t\mathbf{x}_0 + b_t\mathbf{x}_1 + c_t\mathbf{w}$ between any two distributions, with independent Gaussian $\mathbf{w}$ and c_t zero at both ends, whose marginals an ODE or a family of SDEs can generate (2 works through the construction).

38.7.4.2 Straight Couplings and Optimal Couplings

A pairing that reflow leaves unchanged has straight paths that do not cross, and in more than one dimension it need not be optimal: the quarter-turn pairing of Section 38.6.3 costs 4 where $W_2^2 = 0$. Liu (2022) restricts the velocity field to a gradient $\nabla\phi_t$ and argues that the pairings this restricted reflow leaves unchanged are the optimal ones.

38.7.4.3 Sliced Score Matching

Implicit score matching needs the divergence of the score network, the trace of a $d \times d$ Jacobian, which costs d backward passes. Sliced score matching (Song et al. 2019) replaces the trace by Hutchinson's estimate from Section 38.2.3, one vector–Jacobian product per random direction. It learns the score of the clean data without adding noise, at the price of the estimator's variance.

References

Albergo, Michael S., Nicholas M. Boffi, and Eric Vanden-Eijnden. 2023. "Stochastic Interpolants: A Unifying Framework for Flows and Diffusions." *arXiv Preprint arXiv:2303.08797*.

Alemi, Alexander A., Ian Fischer, Joshua V. Dillon, and Kevin Murphy. 2017. "Deep Variational Information Bottleneck." *International Conference on Learning Representations*.

Amari, Shun-ichi. 1998. "Natural Gradient Works Efficiently in Learning." *Neural Computation* 10 (2): 251–76.

Amari, Shun-ichi. 2016. *Information Geometry and Its Applications*. Springer.

Anderson, Brian D. O. 1982. "Reverse-Time Diffusion Equation Models." *Stochastic Processes and Their Applications* 12 (3): 313–26.

Angelopoulos, Anastasios N., and Stephen Bates. 2023. "Conformal Prediction: A Gentle Introduction." *Foundations and Trends in Machine Learning* 16 (4): 494–591. https://doi.org/10.1561/2200000101.

Arıkan, Erdal. 2009. "Channel Polarization: A Method for Constructing Capacity-Achieving Codes for Symmetric Binary-Input Memoryless Channels." *IEEE Transactions on Information Theory* 55 (7): 3051–73. https://doi.org/10.1109/TIT.2009.2021379.

Arjevani, Yossi, Yair Carmon, John C. Duchi, Dylan J. Foster, Nathan Srebro, and Blake Woodworth. 2023. "Lower Bounds for Non-Convex Stochastic Optimization." *Mathematical Programming* 199: 165–214.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou. 2017. "Wasserstein Generative Adversarial Networks." *Proceedings of the 34th International Conference on Machine Learning*, 214–23.

Armijo, Larry. 1966. "Minimization of Functions Having Lipschitz Continuous First Partial Derivatives." *Pacific Journal of Mathematics* 16 (1): 1–3.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. "Layer Normalization." *ArXiv:1607.06450*. https://arxiv.org/abs/1607.06450.

Bach, Francis, and Eric Moulines. 2013. "Non-Strongly-Convex Smooth Stochastic Approx-

imation with Convergence Rate $O(1/n)$." *Advances in Neural Information Processing Systems* 26.

Bai, Shaojie, J. Zico Kolter, and Vladlen Koltun. 2019. "Deep Equilibrium Models." *Advances in Neural Information Processing Systems* 32.

Baik, Jinho, and Jack W. Silverstein. 2006. "Eigenvalues of Large Sample Covariance Matrices of Spiked Population Models." *Journal of Multivariate Analysis* 97 (6): 1382–408. https://doi.org/10.1016/j.jmva.2005.08.003.

Baldi, Pierre, and Kurt Hornik. 1989. "Neural Networks and Principal Component Analysis: Learning from Examples Without Local Minima." *Neural Networks* 2 (1): 53–58.

Ballé, Johannes, Valero Laparra, and Eero P. Simoncelli. 2017. "End-to-End Optimized Image Compression." *International Conference on Learning Representations*.

Ballé, Johannes, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. "Variational Image Compression with a Scale Hyperprior." *International Conference on Learning Representations*.

Barber, David, and Felix Agakov. 2003. "The IM Algorithm: A Variational Approach to Information Maximization." *Advances in Neural Information Processing Systems* 16.

Barrett, David G. T., and Benoit Dherin. 2021. "Implicit Gradient Regularization." *International Conference on Learning Representations*.

Bartlett, Peter L., Philip M. Long, Gábor Lugosi, and Alexander Tsigler. 2020. "Benign Overfitting in Linear Regression." *Proceedings of the National Academy of Sciences* 117 (48): 30063–70.

Bartlett, Peter L., and Shahar Mendelson. 2002. "Rademacher and Gaussian Complexities: Risk Bounds and Structural Results." *Journal of Machine Learning Research* 3: 463–82.

Baur, Walter, and Volker Strassen. 1983. "The Complexity of Partial Derivatives." *Theoretical Computer Science* 22 (3): 317–30.

Baydin, Atılım Güneş, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. 2022. "Gradients Without Backpropagation." *arXiv Preprint arXiv:2202.08587*.

Beck, Amir, and Marc Teboulle. 2003. "Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization." *Operations Research Letters* 31 (3): 167–75. https://doi.org/10.1016/S0167-6377(02)00231-6.

Beck, Amir, and Marc Teboulle. 2009. "A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems." *SIAM Journal on Imaging Sciences* 2 (1): 183–202.

Behrmann, Jens, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. 2019. "Invertible Residual Networks." *International Conference on Machine Learning*, 573–82.

Belghazi, Mohamed Ishmael, Aristide Baratin, Sai Rajeshwar, et al. 2018. "Mutual Information Neural Estimation." *Proceedings of the 35th International Conference on Machine Learning*, 531–40.

Benamou, Jean-David, and Yann Brenier. 2000. "A Computational Fluid Mechanics Solution to the Monge–Kantorovich Mass Transfer Problem." *Numerische Mathematik* 84 (3): 375–93.

Benaych-Georges, Florent, and Raj Rao Nadakuditi. 2012. "The Singular Values and Vectors of Low Rank Perturbations of Large Rectangular Random Matrices." *Journal of Multivariate Analysis* 111: 120–35. https://doi.org/10.1016/j.jmva.2012.04.019.

Bengio, Yoshua, Nicholas Léonard, and Aaron Courville. 2013. "Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation." *arXiv Preprint arXiv:1308.3432*.

Bengio, Yoshua, Patrice Simard, and Paolo Frasconi. 1994. "Learning Long-Term Dependencies with Gradient Descent Is Difficult." *IEEE Transactions on Neural Networks* 5 (2): 157–66. https://doi.org/10.1109/72.279181.

Benjamini, Yoav, and Yosef Hochberg. 1995. "Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing." *Journal of the Royal Statistical Society: Series B (Methodological)* 57 (1): 289–300.

Bernstein, Jeremy, and Laker Newhouse. 2024. "Old Optimizer, New Norm: An Anatomy." *arXiv Preprint arXiv:2409.20325*.

Bertoin, David, Jérôme Bolte, Sébastien Gerchinovitz, and Edouard Pauwels. 2021. "Numerical Influence of ReLU'(0) on Backpropagation." *Advances in Neural Information Processing Systems* 34: 468–79. https://arxiv.org/abs/2106.12915.

Billingsley, Patrick. 1995. *Probability and Measure*. 3rd ed. Wiley.

Biroli, Giulio, Tony Bonnaire, Valentin de Bortoli, and Marc Mézard. 2024. "Dynamical Regimes of Diffusion Models." *Nature Communications* 15: 9957. https://arxiv.org/abs/2402.18491.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer. https://doi.org/10.1007/978-0-387-45528-0.

Black, Fischer, and Myron Scholes. 1973. "The Pricing of Options and Corporate Liabilities." *Journal of Political Economy* 81: 637–54. https://doi.org/10.1086/260062.

Blier, Léonard, and Yann Ollivier. 2018. "The Description Length of Deep Learning Models." *Advances in Neural Information Processing Systems* 31. https://arxiv.org/abs/1802.07044.

Blum, Avrim, and Moritz Hardt. 2015. "The Ladder: A Reliable Leaderboard for Machine Learning Competitions." *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of machine learning research, vol. 37: 1006–14.

Bogachev, Vladimir I., Nicolai V. Krylov, Michael Röckner, and Stanislav V. Shaposhnikov. 2015. *Fokker–Planck–Kolmogorov Equations*. Vol. 207. Mathematical Surveys and Monographs. American Mathematical Society.

Bolte, Jérôme, and Edouard Pauwels. 2021. "Conservative Set Valued Fields, Automatic Differentiation, Stochastic Gradient Methods and Deep Learning." *Mathematical Programming* 188 (1): 19–51.

Bonferroni, Carlo E. 1936. "Teoria Statistica Delle Classi e Calcolo Delle Probabilità." *Pubblicazioni Del R. Istituto Superiore Di Scienze Economiche e Commerciali Di Firenze* 8: 3–62.

Bonneel, Nicolas, Julien Rabin, Gabriel Peyré, and Hanspeter Pfister. 2015. "Sliced and Radon Wasserstein Barycenters of Measures." *Journal of Mathematical Imaging and Vision* 51 (1): 22–45. https://doi.org/10.1007/s10851-014-0506-3.

Bottou, Léon, Frank E. Curtis, and Jorge Nocedal. 2018. "Optimization Methods for Large-Scale Machine Learning." *SIAM Review* 60 (2): 223–311.

Boucheron, Stéphane, Olivier Bousquet, and Gábor Lugosi. 2005. "Theory of Classification: A Survey of Some Recent Advances." *ESAIM: Probability and Statistics* 9: 323–75. https://doi.org/10.1214/154957805100000046.

Boucheron, Stéphane, Gábor Lugosi, and Pascal Massart. 2013. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press.

Bousquet, Olivier, and André Elisseeff. 2002. "Stability and Generalization." *Journal of Machine Learning Research* 2: 499–526. https://www.jmlr.org/papers/v2/bousquet02a.html.

Boyd, Stephen, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. 2011. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers." *Foundations and Trends in Machine Learning* 3 (1): 1–122. https://doi.org/10.1561/2200000016.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/.

Bray, Alan J., and David S. Dean. 2007. "Statistics of Critical Points of Gaussian Fields on Large-Dimensional Spaces." *Physical Review Letters* 98 (15): 150201.

Brenier, Yann. 1991. "Polar Factorization and Monotone Rearrangement of Vector-Valued Functions." *Communications on Pure and Applied Mathematics* 44 (4): 375–417.

Bretagnolle, Jean, and Catherine Huber. 1978. "Estimation Des Densités: Risque Minimax." In *Séminaire de Probabilités XII*, vol. 649. Lecture Notes in Mathematics. Springer.

Brin, Sergey, and Lawrence Page. 1998. "The Anatomy of a Large-Scale Hypertextual Web Search Engine." *Computer Networks and ISDN Systems* 30 (1–7): 107–17.

Burda, Yuri, Roger Grosse, and Ruslan Salakhutdinov. 2016. "Importance Weighted Autoencoders." *International Conference on Learning Representations*. https://arxiv.org/abs/1509

.00519.

Candès, Emmanuel J., and Benjamin Recht. 2009. "Exact Matrix Completion via Convex Optimization." *Foundations of Computational Mathematics* 9 (6): 717–72.

Canonne, Clément L. 2022. "A Short Note on an Inequality Between KL and TV." *arXiv Preprint arXiv:2202.07198*. https://arxiv.org/abs/2202.07198.

Carson, Erin, and Nicholas J. Higham. 2018. "Accelerating the Solution of Linear Systems by Iterative Refinement in Three Precisions." *SIAM Journal on Scientific Computing* 40 (2): A817–47.

Čencov, N. N. 1982. *Statistical Decision Rules and Optimal Inference*. Vol. 53. Translations of Mathematical Monographs. American Mathematical Society.

Chebyshev, Pafnuty L. 1867. "Des Valeurs Moyennes." *Journal de Mathématiques Pures Et Appliquées* 12 (2): 177–84.

Chechik, Gal, Amir Globerson, Naftali Tishby, and Yair Weiss. 2005. "Information Bottleneck for Gaussian Variables." *Journal of Machine Learning Research* 6: 165–88.

Chen, Lei, and Joan Bruna. 2023. "Beyond the Edge of Stability via Two-Step Gradient Updates." *International Conference on Machine Learning*, 4330–91.

Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. 2018. "Neural Ordinary Differential Equations." *Advances in Neural Information Processing Systems* 31.

Chen, Tianqi, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. "Training Deep Nets with Sublinear Memory Cost." *arXiv Preprint arXiv:1604.06174*.

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. "A Simple Framework for Contrastive Learning of Visual Representations." *Proceedings of the 37th International Conference on Machine Learning*, 1597–607.

Cheng, Pengyu, Weituo Hao, Shuyang Dai, Jiachang Liu, Zhe Gan, and Lawrence Carin. 2020. "CLUB: A Contrastive Log-Ratio Upper Bound of Mutual Information." *Proceedings of the 37th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 119: 1779–88.

Church, Kenneth Ward, and Patrick Hanks. 1990. "Word Association Norms, Mutual Information, and Lexicography." *Computational Linguistics* 16 (1): 22–29.

Church, Kenneth W., and William A. Gale. 1995. "Poisson Mixtures." *Natural Language Engineering* 1 (2): 163–90. https://doi.org/10.1017/S1351324900000139.

Chwialkowski, Kacper, Heiko Strathmann, and Arthur Gretton. 2016. "A Kernel Test of Goodness of Fit." *Proceedings of the 33rd International Conference on Machine Learning*, 2606–15.

Coddington, Earl A., and Norman Levinson. 1955. *Theory of Ordinary Differential Equations*.

McGraw-Hill.

Cohen, Jeremy M., Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. 2021. "Gradient Descent on Neural Networks Typically Occurs at the Edge of Stability." *International Conference on Learning Representations*.

Cohen, Joel E., and Charles M. Newman. 1984. "The Stability of Large Random Matrices and Their Products." *The Annals of Probability* 12 (2): 283–310. https://doi.org/10.1214/aop/1176993291.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3): 273–97.

Cover, T, and JM Thomas. 1999. *Elements of Information Theory*. John Wiley & Sons. https://doi.org/10.1002/047174882X.

Cramér, H. 1946. *Mathematical Methods of Statistics.* Princeton University Press.

Csiszár, Imre. 1967. "Information-Type Measures of Difference of Probability Distributions and Indirect Observations." *Studia Scientiarum Mathematicarum Hungarica* 2: 299–318.

Csiszár, Imre. 2008. "Axiomatic Characterizations of Information Measures." *Entropy* 10 (3): 261–73. https://doi.org/10.3390/e10030261.

Cuturi, Marco. 2013. "Sinkhorn Distances: Lightspeed Computation of Optimal Transport." *Advances in Neural Information Processing Systems* 26.

Dahlquist, Germund G. 1963. "A Special Stability Problem for Linear Multistep Methods." *BIT Numerical Mathematics* 3: 27–43.

Damian, Alex, Eshaan Nichani, and Jason D. Lee. 2023. "Self-Stabilization: The Implicit Bias of Gradient Descent at the Edge of Stability." *International Conference on Learning Representations*.

Darmois, Georges. 1935. "Sur Les Lois de Probabilité à Estimation Exhaustive." *Comptes Rendus de l'Académie Des Sciences* 200: 1265–66.

Dasgupta, Sanjoy, and Anupam Gupta. 2003. "An Elementary Proof of a Theorem of Johnson and Lindenstrauss." *Random Structures & Algorithms* 22 (1): 60–65.

Daskalakis, Constantinos, Andrew Ilyas, Vasilis Syrgkanis, and Haoyang Zeng. 2018. "Training GANs with Optimism." *International Conference on Learning Representations*.

Dauphin, Yann N., Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. 2014. "Identifying and Attacking the Saddle Point Problem in High-Dimensional Non-Convex Optimization." *Advances in Neural Information Processing Systems* 27: 2933–41.

Daxberger, Erik, Agustinus Kristiadi, Alexander Immer, Runa Eschenhagen, Matthias Bauer, and Philipp Hennig. 2021. "Laplace Redux – Effortless Bayesian Deep Learning."

Advances in Neural Information Processing Systems 34. https://arxiv.org/abs/2106.14806.

De Bortoli, Valentin, James Thornton, Jeremy Heng, and Arnaud Doucet. 2021. "Diffusion Schrödinger Bridge with Applications to Score-Based Generative Modeling." *Advances in Neural Information Processing Systems* 34: 17695–709.

Dean, David S., and Satya N. Majumdar. 2006. "Large Deviations of Extreme Eigenvalues of Random Matrices." *Physical Review Letters* 97 (16): 160201.

DeepSeek-AI. 2024. "DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model." *arXiv Preprint arXiv:2405.04434*.

Defazio, Aaron, Francis Bach, and Simon Lacoste-Julien. 2014. "SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives." *Advances in Neural Information Processing Systems* 27.

Defazio, Aaron, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. 2023. "Optimal Linear Decay Learning Rate Schedules and Further Refinements." *arXiv Preprint arXiv:2310.07831*.

Delétang, Grégoire, Anian Ruoss, Paul-Ambroise Duquenne, et al. 2023. "Language Modeling Is Compression." *ArXiv:2309.10668*. https://arxiv.org/abs/2309.10668.

Dempster, Arthur P, Nan M Laird, and Donald B Rubin. 1977. "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society: Series B* 39 (1): 1–38.

Deshpande, Ishan, Ziyu Zhang, and Alexander G. Schwing. 2018. "Generative Modeling Using the Sliced Wasserstein Distance." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Dhariwal, Prafulla, and Alex Nichol. 2021. "Diffusion Models Beat GANs on Image Synthesis." *Advances in Neural Information Processing Systems* 34.

DiCiccio, Thomas J., and Bradley Efron. 1996. "Bootstrap Confidence Intervals." *Statistical Science* 11 (3): 189–228. https://doi.org/10.1214/ss/1032280214.

Dinh, Laurent, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. 2017. "Sharp Minima Can Generalize for Deep Nets." *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 70: 1019–28. https://proceedings.mlr.press/v70/dinh17b.html.

Dinh, Laurent, Jascha Sohl-Dickstein, and Samy Bengio. 2017. "Density Estimation Using Real NVP." *International Conference on Learning Representations*. https://openreview.net/forum?id=HkpbnH9lx.

Domingos, Pedro, and Michael Pazzani. 1997. "On the Optimality of the Simple Bayesian Classifier Under Zero-One Loss." *Machine Learning* 29: 103–30.

Donsker, Monroe D., and S. R. Srinivasa Varadhan. 1983. "Asymptotic Evaluation of Certain

Markov Process Expectations for Large Time. IV." *Communications on Pure and Applied Mathematics* 36 (2): 183–212.

Dormand, John R., and Peter J. Prince. 1980. "A Family of Embedded Runge–Kutta Formulae." *Journal of Computational and Applied Mathematics* 6 (1): 19–26.

Dowson, D. C., and B. V. Landau. 1982. "The Fréchet Distance Between Multivariate Normal Distributions." *Journal of Multivariate Analysis* 12 (3): 450–55.

Drori, Yoel, and Marc Teboulle. 2014. "Performance of First-Order Methods for Smooth Convex Minimization: A Novel Approach." *Mathematical Programming* 145 (1–2): 451–82. https://doi.org/10.1007/s10107-013-0653-0.

Du, Simon S., Wei Hu, and Jason D. Lee. 2018. "Algorithmic Regularization in Learning Deep Homogeneous Models: Layers Are Automatically Balanced." *Advances in Neural Information Processing Systems* 31. https://arxiv.org/abs/1806.00900.

Duane, Simon, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth. 1987. "Hybrid Monte Carlo." *Physics Letters B* 195 (2): 216–22.

Duchi, John, Elad Hazan, and Yoram Singer. 2011. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *Journal of Machine Learning Research* 12: 2121–59. https://jmlr.org/papers/v12/duchi11a.html.

Duchi, John, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. 2008. "Efficient Projections onto the ℓ_1-Ball for Learning in High Dimensions." *Proceedings of the 25th International Conference on Machine Learning (ICML)*, 272–79.

Duda, Jarek. 2013. *Asymmetric Numeral Systems: Entropy Coding Combining Speed of Huffman Coding with Compression Rate of Arithmetic Coding*. arXiv:1311.2540. https://arxiv.org/abs/1311.2540.

Dupont, Emilien, Arnaud Doucet, and Yee Whye Teh. 2019. "Augmented Neural ODEs." *Advances in Neural Information Processing Systems* 32.

Dwork, Cynthia, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth. 2015. "Preserving Statistical Validity in Adaptive Data Analysis." *ACM Symposium on Theory of Computing*, 117–26.

Dziugaite, Gintare Karolina, and Daniel M. Roy. 2017. "Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters Than Training Data." *Proceedings of the 33rd Conference on Uncertainty in Artificial Intelligence (UAI)*. https://arxiv.org/abs/1703.11008.

Eckart, Carl, and Gale Young. 1936. "The Approximation of One Matrix by Another of Lower Rank." *Psychometrika* 1 (3): 211–18.

Efron, Bradley. 1979. "Bootstrap Methods: Another Look at the Jackknife." *Annals of Statistics* 7 (1): 1–26.

Efron, Bradley. 2011. "Tweedie's Formula and Selection Bias." *Journal of the American Statistical Association* 106 (496): 1602–14.

Elhage, Nelson, Tristan Hume, Catherine Olsson, et al. 2022. "Toy Models of Superposition." *Transformer Circuits Thread*.

Endres, Dominik M., and Johannes E. Schindelin. 2003. "A New Metric for Probability Distributions." *IEEE Transactions on Information Theory* 49 (7): 1858–60.

Erven, Tim van, and Peter Harremoës. 2014. "Rényi Divergence and Kullback–Leibler Divergence." *IEEE Transactions on Information Theory* 60 (7): 3797–820.

Feydy, Jean, Thibault Séjourné, François-Xavier Vialard, Shun-ichi Amari, Alain Trouvé, and Gabriel Peyré. 2019. "Interpolating Between Optimal Transport and MMD Using Sinkhorn Divergences." *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2681–90.

Finlay, Chris, Jörn-Henrik Jacobsen, Levon Nurbekyan, and Adam M. Oberman. 2020. "How to Train Your Neural ODE: The World of Jacobian and Kinetic Regularization." *International Conference on Machine Learning*.

Fisher, R A. 1925. *Statistical Methods for Research Workers.* Oliver & Boyd.

Fisher, Ronald A. 1935. *The Design of Experiments*. Oliver; Boyd.

Fisher, Ronald A. 1936. "The Use of Multiple Measurements in Taxonomic Problems." *Annals of Eugenics* 7 (2): 179–88. https://doi.org/10.1111/j.1469-1809.1936.tb02137.x.

Fokker, Adriaan D. 1914. "Die Mittlere Energie Rotierender Elektrischer Dipole Im Strahlungsfeld." *Annalen Der Physik* 348 (5): 810–20.

Folland, Gerald B. 1999. *Real Analysis: Modern Techniques and Their Applications*. 2nd ed. Wiley.

Frank, Marguerite, and Philip Wolfe. 1956. "An Algorithm for Quadratic Programming." *Naval Research Logistics Quarterly* 3 (1–2): 95–110. https://doi.org/10.1002/nav.3800030109.

Frey, Brendan J., and Geoffrey E. Hinton. 1997. "Efficient Stochastic Source Coding and an Application to a Bayesian Network Source Model." *The Computer Journal* 40 (2/3): 157–65.

Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. 2010. "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software* 33 (1): 1–22.

Furstenberg, Harry, and Harry Kesten. 1960. "Products of Random Matrices." *The Annals of Mathematical Statistics* 31 (2): 457–69.

Gavish, Matan, and David L. Donoho. 2014. "The Optimal Hard Threshold for Singular Values Is $4/\sqrt{3}$." *IEEE Transactions on Information Theory* 60 (8): 5040–53.

Geman, Stuart. 1986. "The Spectral Radius of Large Random Matrices." *The Annals of Probability* 14 (4): 1318–28.

Geman, Stuart, Elie Bienenstock, and René Doursat. 1992. "Neural Networks and the Bias/Variance Dilemma." *Neural Computation* 4 (1): 1–58.

Geman, Stuart, and Donald Geman. 1984. "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6 (6): 721–41.

Genevay, Aude, Gabriel Peyré, and Marco Cuturi. 2018. "Learning Generative Models with Sinkhorn Divergences." *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*.

Geng, Zhengyang, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. 2021. "On Training Implicit Models." *Advances in Neural Information Processing Systems* 34. https://arxiv.org/abs/2111.05177.

Gershgorin, Semyon A. 1931. "Über Die Abgrenzung Der Eigenwerte Einer Matrix." *Izvestiya Akademii Nauk SSSR, Seriya Matematicheskaya*, no. 6: 749–54.

Geshkovski, Borjan, Cyril Letrouit, Yury Polyanskiy, and Philippe Rigollet. 2023. "A Mathematical Perspective on Transformers." *arXiv Preprint arXiv:2312.10794*.

Geweke, John. 1989. "Bayesian Inference in Econometric Models Using Monte Carlo Integration." *Econometrica* 57 (6): 1317–39. https://doi.org/10.2307/1913710.

Ghadimi, Saeed, and Guanghui Lan. 2013. "Stochastic First- and Zeroth-Order Methods for Nonconvex Stochastic Programming." *SIAM Journal on Optimization* 23 (4): 2341–68. https://arxiv.org/abs/1309.5549.

Gibbs, Josiah Willard. 1902. *Elementary Principles of Statistical Mechanics*. Scribner's.

Giles, Michael B. 2008. "Multilevel Monte Carlo Path Simulation." *Operations Research* 56 (3): 607–17.

Giles, Mike B. 2008. *An Extended Collection of Matrix Derivative Results for Forward and Reverse Mode Algorithmic Differentiation*. NA-08/01. Oxford University Computing Laboratory.

Ginibre, Jean. 1965. "Statistical Ensembles of Complex, Quaternion, and Real Matrices." *Journal of Mathematical Physics* 6 (3): 440–49. https://doi.org/10.1063/1.1704292.

Gnedenko, B. V., and A. N. Kolmogorov. 1954. *Limit Distributions for Sums of Independent Random Variables*. Addison-Wesley.

Gneiting, Tilmann, Fadoua Balabdaoui, and Adrian E. Raftery. 2007. "Probabilistic Forecasts, Calibration and Sharpness." *Journal of the Royal Statistical Society Series B: Statistical Methodology* 69 (2): 243–68. https://doi.org/10.1111/j.1467-9868.2007.00587.x.

Gneiting, Tilmann, and Adrian E. Raftery. 2007. "Strictly Proper Scoring Rules, Prediction,

and Estimation." *Journal of the American Statistical Association* 102 (477): 359–78. https://doi.org/10.1198/016214506000001437.

Goldberg, David. 1991. "What Every Computer Scientist Should Know about Floating-Point Arithmetic." *ACM Computing Surveys* 23 (1): 5–48.

Goldfeld, Ziv, Ewout van den Berg, Kristjan Greenewald, et al. 2019. "Estimating Information Flow in Deep Neural Networks." *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 97: 2299–308.

Golub, Gene H, and Charles F Van Loan. 1996. *Matrix Computations*. Johns Hopkins University Press. https://jhupbooks.press.jhu.edu/title/matrix-computations.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, et al. 2014. "Generative Adversarial Nets." *Advances in Neural Information Processing Systems*, 2672–80. https://doi.org/10.5555/2969033.2969125.

Goyal, Priya, Piotr Dollár, Ross Girshick, et al. 2017. "Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour." *ArXiv:1706.02677*. https://arxiv.org/abs/1706.02677.

Grathwohl, Will, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. 2019. "FFJORD: Free-Form Continuous Dynamics for Scalable Reversible Generative Models." *International Conference on Learning Representations*.

Gretton, Arthur, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. "A Kernel Two-Sample Test." *Journal of Machine Learning Research* 13: 723–73.

Griewank, Andreas. 1992. "Achieving Logarithmic Growth of Temporal and Spatial Complexity in Reverse Automatic Differentiation." *Optimization Methods and Software* 1 (1): 35–54.

Griewank, Andreas, and Andrea Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. 2nd ed. Society for Industrial; Applied Mathematics (SIAM). https://doi.org/10.1137/1.9780898717761.

Grobman, David M. 1959. "Homeomorphism of Systems of Differential Equations." *Doklady Akademii Nauk SSSR* 128: 880–81.

Grünwald, Peter D. 2007. *The Minimum Description Length Principle*. MIT Press. https://doi.org/10.7551/mitpress/4643.001.0001.

Gulrajani, Ishaan, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. 2017. "Improved Training of Wasserstein GANs." *Advances in Neural Information Processing Systems* 30.

Gumbel, Emil J. 1954. *Statistical Theory of Extreme Values and Some Practical Applications*. U.S. National Bureau of Standards.

Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. 2017. "On Calibration of Mod-

ern Neural Networks." *International Conference on Machine Learning*, 1321–30. https://arxiv.org/abs/1706.04599.

Gupta, Suyog, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. "Deep Learning with Limited Numerical Precision." *International Conference on Machine Learning*, 1737–46.

Gupta, Vineet, Tomer Koren, and Yoram Singer. 2018. "Shampoo: Preconditioned Stochastic Tensor Optimization." *Proceedings of the 35th International Conference on Machine Learning (ICML)*, 1842–50.

Hairer, Ernst, Christian Lubich, and Gerhard Wanner. 2006. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2nd ed. Vol. 31. Springer Series in Computational Mathematics. Springer.

Hairer, Ernst, Syvert P. Nørsett, and Gerhard Wanner. 1993. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd ed. Springer.

Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. 2011. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions." *SIAM Review* 53 (2): 217–88.

Hall, Peter. 1992. *The Bootstrap and Edgeworth Expansion*. Springer Series in Statistics. Springer. https://doi.org/10.1007/978-1-4612-4384-7.

Hanin, Boris, and Mihai Nica. 2020. "Products of Many Large Random Matrices and Gradients in Deep Neural Networks." *Communications in Mathematical Physics* 376 (1): 287–322.

Hardt, Moritz, Ben Recht, and Yoram Singer. 2016. "Train Faster, Generalize Better: Stability of Stochastic Gradient Descent." *Proceedings of the 33rd International Conference on Machine Learning*, Proceedings of machine learning research, vol. 48: 1225–34. https://proceedings.mlr.press/v48/hardt16.html.

Hartman, Philip. 1960. "A Lemma in the Theory of Structural Stability of Differential Equations." *Proceedings of the American Mathematical Society* 11 (4): 610–20.

Hastings, W. Keith. 1970. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications." *Biometrika* 57 (1): 97–109.

Haveliwala, Taher H., and Sepandar D. Kamvar. 2003. *The Second Eigenvalue of the Google Matrix*. Stanford University. http://ilpubs.stanford.edu:8090/582/.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." *Proceedings of the IEEE International Conference on Computer Vision*, 1026–34. https://doi.org/10.1109/iccv.2015.123.

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. "Deep Residual Learning for Image Recognition." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 770–78. https://doi.org/10.1109/cvpr.2016.90.

Held, Michael, Philip Wolfe, and Harlan P. Crowder. 1974. "Validation of Subgradient Optimization." *Mathematical Programming* 6: 62–88.

Hestenes, Magnus R., and Eduard Stiefel. 1952. "Methods of Conjugate Gradients for Solving Linear Systems." *Journal of Research of the National Bureau of Standards* 49 (6): 409–36.

Higham, Nicholas J. 2002. *Accuracy and Stability of Numerical Algorithms*. 2nd ed. SIAM.

Higham, Nicholas J., and Theo Mary. 2019. "A New Approach to Probabilistic Rounding Error Analysis." *SIAM Journal on Scientific Computing* 41 (5): A2815–35. https://doi.org/10.1137/18M1226312.

Hill, Bruce M. 1975. "A Simple General Approach to Inference about the Tail of a Distribution." *The Annals of Statistics* 3 (5): 1163–74. https://doi.org/10.1214/aos/1176343247.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. 2015. "Distilling the Knowledge in a Neural Network." *arXiv Preprint arXiv:1503.02531*.

Ho, Jonathan, Ajay Jain, and Pieter Abbeel. 2020. "Denoising Diffusion Probabilistic Models." *Advances in Neural Information Processing Systems* 33: 6840–51. https://arxiv.org/abs/2006.11239.

Ho, Jonathan, and Tim Salimans. 2022. "Classifier-Free Diffusion Guidance." *arXiv Preprint arXiv:2207.12598*.

Hochbruck, Marlis, and Alexander Ostermann. 2010. "Exponential Integrators." *Acta Numerica* 19: 209–86.

Hoeffding, Wassily. 1963. "Probability Inequalities for Sums of Bounded Random Variables." *Journal of the American Statistical Association* 58 (301): 13–30.

Hoffman, Matthew D., and Andrew Gelman. 2014. "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research* 15: 1593–623. https://jmlr.org/papers/v15/hoffman14a.html.

Hoffmann, Jordan, Sebastian Borgeaud, Arthur Mensch, et al. 2022. "Training Compute-Optimal Large Language Models." *Advances in Neural Information Processing Systems* 35. https://arxiv.org/abs/2203.15556.

Horn, Roger A., and Charles R. Johnson. 2012. *Matrix Analysis*. 2nd ed. Cambridge University Press.

Howard, Steven R., Aaditya Ramdas, Jon McAuliffe, and Jasjeet Sekhon. 2021. "Time-Uniform, Nonparametric, Nonasymptotic Confidence Sequences." *The Annals of Statistics* 49 (2): 1055–80. https://doi.org/10.1214/20-AOS1991.

Hu, Edward J., Yelong Shen, Phillip Wallis, et al. 2021. "LoRA: Low-Rank Adaptation of Large Language Models." *arXiv Preprint arXiv:2106.09685*.

Hu, Shengding, Yuge Tu, Xu Han, et al. 2024. "MiniCPM: Unveiling the Potential of Small

Language Models with Scalable Training Strategies." *ArXiv:2404.06395*. https://arxiv.org/abs/2404.06395.

Huffman, David A. 1952. "A Method for the Construction of Minimum-Redundancy Codes." *Proceedings of the IRE* 40 (9): 1098–101. https://doi.org/10.1109/JRPROC.1952.273898.

Hutchinson, Michael F. 1989. "A Stochastic Estimator of the Trace of the Influence Matrix for Laplacian Smoothing Splines." *Communications in Statistics—Simulation and Computation* 18 (3): 1059–76.

Hyvärinen, Aapo. 2005. "Estimation of Non-Normalized Statistical Models by Score Matching." *Journal of Machine Learning Research* 6: 695–709.

IEEE. 2019. *IEEE Standard for Floating-Point Arithmetic*. IEEE Std 754-2019.

Ioannidis, John P. A. 2005. "Why Most Published Research Findings Are False." *PLoS Medicine* 2 (8): e124. https://doi.org/10.1371/journal.pmed.0020124.

Itô, Kiyosi. 1944. "Stochastic Integral." *Proceedings of the Imperial Academy* 20 (8): 519–24.

Jaggi, Martin. 2013. "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization." *Proceedings of the 30th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 28: 427–35.

James, W., and Charles Stein. 1961. "Estimation with Quadratic Loss." *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability* 1: 361–79.

Jang, Eric, Shixiang Gu, and Ben Poole. 2017. "Categorical Reparameterization with Gumbel-Softmax." *International Conference on Learning Representations*.

Jaynes, Edwin T. 1957. "Information Theory and Statistical Mechanics." *Physical Review* 106 (4): 620–30. https://doi.org/10.1103/PhysRev.106.620.

Jaynes, Edwin T. 1968. "Prior Probabilities." *IEEE Transactions on Systems Science and Cybernetics* 4 (3): 227–41. https://doi.org/10.1109/TSSC.1968.300117.

Jelinek, Frederick, Robert L. Mercer, Lalit R. Bahl, and James K. Baker. 1977. "Perplexity—a Measure of the Difficulty of Speech Recognition Tasks." *Journal of the Acoustical Society of America* 62 (S1): S63.

Johari, Ramesh, Pete Koomen, Leonid Pekelis, and David Walsh. 2017. "Peeking at A/B Tests: Why It Matters, and What to Do about It." *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1517–25. https://doi.org/10.1145/3097983.3097992.

Johnson, Rie, and Tong Zhang. 2013. "Accelerating Stochastic Gradient Descent Using Predictive Variance Reduction." *Advances in Neural Information Processing Systems* 26: 315–23.

Johnson, William B., and Joram Lindenstrauss. 1984. "Extensions of Lipschitz Mappings

into a Hilbert Space." *Contemporary Mathematics* 26: 189–206.

Jordan, Keller, Yuchen Jin, Vlado Boza, et al. 2024. *Muon: An Optimizer for Hidden Layers in Neural Networks*. Https://kellerjordan.github.io/posts/muon/.

Jordan, Richard, David Kinderlehrer, and Felix Otto. 1998. "The Variational Formulation of the Fokker–Planck Equation." *SIAM Journal on Mathematical Analysis* 29 (1): 1–17.

Kahan, William. 1965. "Further Remarks on Reducing Truncation Errors." *Communications of the ACM* 8 (1): 40.

Kaplan, Jared, Sam McCandlish, Tom Henighan, et al. 2020. "Scaling Laws for Neural Language Models." *ArXiv:2001.08361*. https://arxiv.org/abs/2001.08361.

Karimi, Hamed, Julie Nutini, and Mark Schmidt. 2016. "Linear Convergence of Gradient and Proximal-Gradient Methods Under the Polyak–Łojasiewicz Condition." *Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, 795–811.

Karras, Tero, Miika Aittala, Timo Aila, and Samuli Laine. 2022. "Elucidating the Design Space of Diffusion-Based Generative Models." *Advances in Neural Information Processing Systems* 35.

Karush, William. 1939. "Minima of Functions of Several Variables with Inequalities as Side Constraints." Master's thesis, Department of Mathematics, University of Chicago.

Kidger, Patrick. 2022. "On Neural Differential Equations." *arXiv Preprint arXiv:2202.02435*.

Kidger, Patrick, James Morrill, James Foster, and Terry Lyons. 2020. "Neural Controlled Differential Equations for Irregular Time Series." *Advances in Neural Information Processing Systems* 33: 6696–707.

Kingma, Diederik P, and Jimmy Ba. 2015. "Adam: A Method for Stochastic Optimization." *International Conference on Learning Representations*. https://arxiv.org/abs/1412.6980.

Kingma, Diederik P., Tim Salimans, Ben Poole, and Jonathan Ho. 2021. "Variational Diffusion Models." *Advances in Neural Information Processing Systems* 34.

Kingma, Diederik P., and Max Welling. 2014. "Auto-Encoding Variational Bayes." *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1312.6114.

Kloeden, Peter E., and Eckhard Platen. 1992. *Numerical Solution of Stochastic Differential Equations*. Springer.

Kolda, Tamara G., and Brett W. Bader. 2009. "Tensor Decompositions and Applications." *SIAM Review* 51 (3): 455–500.

Koller, Daphne, and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press. https://doi.org/10.7551/mitpress/7432.001.0001.

Kolmogorov, Andrey N. 1931. “Über Die Analytischen Methoden in Der Wahrscheinlichkeitsrechnung.” *Mathematische Annalen* 104: 415–58.

Koopman, Bernard O. 1936. “On Distributions Admitting a Sufficient Statistic.” *Transactions of the American Mathematical Society* 39 (3): 399–409.

Kraskov, Alexander, Harald Stögbauer, and Peter Grassberger. 2004. “Estimating Mutual Information.” *Physical Review E* 69 (6): 066138.

Kuhn, Harold W., and Albert W. Tucker. 1951. “Nonlinear Programming.” *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, 481–92.

Kullback, Solomon, and Richard A. Leibler. 1951. “On Information and Sufficiency.” *Annals of Mathematical Statistics* 22 (1): 79–86.

Kynkäänniemi, Tuomas, Miika Aittala, Tero Karras, Samuli Laine, Timo Aila, and Jaakko Lehtinen. 2024. “Applying Guidance in a Limited Interval Improves Sample and Distribution Quality in Diffusion Models.” *Advances in Neural Information Processing Systems* 37. https://arxiv.org/abs/2404.07724.

Laplace, Pierre-Simon. 1814. *Essai Philosophique Sur Les Probabilités*. Courcier.

Läuchli, Peter. 1961. “Jordan-Elimination Und Ausgleichung Nach Kleinsten Quadraten.” *Numerische Mathematik* 3 (1): 226–40.

LeCun, Yann, Leon Bottou, G Orr, and Klaus-Robert Muller. 1998. “Efficient Backprop.” In *Neural Networks: Tricks of the Trade*. Springer. https://doi.org/10.1007/3-540-49430-8_2.

LeCun, Yann, Sumit Chopra, Raia Hadsell, Marc’Aurelio Ranzato, and Fu Jie Huang. 2006. “A Tutorial on Energy-Based Learning.” In *Predicting Structured Data*. MIT Press.

Lessard, Laurent, Benjamin Recht, and Andrew Packard. 2016. “Analysis and Design of Optimization Algorithms via Integral Quadratic Constraints.” *SIAM Journal on Optimization* 26 (1): 57–95.

Levin, David A., and Yuval Peres. 2017. *Markov Chains and Mixing Times*. 2nd ed. American Mathematical Society.

Levy, Omer, and Yoav Goldberg. 2014. “Neural Word Embedding as Implicit Matrix Factorization.” *Advances in Neural Information Processing Systems* 27.

Li, Ming, and Paul Vitányi. 2019. *An Introduction to Kolmogorov Complexity and Its Applications*. 4th ed. Texts in Computer Science. Springer. https://doi.org/10.1007/978-3-030-11298-1.

Li, Qianxiao, Cheng Tai, and Weinan E. 2017. “Stochastic Modified Equations and Adaptive Stochastic Gradient Algorithms.” *Proceedings of the 34th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 70: 2101–10.

Li, Xuechen, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. 2020. “Scal-

able Gradients for Stochastic Differential Equations." *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, Proceedings of machine learning research, vol. 108: 3870–82.

Li, Yujia, Kevin Swersky, and Richard Zemel. 2015. "Generative Moment Matching Networks." *Proceedings of the 32nd International Conference on Machine Learning*, 1718–27.

Lin, Jianhua. 1991. "Divergence Measures Based on the Shannon Entropy." *IEEE Transactions on Information Theory* 37 (1): 145–51.

Linnainmaa, Seppo. 1970. "The Representation of the Cumulative Rounding Error of an Algorithm as a Taylor Expansion of the Local Rounding Errors." Master's thesis, University of Helsinki.

Lipman, Yaron, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. 2023. "Flow Matching for Generative Modeling." *International Conference on Learning Representations*.

Liu, Chaoyue, Libin Zhu, and Mikhail Belkin. 2022. "Loss Landscapes and Optimization in over-Parameterized Non-Linear Systems and Neural Networks." *Applied and Computational Harmonic Analysis* 59: 85–116.

Liu, Dong C, and Jorge Nocedal. 1989. "On the Limited Memory BFGS Method for Large Scale Optimization." *Mathematical Programming* 45 (1): 503–28. https://doi.org/10.1007/bf01589116.

Liu, Qiang. 2022. "Rectified Flow: A Marginal Preserving Approach to Optimal Transport." *arXiv Preprint arXiv:2209.14577*.

Liu, Qiang, Jason D. Lee, and Michael I. Jordan. 2016. "A Kernelized Stein Discrepancy for Goodness-of-Fit Tests." *Proceedings of the 33rd International Conference on Machine Learning*, 276–84.

Liu, Qiang, and Dilin Wang. 2016. "Stein Variational Gradient Descent: A General Purpose Bayesian Inference Algorithm." *Advances in Neural Information Processing Systems* 29.

Liu, Shih-Yang, Chien-Yi Wang, Hongxu Yin, et al. 2024. "DoRA: Weight-Decomposed Low-Rank Adaptation." *International Conference on Machine Learning*.

Liu, Xingchao, Chengyue Gong, and Qiang Liu. 2023. "Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow." *International Conference on Learning Representations*.

Łojasiewicz, Stanisław. 1963. "Une Propriété Topologique Des Sous-Ensembles Analytiques réels." In *Les Équations Aux dérivées Partielles*. Éditions du CNRS.

Loshchilov, Ilya, and Frank Hutter. 2016. "SGDR: Stochastic Gradient Descent with Warm Restarts." *ArXiv:1608.03983*. https://arxiv.org/abs/1608.03983.

Loshchilov, Ilya, and Frank Hutter. 2019. "Decoupled Weight Decay Regularization." *Inter-*

national Conference on Learning Representations. https://arxiv.org/abs/1711.05101.

Lu, Cheng, Kaiwen Zheng, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022. "Maximum Likelihood Training for Score-Based Diffusion ODEs by High-Order Denoising Score Matching." *International Conference on Machine Learning*.

Lu, Cheng, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. 2022. "DPM-Solver: A Fast ODE Solver for Diffusion Probabilistic Model Sampling in Around 10 Steps." *Advances in Neural Information Processing Systems* 35.

Ma, Siyuan, Raef Bassily, and Mikhail Belkin. 2018. "The Power of Interpolation: Understanding the Effectiveness of SGD in Modern over-Parametrized Learning." *Proceedings of the 35th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 80: 3325–34.

MacKay, David J. C. 1992. "The Evidence Framework Applied to Classification Networks." *Neural Computation* 4 (5): 720–36. https://doi.org/10.1162/neco.1992.4.5.720.

Maddison, Chris J., Andriy Mnih, and Yee Whye Teh. 2017. "The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables." *International Conference on Learning Representations*.

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2018. "Towards Deep Learning Models Resistant to Adversarial Attacks." *International Conference on Learning Representations*. https://arxiv.org/abs/1706.06083.

Mahoney, Michael W., and Petros Drineas. 2009. "CUR Matrix Decompositions for Improved Data Analysis." *Proceedings of the National Academy of Sciences* 106 (3): 697–702.

Mandt, Stephan, Matthew D. Hoffman, and David M. Blei. 2017. "Stochastic Gradient Descent as Approximate Bayesian Inference." *Journal of Machine Learning Research* 18 (134): 1–35.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press. https://doi.org/10.1017/CBO9780511809071.

Marchenko, Vladimir A., and Leonid A. Pastur. 1967. "Distribution of Eigenvalues for Some Sets of Random Matrices." *Mathematics of the USSR-Sbornik* 1 (4): 457–83.

Maron, Melvin E. 1961. "Automatic Indexing: An Experimental Inquiry." *Journal of the ACM* 8 (3): 404–17.

Martens, James. 2010. "Deep Learning via Hessian-Free Optimization." *Proceedings of the 27th International Conference on Machine Learning*, 735–42.

Martens, James. 2020. "New Insights and Perspectives on the Natural Gradient Method." *Journal of Machine Learning Research* 21 (146).

Martens, James, and Roger Grosse. 2015. "Optimizing Neural Networks with Kronecker-

Factored Approximate Curvature." *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 2408–17.

Martin, Charles H., and Michael W. Mahoney. 2021. "Implicit Self-Regularization in Deep Neural Networks: Evidence from Random Matrix Theory and Implications for Learning." *Journal of Machine Learning Research* 22 (165): 1–73.

Martins, André F. T., and Ramón F. Astudillo. 2016. "From Softmax to Sparsemax: A Sparse Model of Attention and Multi-Label Classification." *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 1614–23.

Maruyama, Gisiro. 1955. "Continuous Markov Processes and Stochastic Equations." *Rendiconti Del Circolo Matematico Di Palermo* 4: 48–90.

Mazumder, Rahul, Trevor Hastie, and Robert Tibshirani. 2010. "Spectral Regularization Algorithms for Learning Large Incomplete Matrices." *Journal of Machine Learning Research* 11 (80): 2287–322.

McAllester, David A. 1999. "Some PAC-Bayesian Theorems." *Machine Learning* 37 (3): 355–63. https://doi.org/10.1023/A:1007618624809.

McAllester, David, and Karl Stratos. 2020. "Formal Limitations on the Measurement of Mutual Information." *International Conference on Artificial Intelligence and Statistics*, 875–84.

McCandlish, Sam, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. 2018. "An Empirical Model of Large-Batch Training." *arXiv Preprint arXiv:1812.06162*.

McDiarmid, Colin. 1989. "On the Method of Bounded Differences." In *Surveys in Combinatorics*. Cambridge University Press.

McFadden, Daniel. 1974. "Conditional Logit Analysis of Qualitative Choice Behavior." In *Frontiers in Econometrics*, edited by Paul Zarembka. Academic Press.

McMillan, Brockway. 1956. "Two Inequalities Implied by Unique Decipherability." *IRE Transactions on Information Theory* 2 (4): 115–16.

Meng, Fanxu, Zhaohui Wang, and Muhan Zhang. 2024. "PiSSA: Principal Singular Values and Singular Vectors Adaptation of Large Language Models." *Advances in Neural Information Processing Systems*.

Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. 1953. "Equation of State Calculations by Fast Computing Machines." *The Journal of Chemical Physics* 21 (6): 1087–92.

Metropolis, Nicholas, and Stanislaw Ulam. 1949. "The Monte Carlo Method." *Journal of the American Statistical Association* 44 (247): 335–41.

Miller, Evan. 2024. "Adding Error Bars to Evals: A Statistical Approach to Language Model Evaluations." *arXiv Preprint arXiv:2411.00640*.

Milstein, Grigori N. 1975. "Approximate Integration of Stochastic Differential Equations." *Theory of Probability and Its Applications* 19 (3): 557–62.

Mironov, Ilya. 2017. "Rényi Differential Privacy." *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 263–75.

Mirsky, Leon. 1960. "Symmetric Gauge Functions and Unitarily Invariant Norms." *The Quarterly Journal of Mathematics* 11 (1): 50–59.

Miyato, Takeru, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. "Spectral Normalization for Generative Adversarial Networks." *International Conference on Learning Representations*.

Mohamed, Shakir, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2020. "Monte Carlo Gradient Estimation in Machine Learning." *Journal of Machine Learning Research* 21 (132): 1–62.

Moler, Cleve B. 1967. "Iterative Refinement in Floating Point." *Journal of the ACM* 14 (2): 316–21.

Müller, Alfred. 1997. "Integral Probability Metrics and Their Generating Classes of Functions." *Advances in Applied Probability* 29 (2): 429–43.

Müller, Rafael, Simon Kornblith, and Geoffrey Hinton. 2019. "When Does Label Smoothing Help?" *Advances in Neural Information Processing Systems* 32. https://arxiv.org/abs/1906.02629.

Naumann, Uwe. 2008. "Optimal Jacobian Accumulation Is NP-Complete." *Mathematical Programming* 112 (2): 427–41.

Neal, Radford M. 2011. "MCMC Using Hamiltonian Dynamics." In *Handbook of Markov Chain Monte Carlo*. Chapman; Hall/CRC. https://doi.org/10.1201/b10905-6.

Nemirovski, Arkadi. 2004. "Prox-Method with Rate of Convergence $O(1/t)$ for Variational Inequalities with Lipschitz Continuous Monotone Operators and Smooth Convex-Concave Saddle Point Problems." *SIAM Journal on Optimization* 15 (1): 229–51. https://doi.org/10.1137/S1052623403425629.

Nemirovski, Arkadi, and David Yudin. 1983. *Problem Complexity and Method Efficiency in Optimization*. Wiley.

Nesterov, Yu. 2018. *Lectures on Convex Optimization*. Springer. https://doi.org/10.1007/978-3-319-91578-4.

Nesterov, Yurii. 1983. "A Method of Solving a Convex Programming Problem with Convergence Rate $O(1/k^2)$." *Soviet Mathematics Doklady* 27 (2): 372–76.

Nesterov, Yurii. 2005. "Smooth Minimization of Non-Smooth Functions." *Mathematical Programming* 103 (1): 127–52. https://doi.org/10.1007/s10107-004-0552-5.

Nesterov, Yurii. 2012. "Efficiency of Coordinate Descent Methods on Huge-Scale Optimization Problems." *SIAM Journal on Optimization* 22 (2): 341–62.

Neyman, Jerzy. 1937. "Outline of a Theory of Statistical Estimation Based on the Classical Theory of Probability." *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences* 236 (767): 333–80. https://doi.org/10.2307/jj.8501421.24.

Ng, Andrew Y., and Michael I. Jordan. 2002. "On Discriminative Vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes." *Advances in Neural Information Processing Systems* 14: 841–48.

Nguyen, XuanLong, Martin J. Wainwright, and Michael I. Jordan. 2010. "Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization." *IEEE Transactions on Information Theory* 56 (11): 5847–61.

Nigam, Kamal, Andrew Kachites McCallum, Sebastian Thrun, and Tom Mitchell. 2000. "Text Classification from Labeled and Unlabeled Documents Using EM." *Machine Learning* 39 (2–3): 103–34. https://doi.org/10.1023/A:1007692713085.

Nocedal, Jorge, and Stephen J. Wright. 2006. *Numerical Optimization*. 2nd ed. Springer.

Nowozin, Sebastian, Botond Cseke, and Ryota Tomioka. 2016. "F-GAN: Training Generative Neural Samplers Using Variational Divergence Minimization." *Advances in Neural Information Processing Systems* 29.

Øksendal, Bernt. 2003. *Stochastic Differential Equations: An Introduction with Applications*. 6th ed. Springer.

Oord, Aaron van den, Yazhe Li, and Oriol Vinyals. 2018. "Representation Learning with Contrastive Predictive Coding." *arXiv Preprint arXiv:1807.03748*.

Oseledets, Ivan V. 2011. "Tensor-Train Decomposition." *SIAM Journal on Scientific Computing* 33 (5): 2295–317.

Owen, Art B. 2013. *Monte Carlo Theory, Methods and Examples*.

Paley, Raymond E. A. C., Norbert Wiener, and Antoni Zygmund. 1933. "Notes on Random Functions." *Mathematische Zeitschrift* 37: 647–68.

Papamakarios, George, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. 2021. "Normalizing Flows for Probabilistic Modeling and Inference." *Journal of Machine Learning Research* 22 (57): 1–64.

Parikh, Neal, and Stephen Boyd. 2014. "Proximal Algorithms." *Foundations and Trends in Optimization* 1 (3): 127–239. https://doi.org/10.1561/2400000003.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio. 2013. "On the Difficulty of Training Recurrent Neural Networks." *International Conference on Machine Learning*, 1310–18.

Pearlmutter, Barak A. 1994. "Fast Exact Multiplication by the Hessian." *Neural Computation* 6 (1): 147–60. https://doi.org/10.1162/neco.1994.6.1.147.

Pennington, Jeffrey, Samuel Schoenholz, and Surya Ganguli. 2017. "Resurrecting the Sigmoid in Deep Learning Through Dynamical Isometry: Theory and Practice." *Advances in Neural Information Processing Systems*, 4785–95. https://proceedings.neurips.cc/paper/2017/hash/a9fc2d3b0c721b5b3f0b3f11bb24a75c-Abstract.html.

Peyré, Gabriel, and Marco Cuturi. 2019. "Computational Optimal Transport." *Foundations and Trends in Machine Learning* 11 (5–6): 355–607.

Pinsker, Mark S. 1964. *Information and Information Stability of Random Variables and Processes*. Holden-Day.

Pitman, Edwin J. G. 1936. "Sufficient Statistics and Intrinsic Accuracy." *Mathematical Proceedings of the Cambridge Philosophical Society* 32 (4): 567–79.

Planck, Max. 1917. "Über Einen Satz Der Statistischen Dynamik Und Seine Erweiterung in Der Quantentheorie." *Sitzungsberichte Der Preussischen Akademie Der Wissenschaften*, 324–41.

Platt, John C. 1999. "Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods." In *Advances in Large Margin Classifiers*, edited by Alexander J. Smola, Peter Bartlett, Bernhard Schölkopf, and Dale Schuurmans. MIT Press.

Polyak, Boris T. 1964. "Some Methods of Speeding up the Convergence of Iteration Methods." *USSR Computational Mathematics and Mathematical Physics* 4 (5): 1–17. https://doi.org/10.1016/0041-5553(64)90137-5.

Polyak, Boris T. 1963. "Gradient Methods for the Minimisation of Functionals." *USSR Computational Mathematics and Mathematical Physics* 3 (4): 864–78.

Polyak, Boris T., and Anatoli B. Juditsky. 1992. "Acceleration of Stochastic Approximation by Averaging." *SIAM Journal on Control and Optimization* 30 (4): 838–55.

Pontryagin, Lev S., Vladimir G. Boltyanskii, Revaz V. Gamkrelidze, and Evgenii F. Mishchenko. 1962. *The Mathematical Theory of Optimal Processes*. Interscience.

Pooladian, Aram-Alexandre, Heli Ben-Hamu, Carles Domingo-Enrich, Brandon Amos, Yaron Lipman, and Ricky T. Q. Chen. 2023. "Multisample Flow Matching: Straightening Flows with Minibatch Couplings." *Proceedings of the 40th International Conference on Machine Learning*.

Poole, Ben, Sherjil Ozair, Aaron van den Oord, Alexander A. Alemi, and George Tucker. 2019. "On Variational Bounds of Mutual Information." *Proceedings of the 36th International Conference on Machine Learning*, 5171–80.

Rademacher, Hans. 1919. "Über Partielle Und Totale Differenzierbarkeit von Funktionen Mehrerer Variabeln Und über Die Transformation Der Doppelintegrale." *Mathematische*

Annalen 79 (4): 340–59.

Radford, Alec, Jong Wook Kim, Chris Hallacy, et al. 2021. "Learning Transferable Visual Models from Natural Language Supervision." *International Conference on Machine Learning*, 8748–63. https://proceedings.mlr.press/v139/radford21a.html.

Radhakrishna Rao, C. 1945. "Information and Accuracy Attainable in the Estimation of Statistical Parameters." *Bulletin of the Calcutta Mathematical Society* 37 (3): 81–91. https://doi.org/10.1007/BF02908227.

Rahimi, Ali, and Benjamin Recht. 2007. "Random Features for Large-Scale Kernel Machines." *Advances in Neural Information Processing Systems* 20.

Reddi, Sashank J, Satyen Kale, and Sanjiv Kumar. 2018. "On the Convergence of Adam and Beyond." *International Conference on Learning Representations*. https://arxiv.org/abs/1904.09237.

Rényi, Alfréd. 1961. "On Measures of Entropy and Information." *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, 547–61.

Rezende, Danilo, and Shakir Mohamed. 2015. "Variational Inference with Normalizing Flows." *International Conference on Machine Learning*, 1530–38. https://proceedings.mlr.press/v37/rezende15.html.

Richardson, Tom, and Rüdiger Urbanke. 2008. *Modern Coding Theory*. Cambridge University Press. https://doi.org/10.1017/CBO9780511791338.

Risken, Hannes. 1996. *The Fokker–Planck Equation: Methods of Solution and Applications*. 2nd ed. Springer.

Rissanen, Jorma J. 1976. "Generalized Kraft Inequality and Arithmetic Coding." *IBM Journal of Research and Development* 20 (3): 198–203.

Robbins, Herbert, and Sutton Monro. 1951. "A Stochastic Approximation Method." *The Annals of Mathematical Statistics* 22 (3): 400–407.

Roberts, Gareth O., Andrew Gelman, and Walter R. Gilks. 1997. "Weak Convergence and Optimal Scaling of Random Walk Metropolis Algorithms." *The Annals of Applied Probability* 7 (1): 110–20. https://doi.org/10.1214/aoap/1034625254.

Roberts, Gareth O., and Richard L. Tweedie. 1996. "Exponential Convergence of Langevin Distributions and Their Discrete Approximations." *Bernoulli* 2 (4): 341–63.

Rockafellar, R. T. 1970. *Convex Analysis*. Princeton University Press. https://doi.org/10.1515/9781400873173.

Rombach, Robin, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2022. "High-Resolution Image Synthesis with Latent Diffusion Models." *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10684–95.

Rubanova, Yulia, Ricky T. Q. Chen, and David Duvenaud. 2019. "Latent Ordinary Differential Equations for Irregularly-Sampled Time Series." *Advances in Neural Information Processing Systems* 32.

Rudin, Walter. 1976. *Principles of Mathematical Analysis*. 3rd ed. McGraw-Hill.

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams. 1986. "Learning Representations by Back-Propagating Errors." *Nature* 323 (6088): 533–36. https://doi.org/10.1038/323533a0.

Sahami, Mehran, Susan Dumais, David Heckerman, and Eric Horvitz. 1998. "A Bayesian Approach to Filtering Junk e-Mail." *AAAI Workshop on Learning for Text Categorization*.

Särkkä, Simo, and Arno Solin. 2019. *Applied Stochastic Differential Equations*. Cambridge University Press.

Saxe, Andrew M., Yamini Bansal, Joel Dapello, et al. 2018. "On the Information Bottleneck Theory of Deep Learning." *International Conference on Learning Representations*.

Saxe, Andrew M., James L. McClelland, and Surya Ganguli. 2014. "Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks." *International Conference on Learning Representations*. https://arxiv.org/abs/1312.6120.

Schölkopf, Bernhard, and Alexander J Smola. 2002. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press. https://doi.org/10.7551/mitpress/4175.001.0001.

Seitzer, Maximilian, Arash Tavakoli, Dimitrije Antic, and Georg Martius. 2022. "On the Pitfalls of Heteroscedastic Uncertainty Estimation with Probabilistic Neural Networks." *International Conference on Learning Representations*. https://arxiv.org/abs/2203.09168.

Shannon, Claude E. 1959. "Coding Theorems for a Discrete Source with a Fidelity Criterion." *IRE National Convention Record* 7 (4): 142–63.

Shannon, Claude Elwood. 1948. "A Mathematical Theory of Communication." *The Bell System Technical Journal* 27 (3): 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Shannon, Claude Elwood. 1951. "Prediction and Entropy of Printed English." *The Bell System Technical Journal* 30 (1): 50–64. https://doi.org/10.1002/j.1538-7305.1951.tb01366.x.

Shwartz-Ziv, Ravid, and Naftali Tishby. 2017. "Opening the Black Box of Deep Neural Networks via Information." *arXiv Preprint arXiv:1703.00810*.

Simsekli, Umut, Levent Sagun, and Mert Gurbuzbalaban. 2019. "A Tail-Index Analysis of Stochastic Gradient Noise in Deep Neural Networks." *Proceedings of the 36th International Conference on Machine Learning*, Proceedings of machine learning research, vol. 97: 5827–37. https://proceedings.mlr.press/v97/simsekli19a.html.

Slater, Morton. 1950. *Lagrange Multipliers Revisited*. Discussion Paper Mathematics 403. Cowles Commission for Research in Economics.

Smith, Daniel G. A., and Johnnie Gray. 2018. "Opt_einsum: A Python Package for Optimizing Contraction Order for Einsum-Like Expressions." *Journal of Open Source Software* 3 (26): 753.

Sohl-Dickstein, Jascha, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. 2015. "Deep Unsupervised Learning Using Nonequilibrium Thermodynamics." *International Conference on Machine Learning*, 2256–65. https://proceedings.mlr.press/v37/sohl-dickstein15.html.

Song, Jiaming, and Stefano Ermon. 2020. "Understanding the Limitations of Variational Mutual Information Estimators." *International Conference on Learning Representations*.

Song, Jiaming, Chenlin Meng, and Stefano Ermon. 2021. "Denoising Diffusion Implicit Models." *International Conference on Learning Representations*.

Song, Yang, Conor Durkan, Iain Murray, and Stefano Ermon. 2021. "Maximum Likelihood Training of Score-Based Diffusion Models." *Advances in Neural Information Processing Systems* 34.

Song, Yang, and Stefano Ermon. 2019. "Generative Modeling by Estimating Gradients of the Data Distribution." *Advances in Neural Information Processing Systems* 32. https://arxiv.org/abs/1907.05600.

Song, Yang, Sahaj Garg, Jiaxin Shi, and Stefano Ermon. 2019. "Sliced Score Matching: A Scalable Approach to Density and Score Estimation." *Proceedings of the 35th Conference on Uncertainty in Artificial Intelligence*, Proceedings of machine learning research, vol. 115: 574–84. https://arxiv.org/abs/1905.07088.

Song, Yang, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. 2021. "Score-Based Generative Modeling Through Stochastic Differential Equations." *International Conference on Learning Representations*. https://arxiv.org/abs/2011.13456.

Soudry, Daniel, Elad Hoffer, Mor Shpigel Nacson, Suriya Gunasekar, and Nathan Srebro. 2018. "The Implicit Bias of Gradient Descent on Separable Data." *Journal of Machine Learning Research* 19 (70): 1–57.

Squire, William, and George Trapp. 1998. "Using Complex Variables to Estimate Derivatives of Real Functions." *SIAM Review* 40 (1): 110–12. https://doi.org/10.1137/S003614459631241X.

Stein, Charles. 1986. *Approximate Computation of Expectations*. Vol. 7. Institute of Mathematical Statistics Lecture Notes–Monograph Series. Institute of Mathematical Statistics.

Stein, Charles M. 1981. "Estimation of the Mean of a Multivariate Normal Distribution." *Annals of Statistics* 9 (6): 1135–51.

Sterbenz, Pat H. 1974. *Floating-Point Computation*. Prentice-Hall.

Stratonovich, Ruslan L. 1966. "A New Representation for Stochastic Integrals and Equa-

tions." *SIAM Journal on Control* 4 (2): 362–71.

Student (Gosset, William S.). 1908. "The Probable Error of a Mean." *Biometrika* 6 (1): 1–25.

Sutskever, Ilya, James Martens, George Dahl, and Geoffrey Hinton. 2013. "On the Importance of Initialization and Momentum in Deep Learning." *International Conference on Machine Learning*, 1139–47. https://proceedings.mlr.press/v28/sutskever13.html.

Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. "Rethinking the Inception Architecture for Computer Vision." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2818–26. https://doi.org/10.1109/cvpr.2016.308.

Székely, Gábor J., and Maria L. Rizzo. 2013. "Energy Statistics: A Class of Statistics Based on Distances." *Journal of Statistical Planning and Inference* 143 (8): 1249–72.

Tao, Terence, and Van Vu. 2010. "Random Matrices: Universality of ESDs and the Circular Law." *Annals of Probability* 38 (5): 2023–65.

Taylor, Adrien B., Julien M. Hendrickx, and François Glineur. 2018. "Exact Worst-Case Convergence Rates of the Proximal Gradient Method for Composite Convex Minimization." *Journal of Optimization Theory and Applications* 178 (2): 455–76.

Theis, Lucas, Aäron van den Oord, and Matthias Bethge. 2016. "A Note on the Evaluation of Generative Models." *International Conference on Learning Representations*.

Tibshirani, Robert. 1996. "Regression Shrinkage and Selection via the Lasso." *Journal of the Royal Statistical Society: Series B* 58 (1): 267–88.

Tieleman, Tijmen, and Geoffrey Hinton. 2012. "Divide the Gradient by a Running Average of Its Recent Magnitude." In *COURSERA: Neural Networks for Machine Learning, Lecture 6.5-Rmsprop*. https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

Tishby, Naftali, Fernando C. Pereira, and William Bialek. 1999. "The Information Bottleneck Method." *Proceedings of the 37th Annual Allerton Conference on Communication, Control, and Computing*, 368–77.

Tishby, Naftali, and Noga Zaslavsky. 2015. "Deep Learning and the Information Bottleneck Principle." *IEEE Information Theory Workshop (ITW)*, 1–5. https://doi.org/10.1109/ITW.2015.7133169.

Tong, Alexander, Kilian Fatras, Nikolay Malkin, et al. 2024. "Improving and Generalizing Flow-Based Generative Models with Minibatch Optimal Transport." *Transactions on Machine Learning Research*.

Townsend, James, Thomas Bird, and David Barber. 2019. "Practical Lossless Compression with Latent Variables Using Bits Back Coding." *International Conference on Learning Representations*. https://arxiv.org/abs/1901.04866.

Trefethen, Lloyd N., and David Bau. 1997. *Numerical Linear Algebra*. SIAM.

Trefethen, Lloyd N., and Mark Embree. 2005. *Spectra and Pseudospectra: The Behavior of Nonnormal Matrices and Operators*. Princeton University Press.

Tschannen, Michael, Josip Djolonga, Paul K. Rubenstein, Sylvain Gelly, and Mario Lucic. 2020. "On Mutual Information Maximization for Representation Learning." *International Conference on Learning Representations*.

Tseng, Paul. 2001. "Convergence of a Block Coordinate Descent Method for Nondifferentiable Minimization." *Journal of Optimization Theory and Applications* 109 (3): 475–94.

Tucker, George, Andriy Mnih, Chris J. Maddison, John Lawson, and Jascha Sohl-Dickstein. 2017. "REBAR: Low-Variance, Unbiased Gradient Estimates for Discrete Latent Variable Models." *Advances in Neural Information Processing Systems* 30.

Uhlenbeck, George E., and Leonard S. Ornstein. 1930. "On the Theory of the Brownian Motion." *Physical Review* 36 (5): 823–41.

Vapnik, V., and A. Chervonenkis. 1971. "On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities." *Theory Probab. Appl.* 16 (2): 264–81. https://doi.org/10.1137/1116025.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, et al. 2017. "Attention Is All You Need." *Advances in Neural Information Processing Systems*, 5998–6008. https://doi.org/10.5555/3295222.3295349.

Vaswani, Sharan, Francis Bach, and Mark Schmidt. 2019. "Fast and Faster Convergence of SGD for over-Parameterized Models and an Accelerated Perceptron." *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, Proceedings of machine learning research, vol. 89: 1195–204.

Veit, Andreas, Michael J Wilber, and Serge Belongie. 2016. "Residual Networks Behave Like Ensembles of Relatively Shallow Networks." *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/1605.06431.

Vershynin, Roman. 2018. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press.

Vincent, Pascal. 2011. "A Connection Between Score Matching and Denoising Autoencoders." *Neural Computation* 23 (7): 1661–74.

Vovk, Vladimir, Alexander Gammerman, and Glenn Shafer. 2022. *Algorithmic Learning in a Random World*. 2nd ed. Springer. https://doi.org/10.1007/978-3-031-06649-8.

Wang, Feng, and Huaping Liu. 2021. "Understanding the Behaviour of Contrastive Loss." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2495–504.

Wang, Sinong, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. "Linformer: Self-Attention with Linear Complexity." *arXiv Preprint arXiv:2006.04768*.

Wasserman, Larry. 2013. *All of Statistics: A Concise Course in Statistical Inference*. Springer. https://link.springer.com/book/10.1007/978-0-387-21736-9.

Wasserstein, Ronald L, and Nicole A Lazar. 2016. "The ASA Statement on p-Values: Context, Process, and Purpose." *The American Statistician* 70 (2): 129–33. https://doi.org/10.1080/00031305.2016.1154108.

Weed, Jonathan, and Francis Bach. 2019. "Sharp Asymptotic and Finite-Sample Rates of Convergence of Empirical Measures in Wasserstein Distance." *Bernoulli* 25 (4A): 2620–48.

Welch, Lloyd R. 1974. "Lower Bounds on the Maximum Cross Correlation of Signals." *IEEE Transactions on Information Theory* 20 (3): 397–99. https://doi.org/10.1109/TIT.1974.1055219.

Welford, B. P. 1962. "Note on a Method for Calculating Corrected Sums of Squares and Products." *Technometrics* 4 (3): 419–20.

Welling, Max, and Yee W Teh. 2011. "Bayesian Learning via Stochastic Gradient Langevin Dynamics." *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 681–88. https://dl.acm.org/doi/10.5555/3104482.3104568.

Wengert, Robert Edwin. 1964. "A Simple Automatic Derivative Evaluation Program." *Communications of the ACM* 7 (8): 463–64. https://doi.org/10.1145/355588.365726.

Wenliang, Li K., and Heishiro Kanagawa. 2020. "Blindness of Score-Based Methods to Isolated Components and Mixing Proportions." *arXiv Preprint arXiv:2008.10087*. https://arxiv.org/abs/2008.10087.

White, Halbert. 1982. "Maximum Likelihood Estimation of Misspecified Models." *Econometrica* 50 (1): 1–25.

White, Tom. 2016. "Sampling Generative Networks." *arXiv Preprint arXiv:1609.04468*.

Wiener, Norbert. 1923. "Differential-Space." *Journal of Mathematics and Physics* 2: 131–74.

Williams, Ronald J. 1992. "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning." *Machine Learning* 8: 229–56.

Witten, Ian H., Radford M. Neal, and John G. Cleary. 1987. "Arithmetic Coding for Data Compression." *Communications of the ACM* 30 (6): 520–40.

Wong, Eugene, and Moshe Zakai. 1965. "On the Convergence of Ordinary Integrals to Stochastic Integrals." *The Annals of Mathematical Statistics* 36 (5): 1560–64.

Woodruff, David P. 2014. "Sketching as a Tool for Numerical Linear Algebra." *Foundations and Trends in Theoretical Computer Science* 10 (1–2): 1–157.

Wu, C. F. Jeff. 1983. "On the Convergence Properties of the EM Algorithm." *Annals of Statistics* 11 (1): 95–103.

Wu Fung, Samy, Howard Heaton, Qiuwei Li, Daniel McKenzie, Stanley Osher, and Wotao Yin. 2022. "JFB: Jacobian-Free Backpropagation for Implicit Networks." *Proceedings of the AAAI Conference on Artificial Intelligence* 36: 6648–56. https://doi.org/10.1609/aaai.v36i6.20619.

Xiao, Han, Kashif Rasul, and Roland Vollgraf. 2017. "Fashion-MNIST: A Novel Image Dataset for Benchmarking Machine Learning Algorithms." *ArXiv:1708.07747*. https://arxiv.org/abs/1708.07747.

Xie, Shuo, and Zhiyuan Li. 2024. "Implicit Bias of AdamW: ℓ_∞ Norm Constrained Optimization." *arXiv Preprint arXiv:2404.04454*.

Xu, Yilun, Shengjia Zhao, Jiaming Song, Russell Stewart, and Stefano Ermon. 2020. "A Theory of Usable Information Under Computational Constraints." *International Conference on Learning Representations*.

Yin, Y. Q., Z. D. Bai, and P. R. Krishnaiah. 1988. "On the Limit of the Largest Eigenvalue of the Large Dimensional Sample Covariance Matrix." *Probability Theory and Related Fields* 78 (4): 509–21.

Zadrozny, Bianca, and Charles Elkan. 2002. "Transforming Classifier Scores into Accurate Multiclass Probability Estimates." *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 694–99. https://doi.org/10.1145/775047.775151.

Zhang, Chiyuan, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. 2021. "Understanding Deep Learning (Still) Requires Rethinking Generalization." *Communications of the ACM* 64 (3): 107–15. https://doi.org/10.1145/3446776.

Zhang, Qinsheng, and Yongxin Chen. 2023. "Fast Sampling of Diffusion Models with Exponential Integrator." *International Conference on Learning Representations*. https://arxiv.org/abs/2204.13902.

Zinkevich, Martin. 2003. "Online Convex Programming and Generalized Infinitesimal Gradient Ascent." *Proceedings of the 20th International Conference on Machine Learning*, 928–36.

Ziv, Jacob, and Abraham Lempel. 1977. "A Universal Algorithm for Sequential Data Compression." *IEEE Transactions on Information Theory* 23 (3): 337–43. https://doi.org/10.1109/TIT.1977.1055714.

Ziv, Jacob, and Abraham Lempel. 1978. "Compression of Individual Sequences via Variable-Rate Coding." *IEEE Transactions on Information Theory* 24 (5): 530–36. https://doi.org/10.1109/TIT.1978.1055934.