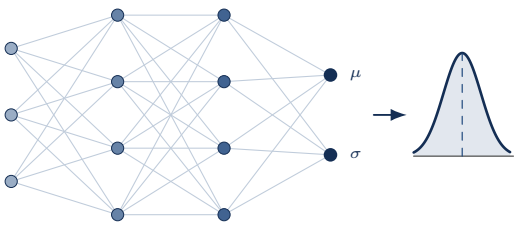
Principles of Machine Learning

A Probabilistic Approach to Pattern Recognition and Deep Learning



Taka Khoo

Thayer School of Engineering
Dartmouth College

Principles of Machine Learning: A Probabilistic Approach to Pattern Recognition and Deep Learning. First edition, first printing.

Developed from the lectures of Professor Peter Chin for ENGS 106, *Principles of Machine Learning*, at the Thayer School of Engineering, Dartmouth College. The mathematical development follows Christopher M. Bishop, *Pattern Recognition and Machine Learning*, and Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep Learning*; the preface tells that story.

A complete, self-contained development of machine learning, from the rules of probability to the architecture of a Transformer. Every definition is stated, every symbol named, every proof written out in full, and every problem solved step by step, with each algorithm traced once by hand on real numbers. The body stands on its own as a textbook.

Written at the Thayer School of Engineering, Dartmouth College, Hanover, New Hampshire. Printed in the United States of America.

Suggested citation. T. Khoo, *Principles of Machine Learning: A Probabilistic Approach to Pattern Recognition and Deep Learning*, 1st ed., Thayer School of Engineering, Dartmouth College, Hanover, NH, 2026.

To cite a chapter. T. Khoo, "[chapter title]," in *Principles of Machine Learning*, 1st ed., Thayer School of Engineering, Dartmouth College, 2026, ch. [n]. In a syllabus or reading list this shortens to *Khoo, Principles of Machine Learning*, ch. [n].

Written, typeset, and illustrated by Taka Khoo in Latin Modern with LaTeX; figures drawn in Ti*k*Z. Prepared as a gift for Professor Chin.

For Professor Peter Chin,

who taught me to read the pattern in the data,
and to stay honest about how much it leaves unsaid.

Preface

This book grew out of ENGS 106, *Principles of Machine Learning*, taught by Professor Peter Chin at Dartmouth. Over a ten-week term the course moves from the rules of probability to the architecture of a Transformer, and it does so without ever waving its hands: every model is derived, every estimator is solved for, and every approximation is justified. The course accumulated a large body of material along the way, lecture notes, slide decks, pages of handwritten board work, programming assignments, and a term project. This book gathers all of it into one place, fills in the steps that a live lecture leaves implicit, and arranges the result the way a textbook should be arranged.

What this book is

The goal is completeness without compromise. Every topic Professor Chin covered appears here. Every worked example is carried to its final answer. Every variable is defined where it first appears. When a derivation in the original notes skipped from one line to the next, the intermediate algebra has been restored so that nothing has to be taken on faith. The intent is that you can open to any page, point at any equation, and trace it backward to a definition without leaving the book.

ENGS 106 has a double character, and the book keeps both halves. One half is the classical, Bayesian view of pattern recognition that runs through Christopher Bishop's text: probability distributions and their conjugate priors, the exponential family, linear models for regression and classification read as maximum likelihood and then as full Bayesian inference, kernels, graphical models, and the expectation-maximization algorithm. The other half is the modern, hands-on view of deep learning that Professor Chin develops at the board: empirical risk and the VC dimension, gradient descent and the normal equations derived by matrix calculus, backpropagation written out for a real network, and the architectures that carried the field from LeNet to the Transformer. The two halves are not rivals. The same probabilistic thread runs through both, and the book follows it.

How it is organized

The organization follows the mathematics rather than the calendar. A term runs ten weeks, and a lecture is a unit of time rather than a unit of thought. The chapters are built around ideas that belong together, and a single chapter often draws on several lectures, on the slides, and on the board notes at once. The seven parts mirror the arc Professor Chin laid out at the start: the probabilistic foundations and learning theory, then regression, then classification, then neural networks and deep learning, then kernel methods, then graphical models, and finally unsupervised learning. Each part builds on the ones before it, and the probabilistic point of view set up in the first part is the connective tissue for everything that follows.

How to use it

Each chapter opens with a short roadmap and a suggested reading in the two course texts, Bishop [3] for the probabilistic and kernel material and Goodfellow et al. [13] for the deep-learning material. The body develops the theory with full proofs and runs detailed worked examples in the **Example** environments. Definitions, theorems, and lemmas share a single numbering stream per chapter, so Theorem 3.4 sits right after Definition 3.3 and you never have to guess which counter you are reading.

Every chapter ends with a set of **Exercises**, each followed by a complete solution. These are not decorative. They are the problems the course actually asked, restated in full and solved in full, line by line, together with supplementary problems that round out a topic the lectures only gestured at. Nothing is left as "it can be shown."

The appendices collect the reference material you reach for repeatedly: a linear algebra refresher, a probability and distributions sheet, the matrix calculus and Lagrangian-duality identities used in the optimization-heavy chapters, a guide to the software and the programming assignments, and a bank of practice problems with solutions.

A note on notation

Machine learning inherits two notational traditions, and this course speaks both. Where the development follows Bishop, targets are written t and the design matrix is $\boldsymbol{\Phi}$; where it follows the lecture board, a data set is $\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$ with m examples. Throughout, vectors are bold lowercase ($\boldsymbol{x}$, $\boldsymbol{\theta}$, $\boldsymbol{w}$), matrices are uppercase (A, X, with bold uppercase greek $\boldsymbol{\Sigma}$ for matrices that are naturally greek), and scalars are lowercase italic. Transpose is $A^{\top}$, inverse A^{-1}, and the Moore–Penrose pseudoinverse $A^{\dagger}$. The full table of symbols follows the table of contents, and the conventions are consistent across every chapter.

Acknowledgments

The mathematics, the examples, the analogies, and the way the whole subject hangs together belong to Professor Peter Chin. My part was to write it all down carefully and leave nothing out. Thank you for the class, and for teaching machine learning as the rigorous, beautiful subject it is.

Taka Khoo
Hanover, New Hampshire

How to Use This Book

This is a book you can read front to back, or open to any single result and find it self-contained. This short section explains how it is laid out, how the pieces depend on one another, and how to find any statement by its number.

The shape of the book

The material is organized into seven parts.

Part I, Foundations.
The probabilistic bedrock: the rules of probability and the maximum-likelihood, MAP, and fully Bayesian ways to fit a model; the distributions the rest of the book is built from and their conjugate priors; the exponential family that unifies them; and the learning theory, empirical risk, the bias–variance tradeoff, and the VC dimension, that says when learning from finite data can work at all.

Part II, Regression.
Predicting a continuous target: linear models in a basis of features, solved three ways (gradient descent, the normal equations, and maximum likelihood), with regularization and the bias–variance decomposition; then the full Bayesian treatment, with a posterior over weights and a predictive distribution that reports its own uncertainty.

Part III, Classification.
Predicting a label: linear discriminants, Fisher's projection, and the perceptron; the generative and discriminative probabilistic models, Gaussian discriminant analysis, naive Bayes, and logistic regression with its iterative reweighted least squares; and the Bayesian treatment through the Laplace approximation.

Part IV, Neural Networks and Deep Learning.
Learning the features themselves: feed-forward networks and universal approximation, backpropagation derived in full, and the architectures that defined modern deep learning, the convolutional networks from LeNet to ResNet, and the recurrent and attention based sequence models that lead to the Transformer.

Part V, Kernel Methods.
The dual point of view: rewriting a linear model in terms of inner products, replacing those inner products by a kernel, and arriving at the maximum-margin support vector machine through Lagrangian duality and the Karush–Kuhn–Tucker conditions.

Part VI, Graphical Models.
Structure in joint distributions: directed Bayesian networks and undirected Markov random fields, conditional independence and d-separation, and exact inference by message passing on factor graphs.

Part VII, Unsupervised Learning.
Finding structure without labels: clustering and Gaussian mixtures through the expectation–maximization algorithm, principal and independent component analysis, and sampling methods for the integrals that cannot be done in closed form.

The appendices supply the mathematical background the book assumes, so it stands on its own with only single-variable calculus and basic linear algebra taken for granted: a linear-algebra reference that works the eigendecomposition and the singular value decomposition by hand, a probability and distributions reference, the matrix-calculus identities with a treatment of Lagrangian duality and the KKT conditions, a guide to the software and the programming assignments, and a bank of practice problems plus a full practice midterm and final, all worked in full.

Three kinds of problems

Almost every question in this subject is one of three kinds, and it pays to classify a problem in the first ten seconds.

1. **Compute** something concrete: a posterior distribution, a least-squares fit, a backpropagation step, a kernel matrix, the dual of a support vector machine, an expectation–maximization update. These reward careful arithmetic. Show every step.
2. **Prove** an identity or guarantee: that a covariance matrix is positive semidefinite, that the perceptron converges, that the expectation–maximization algorithm never decreases the likelihood, that weak duality holds. These reward structure. State what you assume, state what you want, then go by construction or contradiction.
3. **Explain** a modeling choice: why a conjugate prior, why cross-entropy rather than squared error, why the radial basis kernel, why a soft margin, why a Bayesian posterior instead of a point estimate. These reward one clear sentence and one supporting fact.

The worked examples in every chapter and the exercises that close each chapter are written with these three categories in mind.

Conventions

- **Numbering.** Definitions, theorems, lemmas, propositions, corollaries, remarks, and examples share a single counter within each chapter. So Definition 3.3 and Theorem 3.4 are consecutive, and you can find any result by its number alone. Equations, figures, and exercises are numbered separately, also by chapter.
- **Cross-references** are live links in the digital version. "See Chapter 15" and similar pointers jump to the exact place.
- **Colored boxes are rare and meaningful.** A box with a dark navy border holds the single canonical statement of a major result. A green box is a step-by-step recipe. A red left-rule flags a subtlety or a common mistake. Everything else is ordinary prose and displayed mathematics.
- **Vectors and matrices.** Vectors are bold lowercase ($\boldsymbol{x}$, $\boldsymbol{\theta}$); matrices are uppercase (A, X), with bold uppercase greek ($\boldsymbol{\Sigma}$) for matrices that are naturally greek; scalars are lowercase italic. Transpose is $A^\top$, inverse A^{-1}, pseudoinverse $A^\dagger$. The full symbol table follows this section.

A map of the book

The chapters follow themes rather than the lecture calendar: a single chapter often gathers several lectures, the slide decks, and the board notes, and the order is the logical one. The table records the core topics of each chapter and the corresponding reading in the course texts, Bishop [3] and, for the deep-learning chapters, Goodfellow et al. [13], so any statement can be placed in the larger arc.

Chapter	Core topics	Reading
1. The Machine Learning Problem	learning from data, overfitting, frequentist vs Bayesian, decision and information theory	Bishop 1
2. Probability Distributions	Bernoulli/Beta, multinomial/Dirichlet, the Gaussian family	Bishop 2.1–2.3
3. The Exponential Family	canonical form, sufficient statistics, conjugacy	Bishop 2.4
4. Learning Theory	empirical risk, bias–variance, the VC dimension	Bishop 1
5. Linear Models for Regression	basis functions, least squares, regularization	Bishop 3.1–3.2
6. Bayesian Linear Regression	parameter posterior, predictive distribution, evidence	Bishop 3.3–3.5
7. Linear Models for Classification	discriminants, Fisher's LDA, the perceptron	Bishop 4.1
8. Probabilistic Classification	GDA, naive Bayes, logistic regression, IRLS	Bishop 4.2–4.3
9. Bayesian Logistic Regression	the Laplace approximation, predictive distribution	Bishop 4.4–4.5
10. Feed-Forward Neural Networks	architecture, universal approximation, losses	Bishop 5.1; DL 6
11. Backpropagation and Training	backprop, the Hessian, regularization, optimizers	Bishop 5.2–5.5; DL 6–8
12. Convolutional Networks	convolution, pooling, LeNet to ResNet	DL 9
13. Sequence Models and Transformers	recurrent nets, LSTM, attention, the Transformer	DL 10
14. Kernel Methods	dual representation, the kernel trick, Mercer	Bishop 6
15. Support Vector Machines	max margin, duality, soft margin, kernels	Bishop 7.1
16. Probabilistic Graphical Models	Bayesian networks, d-separation, Markov random fields	Bishop 8.1–8.3
17. Inference in Graphical Models	message passing, sum–product, max–sum	Bishop 8.4
18. Mixture Models and EM	k-means, Gaussian mixtures, expectation–maximization	Bishop 9
19. PCA and ICA	principal components, independent components, factor analysis	Bishop 12
20. Sampling and Monte Carlo	rejection and importance sampling, MCMC	Bishop 11

Dependencies and reading paths

The book is cumulative. The probabilistic spine is Chapter 1 → Chapter 2 → Chapter 3: these set up maximum likelihood, the Bayesian posterior, the distributions, and the exponential family that everything else specializes. Chapter 4 on learning theory can be read independently, but it is the conceptual justification for the model-selection choices made later. The regression chapters Chapter 5 and Chapter 6 are the cleanest place to see the maximum-likelihood and Bayesian viewpoints side by side, and the classification chapters Chapter 7–Chapter 9 reuse that machinery for discrete targets. Neural networks (Chapters 10 to 13) depend on the gradient-based optimization of Chapter 5 and on backpropagation, which is the chain rule of Chapter 11. The kernel chapters Chapter 14 and Chapter 15 need the Lagrangian duality developed in Appendix C. The graphical-model chapters Chapter 16 and Chapter 17 and the unsupervised chapters Chapters 18 to 20 return to the distributions of Chapter 2 and the Gaussian conditioning of Chapter 6, closing the loop with the probability with which the book began.

Contents

List of Figures

List of Tables

Notation

The conventions below are used consistently throughout the book. Vectors are bold lowercase, matrices are uppercase, and scalars are lowercase italic. Where two traditions collide, the Bishop convention (t for a regression target, $\boldsymbol{\Phi}$ for the design matrix) is used in the Bayesian chapters and the lecture-board convention (m examples $\boldsymbol{x}^{(i)}$, label $y^{(i)}$) in the others; both are listed here.

Sets and numbers

$\mathbb{R},\ \mathbb{R}^n,\ \mathbb{R}^{m\times n}$	real numbers; real n-vectors; real $m \times n$ matrices
$\mathbb{C},\ \mathbb{Z},\ \mathbb{N}$	complex numbers, integers, natural numbers
$\{\,\cdot\,\},\ \{x \mid P(x)\}$	a set; the set of x such that $P(x)$ holds
$\lvert S\rvert$	cardinality of a finite set S
$[k]$	the index set $\{1, 2, \dots, k\}$
$\mathbf{1}[\,\cdot\,]$	indicator: 1 if the statement holds, 0 otherwise

Linear algebra

$\boldsymbol{x},\ \boldsymbol{y},\ \boldsymbol{w},\ \boldsymbol{\theta}$	vectors (bold lowercase)
$A,\ X,\ \boldsymbol{\Sigma},\ \boldsymbol{\Phi}$	matrices (uppercase; bold greek for greek matrices)
$x_i,\ a_{ij}$	the ith entry of $\boldsymbol{x}$; the (i, j) entry of A
$\boldsymbol{a}_j$	the jth column of A
$A^\top,\ \boldsymbol{x}^\top$	transpose
$A^{-1},\ A^\dagger$	inverse (when it exists); Moore–Penrose pseudoinverse
$I,\ I_n$	identity matrix (of size n)
$\mathbf{0},\ \mathbf{1},\ \boldsymbol{e}_i$	all-zeros, all-ones, and ith standard basis vectors
$\operatorname{tr}(A),\ \operatorname{rank}(A),\ \det(A)$	trace, rank, determinant
$\operatorname{diag}(\boldsymbol{d})$	diagonal matrix with diagonal $\boldsymbol{d}$
$\operatorname{Col}(A),\ \operatorname{Row}(A),\ \operatorname{Nul}(A)$	column space, row space, null space
$\operatorname{span}\{\boldsymbol{v}_1, \dots, \boldsymbol{v}_k\}$	span (set of all linear combinations)
$\lVert\boldsymbol{x}\rVert,\ \lVert\boldsymbol{x}\rVert_1,\ \lVert\boldsymbol{x}\rVert_\infty$	Euclidean (ℓ_2), ℓ_1, and ℓ_∞ norms
$\langle \boldsymbol{x}, \boldsymbol{y}\rangle = \boldsymbol{x}^\top \boldsymbol{y}$	inner (dot) product
$A \succeq 0,\ A \succ 0$	A is positive semidefinite; positive definite

Probability and distributions

$\mathbb{P}(A)$	probability of event A
$p(x),\ p(x \mid y)$	density/mass of x; conditional density of x given y
$\mathbb{E}[X],\ \mathrm{Var}(X),\ \mathrm{Cov}(\boldsymbol{x})$	expectation; variance; covariance matrix
$X \sim p,\ X \overset{\text{iid}}{\sim} p$	X is distributed as p; independent and identically so
$X \perp\!\!\!\perp Y$	X and Y are independent
$\mathrm{Bern}(\mu),\ \mathrm{Bin}(N, \mu)$	Bernoulli, Binomial
$\mathrm{Cat}(\boldsymbol{\pi}),\ \mathrm{Mult}(N, \boldsymbol{\pi})$	Categorical, Multinomial
$\mathrm{Beta}(a, b),\ \mathrm{Dir}(\boldsymbol{\alpha})$	Beta, Dirichlet
$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	(multivariate) Gaussian / normal
$\mathrm{Gam}(a, b),\ \mathcal{W}(\boldsymbol{W}, \nu),\ \mathrm{St}(\mu, \lambda, \nu)$	Gamma, Wishart, Student's t
$\boldsymbol{\eta},\ \boldsymbol{u}(\boldsymbol{x}),\ g(\boldsymbol{\eta})$	natural parameters; sufficient statistics; normalizer (exponential family)
$\mathrm{H}[p],\ \mathrm{KL}(p \,\|\, q)$	entropy; Kullback–Leibler divergence
$\hat{\theta}_{\mathrm{ML}},\ \hat{\theta}_{\mathrm{MAP}}$	maximum likelihood, maximum a posteriori estimates

Calculus and optimization

$\frac{\partial f}{\partial x}$	partial derivative
$\nabla f,\ \nabla_{\boldsymbol{\theta}} f$	gradient (with respect to $\boldsymbol{\theta}$)
$\nabla^2 f,\ \boldsymbol{J}$	Hessian; Jacobian
$\arg\min_{\boldsymbol{x}} f(\boldsymbol{x}),\ \arg\max_{\boldsymbol{x}} f(\boldsymbol{x})$	a minimizer; a maximizer of f
$f \in \mathcal{O}(g)$	f grows no faster than g
$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$	Lagrangian
$\mathrm{dom}\, f,\ \mathrm{epi}\, f$	domain; epigraph
$:=$	"is defined to equal"

Machine learning

$\mathcal{D} = \{(\boldsymbol{x}^{(i)}, y^{(i)})\}_{i=1}^{m}$	training set of m examples (lecture-board convention)
$t,\ \boldsymbol{t}$	regression target; vector of targets (Bishop convention)
$\boldsymbol{\Phi},\ \phi_j(\boldsymbol{x}),\ \boldsymbol{\phi}(\boldsymbol{x})$	design matrix $\boldsymbol{\Phi}_{ij} = \phi_j(\boldsymbol{x}^{(i)})$; basis function; feature map
$\boldsymbol{w},\ \boldsymbol{\theta},\ b$	weight vector; parameters; bias
$\alpha,\ \beta$	prior precision; noise precision
$\boldsymbol{m}_N,\ \boldsymbol{S}_N$	posterior mean and covariance of the weights
$k(\boldsymbol{x}, \boldsymbol{x}'),\ K$	kernel function; Gram (kernel) matrix
$\sigma(a) = \frac{1}{1+e^{-a}}$, softmax	logistic sigmoid; softmax
$a_j^{[\ell]},\ z_j^{[\ell]},\ \delta_j^{[\ell]}$	pre-activation, activation, and backprop error of unit j in layer ℓ
$\gamma(z_{nk})$	responsibility of mixture component k for example n
$\mathrm{pa}(x_i)$	the parents of node x_i in a graphical model
$X \perp\!\!\!\perp Y \mid Z$	X and Y are conditionally independent given Z

PART I

Foundations

CHAPTER 1

The Machine Learning Problem

Machine learning is not magic. It is mathematics, probability, linear algebra, and optimization, applied with care to data.

after Professor Peter Chin, ENGS 106

This chapter sets the stage for the whole book. We say what the machine learning problem actually is, we meet the single example that motivates almost everything that follows (fitting a curve to noisy data), and we lay down the three ways one can fit a model to data: maximum likelihood, the maximum a posteriori estimate, and the full Bayesian posterior. Those three principles run through every later chapter, from linear regression to the Transformer. We close with the two pieces of mathematical infrastructure that turn a fitted model into a decision and that measure how much one distribution tells us about another: decision theory and information theory.

Roadmap

Suggested reading: Bishop [3], Chapter 1, and for the historical and deep-learning context, Goodfellow et al. [13], Chapter 1.

We cover: the supervised, unsupervised, and reinforcement settings (Section 1.1); polynomial curve fitting and the overfitting it exposes (Section 1.2); model complexity and generalization (Section 1.3); the likelihood, the prior, the posterior, and the predictive distribution, with the frequentist and Bayesian viewpoints side by side (Section 1.4); decision theory and the optimal predictor (Section 1.5); and entropy, relative entropy, and mutual information (Section 1.6).

1.1 What machine learning is

A useful working definition is the one due to Mitchell [31]. A computer program learns from experience E with respect to a task T and a performance measure P if its performance at T, as measured by P, improves with experience E. The three pieces are worth naming because every machine learning problem in this book fixes them in turn. The task T might be predicting a house price, recognizing a handwritten digit, or clustering documents by topic. The experience E is the data: a collection of examples. The performance measure P is a number we can compute and try to improve, an error rate or a likelihood or a reconstruction loss.

What separates this from ordinary programming is that we do not write the rule that maps an input to an output. We write a procedure that, given data, produces the rule. The rule is fit to the data rather than dictated to it. This is why the subject leans so heavily on probability and optimization: probability to model the data and our uncertainty about it, optimization to do the fitting.

1.1.1 Three kinds of learning

The problems in this book fall into three families.

Supervised learning.
The data come in input-output pairs, and the goal is to predict the output for a new input. We write the training set as

$$\mathcal{D} = \left\{(\boldsymbol{x}^{(i)}, y^{(i)})\right\}_{i=1}^{m}, \qquad \boldsymbol{x}^{(i)} \in \mathbb{R}^n, \tag{1.1}$$

Figure 1.1. A few landmarks on the road from least squares to the Transformer. Each appears as a chapter or a section in this book. The field itself was named at the Dartmouth summer workshop of 1956, which falls between the artificial neuron (1943) and the perceptron (1958) marked here.

with m labeled examples, each an input vector $\boldsymbol{x}^{(i)}$ and a target $y^{(i)}$. When the target is continuous ($y \in \mathbb{R}$) the problem is *regression*; when it is one of a finite set of labels the problem is *classification*. Chapters 5 through 9 are about supervised learning, and so are the neural networks of Part IV and the kernel machines of Part V.

Unsupervised learning.
The data are inputs alone, $\mathcal{D} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{m}$, with no targets, and the goal is to find structure: clusters of similar points, a low-dimensional subspace that captures most of the variation, or a density that could have generated the data. The mixture models of Chapter 18 and the component analyses of Chapter 19 are unsupervised.

Reinforcement learning.
An agent takes actions in an environment and receives rewards, and the goal is a policy that maximizes reward over time. We do not treat reinforcement learning in this book, but the probabilistic and optimization tools developed here are its foundation.

Remark 1.1 *(A note on notation).* The supervised set in Eq. (1.1) uses the convention of the lecture board: m examples, inputs $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$, label $y^{(i)}$. The probabilistic chapters that follow Bishop will write a regression target as t and collect the design information into a matrix $\boldsymbol{\Phi}$. Both conventions appear in the literature, and both appear here; the symbol table after the contents lists them together.

1.1.2 A field that began at Dartmouth

It is worth pausing on the history, because some of it happened here. The phrase "artificial intelligence" was coined in the proposal for a summer workshop held at Dartmouth College in 1956, written by John McCarthy, Marvin Minsky, Nathaniel Rochester, and Claude Shannon [28]. The proposal conjectured "that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it." The seventy years since have been a long argument with that sentence, and machine learning is the part of the argument that has gone best.

The mathematical lineage is older and runs straight through the chapters ahead. Figure 1.1 lays it out. Least squares, the workhorse of Chapter 5, goes back to Gauss and Legendre around 1800. The artificial neuron of McCulloch and Pitts [29] and the perceptron of Rosenblatt [32] are the seeds of Part IV. The statistical learning theory of Vapnik and Chervonenkis [41] underwrites Chapter 4, and the support vector machine [7] is the subject of Chapter 15. The convolutional network of LeCun et al. [25], the deep network of Krizhevsky et al. [23], and the attention mechanism of Vaswani et al. [42] are the modern monuments of Part IV.

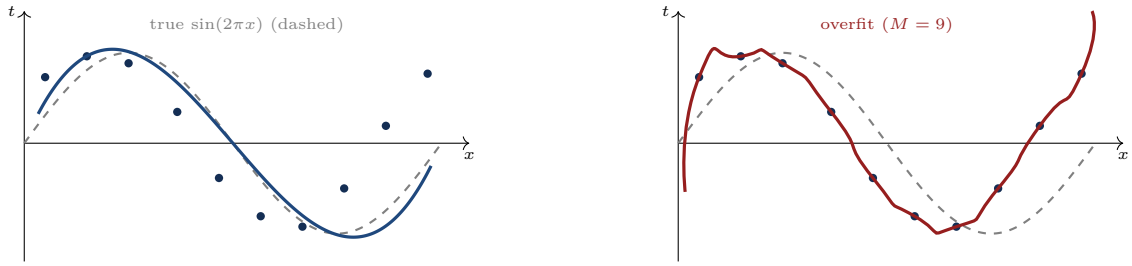
1.2 A first example: fitting a curve

Almost every tension in the subject shows up in the simplest supervised problem one can write down, so we start there. Suppose we are given m points $(x^{(i)}, t^{(i)})$ with scalar inputs and targets, and we believe the targets were produced by some smooth function corrupted by noise. The example to keep in mind, following Bishop [3], is data generated from $\sin(2\pi x)$ with a little Gaussian noise added to each target. We do not get to see the sine; we see only the points, and we would like to recover a function that predicts t at a new x.

A natural family of candidate functions is the polynomials of degree M,

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j. \tag{1.2}$$

This is our first model. Notice that although y is a nonlinear function of the input x, it is a *linear* function of the parameters $\boldsymbol{w} = (w_0, \ldots, w_M)$. That linearity is the entire reason the fit will have a clean closed form, and it is



(a) A flexible-enough model ($M = 3$) tracks the underlying function.

(b) Too flexible a model ($M = 9$) interpolates the noise and oscillates.

Figure 1.2. Ten noisy samples of $\sin(2\pi x)$ (one period shown along x). A degree-3 polynomial recovers the shape; a degree-9 polynomial has enough freedom to pass through every point, fitting the noise and swinging violently between samples. Schematic, after Bishop [3].

the defining feature of the linear models of Chapter 5. We choose $\boldsymbol{w}$ to make the curve pass close to the data, by minimizing the sum of squared errors

$$E(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{m}\big(y(x^{(i)}, \boldsymbol{w}) - t^{(i)}\big)^2. \tag{1.3}$$

Chapter 5 solves this minimization in three different ways and explains the factor of $\frac{1}{2}$. For now the only thing that matters is the qualitative behavior as we change the one knob we have not fixed: the degree M, which controls how flexible the model is.

Figure 1.2 shows what happens. With $M = 0$ or $M = 1$ the polynomial is a flat line or a single slope, too rigid to bend with the sine; this is *underfitting*. With $M = 3$ the curve has enough freedom to follow the true function and ignores the noise. With $M = 9$, the polynomial has ten coefficients for ten data points, so it can be made to pass exactly through every one of them. The training error Eq. (1.3) drops to zero, and yet the curve is useless: it oscillates wildly between the data points and would predict nonsense at a new x. This is *overfitting*. The model has fit the noise instead of the signal.

Remark 1.2 *(The symptom in the coefficients).* The overfit polynomial betrays itself in the size of its coefficients. As M grows toward the number of data points, the least-squares coefficients in Eq. (1.2) grow enormous, into the hundreds of thousands and beyond, with large positive and negative values nearly cancelling [3, Table 1.1]. The curve achieves zero training error by setting up huge opposing terms that cancel at the data points and explode between them. This observation, that overfitting lives in large parameter values, is exactly what regularization will exploit in Chapter 5.

1.3 Model complexity and generalization

The quantity we actually care about is not the error on the training data. It is the error the model will make on data it has not seen. Call the first the *training error* and the second the *generalization error*. Overfitting is precisely the gap between them: a model can drive training error to zero while its generalization error grows.

Definition 1.3 (Training and generalization error). For a model h and a per-example loss ℓ, the *training error* (or empirical error) on a data set $\mathcal{D}$ of size m is the average loss on that set,

$$\widehat{\varepsilon}_{\mathcal{D}}(h) = \frac{1}{m}\sum_{i=1}^{m}\ell\big(h(\boldsymbol{x}^{(i)}), y^{(i)}\big),$$

and the *generalization error* is the expected loss on a fresh example drawn from the same distribution $p(\boldsymbol{x}, y)$ that produced the data,

$$\varepsilon(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim p}\big[\ell(h(\boldsymbol{x}), y)\big].$$

We can only ever measure $\widehat{\varepsilon}_{\mathcal{D}}$, but we want ε to be small. The practical device is to hold out part of the data as a *test set*, never used in fitting, and to estimate the generalization error on it. Figure 1.3 plots both errors against

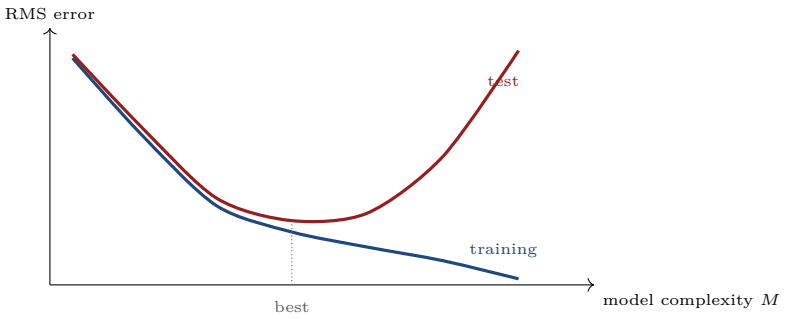


Figure 1.3. Training error falls monotonically with model complexity and reaches zero once the model can interpolate the data. Test (generalization) error is U-shaped: the model is too rigid on the left and overfits on the right. The art is to find the bottom of the valley.

model complexity M for the curve-fitting example. The training error falls monotonically as M grows, reaching zero when the model can interpolate the data. The test error tells a different story: it falls at first, as the model gains the flexibility it needs, then rises as the model begins to fit the noise. The best M sits at the bottom of that valley.

Two levers move that valley. The first is the amount of data. A model that overfits on ten points may fit beautifully on a hundred, because there is no longer enough freedom to chase every point. As a rule of thumb, the more flexible the model, the more data it needs. The second lever is *regularization*: adding a penalty on large parameter values to Eq. (1.3), which discourages the cancelling coefficients that overfitting relies on. Chapter 5 makes this precise, and Chapter 4 turns the whole picture into theorems by asking, for a given model and a given amount of data, how large the gap between training and generalization error can be. For now the lesson is qualitative and central: the goal of learning is generalization, and fitting the training data is only a means to it.

1.4 Three ways to fit a model

Behind the curve-fitting story is a modeling choice we made silently: minimize squared error. Where did that come from? The cleanest answer, and the one that organizes the rest of the book, is probabilistic. We posit a probability model for how the data were generated, and then we let a principle of inference choose the parameters. There are three such principles, and the difference between them is one of the deep themes of this course.

The motivating question, as Professor Chin poses it, is this. *Given a training set, how likely is it that this training set would occur, under a given model?* That question turns learning into inference. We have a model with parameters $\boldsymbol{\theta}$, and the model assigns a probability $p(\mathcal{D} \mid \boldsymbol{\theta})$ to the data we observed.

1.4.1 Maximum likelihood: the frequentist point estimate

In the frequentist view, the parameters $\boldsymbol{\theta}$ are fixed but unknown constants, and learning returns a single best estimate of them. The natural choice is the value that makes the observed data most probable.

Definition 1.4 (Likelihood and maximum likelihood). The *likelihood* is the probability of the data viewed as a function of the parameters, $L(\boldsymbol{\theta}) = p(\mathcal{D} \mid \boldsymbol{\theta})$. The *maximum likelihood estimate* is

$$\boldsymbol{\theta}^{*}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \; p(\mathcal{D} \mid \boldsymbol{\theta}). \tag{1.4}$$

Because the logarithm is increasing, maximizing the likelihood is the same as maximizing the *log-likelihood* $\log p(\mathcal{D} \mid \boldsymbol{\theta})$, and for data drawn independently the log turns the product over examples into a sum, which is far easier to differentiate:

$$\log p(\mathcal{D} \mid \boldsymbol{\theta}) = \log \prod_{i=1}^{m} p\big(y^{(i)} \mid \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\big) = \sum_{i=1}^{m} \log p\big(y^{(i)} \mid \boldsymbol{x}^{(i)}, \boldsymbol{\theta}\big). \tag{1.5}$$

Maximizing the log-likelihood is the same as minimizing its negative, the *negative log-likelihood*, which is the loss function that most of this book quietly minimizes. We will see in Chapter 5 that if the targets differ from the model

by Gaussian noise, the negative log-likelihood is exactly the sum of squared errors Eq. (1.3). The squared-error criterion we chose by instinct is the maximum likelihood criterion under a Gaussian noise assumption.

Remark 1.5 *(Maximum likelihood can be brittle).* A point estimate commits to one value of $\boldsymbol{\theta}$ and reports nothing about how sure it is. Flip a coin three times and see three heads; the maximum likelihood estimate of the probability of heads is $3/3 = 1$, a confident claim that tails is impossible, on the strength of three flips. The estimate is not wrong about the data, it simply has no way to express that three flips is thin evidence. The two principles that follow address exactly this.

1.4.2 The Bayesian posterior

The Bayesian view treats $\boldsymbol{\theta}$ as a random variable. We encode our beliefs about $\boldsymbol{\theta}$ before seeing data as a *prior* $p(\boldsymbol{\theta})$, and we update those beliefs in light of the data to a *posterior* $p(\boldsymbol{\theta} \mid \mathcal{D})$. The update is Bayes' rule, and it is worth deriving from the definition of conditional probability rather than quoting. The joint distribution factors two ways,

$$p(\boldsymbol{\theta}, \mathcal{D}) = p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}) = p(\boldsymbol{\theta} \mid \mathcal{D})\, p(\mathcal{D}),$$

and equating the two and dividing by $p(\mathcal{D})$ gives the rule.

Bayes' rule for parameters

$$\underbrace{p(\boldsymbol{\theta} \mid \mathcal{D})}_{\text{posterior}} = \frac{\overbrace{p(\mathcal{D} \mid \boldsymbol{\theta})}^{\text{likelihood}}\ \overbrace{p(\boldsymbol{\theta})}^{\text{prior}}}{\underbrace{p(\mathcal{D})}_{\text{evidence}}}, \qquad p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, d\boldsymbol{\theta}. \tag{1.6}$$

The denominator $p(\mathcal{D})$, the *evidence* or *marginal likelihood*, is the probability of the data with the parameters integrated out. It does not depend on $\boldsymbol{\theta}$, so as a function of $\boldsymbol{\theta}$ it is only a normalizing constant: the posterior is proportional to likelihood times prior, $p(\boldsymbol{\theta} \mid \mathcal{D}) \propto p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})$. The evidence is far from useless, though. It is the quantity that scores one entire model against another, and it returns in Chapter 6 as the engine of Bayesian model comparison.

1.4.3 MAP: a posterior point estimate

The full posterior is a distribution over all of parameter space, which can be more than we want or can compute. A middle path between the single number of maximum likelihood and the whole distribution of the Bayesian posterior is to report the single most probable parameter value under the posterior.

Definition 1.6 (Maximum a posteriori estimate). The *maximum a posteriori* (MAP) estimate is the mode of the posterior,

$$\boldsymbol{\theta}^*_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}}\ p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta}), \tag{1.7}$$

where the evidence $p(\mathcal{D})$ was dropped because it does not depend on $\boldsymbol{\theta}$.

Comparing Eq. (1.7) with Eq. (1.4), the MAP estimate is maximum likelihood with an extra factor, the prior. Taking logs, MAP maximizes $\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$: the log-likelihood plus a term that depends only on the parameters. That extra term is a regularizer. We will see in Chapter 5 that a Gaussian prior on the weights turns into exactly the squared-norm penalty of ridge regression, so MAP estimation *is* regularized maximum likelihood, and the strength of the regularizer is the confidence of the prior. When the prior is flat (uninformative), the extra term is constant and MAP collapses back to maximum likelihood.

	Frequentist	**Bayesian**
View of $\boldsymbol{\theta}$	a fixed unknown constant	a random variable with a distribution
What learning returns	a point estimate $\boldsymbol{\theta}^*_{\text{ML}}$	a posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$
Prediction	plug in $\boldsymbol{\theta}^*$	average over the posterior, Eq. (1.8)
Uncertainty	from imagined repeated sampling	from the posterior, directly
Cost	one optimization	an integral, often intractable

Table 1.1. The two viewpoints on what a parameter is and what it means to learn it. MAP estimation, Eq. (1.7), sits between them: a Bayesian prior with a frequentist point estimate.

1.4.4 The Bayesian predictive distribution

A point estimate, whether maximum likelihood or MAP, makes a prediction by plugging the one chosen $\boldsymbol{\theta}$ into the model. The fully Bayesian alternative refuses to choose. To predict the target y at a new input $\boldsymbol{x}$, it averages the model's prediction over the entire posterior, weighting each $\boldsymbol{\theta}$ by how plausible the data have made it.

Predictive distribution

$$p(y \mid \boldsymbol{x}, \mathcal{D}) = \int p(y \mid \boldsymbol{x}, \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, d\boldsymbol{\theta}. \tag{1.8}$$

This integral is the marginalization of the joint $p(y, \boldsymbol{\theta} \mid \boldsymbol{x}, \mathcal{D})$ over $\boldsymbol{\theta}$, followed by one modeling assumption. Writing the joint with the product rule,

$$p(y \mid \boldsymbol{x}, \mathcal{D}) = \int p(y, \boldsymbol{\theta} \mid \boldsymbol{x}, \mathcal{D})\, d\boldsymbol{\theta} = \int p(y \mid \boldsymbol{x}, \boldsymbol{\theta}, \mathcal{D})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, d\boldsymbol{\theta},$$

and then assuming that once $\boldsymbol{\theta}$ is known the new target depends on the past data only through $\boldsymbol{\theta}$, so that $p(y \mid \boldsymbol{x}, \boldsymbol{\theta}, \mathcal{D}) = p(y \mid \boldsymbol{x}, \boldsymbol{\theta})$, gives Eq. (1.8). The payoff is that the prediction carries its own uncertainty. Where the data are plentiful the posterior is sharp and the predictions of the plausible $\boldsymbol{\theta}$ agree, so the predictive distribution is narrow; where the data are scarce the posterior is broad and the plausible models disagree, so the predictive distribution is wide. That widening of the uncertainty band away from the data is the picture on the cover of this book, and it is the subject of Chapter 6.

1.4.5 Frequentist or Bayesian?

Table 1.1 sets the two viewpoints side by side. The practical relationship among the three estimators is clean: MAP interpolates between maximum likelihood and the full posterior. With no prior information MAP equals maximum likelihood; with a great deal of data the likelihood dominates the prior and all three agree; it is in the data-poor regime, the three coin flips, that they part ways, and there the Bayesian machinery earns its keep. This dichotomy recurs in every part of the book, and Professor Chin returns to it again and again. We meet it concretely in the very next chapter, where a Beta prior cures the three-heads pathology.

1.5 From predictions to decisions

Inference gives us a distribution; acting in the world requires a single choice. If we have the posterior predictive $p(y \mid \boldsymbol{x})$ for a new input, how should we turn it into a concrete prediction or label? Decision theory answers this by naming a *loss* for each possible mistake and choosing the action that minimizes the expected loss [3, §1.5].

1.5.1 The optimal predictor for squared loss

Take regression first, with the squared loss $\ell(y, t) = (y - t)^2$. We are free to output any function $y(\boldsymbol{x})$ of the input, and we want to minimize the expected loss

$$\mathbb{E}[\ell] = \iint \big(y(\boldsymbol{x}) - t\big)^2\, p(\boldsymbol{x}, t)\, d\boldsymbol{x}\, dt. \tag{1.9}$$

Which function $y(\boldsymbol{x})$ is best? Because the choice of $y(\boldsymbol{x})$ at one input does not constrain its value at another, we can minimize the integrand pointwise: at each fixed $\boldsymbol{x}$, choose the number $y(\boldsymbol{x})$ that minimizes the inner expectation over t. Differentiating the inner integral with respect to the value $y(\boldsymbol{x})$ and setting it to zero,

$$\frac{\partial}{\partial y(\boldsymbol{x})} \int \big(y(\boldsymbol{x}) - t\big)^2 p(\boldsymbol{x}, t)\, dt = \int 2\big(y(\boldsymbol{x}) - t\big) p(\boldsymbol{x}, t)\, dt = 0,$$

gives $y(\boldsymbol{x}) \int p(\boldsymbol{x}, t)\, dt = \int t\, p(\boldsymbol{x}, t)\, dt$. Dividing by $p(\boldsymbol{x}) = \int p(\boldsymbol{x}, t)\, dt$ and recognizing $p(\boldsymbol{x}, t)/p(\boldsymbol{x}) = p(t \mid \boldsymbol{x})$ leaves the answer.

The regression function

The predictor that minimizes expected squared loss is the conditional mean of the target,

$$y^*(\boldsymbol{x}) = \int t\, p(t \mid \boldsymbol{x})\, dt = \mathbb{E}[t \mid \boldsymbol{x}]. \tag{1.10}$$

This single fact justifies a great deal of what follows. It says the ideal output of a regression model is the conditional average of the target, the *regression function* $\mathbb{E}[t \mid \boldsymbol{x}]$. When Chapter 5 fits a curve by least squares, it is estimating this conditional mean; when Chapter 10 trains a network with a squared-error loss, it is doing the same. We can even read the irreducible difficulty of the problem off Eq. (1.9). Adding and subtracting the optimal predictor inside the square and expanding, the cross term integrates to zero and the expected loss splits into two pieces,

$$\mathbb{E}[\ell] = \underbrace{\int \big(y(\boldsymbol{x}) - \mathbb{E}[t \mid \boldsymbol{x}]\big)^2 p(\boldsymbol{x})\, d\boldsymbol{x}}_{\text{how far our model is from the best possible}} + \underbrace{\iint \big(\mathbb{E}[t \mid \boldsymbol{x}] - t\big)^2 p(\boldsymbol{x}, t)\, d\boldsymbol{x}\, dt}_{\text{intrinsic noise, irreducible}}. \tag{1.11}$$

The first term we can shrink by fitting a better model; the second is the variance of the target around its mean and no model can touch it. This decomposition is the seed of the bias-variance analysis of Chapter 5 and the learning theory of Chapter 4.

1.5.2 Classification and the misclassification loss

For classification the target is a label, and the natural loss is whether we get it wrong. Suppose there are K classes and we assign a loss L_{kj} for deciding class j when the truth is class k. The expected loss of a decision rule that maps each $\boldsymbol{x}$ to a class is

$$\mathbb{E}[L] = \sum_j \int_{\mathcal{R}_j} \sum_k L_{kj}\, p(\boldsymbol{x}, \mathcal{C}_k)\, d\boldsymbol{x},$$

where $\mathcal{R}_j$ is the region of input space we choose to assign to class j. Minimizing this means assigning each $\boldsymbol{x}$ to the class j that makes the inner sum $\sum_k L_{kj}\, p(\mathcal{C}_k \mid \boldsymbol{x})$ smallest. In the common case of the *zero-one loss*, where every mistake costs one and a correct answer costs nothing, this rule reduces to choosing the class with the largest posterior probability,

$$\text{decide } j^*(\boldsymbol{x}) = \arg\max_k\, p(\mathcal{C}_k \mid \boldsymbol{x}), \tag{1.12}$$

the *Bayes classifier*, which has the smallest possible error rate of any classifier. The boundaries between the resulting decision regions are exactly the discriminants we will study in Chapter 7. Where the class posteriors are close to each other the decision is unreliable, and one option is a *reject* region in which the model declines to decide. Chapter 18 returns to this idea in a high-stakes setting, where a soft probability is safer than a hard label.

1.6 Information theory

The probabilistic view raises a measurement question. How much uncertainty is in a distribution, and how far apart are two distributions? Information theory, founded by Shannon [35], gives the answers, and they appear throughout the book: as the cross-entropy loss that trains every classifier and network, as the relative entropy that the EM algorithm of Chapter 18 drives to zero, and as the mutual information that scores a feature.

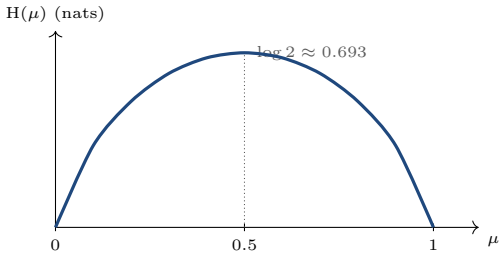


Figure 1.4. The entropy of a Bernoulli random variable as a function of its success probability μ. Certainty at the ends carries no entropy; the fair coin at $\mu = \frac{1}{2}$ carries the most.

1.6.1 Entropy

Definition 1.7 (Entropy). The *entropy* of a discrete distribution p over outcomes is

$$\mathrm{H}[p] = -\sum_{x} p(x) \log p(x),$$

with the convention $0 \log 0 = 0$; for a density the sum becomes the integral $\mathrm{H}[p] = -\int p(x) \log p(x)\, dx$ (the *differential entropy*).

Entropy measures uncertainty. It is largest when the distribution is spread out and smallest when it is concentrated. The cleanest case is a single binary outcome with probability μ of success, the Bernoulli distribution, whose entropy

$$\mathrm{H}(\mu) = -\mu \log \mu - (1-\mu) \log(1-\mu)$$

is plotted in Fig. 1.4. It is zero at $\mu = 0$ and $\mu = 1$, where the outcome is certain, and largest at $\mu = \frac{1}{2}$, where a fair coin gives the most uncertain outcome of all. When the logarithm is taken base two the unit is the *bit*, and the entropy is the average number of yes-or-no questions needed to pin down the outcome.

1.6.2 Relative entropy and the cross-entropy loss

To compare two distributions we use the relative entropy, or Kullback-Leibler divergence [24].

Definition 1.8 (Kullback-Leibler divergence). The *relative entropy* from q to p is

$$\mathrm{KL}(p \,\|\, q) = \int p(x) \log \frac{p(x)}{q(x)}\, dx = -\int p(x) \log \frac{q(x)}{p(x)}\, dx.$$

The divergence is the price, in extra bits, of describing data from p using a code built for q. Its defining property is that it is never negative, and is zero only when the two distributions coincide.

Theorem 1.9 (Gibbs' inequality). For any distributions p and q, $\mathrm{KL}(p \,\|\, q) \geq 0$, with equality if and only if $p = q$.

Proof. The function $-\log$ is strictly convex. Jensen's inequality states that for a convex function f and a random variable Z, $\mathbb{E}[f(Z)] \geq f(\mathbb{E}[Z])$. Apply it with $f = -\log$ and $Z = q(x)/p(x)$, where x is drawn from p:

$$\mathrm{KL}(p \,\|\, q) = \mathbb{E}_p\left[-\log \frac{q(x)}{p(x)}\right] \geq -\log \mathbb{E}_p\left[\frac{q(x)}{p(x)}\right] = -\log \int p(x) \frac{q(x)}{p(x)}\, dx = -\log \int q(x)\, dx = -\log 1 = 0.$$

Because $-\log$ is strictly convex, Jensen's inequality is an equality only when $q(x)/p(x)$ is constant, and since q integrates to one that constant is one, so $p = q$. ∎

We prove Jensen's inequality and use this same argument again in Chapter 18, where the gap $\mathrm{KL}(q \,\|\, p)$ between an approximating distribution and a posterior is exactly what the E step of the EM algorithm drives to zero. Relative entropy also explains the loss function that trains nearly every classifier in this book. Expanding the definition,

$$\mathrm{KL}(p \,\|\, q) = \underbrace{-\int p \log q}_{\text{cross-entropy } \mathrm{H}(p,q)} - \underbrace{\left(-\int p \log p\right)}_{\mathrm{H}[p]},$$

so the *cross-entropy* $\mathrm{H}(p,q) = -\int p \log q$ equals the entropy of p plus the divergence from q to p. When p is the true distribution of the data, $\mathrm{H}[p]$ is fixed, so minimizing the cross-entropy of a model q is the same as minimizing $\mathrm{KL}(p \,\|\, q)$, which is the same as making q match p. Fitting a model by minimizing cross-entropy and fitting it by maximum likelihood are, it turns out, the same procedure, a point we return to in Chapter 8.

1.6.3 Mutual information

Finally, to measure how much two variables tell us about each other, we ask how far their joint distribution is from the product of its marginals, which is what the joint would be if the variables were independent.

Definition 1.10 (Mutual information). The *mutual information* between $\boldsymbol{x}$ and y is the relative entropy between their joint distribution and the product of their marginals,

$$I[\boldsymbol{x}, y] = \mathrm{KL}(p(\boldsymbol{x}, y) \,\|\, p(\boldsymbol{x})\, p(y)) = \iint p(\boldsymbol{x}, y) \log \frac{p(\boldsymbol{x}, y)}{p(\boldsymbol{x})\, p(y)} \, d\boldsymbol{x}\, dy. \tag{1.13}$$

By Theorem 1.9 the mutual information is never negative, and it is zero exactly when $\boldsymbol{x}$ and y are independent, for then $p(\boldsymbol{x}, y) = p(\boldsymbol{x})p(y)$ and the logarithm is $\log 1 = 0$ everywhere. It can also be written $I[\boldsymbol{x}, y] = \mathrm{H}[y] - \mathrm{H}[y \mid \boldsymbol{x}]$, the amount by which knowing $\boldsymbol{x}$ reduces the uncertainty in y. This makes it a natural score for *feature selection*: a feature with high mutual information with the target carries information about it, and one with zero mutual information is, by itself, useless. A simple greedy strategy starts with no features and adds, one at a time, the feature with the highest mutual information with the target, retraining after each addition.

Example 1.11 (Mutual information of a small joint). Let x and y each be binary with the joint distribution

$$p(x, y): \quad p(0,0) = 0.4, \quad p(0,1) = 0.1, \quad p(1,0) = 0.1, \quad p(1,1) = 0.4.$$

The marginals are $p(x{=}0) = p(x{=}1) = 0.5$ and likewise for y, so the product of marginals is 0.25 in every cell. The mutual information, in nats, is

$$I[x, y] = 2\left(0.4 \log \frac{0.4}{0.25}\right) + 2\left(0.1 \log \frac{0.1}{0.25}\right) = 0.8 \log 1.6 + 0.2 \log 0.4 \approx 0.8(0.470) + 0.2(-0.916) \approx 0.193.$$

The value is positive, as it must be, reflecting the fact that x and y agree more often than chance would predict, so each carries information about the other. ◊

Notes and References

The curve-fitting example, the decision-theory development, and the information-theory summary of this chapter follow Chapter 1 of Bishop [3], which remains the single best entry point to the subject. The working definition of learning is from Mitchell [31]. The Dartmouth origin of the term "artificial intelligence" is documented in the reprinted 1955 proposal [28]. Entropy and the foundations of information theory are due to Shannon [35], and the relative entropy to Kullback and Leibler [24]. The historical landmarks in Fig. 1.1 are taken up in detail later: the perceptron [32] and backpropagation [33] in Part IV, the support vector machine [7] in Chapter 15, and the convolutional and attention architectures [23, 25, 42] in Chapters 12 to 13. The deep-learning perspective on the same history is the subject of Chapter 1 of Goodfellow et al. [13].

Exercises

Exercise 1.1. A degree-M polynomial is fit by least squares to m data points. Explain why the training error cannot increase as M grows, and why the model can achieve zero training error once $M = m - 1$. What goes wrong with the fit at that point, and why does collecting more data postpone the problem?

Solution. Increasing M enlarges the family of polynomials: every degree-M polynomial is also a degree-$(M{+}1)$ polynomial with leading coefficient zero. Minimizing over a larger set cannot yield a larger minimum, so the training error is nonincreasing in M. A degree-$(m-1)$ polynomial has m free coefficients, enough to satisfy the m interpolation

equations $y(x^{(i)}) = t^{(i)}$ exactly (the system is a square Vandermonde system, nonsingular when the $x^{(i)}$ are distinct), so the training error is zero. The fit is then an interpolating polynomial that passes through every noisy point and oscillates between them, with huge coefficients; its generalization error is large. More data raises the number of interpolation constraints, so a polynomial of the same degree can no longer bend to meet every point and must instead capture the trend they share. □

Exercise 1.2. A coin lands heads on all three of three tosses. Model the tosses as independent Bernoulli(μ) trials. (a) Find the maximum likelihood estimate of μ. (b) With a prior $p(\mu) \propto \mu^{a-1}(1-\mu)^{b-1}$, derive the MAP estimate, and evaluate it for the flat prior $a = b = 1$ and for the mild prior $a = b = 2$. (c) What does each estimate predict for the probability that the next toss is tails, and which is more reasonable?

Solution. (a) The log-likelihood of three heads is $\log \mu^3 = 3 \log \mu$, increasing in μ, so $\mu^*_{\mathrm{ML}} = 1$. (b) With $m = 3$ heads and $l = 0$ tails the log-posterior is, up to a constant, $(m+a-1)\log\mu + (l+b-1)\log(1-\mu) = (a+2)\log\mu + (b-1)\log(1-\mu)$. Setting its derivative to zero, $(a+2)/\mu = (b-1)/(1-\mu)$, gives

$$\mu^*_{\mathrm{MAP}} = \frac{a+2}{a+b+1}.$$

For the flat prior $a = b = 1$ this is $3/3 = 1$: a uniform prior adds no information, the posterior mode sits at the boundary, and MAP coincides with maximum likelihood. For the mild prior $a = b = 2$, which gently favors a fair-ish coin, it is $4/5 = 0.8$, pulled in from the boundary. (c) Maximum likelihood, and equally the flat-prior MAP, predict probability $1 - 1 = 0$ of tails, declaring tails impossible on the strength of three tosses. The $a = b = 2$ MAP predicts $1 - 0.8 = 0.2$, and the full Bayesian posterior mean $(a+m)/(a+b+m+l) = 5/7 \approx 0.71$ gives a tails probability near 0.29. Either Bayesian summary is the more reasonable, because the prior encodes that a coin can land both ways and three tosses is weak evidence. The subtlety worth keeping is that the *flat* prior does not fix the mode; it takes a prior that actively prefers interior values, or the posterior mean rather than the mode, to resolve the pathology. Chapter 2 develops this Beta-Bernoulli calculation in full and shows the prior acts like extra imagined data. □

Exercise 1.3. Show directly that among all functions $y(\boldsymbol{x})$, the one minimizing the expected squared loss $\mathbb{E}[(y(\boldsymbol{x}) - t)^2]$ is the conditional mean $\mathbb{E}[t \mid \boldsymbol{x}]$, by writing $y(\boldsymbol{x}) - t = (y(\boldsymbol{x}) - \mathbb{E}[t \mid \boldsymbol{x}]) + (\mathbb{E}[t \mid \boldsymbol{x}] - t)$ and expanding.

Solution. Square the displayed sum and take the expectation conditioned on $\boldsymbol{x}$. The cross term is $2(y(\boldsymbol{x}) - \mathbb{E}[t \mid \boldsymbol{x}])\,\mathbb{E}[\,\mathbb{E}[t \mid \boldsymbol{x}] - t \mid \boldsymbol{x}]$, and the inner expectation is $\mathbb{E}[t \mid \boldsymbol{x}] - \mathbb{E}[t \mid \boldsymbol{x}] = 0$, so the cross term vanishes. What remains is $\mathbb{E}[(y(\boldsymbol{x}) - t)^2 \mid \boldsymbol{x}] = (y(\boldsymbol{x}) - \mathbb{E}[t \mid \boldsymbol{x}])^2 + \mathrm{Var}(t \mid \boldsymbol{x})$. The second term does not depend on our choice of $y(\boldsymbol{x})$, and the first is a square, minimized at zero by taking $y(\boldsymbol{x}) = \mathbb{E}[t \mid \boldsymbol{x}]$. Averaging over $\boldsymbol{x}$ gives Eq. (1.11) and confirms Eq. (1.10). □

Exercise 1.4. (a) Compute the entropy in nats of a fair four-sided die and of a biased one with probabilities $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8})$. (b) Use Theorem 1.9 to show that among all distributions on K outcomes, the uniform distribution has the largest entropy.

Solution. (a) Fair: $\mathrm{H} = -\sum \frac{1}{4}\log\frac{1}{4} = \log 4 \approx 1.386$ nats. Biased: $\mathrm{H} = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{4}\log\frac{1}{4} - 2\cdot\frac{1}{8}\log\frac{1}{8} = \frac{1}{2}\log 2 + \frac{1}{4}\log 4 + \frac{1}{4}\log 8 = (\frac{1}{2} + \frac{1}{2} + \frac{3}{4})\log 2 = \frac{7}{4}\log 2 \approx 1.213$ nats, less than the fair die, as expected. (b) Let u be uniform, $u(x) = 1/K$, and p arbitrary. Then $\mathrm{KL}(p \,\|\, u) = \sum_x p(x) \log \frac{p(x)}{1/K} = \log K - \mathrm{H}[p] \geq 0$ by Theorem 1.9, so $\mathrm{H}[p] \leq \log K = \mathrm{H}[u]$, with equality only when $p = u$. □

Exercise 1.5. Two binary features x_1, x_2 and a binary label y have the property that $x_1 = y$ always, while x_2 is an independent fair coin. Without computing, state the mutual information $I[x_1, y]$ and $I[x_2, y]$ qualitatively, and say which feature a mutual-information criterion selects first. Then verify $I[x_2, y] = 0$ from the definition.

Solution. Because x_1 determines y exactly, knowing x_1 removes all uncertainty in y, so $I[x_1, y] = \mathrm{H}[y]$, the largest value possible. Because x_2 is independent of y, it carries no information, so $I[x_2, y] = 0$. A mutual-information criterion selects x_1 first. To verify the second claim, independence gives $p(x_2, y) = p(x_2)p(y)$, so the logarithm in Eq. (1.13) is $\log 1 = 0$ in every cell and the sum is zero. □

CHAPTER 2

Probability Distributions

A data vector is empty until you fix the basis, and a random vector is empty until you fix its mean and covariance.

after Professor Peter Chin, ENGS 106

Chapter 1 set out three ways to fit a model, maximum likelihood, the maximum a posteriori estimate, and the full Bayesian posterior, but it left the models themselves abstract. This chapter supplies the building blocks: the handful of probability distributions from which almost every model in the book is assembled. We meet them in families. The binary variables (Bernoulli, binomial) and their Beta prior come first, then the multi-way variables (categorical, multinomial) and their Dirichlet prior, and finally the continuous workhorse of the whole subject, the Gaussian, in one dimension and in many. For each family we do the same three things: write the distribution, find its maximum likelihood estimate, and pair it with a conjugate prior that makes Bayesian updating into arithmetic. The thread that runs through all of it is conjugacy, the property that keeps the posterior in the same family as the prior, and it is the reason these particular distributions, and not others, are the ones the course returns to again and again.

Roadmap

Suggested reading: Bishop [3], Chapter 2 (Sections 2.1–2.3); Deisenroth et al. [9], Sections 6.4–6.6 for the same material from the mathematics side.

We cover: the rules of probability and the moments (Section 2.1); the Bernoulli and binomial and their maximum likelihood estimate (Sections 2.2 to 2.3); Bayes' theorem, the Beta prior, conjugacy, and the predictive distribution that resolves the coin-flip pathology of Chapter 1 (Sections 2.4 to 2.5); the categorical, multinomial, and Dirichlet (Section 2.6); the univariate Gaussian and its normalization derived from scratch (Section 2.7); the multivariate Gaussian, its geometry, its maximum likelihood estimate, and the bias of the variance estimate (Section 2.8); conditional and marginal Gaussians (Section 2.9); and covariance (Section 2.10).

2.1 The rules of probability

All of probability rests on two rules. For random variables X, Y with a joint distribution $p(X, Y)$, the *sum rule* recovers one variable by summing (integrating) out the other, and the *product rule* factors a joint into a conditional times a marginal:

$$p(X) = \sum_y p(X, Y{=}y) \quad \text{(sum rule)}, \qquad p(X, Y) = p(X \mid Y)\, p(Y) \quad \text{(product rule)}. \tag{2.1}$$

For continuous variables the sum becomes an integral. Bayes' theorem, which drives the whole chapter, is nothing more than the product rule written both ways and rearranged (Section 2.4). Two variables are *independent* if $p(X, Y) = p(X)p(Y)$, equivalently $p(X \mid Y) = p(X)$: knowing one says nothing about the other.

The *expectation* of a random variable is its probability-weighted average, and the *variance* measures its spread about that average:

$$\mathbb{E}[X] = \sum_x x\, p(x), \qquad \mathrm{Var}(X) = \mathbb{E}\big[(X - \mathbb{E}[X])^2\big] = \mathbb{E}[X^2] - \mathbb{E}[X]^2, \tag{2.2}$$

again with an integral for continuous variables. The second form of the variance is worth deriving once, because the same expand-and-cancel move recurs throughout the chapter. Writing $\mu = \mathbb{E}[X]$ and using linearity of expectation,

$$\mathrm{Var}(X) = \mathbb{E}[(X-\mu)^2] = \mathbb{E}[X^2 - 2\mu X + \mu^2] = \mathbb{E}[X^2] - 2\mu\,\mathbb{E}[X] + \mu^2 = \mathbb{E}[X^2] - 2\mu^2 + \mu^2 = \mathbb{E}[X^2] - \mu^2. \tag{2.3}$$

Expectation is *linear*, $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$, whether or not X and Y are independent. Variance is not linear; it adds across independent variables, and the cross term that makes the difference is the covariance of Section 2.10.

One more rule is needed for continuous variables: when we transform a variable, its density does not simply carry along, because a density is a probability per unit length and the transformation stretches that length. If $y = g(x)$ is a smooth invertible map, then matching probability mass on corresponding intervals, $p_y(y)\,|dy| = p_x(x)\,|dx|$, gives the *change-of-variables* formula

$$p_y(y) = p_x(x)\left|\frac{dx}{dy}\right| = p_x\big(g^{-1}(y)\big)\left|\frac{d}{dy} g^{-1}(y)\right|. \tag{2.4}$$

The Jacobian factor $|dx/dy|$ is what distinguishes a density from a plain function, and forgetting it is one of the most common errors in the subject.

2.2 Binary variables: the Bernoulli and binomial

The simplest random variable is the outcome of a single coin flip.

Definition 2.1 (Bernoulli distribution). A random variable $x \in \{0, 1\}$ is *Bernoulli* with parameter $\mu \in [0, 1]$, written $x \sim \mathrm{Bern}(\mu)$, if $\mathbb{P}(x{=}1) = \mu$ and $\mathbb{P}(x{=}0) = 1 - \mu$. Both cases are captured in one expression,

$$\mathrm{Bern}(x \mid \mu) = \mu^x (1-\mu)^{1-x}, \qquad x \in \{0, 1\}, \tag{2.5}$$

since the exponents select the right factor: at $x{=}1$ it reads μ, at $x{=}0$ it reads $1 - \mu$.

Its mean and variance follow from Eq. (2.2). Because x takes only the values 0 and 1, we have $x^2 = x$, so $\mathbb{E}[x^2] = \mathbb{E}[x]$, and

$$\mathbb{E}[x] = 0 \cdot (1-\mu) + 1 \cdot \mu = \mu, \qquad \mathrm{Var}(x) = \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \mu - \mu^2 = \mu(1-\mu). \tag{2.6}$$

The variance is largest at $\mu = \frac{1}{2}$, the most uncertain coin, and vanishes at $\mu \in \{0, 1\}$, a certain outcome. This is the same shape as the binary entropy of Fig. 1.4, for the same reason: both measure uncertainty.

Now flip the same coin N times independently and count the heads. The count is binomial.

Definition 2.2 (Binomial distribution). The number m of ones in N independent $\mathrm{Bern}(\mu)$ trials has the *binomial* distribution

$$\mathrm{Bin}(m \mid N, \mu) = \binom{N}{m} \mu^m (1-\mu)^{N-m}, \qquad \binom{N}{m} = \frac{N!}{m!\,(N-m)!}. \tag{2.7}$$

The factor $\mu^m(1-\mu)^{N-m}$ is the probability of one particular sequence with m heads, and the binomial coefficient counts how many such sequences there are.

Its moments come for free from the Bernoulli's, because a binomial count is a sum of N independent Bernoulli variables, $m = \sum_{i=1}^{N} x_i$. Linearity of expectation gives the mean immediately, and independence lets the variances add:

$$\mathbb{E}[m] = \sum_{i=1}^{N} \mathbb{E}[x_i] = N\mu, \qquad \mathrm{Var}(m) = \sum_{i=1}^{N} \mathrm{Var}(x_i) = N\mu(1-\mu). \tag{2.8}$$

Both reduce to Eq. (2.6) when $N = 1$, as they must.

2.3 Maximum likelihood for the Bernoulli

We rarely know μ; we see data and must infer it. Suppose we observe N independent flips $\mathcal{D} = \{x_1, \ldots, x_N\}$, $x_i \overset{\text{iid}}{\sim} \mathrm{Bern}(\mu)$, and write $m = \sum_i x_i$ for the number of heads. Because the flips are independent, the probability of the whole data set, viewed as a function of μ, is the product of the individual masses. This is the *likelihood*:

$$p(\mathcal{D} \mid \mu) = \prod_{i=1}^{N} \mu^{x_i}(1-\mu)^{1-x_i} = \mu^{\sum_i x_i}(1-\mu)^{\sum_i (1-x_i)} = \mu^m (1-\mu)^{N-m}. \tag{2.9}$$

Intuition. The one expression $p(\mathcal{D} \mid \mu)$ is read two ways. Fix μ and let the data vary, and it is a probability that sums to one over all data sets. Fix the observed data and let μ vary, and it is a likelihood, a curve over $[0, 1]$ that does not integrate to one. Maximum likelihood asks the natural question, which coin bias would have made what I actually saw least surprising, and answers by climbing that curve to its peak.

Proposition 2.3 (Bernoulli MLE). The maximum likelihood estimate of the Bernoulli parameter is the sample mean, $\hat{\mu}_{\mathrm{ML}} = m/N = \frac{1}{N}\sum_i x_i$.

Proof. Maximizing the likelihood is the same as maximizing its logarithm, since the logarithm is increasing, and the logarithm turns the product into a sum that is easy to differentiate. The *log-likelihood* is

$$\ell(\mu) = \log p(\mathcal{D} \mid \mu) = m \log \mu + (N - m)\log(1-\mu).$$

Differentiate with respect to μ and set the derivative to zero,

$$\ell'(\mu) = \frac{m}{\mu} - \frac{N-m}{1-\mu} = 0 \implies m(1-\mu) = (N-m)\mu \implies m - m\mu = N\mu - m\mu \implies m = N\mu,$$

so $\hat{\mu} = m/N$. The second derivative $\ell''(\mu) = -m/\mu^2 - (N-m)/(1-\mu)^2 < 0$ is negative throughout $(0, 1)$, so ℓ is concave and the stationary point is the global maximum. ∎

The estimate is the obvious one, the fraction of flips that came up heads, but it is brittle. With three heads in three flips it returns $\hat{\mu} = 1$, declaring tails impossible on the strength of three flips, exactly the pathology flagged in Chapter 1. It also reports a single number with no measure of confidence. Both shortcomings are what the Bayesian treatment repairs, and the repair is built on Bayes' theorem.

2.4 Bayes' theorem and the Beta prior

The product rule Eq. (2.1) factors a joint two ways, and equating them and dividing gives Bayes' theorem.

Theorem 2.4 (Bayes' theorem). For random variables with $p(Y) > 0$,

$$p(X \mid Y) = \frac{p(Y \mid X)\, p(X)}{p(Y)}. \tag{2.10}$$

When X is a parameter θ and Y is the data $\mathcal{D}$, the four pieces have the names introduced in Chapter 1: posterior, likelihood, prior, and evidence. Before turning Bayes' theorem on the coin, it pays to see it on events, where the arithmetic is most transparent and the lesson most useful.

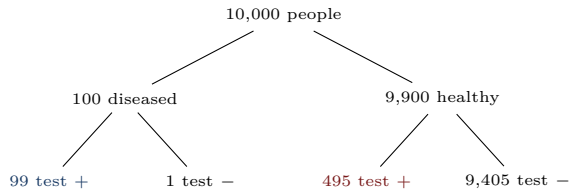


Figure 2.1. Splitting 10,000 people by disease status and then by test result makes the base-rate effect concrete. Because only 1% are sick, the few true positives are buried under false positives drawn from the healthy majority, and a positive result means only a one-in-six chance of disease.

Example 2.5 (Disease testing and the base rate). A test for a disease has sensitivity $\mathbb{P}(+ \mid D) = 0.99$ and false-positive rate $\mathbb{P}(+ \mid \neg D) = 0.05$, and the disease has prevalence $\mathbb{P}(D) = 0.01$. A patient tests positive. What is the probability they have the disease? By Bayes' theorem, with the denominator expanded by the sum rule,

$$\mathbb{P}(D \mid +) = \frac{\mathbb{P}(+ \mid D)\mathbb{P}(D)}{\mathbb{P}(+ \mid D)\mathbb{P}(D) + \mathbb{P}(+ \mid \neg D)\mathbb{P}(\neg D)} = \frac{(0.99)(0.01)}{(0.99)(0.01) + (0.05)(0.99)} = \frac{0.0099}{0.0594} = \frac{1}{6} \approx 16.7\%.$$

A highly accurate test yields only a one-in-six chance of disease, because the disease is rare to begin with. The reasoning is clearest in natural frequencies (Fig. 2.1): of 10,000 people, the 99 true positives are outnumbered five to one by the 495 false positives drawn from the large healthy majority. Ignoring the small prior $\mathbb{P}(D)$, the *base rate*, is the classic error this calculation guards against, and it returns in Chapter 8 as the heart of the naive Bayes classifier. ◊

To be Bayesian about the coin we need a prior on $\mu \in [0, 1]$. The natural choice is the distribution whose density has the same algebraic shape as the Bernoulli likelihood, the *Beta distribution*.

Definition 2.6 (Beta distribution). For shape parameters $a, b > 0$, the density of $\mu \sim \mathrm{Beta}(a, b)$ on $[0, 1]$ is

$$\mathrm{Beta}(\mu \mid a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\,\Gamma(b)}\,\mu^{a-1}(1-\mu)^{b-1}, \tag{2.11}$$

where Γ is the gamma function ($\Gamma(k) = (k-1)!$ for integer k), and the leading constant makes the density integrate to one. Its mean is $\mathbb{E}[\mu] = a/(a+b)$, and its mode, for $a, b > 1$, is $(a-1)/(a+b-2)$.

The exponents $a-1$ and $b-1$ make the Beta density a rescaled copy of the Bernoulli likelihood Eq. (2.9), and that shared form is exactly what keeps the prior and posterior in the same family.

Definition 2.7 (Conjugate prior). A prior is *conjugate* to a likelihood if the resulting posterior belongs to the same family as the prior. Conjugacy turns Bayesian updating into bookkeeping on the parameters.

2.5 Bayesian inference for the Bernoulli

Put a $\mathrm{Beta}(a, b)$ prior on μ and observe m heads in N flips, that is m heads and $l = N - m$ tails. Multiply the likelihood Eq. (2.9) by the prior Eq. (2.11), keeping only the factors that depend on μ (everything else is absorbed into the normalizing constant):

$$p(\mu \mid \mathcal{D}) \;\propto\; \underbrace{\mu^{m}(1-\mu)^{l}}_{\text{likelihood}} \cdot \underbrace{\mu^{a-1}(1-\mu)^{b-1}}_{\text{prior}} = \mu^{(m+a)-1}(1-\mu)^{(l+b)-1}. \tag{2.12}$$

The right-hand side is the kernel of another Beta density. We do not even need to recompute the normalizer; we recognize the form and read off the parameters.

The Beta–Bernoulli update

$$\mu \sim \mathrm{Beta}(a, b) \quad \text{and} \quad m \text{ heads},\ l \text{ tails} \qquad \Longrightarrow \qquad \mu \mid \mathcal{D} \sim \mathrm{Beta}(a + m,\ b + l).$$

Updating simply adds the observed heads to a and the observed tails to b. The prior parameters therefore act as *pseudo-counts*: a imaginary prior heads and b imaginary prior tails, mixed in with the real data. This is the precise sense in which a prior "acts like extra data," promised in Chapter 1.

2.5.1 Sequential updating

Because conjugacy keeps the posterior in the Beta family, we can absorb data one point at a time, and yesterday's posterior becomes today's prior. Start from $\mathrm{Beta}(a, b)$; see one head, move to $\mathrm{Beta}(a + 1, b)$; see one tail, move to $\mathrm{Beta}(a+1, b+1)$; and so on. Processing the N flips one at a time, or all at once with counts (m, l), lands on the identical posterior $\mathrm{Beta}(a + m, b + l)$, because addition does not care about the order of the summands. This consistency, that the answer does not depend on whether data arrives in a batch or in a stream, is a hallmark of Bayesian inference and a practical convenience for online learning.

2.5.2 The posterior mean and the predictive distribution

From the Beta mean, the posterior mean of μ is

$$\mathbb{E}[\mu \mid \mathcal{D}] = \frac{a + m}{a + b + N}, \tag{2.13}$$

the maximum likelihood fraction m/N smoothed by the pseudo-counts. Two limits confirm the picture. With no data ($N = 0$) it returns the prior mean $a/(a + b)$. As $N \to \infty$ the pseudo-counts a, b become negligible beside m, l, and the posterior mean tends to m/N, the MLE, while the posterior variance shrinks to zero. With enough data the prior washes out and the Bayesian and frequentist answers coincide, the convergence anticipated in Table 1.1.

What we usually want is not μ itself but a prediction about the next flip. The fully Bayesian prediction averages the model over the posterior, the predictive distribution of Eq. (1.8). For the Bernoulli this average is exactly the posterior mean, and it is worth seeing why in full:

$$\mathbb{P}(x_{N+1}{=}1 \mid \mathcal{D}) = \int_0^1 \mathbb{P}(x_{N+1}{=}1 \mid \mu)\, p(\mu \mid \mathcal{D})\, d\mu = \int_0^1 \mu\, p(\mu \mid \mathcal{D})\, d\mu = \mathbb{E}[\mu \mid \mathcal{D}] = \frac{a + m}{a + b + N}. \tag{2.14}$$

The first step is the definition of the predictive distribution; the second uses $\mathbb{P}(x{=}1 \mid \mu) = \mu$; the third is the definition of the posterior mean; the last is Eq. (2.13). This is *Laplace's rule of succession*, and it is the clean resolution of the three-heads pathology. With a uniform prior $a = b = 1$ and three heads ($m = 3$, $N = 3$),

$$\mathbb{P}(x_4{=}1 \mid \mathcal{D}) = \frac{1 + 3}{1 + 1 + 3} = \frac{4}{5},$$

so the predicted chance of a fourth head is $4/5$, short of the maximum likelihood estimate's 1. The model now allows that tails, though unseen, remains possible, precisely because the prior contributed one imaginary tail.

Example 2.8 (Posterior, posterior mean, and MAP). Start from a mild prior $\mathrm{Beta}(2, 2)$, centered at $\mu = \frac{1}{2}$ and gently favoring a fair coin, and observe $m = 8$ heads in $N = 10$ flips ($l = 2$). The posterior is $\mathrm{Beta}(2 + 8,\ 2 + 2) = \mathrm{Beta}(10, 4)$. Three summaries of μ compare as follows:

$$\hat{\mu}_{\mathrm{ML}} = \frac{m}{N} = 0.80, \qquad \mathbb{E}[\mu \mid \mathcal{D}] = \frac{10}{14} = \frac{5}{7} \approx 0.714, \qquad \hat{\mu}_{\mathrm{MAP}} = \frac{10 - 1}{14 - 2} = \frac{9}{12} = 0.75,$$

where the MAP estimate is the posterior mode $(a' - 1)/(a' + b' - 2)$ for the posterior $\mathrm{Beta}(a', b')$. Both Bayesian summaries sit between the data (0.80) and the prior mean (0.5): the prior, worth four pseudo-flips, pulls the estimate toward fairness, an effect that fades as the data grows (Fig. 2.2). ◊

Remark 2.9 *(MAP estimation is regularization).* Taking the logarithm of Bayes' theorem turns the MAP estimate into a sum of two terms, $\hat{\theta}_{\mathrm{MAP}} = \arg\max_\theta \big[\log p(\mathcal{D} \mid \theta) + \log p(\theta)\big]$: a data-fit term and a prior term. This is exactly the shape of a regularized objective, a loss plus a penalty. We will see in Chapter 5 that a Gaussian prior on regression

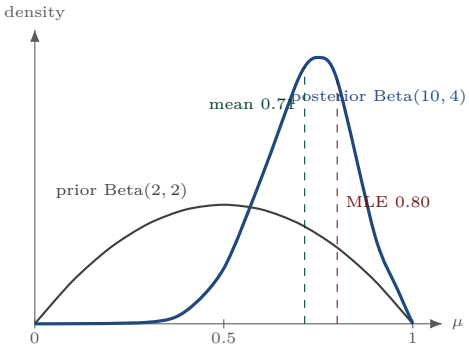


Figure 2.2. The gentle prior Beta(2, 2) becomes the sharper posterior Beta(10, 4) after 8 heads in 10 flips. The data pulls belief toward 0.8, but the prior holds the posterior mean back to 0.71; more data would sharpen the posterior and erase the gap.

weights produces precisely the squared-norm penalty of ridge regression, so ridge regression *is* MAP estimation under a Gaussian prior. The Beta prior plays the same regularizing role here, keeping the estimate sane when the data is thin.

2.6 Multiple outcomes: the multinomial and Dirichlet

A die has more than two faces, so we generalize from two outcomes to K.

Definition 2.10 (Categorical and multinomial distributions). A *categorical* variable takes one of K values with probabilities $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)$, $\mu_k \geq 0$, $\sum_k \mu_k = 1$. Encoding the outcome as a one-hot vector $\boldsymbol{x} \in \{0,1\}^K$ (a single 1 in the chosen position) gives $\mathrm{Cat}(\boldsymbol{x} \mid \boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k}$. Repeating the draw N times and recording the counts m_k gives the *multinomial*

$$\mathrm{Mult}(\boldsymbol{m} \mid \boldsymbol{\mu}, N) = \binom{N}{m_1, \dots, m_K} \prod_{k=1}^{K} \mu_k^{m_k}, \qquad \binom{N}{m_1, \dots, m_K} = \frac{N!}{m_1! \cdots m_K!}, \tag{2.15}$$

the multinomial coefficient counting the orderings that produce the same histogram. With $K = 2$ this is the binomial; with $N = 1$ it is the categorical.

The parameter $\boldsymbol{\mu}$ lives on the *probability simplex*, the flat $(K-1)$-dimensional triangle $\{\boldsymbol{\mu} : \mu_k \geq 0, \ \sum_k \mu_k = 1\}$, so estimating it is a constrained problem. The constraint is what makes the maximum likelihood derivation more interesting than the Bernoulli's.

Proposition 2.11 (Multinomial MLE). The maximum likelihood estimate of the category probabilities is the empirical frequency, $\hat{\mu}_k = m_k / N$.

Proof. Dropping the constant multinomial coefficient, the log-likelihood is $\ell(\boldsymbol{\mu}) = \sum_k m_k \log \mu_k$, to be maximized subject to $\sum_k \mu_k = 1$. We cannot maximize each μ_k freely, because raising one lowers the others, so we enforce the constraint with a Lagrange multiplier λ (the method is developed in full in Appendix C). Form the Lagrangian

$$\mathcal{L}(\boldsymbol{\mu}, \lambda) = \sum_{k=1}^{K} m_k \log \mu_k + \lambda \Big(1 - \sum_{k=1}^{K} \mu_k \Big).$$

Setting the partial derivative in μ_k to zero,

$$\frac{\partial \mathcal{L}}{\partial \mu_k} = \frac{m_k}{\mu_k} - \lambda = 0 \implies \mu_k = \frac{m_k}{\lambda}.$$

The multiplier is fixed by the constraint: summing $\mu_k = m_k / \lambda$ over k and using $\sum_k \mu_k = 1$ and $\sum_k m_k = N$ gives $1 = N/\lambda$, so $\lambda = N$ and $\hat{\mu}_k = m_k / N$. ∎

The estimate is the fraction of trials landing in each category, the multi-class version of $\hat{\mu} = m/N$. Its weakness is the same too: an unseen category ($m_k = 0$) is assigned probability zero, an overconfident verdict the Bayesian update repairs. The conjugate prior is the Dirichlet, the multivariate Beta.

Definition 2.12 (Dirichlet distribution). For concentration parameters $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_K)$ with $\alpha_k > 0$ and $\alpha_0 = \sum_k \alpha_k$, the density on the simplex is

$$\mathrm{Dir}(\boldsymbol{\mu} \mid \boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\prod_{k=1}^{K} \Gamma(\alpha_k)} \prod_{k=1}^{K} \mu_k^{\alpha_k - 1}. \tag{2.16}$$

Multiplying the multinomial likelihood by the Dirichlet prior adds the exponents, exactly as in the Beta-Bernoulli case,

$$p(\boldsymbol{\mu} \mid \boldsymbol{m}) \;\propto\; \Big(\prod_k \mu_k^{m_k}\Big)\Big(\prod_k \mu_k^{\alpha_k - 1}\Big) = \prod_k \mu_k^{(\alpha_k + m_k) - 1},$$

so the posterior is again Dirichlet, $\boldsymbol{\mu} \mid \boldsymbol{m} \sim \mathrm{Dir}(\boldsymbol{\alpha} + \boldsymbol{m})$, with mean

$$\mathbb{E}[\mu_k \mid \boldsymbol{m}] = \frac{\alpha_k + m_k}{\alpha_0 + N}, \tag{2.17}$$

the MLE smoothed by the pseudo-counts α_k. Because $\alpha_k > 0$, no category is ever assigned probability zero, which cures the overconfidence and, in the language of Chapter 8, is precisely add-α (Laplace) smoothing.

2.7 The univariate Gaussian

For continuous data the central model is the Gaussian. We meet it in one dimension first, and we derive its normalizing constant from scratch, because the calculation is a small classic and the user of a distribution should know where its constants come from.

Definition 2.13 (Univariate Gaussian). A scalar x is *Gaussian* (normal) with mean μ and variance σ^2, written $x \sim \mathcal{N}(\mu, \sigma^2)$, if it has density

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\!\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big). \tag{2.18}$$

The reciprocal variance $\beta = 1/\sigma^2$ is the *precision*, used heavily in the regression chapters.

2.7.1 Where the normalizing constant comes from

Why the factor $1/\sqrt{2\pi\sigma^2}$? It is whatever makes the density integrate to one. Substituting $z = (x-\mu)/\sigma$ reduces the question to evaluating the *Gaussian integral* $I = \int_{-\infty}^{\infty} e^{-z^2/2}\,dz$. The trick, due to Poisson, is to square it and pass to polar coordinates, turning a product of two one-dimensional integrals into one two-dimensional integral:

$$I^2 = \Big(\int_{-\infty}^{\infty} e^{-x^2/2}\,dx\Big)\Big(\int_{-\infty}^{\infty} e^{-y^2/2}\,dy\Big) = \iint_{\mathbb{R}^2} e^{-(x^2+y^2)/2}\,dx\,dy.$$

In polar coordinates $x = r\cos\theta$, $y = r\sin\theta$, the area element is $dx\,dy = r\,dr\,d\theta$ and $x^2 + y^2 = r^2$, so

$$I^2 = \int_0^{2\pi}\!\!\int_0^{\infty} e^{-r^2/2}\, r\,dr\,d\theta = \int_0^{2\pi} d\theta \int_0^{\infty} r\,e^{-r^2/2}\,dr = 2\pi \cdot \Big[-e^{-r^2/2}\Big]_0^{\infty} = 2\pi \cdot (0 - (-1)) = 2\pi.$$

The radial integral is elementary because the extra factor of r is exactly the derivative of $-r^2/2$ inside the exponential. Hence $I = \sqrt{2\pi}$, and undoing the substitution $z = (x-\mu)/\sigma$ (which contributes a factor σ from $dx = \sigma\,dz$) gives $\int \mathcal{N}(x \mid \mu, \sigma^2)\,dx = 1$ precisely when the constant is $1/\sqrt{2\pi\sigma^2}$.

A similar computation, differentiating under the integral or integrating by parts, confirms that the parameters are named correctly: $\mathbb{E}[x] = \mu$ and $\mathrm{Var}(x) = \sigma^2$. The familiar 68–95–99.7 rule (Fig. 2.3) records how much mass lies within one, two, and three standard deviations of the mean.

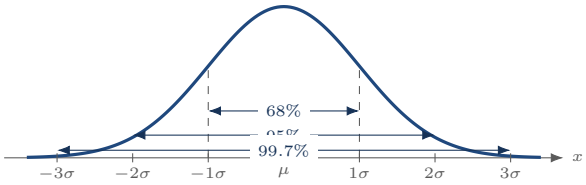


Figure 2.3. The Gaussian density and the 68–95–99.7 rule: about 68% of the mass lies within $\pm 1\sigma$ of the mean, 95% within $\pm 2\sigma$, and 99.7% within $\pm 3\sigma$.

Intuition (Why the Gaussian is everywhere). Two reasons. First, among all distributions with a given mean and variance, the Gaussian has the largest entropy (Remark 2.14), so it is the least committal choice once you have fixed the center and spread and wish to assume nothing more. Second, the central limit theorem says a sum of many small independent effects looks Gaussian regardless of their individual shapes, so measurement noise, aggregated errors, and sample averages all drift toward it. This is why least squares in Chapter 5 quietly assumes Gaussian noise, and why so much of machine learning is Gaussian underneath.

Remark 2.14 *(The Gaussian as the maximum-entropy distribution).* Professor Chin derives the Gaussian, rather than postulating it, by asking which density on $\mathbb{R}$ has the most entropy subject to a fixed mean μ and variance σ^2. Maximize $\mathrm{H}[p] = -\int p \log p$ subject to the three constraints $\int p = 1$, $\int x\,p = \mu$, and $\int (x-\mu)^2 p = \sigma^2$. Introducing multipliers $\lambda_0, \lambda_1, \lambda_2$ and setting the functional derivative of $-\int p \log p + \lambda_0 \int p + \lambda_1 \int xp + \lambda_2 \int (x-\mu)^2 p$ to zero gives, for every x,

$$-\log p(x) - 1 + \lambda_0 + \lambda_1 x + \lambda_2 (x-\mu)^2 = 0 \implies p(x) = \exp\big(\lambda_0 - 1 + \lambda_1 x + \lambda_2 (x-\mu)^2\big).$$

The result is the exponential of a quadratic in x, which is a Gaussian; solving for the multipliers so the constraints hold recovers exactly Eq. (2.18). The Gaussian is the honest choice when all you claim to know is a mean and a variance.

2.8 The multivariate Gaussian

The multivariate version packages a mean vector and a covariance matrix into the same exponential shape.

Definition 2.15 (Multivariate Gaussian). A random vector $\boldsymbol{x} \in \mathbb{R}^d$ is Gaussian, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, if it has density

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \exp\Big(-\tfrac{1}{2} (\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})\Big), \tag{2.19}$$

with mean $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{x}] \in \mathbb{R}^d$ and covariance $\boldsymbol{\Sigma} = \mathbb{E}[(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^\top] \in \mathbb{R}^{d \times d}$, symmetric and positive semidefinite; $|\boldsymbol{\Sigma}|$ is the determinant.

The exponent is a quadratic form in $\boldsymbol{x} - \boldsymbol{\mu}$, so the level sets where it is constant are ellipsoids centered at $\boldsymbol{\mu}$. By the spectral theorem applied to $\boldsymbol{\Sigma}$ (see Appendix A), the principal axes of these ellipsoids point along the eigenvectors of $\boldsymbol{\Sigma}$, with semi-axis lengths proportional to the square roots of the eigenvalues: a large variance in some direction stretches the ellipsoid along it.

Definition 2.16 (Mahalanobis distance). The quantity in the exponent, $d_{\boldsymbol{\Sigma}}(\boldsymbol{x}, \boldsymbol{\mu}) = \sqrt{(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})}$, is the *Mahalanobis distance* from $\boldsymbol{x}$ to the mean. It is Euclidean distance measured in units of standard deviation along each principal axis: a point two standard deviations out in a low-variance direction is farther, in Mahalanobis terms, than a point the same Euclidean distance out along a high-variance direction. The density depends on $\boldsymbol{x}$ only through this distance, so all points on one contour ellipsoid are equally probable. When $\boldsymbol{\Sigma} = I$ it reduces to ordinary distance and the contours are spheres.

Example 2.17 (A two-dimensional Gaussian). Let $\boldsymbol{\mu} = \mathbf{0}$ and $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$. The determinant is $|\boldsymbol{\Sigma}| = 2 \cdot 2 - 1 \cdot 1 = 3$, and the inverse is $\boldsymbol{\Sigma}^{-1} = \frac{1}{3}\left[\begin{smallmatrix} 2 & -1 \\ -1 & 2 \end{smallmatrix}\right]$. The eigenvalues solve $(2-\lambda)^2 - 1 = 0$, giving $\lambda_1 = 3$ with eigenvector $(1, 1)$

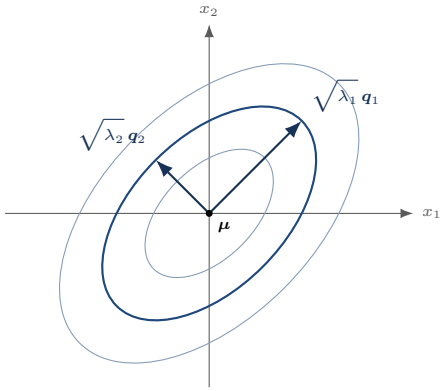


Figure 2.4. Contours of a Gaussian with $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$: ellipses centered at $\boldsymbol{\mu}$, axes along the eigenvectors $\boldsymbol{q}_1 = \frac{1}{\sqrt{2}}(1,1)$, $\boldsymbol{q}_2 = \frac{1}{\sqrt{2}}(1,-1)$ with semi-axes $\sqrt{\lambda_1} = \sqrt{3}$, $\sqrt{\lambda_2} = 1$. Positive covariance tilts the ellipse onto the diagonal.

and $\lambda_2 = 1$ with eigenvector $(1,-1)$, so the contours are ellipses elongated along the 45° line (Fig. 2.4). To see that Mahalanobis distance is the right yardstick, compare $\boldsymbol{x} = (1,1)$ along the long axis with $\boldsymbol{z} = (2,0)$ across it:

$$\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x} = \tfrac{1}{3}(1,1)\begin{bmatrix}1\\1\end{bmatrix} = \tfrac{2}{3}, \qquad \boldsymbol{z}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{z} = \tfrac{1}{3}(2,0)\begin{bmatrix}4\\-2\end{bmatrix} = \tfrac{8}{3}.$$

So $(1,1)$ has Mahalanobis distance $\sqrt{2/3} \approx 0.82$ while $(2,0)$ has $\sqrt{8/3} \approx 1.63$: the point lying along the cloud's long axis is statistically nearer, though both are a couple of units out in plain Euclidean terms. ◊

The Gaussian is closed under the operations we care about. Any affine image is Gaussian,

$$\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \implies A\boldsymbol{x} + \boldsymbol{b} \sim \mathcal{N}(A\boldsymbol{\mu} + \boldsymbol{b},\ A\boldsymbol{\Sigma}A^\top), \tag{2.20}$$

and, as the next section shows, every marginal and every conditional of a Gaussian is again Gaussian. No other continuous distribution is this well behaved, which is the practical reason the Gaussian dominates.

2.8.1 Maximum likelihood, and the bias of the variance estimate

Given independent samples $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, maximizing the log-likelihood (a matrix-calculus computation carried out in Appendix C) yields the natural estimates, the sample mean and the sample covariance:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i, \qquad \hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}})^\top. \tag{2.21}$$

The mean estimate is unbiased, but the variance estimate is not, and the size of the bias is worth deriving because it is the reason statistical software divides by $N-1$. We do the scalar case, which carries all the ideas. Two facts about one Gaussian sample x_i are needed, both from Eq. (2.2): $\mathbb{E}[x_i] = \mu$ and $\mathbb{E}[x_i^2] = \mu^2 + \sigma^2$.

First, the easy case, when the *true* mean μ is known. Then the estimator $\frac{1}{N}\sum_i (x_i - \mu)^2$ is unbiased:

$$\mathbb{E}\Big[\tfrac{1}{N}\sum_{i=1}^{N}(x_i-\mu)^2\Big] = \tfrac{1}{N}\sum_{i=1}^{N}\mathbb{E}[(x_i-\mu)^2] = \tfrac{1}{N}\sum_{i=1}^{N}\mathrm{Var}(x_i) = \tfrac{1}{N}\cdot N\sigma^2 = \sigma^2.$$

Now the case that actually occurs, when the mean is replaced by the sample mean $\bar{x} = \frac{1}{N}\sum_i x_i$. The estimator $\hat{\sigma}^2 = \frac{1}{N}\sum_i (x_i - \bar{x})^2$ is biased *low*. Add and subtract μ inside the square and expand:

$$\sum_i (x_i - \bar{x})^2 = \sum_i \big[(x_i - \mu) - (\bar{x} - \mu)\big]^2 = \sum_i (x_i-\mu)^2 - 2(\bar{x}-\mu)\sum_i (x_i - \mu) + N(\bar{x}-\mu)^2.$$

The middle sum is $\sum_i (x_i - \mu) = N(\bar{x} - \mu)$, so the middle term is $-2N(\bar{x}-\mu)^2$ and the three terms collapse to

$$\sum_i (x_i - \bar{x})^2 = \sum_i (x_i - \mu)^2 - N(\bar{x} - \mu)^2.$$

Take expectations. The first sum has expectation $N\sigma^2$, as above. For the second, $\bar{x}$ is the average of N independent samples, so its variance is $\mathrm{Var}(\bar{x}) = \sigma^2/N$, hence $\mathbb{E}[(\bar{x}-\mu)^2] = \sigma^2/N$ and $\mathbb{E}[N(\bar{x}-\mu)^2] = \sigma^2$. Therefore

$$\mathbb{E}[\hat{\sigma}^2] = \frac{1}{N}\left(N\sigma^2 - \sigma^2\right) = \frac{N-1}{N}\sigma^2. \tag{2.22}$$

The maximum likelihood variance is too small by the factor $(N-1)/N$, because it measures spread about the sample mean, which sits a little closer to the data than the true mean does. Dividing the sum of squares by $N-1$ instead of N removes the bias, and that is the origin of the "$N-1$" in the unbiased sample variance. The correction vanishes as $N \to \infty$, where the two estimators agree.

2.9 Conditional and marginal Gaussians

The closure of the Gaussian under conditioning is the single fact that powers Bayesian linear regression in Chapter 6, so we derive it. Partition a Gaussian vector $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ into two blocks $\boldsymbol{x} = (\boldsymbol{x}_a, \boldsymbol{x}_b)$, and partition the mean and covariance to match,

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_a \\ \boldsymbol{\mu}_b \end{pmatrix}, \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{aa} & \boldsymbol{\Sigma}_{ab} \\ \boldsymbol{\Sigma}_{ba} & \boldsymbol{\Sigma}_{bb} \end{pmatrix}, \qquad \boldsymbol{\Sigma}^{-1} = \boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Lambda}_{aa} & \boldsymbol{\Lambda}_{ab} \\ \boldsymbol{\Lambda}_{ba} & \boldsymbol{\Lambda}_{bb} \end{pmatrix},$$

where $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$ is the *precision matrix*. The precision, rather than the covariance, is what makes conditioning clean.

Theorem 2.18 (Conditional and marginal of a Gaussian). If $\boldsymbol{x} = (\boldsymbol{x}_a, \boldsymbol{x}_b)$ is jointly Gaussian, then the conditional $\boldsymbol{x}_a \mid \boldsymbol{x}_b$ and the marginal $\boldsymbol{x}_a$ are both Gaussian, with

$$\boldsymbol{x}_a \mid \boldsymbol{x}_b \sim \mathcal{N}\big(\boldsymbol{\mu}_{a|b},\ \boldsymbol{\Lambda}_{aa}^{-1}\big), \qquad \boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b), \tag{2.23}$$

$$\boldsymbol{x}_a \sim \mathcal{N}(\boldsymbol{\mu}_a,\ \boldsymbol{\Sigma}_{aa}). \tag{2.24}$$

Proof sketch for the conditional. Fixing $\boldsymbol{x}_b$, the conditional density is proportional to the joint, so we examine the exponent of Eq. (2.19) as a function of $\boldsymbol{x}_a$ alone. Expanding the quadratic form with the partitioned precision,

$$-\tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Lambda}(\boldsymbol{x}-\boldsymbol{\mu}) = -\tfrac{1}{2}(\boldsymbol{x}_a-\boldsymbol{\mu}_a)^\top \boldsymbol{\Lambda}_{aa}(\boldsymbol{x}_a-\boldsymbol{\mu}_a) - (\boldsymbol{x}_a-\boldsymbol{\mu}_a)^\top \boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b-\boldsymbol{\mu}_b) + \text{const},$$

where "const" collects terms free of $\boldsymbol{x}_a$. This is a quadratic in $\boldsymbol{x}_a$ whose second-order coefficient is $\boldsymbol{\Lambda}_{aa}$, so the conditional is Gaussian with covariance $\boldsymbol{\Lambda}_{aa}^{-1}$. Completing the square (matching the linear term) identifies the mean as $\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a - \boldsymbol{\Lambda}_{aa}^{-1}\boldsymbol{\Lambda}_{ab}(\boldsymbol{x}_b - \boldsymbol{\mu}_b)$, which is Eq. (2.23). The marginal Eq. (2.24) follows by integrating $\boldsymbol{x}_b$ out, and reads off the corresponding blocks of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ directly. ∎

The crucial qualitative fact is in Eq. (2.23): the conditional mean $\boldsymbol{\mu}_{a|b}$ is a *linear* (affine) function of the conditioning variable $\boldsymbol{x}_b$, and the conditional covariance $\boldsymbol{\Lambda}_{aa}^{-1}$ does *not* depend on $\boldsymbol{x}_b$ at all. Rewriting the mean in terms of the covariance blocks (via the block-inverse identities) puts it in the form

$$\mathbb{E}[\boldsymbol{x}_a \mid \boldsymbol{x}_b] = \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\boldsymbol{x}_b - \boldsymbol{\mu}_b), \tag{2.25}$$

whose coefficient $\boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}$ is exactly the matrix of least-squares regression weights of Chapter 5.

Remark 2.19 *(Conditioning is linear regression).* Equation (2.25) says the best estimate of one part of a Gaussian vector given another part is a linear function of the observed part, with coefficients built from covariances. Linear regression is what Gaussian conditioning looks like, and this is why the Gaussian and the linear model are inseparable in Chapters 5 to 6. The same partitioned-Gaussian algebra, applied to a linear-Gaussian model $p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1})$ and $p(\boldsymbol{y} \mid \boldsymbol{x}) = \mathcal{N}(A\boldsymbol{x} + \boldsymbol{b}, \boldsymbol{L}^{-1})$, yields a Gaussian marginal $p(\boldsymbol{y})$ and a Gaussian posterior $p(\boldsymbol{x} \mid \boldsymbol{y})$ in closed form; we derive that special case where it is needed, in the Bayesian linear regression of Chapter 6.

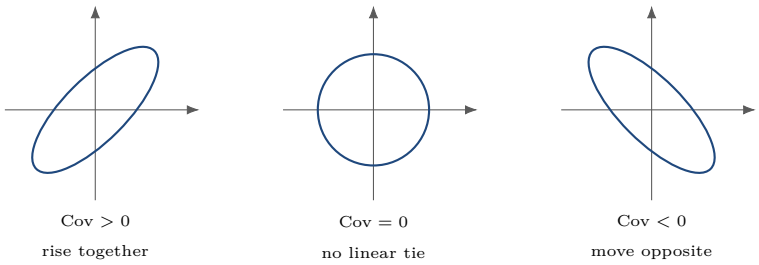


Figure 2.5. Positive covariance tilts the ellipse up the diagonal, negative covariance tilts it down, and zero covariance leaves it axis-aligned. The covariance matrix records this tilt for every pair of coordinates at once.

2.10 Covariance and the covariance matrix

The off-diagonal structure of $\boldsymbol{\Sigma}$ deserves its own treatment, because it is how the Gaussian encodes relationships between coordinates.

Definition 2.20 (Covariance). The *covariance* of two scalar random variables is

$$\operatorname{Cov}(X, Y) = \mathbb{E}\big[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])\big] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y], \tag{2.26}$$

positive when X and Y tend to deviate from their means together, negative when in opposite directions, and zero when there is no *linear* relationship. Its self-version is the variance, $\operatorname{Cov}(X, X) = \operatorname{Var}(X)$.

Independence is strictly stronger than zero covariance.

Proposition 2.21 (Independence implies uncorrelated, not conversely). If $X \perp\!\!\!\perp Y$ then $\operatorname{Cov}(X, Y) = 0$. The converse fails in general.

Proof. Independence gives $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$, so $\operatorname{Cov}(X, Y) = 0$ by Eq. (2.26). For the converse, let $X \sim \operatorname{Unif}(-1, 1)$ and $Y = X^2$. By symmetry $\mathbb{E}[X] = 0$ and $\mathbb{E}[XY] = \mathbb{E}[X^3] = 0$ (an odd moment of a symmetric distribution), so $\operatorname{Cov}(X, Y) = 0$; yet Y is a deterministic function of X, so they are as dependent as two variables can be. Zero covariance rules out only linear association. For jointly Gaussian variables, however, zero covariance does imply independence, because then the joint density factors. ∎

For a random vector $\boldsymbol{x} \in \mathbb{R}^d$, the covariances of all coordinate pairs assemble into the covariance matrix $\boldsymbol{\Sigma} = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top]$, with $\Sigma_{ij} = \operatorname{Cov}(x_i, x_j)$: variances on the diagonal, covariances off it. It is symmetric, and it is positive semidefinite.

Proposition 2.22 (Covariance matrices are positive semidefinite). For any random vector $\boldsymbol{x}$ with covariance matrix $\boldsymbol{\Sigma}$ and any fixed $\boldsymbol{a} \in \mathbb{R}^d$, $\boldsymbol{a}^\top \boldsymbol{\Sigma} \boldsymbol{a} = \operatorname{Var}(\boldsymbol{a}^\top \boldsymbol{x}) \geq 0$, so $\boldsymbol{\Sigma} \succeq 0$.

Proof. The scalar $\boldsymbol{a}^\top \boldsymbol{x}$ has variance

$$\operatorname{Var}(\boldsymbol{a}^\top \boldsymbol{x}) = \mathbb{E}\big[(\boldsymbol{a}^\top(\boldsymbol{x} - \boldsymbol{\mu}))^2\big] = \mathbb{E}\big[\boldsymbol{a}^\top(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{a}\big] = \boldsymbol{a}^\top \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top]\, \boldsymbol{a} = \boldsymbol{a}^\top \boldsymbol{\Sigma} \boldsymbol{a},$$

which is nonnegative as the expectation of a square. Holding for every $\boldsymbol{a}$, this is the definition of positive semidefiniteness. ∎

Because $\boldsymbol{\Sigma}$ is symmetric and positive semidefinite, the spectral theorem gives $\boldsymbol{\Sigma} = Q \boldsymbol{\Lambda} Q^\top$ with orthonormal eigenvectors and nonnegative eigenvalues. The eigenvectors are the principal axes of the data's spread and the eigenvalues are the variances along them, which is exactly the content of principal component analysis in Chapter 19. Rotating into the eigenbasis, $A = Q^\top$, diagonalizes the covariance by Eq. (2.20), so the rotated coordinates are uncorrelated. That decorrelation is the engine of PCA.

Distribution	Support	Parameters	Mean	Conjugate prior
Bernoulli	$\{0,1\}$	μ	μ	Beta
Binomial	$\{0,\dots,N\}$	μ, N	$N\mu$	Beta
Categorical	$\{1,\dots,K\}$	$\boldsymbol{\mu}$	$\boldsymbol{\mu}$	Dirichlet
Multinomial	counts, $\sum = N$	$\boldsymbol{\mu}, N$	$N\boldsymbol{\mu}$	Dirichlet
Beta	$[0,1]$	a, b	$a/(a+b)$	n/a
Dirichlet	simplex	$\boldsymbol{\alpha}$	$\boldsymbol{\alpha}/\alpha_0$	n/a
Gaussian	$\mathbb{R}^d$	$\boldsymbol{\mu}, \boldsymbol{\Sigma}$	$\boldsymbol{\mu}$	Gaussian (on $\boldsymbol{\mu}$)

Table 2.1. The discrete families pair with a counting conjugate prior (Beta, Dirichlet); the Gaussian is conjugate to itself for the mean. Every mean is the parameter that the corresponding maximum likelihood estimate recovers as a frequency or an average.

Example 2.23 (Sample covariance and its principal axes). Take five points in $\mathbb{R}^2$: $(1,2), (2,3), (3,5), (4,4), (5,6)$. Their mean is $\bar{\boldsymbol{x}} = \frac{1}{5}(15,20) = (3,4)$. Centering each point gives $(-2,-2), (-1,-1), (0,1), (1,0), (2,2)$. The (unbiased) sample covariance sums the outer products and divides by $N-1=4$:

$$\boldsymbol{\Sigma} = \frac{1}{4}\sum_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^\top = \frac{1}{4}\begin{bmatrix} 10 & 9 \\ 9 & 10 \end{bmatrix} = \begin{bmatrix} 2.5 & 2.25 \\ 2.25 & 2.5 \end{bmatrix}.$$

The positive off-diagonal says x_1 and x_2 rise together. Its eigenvalues are $2.5 \pm 2.25 = \{4.75,\ 0.25\}$ with eigenvectors $\frac{1}{\sqrt{2}}(1,1)$ and $\frac{1}{\sqrt{2}}(1,-1)$, so the principal axis is the 45° direction, carrying $4.75/(4.75+0.25) = 95\%$ of the total variance. This eigendecomposition of the covariance is principal component analysis, here by hand on five points. $\lozenge$

2.11 The distributions at a glance

Table 2.1 collects the distributions of this chapter. Reading it traces the two ways the coin flip generalizes, across to more outcomes and in aggregate to counts, with the continuous Gaussian sitting apart as the limit a sum of many trials approaches. Every discrete family pairs with a counting conjugate prior, and the common structure, likelihood times conjugate prior gives a same-family posterior, is the unifying theme that Chapter 3 explains in one stroke through the exponential family.

Notes and References

The development of this chapter follows Chapter 2 of Bishop [3], with the mathematics-of-machine-learning perspective of Deisenroth et al. [9], Chapter 6. The conjugacy of the Beta and Dirichlet priors, the partitioned-Gaussian conditioning of Theorem 2.18, and the maximum-entropy characterization of the Gaussian (Remark 2.14) are all standard there. Laplace's rule of succession, Eq. (2.14), dates to Laplace's work on the probability of the sun rising. The bias correction of Eq. (2.22) is the reason for the $N-1$ in the unbiased sample variance, treated in any statistics text. The exponential family of Chapter 3 will show that the Bernoulli, multinomial, and Gaussian are three faces of one general form, and that conjugacy is a property of that form rather than a coincidence of these examples.

Exercises

Exercise 2.1. Let $x_1,\dots,x_N \overset{\text{iid}}{\sim} \text{Bern}(\mu)$ with $m = \sum_i x_i$. Derive the maximum likelihood estimate $\hat{\mu}$ from the likelihood, and verify the stationary point is a maximum.

Solution. The likelihood is $\mu^m(1-\mu)^{N-m}$, with log-likelihood $\ell(\mu) = m\log\mu + (N-m)\log(1-\mu)$. Then $\ell'(\mu) = m/\mu - (N-m)/(1-\mu) = 0$ gives $m(1-\mu) = (N-m)\mu$, hence $m = N\mu$ and $\hat{\mu} = m/N$. The second derivative $\ell''(\mu) = -m/\mu^2 - (N-m)/(1-\mu)^2 < 0$ throughout $(0,1)$, so ℓ is concave and $\hat{\mu} = m/N$ is the global maximizer. $\square$

Exercise 2.2. A test has sensitivity $\mathbb{P}(+ \mid D) = 0.99$ and false-positive rate $\mathbb{P}(+ \mid \neg D) = 0.05$, with prevalence $\mathbb{P}(D) = 0.01$. Find $\mathbb{P}(D \mid +)$, and explain why it is so much smaller than the test's accuracy.

Solution. By Bayes' theorem with the sum rule in the denominator, $\mathbb{P}(D \mid +) = \frac{(0.99)(0.01)}{(0.99)(0.01)+(0.05)(0.99)} = \frac{0.0099}{0.0594} = \frac{1}{6} \approx 16.7\%$. The small base rate is the reason: among 10,000 people only about 100 are sick (almost all testing positive), while about 5% of the 9,900 healthy people, roughly 495, also test positive, so true positives are outnumbered five to one. □

Exercise 2.3. With a Beta$(2, 2)$ prior on a coin's bias, you observe 8 heads in 10 flips. (a) Find the posterior. (b) Find its mean and the MAP estimate, and compare with the MLE. (c) Find the predicted probability that the next flip is heads.

Solution. (a) The posterior is $\text{Beta}(2+8,\ 2+2) = \text{Beta}(10, 4)$. (b) Mean $\frac{10}{14} = \frac{5}{7} \approx 0.714$; MAP (mode) $\frac{10-1}{14-2} = \frac{9}{12} = 0.75$; MLE $\frac{8}{10} = 0.80$. The Bayesian estimates lie between the data and the prior mean 0.5, pulled toward fairness by the four pseudo-flips. (c) By Eq. (2.14) the predictive probability is the posterior mean, $\frac{a+m}{a+b+N} = \frac{2+8}{2+2+10} = \frac{10}{14} \approx 0.714$. □

Exercise 2.4. A four-sided die is rolled 10 times, giving counts $\boldsymbol{m} = (2, 3, 4, 1)$. (a) Find the maximum likelihood estimate of the face probabilities. (b) With a uniform Dirichlet prior $\boldsymbol{\alpha} = (1, 1, 1, 1)$, find the posterior mean.

Solution. (a) By Proposition 2.11 the MLE is the empirical frequency $\hat{\boldsymbol{\mu}} = \boldsymbol{m}/N = (0.2, 0.3, 0.4, 0.1)$. (b) The posterior is $\text{Dir}(\boldsymbol{\alpha} + \boldsymbol{m}) = \text{Dir}(3, 4, 5, 2)$ with $\alpha_0 + N = 4 + 10 = 14$, so the posterior mean is $\left(\frac{3}{14}, \frac{4}{14}, \frac{5}{14}, \frac{2}{14}\right) \approx (0.214, 0.286, 0.357, 0.143)$, each pulled slightly toward uniformity by one pseudo-count per face. □

Exercise 2.5. Show that $\int_{-\infty}^{\infty} e^{-z^2/2}\,dz = \sqrt{2\pi}$ by squaring the integral and passing to polar coordinates, and deduce the normalizing constant of $\mathcal{N}(x \mid \mu, \sigma^2)$.

Solution. Let $I = \int e^{-z^2/2}\,dz$. Then $I^2 = \iint_{\mathbb{R}^2} e^{-(x^2+y^2)/2}\,dx\,dy$. In polar coordinates $dx\,dy = r\,dr\,d\theta$ and $x^2 + y^2 = r^2$, so $I^2 = \int_0^{2\pi} d\theta \int_0^{\infty} re^{-r^2/2}\,dr = 2\pi\,[-e^{-r^2/2}]_0^{\infty} = 2\pi$. Hence $I = \sqrt{2\pi}$. Substituting $z = (x - \mu)/\sigma$ contributes a factor σ, so $\int e^{-(x-\mu)^2/2\sigma^2}\,dx = \sigma\sqrt{2\pi}$, and the density must carry the reciprocal constant $1/\sqrt{2\pi\sigma^2}$ to integrate to one. □

Exercise 2.6. For $x_1, \ldots, x_N \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$, show that the maximum likelihood variance $\hat{\sigma}^2 = \frac{1}{N}\sum_i (x_i - \bar{x})^2$, using the sample mean $\bar{x}$, satisfies $\mathbb{E}[\hat{\sigma}^2] = \frac{N-1}{N}\sigma^2$.

Solution. Write $\sum_i (x_i - \bar{x})^2 = \sum_i (x_i - \mu)^2 - N(\bar{x} - \mu)^2$ (add and subtract μ, expand, and use $\sum_i (x_i - \mu) = N(\bar{x} - \mu)$). Taking expectations, $\mathbb{E}[\sum_i (x_i - \mu)^2] = N\sigma^2$ and $\mathbb{E}[N(\bar{x} - \mu)^2] = N \operatorname{Var}(\bar{x}) = N(\sigma^2/N) = \sigma^2$, since $\operatorname{Var}(\bar{x}) = \sigma^2/N$ for an average of N independent samples. Therefore $\mathbb{E}[\hat{\sigma}^2] = \frac{1}{N}(N\sigma^2 - \sigma^2) = \frac{N-1}{N}\sigma^2$, biased low by the factor $(N-1)/N$, which is why the unbiased estimator divides by $N - 1$. □

Exercise 2.7. Give two random variables that are uncorrelated but not independent, and verify both claims. Why does the example not contradict the fact that, for jointly Gaussian variables, uncorrelated implies independent?

Solution. Let $X \sim \text{Unif}(-1, 1)$ and $Y = X^2$. By symmetry $\mathbb{E}[X] = 0$ and $\mathbb{E}[X^3] = 0$, so $\operatorname{Cov}(X, Y) = \mathbb{E}[X^3] - \mathbb{E}[X]\mathbb{E}[X^2] = 0$: uncorrelated. They are not independent, since Y is determined by X; for instance $\mathbb{P}(Y < \frac{1}{4} \mid X = 0) = 1 \neq \mathbb{P}(Y < \frac{1}{4})$. There is no contradiction because (X, Y) is not jointly Gaussian ($Y = X^2$ is not even Gaussian), and the "uncorrelated implies independent" fact holds only for jointly Gaussian variables. □

CHAPTER 3

The Exponential Family and Conjugacy

The Bernoulli, the multinomial, and the Gaussian are not three distributions. They are one distribution, written three ways.

after Professor Peter Chin, ENGS 106

Chapter 2 marched through a zoo of distributions, and each time the same small miracle occurred. The maximum likelihood estimate was always a simple average of something. The conjugate prior always existed, and the posterior was always found by adding the data's counts to the prior's pseudo-counts. This was no coincidence. The Bernoulli, the binomial, the categorical, the multinomial, the Gaussian, the Beta, the Dirichlet, the gamma, and the Poisson are all members of a single family, the *exponential family*, and the three recurring miracles, a sufficient statistic, a moment formula, and automatic conjugacy, are properties of the family as a whole. This chapter is the payoff for the bookkeeping of the last one: we write the general form once, and watch the special cases fall out. As a bonus, the two functions that will run every classifier in Chapters 7 to 9, the logistic sigmoid and the softmax, drop out of the family as the maps that convert its natural parameters back into probabilities.

Roadmap

Suggested reading: Bishop [3], Section 2.4 (the exponential family) and Section 2.5 (nonparametric density estimation).

We cover: the general form and its four ingredients (Section 3.1); the Bernoulli, categorical, and Gaussian as members, from which the sigmoid and softmax emerge (Section 3.2); sufficient statistics and data reduction (Section 3.3); moments from the log-partition function (Section 3.4); maximum likelihood as moment matching (Section 3.5); conjugate priors for the whole family at once (Section 3.6); the bridge to logistic regression and generalized linear models (Section 3.7); and, when no parametric family fits, nonparametric density estimation (Section 3.8).

3.1 The general form

Definition 3.1 (Exponential family). A family of distributions over $\boldsymbol{x}$, parameterized by $\boldsymbol{\eta}$, is an *exponential family* if its density (or mass) can be written

$$p(\boldsymbol{x} \mid \boldsymbol{\eta}) = h(\boldsymbol{x})\, g(\boldsymbol{\eta})\, \exp\!\big(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{x})\big), \tag{3.1}$$

where $\boldsymbol{\eta}$ are the *natural parameters*, $\boldsymbol{u}(\boldsymbol{x})$ is the vector of *sufficient statistics*, $h(\boldsymbol{x}) \geq 0$ is the *base measure*, and $g(\boldsymbol{\eta})$ is the *normalizer* that makes the density integrate to one:

$$g(\boldsymbol{\eta})^{-1} = \int h(\boldsymbol{x})\, \exp\!\big(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{x})\big)\, d\boldsymbol{x}. \tag{3.2}$$

The shape to remember is *base measure, times the exponential of something linear in the sufficient statistics.* Everything specific to a member, what makes a Bernoulli a Bernoulli and a Gaussian a Gaussian, is packed into the choice of $\boldsymbol{u}(\boldsymbol{x})$ and $h(\boldsymbol{x})$; the natural parameter $\boldsymbol{\eta}$ is just the knob, and $g(\boldsymbol{\eta})$ is whatever it must be to normalize. The single most useful object derived from Eq. (3.2) is the *log-partition function*

$$A(\boldsymbol{\eta}) = -\log g(\boldsymbol{\eta}) = \log \int h(\boldsymbol{x})\, \exp\!\big(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{x})\big)\, d\boldsymbol{x}, \tag{3.3}$$

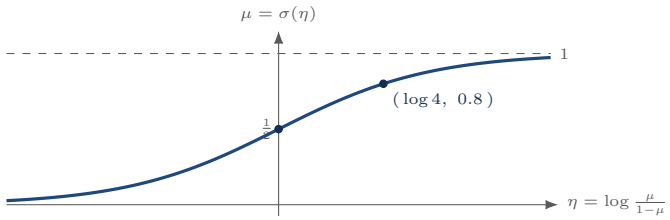


Figure 3.1. The natural parameter of a Bernoulli is the log-odds η, an unbounded real number, and the logistic sigmoid maps it back to a probability $\mu \in (0, 1)$. The fair coin sits at the center $(0, \frac{1}{2})$; the coin of Example 3.2 sits at $(\log 4, 0.8)$.

so the density reads $p(\boldsymbol{x} \mid \boldsymbol{\eta}) = h(\boldsymbol{x}) \exp(\boldsymbol{\eta}^\top \boldsymbol{u}(\boldsymbol{x}) - A(\boldsymbol{\eta}))$. We will see in Section 3.4 that the derivatives of $A(\boldsymbol{\eta})$ are the moments of $\boldsymbol{u}(\boldsymbol{x})$, which is the reason the family is so easy to work with.

3.2 The members, and where the sigmoid and softmax come from

Three derivations show how familiar distributions fold into Eq. (3.1). In each, the move is the same: take the logarithm of the density, and read off which function of $\boldsymbol{x}$ multiplies which function of the parameter.

3.2.1 The Bernoulli, and the logistic sigmoid

Write the Bernoulli mass $\mu^x(1-\mu)^{1-x}$ as the exponential of its logarithm:

$$\begin{aligned} \mathrm{Bern}(x \mid \mu) &= \exp\Big(x \log \mu + (1-x)\log(1-\mu)\Big) \\ &= \exp\left(x \log \frac{\mu}{1-\mu} + \log(1-\mu)\right) = (1-\mu)\, \exp\left(x \log \frac{\mu}{1-\mu}\right). \end{aligned} \tag{3.4}$$

Comparing with Eq. (3.1), the sufficient statistic is $u(x) = x$, the base is $h(x) = 1$, and the natural parameter is the *log-odds*

$$\eta = \log \frac{\mu}{1-\mu}, \tag{3.5}$$

the logarithm of the ratio of success to failure. This is worth dwelling on, because inverting it produces the most important function in the classification half of the book. Solving Eq. (3.5) for μ: exponentiate to get $e^\eta = \mu/(1-\mu)$, so $\mu = e^\eta(1-\mu) = e^\eta - e^\eta\mu$, hence $\mu(1+e^\eta) = e^\eta$ and

$$\mu = \frac{e^\eta}{1+e^\eta} = \frac{1}{1+e^{-\eta}} = \sigma(\eta). \tag{3.6}$$

The map from the natural parameter back to the probability is the *logistic sigmoid*. The Bernoulli's normalizer is $g(\eta) = 1 - \mu = \sigma(-\eta) = 1/(1+e^\eta)$, so its log-partition function is $A(\eta) = \log(1+e^\eta)$, a function we will differentiate in a moment.

The sigmoid is the Bernoulli's inverse link

The natural parameter of a Bernoulli is the log-odds $\eta = \log \frac{\mu}{1-\mu}$, and the probability is recovered by the logistic sigmoid $\mu = \sigma(\eta) = 1/(1+e^{-\eta})$. Modeling the log-odds as a linear function of the input, $\eta = \boldsymbol{w}^\top \boldsymbol{x}$, is exactly logistic regression (Chapter 8).

Example 3.2 (Log-odds by hand). A coin with $\mu = 0.8$ has log-odds $\eta = \log(0.8/0.2) = \log 4 \approx 1.386$. Going back, $\sigma(1.386) = 1/(1+e^{-1.386}) = 1/(1+0.25) = 0.8$, recovering μ. A fair coin $\mu = 0.5$ has $\eta = \log 1 = 0$, the center of the sigmoid, and $\sigma(0) = 0.5$. As $\mu \to 1$ the log-odds $\eta \to +\infty$, and as $\mu \to 0$ it goes to $-\infty$: the sigmoid squashes the whole real line onto the open interval $(0, 1)$, which is why it is the natural way to turn an unbounded linear score into a probability. ◊

Distribution	$\boldsymbol{u}(x)$	**natural param.** $\boldsymbol{\eta}$	**inverse link**	$A(\boldsymbol{\eta})$
Bernoulli	x	$\log \frac{\mu}{1-\mu}$	sigmoid $\mu = \sigma(\eta)$	$\log(1+e^{\eta})$
Categorical	$\boldsymbol{x}$ (one-hot)	$\log \mu_k$	softmax	$\log \sum_k e^{\eta_k}$
Gaussian (fixed σ^2)	x	μ/σ^2	$\mu = \sigma^2 \eta$	$\sigma^2 \eta^2/2$
Poisson	x	$\log \lambda$	$\lambda = e^{\eta}$	e^{η}

Table 3.1. Four members written in canonical form. The natural parameter is a transform of the usual parameter, and the inverse link maps it back. The sigmoid and softmax, which run every classifier in Part III, are simply the inverse links of the Bernoulli and categorical.

3.2.2 The categorical, and the softmax

The same calculation on K outcomes produces the softmax. Writing the categorical mass with a one-hot $\boldsymbol{x}$,

$$\mathrm{Cat}(\boldsymbol{x} \mid \boldsymbol{\mu}) = \prod_{k=1}^{K} \mu_k^{x_k} = \exp\Big(\sum_{k=1}^{K} x_k \log \mu_k\Big),$$

the sufficient statistic is $\boldsymbol{x}$ itself and the natural parameters are $\eta_k = \log \mu_k$. Because the probabilities are constrained by $\sum_k \mu_k = 1$, the map back to probabilities must renormalize, and inverting $\eta_k = \log \mu_k$ subject to that constraint gives

$$\mu_k = \frac{e^{\eta_k}}{\sum_{j=1}^{K} e^{\eta_j}} = \mathrm{softmax}(\boldsymbol{\eta})_k. \tag{3.7}$$

The softmax is to the categorical exactly what the sigmoid is to the Bernoulli, the inverse link that turns a vector of unbounded scores into a probability vector on the simplex, and with $K = 2$ it reduces to the sigmoid. It is the output layer of every multi-class classifier and neural network in this book.

3.2.3 The Gaussian

The Gaussian fits the same mold, with a two-component sufficient statistic. Expand the exponent of $\mathcal{N}(x \mid \mu, \sigma^2)$:

$$-\frac{(x-\mu)^2}{2\sigma^2} = -\frac{x^2 - 2\mu x + \mu^2}{2\sigma^2} = \frac{\mu}{\sigma^2}\, x \;-\; \frac{1}{2\sigma^2}\, x^2 \;-\; \frac{\mu^2}{2\sigma^2}.$$

The terms linear in the data are $(\mu/\sigma^2)\, x$ and $(-1/2\sigma^2)\, x^2$, so the sufficient statistic is the pair $\boldsymbol{u}(x) = (x,\ x^2)$ and the natural parameters are

$$\boldsymbol{\eta} = \Big(\frac{\mu}{\sigma^2},\ -\frac{1}{2\sigma^2}\Big), \qquad h(x) = \frac{1}{\sqrt{2\pi}}, \tag{3.8}$$

with the rest, $-\mu^2/2\sigma^2$ and the $1/\sigma$ factor, absorbed into $g(\boldsymbol{\eta})$. The lesson is that a Gaussian's sufficient statistics are the first two powers of x: to fit a Gaussian, all you ever need from the data are the sum of the values and the sum of their squares, a fact we make precise next.

3.3 Sufficient statistics and data reduction

The name "sufficient statistic" is earned by what happens with a whole data set. For N independent points $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, the likelihood is a product, and in the exponential family the product collapses neatly:

$$p(\mathcal{D} \mid \boldsymbol{\eta}) = \prod_{n=1}^{N} h(\boldsymbol{x}_n)\, g(\boldsymbol{\eta})\, \exp\big(\boldsymbol{\eta}^{\top} \boldsymbol{u}(\boldsymbol{x}_n)\big) = \Big(\prod_{n} h(\boldsymbol{x}_n)\Big) g(\boldsymbol{\eta})^N \exp\Big(\boldsymbol{\eta}^{\top} \sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n)\Big). \tag{3.9}$$

Look at where the parameter $\boldsymbol{\eta}$ touches the data: only through the single sum $\sum_n \boldsymbol{u}(\boldsymbol{x}_n)$. Two data sets with the same value of this sum give the same likelihood for every $\boldsymbol{\eta}$, hence the same inference. The sum is a *sufficient statistic*: it distills everything the data say about the parameter into a fixed-size summary, no matter how large N is.

Data reduction

In an exponential family, the only thing the data tell you about $\boldsymbol{\eta}$ is the value of $\sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n)$. For a Bernoulli that is the head count $\sum_n x_n$; for a Gaussian it is the pair $\left(\sum_n x_n,\ \sum_n x_n^2\right)$. A million coin flips compress, with no loss, to a single integer.

This is the theoretical reason a streaming average works: to maintain a Gaussian estimate over an endless stream, you keep a running sum and a running sum of squares, and throw every data point away after reading it.

3.4 Moments from the log-partition function

The log-partition function $A(\boldsymbol{\eta})$ of Eq. (3.3) is a generating function: differentiating it produces the moments of the sufficient statistic, with no integration required. We prove the first-moment identity in the scalar case; the vector case is identical with gradients in place of derivatives.

Theorem 3.3 (Moments of the sufficient statistic). For an exponential family with log-partition function $A(\boldsymbol{\eta})$,

$$\nabla A(\boldsymbol{\eta}) = \mathbb{E}[\boldsymbol{u}(\boldsymbol{x})], \qquad \nabla^2 A(\boldsymbol{\eta}) = \mathrm{Cov}[\boldsymbol{u}(\boldsymbol{x})]. \tag{3.10}$$

Proof. Start from the definition $A(\eta) = \log \int h(x)\, e^{\eta u(x)}\, dx$. Differentiate with respect to η, using the chain rule on the logarithm and differentiating under the integral sign:

$$A'(\eta) = \frac{\frac{d}{d\eta} \int h(x)\, e^{\eta u(x)}\, dx}{\int h(x)\, e^{\eta u(x)}\, dx} = \frac{\int h(x)\, u(x)\, e^{\eta u(x)}\, dx}{\int h(x)\, e^{\eta u(x)}\, dx}.$$

The denominator is $e^{A(\eta)} = 1/g(\eta)$, so multiplying it back in,

$$A'(\eta) = \int u(x)\, \underbrace{h(x)\, g(\eta)\, e^{\eta u(x)}}_{= p(x \mid \eta)}\, dx = \int u(x)\, p(x \mid \eta)\, dx = \mathbb{E}[u(x)].$$

The integrand in the middle is exactly the density Eq. (3.1), so the integral is the expectation of $u(x)$, proving the first identity. Differentiating once more, the quotient rule produces $A''(\eta) = \mathbb{E}[u^2] - (\mathbb{E}[u])^2 = \mathrm{Var}[u]$, the second identity. (In the vector case the same steps give the gradient and the covariance matrix.) ■

Example 3.4 (The Bernoulli mean, the easy way). The Bernoulli has $A(\eta) = \log(1 + e^{\eta})$. Differentiating,

$$A'(\eta) = \frac{e^{\eta}}{1 + e^{\eta}} = \frac{1}{1 + e^{-\eta}} = \sigma(\eta) = \mu.$$

So $\mathbb{E}[u(x)] = \mathbb{E}[x] = \mu$, recovering the Bernoulli mean of Chapter 2 by a single derivative rather than a sum. Differentiating again, $A''(\eta) = \sigma(\eta)(1 - \sigma(\eta)) = \mu(1 - \mu)$, which is the Bernoulli variance. The derivative of the log-partition function knows all the moments. ♢

3.5 Maximum likelihood is moment matching

Maximum likelihood in an exponential family has a single clean form, and Theorem 3.3 delivers it in two lines. Take the logarithm of the likelihood Eq. (3.9), keeping only the parameter-dependent terms and using $\log g(\boldsymbol{\eta}) = -A(\boldsymbol{\eta})$:

$$\ell(\boldsymbol{\eta}) = \boldsymbol{\eta}^{\top} \sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n) - N\, A(\boldsymbol{\eta}) + \text{const.}$$

Set the gradient to zero, and substitute $\nabla A(\boldsymbol{\eta}) = \mathbb{E}[\boldsymbol{u}(\boldsymbol{x})]$ from Eq. (3.10):

$$\nabla \ell(\boldsymbol{\eta}) = \sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n) - N\, \nabla A(\boldsymbol{\eta}) = \mathbf{0} \implies \mathbb{E}_{\hat{\boldsymbol{\eta}}}[\boldsymbol{u}(\boldsymbol{x})] = \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n).$$

Maximum likelihood as moment matching

The maximum likelihood estimate sets the model's expected sufficient statistic equal to the empirical average of that statistic,

$$\mathbb{E}_{\hat{\boldsymbol{\eta}}}[\boldsymbol{u}(\boldsymbol{x})] = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n). \tag{3.11}$$

"Choose the parameter under which the model expects, on average, exactly what you saw."

Every maximum likelihood estimate of Chapter 2 is a special case. For the Bernoulli, $\boldsymbol{u}(x) = x$ and $\mathbb{E}[x] = \mu$, so moment matching reads $\hat{\mu} = \frac{1}{N}\sum_n x_n = m/N$, the head fraction. For the Gaussian, $\boldsymbol{u}(x) = (x, x^2)$, so $\hat{\mu} = \frac{1}{N}\sum_n x_n$ and $\mathbb{E}[x^2] = \frac{1}{N}\sum_n x_n^2$; the second equation, with $\mathbb{E}[x^2] = \mu^2 + \sigma^2$, gives $\hat{\sigma}^2 = \frac{1}{N}\sum_n x_n^2 - \hat{\mu}^2$, the sample variance. One principle, Eq. (3.11), reproduces them all.

3.6 Conjugate priors, for the whole family at once

The conjugacy that felt like a series of lucky coincidences in Chapter 2 is, in fact, guaranteed for every exponential family by the structure of Eq. (3.1). The trick is to choose a prior that has the same algebraic shape as the likelihood, so that multiplying them just adds exponents.

Theorem 3.5 (Conjugate prior for an exponential family). For the likelihood Eq. (3.1), the prior

$$p(\boldsymbol{\eta} \mid \boldsymbol{\chi}, \nu) \propto g(\boldsymbol{\eta})^{\nu} \exp\!\big(\nu\, \boldsymbol{\chi}^{\top}\boldsymbol{\eta}\big) \tag{3.12}$$

is conjugate: after observing $\mathcal{D} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, the posterior is of the same form, with updated hyperparameters

$$\nu \;\to\; \nu + N, \qquad \nu\boldsymbol{\chi} \;\to\; \nu\boldsymbol{\chi} + \sum_{n=1}^{N} \boldsymbol{u}(\boldsymbol{x}_n).$$

Proof. Multiply the prior Eq. (3.12) by the likelihood Eq. (3.9), keeping only the $\boldsymbol{\eta}$-dependent factors:

$$p(\boldsymbol{\eta} \mid \mathcal{D}) \propto \underbrace{g(\boldsymbol{\eta})^{\nu} e^{\nu \boldsymbol{\chi}^{\top}\boldsymbol{\eta}}}_{\text{prior}} \cdot \underbrace{g(\boldsymbol{\eta})^{N} e^{\boldsymbol{\eta}^{\top}\sum_n \boldsymbol{u}(\boldsymbol{x}_n)}}_{\text{likelihood}} = g(\boldsymbol{\eta})^{\,\nu+N} \exp\!\Big(\boldsymbol{\eta}^{\top}\big(\nu\boldsymbol{\chi} + \textstyle\sum_n \boldsymbol{u}(\boldsymbol{x}_n)\big)\Big).$$

This is Eq. (3.12) again, with ν replaced by $\nu + N$ and $\nu\boldsymbol{\chi}$ replaced by $\nu\boldsymbol{\chi} + \sum_n \boldsymbol{u}(\boldsymbol{x}_n)$. The posterior is in the same family as the prior, which is the definition of conjugacy. ∎

The interpretation is the pseudo-count picture of Chapter 2, now explained: the hyperparameter ν is a number of *prior pseudo-observations*, and $\nu\boldsymbol{\chi}$ is the sufficient-statistic total those pseudo-observations would have produced. Updating just adds the real data's count N to ν and the real data's sufficient-statistic sum to $\nu\boldsymbol{\chi}$. The Beta prior on a Bernoulli is Eq. (3.12) with the pseudo-counts a, b; the Dirichlet on a categorical is the same with pseudo-counts $\boldsymbol{\alpha}$; the Gaussian prior on a Gaussian mean is the same again. One theorem subsumes the entire conjugacy table of Chapter 2.

3.7 The bridge to logistic regression

The inverse links of Section 3.2 are the reason the classifiers of Part III look the way they do. A *generalized linear model* takes an exponential-family output distribution and makes its natural parameter a linear function of the input, $\boldsymbol{\eta} = \boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x})$. The inverse link then turns that linear score into the distribution's parameter. Three choices of output family give three familiar models:

- a Gaussian output with $\eta = \boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x})$ and the identity link is *linear regression* (Chapter 5);
- a Bernoulli output with the sigmoid link, $\mu = \sigma(\boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x}))$, is *logistic regression* (Chapter 8);
- a categorical output with the softmax link is *softmax (multinomial logistic) regression* (Chapter 8).

In each case the model says, "the natural parameter of the output is linear in the features," and the exponential family supplies the function that converts that parameter back into a mean or a probability. This is why the sigmoid

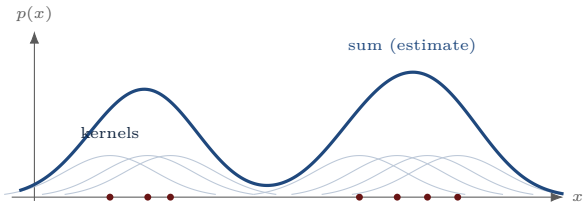


Figure 3.2. A Gaussian kernel (light) is centered on each data point (red), and their average (dark) is the density estimate. It is tall where points cluster and low between clusters, with no bins to place. The bandwidth h sets the width of each kernel and trades spikiness against oversmoothing.

and softmax are the canonical inverse links handed to us by Section 3.2 rather than arbitrary squashing functions chosen for convenience, and the cross-entropy loss that trains them is the negative log-likelihood of the corresponding exponential family.

3.8 When no parametric family fits: nonparametric density estimation

Sometimes the data are not shaped like any member of the exponential family, perhaps multimodal, perhaps lumpy, and forcing a single Gaussian on them would be a lie. The *nonparametric* methods estimate a density directly from the data, letting the shape of the estimate follow the shape of the sample, with a single smoothing knob in place of a fixed parametric form.

The simplest idea is the *histogram*: partition the space into bins of width h and let the density in a bin be the fraction of points it contains, divided by its width. The bin width h is the knob, and it already shows the central tension of the subject. Narrow bins track every wiggle of the data, including noise, and overfit; wide bins wash out real structure and underfit. This is the bias-variance tradeoff of Chapter 1, now controlling a density estimate.

Kernel density estimation removes the histogram's arbitrary bin edges by placing a small smooth bump, a *kernel*, at each data point and summing them.

Definition 3.6 (Kernel density estimator). Given points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N \in \mathbb{R}^D$ and a bandwidth $h > 0$, the Gaussian kernel density estimate is

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi h^2)^{D/2}} \exp\!\Big(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_n\|^2}{2h^2}\Big). \tag{3.13}$$

Each term is a Gaussian of width h centered on a data point, and the average of N of them is a smooth density that is tall where points cluster and low where they are sparse (Fig. 3.2). The bandwidth h plays the role of the bin width: too small and the estimate is a spiky comb with one peak per point (high variance), too large and distinct clusters merge into one broad mound (high bias). There is no parameter to fit by maximum likelihood here, only h to choose, typically by cross-validation (Chapter 4).

Example 3.7 (A kernel density estimate by hand). Take three points on the line, $x_1 = 0$, $x_2 = 1$, $x_3 = 3$, with bandwidth $h = 1$ (so each kernel is $\frac{1}{\sqrt{2\pi}} e^{-(x - x_n)^2/2}$). The estimate at the test point $x = 1$ is the average of the three kernels evaluated there:

$$p(1) = \frac{1}{3} \cdot \frac{1}{\sqrt{2\pi}} \Big(e^{-(1-0)^2/2} + e^{-(1-1)^2/2} + e^{-(1-3)^2/2} \Big) = \frac{1}{3\sqrt{2\pi}} \big(e^{-1/2} + 1 + e^{-2} \big).$$

Numerically $e^{-1/2} = 0.607$, $e^{-2} = 0.135$, so $p(1) = \frac{1}{3(2.507)}(1.742) \approx 0.232$. The nearby points at 0 and 1 contribute most, the far point at 3 contributes little, and the estimate needs no parametric assumption at all. ◊

A close relative replaces the fixed bandwidth with a fixed count: k-nearest-neighbor density estimation grows a region around $\boldsymbol{x}$ until it contains k points and sets the density inversely proportional to the region's volume. The same idea, used for prediction rather than density, is the k-nearest-neighbor classifier of Chapter 8: to label a new point, look at its k closest neighbors and take a vote.

3.9 Worked canonical forms, end to end

The derivations in Section 3.2 read off the four ingredients of Eq. (3.1) for each member. It is worth running the whole pipeline once on concrete numbers: name the natural parameter η, the sufficient statistic $\boldsymbol{u}(x)$, and the log-partition $A(\eta)$, then recover the mean and variance from the derivatives of A and check them against the textbook formulas. We do this for the Bernoulli and the fixed-variance Gaussian, the two cases that drive every classifier and regressor of Part III.

Example 3.8 (The Bernoulli, fully unpacked). Fix a coin with $\mu = 0.7$. From Eq. (3.4) the canonical pieces are the sufficient statistic $\boldsymbol{u}(x) = x$, the base $h(x) = 1$, the natural parameter $\eta = \log \frac{\mu}{1-\mu}$, and the log-partition $A(\eta) = \log(1 + e^{\eta})$. Putting in $\mu = 0.7$,

$$\eta = \log \frac{0.7}{0.3} = \log \tfrac{7}{3} \approx 0.847, \qquad e^{\eta} = \tfrac{7}{3} \approx 2.333. \tag{3.14}$$

Two cross-checks confirm the bookkeeping. First, the inverse link returns the probability: $\sigma(0.847) = 1/(1+e^{-0.847}) = 1/(1 + 0.429) = 0.7$, recovering μ. Second, the log-partition agrees with the normalizer $-\log(1-\mu)$, since

$$A(\eta) = \log\big(1 + \tfrac{7}{3}\big) = \log \tfrac{10}{3} \approx 1.204, \qquad -\log(1-\mu) = -\log 0.3 \approx 1.204, \tag{3.15}$$

the same number, as it must be because $g(\eta) = 1 - \mu = e^{-A(\eta)}$. The whole distribution is now pinned down by the single scalar $\eta \approx 0.847$, with every other quantity a fixed function of it. ◊

Example 3.9 (The fixed-variance Gaussian, fully unpacked). Hold the variance fixed at $\sigma^2 = 4$ and treat only the mean μ as unknown, the setting of Table 3.1. With σ^2 constant the x^2 term in Eq. (3.8) is no longer parameter-dependent, so it folds into the base measure and the sufficient statistic collapses to $\boldsymbol{u}(x) = x$ alone, with the scalar natural parameter and log-partition

$$\eta = \frac{\mu}{\sigma^2}, \qquad A(\eta) = \frac{\sigma^2 \eta^2}{2}. \tag{3.16}$$

The log-partition is read off by collecting the η-dependent part of the normalizer: the exponent contributes $-\mu^2/2\sigma^2 = -(\sigma^2\eta)^2/2\sigma^2 = -\sigma^2\eta^2/2$ to $\log g$, so $A = -\log g = \sigma^2\eta^2/2$. Take $\mu = 1$, so that

$$\eta = \frac{1}{4} = 0.25, \qquad A(\eta) = \frac{4\,(0.25)^2}{2} = \frac{4 \cdot 0.0625}{2} = 0.125. \tag{3.17}$$

The inverse link is the affine map $\mu = \sigma^2\eta$, here $4 \cdot 0.25 = 1$, returning the mean. Unlike the Bernoulli, the Gaussian's link is linear, which is the reason ordinary linear regression (a Gaussian output with $\eta = \boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x})$) needs no squashing function at its output. ◊

Example 3.10 (Mean and variance from A' and A'', with numbers). Continue the fixed-variance Gaussian of Example 3.9 with $\sigma^2 = 4$ and $\eta = 0.25$. Theorem 3.3 says the first two derivatives of the log-partition are the mean and variance of the sufficient statistic. Differentiating $A(\eta) = \sigma^2\eta^2/2$ twice,

$$A'(\eta) = \sigma^2\eta, \qquad A''(\eta) = \sigma^2. \tag{3.18}$$

Evaluating at $\eta = 0.25$ gives $A'(0.25) = 4 \cdot 0.25 = 1 = \mu$, the mean, and $A''(0.25) = 4 = \sigma^2$, the variance. Both come out without computing a single integral. The check against Example 3.4 is instructive: there the Bernoulli's $A''(\eta) = \mu(1-\mu)$ *depended* on η, because a Bernoulli's variance is tied to its mean, whereas here A'' is the constant σ^2, because a fixed-variance Gaussian's spread does not move with its mean. The curvature of the log-partition function reports the variance in both cases. ◊

3.10 A conjugate update with numbers

Theorem 3.5 promises that a conjugate update is just addition of pseudo-counts. The cleanest way to feel this is to run one Beta–Bernoulli update on numbers and watch the posterior mean settle between the prior and the data. This complements the abstract pseudo-count picture of Section 3.6 with an arithmetic instance, and it sets up the Gamma–Poisson update of the exercises.

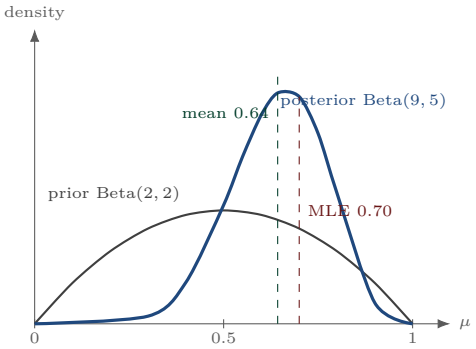


Figure 3.3. The mild prior Beta$(2,2)$ becomes the sharper posterior Beta$(9,5)$ after 7 heads in 10 flips. The posterior mean 0.64 is pulled back from the maximum likelihood estimate 0.70 by the prior's four pseudo-flips. The curves are the exact Beta densities, plotted on the μ-axis rescaled so that $\mu = 1$ falls at $x = 5$.

Example 3.11 (Beta–Bernoulli update, prior to posterior). Start from a mild prior Beta$(2,2)$, symmetric and gently favoring a fair coin ($a = 2$ pseudo-heads, $b = 2$ pseudo-tails, prior mean $\frac{2}{4} = 0.5$). Observe $m = 7$ heads in $N = 10$ flips, so $l = N - m = 3$ tails. For the Bernoulli the sufficient statistic is $\boldsymbol{u}(x) = x$, so the data's sufficient-statistic sum is the head count $\sum_n x_n = m = 7$. The conjugate update of Theorem 3.5 adds the data's counts to the prior's pseudo-counts,

$$\text{Beta}(2,2) \xrightarrow{\text{7 heads, 3 tails}} \text{Beta}(2+7,\ 2+3) = \text{Beta}(9,5). \tag{3.19}$$

The three standard summaries of μ then compare as

$$\hat{\mu}_{\text{ML}} = \frac{m}{N} = \frac{7}{10} = 0.70, \qquad \mathbb{E}[\mu \mid \mathcal{D}] = \frac{a'}{a'+b'} = \frac{9}{14} \approx 0.643, \qquad \hat{\mu}_{\text{MAP}} = \frac{a'-1}{a'+b'-2} = \frac{8}{12} \approx 0.667, \tag{3.20}$$

where $a' = 9$, $b' = 5$, and the MAP estimate is the posterior mode $(a'-1)/(a'+b'-2)$ of a Beta. The posterior mean 0.643 sits between the data's 0.70 and the prior's 0.50: the prior, worth four pseudo-flips, pulls the estimate toward fairness, while the ten real flips pull it toward 0.7. The MAP estimate 0.667 lands a little above the mean, since the Beta posterior is skewed (Fig. 3.3). With more data the four pseudo-flips would be swamped and all three summaries would converge on the empirical frequency. ◊

Remark 3.12 *(Conjugacy turns inference into counting).* Example 3.11 is the whole point of Theorem 3.5 on one line of arithmetic. No integral, no normalizing constant, no optimization: the posterior Beta$(9,5)$ is read off by adding $(7,3)$ to $(2,2)$, and the posterior mean is a ratio of the resulting counts. The same recipe runs the Dirichlet–categorical update (add the class counts to the pseudo-counts) and, in the exercises, the Gamma–Poisson update (add the event total to the shape and the number of intervals to the rate). Whenever the prior is the family's conjugate prior, Bayesian updating reduces to bookkeeping on the sufficient statistic.

Notes and References

The exponential family, its sufficient statistics, the moment-generating role of the log-partition function, and the general conjugate prior all follow Section 2.4 of Bishop [3]; the nonparametric methods of Section 3.8 are Section 2.5. The view of the sigmoid and softmax as the inverse links of the Bernoulli and categorical, and of logistic and softmax regression as generalized linear models, is developed further in Chapters 7 to 8; the classic reference for generalized linear models is McCullagh and Nelder. The data-reduction property of sufficient statistics is the practical reason streaming and large-scale estimation in an exponential family are cheap. Theorem 3.3 is the entry point to the wider theory of exponential families in statistics, where $A(\boldsymbol{\eta})$ is studied as a cumulant generating function.

Exercises

Exercise 3.1. Write the Bernoulli distribution in exponential-family form and identify $u(x)$, η, $h(x)$, and $g(\eta)$. Show that the inverse map from η to μ is the logistic sigmoid.

Solution. From $\mathrm{Bern}(x \mid \mu) = (1-\mu)\exp(x \log \frac{\mu}{1-\mu})$ we read $u(x) = x$, $\eta = \log \frac{\mu}{1-\mu}$, $h(x) = 1$, and $g(\eta) = 1 - \mu$. Solving $e^\eta = \mu/(1-\mu)$ for μ gives $\mu = e^\eta/(1+e^\eta) = 1/(1+e^{-\eta}) = \sigma(\eta)$, and then $g(\eta) = 1 - \mu = 1/(1+e^\eta)$, so $A(\eta) = -\log g = \log(1+e^\eta)$. □

Exercise 3.2. Put the univariate Gaussian $\mathcal{N}(x \mid \mu, \sigma^2)$ into exponential-family form. What are the sufficient statistics, and what does this say about which numbers you must keep from a data set to fit a Gaussian?

Solution. Expanding the exponent, $-(x-\mu)^2/2\sigma^2 = (\mu/\sigma^2)x - (1/2\sigma^2)x^2 - \mu^2/2\sigma^2$, so $\boldsymbol{u}(x) = (x, x^2)$ with natural parameters $\boldsymbol{\eta} = (\mu/\sigma^2,\ -1/2\sigma^2)$ and $h(x) = 1/\sqrt{2\pi}$. The sufficient statistic for N points is $(\sum_n x_n,\ \sum_n x_n^2)$, so the sum of the values and the sum of their squares are all you need; the individual data points can be discarded. □

Exercise 3.3. Use $A(\eta) = \log(1+e^\eta)$ for the Bernoulli to compute $\mathbb{E}[x]$ and $\mathrm{Var}(x)$ by differentiation, and check they match the Bernoulli moments of Chapter 2.

Solution. $A'(\eta) = e^\eta/(1+e^\eta) = \sigma(\eta) = \mu = \mathbb{E}[x]$, and $A''(\eta) = \sigma(\eta)(1-\sigma(\eta)) = \mu(1-\mu) = \mathrm{Var}(x)$, by Theorem 3.3. These are exactly the Bernoulli mean μ and variance $\mu(1-\mu)$. □

Exercise 3.4. Starting from Eq. (3.11), derive the maximum likelihood estimates of the mean and variance of a Gaussian.

Solution. The Gaussian sufficient statistics are $u(x) = (x, x^2)$, so moment matching gives $\mathbb{E}[x] = \frac{1}{N}\sum_n x_n$ and $\mathbb{E}[x^2] = \frac{1}{N}\sum_n x_n^2$. The first yields $\hat{\mu} = \frac{1}{N}\sum_n x_n$. For the second, $\mathbb{E}[x^2] = \mu^2 + \sigma^2$, so $\hat{\sigma}^2 = \frac{1}{N}\sum_n x_n^2 - \hat{\mu}^2 = \frac{1}{N}\sum_n (x_n - \hat{\mu})^2$, the sample variance. □

Exercise 3.5. Using Theorem 3.5, explain how the Beta posterior $\mathrm{Beta}(a+m, b+l)$ of Chapter 2 arises as the general exponential-family conjugate update, and identify the pseudo-counts.

Solution. The Bernoulli has $u(x) = x$, so the data's sufficient-statistic sum is the head count $\sum_n x_n = m$. The conjugate prior Eq. (3.12) updates $\nu \to \nu + N$ and $\nu\boldsymbol{\chi} \to \nu\boldsymbol{\chi} + m$. Matching to the Beta, the prior pseudo-counts are a heads and b tails; observing m heads and $l = N - m$ tails sends $a \to a + m$ and $b \to b + l$, giving the posterior $\mathrm{Beta}(a+m, b+l)$. The hyperparameters a, b are the pseudo-observation counts that the general theorem calls $\nu\boldsymbol{\chi}$ and ν. □

Exercise 3.6. Put the Poisson distribution $\mathrm{Poisson}(x \mid \lambda) = \lambda^x e^{-\lambda}/x!$, for counts $x \in \{0, 1, 2, \dots\}$, into exponential-family form. Identify the sufficient statistic $\boldsymbol{u}(x)$, the base measure $h(x)$, the natural parameter η, and the log-partition $A(\eta)$. Then use $A(\eta)$ and Theorem 3.3 to confirm the Poisson mean and variance are both λ.

Solution. Write the mass as the exponential of its logarithm:

$$\mathrm{Poisson}(x \mid \lambda) = \frac{1}{x!}\exp\big(x \log \lambda - \lambda\big).$$

Comparing with Eq. (3.1), the base is $h(x) = 1/x!$, the sufficient statistic is $\boldsymbol{u}(x) = x$, and the natural parameter is $\eta = \log \lambda$, so the inverse link is $\lambda = e^\eta$. The normalizer carries the $e^{-\lambda}$ factor, $g(\eta) = e^{-\lambda} = e^{-e^\eta}$, so the log-partition is

$$A(\eta) = -\log g(\eta) = \lambda = e^\eta.$$

Differentiating, $A'(\eta) = e^\eta = \lambda$ and $A''(\eta) = e^\eta = \lambda$, so by Theorem 3.3 both the mean $\mathbb{E}[x]$ and the variance $\mathrm{Var}(x)$ equal λ, the well-known equal-mean-and-variance property of the Poisson. These match the Poisson row of Table 3.1. □

Exercise 3.7. The conjugate prior for the Poisson rate λ is the Gamma distribution. In its shape-rate form, $\mathrm{Gam}(\lambda \mid a, b) \propto \lambda^{a-1} e^{-b\lambda}$ for $\lambda > 0$, with mean a/b. Suppose you place a prior $\mathrm{Gam}(2, 1)$ on λ and then observe counts $x_1 = 3$, $x_2 = 5$, $x_3 = 4$ in $N = 3$ equal time intervals. Find the posterior, its mean, and compare that mean with the maximum likelihood estimate $\hat{\lambda}_{\mathrm{ML}} = \frac{1}{N}\sum_n x_n$.

Solution. The Poisson likelihood for N counts collapses, as in Eq. (3.9), through the sufficient-statistic sum $s = \sum_n x_n$:

$$\prod_{n=1}^{N} \frac{\lambda^{x_n} e^{-\lambda}}{x_n!} \;\propto\; \lambda^{\,s}\, e^{-N\lambda}, \qquad s = \sum_{n=1}^{N} x_n.$$

Multiplying by the prior $\mathrm{Gam}(2, 1) \propto \lambda^{a-1} e^{-b\lambda}$ with $a = 2$, $b = 1$ adds the exponents,

$$p(\lambda \mid \mathcal{D}) \propto \lambda^{\,a-1+s}\, e^{-(b+N)\lambda} = \lambda^{\,(a+s)-1}\, e^{-(b+N)\lambda},$$

which is again a Gamma, $\mathrm{Gam}(a + s,\ b + N)$. This is exactly the general update of Theorem 3.5: the shape absorbs the event total s and the rate absorbs the count of intervals N. Here $s = 3 + 5 + 4 = 12$ and $N = 3$, so the posterior is

$$\mathrm{Gam}(2 + 12,\ 1 + 3) = \mathrm{Gam}(14, 4), \qquad \mathbb{E}[\lambda \mid \mathcal{D}] = \frac{a + s}{b + N} = \frac{14}{4} = 3.5.$$

The maximum likelihood estimate is $\hat{\lambda}_{\mathrm{ML}} = s/N = 12/3 = 4$. The posterior mean 3.5 sits between the prior mean $a/b = 2$ and the data's 4, pulled down from the MLE because the prior acts like one extra interval ($b = 1$) carrying $a = 2$ prior events; with more intervals the prior's single pseudo-interval would be swamped and the posterior mean would approach 4. □

Exercise 3.8. For data $x_1 = 0$, $x_2 = 2$ with a Gaussian kernel of bandwidth $h = 1$, write the kernel density estimate $p(x)$ and evaluate it at $x = 1$. What happens to the estimate as $h \to 0$ and as $h \to \infty$?

Solution. The estimate is $p(x) = \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}} \left(e^{-(x)^2/2} + e^{-(x-2)^2/2} \right)$. At $x = 1$, both exponents are $-(1)^2/2 = -1/2$, so $p(1) = \frac{1}{\sqrt{2\pi}} e^{-1/2} = (0.399)(0.607) \approx 0.242$. As $h \to 0$ each kernel becomes an infinitely tall spike at its data point, so the estimate is a comb of spikes (extreme overfitting); as $h \to \infty$ the kernels flatten and merge into one very broad bump that ignores the two-point structure (extreme oversmoothing). □

CHAPTER 4

Learning Theory

There are three questions to ask of any learning machine: does it generalize, is it robust, and can we explain it? This chapter is about the first.

after Professor Peter Chin, ENGS 106

Chapter 1 drew the central picture of the subject: training error falls as a model grows more complex, but test error follows a U, and the gap between the two is overfitting. That chapter left the gap qualitative. This one makes it a theorem. The question is precise: if a model fits the training data well, what can we *guarantee* about its error on data it has never seen? The answer rests on a single phenomenon, *concentration*: the average of many independent random variables clusters tightly around its mean. We build the concentration tools in order of sharpness, Markov, Chebyshev, then the exponentially tight Chernoff and Hoeffding bounds, and then spend them. Combined with the union bound they buy a generalization guarantee for empirical risk minimization, a sample-complexity rule, the bias-variance tradeoff in quantitative form, and the VC dimension, the honest measure of an infinite model's capacity.

Roadmap

Suggested reading: Shalev-Shwartz and Ben-David [34], Chapters 2–6; Bishop [3], Section 1.3 and Deisenroth et al. [9], Section 8.2 for the framing.

We cover: the learning setup and empirical risk (Section 4.1); Markov, Chebyshev, and why polynomial bounds fail (Sections 4.2 to 4.4); the Chernoff and Hoeffding bounds (Section 4.5); generalization via the union bound, sample complexity, and PAC learning (Section 4.6); the finite-precision bridge to infinite classes (Section 4.7); the bias-variance tradeoff (Section 4.8); the VC dimension (Section 4.9); and model selection by cross-validation (Section 4.10).

4.1 The learning setup

Fix a hypothesis h mapping inputs to labels, and a training set $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ drawn independently from an unknown data distribution $\mathcal{D}$. The two quantities of Chapter 1 reappear with names we will keep: the *empirical risk* (training error) and the *true risk* (generalization error),

$$\widehat{\varepsilon}_S(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{1}[h(\boldsymbol{x}_i) \neq y_i], \qquad \varepsilon(h) = \mathbb{P}_{(\boldsymbol{x},y)\sim\mathcal{D}}\big[h(\boldsymbol{x}) \neq y\big], \tag{4.1}$$

the fraction of training examples misclassified, and the probability of misclassifying a fresh example. Learning by *empirical risk minimization* (ERM) returns the hypothesis with smallest training error, $\hat{h} = \arg\min_{h\in\mathcal{H}} \widehat{\varepsilon}_S(h)$, over a hypothesis class $\mathcal{H}$. We can compute $\widehat{\varepsilon}_S$ but only want ε small, and the whole theory is about controlling their difference. The engine that makes it possible is that, for a fixed h, the indicators $Z_i = \mathbf{1}[h(\boldsymbol{x}_i) \neq y_i]$ are independent Bernoulli variables with mean $\varepsilon(h)$, so the training error is their sample mean. Concentration of that mean is what we now quantify.

4.2 Markov's inequality

The crudest tail bound uses only the mean of a nonnegative variable.

Theorem 4.1 (Markov's inequality). If $X \geq 0$ and $a > 0$, then $\mathbb{P}(X \geq a) \leq \mathbb{E}[X]/a$.

Proof. Split the expectation at the threshold a. Both pieces are nonnegative because $X \geq 0$, and on the event $\{X \geq a\}$ the integrand is at least a:

$$\mathbb{E}[X] = \mathbb{E}[X\,\mathbf{1}[X < a]] + \mathbb{E}[X\,\mathbf{1}[X \geq a]] \geq \mathbb{E}[X\,\mathbf{1}[X \geq a]] \geq a\,\mathbb{E}[\mathbf{1}[X \geq a]] = a\,\mathbb{P}(X \geq a).$$

Divide by a. ∎

Example 4.2 (Server response time). If requests take $\mathbb{E}[X] = 2$ minutes on average, then knowing nothing else, $\mathbb{P}(X \geq 10) \leq 2/10 = 0.2$: at most a fifth of requests can take ten minutes or more. The bound is weak, but it asks almost nothing of the distribution. ◊

4.3 Chebyshev's inequality

Bringing in the variance sharpens the bound and makes it two-sided.

Theorem 4.3 (Chebyshev's inequality). If X has mean μ and variance σ^2, then for any $a > 0$, $\mathbb{P}(|X - \mu| \geq a) \leq \sigma^2/a^2$.

Proof. Apply Markov to the nonnegative variable $Y = (X-\mu)^2$, whose mean is $\mathbb{E}[Y] = \mathrm{Var}(X) = \sigma^2$. Since $|X-\mu| \geq a$ exactly when $Y \geq a^2$,

$$\mathbb{P}(|X - \mu| \geq a) = \mathbb{P}(Y \geq a^2) \leq \frac{\mathbb{E}[Y]}{a^2} = \frac{\sigma^2}{a^2}.$$

∎

Applied to a sample mean of m independent variables, whose variance is σ^2/m, Chebyshev gives $\mathbb{P}(|\bar{X} - \mu| \geq a) \leq \sigma^2/(ma^2) \to 0$: the sample mean converges to the true mean, which is the weak law of large numbers. This already shows training error approaches true error, but far too slowly for the guarantees we want.

4.4 Why polynomial bounds are not enough

Markov decays like $1/a$ and Chebyshev like $1/a^2$, both polynomially. Learning theory needs exponential confidence. Take $m = 1000$ fair coin flips, $X \sim \mathrm{Bin}(1000, \frac{1}{2})$ with $\mu = 500$, $\sigma^2 = 250$, and ask for $\mathbb{P}(X \geq 600)$:

$$\text{Markov: } \frac{500}{600} \approx 0.83, \qquad \text{Chebyshev: } \mathbb{P}(|X - 500| \geq 100) \leq \frac{250}{100^2} = 0.025.$$

Markov is useless and Chebyshev still allows 2.5%, while the true probability is about 10^{-10}. We need a bound that decays *exponentially* in the sample size. That is the Chernoff bound, whose engine is the moment-generating function. Figure 4.1 shows the qualitative gap.

4.5 Moment-generating functions and the Chernoff bound

Definition 4.4 (Moment-generating function). The *moment-generating function* of X is $M_X(s) = \mathbb{E}[e^{sX}]$, for s where the expectation is finite.

Two properties make it the right tool. For a Bernoulli $Y \sim \mathrm{Bern}(p)$,

$$M_Y(s) = (1 - p) + p\,e^s = 1 + p(e^s - 1) \leq e^{p(e^s - 1)}, \tag{4.2}$$

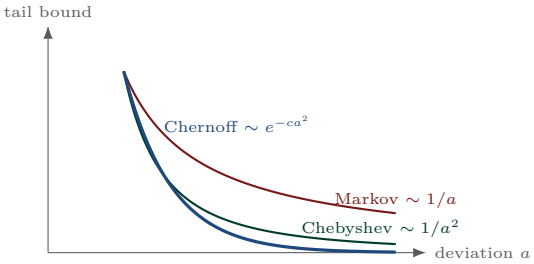


Figure 4.1. Markov decays like $1/a$, Chebyshev like $1/a^2$, but the Chernoff bound decays exponentially. Exponential concentration is what makes high-confidence finite-sample guarantees possible. (Schematic.)

Inequality	Assumes	Bounds	Decay
Markov	$X \geq 0$, mean	$\mathbb{P}(X \geq a) \leq \mathbb{E}[X]/a$	$1/a$
Chebyshev	mean, variance	$\mathbb{P}(\lvert X-\mu\rvert \geq a) \leq \sigma^2/a^2$	$1/a^2$
Chernoff	sum of indep. Bernoulli	$\leq e^{-\mu\delta^2/3}$	exponential
Hoeffding	indep., bounded $[0,1]$	$\leq 2e^{-2m\gamma^2}$	exponential

Table 4.1. In increasing order of assumptions and sharpness. More structure (a variance, then independence and boundedness) buys faster decay. Chernoff and Hoeffding are the exponential bounds learning theory runs on.

using $1+x \leq e^x$. And the MGF of a sum of independent variables factorizes, because the exponential of a sum is a product and the expectation of a product of independent terms is the product of expectations:

$$X=\sum_{i=1}^{m} X_i \text{ independent} \implies M_X(s)=\mathbb{E}\Big[e^{s\sum_i X_i}\Big]=\prod_{i=1}^{m}\mathbb{E}[e^{sX_i}]=\prod_{i=1}^{m} M_{X_i}(s). \tag{4.3}$$

For independent Bernoulli variables with means summing to μ, the two combine to $M_X(s) \leq e^{(e^s-1)\mu}$.

Theorem 4.5 (Chernoff bound). Let $X=\sum_{i=1}^{m} X_i$ be a sum of independent Bernoulli variables with $\mu=\mathbb{E}[X]$. For any $\delta>0$,

$$\mathbb{P}\big(X \geq (1+\delta)\mu\big) \leq \exp\Big(-\mu\big[(1+\delta)\ln(1+\delta)-\delta\big]\Big) \leq e^{-\mu\delta^2/3}. \tag{4.4}$$

Proof. The idea is to apply Markov not to X but to e^{sX}, which amplifies large deviations. For any $s>0$, monotonicity of $e^{s(\cdot)}$ and Markov give

$$\mathbb{P}\big(X \geq (1+\delta)\mu\big)=\mathbb{P}\big(e^{sX} \geq e^{s(1+\delta)\mu}\big) \leq \frac{\mathbb{E}[e^{sX}]}{e^{s(1+\delta)\mu}}=\frac{M_X(s)}{e^{s(1+\delta)\mu}}.$$

Bound the MGF by $M_X(s) \leq e^{(e^s-1)\mu}$, so $\mathbb{P}(X \geq (1+\delta)\mu) \leq \exp(\mu[(e^s-1)-s(1+\delta)])$. This holds for every $s>0$, so minimize the exponent $f(s)=(e^s-1)-s(1+\delta)$. Setting $f'(s)=e^s-(1+\delta)=0$ gives the optimal $s=\ln(1+\delta)$, where $e^s-1=\delta$ and $s(1+\delta)=(1+\delta)\ln(1+\delta)$, yielding the exponent $\mu[\delta-(1+\delta)\ln(1+\delta)]$. The looser form $e^{-\mu\delta^2/3}$ follows from $(1+\delta)\ln(1+\delta)-\delta \geq \delta^2/3$ on $\delta \in (0,1]$. ∎

For bounded variables there is a closely related additive form, *Hoeffding's inequality*, which is the one we use because classification errors are bounded $[0,1]$ averages: for independent $Z_1,\dots,Z_m \in [0,1]$ with sample mean $\bar{Z}$,

$$\mathbb{P}\big(\lvert\bar{Z}-\mathbb{E}[\bar{Z}]\rvert \geq \gamma\big) \leq 2e^{-2m\gamma^2}. \tag{4.5}$$

Example 4.6 (The coin, revisited). For $X \sim \mathrm{Bin}(1000,\tfrac{1}{2})$ and $X \geq 600=(1+0.2)\mu$, the Chernoff bound gives $\exp(-500[1.2\ln 1.2-0.2]) \approx 8.3\times 10^{-5}$, against Chebyshev's 0.025: hundreds of times tighter, and far closer to the truth ($\approx 10^{-10}$). ◊

4.6 Generalization: from one hypothesis to a class

For a single fixed h, the error indicators are independent Bernoulli with mean $\varepsilon(h)$, so the training error is their sample mean and $\mathbb{E}[\widehat{\varepsilon}_S(h)] = \varepsilon(h)$: training error is an unbiased estimate of true error. Hoeffding then controls the gap for that one h, $\mathbb{P}(|\widehat{\varepsilon}_S(h) - \varepsilon(h)| > \gamma) \le 2e^{-2m\gamma^2}$.

But learning chooses $\hat{h}$ *after* seeing the data, so the bound must hold *simultaneously* for every hypothesis in the class, or the search could cherry-pick one that fits by luck. The *union bound*, $\mathbb{P}(\bigcup_i A_i) \le \sum_i \mathbb{P}(A_i)$, pays exactly that price. For a finite class with $|\mathcal{H}| = k$,

$$\mathbb{P}\Big(\exists h \in \mathcal{H} :\ |\widehat{\varepsilon}_S(h) - \varepsilon(h)| > \gamma\Big) \le k \cdot 2e^{-2m\gamma^2}. \tag{4.6}$$

Setting the right side to δ and solving for γ gives the *uniform convergence* guarantee: with probability at least $1 - \delta$,

$$|\widehat{\varepsilon}_S(h) - \varepsilon(h)| \le \sqrt{\frac{\ln(2k/\delta)}{2m}} \quad \text{for all } h \in \mathcal{H}. \tag{4.7}$$

Proposition 4.7 (Sample complexity and the ERM guarantee). To make the uniform gap at most γ with confidence $1 - \delta$, it suffices that

$$m \ge \frac{\ln(2k/\delta)}{2\gamma^2}. \tag{4.8}$$

Then the ERM hypothesis is near-optimal, $\varepsilon(\hat{h}) \le \min_{h \in \mathcal{H}} \varepsilon(h) + 2\gamma$.

Why the ERM bound holds. Let h^* be the best hypothesis in the class. Uniform convergence applied twice gives $\varepsilon(\hat{h}) \le \widehat{\varepsilon}_S(\hat{h}) + \gamma$ (the gap for $\hat{h}$), then $\widehat{\varepsilon}_S(\hat{h}) \le \widehat{\varepsilon}_S(h^*)$ (because $\hat{h}$ minimizes training error), then $\widehat{\varepsilon}_S(h^*) \le \varepsilon(h^*) + \gamma$ (the gap for h^*). Chaining, $\varepsilon(\hat{h}) \le \varepsilon(h^*) + 2\gamma$. ∎

The sample size grows only *logarithmically* in the number of hypotheses k and in $1/\delta$, but quadratically in $1/\gamma$: with $k = 1000$, $\gamma = 0.05$, $\delta = 0.05$, Eq. (4.8) gives $m \ge \ln(40000)/(2 \cdot 0.0025) \approx 2120$. A guarantee of this shape, "with probability at least $1 - \delta$ (*probably*) the error is within γ of the best (*approximately correct*)," is *PAC* learning, after Valiant's "probably approximately correct" [40]. The two knobs are the confidence δ and the accuracy γ; the whole edifice rests on one idea, Hoeffding concentration for a fixed h, made uniform by the union bound.

4.7 The finite-precision bridge to infinite classes

The bound Eq. (4.7) contains $\ln k$, which is useless for an infinite class such as all linear classifiers. There is a quick and surprisingly informative fix, due to a counting argument Professor Chin emphasizes. A computer never really stores a continuum: a parameter held in a 64-bit floating-point number takes at most 2^{64} distinct values. A model with d real parameters therefore realizes at most

$$k \le \left(2^{64}\right)^d = 2^{64d}$$

distinct hypotheses. Substituting this k into the sample complexity Eq. (4.8),

$$m \gtrsim \frac{1}{2\gamma^2}\left(\ln\frac{2}{\delta} + 64d\ln 2\right) = \mathcal{O}(d). \tag{4.9}$$

The required number of samples is *linear in the number of parameters.* This is both a comfort and a clue. The comfort is that a model with a million parameters can be learned from a number of samples proportional to a million rather than exponential in it. The clue is that what matters is some notion of "effective number of parameters," not the literal count of representable hypotheses. The bit-counting bound is loose, because two different bit patterns often give the same classifier. The right, precision-free measure of capacity is the VC dimension of Section 4.9, which counts not hypotheses but distinguishable labelings.

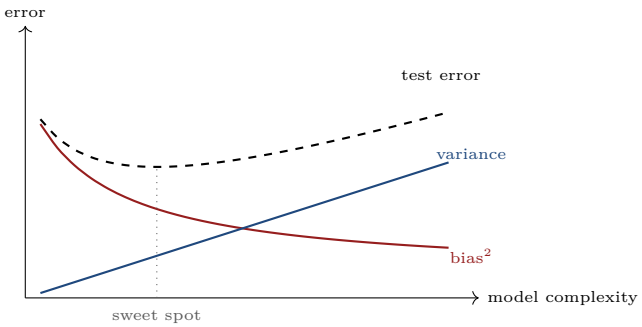


Figure 4.2. As the class grows more complex, bias falls (it can fit the truth) but variance rises (it can fit the noise). Test error is minimized at intermediate complexity, the quantitative version of the U-curve of Chapter 1.

4.8 The bias-variance tradeoff

The bound Eq. (4.7) says a richer class (larger k) costs more samples, hinting at a tension between fitting power and reliability. Decomposing the expected squared error of a prediction $\hat{y}$ for a target y makes it exact. Writing $\mathbb{E}[\hat{y}]$ for the average prediction over training sets,

$$\mathbb{E}\big[(y-\hat{y})^2\big] = \underbrace{(\mathbb{E}[\hat{y}]-y)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}[(\hat{y}-\mathbb{E}[\hat{y}])^2]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}}. \tag{4.10}$$

A simple class has high bias (it underfits, unable to represent the truth) but low variance; a complex class has low bias but high variance (it overfits, chasing the noise in the particular sample). The union-bound term $\sqrt{\ln k/2m}$ in Eq. (4.7) is exactly a variance penalty that grows with capacity, so test error behaves like $\text{bias}^2 + \mathcal{O}(\ln k/m) + \text{noise}$, minimized at intermediate complexity (Fig. 4.2). This is the quantitative form of the U-shaped test-error curve of Chapter 1, and more data lets you afford a richer class safely. The same decomposition returns in Chapter 5 for the squared loss of regression.

4.9 The VC dimension

The right capacity measure for an infinite class is not how many hypotheses it has but how many points it can label in *every* possible way.

Definition 4.8 (Shattering and VC dimension). A class $\mathcal{H}$ *shatters* a set of points $\{\boldsymbol{x}_1,\dots,\boldsymbol{x}_n\}$ if for every one of the 2^n binary labelings there is some $h\in\mathcal{H}$ realizing it. The *Vapnik-Chervonenkis dimension* $d_{\text{VC}}(\mathcal{H})$ is the size of the largest set it can shatter.

Example 4.9 (Linear classifiers in the plane). For lines in $\mathbb{R}^2$, $h_{\boldsymbol{w},b}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^\top\boldsymbol{x}+b)$, three points in general position can be split in all $2^3 = 8$ ways (Fig. 4.3), but four points cannot: the XOR labeling, with the two diagonal pairs in opposite classes, defeats every line (Fig. 4.4). Hence $d_{\text{VC}} = 3$. In general, linear classifiers in $\mathbb{R}^d$ have $d_{\text{VC}} = d+1$, matching the d weights plus one bias. The literal parameter count and the VC dimension agree here, which is why the bit-counting bound of Section 4.7 had the right linear-in-d form. ◊

Example 4.10 (Intervals on the line have $d_{\text{VC}} = 2$). Let $\mathcal{H} = \{\mathbf{1}[x\in[a,b]]\}$ label + inside an interval, − outside. Two points $x_1 < x_2$ are shattered: all four labelings are realizable, $(-,-)$ by an interval off to the left, $(+,-)$ by an interval covering only x_1, $(-,+)$ by one covering only x_2, and $(+,+)$ by $[x_1,x_2]$. But three points $x_1<x_2<x_3$ cannot be: the labeling $(+,-,+)$ needs an interval that contains x_1 and x_3 yet excludes the x_2 between them, impossible for a connected set. So $d_{\text{VC}} = 2$, matching the two parameters a, b. ◊

The VC dimension replaces $\ln k$ in the generalization bound: for a class of VC dimension d_{VC}, with high probability the gap between training and test error is $\mathcal{O}\big(\sqrt{d_{\text{VC}}\ln(m/d_{\text{VC}})/m}\big)$ [41]. A finite VC dimension makes even an infinite class learnable, and it is the honest capacity in the bias-variance picture: larger d_{VC} lowers bias but demands more data to keep variance in check. Table 4.2 collects common values.

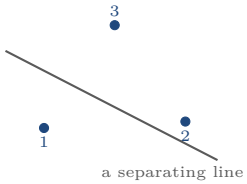


Figure 4.3. Three points in general position are shattered by lines: tilting and shifting the line realizes all $2^3 = 8$ labelings.

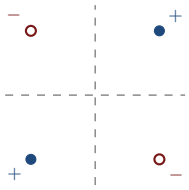


Figure 4.4. The XOR labeling defeats every line: the two + points lie on one diagonal, the two − on the other. So four points cannot be shattered.

Hypothesis class	d_{VC}	**Free parameters**
Thresholds $\mathbf{1}[x \geq t]$ on $\mathbb{R}$	1	1 (t)
Intervals $\mathbf{1}[x \in [a, b]]$ on $\mathbb{R}$	2	2 (a, b)
Linear classifiers in $\mathbb{R}^d$	$d + 1$	$d + 1$ ($\boldsymbol{w}, b$)
Axis-aligned rectangles in $\mathbb{R}^2$	4	4 (sides)

Table 4.2. VC dimension usually tracks the number of free parameters, formalizing the intuition that capacity is "degrees of freedom." Replacing $\ln k$ by d_{VC} makes an infinite class learnable.

4.10 Model selection by cross-validation

The theory says complexity must be tuned to the data, and in practice the tuning is done by holding data out. The training set alone cannot choose between models, because more complex models always have smaller training error, the monotone curve of Chapter 1. The remedy is to estimate the true error on data the model never saw during fitting.

The simplest protocol splits the data three ways: a *training* set to fit each candidate model, a *validation* set to compare them and pick the best complexity, and a *test* set, touched once at the very end, to report an honest final error. The validation error stands in for the true error in the model-selection step, exactly the quantity Eq. (4.7) controls, while the held-out test set guards against having implicitly fit the validation set through many comparisons.

When data is scarce, *K-fold cross-validation* reuses it more efficiently: partition the training data into K equal folds, and for each candidate model train on $K - 1$ folds and validate on the held-out fold, rotating through all K choices, then average the K validation errors. Every point serves once for validation and $K - 1$ times for training, so the estimate uses all the data while never validating on a point used to fit. Having chosen the best complexity, one refits on all the training data. A common warning closes the loop: choosing hyperparameters by minimizing error on the same set used to report the final number reintroduces exactly the optimistic bias the theory warns against, which is why the test set is spent only once.

Notes and References

The concentration inequalities and the generalization bounds of this chapter are developed in detail by Shalev-Shwartz and Ben-David [34], the standard modern text on learning theory, and summarized in Section 1.3 of Bishop [3]. The PAC framework is due to Valiant [40], and the VC dimension and its generalization bound to Vapnik and Chervonenkis [41]. The finite-precision argument of Section 4.7 is a folklore counting bound that makes the infinite-class problem concrete before the VC machinery; it appears in this form in the lecture notes of Andrew Ng's CS229. The bias-variance decomposition recurs for regression in Chapter 5, and the cross-validation protocol of Section 4.10 is the practical companion to every later chapter that has a complexity knob to set.

Exercises

Exercise 4.1. A nonnegative random variable has mean $\mu = 5$ and variance $\sigma^2 = 4$. (a) Bound $\mathbb{P}(X \geq 20)$ with Markov. (b) Bound $\mathbb{P}(|X - 5| \geq 6)$ with Chebyshev. (c) Which used more information?

Solution. (a) Markov: $\mathbb{P}(X \geq 20) \leq 5/20 = 0.25$. (b) Chebyshev: $\mathbb{P}(|X - 5| \geq 6) \leq 4/36 = 1/9 \approx 0.111$. (c) Chebyshev uses the variance in addition to the mean, so it can give a sharper, two-sided statement; Markov needs only nonnegativity and the mean, so it is more general but looser. □

Exercise 4.2. For $n = 1000$ fair coin flips, compare the Chebyshev and Chernoff bounds on $\mathbb{P}(X \geq 600)$.

Solution. Here $\mu = 500$, $\sigma^2 = np(1-p) = 250$, and $600 = (1+\delta)\mu$ with $\delta = 0.2$. Chebyshev: $\mathbb{P}(|X - 500| \geq 100) \leq 250/100^2 = 0.025$. Chernoff: $\mathbb{P}(X \geq 600) \leq \exp(-500[1.2 \ln 1.2 - 0.2]) \approx 8.3 \times 10^{-5}$, about 300 times smaller and far closer to the truth ($\approx 10^{-10}$). □

Exercise 4.3. A finite hypothesis class has $k = 1000$ members. How many iid samples guarantee that, with probability at least 0.95, every hypothesis has training error within $\gamma = 0.05$ of its true error?

Solution. By Eq. (4.8) with $\delta = 0.05$, $m \geq \ln(2k/\delta)/(2\gamma^2) = \ln(40000)/(2 \cdot 0.0025) = 10.60/0.005 \approx 2120$. The dependence on k and δ is only logarithmic, but halving the allowed gap γ would quadruple the required m. □

Exercise 4.4. Find the VC dimension of threshold classifiers $h_t(x) = \mathbf{1}[x \geq t]$ on the line.

Solution. One point is shattered: $t > x_1$ labels it 0, $t \leq x_1$ labels it 1. Two points $x_1 < x_2$ are not: a threshold labels everything right of t as 1, so the labeling ($x_1 \mapsto 1,\ x_2 \mapsto 0$) would need $x_1 \geq t > x_2$, impossible since $x_1 < x_2$. Hence $d_{\mathrm{VC}} = 1$, matching the single parameter t. □

Exercise 4.5. Using Eq. (4.10) and the bound Eq. (4.7), explain why fitting a degree-20 polynomial to 15 points generalizes poorly, and what changes with far more data.

Solution. A degree-20 class is very expressive, so its bias is small. But with only $m = 15$ points it has more than enough freedom to interpolate the sample, including its noise, so the variance term is large and dominates the test error (overfitting). In the bound, the effective capacity is large while m is small, so the gap $\sqrt{(\text{capacity})/m}$ is large and training error badly understates test error. Collecting far more data shrinks that gap, the variance falls, and the low bias finally pays off, the quantitative form of "more data lets you safely use a more complex model." □

Exercise 4.6. A linear model in $\mathbb{R}^{10}$ stores each of its 11 parameters as a 32-bit float. Bound the number of distinct hypotheses and the sample complexity to reach $\gamma = 0.1$, $\delta = 0.05$. Comment on the dependence on the parameter count.

Solution. With $d = 11$ parameters at 32 bits, $k \leq (2^{32})^{11} = 2^{352}$. By Eq. (4.8), $m \geq \ln(2k/\delta)/(2\gamma^2) = (\ln(2/\delta) + 352 \ln 2)/(2 \cdot 0.01) = (3.69 + 244.0)/0.02 \approx 12{,}400$. The dominant term is $352 \ln 2$, linear in the number of parameters, so $m = \mathcal{O}(d)$: sample complexity scales with the parameter count rather than exponentially in it. The bound is loose because many bit patterns give the same classifier; the VC dimension $d + 1 = 11$ is the tighter capacity. □

PART II

Regression

CHAPTER 5

Linear Models for Regression

There are three roads to the same weights: walk downhill, drop a perpendicular, or assume Gaussian noise. They meet at the normal equations.

after Professor Peter Chin, ENGS 106

Chapter 1 fit a curve to noisy data and used it to motivate the whole subject; this chapter does the fitting properly. The model is the linear-in-parameters model, the polynomial of Chapter 1 generalized to an arbitrary set of features, and the task is to choose its weights. We find them three different ways, and the punchline is that all three give the same answer. Gradient descent walks downhill on the error surface. The normal equations drop a perpendicular from the data onto the space the model can reach. Maximum likelihood, assuming Gaussian noise, turns the sum of squared errors into a negative log-likelihood. The three roads meet at one small linear system, and seeing why they agree is the best way to understand what least squares actually does. We then add the two refinements every practitioner needs: regularization, to keep the weights from exploding (the overfitting symptom of Chapter 1), and the bias-variance view of what regularization buys.

Roadmap

Suggested reading: Bishop [3], Sections 3.1–3.2; Deisenroth et al. [9], Chapter 9.

We cover: linear basis-function models and the design matrix (Section 5.1); the sum-of-squares error (Section 5.2); the three roads, gradient descent (Section 5.3), the normal equations and projection geometry (Section 5.4), and maximum likelihood under Gaussian noise (Section 5.5); regularized least squares, ridge and lasso (Section 5.6); the bias-variance decomposition (Section 5.7); and locally weighted regression (Section 5.8).

5.1 Linear basis-function models

The polynomial $y(x, \boldsymbol{w}) = \sum_j w_j x^j$ of Chapter 1 is one instance of a general and surprisingly powerful idea: take a fixed set of functions of the input, the *basis functions* or *features*, and predict with a weighted sum of them.

Definition 5.1 (Linear basis-function model). Given basis functions $\phi_0, \phi_1, \ldots, \phi_{M-1}$, a *linear model* is

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} w_j \, \phi_j(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}), \tag{5.1}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_0(\boldsymbol{x}), \ldots, \phi_{M-1}(\boldsymbol{x}))^\top$ and conventionally $\phi_0(\boldsymbol{x}) = 1$, so that w_0 is a bias (intercept) term.

The word "linear" refers to the parameters rather than the input. The model is a linear function of $\boldsymbol{w}$, which is what makes the fit have a closed form, but it can be a wildly nonlinear function of $\boldsymbol{x}$ through the choice of ϕ_j. Figure 5.1 shows the three families used throughout the book: polynomial features $\phi_j(x) = x^j$, Gaussian (radial basis) features $\phi_j(x) = \exp(-(x - \mu_j)^2 / 2s^2)$ that respond to a neighborhood of a center μ_j, and sigmoidal features $\phi_j(x) = \sigma((x - \mu_j)/s)$ that switch on past a threshold. Fitting the weights of a Gaussian-basis model is still a linear least-squares problem, even though the resulting curve is a smooth nonlinear function of x; the nonlinearity is baked into the fixed features, and only the weights are learned. (Learning the features themselves is what neural networks do, in Chapter 10.)

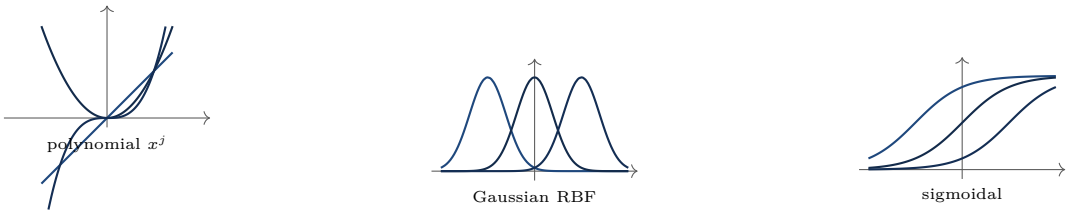


Figure 5.1. Polynomial features grow without bound, Gaussian radial features respond to a local neighborhood of a center, and sigmoidal features switch on past a threshold. A linear model is a weighted sum of such features; the weights are learned, the features are fixed.

Stack the predictions over a data set $\{(\boldsymbol{x}_i, t_i)\}_{i=1}^m$ into a matrix. The *design matrix* $\boldsymbol{\Phi} \in \mathbb{R}^{m \times M}$ has the features of example i in its ith row, $\Phi_{ij} = \phi_j(\boldsymbol{x}_i)$, so the vector of all m predictions is $\boldsymbol{\Phi}\boldsymbol{w}$, and the vector of targets is $\boldsymbol{t} = (t_1, \dots, t_m)^\top$.

5.2 The sum-of-squares error

The fit minimizes the total squared discrepancy between predictions and targets,

$$J(\boldsymbol{w}) = \tfrac{1}{2} \sum_{i=1}^{m} \big(y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i\big)^2 = \tfrac{1}{2} \|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}\|^2 = \tfrac{1}{2} (\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})^\top (\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}). \tag{5.2}$$

The factor $\tfrac{1}{2}$ is a convenience that cancels the 2 from differentiating the square. As a function of $\boldsymbol{w}$, J is a quadratic with a positive semidefinite Hessian $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ (Section 5.4), so it is convex and every stationary point is a global minimum. There is no risk of a bad local optimum; the only question is how to reach the global one, and that is where the three roads diverge.

5.3 Road one: gradient descent

The first road is iterative: start anywhere and repeatedly step in the direction that decreases J fastest. To know which direction that is, recall what the gradient means. The directional derivative of J along a unit vector $\boldsymbol{u}$ is $D_{\boldsymbol{u}}J = \nabla J^{\top}\boldsymbol{u} = \|\nabla J\|\cos\theta$, where θ is the angle between $\boldsymbol{u}$ and the gradient. This is most negative when $\cos\theta = -1$, that is when $\boldsymbol{u}$ points opposite to the gradient. The direction of steepest descent is $-\nabla J$, which is why gradient descent steps that way.

Gradient descent

Repeat $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha\,\nabla J(\boldsymbol{w})$, where $\alpha > 0$ is the *learning rate*. For the squared error Eq. (5.2),

$$\nabla J(\boldsymbol{w}) = \boldsymbol{\Phi}^{\top}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}) = \sum_{i=1}^{m}\big(y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i\big)\boldsymbol{\phi}(\boldsymbol{x}_i). \tag{5.3}$$

The gradient has the "error times input" structure that recurs in every model in this book: each example contributes its prediction error $y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i$ times its feature vector $\boldsymbol{\phi}(\boldsymbol{x}_i)$. Examples the model already predicts well contribute little; badly predicted examples pull hardest. We derive Eq. (5.3) in full in Appendix C; differentiating $\frac{1}{2}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})^{\top}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})$ gives $\boldsymbol{\Phi}^{\top}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})$ by the matrix-calculus product rule.

When m is large, summing over all examples for each step is expensive, and one instead uses *stochastic* gradient descent, which estimates the gradient from a single example or a small *mini-batch* and takes many cheap, noisy steps. This is the workhorse that trains the neural networks of Chapter 10, and we return to its variants (momentum, Adam) in Chapter 11.

Example 5.2 (One gradient-descent step). Fit a line $y = w_0 + w_1 x$ to the three points $(1,1), (2,2), (3,2)$, so $\boldsymbol{\Phi} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{smallmatrix}\right]$ and $\boldsymbol{t} = (1,2,2)^{\top}$. Start at $\boldsymbol{w} = (0,0)$, where every prediction is zero and the residual is $\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t} = (-1,-2,-2)^{\top}$. The gradient Eq. (5.3) is

$$\nabla J = \boldsymbol{\Phi}^{\top}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix}\begin{bmatrix} -1 \\ -2 \\ -2 \end{bmatrix} = \begin{bmatrix} -5 \\ -11 \end{bmatrix}.$$

With learning rate $\alpha = 0.05$, the step is $\boldsymbol{w} \leftarrow (0,0) - 0.05(-5,-11) = (0.25, 0.55)$, moving the weights toward the optimum $(0.667, 0.5)$ we compute exactly in Example 5.3. Repeating the step drives $\boldsymbol{w}$ to that optimum. ◊

5.4 Road two: the normal equations and projection

The second road skips the iteration and jumps straight to the bottom of the quadratic by setting the gradient Eq. (5.3) to zero:

$$\boldsymbol{\Phi}^{\top}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}) = \boldsymbol{0} \implies \boldsymbol{\Phi}^{\top}\boldsymbol{\Phi}\,\boldsymbol{w} = \boldsymbol{\Phi}^{\top}\boldsymbol{t}.$$

The normal equations

The least-squares weights solve $\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi}\,\boldsymbol{w} = \boldsymbol{\Phi}^{\top}\boldsymbol{t}$, and when $\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi}$ is invertible,

$$\boldsymbol{w}^{*} = (\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\top}\,\boldsymbol{t} = \boldsymbol{\Phi}^{\dagger}\,\boldsymbol{t}, \tag{5.4}$$

where $\boldsymbol{\Phi}^{\dagger} = (\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\top}$ is the Moore–Penrose pseudoinverse.

The name "normal" is geometric. The model can only produce predictions of the form $\boldsymbol{\Phi}\boldsymbol{w}$, which sweep out the column space of $\boldsymbol{\Phi}$, an M-dimensional subspace of $\mathbb{R}^m$. The target $\boldsymbol{t}$ generally lies outside that subspace (no exact fit exists), so the best we can do is the point in the subspace closest to $\boldsymbol{t}$, which is the orthogonal projection of $\boldsymbol{t}$ onto $\mathrm{Col}(\boldsymbol{\Phi})$. "Closest" means the residual $\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}^{*}$ is perpendicular to the subspace, hence orthogonal to every column of $\boldsymbol{\Phi}$, which is exactly the statement $\boldsymbol{\Phi}^{\top}(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}^{*}) = \boldsymbol{0}$, the normal equations again. Figure 5.2 is the picture: least squares drops a perpendicular from the data onto the space the model can reach. The projection itself is $\boldsymbol{\Phi}\boldsymbol{w}^{*} = \boldsymbol{\Phi}(\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\top}\boldsymbol{t} = P\boldsymbol{t}$, where $P = \boldsymbol{\Phi}(\boldsymbol{\Phi}^{\top}\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^{\top}$ is the projection matrix onto $\mathrm{Col}(\boldsymbol{\Phi})$.

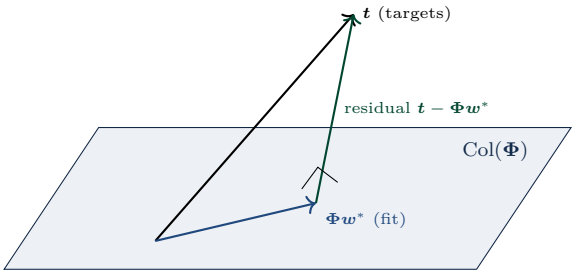


Figure 5.2. The model can only reach the column space of $\boldsymbol{\Phi}$. The best fit $\boldsymbol{\Phi}\boldsymbol{w}^*$ is the orthogonal projection of the target $\boldsymbol{t}$ onto that subspace, and the residual stands perpendicular to it. "Perpendicular" is the normal equations $\boldsymbol{\Phi}^\top(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}^*) = \mathbf{0}$.

Example 5.3 (A least-squares line by hand). Fit $y = w_0 + w_1 x$ to $(1,1),(2,2),(3,2)$. With $\boldsymbol{\Phi} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{smallmatrix}\right]$ and $\boldsymbol{t} = (1,2,2)^\top$, assemble the two small matrices the normal equations need:

$$\boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}, \qquad \boldsymbol{\Phi}^\top\boldsymbol{t} = \begin{bmatrix} 1+2+2 \\ 1{\cdot}1 + 2{\cdot}2 + 3{\cdot}2 \end{bmatrix} = \begin{bmatrix} 5 \\ 11 \end{bmatrix}.$$

The determinant of $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is $3{\cdot}14 - 6{\cdot}6 = 6$, so its inverse is $\frac{1}{6}\left[\begin{smallmatrix} 14 & -6 \\ -6 & 3 \end{smallmatrix}\right]$, and

$$\boldsymbol{w}^* = \frac{1}{6}\begin{bmatrix} 14 & -6 \\ -6 & 3 \end{bmatrix}\begin{bmatrix} 5 \\ 11 \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 70 - 66 \\ -30 + 33 \end{bmatrix} = \frac{1}{6}\begin{bmatrix} 4 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.667 \\ 0.5 \end{bmatrix}.$$

The fitted line is $y = 0.667 + 0.5\,x$. Its predictions at $x = 1,2,3$ are $1.167, 1.667, 2.167$, so the residuals are $-0.167, +0.333, -0.167$. As a check on the geometry, the residuals sum to zero (orthogonality to the all-ones column ϕ_0) and their inner product with $(1,2,3)$ is $-0.167 + 0.667 - 0.5 = 0$ (orthogonality to the x column): the residual is perpendicular to the column space, exactly as Fig. 5.2 demands. ◊

5.5 Road three: maximum likelihood under Gaussian noise

The third road explains *why* squared error, rather than some other measure of fit, is the natural thing to minimize. Posit a probability model for how the targets were generated: the true value is the model output corrupted by additive Gaussian noise,

$$t = y(\boldsymbol{x}, \boldsymbol{w}) + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \beta^{-1}), \tag{5.5}$$

so that $t \mid \boldsymbol{x} \sim \mathcal{N}(y(\boldsymbol{x},\boldsymbol{w}), \beta^{-1})$ with precision $\beta = 1/\sigma^2$. Treating the m targets as independent, the log-likelihood is

$$\log p(\boldsymbol{t} \mid \boldsymbol{w}, \beta) = \sum_{i=1}^{m} \log \mathcal{N}(t_i \mid y(\boldsymbol{x}_i,\boldsymbol{w}), \beta^{-1}) = \frac{m}{2}\log\beta - \frac{m}{2}\log(2\pi) - \beta \cdot \underbrace{\tfrac{1}{2}\sum_{i=1}^{m}(y(\boldsymbol{x}_i,\boldsymbol{w}) - t_i)^2}_{J(\boldsymbol{w})}.$$

Only the last term depends on $\boldsymbol{w}$, and it is $-\beta J(\boldsymbol{w})$. Maximizing the log-likelihood over $\boldsymbol{w}$ therefore *minimizes* $J(\boldsymbol{w})$, the sum-of-squares error.

Least squares is Gaussian maximum likelihood

Under the Gaussian noise model Eq. (5.5), the maximum likelihood weights are exactly the least-squares weights Eq. (5.4). The choice of squared error is the choice of Gaussian noise.

This closes a loop opened in Chapter 1. There, decision theory showed the optimal predictor under squared loss is the conditional mean $\mathbb{E}[t \mid \boldsymbol{x}]$; here, the Gaussian model makes that conditional mean $y(\boldsymbol{x},\boldsymbol{w}) = \boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x})$, and maximum likelihood finds the weights. The precision is fit too: setting $\partial/\partial\beta = 0$ gives $1/\beta_{\mathrm{ML}} = \frac{1}{m}\sum_i (y(\boldsymbol{x}_i, \boldsymbol{w}_{\mathrm{ML}}) - t_i)^2$, the mean squared residual, the noise variance read off the leftover error. And the connection runs the other way as well: if the noise were Laplacian rather than Gaussian, maximum likelihood would minimize the sum of *absolute* errors instead, the loss function chosen by the noise model.

Figure 5.3. Minimizing the error (green ellipses) subject to a norm budget (blue ball) touches where the contours first meet the ball. The round ℓ_2 ball is touched off-axis, so all weights are nonzero; the diamond ℓ_1 ball is usually touched at a corner, where a weight is exactly zero. Sparsity is a corner effect.

5.6 Regularized least squares: ridge and lasso

The overfitting of Chapter 1 announced itself as weights exploding to enormous opposite-signed values. The direct cure is to penalize large weights, adding a term to the error that grows with $\|\boldsymbol{w}\|$.

Ridge regression (also called weight decay, or Tikhonov regularization) adds the squared ℓ_2 norm:

$$J_{\text{ridge}}(\boldsymbol{w}) = \tfrac{1}{2}\|\boldsymbol{\Phi w} - \boldsymbol{t}\|^2 + \tfrac{\lambda}{2}\|\boldsymbol{w}\|^2. \tag{5.6}$$

Its gradient is $\boldsymbol{\Phi}^\top(\boldsymbol{\Phi w} - \boldsymbol{t}) + \lambda \boldsymbol{w}$, and setting it to zero gives the regularized normal equations, whose only change from Eq. (5.4) is a λ added along the diagonal:

$$\boldsymbol{w}_{\text{ridge}} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda I)^{-1} \boldsymbol{\Phi}^\top \boldsymbol{t}. \tag{5.7}$$

That added diagonal does double duty. It shrinks the weights toward zero, the more so as λ grows, and it makes $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda I$ invertible even when $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is singular (more features than data), curing both overfitting and ill-conditioning at once.

Remark 5.4 *(Ridge is MAP with a Gaussian prior).* The penalty is not arbitrary. Place a Gaussian prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \alpha^{-1} I)$ on the weights and take the MAP estimate of Chapter 1. Maximizing $\log p(\boldsymbol{t} \mid \boldsymbol{w}) + \log p(\boldsymbol{w})$, the log-prior contributes $-\frac{\alpha}{2}\|\boldsymbol{w}\|^2$, so MAP minimizes $\beta J(\boldsymbol{w}) + \frac{\alpha}{2}\|\boldsymbol{w}\|^2$, which is Eq. (5.6) with $\lambda = \alpha/\beta$. Ridge regression *is* MAP estimation under a Gaussian weight prior, the concrete instance of the regularization-is-a-prior principle of Chapter 3, and the strength λ is the ratio of noise precision to prior precision.

Example 5.5 (Ridge shrinks the fit). Return to the line of Example 5.3 and add a ridge penalty $\lambda = 6$. The system becomes $(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + 6I)\boldsymbol{w} = \boldsymbol{\Phi}^\top \boldsymbol{t}$, that is $\left[\begin{smallmatrix} 9 & 6 \\ 6 & 20 \end{smallmatrix}\right]\boldsymbol{w} = \left[\begin{smallmatrix} 5 \\ 11 \end{smallmatrix}\right]$. The determinant is $9{\cdot}20 - 36 = 144$, so

$$\boldsymbol{w}_{\text{ridge}} = \frac{1}{144}\begin{bmatrix} 20 & -6 \\ -6 & 9 \end{bmatrix}\begin{bmatrix} 5 \\ 11 \end{bmatrix} = \frac{1}{144}\begin{bmatrix} 100 - 66 \\ -30 + 99 \end{bmatrix} = \frac{1}{144}\begin{bmatrix} 34 \\ 69 \end{bmatrix} = \begin{bmatrix} 0.236 \\ 0.479 \end{bmatrix}.$$

Both weights are pulled toward zero from the unregularized $(0.667, 0.5)$, the intercept most. As $\lambda \to \infty$ the fit collapses to the flat line $\boldsymbol{w} = \boldsymbol{0}$; as $\lambda \to 0$ it returns to ordinary least squares. The knob λ slides the fit between the data and the prior, and it is chosen by cross-validation (Chapter 4). ◊

A different penalty produces a qualitatively different fit. The *lasso* uses the ℓ_1 norm, $J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{\Phi w} - \boldsymbol{t}\|^2 + \lambda\|\boldsymbol{w}\|_1$, and the change from squaring to absolute value makes the lasso drive some weights exactly to zero, performing feature selection as it fits. The reason is geometric (Fig. 5.3): minimizing the error subject to a norm budget means inflating the elliptical error contours until they first touch the constraint ball. The ℓ_2 ball is round, so the touch point is generically off-axis with all weights nonzero; the ℓ_1 ball is a diamond with corners on the axes, and the contours typically touch a corner, where some coordinates are exactly zero. The general ℓ_q penalty $\frac{\lambda}{2}\sum_j |w_j|^q$ interpolates: $q = 2$ is ridge, $q = 1$ is lasso, and $q < 1$ is sparser still but no longer convex.

5.7 The bias-variance decomposition

Regularization trades fit for stability, and the bias-variance decomposition of Chapter 4 says exactly what is traded. For squared loss, the expected error of a fitted model at a point $\boldsymbol{x}$, averaged over the random training set, splits into three pieces:

$$\mathbb{E}\big[(t - y(\boldsymbol{x}))^2\big] = \underbrace{\big(\mathbb{E}[y(\boldsymbol{x})] - \mathbb{E}[t \mid \boldsymbol{x}]\big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}\big[(y(\boldsymbol{x}) - \mathbb{E}[y(\boldsymbol{x})])^2\big]}_{\text{variance}} + \underbrace{\sigma^2}_{\text{noise}}. \tag{5.8}$$

The *bias* measures how far the average fit is from the truth, the *variance* how much the fit jumps around as the training set changes, and the *noise* is the irreducible floor from Chapter 1. Regularization moves a model along this tradeoff. A large λ forces the weights small, producing a rigid, nearly flat fit that barely changes between training sets (low variance) but cannot follow the true function (high bias). A small λ lets the fit chase every data set (high variance, low bias). The cross-validated choice of λ sits at the bottom of the U-shaped sum, the same valley first drawn in Chapter 1, now controlled by a continuous knob rather than the integer degree.

5.8 Locally weighted regression

A fixed global model commits to one functional form everywhere. *Locally weighted* regression refuses to: to predict at a query point x_*, it fits a fresh simple model using only the nearby data, weighting each training point by its closeness to x_*. The weighted error is

$$J_{x_*}(\boldsymbol{w}) = \tfrac{1}{2}\sum_{i=1}^{m} r_i \,(y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i)^2, \qquad r_i = \exp\!\Big(-\frac{(x_i - x_*)^2}{2\tau^2}\Big), \tag{5.9}$$

with a Gaussian weight r_i that is near one for points close to x_* and near zero for distant ones. The *bandwidth* τ sets the size of the neighborhood, and it is the same kind of knob as the kernel bandwidth of Chapter 3: small τ gives a wiggly fit that follows local noise, large τ recovers ordinary global regression. The solution is weighted least squares, $\boldsymbol{w}_* = (\boldsymbol{\Phi}^\top R \boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top R \boldsymbol{t}$ with $R = \operatorname{diag}(r_1, \ldots, r_m)$, and because the weights depend on x_*, the model is refit for every prediction. It is *nonparametric*: there is no fixed set of weights to store, only the data and the bandwidth, and the model grows in flexibility with the data. The same locality idea, applied to classification, is the k-nearest-neighbor classifier of Chapter 8.

For multiple outputs, finally, nothing changes but the shape: stacking K target columns into a matrix $T \in \mathbb{R}^{m \times K}$, the least-squares solution is $W = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top T$, one independent regression per output column, sharing the same design matrix and hence the same pseudoinverse.

5.9 A least-squares fit from start to finish

The three roads and their geometry are easier to trust once you have driven the whole route on numbers. This section fits a line to four points by the normal equations, checks the projection geometry on the residual, then reuses the same data twice more: once with a ridge penalty, to watch the weights shrink, and once with a quadratic feature map, to fit a curve. Every arithmetic step is shown so the closed form of Section 5.4 stops being a formula and becomes a calculation you can redo.

Example 5.6 (A line by the normal equations, every step). Fit $y = w_0 + w_1 x$ to the four points $(0, 1)$, $(1, 2)$, $(2, 2)$, $(3, 5)$. The basis is $\boldsymbol{\phi}(x) = (1, x)^\top$, so the design matrix stacks a constant column and an x column,

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}, \qquad \boldsymbol{t} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 5 \end{bmatrix}.$$

The normal equations $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}\, \boldsymbol{w} = \boldsymbol{\Phi}^\top \boldsymbol{t}$ need two small products. The left side accumulates sums of 1, x, and x^2 over the four rows,

$$\boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \begin{bmatrix} \sum 1 & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} 4 & 6 \\ 6 & 14 \end{bmatrix},$$

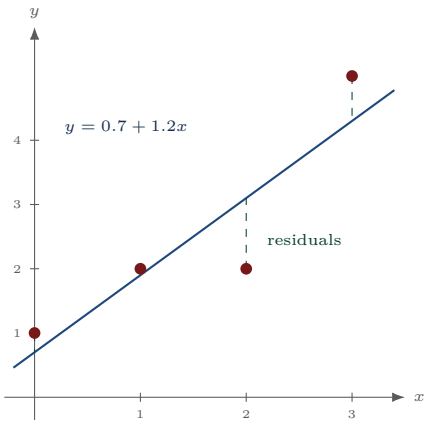


Figure 5.4. The four data points (red) and the least-squares line $y = 0.7 + 1.2x$ from Example 5.6. Dashed segments are the residuals, the vertical gaps the fit leaves. Their signed lengths $0.3, 0.1, -1.1, 0.7$ sum to zero and are orthogonal to the x values, the projection condition of Fig. 5.2 confirmed on numbers.

using $\sum x_i = 0+1+2+3 = 6$ and $\sum x_i^2 = 0+1+4+9 = 14$. The right side accumulates $\sum t_i$ and $\sum x_i t_i$,

$$\boldsymbol{\Phi}^\top \boldsymbol{t} = \begin{bmatrix} 1+2+2+5 \\ 0{\cdot}1 + 1{\cdot}2 + 2{\cdot}2 + 3{\cdot}5 \end{bmatrix} = \begin{bmatrix} 10 \\ 21 \end{bmatrix}.$$

The determinant of $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is $4{\cdot}14 - 6{\cdot}6 = 56 - 36 = 20$, so the inverse is $\frac{1}{20}\left[\begin{smallmatrix} 14 & -6 \\ -6 & 4 \end{smallmatrix}\right]$ and the weights are

$$\boldsymbol{w}^* = \frac{1}{20}\begin{bmatrix} 14 & -6 \\ -6 & 4 \end{bmatrix}\begin{bmatrix} 10 \\ 21 \end{bmatrix} = \frac{1}{20}\begin{bmatrix} 140 - 126 \\ -60 + 84 \end{bmatrix} = \frac{1}{20}\begin{bmatrix} 14 \\ 24 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 1.2 \end{bmatrix}.$$

The fitted line is $y = 0.7 + 1.2\,x$ (Fig. 5.4). Its predictions at $x = 0, 1, 2, 3$ are $0.7, 1.9, 3.1, 4.3$, so the residuals $\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}^*$ are 0.3, 0.1, -1.1, 0.7. As the projection picture of Fig. 5.2 demands, the residual is orthogonal to both columns of $\boldsymbol{\Phi}$: it sums to $0.3 + 0.1 - 1.1 + 0.7 = 0$ (orthogonal to the constant column), and its inner product with the x column is

$$0{\cdot}0.3 + 1{\cdot}0.1 + 2{\cdot}(-1.1) + 3{\cdot}0.7 = 0.1 - 2.2 + 2.1 = 0.$$

The fit is the orthogonal projection of $\boldsymbol{t}$ onto $\mathrm{Col}(\boldsymbol{\Phi})$, reached here by hand. ◊

Section 5.6 argued that ridge regression only adds λ to the diagonal of the normal equations and shrinks the weights toward zero. On the same four points the effect is concrete.

Example 5.7 (Ridge on the same data, weights shrinking). Take the data and design matrix of Example 5.6 and add a ridge penalty $\lambda = 4$. The only change to the normal equations is λ along the diagonal,

$$\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda I = \begin{bmatrix} 4 & 6 \\ 6 & 14 \end{bmatrix} + \begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 8 & 6 \\ 6 & 18 \end{bmatrix},$$

while the right side $\boldsymbol{\Phi}^\top\boldsymbol{t} = (10, 21)^\top$ is untouched, since the penalty acts on $\boldsymbol{w}$ alone and leaves the data terms alone. The determinant is now $8{\cdot}18 - 6{\cdot}6 = 144 - 36 = 108$, so

$$\boldsymbol{w}_{\text{ridge}} = \frac{1}{108}\begin{bmatrix} 18 & -6 \\ -6 & 8 \end{bmatrix}\begin{bmatrix} 10 \\ 21 \end{bmatrix} = \frac{1}{108}\begin{bmatrix} 180 - 126 \\ -60 + 168 \end{bmatrix} = \frac{1}{108}\begin{bmatrix} 54 \\ 108 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.0 \end{bmatrix}.$$

Both coefficients move toward zero relative to the ordinary least-squares $(0.7, 1.2)$: the intercept drops from 0.7 to 0.5 and the slope from 1.2 to 1.0. The penalty also raised the diagonal entries from $4, 14$ to $8, 18$ and the determinant from 20 to 108, which is the other thing λ buys, a better-conditioned system that stays invertible no matter how the data is arranged. Sending $\lambda \to \infty$ would crush both weights to zero (the flat line $\boldsymbol{w} = \boldsymbol{0}$); sending $\lambda \to 0$ would restore $(0.7, 1.2)$. ◊

5.10 Fitting a curve: a quadratic by least squares

Nothing in the normal equations cares whether the columns of $\boldsymbol{\Phi}$ hold the raw input or nonlinear features of it. Definition 5.1 made this the whole point of a linear basis-function model: choose richer features ϕ_j, fill the design matrix with their values, and the same closed form fits a curve. We work one quadratic fit in full to see the design matrix grow a column and the curve bend.

Example 5.8 (A quadratic fit by least squares). Fit $y = w_0 + w_1 x + w_2 x^2$ to five points on a symmetric grid, $x = -2, -1, 0, 1, 2$ with targets $t = 1, 2, 2, 3, 7$. The feature map is now $\boldsymbol{\phi}(x) = (1, x, x^2)^\top$, so the design matrix carries three columns,

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}, \qquad \boldsymbol{t} = (1, 2, 2, 3, 7)^\top.$$

The symmetric grid makes the bookkeeping clean, because every odd power of x sums to zero: $\sum x_i = 0$ and $\sum x_i^3 = 0$. Only the even moments survive, $\sum x_i^2 = 4 + 1 + 0 + 1 + 4 = 10$ and $\sum x_i^4 = 16 + 1 + 0 + 1 + 16 = 34$, so

$$\boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \begin{bmatrix} 5 & 0 & 10 \\ 0 & 10 & 0 \\ 10 & 0 & 34 \end{bmatrix}, \qquad \boldsymbol{\Phi}^\top \boldsymbol{t} = \begin{bmatrix} \sum t_i \\ \sum x_i t_i \\ \sum x_i^2 t_i \end{bmatrix} = \begin{bmatrix} 15 \\ 13 \\ 37 \end{bmatrix},$$

where $\sum x_i t_i = -2 + (-2) + 0 + 3 + 14 = 13$ and $\sum x_i^2 t_i = 4 + 2 + 0 + 3 + 28 = 37$. The zeros in $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$ split the 3×3 system into two independent pieces. The middle row reads $10\, w_1 = 13$, so the linear coefficient is

$$w_1 = \tfrac{13}{10} = 1.3,$$

and the first and third rows form a 2×2 system in w_0 and w_2,

$$\begin{bmatrix} 5 & 10 \\ 10 & 34 \end{bmatrix} \begin{bmatrix} w_0 \\ w_2 \end{bmatrix} = \begin{bmatrix} 15 \\ 37 \end{bmatrix}, \qquad \det = 5{\cdot}34 - 10{\cdot}10 = 70.$$

By Cramer's rule, $w_0 = (15{\cdot}34 - 10{\cdot}37)/70 = (510 - 370)/70 = 140/70 = 2$ and $w_2 = (5{\cdot}37 - 10{\cdot}15)/70 = (185 - 150)/70 = 35/70 = 0.5$. The fitted parabola is

$$y = 2 + 1.3\, x + 0.5\, x^2. \tag{5.10}$$

This is a genuine least-squares fit rather than an interpolation: with five points and three weights the curve cannot pass through every target. Its predictions at $x = -2, \ldots, 2$ are $1.4, 1.2, 2.0, 3.8, 6.6$, leaving residuals $-0.4, 0.8, 0.0, -0.8, 0.4$. They sum to zero (orthogonal to the constant column) and, being antisymmetric on a symmetric grid, are orthogonal to the x^2 column as well, $\sum x_i^2 e_i = 4(-0.4) + 1(0.8) + 0 + 1(-0.8) + 4(0.4) = 0$. The same projection that placed the line now places the parabola. ◊

Remark 5.9 *(The fit is linear in the weights, curved in x).* Example 5.8 fits a curved function of x with the identical machinery that fit a straight line, because Eq. (5.10) is linear in the weights w_0, w_1, w_2. The nonlinearity lives entirely in the fixed column x^2 of the design matrix. Swap that column for a Gaussian feature $\exp(-(x-\mu)^2/2s^2)$ and the fit becomes a bump of the kind drawn in Fig. 5.1, still by the same normal equations. Learning the features themselves, rather than fixing them in advance, is the step that leads to neural networks in Chapter 10.

The bias-variance split of Eq. (5.8) is also worth seeing on numbers, because it explains why the shrunk fit of Example 5.7 can beat the unregularized one even though it fits the training data less well.

Remark 5.10 *(Bias and variance on numbers).* Suppose the truth at a query point is $\mathbb{E}[t \mid \boldsymbol{x}] = 2$ and the noise variance is $\sigma^2 = 0.5$. Imagine drawing two equally likely training sets and refitting each time. A flexible model swings hard with the data, predicting 1 on one draw and 3 on the other. Its average prediction is $\mathbb{E}[y] = \frac{1}{2}(1 + 3) = 2$, so its squared bias is $(\mathbb{E}[y] - 2)^2 = 0$, but its variance is

$$\tfrac{1}{2}\left[(1-2)^2 + (3-2)^2\right] = \tfrac{1}{2}(1+1) = 1,$$

giving an expected error of $\text{bias}^2 + \text{variance} + \sigma^2 = 0 + 1 + 0.5 = 1.5$ by Eq. (5.8). A heavily regularized model barely moves, predicting 1.4 and 1.6 on the two draws. Its average is 1.5, so its squared bias is $(1.5-2)^2 = 0.25$, but its variance collapses to

$$\tfrac{1}{2}\left[(1.4-1.5)^2 + (1.6-1.5)^2\right] = \tfrac{1}{2}(0.01+0.01) = 0.01,$$

for an expected error of $0.25 + 0.01 + 0.5 = 0.76$. The rigid model accepts a small bias and is rewarded with a large drop in variance, so it predicts better on average despite never fitting any single training set as closely. That trade is exactly what the knob λ controls.

Notes and References

The linear basis-function model, the three views of least squares, and the bias-variance decomposition follow Sections 3.1 and 3.2 of Bishop [3]; the projection geometry of Section 5.4 is developed at length in linear-algebra terms by Strang [37], and the pseudoinverse is reviewed in Appendix A. The matrix-calculus derivation of the gradient Eq. (5.3) and the normal equations is carried out in Appendix C. Ridge regression is due to Hoerl and Kennard; the lasso to Tibshirani; the unifying ℓ_q and Bayesian-prior view is in Bishop [3] and Hastie et al. [14]. Gradient descent and its stochastic variants, previewed here, are the subject of Chapter 11, and the full Bayesian treatment of this same linear model, with a posterior over $\boldsymbol{w}$ and a predictive distribution that reports its own uncertainty, is the next chapter, Chapter 6.

Exercises

Exercise 5.1. Fit a line $y = w_0 + w_1 x$ to the points $(0,1), (1,3), (2,4)$ by the normal equations. Verify that the residual is orthogonal to both columns of the design matrix.

Solution. With $\boldsymbol{\Phi} = \left[\begin{smallmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{smallmatrix}\right]$ and $\boldsymbol{t} = (1,3,4)^\top$, $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \left[\begin{smallmatrix} 3 & 3 \\ 3 & 5 \end{smallmatrix}\right]$ and $\boldsymbol{\Phi}^\top\boldsymbol{t} = (8,11)^\top$. The determinant is $15 - 9 = 6$, so $\boldsymbol{w}^* = \frac{1}{6}\left[\begin{smallmatrix} 5 & -3 \\ -3 & 3 \end{smallmatrix}\right](8,11)^\top = \frac{1}{6}(40-33,\ -24+33)^\top = \frac{1}{6}(7,9)^\top = (1.167, 1.5)^\top$. Predictions $1.167, 2.667, 4.167$ give residuals $-0.167, 0.333, -0.167$, which sum to zero (orthogonal to ϕ_0) and dot $(0,1,2)$ to $0 + 0.333 - 0.333 = 0$ (orthogonal to the x column). □

Exercise 5.2. For the data of Exercise 5.1, take one gradient-descent step from $\boldsymbol{w} = (0,0)$ with learning rate $\alpha = 0.1$, and confirm it moves toward the least-squares solution.

Solution. At $\boldsymbol{w} = (0,0)$ the residual is $\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t} = (-1,-3,-4)^\top$, so $\nabla J = \boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}) = (-8,-11)^\top$. The step is $\boldsymbol{w} \leftarrow (0,0) - 0.1(-8,-11) = (0.8, 1.1)$, which moves from the origin toward the optimum $(1.167, 1.5)$ of Exercise 5.1. □

Exercise 5.3. Show that maximizing the likelihood under the Gaussian noise model Eq. (5.5) is equivalent to minimizing the sum-of-squares error, and find the maximum likelihood estimate of the noise precision β.

Solution. The log-likelihood is $\frac{m}{2}\log\beta - \frac{m}{2}\log(2\pi) - \beta J(\boldsymbol{w})$ with $J(\boldsymbol{w}) = \frac{1}{2}\sum_i (y(\boldsymbol{x}_i, \boldsymbol{w}) - t_i)^2$. Only $-\beta J(\boldsymbol{w})$ depends on $\boldsymbol{w}$, and since $\beta > 0$, maximizing over $\boldsymbol{w}$ minimizes $J(\boldsymbol{w})$: Gaussian MLE is least squares. Differentiating in β: $\frac{m}{2\beta} - J(\boldsymbol{w}) = 0$ gives $1/\beta_{\text{ML}} = 2J(\boldsymbol{w}_{\text{ML}})/m = \frac{1}{m}\sum_i (y(\boldsymbol{x}_i, \boldsymbol{w}_{\text{ML}}) - t_i)^2$, the mean squared residual. □

Exercise 5.4. Derive the ridge solution Eq. (5.7) from the objective Eq. (5.6), and explain why $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda I$ is always invertible for $\lambda > 0$ even when $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is singular.

Solution. The gradient of $\frac{1}{2}\|\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}\|^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$ is $\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}) + \lambda\boldsymbol{w} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda I)\boldsymbol{w} - \boldsymbol{\Phi}^\top\boldsymbol{t}$; setting it to zero gives $\boldsymbol{w} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda I)^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}$. The matrix $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is positive semidefinite, so its eigenvalues are ≥ 0; adding λI shifts every eigenvalue up by $\lambda > 0$, making them all strictly positive, so the matrix is positive definite and hence invertible regardless of whether $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ was singular. □

Exercise 5.5. Fit a line $y = w_0 + w_1 x$ to $(1,2),(2,2),(3,4)$, first by ordinary least squares and then with a ridge penalty $\lambda = 1$. Report both weight vectors and state which coordinate shrinks.

Solution. The design matrix is $\boldsymbol{\Phi} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{smallmatrix}\right]$ with $\boldsymbol{t} = (2,2,4)^\top$, so $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \left[\begin{smallmatrix} 3 & 6 \\ 6 & 14 \end{smallmatrix}\right]$ and $\boldsymbol{\Phi}^\top\boldsymbol{t} = (8,18)^\top$ (using $\sum t_i = 8$ and $\sum x_i t_i = 2{+}4{+}12 = 18$). For ordinary least squares the determinant is $3{\cdot}14{-}36 = 6$, so $\boldsymbol{w}^* = \frac{1}{6}\left[\begin{smallmatrix} 14 & -6 \\ -6 & 3 \end{smallmatrix}\right](8,18)^\top = \frac{1}{6}(112{-}108,\ -48{+}54)^\top = \frac{1}{6}(4,6)^\top = (0.667, 1.0)$. Adding $\lambda = 1$ gives $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}{+}I = \left[\begin{smallmatrix} 4 & 6 \\ 6 & 15 \end{smallmatrix}\right]$ with determinant $60{-}36 = 24$, so $\boldsymbol{w}_{\text{ridge}} = \frac{1}{24}\left[\begin{smallmatrix} 15 & -6 \\ -6 & 4 \end{smallmatrix}\right](8,18)^\top = \frac{1}{24}(120-108,\ -48+72)^\top = \frac{1}{24}(12,24)^\top = (0.5, 1.0)$. The intercept shrinks from 0.667 to 0.5; here the slope is unchanged, since the penalty acts through the full matrix rather than on each weight separately. □

Exercise 5.6. Fit $y = w_0 + w_2 x^2$ (no linear term) to the points $(-1,2),(0,1),(1,4)$ by the normal equations. Give the design matrix, the fitted curve, and verify the residual is orthogonal to the x^2 column.

Solution. The feature map is $\boldsymbol{\phi}(x) = (1, x^2)^\top$, so $\boldsymbol{\Phi} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 0 \\ 1 & 1 \end{smallmatrix}\right]$ and $\boldsymbol{t} = (2,1,4)^\top$. Then $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \left[\begin{smallmatrix} 3 & 2 \\ 2 & 2 \end{smallmatrix}\right]$ (the off-diagonal is $\sum x_i^2 = 2$, the corner is $\sum x_i^4 = 2$) and $\boldsymbol{\Phi}^\top\boldsymbol{t} = (\sum t_i,\ \sum x_i^2 t_i)^\top = (7,6)^\top$. The determinant is $3{\cdot}2 - 2{\cdot}2 = 2$, so $\boldsymbol{w}^* = \frac{1}{2}\left[\begin{smallmatrix} 2 & -2 \\ -2 & 3 \end{smallmatrix}\right](7,6)^\top = \frac{1}{2}(14-12,\ -14+18)^\top = \frac{1}{2}(2,4)^\top = (1,2)$. The fitted curve is $y = 1 + 2x^2$. Its predictions at $x = -1, 0, 1$ are $3, 1, 3$, so the residuals are $-1, 0, 1$, which sum to zero (orthogonal to the constant column) and satisfy $\sum x_i^2 e_i = 1{\cdot}(-1) + 0{\cdot}0 + 1{\cdot}1 = 0$ (orthogonal to the x^2 column). □

Exercise 5.7. Explain geometrically why the lasso (ℓ_1 penalty) produces weights that are exactly zero, while ridge (ℓ_2) does not.

Solution. Constrained least squares inflates the elliptical error contours until they first meet the norm-budget ball. The ℓ_2 ball is round and smooth, so the contact point is generically in the interior of a quadrant with all coordinates nonzero. The ℓ_1 ball is a diamond whose corners lie on the coordinate axes, where some coordinates are exactly zero; because the corners stick out, the inflating contours usually touch a corner first, setting those coordinates to zero. So the lasso selects features while it fits, whereas ridge only shrinks them. □

CHAPTER 6

Bayesian Linear Regression

A point estimate tells you the line. The posterior tells you how much to trust it.

after Professor Peter Chin, ENGS 106

Chapter 5 fit a single best weight vector $\boldsymbol{w}^*$ and drew a single best curve. But how much should we trust that curve, especially out where the data is thin? Maximum likelihood has no answer; it returns one line and no sense of its own reliability, and we saw in Chapter 1 how badly a point estimate can mislead on little data. This chapter gives the Bayesian treatment of the very same linear model, the third of the three estimation principles of Chapter 1 made fully concrete. Instead of one weight vector we carry a whole posterior distribution over weights, and instead of one prediction we produce a predictive distribution whose spread is honest about uncertainty: tight where data is dense, wide where it is scarce. That widening band is the image on the cover of this book, and by the end of the chapter we will have derived it.

Roadmap

Suggested reading: Bishop [3], Sections 3.3–3.5; Deisenroth et al. [9], Section 9.3.

We cover: the Gaussian posterior over the weights and its identity with ridge regression (Section 6.1); the predictive distribution and its two sources of uncertainty (Section 6.2); sequential updating (Section 6.3); the equivalent kernel that foreshadows Chapter 14 (Section 6.4); and the evidence, which tunes the hyperparameters and embodies Occam's razor (Section 6.5).

6.1 The posterior over weights

Keep the model of Chapter 5: targets are the linear model plus Gaussian noise of precision β, so the likelihood of the data $\boldsymbol{t} = (t_1, \dots, t_m)$ given the weights is

$$p(\boldsymbol{t} \mid \boldsymbol{w}) = \prod_{i=1}^{m} \mathcal{N}\big(t_i \mid \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_i), \beta^{-1}\big) = \mathcal{N}(\boldsymbol{t} \mid \boldsymbol{\Phi}\boldsymbol{w}, \beta^{-1} I).$$

Now add what Chapter 5 lacked: a prior on the weights. The conjugate choice, from the exponential-family logic of Chapter 3, is Gaussian. Take a zero-mean isotropic prior with precision α,

$$p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{0}, \alpha^{-1} I). \tag{6.1}$$

Because a Gaussian likelihood times a Gaussian prior is a Gaussian posterior (this is exactly Bayes' theorem for a linear-Gaussian model, Theorem 2.18), the posterior is Gaussian, and we only need its mean and covariance.

Theorem 6.1 (Gaussian posterior). Under the likelihood above and the prior Eq. (6.1), the posterior over the weights is $p(\boldsymbol{w} \mid \boldsymbol{t}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{m}_N, \boldsymbol{S}_N)$ with

$$\boldsymbol{S}_N^{-1} = \alpha I + \beta\, \boldsymbol{\Phi}^\top \boldsymbol{\Phi}, \qquad \boldsymbol{m}_N = \beta\, \boldsymbol{S}_N\, \boldsymbol{\Phi}^\top \boldsymbol{t}. \tag{6.2}$$

Proof. The posterior is proportional to likelihood times prior, so its logarithm is, up to a constant in $\boldsymbol{w}$,

$$\log p(\boldsymbol{w} \mid \boldsymbol{t}) = -\frac{\beta}{2}(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w})^\top (\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}) - \frac{\alpha}{2}\boldsymbol{w}^\top \boldsymbol{w} + \text{const}.$$

Expand and collect by powers of $\boldsymbol{w}$. The quadratic term is $-\frac{1}{2}\boldsymbol{w}^\top(\alpha I+\beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{w}$ and the linear term is $+\beta\boldsymbol{w}^\top\boldsymbol{\Phi}^\top\boldsymbol{t}$. A Gaussian $\mathcal{N}(\boldsymbol{w}\mid\boldsymbol{m}_N,\boldsymbol{S}_N)$ has log-density with quadratic term $-\frac{1}{2}\boldsymbol{w}^\top\boldsymbol{S}_N^{-1}\boldsymbol{w}$ and linear term $+\boldsymbol{w}^\top\boldsymbol{S}_N^{-1}\boldsymbol{m}_N$. Matching the quadratic terms gives $\boldsymbol{S}_N^{-1}=\alpha I+\beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$, and matching the linear terms gives $\boldsymbol{S}_N^{-1}\boldsymbol{m}_N=\beta\boldsymbol{\Phi}^\top\boldsymbol{t}$, that is $\boldsymbol{m}_N=\beta\boldsymbol{S}_N\boldsymbol{\Phi}^\top\boldsymbol{t}$. ■

The posterior mean $\boldsymbol{m}_N$ deserves a second look. Substituting $\boldsymbol{S}_N$,

$$\boldsymbol{m}_N=\beta(\alpha I+\beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}=(\tfrac{\alpha}{\beta}I+\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t},$$

which is exactly the ridge regression solution Eq. (5.7) with regularization strength $\lambda=\alpha/\beta$. This confirms Remark 5.4 from the other direction: the ridge point estimate is the *mode* (and here also the mean) of this Gaussian posterior. The Bayesian treatment does not discard ridge regression; it explains it as the center of a distribution and then hands us the rest of the distribution, the part that quantifies uncertainty. As data accumulates, the $\beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ term grows and dominates the prior's αI, so the posterior covariance $\boldsymbol{S}_N$ shrinks toward zero and the posterior concentrates on the maximum likelihood weights: the prior washes out, exactly as in the coin of Chapter 2.

Example 6.2 (A posterior over a line). Fit a line with features $\boldsymbol{\phi}(x)=(1,x)$, prior precision $\alpha=1$, and noise precision $\beta=1$, to the two points $(-1,-1)$ and $(1,1)$. The design matrix is $\boldsymbol{\Phi}=\left[\begin{smallmatrix}1&-1\\1&1\end{smallmatrix}\right]$, so $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}=\left[\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right]$ and

$$\boldsymbol{S}_N^{-1}=I+\begin{bmatrix}2&0\\0&2\end{bmatrix}=\begin{bmatrix}3&0\\0&3\end{bmatrix},\qquad \boldsymbol{S}_N=\tfrac{1}{3}I.$$

With $\boldsymbol{\Phi}^\top\boldsymbol{t}=(\,(-1)+1,\ (-1)(-1)+(1)(1)\,)=(0,2)$, the posterior mean is $\boldsymbol{m}_N=\beta\boldsymbol{S}_N\boldsymbol{\Phi}^\top\boldsymbol{t}=\frac{1}{3}(0,2)=(0,\ \frac{2}{3})$. The data alone would give a slope of 1 (the line through the two points), but the prior pulls the posterior-mean slope down to $\frac{2}{3}$, and the posterior covariance $\frac{1}{3}I$ records that both weights are still uncertain. We use this posterior in Example 6.3 to predict with error bars. ◊

6.2 The predictive distribution

A posterior over weights is not yet a prediction. To predict the target t at a new input $\boldsymbol{x}$, we use the predictive distribution of Chapter 1, averaging the model's output over the whole posterior rather than plugging in one weight vector:

$$p(t\mid\boldsymbol{x},\boldsymbol{t})=\int p(t\mid\boldsymbol{x},\boldsymbol{w})\,p(\boldsymbol{w}\mid\boldsymbol{t})\,d\boldsymbol{w}=\int\mathcal{N}\big(t\mid\boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}),\beta^{-1}\big)\,\mathcal{N}(\boldsymbol{w}\mid\boldsymbol{m}_N,\boldsymbol{S}_N)\,d\boldsymbol{w}. \tag{6.3}$$

This is a convolution of two Gaussians, which is again Gaussian, and its mean and variance are the sum of the contributions.

Predictive distribution

The predictive distribution is Gaussian,

$$p(t\mid\boldsymbol{x},\boldsymbol{t})=\mathcal{N}\big(t\,\big|\,\boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}),\ \sigma_N^2(\boldsymbol{x})\big),\qquad \sigma_N^2(\boldsymbol{x})=\underbrace{\frac{1}{\beta}}_{\text{noise}}+\underbrace{\boldsymbol{\phi}(\boldsymbol{x})^\top\boldsymbol{S}_N\,\boldsymbol{\phi}(\boldsymbol{x})}_{\text{model uncertainty}}. \tag{6.4}$$

The predictive mean is the posterior-mean model $\boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x})$, the same curve ridge regression would draw. The variance is the chapter's whole point. It is the sum of two terms. The first, $1/\beta$, is the irreducible noise in the data, the floor that no amount of data can lower, the same noise term that appeared in the bias-variance decomposition of Chapter 5. The second, $\boldsymbol{\phi}(\boldsymbol{x})^\top\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x})$, is the uncertainty contributed by our ignorance of the true weights. It depends on *where* we predict: it is small for inputs $\boldsymbol{x}$ whose feature vectors resemble those of the training data, and large for inputs in directions the data never explored. As more data arrives, $\boldsymbol{S}_N$ shrinks, this term shrinks with it, and the predictive distribution narrows toward the pure-noise floor $1/\beta$. Figure 6.1 is the picture: the band hugs the data and flares out beyond it.

Example 6.3 (Predicting with error bars). Continue Example 6.2, where $\boldsymbol{m}_N=(0,\frac{2}{3})$, $\boldsymbol{S}_N=\frac{1}{3}I$, and $\beta=1$. At a test input x, the feature vector is $\boldsymbol{\phi}(x)=(1,x)$, so the predictive mean is $\boldsymbol{m}_N^\top\boldsymbol{\phi}(x)=\frac{2}{3}x$, and the predictive variance is

$$\sigma_N^2(x)=\frac{1}{\beta}+\boldsymbol{\phi}(x)^\top\boldsymbol{S}_N\boldsymbol{\phi}(x)=1+\tfrac{1}{3}(1+x^2).$$

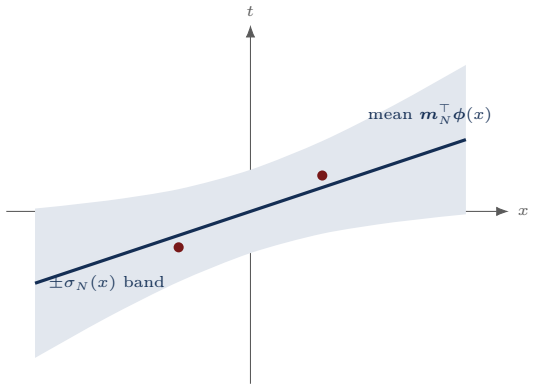


Figure 6.1. The predictive mean is the posterior-mean line, but the $\pm\sigma_N(x)$ band of Eq. (6.4) is narrowest near the two data points (red) and flares outward where the data is absent. The flaring is the model-uncertainty term $\boldsymbol{\phi}(x)^\top \boldsymbol{S}_N \boldsymbol{\phi}(x)$; the constant floor is the noise $1/\beta$. This is the image on the cover.

At the center of the data, $x = 0$, this is $1 + \frac{1}{3} = \frac{4}{3}$, with predictive standard deviation 1.15. Far from the data, at $x = 3$, it is $1 + \frac{1}{3}(10) = \frac{13}{3} \approx 4.33$, standard deviation 2.08. The error bar nearly doubles as we extrapolate away from the data, even though the predictive mean is just the straight line $\frac{2}{3}x$. A point estimate would have reported the same line with no warning; the Bayesian predictive variance flags that the prediction at $x = 3$ is far less trustworthy than the one at $x = 0$. ◊

6.3 Sequential updating

Because the prior and posterior are both Gaussian, learning can proceed one data point at a time, with the posterior after seeing n points serving as the prior for point $n+1$, exactly the sequential picture of the Beta-Bernoulli update in Chapter 2. Starting from the broad prior $\mathcal{N}(\mathbf{0}, \alpha^{-1} I)$, which as a function over weight space is a wide blob centered at the origin, each new observation multiplies in a likelihood factor and the posterior contracts toward the region of weight space consistent with the data seen so far. After a single point, the posterior is a ridge of probability (one point constrains a line only up to a one-parameter family); after two well-separated points it tightens to a small blob around the true weights; after many points it is a sharp spike at the maximum likelihood solution. The final posterior is the same whether the data is processed in a batch through Eq. (6.2) or absorbed one point at a time, because Gaussian conditioning, like addition, does not care about order. This makes Bayesian linear regression natural for online learning, where data arrives in a stream and a fixed-size posterior $(\boldsymbol{m}_N, \boldsymbol{S}_N)$ is all that need be carried.

6.4 The equivalent kernel

There is a hidden structure in the predictive mean worth surfacing, because it is the seed of Chapter 14. Substitute $\boldsymbol{m}_N = \beta \boldsymbol{S}_N \boldsymbol{\Phi}^\top \boldsymbol{t}$ into the predictive mean:

$$\boldsymbol{m}_N^\top \boldsymbol{\phi}(\boldsymbol{x}) = \beta\, \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{S}_N \boldsymbol{\Phi}^\top \boldsymbol{t} = \sum_{i=1}^{m} \underbrace{\beta\, \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_i)}_{k(\boldsymbol{x}, \boldsymbol{x}_i)}\, t_i = \sum_{i=1}^{m} k(\boldsymbol{x}, \boldsymbol{x}_i)\, t_i.$$

The prediction at a new $\boldsymbol{x}$ is a *weighted sum of the training targets*, with weights $k(\boldsymbol{x}, \boldsymbol{x}_i) = \beta \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{S}_N \boldsymbol{\phi}(\boldsymbol{x}_i)$ given by the *equivalent kernel*. The kernel is large when $\boldsymbol{x}$ and $\boldsymbol{x}_i$ have similar feature vectors and small when they do not, so the prediction at $\boldsymbol{x}$ leans on nearby training points, just as locally weighted regression (Section 5.8) and the kernel density estimate of Chapter 3 did. This is the first hint of a profound shift of viewpoint: a model written in terms of features $\boldsymbol{\phi}$ can be rewritten entirely in terms of a kernel k that compares inputs, and once it is, the features can disappear. Chapter 14 makes this the centerpiece of an entire family of methods.

6.5 The evidence and Occam's razor

We have treated the prior precision α and the noise precision β as known, but in practice they are hyperparameters to be set. The Bayesian answer is the *evidence*, the marginal likelihood of Chapter 1: the probability of the data with the weights integrated out,

$$p(\boldsymbol{t} \mid \alpha, \beta) = \int p(\boldsymbol{t} \mid \boldsymbol{w}, \beta)\, p(\boldsymbol{w} \mid \alpha)\, d\boldsymbol{w}. \tag{6.5}$$

Both factors are Gaussian, so the integral is a Gaussian convolution that can be done in closed form, giving an explicit function of α and β. Choosing the hyperparameters to maximize the evidence, the *evidence approximation* or *empirical Bayes*, tunes the model to the data without a separate validation set.

The reason this works, and does not simply prefer the most flexible model the way training error does, is Occam's razor, built into the evidence automatically. The evidence is a probability distribution over data sets, so it must integrate to one over all possible $\boldsymbol{t}$. A very flexible model (small α, weak prior, so the weights roam freely) can explain a huge variety of data sets, but precisely because its probability mass is spread thin over all of them, it assigns only a little to the particular data we observed. A very rigid model (large α) concentrates its mass on a narrow set of data sets and assigns high probability to those, but low probability to data outside its range. The evidence is largest for the model whose flexibility matches the data: complex enough to place the observed data in its high-probability region, simple enough not to have squandered its probability mass elsewhere. Maximizing the evidence therefore selects an intermediate complexity, the same sweet spot the bias-variance tradeoff and cross-validation of Chapter 4 sought, now found by a single integral. The same evidence framework, applied to neural networks, is the Bayesian model comparison of Chapter 9.

6.6 A sequential update worked end to end

The sections above derived every formula this chapter needs. To see them move, we now run the smallest nontrivial model, $y = wx$ with a single weight, all the way from the prior to a predictive band, putting numbers on Eq. (6.2) and Eq. (6.4). Dropping the intercept keeps the posterior one-dimensional, so each update is a single mean and a single variance that we can watch tighten.

For one weight the feature is $\boldsymbol{\phi}(x) = x$, the design matrix $\boldsymbol{\Phi}$ is the column of the x_n, and the posterior of Theorem 6.1 becomes scalar:

$$S_N^{-1} = \alpha + \beta \sum_n x_n^2, \qquad m_N = \beta\, S_N \sum_n x_n t_n. \tag{6.6}$$

The precision is the prior precision α plus a data term that only ever grows, so S_N can only shrink as points arrive. That monotone tightening is the whole story of sequential learning, and here it is just one number going down.

Example 6.4 (The posterior tightening one point at a time). Take prior precision $\alpha = 2$ and noise precision $\beta = 1$, with a true weight near $w = 1.5$. The prior is $p(w) = \mathcal{N}(w \mid 0, \frac{1}{2})$, a blob of variance $S_0 = 1/\alpha = 0.5$ (standard deviation 0.707) centered at the origin: before any data the model guesses $w = 0$ and is quite unsure.

First point $(x_1, t_1) = (1, 1.5)$. By Eq. (6.6),

$$S_1^{-1} = 2 + (1)(1^2) = 3, \qquad S_1 = \tfrac{1}{3} \approx 0.333,$$

and with $\sum x_n t_n = (1)(1.5) = 1.5$ the mean is $m_1 = (1)(\frac{1}{3})(1.5) = 0.5$. One point has pulled the estimate off the origin and cut the variance from 0.5 to 0.333.

Second point $(x_2, t_2) = (2, 3)$, consistent with the same line. Now

$$S_2^{-1} = 2 + (1)(1^2 + 2^2) = 7, \qquad S_2 = \tfrac{1}{7} \approx 0.143,$$

and $\sum x_n t_n = (1)(1.5) + (2)(3) = 7.5$, so $m_2 = (1)(\frac{1}{7})(7.5) = \frac{15}{14} \approx 1.071$. The mean has climbed toward the true 1.5 (the prior still pulls it down, honestly so on two points) and the standard deviation has fallen from 0.707 to 0.577 to 0.378. The posterior is contracting toward the data, exactly the picture Section 6.3 described in words.

The same answer comes out if we feed the points in one at a time, using the one-point posterior as the prior for the next. Treating $(S_1^{-1}, m_1) = (3, 0.5)$ as the prior and adding the second point,

$$S_2^{-1} = 3 + (1)(2^2) = 7, \qquad m_2 = S_2\big(S_1^{-1} m_1 + \beta\, x_2 t_2\big) = \tfrac{1}{7}\big(3(0.5) + 6\big) = \tfrac{7.5}{7} = \tfrac{15}{14},$$

identical to the batch result. Gaussian conditioning does not care about order, so streaming the data and processing it all at once land on the same posterior. ◊

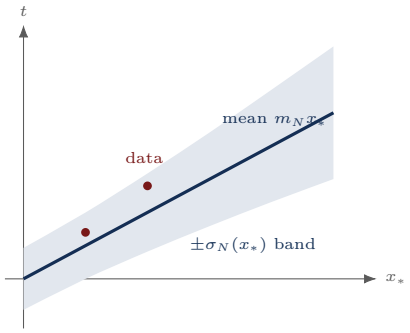


Figure 6.2. The predictive band of Example 6.6 for $y = wx$. The mean is the posterior-mean line $m_N x_*$ through the two training points (red), and the $\pm\sigma_N(x_*)$ band of Eq. (6.7) is tightest at the origin, where the single feature $\boldsymbol{\phi}(x) = x$ vanishes, and flares outward as x_* grows. The widening is the model-uncertainty term $x_*^2 S_2$; the floor is the noise $1/\beta$.

Remark 6.5 *(The posterior mean is ridge, on numbers).* The identity of Section 6.1, that the posterior mean equals ridge with $\lambda = \alpha/\beta$ (derived in Exercise 6.1), is easy to check on Example 6.4. Here $\lambda = \alpha/\beta = 2/1 = 2$, and ridge for a single feature is $w = (\sum_n x_n t_n)/(\sum_n x_n^2 + \lambda)$. With $\sum x_n t_n = 7.5$ and $\sum x_n^2 = 5$,

$$w_{\text{ridge}} = \frac{7.5}{5+2} = \frac{7.5}{7} = \frac{15}{14} \approx 1.071,$$

the same m_2 the posterior gave. The MAP estimate and the penalized least-squares fit are one point; the posterior simply wraps a calibrated variance around it.

Example 6.6 (The predictive band on numbers). Carry the posterior of Example 6.4 forward, $m_2 = \frac{15}{14}$, $S_2 = \frac{1}{7}$, $\beta = 1$, into the predictive distribution Eq. (6.4). With the single feature $\boldsymbol{\phi}(x_*) = x_*$, the predictive mean and variance are

$$m(x_*) = m_2\, x_* = \tfrac{15}{14}\, x_*, \qquad \sigma_N^2(x_*) = \frac{1}{\beta} + x_*^2\, S_2 = 1 + \frac{x_*^2}{7}. \tag{6.7}$$

The noise floor $1/\beta = 1$ is the same everywhere; the model-uncertainty term $x_*^2 S_2$ is what widens the band, and because the lone feature $\boldsymbol{\phi}(x) = x$ vanishes at the origin, the band is at its narrowest there rather than at the data centroid. Reading off three test points:

$$\sigma_N^2(0) = 1, \qquad \sigma_N^2(1.5) = 1 + \tfrac{2.25}{7} \approx 1.321,$$

and far outside the data, at $x_* = 5$, $\sigma_N^2(5) = 1 + \frac{25}{7} \approx 4.571$. The predictive standard deviation therefore grows from 1.00 at the origin, to 1.15 inside the data range, to 2.14 at $x_* = 5$, while the mean is only the straight line $\frac{15}{14} x_*$. The prediction at $x_* = 5$ carries roughly twice the error bar of the one at the origin, the model's honest admission that it has never seen data that far out. Figure 6.2 draws the band. ◊

Notes and References

The Gaussian posterior, the predictive distribution, the equivalent kernel, and the evidence framework all follow Sections 3.3–3.5 of Bishop [3], which also contains the celebrated figures of the sequential posterior contracting as data arrives. The identity of the posterior mean with the ridge solution ties this chapter back to Chapter 5 and to the MAP-is-regularization principle of Chapter 3. The equivalent kernel is the bridge to the kernel methods of Chapter 14 and, beyond them, to Gaussian processes, the nonparametric limit of Bayesian linear regression in which the feature space is taken to be infinite. The evidence approximation and its use for setting α and β are due to MacKay [26], whose Bayesian framework for neural networks returns in Chapter 9.

Exercises

Exercise 6.1. Derive the posterior covariance and mean Eq. (6.2) by completing the square in the log-posterior, and show the posterior mean equals the ridge solution with $\lambda = \alpha/\beta$.

Solution. The log-posterior is $-\frac{\beta}{2}(\boldsymbol{t}-\boldsymbol{\Phi w})^\top(\boldsymbol{t}-\boldsymbol{\Phi w}) - \frac{\alpha}{2}\boldsymbol{w}^\top\boldsymbol{w} + \text{const}$. The terms quadratic in $\boldsymbol{w}$ are $-\frac{1}{2}\boldsymbol{w}^\top(\alpha I + \beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi})\boldsymbol{w}$, so $\boldsymbol{S}_N^{-1} = \alpha I + \beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$. The terms linear in $\boldsymbol{w}$ are $\beta\boldsymbol{w}^\top\boldsymbol{\Phi}^\top\boldsymbol{t}$, which a Gaussian writes as $\boldsymbol{w}^\top\boldsymbol{S}_N^{-1}\boldsymbol{m}_N$, so $\boldsymbol{m}_N = \beta\boldsymbol{S}_N\boldsymbol{\Phi}^\top\boldsymbol{t} = \beta(\alpha I + \beta\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t} = (\frac{\alpha}{\beta}I + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{t}$, which is Eq. (5.7) with $\lambda = \alpha/\beta$. □

Exercise 6.2. For the posterior of Example 6.2 ($\boldsymbol{m}_N = (0, \frac{2}{3})$, $\boldsymbol{S}_N = \frac{1}{3}I$, $\beta = 1$), compute the predictive mean and variance at $x = 0$ and $x = 2$, and interpret the difference.

Solution. The predictive mean is $\boldsymbol{m}_N^\top(1, x) = \frac{2}{3}x$, so it is 0 at $x = 0$ and $\frac{4}{3}$ at $x = 2$. The predictive variance is $\sigma_N^2(x) = 1 + \frac{1}{3}(1 + x^2)$, so $\sigma_N^2(0) = \frac{4}{3}$ ($\sigma_N = 1.15$) and $\sigma_N^2(2) = 1 + \frac{5}{3} = \frac{8}{3}$ ($\sigma_N = 1.63$). The error bar grows as we move from the data center at $x = 0$ outward to $x = 2$, because the model-uncertainty term $\boldsymbol{\phi}(x)^\top\boldsymbol{S}_N\boldsymbol{\phi}(x) = \frac{1}{3}(1 + x^2)$ increases with $|x|$, while the noise floor $1/\beta = 1$ stays fixed. □

Exercise 6.3. Show that the predictive mean of Bayesian linear regression can be written as a weighted sum $\sum_i k(\boldsymbol{x}, \boldsymbol{x}_i)\, t_i$ of the training targets, and give the equivalent kernel k.

Solution. Substituting $\boldsymbol{m}_N = \beta\boldsymbol{S}_N\boldsymbol{\Phi}^\top\boldsymbol{t}$ into the predictive mean, $\boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}) = \beta\boldsymbol{\phi}(\boldsymbol{x})^\top\boldsymbol{S}_N\boldsymbol{\Phi}^\top\boldsymbol{t}$. Writing the matrix product $\boldsymbol{\Phi}^\top\boldsymbol{t} = \sum_i \boldsymbol{\phi}(\boldsymbol{x}_i)t_i$ as a sum over training points, $\boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}) = \sum_i \beta\boldsymbol{\phi}(\boldsymbol{x})^\top\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_i)\, t_i = \sum_i k(\boldsymbol{x}, \boldsymbol{x}_i)\, t_i$ with equivalent kernel $k(\boldsymbol{x}, \boldsymbol{x}_i) = \beta\boldsymbol{\phi}(\boldsymbol{x})^\top\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_i)$. It is large when $\boldsymbol{x}$ and $\boldsymbol{x}_i$ have similar features, so nearby training points dominate the prediction. □

Exercise 6.4. For the single-weight model $y = wx$ with prior $p(w) = \mathcal{N}(w \mid 0, \alpha^{-1})$, prior precision $\alpha = 1$, and noise precision $\beta = 2$, use the scalar update Eq. (6.6). Give the posterior mean and variance after the single point $(x_1, t_1) = (2, 2)$, then update with a second point $(x_2, t_2) = (1, 1)$ and give the new posterior. Confirm the second update gives the same answer whether you absorb both points at once or feed the second point to the one-point posterior.

Solution. After the first point, $S_1^{-1} = \alpha + \beta x_1^2 = 1 + 2(4) = 9$, so $S_1 = 1/9 \approx 0.111$, and $m_1 = \beta S_1(x_1 t_1) = 2(\frac{1}{9})(4) = \frac{8}{9} \approx 0.889$. Adding the second point in batch, $S_2^{-1} = 1 + 2(2^2 + 1^2) = 11$, so $S_2 = 1/11 \approx 0.091$, and with $\sum x_n t_n = (2)(2) + (1)(1) = 5$ the mean is $m_2 = 2(\frac{1}{11})(5) = \frac{10}{11} \approx 0.909$. Sequentially, treat $(S_1^{-1}, m_1) = (9, \frac{8}{9})$ as the prior: $S_2^{-1} = 9 + 2(1^2) = 11$, and $m_2 = S_2(S_1^{-1}m_1 + \beta x_2 t_2) = \frac{1}{11}(9 \cdot \frac{8}{9} + 2) = \frac{1}{11}(8 + 2) = \frac{10}{11}$, the same value. The variance falls from 1 to 1/9 to 1/11 and the mean climbs from 0 toward the true $w = 1$. □

Exercise 6.5. Using the posterior from Exercise 6.4 after both points ($m_2 = \frac{10}{11}$, $S_2 = \frac{1}{11}$, $\beta = 2$, feature $\boldsymbol{\phi}(x) = x$), compute the predictive mean and variance Eq. (6.4) at $x_* = 0$ and $x_* = 3$, and explain why the band is narrower at $x_* = 0$.

Solution. The predictive mean is $m_2 x_* = \frac{10}{11}x_*$ and the variance is $\sigma_N^2(x_*) = 1/\beta + x_*^2 S_2 = \frac{1}{2} + \frac{x_*^2}{11}$. At $x_* = 0$ the mean is 0 and the variance is $\frac{1}{2}$ ($\sigma_N \approx 0.707$). At $x_* = 3$ the mean is $\frac{30}{11} \approx 2.727$ and the variance is $\frac{1}{2} + \frac{9}{11} \approx 1.318$ ($\sigma_N \approx 1.148$). The band is narrower at $x_* = 0$ because the single feature $\boldsymbol{\phi}(x) = x$ vanishes there, so the model-uncertainty term $x_*^2 S_2$ is zero and only the noise floor $1/\beta$ remains; farther out the model term grows as x_*^2 and the band widens. □

Exercise 6.6. Explain why maximizing the evidence $p(\boldsymbol{t} \mid \alpha, \beta)$ does not simply select the most flexible model (smallest α), unlike maximizing the training likelihood.

Solution. The evidence integrates the likelihood over the prior, so it is a normalized distribution over data sets and must sum to one over all possible $\boldsymbol{t}$. A very flexible model (small α) can fit many data sets, but spreads its probability mass thinly across all of them, so it assigns little to the particular observed data. A rigid model concentrates mass narrowly, assigning high probability to a few data sets and near zero elsewhere. The evidence is maximized at the

intermediate complexity that places the observed data in a high-probability region without wasting mass on data sets that never occur, which is Occam's razor expressed quantitatively. Maximizing training likelihood lacks this normalization and so always favors more flexibility, leading to overfitting. □

PART III

Classification

CHAPTER 7

Linear Models for Classification

A classifier is a fence. The only questions are where to put it, which way it faces, and how to move it when it is wrong.

after Professor Peter Chin, ENGS 106

Regression predicts a number; classification predicts a label. The change of target, from a continuous t to a discrete class, is small to state and large in consequence, because the geometry becomes about *dividing* the input space rather than fitting a surface over it. A linear classifier divides the space with a flat boundary, a line in two dimensions, a plane in three, a hyperplane in general, and the art is choosing where that boundary goes. There are three ways to choose it, and they organize the next three chapters. A *discriminant function* maps an input straight to a class with no probabilities in between; that is this chapter, together with the perceptron, the first learning algorithm. A *generative* model describes how each class produces its data; a *discriminative* model describes the class probability directly. Those two are Chapter 8. We begin with the geometry, because every later method, logistic regression, the support vector machine, even a neuron, is in the end a decision about where to put a hyperplane.

Roadmap

Suggested reading: Bishop [3], Section 4.1; Deisenroth et al. [9], Section 9.2 for the geometry.

We cover: discriminant functions and the geometry of a decision boundary (Section 7.1); the multiclass problem and convex decision regions (Section 7.2); least squares for classification and why it is fragile (Section 7.3); Fisher's linear discriminant (Section 7.4); and the perceptron, with a convergence theorem and a full hand-traced run (Section 7.5).

7.1 Discriminant functions and the decision boundary

Take two classes, $\mathcal{C}_1$ and $\mathcal{C}_2$. A *linear discriminant* is an affine function of the input,

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0, \tag{7.1}$$

used by the rule: assign $\boldsymbol{x}$ to $\mathcal{C}_1$ if $y(\boldsymbol{x}) \geq 0$ and to $\mathcal{C}_2$ otherwise. The vector $\boldsymbol{w}$ is the *weight vector* and w_0 is the *bias*. The set of inputs exactly on the fence, $\{\boldsymbol{x} : y(\boldsymbol{x}) = 0\}$, is the *decision boundary*, and Eq. (7.1) makes it the hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + w_0 = 0$.

Two geometric facts about that hyperplane are used constantly, so we derive them. First, $\boldsymbol{w}$ is perpendicular to the boundary. If $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$ are two points on the boundary, then $y(\boldsymbol{x}_A) = y(\boldsymbol{x}_B) = 0$, so subtracting, $\boldsymbol{w}^\top(\boldsymbol{x}_A - \boldsymbol{x}_B) = 0$: the weight vector is orthogonal to every direction lying in the boundary, hence normal to it. Second, the value $y(\boldsymbol{x})$ measures signed distance to the boundary, scaled by $\|\boldsymbol{w}\|$. Write any point as its projection onto the boundary plus a displacement along the unit normal, $\boldsymbol{x} = \boldsymbol{x}_\perp + r\,\boldsymbol{w}/\|\boldsymbol{w}\|$, where $\boldsymbol{x}_\perp$ is on the boundary and r is the signed distance. Apply y:

$$y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0 = \boldsymbol{w}^\top \boldsymbol{x}_\perp + w_0 + r\,\frac{\boldsymbol{w}^\top \boldsymbol{w}}{\|\boldsymbol{w}\|} = \underbrace{y(\boldsymbol{x}_\perp)}_{=0} + r\,\|\boldsymbol{w}\| = r\,\|\boldsymbol{w}\|,$$

since $\boldsymbol{w}^\top \boldsymbol{w} = \|\boldsymbol{w}\|^2$ and $\boldsymbol{x}_\perp$ is on the boundary. Solving, the signed distance from $\boldsymbol{x}$ to the boundary is

$$r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|} = \frac{\boldsymbol{w}^\top \boldsymbol{x} + w_0}{\|\boldsymbol{w}\|}. \tag{7.2}$$

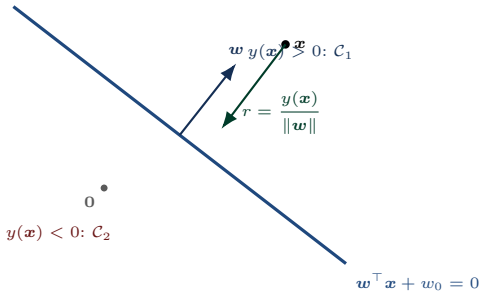


Figure 7.1. The decision boundary is the hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + w_0 = 0$. The weight vector $\boldsymbol{w}$ is normal to it, the value $y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + w_0$ is the signed distance to the boundary times $\|\boldsymbol{w}\|$, and the bias w_0 sets the boundary's offset $w_0/\|\boldsymbol{w}\|$ from the origin.

Setting $\boldsymbol{x} = \mathbf{0}$ shows the boundary's distance from the origin is $w_0/\|\boldsymbol{w}\|$, so the bias positions the fence and the weight vector aims it. Figure 7.1 collects the picture.

Example 7.1 (Distance to a boundary). Let the discriminant be $y(\boldsymbol{x}) = x_1 + x_2 - 2$, so $\boldsymbol{w} = (1, 1)$, $w_0 = -2$, and the boundary is the line $x_1 + x_2 = 2$. The norm is $\|\boldsymbol{w}\| = \sqrt{1^2 + 1^2} = \sqrt{2}$. At the point $\boldsymbol{x} = (2, 2)$ the discriminant is $y = 2 + 2 - 2 = 2 > 0$, so $\boldsymbol{x}$ is in $\mathcal{C}_1$, at signed distance $r = y/\|\boldsymbol{w}\| = 2/\sqrt{2} = \sqrt{2} \approx 1.41$ from the boundary. At the origin $\boldsymbol{x} = (0, 0)$, $y = -2 < 0$, so it is in $\mathcal{C}_2$, at signed distance $-2/\sqrt{2} = -\sqrt{2}$ (the sign records which side). The two points are equally far from the fence, on opposite sides. ◊

7.2 Multiple classes and convex decision regions

With $K > 2$ classes the obvious patches fail. A *one-versus-the-rest* scheme builds $K - 1$ binary classifiers, each separating one class from all others, but leaves regions claimed by several classifiers at once or by none. A *one-versus-one* scheme builds a classifier for every pair, $K(K-1)/2$ of them, and votes, but produces regions where the vote ties. Both leave *ambiguous regions* (Fig. 7.2).

The clean fix is a single discriminant per class,

$$y_k(\boldsymbol{x}) = \boldsymbol{w}_k^\top \boldsymbol{x} + w_{k0}, \qquad k = 1, \ldots, K, \tag{7.3}$$

with the rule: assign $\boldsymbol{x}$ to the class with the largest discriminant, $k^*(\boldsymbol{x}) = \arg\max_k y_k(\boldsymbol{x})$. The boundary between classes j and k is where their discriminants tie, $y_j(\boldsymbol{x}) = y_k(\boldsymbol{x})$, a hyperplane. This rule never leaves a point unassigned or doubly assigned, and its regions are well behaved.

Proposition 7.2 (Decision regions are convex). Under the rule $k^*(\boldsymbol{x}) = \arg\max_k y_k(\boldsymbol{x})$ with linear discriminants Eq. (7.3), each decision region $\mathcal{R}_k = \{\boldsymbol{x} : y_k(\boldsymbol{x}) \geq y_j(\boldsymbol{x})\ \forall j\}$ is convex (and connected).

Proof. Take two points $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$ both in $\mathcal{R}_k$, and any point on the segment between them, $\boldsymbol{x} = \eta \boldsymbol{x}_A + (1 - \eta)\boldsymbol{x}_B$ with $\eta \in [0, 1]$. Because each y_j is affine, it respects this combination: $y_j(\boldsymbol{x}) = \eta\, y_j(\boldsymbol{x}_A) + (1 - \eta)\, y_j(\boldsymbol{x}_B)$ for every j, including k. Since $\boldsymbol{x}_A, \boldsymbol{x}_B \in \mathcal{R}_k$ we have $y_k(\boldsymbol{x}_A) \geq y_j(\boldsymbol{x}_A)$ and $y_k(\boldsymbol{x}_B) \geq y_j(\boldsymbol{x}_B)$ for all j. Multiplying the first by $\eta \geq 0$, the second by $1 - \eta \geq 0$, and adding,

$$\eta\, y_k(\boldsymbol{x}_A) + (1{-}\eta)\, y_k(\boldsymbol{x}_B) \;\geq\; \eta\, y_j(\boldsymbol{x}_A) + (1{-}\eta)\, y_j(\boldsymbol{x}_B), \quad \text{that is} \quad y_k(\boldsymbol{x}) \geq y_j(\boldsymbol{x}) \text{ for all } j,$$

so $\boldsymbol{x} \in \mathcal{R}_k$. Every point between two members of $\mathcal{R}_k$ is also a member, which is the definition of convex. ■

Convexity is a sanity check we will want again: it means a linear classifier carves the input space into clean, contiguous territories with no islands, the kind of structure Fig. 7.2 shows on the right.

Figure 7.2. One-versus-rest classifiers leave a region claimed by both or neither (left). The single-discriminant-per-class rule $k^* = \arg\max_k y_k(\boldsymbol{x})$ partitions the space into convex regions meeting at a point (right), with no ambiguity.

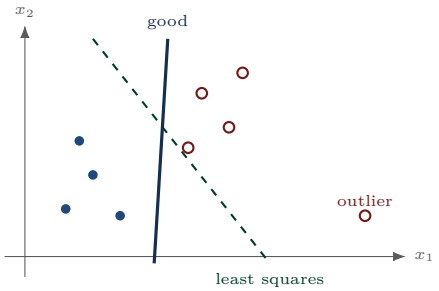


Figure 7.3. Two well-separated classes (filled blue, open red) and one distant red outlier. The good boundary (solid) cleanly separates the bulk; the least-squares boundary (dashed) is rotated toward the outlier, because squared error penalizes the outlier's large positive margin and tries to reduce it, eroding the separation of the bulk. Logistic regression (Chapter 8) keeps the boundary upright.

7.3 Least squares for classification, and why it is fragile

We already know how to fit linear models by least squares (Chapter 5), so a tempting shortcut is to reuse it for classification: encode the labels as numbers, say $t = +1$ for $\mathcal{C}_1$ and $t = -1$ for $\mathcal{C}_2$ (or one-hot vectors for K classes), and fit the discriminant $y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$ by minimizing the same sum of squared errors $\sum_i (\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_i) - t_i)^2$, with the closed-form solution $\boldsymbol{w} = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{t}$ from the normal equations. It sometimes works, and it is appealingly simple, but it is fragile, for a reason worth understanding because it explains why the rest of Part III exists.

The trouble is that squared error penalizes predictions that are *too correct*. A point of class $\mathcal{C}_1$ that sits far on the correct side of the boundary has $y(\boldsymbol{x})$ much larger than its target $t = +1$, so $(y(\boldsymbol{x}) - 1)^2$ is large, and least squares, trying to shrink that error, drags the boundary toward the well-classified point and away from where it belongs. A few outliers, points correct but distant, can therefore swing the boundary and misclassify points near it (Fig. 7.3). The deeper issue is a modeling mismatch: least squares is maximum likelihood under Gaussian noise (Chapter 5), and a binary label is not Gaussian. The right noise model for a binary target is Bernoulli, and using it leads to logistic regression in Chapter 8, which does not punish confident correct predictions and is robust to exactly the outliers that break least squares.

The drag is easiest to believe once you watch it happen on numbers.

Example 7.3 (An outlier moves the least-squares boundary). Fit a one-dimensional least-squares classifier $y(x) = wx + b$ to targets $t = \pm 1$, with class $+1$ at $x = 1, 2$ and class -1 at $x = -1, -2$. By symmetry the fit passes through the origin ($b = 0$) with slope $w = \sum x_n t_n / \sum x_n^2 = (1 + 2 + 1 + 2)/(1 + 4 + 1 + 4) = 6/10 = 0.6$, so the boundary $y = 0$ sits at $x = 0$, splitting the classes perfectly.

Now add a single correct-but-distant point: another class-$+1$ example at $x = 10$. The normal equations $\begin{bmatrix} \sum x^2 & \sum x \\ \sum x & N \end{bmatrix} \begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} \sum xt \\ \sum t \end{bmatrix}$ now read $\begin{bmatrix} 110 & 10 \\ 10 & 5 \end{bmatrix} \begin{bmatrix} w \\ b \end{bmatrix} = \begin{bmatrix} 16 \\ 1 \end{bmatrix}$, with determinant $110(5) - 10(10) = 450$, so

$$w = \frac{16(5) - 10(1)}{450} = \frac{70}{450} = 0.156, \qquad b = \frac{110(1) - 10(16)}{450} = \frac{-50}{450} = -0.111.$$

The boundary $y = 0$ has moved to $x = -b/w = 0.111/0.156 = 0.71$. A genuine class-+1 point at $x = 0.5$ now scores $y = 0.156(0.5) - 0.111 = -0.033 < 0$ and is *misclassified*, even though one correct point was all we added. Least squares could not leave the distant point's large positive margin alone, so it tilted the fence and sacrificed a point near the boundary. Logistic regression (Chapter 8), whose loss saturates instead of growing with the margin, does not move. ◊

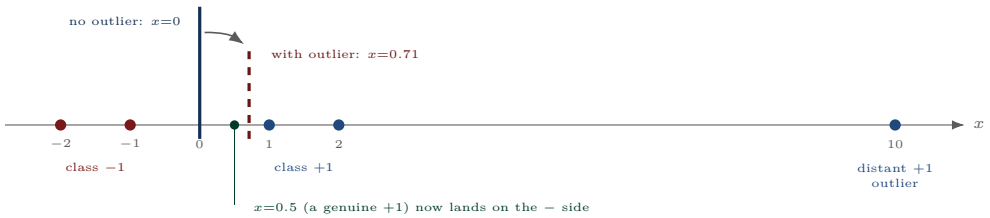


Figure 7.4. The least-squares classification boundary of Example 7.3. Without the distant point it sits at $x = 0$ and separates the classes; adding one correct-but-far +1 point at $x = 10$ pulls it to $x = 0.71$, which now misreads the genuine +1 point at $x = 0.5$. Squared error cannot ignore the outlier's large margin.

With more than two classes the same loss does something worse than tilt a fence: it can erase a class from the answer entirely. Encode K classes by one-hot targets and fit each column of the target by least squares; the argmax rule then reads off the largest discriminant. When the classes sit in a line, the middle one can fail to win *anywhere*, a defect known as *masking*, and it shows up cleanly on small numbers.

Example 7.4 (Least squares masks the middle class). Three classes on the line x, one point each, with one-hot targets: $\mathcal{C}_1$ at $x = -2$ with target $(1, 0, 0)$, $\mathcal{C}_2$ at $x = 0$ with target $(0, 1, 0)$, and $\mathcal{C}_3$ at $x = 2$ with target $(0, 0, 1)$. Fit $y_k(x) = w_{k0} + w_{k1}x$ for each class by least squares using $\boldsymbol{\phi}(x) = (1, x)$, so the design matrix and its normal matrix are

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & -2 \\ 1 & 0 \\ 1 & 2 \end{bmatrix}, \qquad \boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \begin{bmatrix} 3 & 0 \\ 0 & 8 \end{bmatrix}, \qquad (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/8 \end{bmatrix}.$$

The three target columns are $\boldsymbol{t}_1 = (1, 0, 0)$, $\boldsymbol{t}_2 = (0, 1, 0)$, $\boldsymbol{t}_3 = (0, 0, 1)$, giving $\boldsymbol{\Phi}^\top \boldsymbol{t}_1 = (1, -2)$, $\boldsymbol{\Phi}^\top \boldsymbol{t}_2 = (1, 0)$, and $\boldsymbol{\Phi}^\top \boldsymbol{t}_3 = (1, 2)$. Each weight vector is $\boldsymbol{w}_k = (\boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{t}_k$, so

$$\boldsymbol{w}_1 = (\tfrac{1}{3}, -\tfrac{1}{4}), \qquad \boldsymbol{w}_2 = (\tfrac{1}{3}, 0), \qquad \boldsymbol{w}_3 = (\tfrac{1}{3}, \tfrac{1}{4}),$$

that is $y_1(x) = \frac{1}{3} - \frac{1}{4}x$, $y_2(x) = \frac{1}{3}$, and $y_3(x) = \frac{1}{3} + \frac{1}{4}x$. Now apply the argmax rule. The middle discriminant y_2 is the flat constant $\frac{1}{3}$, while y_1 falls and y_3 rises with x. For $x < 0$ we have $y_1 > \frac{1}{3} > y_3$, so $\mathcal{C}_1$ wins; for $x > 0$, $y_3 > \frac{1}{3} > y_1$, so $\mathcal{C}_3$ wins; and the comparison $y_1 > y_3 \Leftrightarrow x < 0$ never lets $\mathcal{C}_2$ take the lead. The only place y_2 ties for first is the single point $x = 0$, where all three equal $\frac{1}{3}$. The middle class owns no interval at all: its region is masked.

Check it on the data. At $x = -2$: $y_1 = \frac{1}{3} + \frac{1}{2} = \frac{5}{6}$, $y_2 = \frac{1}{3}$, $y_3 = \frac{1}{3} - \frac{1}{2} = -\frac{1}{6}$, so $\mathcal{C}_1$, correct. At $x = 2$: by the mirror symmetry $\mathcal{C}_3$, correct. But the $\mathcal{C}_2$ training point at $x = 0$ lands in a three-way tie rather than a clean win, and any test point drawn from $\mathcal{C}_2$ is handed to a neighbour. The cause is the one from the two-class case: least squares forces each indicator to track a ±-style line, and a class boxed in by two others cannot be the largest of three lines anywhere on the interior. Softmax regression (Chapter 8) replaces the lines by competing exponentials and gives the middle class a region of its own. ◊

7.4 Fisher's linear discriminant

A different idea is to forget the boundary for a moment and ask for the best *direction* to project the data onto. Reducing each input $\boldsymbol{x}$ to a single number $z = \boldsymbol{w}^\top \boldsymbol{x}$ collapses the classification to one dimension, where a threshold separates the classes. Which direction $\boldsymbol{w}$ makes the two classes separate most cleanly after projection? Fisher's answer balances two competing goals: push the class *means* far apart, and keep each class's *spread* small, so the projected clouds do not overlap.

Let the two classes have means $\boldsymbol{m}_1$ and $\boldsymbol{m}_2$ and let S_W be the *within-class scatter*, the total spread of the points around their own class means, $S_W = \sum_{i \in \mathcal{C}_1} (\boldsymbol{x}_i - \boldsymbol{m}_1)(\boldsymbol{x}_i - \boldsymbol{m}_1)^\top + \sum_{i \in \mathcal{C}_2} (\boldsymbol{x}_i - \boldsymbol{m}_2)(\boldsymbol{x}_i - \boldsymbol{m}_2)^\top$. After projecting onto

$\boldsymbol{w}$, the squared distance between the projected means is $(\boldsymbol{w}^\top(\boldsymbol{m}_2 - \boldsymbol{m}_1))^2 = \boldsymbol{w}^\top S_B \boldsymbol{w}$ with the *between-class scatter* $S_B = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^\top$, and the projected within-class spread is $\boldsymbol{w}^\top S_W \boldsymbol{w}$. Fisher's criterion is their ratio,

$$J(\boldsymbol{w}) = \frac{\boldsymbol{w}^\top S_B\, \boldsymbol{w}}{\boldsymbol{w}^\top S_W\, \boldsymbol{w}}, \tag{7.4}$$

the between-class separation per unit of within-class spread, to be maximized.

Proposition 7.5 (Fisher's direction). The direction maximizing Eq. (7.4) is $\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$.

Proof. At a maximum the gradient of the ratio vanishes, which (by the quotient rule, or by noting the ratio is unchanged if we rescale $\boldsymbol{w}$) happens when the numerator and denominator gradients are parallel: $(\boldsymbol{w}^\top S_W \boldsymbol{w})\, S_B \boldsymbol{w} = (\boldsymbol{w}^\top S_B \boldsymbol{w})\, S_W \boldsymbol{w}$. The scalars $\boldsymbol{w}^\top S_W \boldsymbol{w}$ and $\boldsymbol{w}^\top S_B \boldsymbol{w}$ only rescale, and since $S_B \boldsymbol{w} = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^\top \boldsymbol{w}$ points along $(\boldsymbol{m}_2 - \boldsymbol{m}_1)$ for any $\boldsymbol{w}$, dropping the scalars leaves $S_W \boldsymbol{w} \propto (\boldsymbol{m}_2 - \boldsymbol{m}_1)$, that is $\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$. ■

The result is intuitive. If the within-class scatter were the identity, $S_W = I$, the best direction would be simply $\boldsymbol{m}_2 - \boldsymbol{m}_1$, the line joining the means. The factor S_W^{-1} corrects for unequal spread: it stretches the direction toward axes where the classes are *tight* (small within-class variance, hence discriminative) and away from axes where they are diffuse.

Example 7.6 (Fisher reweights toward the tight direction). Suppose two classes have means $\boldsymbol{m}_1 = (0, 0)$ and $\boldsymbol{m}_2 = (3, 3)$, so the difference is $\boldsymbol{m}_2 - \boldsymbol{m}_1 = (3, 3)$, and the within-class scatter (computed from the data) is the diagonal matrix $S_W = \left[\begin{smallmatrix} 2 & 0 \\ 0 & 0.5 \end{smallmatrix}\right]$, meaning the classes spread twice as much along x_1 as the unit and half as much along x_2. The naive direction joining the means is $(3, 3)$, equally weighted. Fisher instead computes $S_W^{-1} = \left[\begin{smallmatrix} 0.5 & 0 \\ 0 & 2 \end{smallmatrix}\right]$ and

$$\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1) = \begin{bmatrix} 0.5 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 1.5 \\ 6 \end{bmatrix} \propto (1, 4).$$

Fisher tilts the projection strongly toward x_2, because the classes are tight along x_2 (within-class variance 0.5) and that axis separates them more reliably; the diffuse x_1 axis is down-weighted. Projecting on $(1, 4)$ rather than $(1, 1)$ gives less overlap between the two projected clouds. ◊

Figure 7.5 shows why the correction matters. The two class clouds are wide along x_1 and narrow along x_2. Projecting onto the line joining the means (the naive direction) mixes in the wide x_1 spread, and the two projected clouds overlap; projecting onto Fisher's direction, tilted toward the tight x_2 axis, keeps them apart.

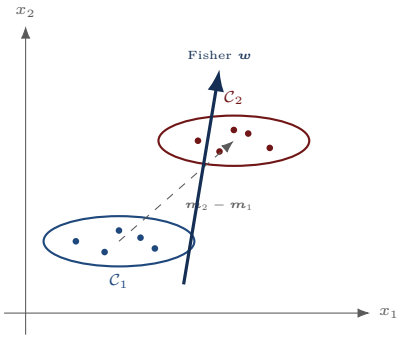


Figure 7.5. Both classes are spread wide along x_1 and tight along x_2. The naive direction $\boldsymbol{m}_2 - \boldsymbol{m}_1$ (dashed) drags the wide x_1 spread into the projection and the clouds overlap; Fisher's direction (solid), tilted by S_W^{-1} toward the tight x_2 axis, separates them. Fisher maximizes between-class separation per unit of within-class spread.

Remark 7.7 *(A purely one-dimensional classifier).* Once the data is projected to a single number $z = \boldsymbol{w}^\top \boldsymbol{x}$, choosing the threshold is itself a small problem, and Professor Chin poses a clean version of it. Sort the projected points and define the cumulative label count $F(z) = \sum_{n : z_n \le z} t_n$, with $t_n = \pm 1$. The threshold that best separates the classes is the z that extremizes F, since F rises through every positive point and falls through every negative one, and its extreme is where positives have accumulated on one side and negatives on the other. It is a reminder that even after the linear algebra picks a direction, classification ends in the humblest possible step: a threshold on the line.

The previous example took the scatter S_W as given. In practice S_W is built from the data, so it is worth doing the whole computation from raw points: form the class means, accumulate the within-class scatter as a sum of outer products, invert it, and apply it to the mean difference.

Example 7.8 (Fisher's direction from raw points). Three points per class. Class $\mathcal{C}_1$ is $(1,1)$, $(2,2)$, $(3,0)$, and class $\mathcal{C}_2$ is $(5,4)$, $(6,5)$, $(7,3)$. The class means average the coordinates,

$$\boldsymbol{m}_1 = \tfrac{1}{3}\big((1,1)+(2,2)+(3,0)\big) = (2,1),$$
$$\boldsymbol{m}_2 = \tfrac{1}{3}\big((5,4)+(6,5)+(7,3)\big) = (6,4).$$

The within-class scatter sums the outer products $(\boldsymbol{x}-\boldsymbol{m})(\boldsymbol{x}-\boldsymbol{m})^\top$ over each class. For $\mathcal{C}_1$ the centered points are $(1,1)-\boldsymbol{m}_1 = (-1,0)$, $(2,2)-\boldsymbol{m}_1 = (0,1)$, and $(3,0)-\boldsymbol{m}_1 = (1,-1)$, whose outer products add to

$$\left[\begin{smallmatrix}1&0\\0&0\end{smallmatrix}\right] + \left[\begin{smallmatrix}0&0\\0&1\end{smallmatrix}\right] + \left[\begin{smallmatrix}1&-1\\-1&1\end{smallmatrix}\right] = \left[\begin{smallmatrix}2&-1\\-1&2\end{smallmatrix}\right].$$

The centered points of $\mathcal{C}_2$ are $(-1,0)$, $(0,1)$, $(1,-1)$ as well (the class has the same shape, shifted), so its scatter is the same matrix, and

$$S_W = \left[\begin{smallmatrix}2&-1\\-1&2\end{smallmatrix}\right] + \left[\begin{smallmatrix}2&-1\\-1&2\end{smallmatrix}\right] = \left[\begin{smallmatrix}4&-2\\-2&4\end{smallmatrix}\right].$$

Its determinant is $4\cdot 4-(-2)(-2) = 12$, so the inverse is

$$S_W^{-1} = \frac{1}{12}\left[\begin{smallmatrix}4&2\\2&4\end{smallmatrix}\right] = \left[\begin{smallmatrix}1/3&1/6\\1/6&1/3\end{smallmatrix}\right].$$

The mean difference is $\boldsymbol{m}_2-\boldsymbol{m}_1 = (4,3)$, and Fisher's direction is

$$\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2-\boldsymbol{m}_1) = \begin{bmatrix}1/3&1/6\\1/6&1/3\end{bmatrix}\begin{bmatrix}4\\3\end{bmatrix} = \begin{bmatrix}4/3+1/2\\2/3+1\end{bmatrix} = \begin{bmatrix}11/6\\10/6\end{bmatrix} \propto (11,10).$$

The naive line joining the means points along $(4,3)$, slope $3/4 = 0.75$; Fisher tilts it to $(11,10)$, slope $10/11 \approx 0.91$. The off-diagonal -2 in S_W records that the two coordinates vary together within each class, and S_W^{-1} rotates the projection to undo that correlation, which the bare mean difference ignores. ◊

7.5 The perceptron

The perceptron of Rosenblatt [32] is the first learning *algorithm* in this book, the ancestor of the neural networks of Chapter 10, and it is pure boundary-pushing. It uses labels $t_n \in \{-1,+1\}$ and the discriminant $y(\boldsymbol{x}) = \boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x})$, classifying by its sign, $\hat{t} = \operatorname{sign}(\boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}))$, where the bias is folded into $\boldsymbol{w}$ by appending a constant feature $\phi_0 = 1$. A point is correctly classified exactly when $t_n\,\boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}_n) > 0$, the predicted sign agreeing with the label. The learning rule does nothing on correct points and, on a misclassified point, nudges $\boldsymbol{w}$ to fix it.

The perceptron update

Cycle through the data. For each example $(\boldsymbol{\phi}(\boldsymbol{x}_n), t_n)$, if it is *misclassified*, that is if $t_n\,\boldsymbol{w}^\top\boldsymbol{\phi}(\boldsymbol{x}_n) \le 0$, update

$$\boldsymbol{w} \;\leftarrow\; \boldsymbol{w} + \eta\, t_n\, \boldsymbol{\phi}(\boldsymbol{x}_n), \tag{7.5}$$

with learning rate $\eta > 0$; if it is correctly classified, leave $\boldsymbol{w}$ unchanged. Repeat until no point is misclassified.

The update is the natural correction. The change moves the discriminant at the offending point by $t_n\,\eta\,\boldsymbol{\phi}(\boldsymbol{x}_n)^\top\boldsymbol{\phi}(\boldsymbol{x}_n) = \eta\, t_n\|\boldsymbol{\phi}(\boldsymbol{x}_n)\|^2$, which has the sign of t_n, so it pushes $y(\boldsymbol{x}_n)$ in the direction of the correct label: up for a positive point, down for a negative one. Geometrically, the boundary rotates toward a misclassified positive point and away from a misclassified negative one. There are exactly three cases at each example: a correct point (no change); a positive point with $\hat{t} = -1$ (add $\eta\boldsymbol{\phi}(\boldsymbol{x}_n)$); a negative point with $\hat{t} = +1$ (subtract $\eta\boldsymbol{\phi}(\boldsymbol{x}_n)$).

Example 7.9 (Running the perceptron by hand). Take two points in the plane with a bias feature appended, so each input is $\boldsymbol{\phi}(\boldsymbol{x}) = (1, x_1, x_2)$. The data is

$$\boldsymbol{\phi}_1 = (1,1,1),\ t_1 = +1, \qquad \boldsymbol{\phi}_2 = (1,2,-2),\ t_2 = -1,$$

the learning rate is $\eta = 1$, and we start from $\boldsymbol{w} = (0, 0, 0)$.

Example 1. The score is $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = (0,0,0) \cdot (1,1,1) = 0$, and the classification test $t_1\, \boldsymbol{w}^\top \boldsymbol{\phi}_1 = (+1)(0) = 0 \leq 0$ counts as misclassified. Apply the update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\, t_1\, \boldsymbol{\phi}_1 = (0,0,0) + (1)(+1)(1,1,1) = (1,1,1)$.

Example 2. With $\boldsymbol{w} = (1,1,1)$, the score is $\boldsymbol{w}^\top \boldsymbol{\phi}_2 = (1,1,1) \cdot (1,2,-2) = 1 + 2 - 2 = 1$, and the test $t_2\, \boldsymbol{w}^\top \boldsymbol{\phi}_2 = (-1)(1) = -1 \leq 0$ is misclassified (the point is predicted positive but is negative). Update $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\, t_2\, \boldsymbol{\phi}_2 = (1,1,1) + (1)(-1)(1,2,-2) = (1,1,1) + (-1,-2,2) = (0,-1,3)$.

Check. Sweep again with $\boldsymbol{w} = (0,-1,3)$. For example 1, $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = 0 - 1 + 3 = 2 > 0$, predicted $+1$, matches $t_1 = +1$: correct. For example 2, $\boldsymbol{w}^\top \boldsymbol{\phi}_2 = 0 - 2 - 6 = -8 < 0$, predicted -1, matches $t_2 = -1$: correct. No point is misclassified, so the perceptron has converged after two updates, one of each sign. The final boundary is $-x_1 + 3x_2 = 0$, the line $x_2 = x_1/3$, with the positive point above it and the negative point below (Fig. 7.6). ◊

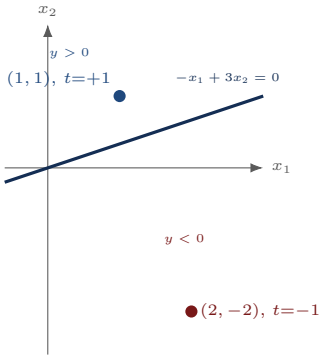


Figure 7.6. The boundary $-x_1 + 3x_2 = 0$ found by the hand-traced perceptron of Example 7.9 after two updates. The positive point $(1,1)$ lies in the half-plane $y > 0$ and the negative point $(2,-2)$ in $y < 0$, so both are correctly classified and the algorithm stops.

A larger run shows the sweeps. Take four augmented points $\boldsymbol{\phi} = (1, x_1, x_2)$: $\boldsymbol{\phi}_1 = (1,2,0)$ and $\boldsymbol{\phi}_2 = (1,0,2)$ with $t = +1$, and $\boldsymbol{\phi}_3 = (1,-2,0)$ and $\boldsymbol{\phi}_4 = (1,0,-2)$ with $t = -1$; start at $\boldsymbol{w} = (0,0,0)$, $\eta = 1$. *Sweep 1:* point 1 scores 0 (≤ 0, misclassified), update to $\boldsymbol{w} = (0,0,0) + (1,2,0) = (1,2,0)$; point 2 scores $\boldsymbol{w}^\top \boldsymbol{\phi}_2 = 1 > 0$, correct; point 3 scores $1 - 4 = -3$, and $t_3(-3) = 3 > 0$, correct; point 4 scores 1, and $t_4(1) = -1 \leq 0$, misclassified, update to $\boldsymbol{w} = (1,2,0) - (1,0,-2) = (0,2,2)$. *Sweep 2:* the four scores are now $4, 4, -4, -4$, each with the right sign, so no point updates and the algorithm stops. Two updates produced $\boldsymbol{w} = (0,2,2)$, the boundary $x_1 + x_2 = 0$, with both positive points on the side $x_1 + x_2 > 0$ and both negatives on the other.

Does this process always stop? When the data can be separated by a hyperplane, yes.

Theorem 7.10 (Perceptron convergence). If the training data is linearly separable, then there is a margin $\gamma > 0$ and a unit weight vector $\boldsymbol{w}^*$ with $t_n\, \boldsymbol{w}^{*\top} \boldsymbol{\phi}(\boldsymbol{x}_n) \geq \gamma$ for all n, and the perceptron makes at most $(R/\gamma)^2$ updates before classifying every point correctly, where $R = \max_n \|\boldsymbol{\phi}(\boldsymbol{x}_n)\|$.

Proof idea. Track the alignment $\boldsymbol{w}^{*\top} \boldsymbol{w}$ between the running weights and the separator. Each update adds $t_n \boldsymbol{\phi}(\boldsymbol{x}_n)$ to $\boldsymbol{w}$, so the alignment grows by $t_n\, \boldsymbol{w}^{*\top} \boldsymbol{\phi}(\boldsymbol{x}_n) \geq \gamma$ every update: after M updates, $\boldsymbol{w}^{*\top} \boldsymbol{w} \geq M\gamma$. Meanwhile the squared length $\|\boldsymbol{w}\|^2$ grows slowly, because an update happens only on a misclassified point ($t_n \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) \leq 0$), so $\|\boldsymbol{w} + t_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|^2 = \|\boldsymbol{w}\|^2 + 2 t_n \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + \|\boldsymbol{\phi}(\boldsymbol{x}_n)\|^2 \leq \|\boldsymbol{w}\|^2 + R^2$: after M updates $\|\boldsymbol{w}\|^2 \leq MR^2$. Combining, the cosine between $\boldsymbol{w}$ and $\boldsymbol{w}^*$ is at least $M\gamma / (\sqrt{M} R) = \sqrt{M}\, \gamma / R$, which cannot exceed 1, forcing $M \leq (R/\gamma)^2$. ■

The bound says a wide margin (large γ) and small data (small R) make the perceptron converge fast, an intuition the support vector machine of Chapter 15 will turn into an objective by *maximizing* the margin directly. The perceptron has two real limitations that motivate the rest of Part III. If the data is not linearly separable, it never stops, cycling forever. And even when it succeeds, it returns just a boundary, with no probability and no notion of confidence: it cannot say a point near the fence is a closer call than one far from it. The probabilistic classifiers of Chapter 8 supply exactly what it lacks.

The bound $(R/\gamma)^2$ is not just an existence claim; it is a number one can compute before training and then check against the run.

Example 7.11 (The convergence bound on numbers). Four augmented points $\boldsymbol{\phi} = (1, x_1, x_2)$: $\boldsymbol{\phi}_1 = (1, 2, 1)$ and $\boldsymbol{\phi}_2 = (1, 1, 2)$ with $t = +1$, and $\boldsymbol{\phi}_3 = (1, -2, -1)$ and $\boldsymbol{\phi}_4 = (1, -1, -2)$ with $t = -1$. First the two quantities in the bound. The data radius is the largest feature norm; every point has $\|\boldsymbol{\phi}_n\|^2 = 1 + 4 + 1 = 6$, so $R = \sqrt{6} \approx 2.449$. For the margin, take the separator $\boldsymbol{w}^* \propto (0, 1, 1)$, which classifies by the sign of $x_1 + x_2$. Its raw scores $t_n(x_1 + x_2)$ are $3, 3, 3, 3$, and normalizing by $\|(0, 1, 1)\| = \sqrt{2}$ gives the unit-vector margin

$$\gamma = \min_n t_n \, \boldsymbol{w}^{*\top} \boldsymbol{\phi}_n / \|(0, 1, 1)\| = 3/\sqrt{2} \approx 2.121.$$

The bound on the number of updates is therefore

$$\left(\frac{R}{\gamma}\right)^2 = \frac{6}{(3/\sqrt{2})^2} = \frac{6}{9/2} = \frac{12}{9} = \frac{4}{3} \approx 1.33,$$

so the perceptron must finish in at most one update. Run it with $\eta = 1$ from $\boldsymbol{w} = (0, 0, 0)$. Point 1 scores $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = 0$, and $t_1 \cdot 0 = 0 \le 0$, so it updates to $\boldsymbol{w} = (0, 0, 0) + (1, 2, 1) = (1, 2, 1)$. The remaining points already pass: $\boldsymbol{w}^\top \boldsymbol{\phi}_2 = 1 + 2 + 2 = 5$ with $t_2 = +1$; $\boldsymbol{w}^\top \boldsymbol{\phi}_3 = 1 - 4 - 1 = -4$ with $t_3 = -1$; $\boldsymbol{w}^\top \boldsymbol{\phi}_4 = 1 - 2 - 2 = -3$ with $t_4 = -1$. A second sweep rechecks point 1, $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = 1 + 4 + 1 = 6 > 0$, correct, and nothing changes. The run took exactly one update, matching the bound's ceiling of $\lfloor 4/3 \rfloor = 1$. A wide margin relative to the data radius is what makes the guarantee small, the same quantity the support vector machine of Chapter 15 maximizes on purpose. ◊

Notes and References

The discriminant-function geometry, the convexity of decision regions, the fragility of least squares for classification, and Fisher's linear discriminant follow Section 4.1 of Bishop [3]. Fisher's criterion is from Fisher [11], the same paper that introduced the iris data set still used to teach classification. The perceptron and its learning rule are due to Rosenblatt [32]; the convergence theorem of Theorem 7.10 is Novikoff's 1962 result, presented here in the margin-based form that anticipates the support vector machine of Chapter 15. The threshold rule of Remark 7.7 is a device of Professor Chin's lectures. The probabilistic alternatives this chapter motivates, logistic regression and the generative models, are the subject of Chapter 8.

Exercises

Exercise 7.1. A linear discriminant is $y(\boldsymbol{x}) = 2x_1 - x_2 + 1$. (a) Write the decision boundary as an equation and state the weight vector and bias. (b) Classify the points $(0, 0)$ and $(1, 4)$ and give the signed distance of each to the boundary.

Solution. (a) The boundary is the set where $y(\boldsymbol{x}) = 0$, that is $2x_1 - x_2 + 1 = 0$, with weight vector $\boldsymbol{w} = (2, -1)$ and bias $w_0 = 1$; its norm is $\|\boldsymbol{w}\| = \sqrt{2^2 + (-1)^2} = \sqrt{5}$. (b) At $(0, 0)$: $y = 2(0) - 0 + 1 = 1 > 0$, so class $\mathcal{C}_1$, at signed distance $y/\|\boldsymbol{w}\| = 1/\sqrt{5} \approx 0.45$. At $(1, 4)$: $y = 2(1) - 4 + 1 = -1 < 0$, so class $\mathcal{C}_2$, at signed distance $-1/\sqrt{5} \approx -0.45$. The points lie on opposite sides, equidistant from the fence. □

Exercise 7.2. Prove that the decision region $\mathcal{R}_k = \{\boldsymbol{x} : y_k(\boldsymbol{x}) \ge y_j(\boldsymbol{x}) \; \forall j\}$ of the argmax rule with linear discriminants $y_j(\boldsymbol{x}) = \boldsymbol{w}_j^\top \boldsymbol{x} + w_{j0}$ is convex.

Solution. Let $\boldsymbol{x}_A, \boldsymbol{x}_B \in \mathcal{R}_k$ and $\boldsymbol{x} = \eta \boldsymbol{x}_A + (1-\eta)\boldsymbol{x}_B$ for $\eta \in [0, 1]$. Each y_j is affine, so $y_j(\boldsymbol{x}) = \eta \, y_j(\boldsymbol{x}_A) + (1-\eta) \, y_j(\boldsymbol{x}_B)$. Since $y_k(\boldsymbol{x}_A) \ge y_j(\boldsymbol{x}_A)$ and $y_k(\boldsymbol{x}_B) \ge y_j(\boldsymbol{x}_B)$ for all j, multiplying by $\eta \ge 0$ and $1 - \eta \ge 0$ and adding gives $y_k(\boldsymbol{x}) \ge y_j(\boldsymbol{x})$ for all j, so $\boldsymbol{x} \in \mathcal{R}_k$. Thus the segment between any two points of $\mathcal{R}_k$ lies in $\mathcal{R}_k$, which is convexity. □

Exercise 7.3. Two classes have means $\boldsymbol{m}_1 = (1, 1)$ and $\boldsymbol{m}_2 = (5, 3)$ and a common within-class scatter $S_W = \left[\begin{smallmatrix} 4 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Find Fisher's discriminant direction, and compare it with the direction joining the means.

Solution. The direction joining the means is $\boldsymbol{m}_2 - \boldsymbol{m}_1 = (5 - 1, \; 3 - 1) = (4, 2)$. Fisher's direction reweights by the inverse within-class scatter, $\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$. Here $S_W^{-1} = \left[\begin{smallmatrix} 1/4 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, so $\boldsymbol{w} \propto \left[\begin{smallmatrix} 1/4 & 0 \\ 0 & 1 \end{smallmatrix}\right](4, 2) = (1, 2)$. Compared with

the mean-joining direction $(4,2) \propto (2,1)$, Fisher tilts toward x_2, because the classes are four times tighter along x_2 (scatter 1) than along x_1 (scatter 4), making x_2 the more reliable axis for separation. □

Exercise 7.4. Run the perceptron with learning rate $\eta = 1$ from $\boldsymbol{w} = (0,0,0)$ on the two augmented points $\boldsymbol{\phi}_1 = (1,2,1)$ with $t_1 = +1$ and $\boldsymbol{\phi}_2 = (1,-1,-1)$ with $t_2 = -1$. Show each update and verify convergence.

Solution. A point is misclassified when $t_n\, \boldsymbol{w}^\top \boldsymbol{\phi}_n \le 0$, and the update is $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\, t_n\, \boldsymbol{\phi}_n$. Start $\boldsymbol{w} = (0,0,0)$. *Example 1:* $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = 0$, and $t_1\, \boldsymbol{w}^\top \boldsymbol{\phi}_1 = 0 \le 0$, so misclassified; update $\boldsymbol{w} \leftarrow (0,0,0) + (+1)(1,2,1) = (1,2,1)$. *Example 2:* $\boldsymbol{w}^\top \boldsymbol{\phi}_2 = (1,2,1)\cdot(1,-1,-1) = 1-2-1 = -2$, and $t_2\, \boldsymbol{w}^\top \boldsymbol{\phi}_2 = (-1)(-2) = 2 > 0$, so already correct, no update. *Re-check example 1:* $\boldsymbol{w}^\top \boldsymbol{\phi}_1 = (1,2,1)\cdot(1,2,1) = 1+4+1 = 6 > 0$, predicted $+1 = t_1$: correct. Both points are correctly classified after a single update, $\boldsymbol{w} = (1,2,1)$, so the perceptron has converged. □

Exercise 7.5. Explain why the perceptron fails to converge on the XOR data, the four points $(0,0)$ and $(1,1)$ with label -1 and the points $(0,1)$ and $(1,0)$ with label $+1$, connecting your answer to the VC-dimension discussion of Chapter 4.

Solution. XOR is not linearly separable: the two positive points $(0,1),(1,0)$ lie on one diagonal and the two negatives $(0,0),(1,1)$ on the other, and no single straight line can put both positives on one side and both negatives on the other (this is the four-point shattering obstruction that fixes the VC dimension of lines in the plane at three, Chapter 4). Since the perceptron convergence theorem requires linear separability, and XOR violates it, there is no weight vector with all four margins positive; the perceptron will keep finding a misclassified point and updating forever, cycling without settling. Separating XOR requires a nonlinear feature map or a multilayer network, which is exactly the step taken in Chapter 10. □

Exercise 7.6. Class $\mathcal{C}_1$ has the points $(-2,-1)$, $(2,-1)$, $(0,2)$ and class $\mathcal{C}_2$ has $(1,2)$, $(5,2)$, $(3,5)$. Compute the class means, the within-class scatter S_W as a sum of outer products, and Fisher's direction $\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1)$. Compare it with the direction joining the means.

Solution. The means are $\boldsymbol{m}_1 = \frac{1}{3}((-2,-1)+(2,-1)+(0,2)) = (0,0)$ and $\boldsymbol{m}_2 = \frac{1}{3}((1,2)+(5,2)+(3,5)) = (3,3)$. Centering $\mathcal{C}_1$ gives $(-2,-1)$, $(2,-1)$, $(0,2)$, with outer products $\left[\begin{smallmatrix} 4 & 2 \\ 2 & 1 \end{smallmatrix}\right]$, $\left[\begin{smallmatrix} 4 & -2 \\ -2 & 1 \end{smallmatrix}\right]$, and $\left[\begin{smallmatrix} 0 & 0 \\ 0 & 4 \end{smallmatrix}\right]$, which sum to $\left[\begin{smallmatrix} 8 & 0 \\ 0 & 6 \end{smallmatrix}\right]$. Class $\mathcal{C}_2$ has the same centered points $(-2,-1)$, $(2,-1)$, $(0,2)$ shifted by $\boldsymbol{m}_2$, so its scatter is identical, and $S_W = \left[\begin{smallmatrix} 8 & 0 \\ 0 & 6 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 8 & 0 \\ 0 & 6 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 16 & 0 \\ 0 & 12 \end{smallmatrix}\right]$. Then $S_W^{-1} = \left[\begin{smallmatrix} 1/16 & 0 \\ 0 & 1/12 \end{smallmatrix}\right]$, and with $\boldsymbol{m}_2 - \boldsymbol{m}_1 = (3,3)$,

$$\boldsymbol{w} \propto S_W^{-1}(\boldsymbol{m}_2 - \boldsymbol{m}_1) = \left(\tfrac{3}{16},\ \tfrac{3}{12}\right) = \left(\tfrac{3}{16},\ \tfrac{4}{16}\right) \propto (3,4).$$

The mean-joining direction is $(3,3) \propto (1,1)$, slope 1; Fisher tilts to $(3,4)$, slope $4/3$, leaning toward x_2 because the within-class spread is smaller along x_2 (scatter 12) than along x_1 (scatter 16). □

Exercise 7.7. For the augmented points $\boldsymbol{\phi}_1 = (1,1,0)$, $\boldsymbol{\phi}_2 = (1,0,1)$ with $t = +1$ and $\boldsymbol{\phi}_3 = (1,-1,0)$, $\boldsymbol{\phi}_4 = (1,0,-1)$ with $t = -1$, take the separator $\boldsymbol{w}^* \propto (0,1,1)$. Compute $R = \max_n \|\boldsymbol{\phi}_n\|$, the unit-vector margin γ, and the bound $(R/\gamma)^2$. Then run the perceptron ($\eta = 1$, $\boldsymbol{w} = (0,0,0)$) and compare the number of updates with the bound.

Solution. Every point has $\|\boldsymbol{\phi}_n\|^2 = 1+1 = 2$, so $R = \sqrt{2}$. The raw scores $t_n\, \boldsymbol{w}^{*\top} \boldsymbol{\phi}_n$ are all 1 (for example $t_1(x_1+x_2) = (+1)(1) = 1$), and dividing by $\|(0,1,1)\| = \sqrt{2}$ gives $\gamma = 1/\sqrt{2}$. The bound is $(R/\gamma)^2 = 2/(1/2) = 4$. Running the perceptron: *Sweep 1.* Point 1 scores 0, updates to $\boldsymbol{w} = (1,1,0)$; point 2 scores $1 > 0$, correct; point 3 scores $1-1 = 0$ with $t_3 = -1$, $0 \le 0$, updates to $\boldsymbol{w} = (1,1,0) - (1,-1,0) = (0,2,0)$; point 4 scores 0 with $t_4 = -1$, updates to $\boldsymbol{w} = (0,2,0) - (1,0,-1) = (-1,2,1)$. *Sweep 2.* Point 1 scores $-1+2 = 1 > 0$, correct; point 2 scores $-1+1 = 0$ with $t_2 = +1$, updates to $\boldsymbol{w} = (0,2,2)$; points 3 and 4 score -2 with the right sign, correct. *Sweep 3.* All four score ± 2 correctly, so the run stops. It took four updates, exactly the bound $(R/\gamma)^2 = 4$, so the guarantee is tight here. □

Exercise 7.8. A three-class problem uses the linear discriminants $y_1(\boldsymbol{x}) = x_1 + 2x_2$, $y_2(\boldsymbol{x}) = 2x_1 + x_2$, and $y_3(\boldsymbol{x}) = x_1 + x_2 + 1$, with the rule $k^* = \arg\max_k y_k(\boldsymbol{x})$. Classify the points $(1,2)$ and $(2,0)$, showing all three discriminant values.

Solution. At $(1, 2)$: $y_1 = 1 + 4 = 5$, $y_2 = 2 + 2 = 4$, $y_3 = 1 + 2 + 1 = 4$. The largest is y_1, so the point is class 1. At $(2, 0)$: $y_1 = 2 + 0 = 2$, $y_2 = 4 + 0 = 4$, $y_3 = 2 + 0 + 1 = 3$. The largest is y_2, so class 2. The winning discriminant changes with the point, and the boundary between classes j and k is the line $y_j(\boldsymbol{x}) = y_k(\boldsymbol{x})$; for classes 1 and 2 that is $x_1 + 2x_2 = 2x_1 + x_2$, i.e. $x_2 = x_1$. □

CHAPTER 8

Probabilistic Classification

The perceptron tells you which side of the fence a point is on. A probabilistic classifier tells you how sure it is, and that is usually what you actually need.

after Professor Peter Chin, ENGS 106

Chapter 7 drew decision boundaries but reported no probabilities: the perceptron labels a point and says nothing about confidence, and least squares for classification is fragile. This chapter fixes both by modeling probability directly, and it does so along two routes that have competed since the beginning of the subject. The *generative* route models how each class produces its data, $p(\boldsymbol{x} \mid \mathcal{C}_k)$, and turns that around with Bayes' theorem to get the class posterior; Gaussian discriminant analysis and naive Bayes live here. The *discriminative* route models the posterior $p(\mathcal{C}_k \mid \boldsymbol{x})$ directly, without a story for the data; logistic regression lives here. The two routes meet at a beautiful fact: in both, the posterior comes out as the sigmoid (or softmax) of a linear function, exactly the inverse links of the exponential family from Chapter 3. Logistic regression turns out to be the simplest neural network, the gateway to all of Part IV, and we close with how to measure a classifier once it is built.

Roadmap

Suggested reading: Bishop [3], Sections 4.2–4.3; Deisenroth et al. [9], Chapter 9.

We cover: the generative and discriminative routes (Section 8.1); why the class posterior is a sigmoid (Section 8.2); Gaussian discriminant analysis (Section 8.3); naive Bayes with Laplace smoothing (Section 8.4); logistic regression, its convex cross-entropy loss, and its gradient (Section 8.5); iterative reweighted least squares (Section 8.6); multiclass softmax regression (Section 8.7); and evaluating classifiers with ROC curves (Section 8.8).

8.1 Two routes to a class posterior

What we want is the posterior probability $p(\mathcal{C}_k \mid \boldsymbol{x})$ that an input $\boldsymbol{x}$ belongs to class $\mathcal{C}_k$; the decision is then to pick the most probable class, the Bayes classifier of Chapter 1. There are two ways to get there. The *generative* approach learns a full model of each class, the class prior $p(\mathcal{C}_k)$ and the class-conditional density $p(\boldsymbol{x} \mid \mathcal{C}_k)$, and then inverts it with Bayes' theorem,

$$p(\mathcal{C}_k \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \mathcal{C}_k)\, p(\mathcal{C}_k)}{\sum_j p(\boldsymbol{x} \mid \mathcal{C}_j)\, p(\mathcal{C}_j)}.$$

It is called generative because, having $p(\boldsymbol{x} \mid \mathcal{C}_k)$, you can *generate* synthetic examples of each class, and it handles missing features and novelty naturally. The price is that modeling the full density of the inputs is hard, especially in high dimension. The *discriminative* approach skips the data model entirely and parameterizes $p(\mathcal{C}_k \mid \boldsymbol{x})$ directly, fitting it to the labels. It makes fewer assumptions and is often more accurate for the classification task itself, at the cost of being unable to generate data or flag an input that looks like nothing it was trained on. Table 8.1 contrasts them; the rest of the chapter develops one method of each kind and shows they share a form.

	Generative	**Discriminative**
Models	$p(\boldsymbol{x} \mid \mathcal{C}_k)$ and $p(\mathcal{C}_k)$	$p(\mathcal{C}_k \mid \boldsymbol{x})$ directly
Examples	GDA, naive Bayes	logistic regression
Can generate data?	yes	no
Assumptions	more (a full data model)	fewer
Strength	missing data, novelty, few samples	accuracy on the task

Table 8.1. Two routes to the class posterior. Generative models describe how data is produced and invert with Bayes; discriminative models fit the posterior directly.

8.2 The class posterior is a sigmoid

Before committing to any model of the data, one general fact organizes everything. For two classes, write out the posterior from Bayes' theorem and divide the top and bottom by the numerator:

$$p(\mathcal{C}_1 \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x} \mid \mathcal{C}_2)p(\mathcal{C}_2)} = \frac{1}{1 + \dfrac{p(\boldsymbol{x} \mid \mathcal{C}_2)p(\mathcal{C}_2)}{p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1)}} = \frac{1}{1 + e^{-a}} = \sigma(a),$$

where the last step *defines* a as the log of the ratio that appears in the denominator,

$$a(\boldsymbol{x}) = \log \frac{p(\boldsymbol{x} \mid \mathcal{C}_1)\, p(\mathcal{C}_1)}{p(\boldsymbol{x} \mid \mathcal{C}_2)\, p(\mathcal{C}_2)}. \tag{8.1}$$

This a is the *log-odds* of class 1 against class 2, and the posterior is the logistic sigmoid of it. This is the same sigmoid that emerged in Chapter 3 as the inverse link of the Bernoulli, and it is no coincidence: the log-odds is the natural parameter of the class label. For K classes the same computation gives the softmax, $p(\mathcal{C}_k \mid \boldsymbol{x}) = e^{a_k} / \sum_j e^{a_j}$ with $a_k = \log[p(\boldsymbol{x} \mid \mathcal{C}_k)p(\mathcal{C}_k)]$. So whatever generative model we choose, the posterior will be a sigmoid or softmax of some function $a(\boldsymbol{x})$; the model only determines what that function is. We now pick a model and find out.

8.3 Gaussian discriminant analysis

The simplest generative model takes each class-conditional density to be Gaussian. In *Gaussian discriminant analysis* (GDA) the two classes share one covariance,

$$p(\boldsymbol{x} \mid \mathcal{C}_k) = \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}), \quad k = 1, 2, \qquad p(\mathcal{C}_1) = \phi,\ p(\mathcal{C}_2) = 1 - \phi.$$

Substitute these into the log-odds Eq. (8.1). Writing out each Gaussian log-density, $\log \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_k) + \text{const}$, the log-odds is

$$a(\boldsymbol{x}) = -\tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_1)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_1) + \tfrac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_2)^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}_2) + \log \frac{\phi}{1 - \phi}.$$

Expand both quadratics. The term $\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x}$ appears in each with opposite sign and *cancels*, because the two classes share $\boldsymbol{\Sigma}$. What survives is linear in $\boldsymbol{x}$:

$$a(\boldsymbol{x}) = \underbrace{(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^\top \boldsymbol{\Sigma}^{-1}}_{\boldsymbol{w}^\top} \boldsymbol{x} \underbrace{- \tfrac{1}{2}\boldsymbol{\mu}_1^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_1 + \tfrac{1}{2}\boldsymbol{\mu}_2^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{\mu}_2 + \log \tfrac{\phi}{1-\phi}}_{w_0}, \tag{8.2}$$

so $p(\mathcal{C}_1 \mid \boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + w_0)$ with $\boldsymbol{w} = \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$. The decision boundary, where the sigmoid equals $\frac{1}{2}$, is the hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + w_0 = 0$: GDA with a shared covariance is a *linear* classifier, and it even produces the same sigmoid-of-a-line form that logistic regression will assume directly. If the two classes are allowed *different* covariances, the $\boldsymbol{x}^\top \boldsymbol{\Sigma}^{-1} \boldsymbol{x}$ terms no longer cancel and the boundary becomes quadratic, which is *quadratic discriminant analysis*.

The parameters are fit by maximum likelihood, and the estimates are exactly what you would guess: the class frequencies and the class sample means, with a single pooled covariance,

$$\hat{\phi} = \frac{N_1}{N}, \quad \hat{\boldsymbol{\mu}}_k = \frac{1}{N_k} \sum_{i \in \mathcal{C}_k} \boldsymbol{x}_i, \quad \hat{\boldsymbol{\Sigma}} = \frac{1}{N} \sum_k \sum_{i \in \mathcal{C}_k} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_k)^\top, \tag{8.3}$$

where N_k is the number of training points in class k and $N = N_1 + N_2$.

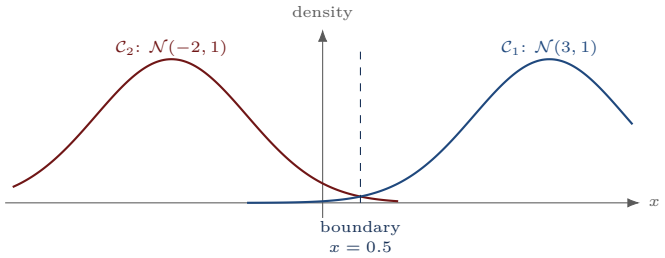


Figure 8.1. The two class-conditional Gaussians of Example 8.1, with shared variance $\sigma^2 = 1$ and means -2 and 3. With equal priors the decision boundary sits at the midpoint $x = 0.5$, where the densities cross; the posterior $p(\mathcal{C}_1 \mid x) = \sigma(5x - 2.5)$ passes through $\frac{1}{2}$ there.

Example 8.1 (GDA on the line). Take one-dimensional data with two points in each class: class $\mathcal{C}_1 = \{2, 4\}$ and class $\mathcal{C}_2 = \{-1, -3\}$, so $N_1 = N_2 = 2$ and $N = 4$. The maximum likelihood parameters from Eq. (8.3) are the class prior $\hat{\phi} = N_1/N = 2/4 = 0.5$; the class means $\hat{\mu}_1 = (2+4)/2 = 3$ and $\hat{\mu}_2 = (-1-3)/2 = -2$; and the pooled variance, summing squared deviations from each class mean, $\hat{\sigma}^2 = \frac{1}{4}\left[(2-3)^2+(4-3)^2+(-1+2)^2+(-3+2)^2\right] = \frac{1}{4}(1+1+1+1) = 1$. The log-odds Eq. (8.2) in one dimension, with $\sigma^2 = 1$, is

$$a(x) = \frac{\mu_1 - \mu_2}{\sigma^2}\, x - \frac{\mu_1^2 - \mu_2^2}{2\sigma^2} + \log\frac{\phi}{1-\phi} = (3 - (-2))\, x - \frac{9-4}{2} + \log 1 = 5x - 2.5,$$

so $p(\mathcal{C}_1 \mid x) = \sigma(5x - 2.5)$. The decision boundary is where $5x - 2.5 = 0$, that is $x = 0.5$, which is the midpoint between the class means 3 and -2, exactly where it should sit for equal priors and equal spread. A test point at $x = 1$ gets $a = 2.5$ and $p(\mathcal{C}_1 \mid 1) = \sigma(2.5) = 0.92$, confidently class 1. ◊

In two dimensions the same model gives the picture most people carry of GDA (Fig. 8.2): two elliptical Gaussian clouds and a straight boundary between them. The boundary is straight precisely because the classes *share* a covariance; letting each class keep its own covariance bends the boundary into a quadratic, the quadratic discriminant analysis (QDA) variant.

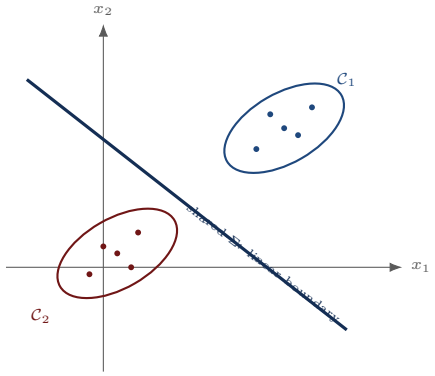


Figure 8.2. Two Gaussian classes with a shared covariance give a *linear* decision boundary, because the log-odds is linear in $\boldsymbol{x}$ (Eq. (8.2)). The weight vector of that boundary is $\Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, the same inverse-covariance reweighting as Fisher's discriminant in Chapter 7. Giving each class its own covariance yields a quadratic boundary (QDA).

GDA assumes a great deal, that each class is Gaussian, and it pays for it: it has $O(n^2)$ parameters (the covariance) against logistic regression's $O(n)$. The trade is the usual one. When the Gaussian assumption holds, GDA is more data-efficient; when it does not, the discriminative logistic regression of Section 8.5, which assumes only the sigmoid-of-a-line form and not the full Gaussian story, is more robust. A useful fact closes the comparison: GDA *implies* logistic regression (its posterior is sigmoid-of-a-line, Eq. (8.2)), but logistic regression does not imply GDA, so logistic regression is the weaker, safer assumption.

8.4 Naive Bayes

When the input is high-dimensional and discrete, modeling the full joint $p(\boldsymbol{x} \mid \mathcal{C}_k)$ is hopeless. Consider spam filtering with a vocabulary of n words, where $\boldsymbol{x} \in \{0,1\}^n$ records which words appear. The joint distribution over all 2^n possible emails needs $2^n - 1$ parameters per class, astronomically many. *Naive Bayes* cuts this down with one bold assumption: that the features are conditionally independent given the class,

$$p(\boldsymbol{x} \mid \mathcal{C}_k) = \prod_{j=1}^{n} p(x_j \mid \mathcal{C}_k). \tag{8.4}$$

The assumption is "naive" because words are obviously correlated, yet it works remarkably well, because the classifier needs only to rank the class posteriors correctly rather than estimate them exactly. It collapses the parameter count from 2^n to $2n + 1$: one word-probability per word per class, plus the prior. The classifier combines Eq. (8.4) with Bayes' theorem, choosing $\arg\max_k p(\mathcal{C}_k) \prod_j p(x_j \mid \mathcal{C}_k)$, and in practice the product is computed in the log domain, $\log p(\mathcal{C}_k) + \sum_j \log p(x_j \mid \mathcal{C}_k)$, to avoid the numerical underflow of multiplying thousands of small probabilities.

The word-probabilities are estimated by counting, but a raw count of zero is a trap: if a word never appeared in spam during training, its maximum likelihood probability is 0, and a single such word drives the entire product to zero, vetoing the class no matter what the other words say. The fix is *Laplace smoothing*, adding one imaginary count to each outcome (the Beta/Dirichlet pseudo-count of Chapter 2). For a binary word $x_j \in \{0,1\}$ the smoothed estimate is

$$\hat{p}(x_j = 1 \mid \mathcal{C}_k) = \frac{(\text{count of } x_j = 1 \text{ in class } k) + 1}{(\text{count of class } k) + 2}, \tag{8.5}$$

where the $+1$ in the numerator and $+2$ in the denominator are one pseudo-count for each of the two outcomes.

Example 8.2 (A naive Bayes spam filter, by hand). Train on five emails: three are spam ($\mathcal{C}_1$) and two are ham ($\mathcal{C}_2$), so the priors are $p(\mathcal{C}_1) = 3/5$ and $p(\mathcal{C}_2) = 2/5$. Track a single word, "free," which appeared in 2 of the 3 spam emails and in 0 of the 2 ham emails. The raw maximum likelihood estimate of $p(\text{free} \mid \text{ham})$ is $0/2 = 0$, which would forbid any email containing "free" from ever being ham, an overconfident verdict on two emails. Laplace smoothing Eq. (8.5) repairs it by adding one pseudo-count per outcome:

$$\hat{p}(\text{free} \mid \text{spam}) = \frac{2+1}{3+2} = \frac{3}{5}, \qquad \hat{p}(\text{free} \mid \text{ham}) = \frac{0+1}{2+2} = \frac{1}{4}.$$

Now classify a new email that contains the word "free." By Bayes' theorem the posterior is proportional to prior times likelihood:

$$p(\text{spam} \mid \text{free}) \propto p(\text{spam})\, \hat{p}(\text{free} \mid \text{spam}) = \tfrac{3}{5} \cdot \tfrac{3}{5} = \tfrac{9}{25} = 0.36,$$

$$p(\text{ham} \mid \text{free}) \propto p(\text{ham})\, \hat{p}(\text{free} \mid \text{ham}) = \tfrac{2}{5} \cdot \tfrac{1}{4} = \tfrac{1}{10} = 0.10.$$

Normalizing so the two add to one, $p(\text{spam} \mid \text{free}) = 0.36/(0.36 + 0.10) = 0.36/0.46 \approx 0.78$. The email is about 78% likely to be spam. Without smoothing the ham probability would have been exactly zero and the email would have been called spam with certainty, on the strength of two training emails; smoothing keeps the verdict confident but not absolute. ◊

For continuous features, *Gaussian naive Bayes* replaces the word-probability with a one-dimensional Gaussian $p(x_j \mid \mathcal{C}_k) = \mathcal{N}(x_j \mid \mu_{jk}, \sigma_{jk}^2)$ per feature per class, fit by the per-class mean and variance of that feature. The conditional-independence assumption then amounts to a diagonal covariance, a cheap special case of the GDA of Section 8.3.

8.5 Logistic regression

The discriminative route assumes the conclusion of Section 8.2 directly: the posterior is the sigmoid of a linear function of the features,

$$p(\mathcal{C}_1 \mid \boldsymbol{x}) = \sigma\big(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})\big), \qquad \sigma(a) = \frac{1}{1 + e^{-a}}, \tag{8.6}$$

and fits $\boldsymbol{w}$ to the labels without any model of $p(\boldsymbol{x})$. This is *logistic regression*, and despite the name it is a classifier. With labels $t_n \in \{0,1\}$ and shorthand $y_n = \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n))$ for the predicted probability, the likelihood of the data treats

each label as a Bernoulli draw with that probability, $p(t_n \mid \boldsymbol{x}_n) = y_n^{t_n}(1-y_n)^{1-t_n}$. The negative log-likelihood is the *cross-entropy error*

$$E(\boldsymbol{w}) = -\sum_{n=1}^{N}\big[t_n \log y_n + (1-t_n)\log(1-y_n)\big]. \tag{8.7}$$

This is the right loss for a binary target, the Bernoulli analogue of the squared error that Gaussian noise produced for regression in Chapter 5; using squared error here, as one might naively try, mismatches the noise model and behaves badly.

To minimize E we need its gradient, and it comes out in the same "error times input" form as linear regression, which is worth deriving because the cancellation is the reason the sigmoid was the right choice. The key identity is that the sigmoid's derivative is $\sigma'(a) = \sigma(a)(1-\sigma(a)) = y(1-y)$. Differentiating one term of Eq. (8.7) with respect to $\boldsymbol{w}$, through $y_n = \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n))$,

$$-\frac{\partial}{\partial \boldsymbol{w}}\big[t_n \log y_n + (1-t_n)\log(1-y_n)\big] = -\Big[\frac{t_n}{y_n} - \frac{1-t_n}{1-y_n}\Big] y_n(1-y_n)\,\boldsymbol{\phi}(\boldsymbol{x}_n) = (y_n - t_n)\,\boldsymbol{\phi}(\boldsymbol{x}_n),$$

where the $y_n(1-y_n)$ from the sigmoid's derivative cancels the denominators exactly. Summing over examples,

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n - t_n)\,\boldsymbol{\phi}(\boldsymbol{x}_n), \tag{8.8}$$

the prediction error $y_n - t_n$ times the input, identical in form to the linear regression gradient of Chapter 5. Unlike linear regression, though, there is no closed form: setting Eq. (8.8) to zero gives transcendental equations, so we minimize by gradient descent, $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla E(\boldsymbol{w})$, or by the faster Newton method of Section 8.6. We are safe doing so, because the error is convex.

Proposition 8.3 (The cross-entropy error is convex). The error $E(\boldsymbol{w})$ of Eq. (8.7) is a convex function of $\boldsymbol{w}$, with Hessian $H = \sum_n y_n(1-y_n)\,\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top \succeq 0$.

Proof. Differentiate the gradient Eq. (8.8) once more. Since $\partial y_n/\partial \boldsymbol{w} = y_n(1-y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)$, the Hessian is $H = \sum_n y_n(1-y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$. Each $y_n(1-y_n) > 0$ because $y_n \in (0,1)$, and each outer product $\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$ is positive semidefinite (for any $\boldsymbol{z}$, $\boldsymbol{z}^\top\boldsymbol{\phi}\boldsymbol{\phi}^\top\boldsymbol{z} = (\boldsymbol{\phi}^\top\boldsymbol{z})^2 \geq 0$), so H is a nonnegative combination of positive semidefinite matrices, hence positive semidefinite. A function with a positive semidefinite Hessian everywhere is convex. ∎

Convexity matters: it means gradient descent and Newton's method find the global optimum, with no bad local minima to trap them, which is exactly why logistic regression is reliable in practice.

Example 8.4 (A logistic-regression gradient step). Restate the setup fully. We have two training examples, $\boldsymbol{x}_1 = (1,2)$ with label $t_1 = 1$ and $\boldsymbol{x}_2 = (0,-1)$ with label $t_2 = 0$, and current weights $\boldsymbol{w} = (0.5, -0.25)$. The model predicts $y_n = \sigma(\boldsymbol{w}^\top\boldsymbol{x}_n)$ with $\sigma(a) = 1/(1+e^{-a})$, the loss is the cross-entropy $E(\boldsymbol{w}) = -\sum_n[t_n \log y_n + (1-t_n)\log(1-y_n)]$, and its gradient is $\nabla E = \sum_n (y_n - t_n)\boldsymbol{x}_n$.

Predictions. The first score is $\boldsymbol{w}^\top\boldsymbol{x}_1 = (0.5)(1) + (-0.25)(2) = 0.5 - 0.5 = 0$, so $y_1 = \sigma(0) = 0.5$. The second is $\boldsymbol{w}^\top\boldsymbol{x}_2 = (0.5)(0) + (-0.25)(-1) = 0.25$, so $y_2 = \sigma(0.25) = 1/(1+e^{-0.25}) = 1/(1+0.779) = 0.562$.

Gradient. The prediction errors are $y_1 - t_1 = 0.5 - 1 = -0.5$ and $y_2 - t_2 = 0.562 - 0 = 0.562$, so

$$\nabla E = (y_1 - t_1)\boldsymbol{x}_1 + (y_2 - t_2)\boldsymbol{x}_2 = (-0.5)(1,2) + (0.562)(0,-1) = (-0.5,-1.0) + (0,-0.562) = (-0.5,-1.562).$$

A gradient-descent step with learning rate $\eta = 0.1$ updates $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta\nabla E = (0.5,-0.25) - 0.1(-0.5,-1.562) = (0.55,-0.094)$, nudging the weights in the direction that raises y_1 toward its label 1 and lowers y_2 toward its label 0. ◊

The single-feature version of Eq. (8.6) is one artificial neuron: a weighted sum of inputs passed through a sigmoid. Training it by gradient descent on the cross-entropy is the simplest case of training a neural network, and Part IV is in large part the story of stacking many such units. Logistic regression is the gateway.

8.6 Iterative reweighted least squares

Gradient descent on the cross-entropy works but can be slow. Because the error is convex with a known Hessian (Proposition 8.3), we can do better with *Newton's method*, which uses curvature to take a smarter step, $\boldsymbol{w} \leftarrow \boldsymbol{w} - H^{-1}\nabla E$. Substituting the gradient $\nabla E = \boldsymbol{\Phi}^\top(\boldsymbol{y} - \boldsymbol{t})$ and the Hessian $H = \boldsymbol{\Phi}^\top R\boldsymbol{\Phi}$, where $\boldsymbol{\Phi}$ is the design matrix and $R = \operatorname{diag}(y_1(1-y_1), \dots, y_N(1-y_N))$ is a diagonal matrix of per-example weights, the Newton update rearranges into a strikingly familiar shape:

$$\boldsymbol{w}^{\text{new}} = (\boldsymbol{\Phi}^\top R\,\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top R\,\boldsymbol{z}, \qquad \boldsymbol{z} = \boldsymbol{\Phi}\boldsymbol{w}^{\text{old}} - R^{-1}(\boldsymbol{y} - \boldsymbol{t}). \tag{8.9}$$

This is exactly the weighted least-squares solution (the normal equations of Chapter 5 with weights R) for a synthetic target $\boldsymbol{z}$. Because the weights R and the target $\boldsymbol{z}$ depend on the current $\boldsymbol{y}$, hence on $\boldsymbol{w}$, they must be recomputed each iteration, which is why the method is called *iteratively reweighted least squares* (IRLS). Each step reweights the examples, emphasizing those near the decision boundary where $y_n(1-y_n)$ is largest (most uncertain), and solves a least-squares problem. Newton's method converges in a handful of iterations where gradient descent might take thousands, the payoff of using second-order curvature.

Example 8.5 (One IRLS step, by hand). Fit a one-parameter logistic model $y(x) = \sigma(wx)$ (no bias) to two points, $(x_1, t_1) = (1, 1)$ and $(x_2, t_2) = (-1, 0)$, starting from $w = 0$. Take one Newton/IRLS step.

Predictions. $y_1 = \sigma(0) = 0.5$, $y_2 = \sigma(0) = 0.5$. *Gradient.* $\nabla E = \sum_n (y_n - t_n)x_n = (0.5-1)(1) + (0.5-0)(-1) = -0.5-0.5 = -1.0$. *Hessian* (here a scalar), $H = \sum_n x_n^2\, y_n(1-y_n) = (1)^2(0.5)(0.5)+(-1)^2(0.5)(0.5) = 0.25+0.25 = 0.5$; the weights $R = \operatorname{diag}(0.25, 0.25)$ are largest because both points sit at the most uncertain output $y = 0.5$. *Newton step.* $w^{\text{new}} = w - H^{-1}\nabla E = 0 - (1/0.5)(-1.0) = 2.0$.

One step jumps from $w = 0$ to $w = 2$, where $y_1 = \sigma(2) = 0.881$ and $y_2 = \sigma(-2) = 0.119$, both now firmly on the correct side. Gradient descent with a small rate would have crept there over many steps; Newton's use of the curvature H took it in one. The same update written in matrix form, $w^{\text{new}} = (\boldsymbol{\Phi}^\top R\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top R\boldsymbol{z}$, is the weighted least-squares solution (8.9). ♢

8.7 Multiclass: softmax regression

For $K > 2$ classes, the sigmoid becomes the softmax of Chapter 3. Give each class its own weight vector $\boldsymbol{w}_k$ and define the score $a_k = \boldsymbol{w}_k^\top \boldsymbol{\phi}(\boldsymbol{x})$; the posterior is

$$p(\mathcal{C}_k \mid \boldsymbol{x}) = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}} = \operatorname{softmax}(\boldsymbol{a})_k. \tag{8.10}$$

A useful picture is the softmax as a firing squad of scores: each a_k fires, exponentiation amplifies the largest, and normalization turns the amplified scores into a probability distribution that always sums to one. The loss is again cross-entropy, $E = -\sum_n \sum_k t_{nk} \log p(\mathcal{C}_k \mid \boldsymbol{x}_n)$ with one-hot labels t_{nk}, and its gradient keeps the error-times-input form, $\nabla_{\boldsymbol{w}_k} E = \sum_n (p(\mathcal{C}_k \mid \boldsymbol{x}_n) - t_{nk})\boldsymbol{\phi}(\boldsymbol{x}_n)$: the predicted probability minus the indicator of the true class, times the input. This single expression is the output-layer gradient of every classification network in Part IV. The decision boundary between classes j and k is where their scores tie, $(\boldsymbol{w}_k - \boldsymbol{w}_j)^\top \boldsymbol{\phi}(\boldsymbol{x}) = 0$, a hyperplane, so softmax regression carves the input space into the convex regions of Chapter 7.

8.8 Evaluating a classifier

A single accuracy number hides what a classifier does, because the two kinds of mistake, a false alarm and a miss, usually have very different costs. Sorting predictions against truth gives the *confusion matrix*, with counts of true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). From these come the standard rates: the *true positive rate* (recall, or sensitivity) $\text{TPR} = \text{TP}/(\text{TP}+\text{FN})$, the fraction of actual positives caught; the *false positive rate* $\text{FPR} = \text{FP}/(\text{FP}+\text{TN})$, the fraction of actual negatives wrongly flagged; the *precision* $\text{TP}/(\text{TP}+\text{FP})$, the fraction of flags that are correct; and the *F1 score* $2\,(\text{precision})(\text{recall})/(\text{precision}+\text{recall})$, their harmonic mean.

A probabilistic classifier outputs $p(\mathcal{C}_1 \mid \boldsymbol{x})$, and turning it into a decision requires a threshold. Sliding the threshold from 0 to 1 traces a curve in the plane of TPR against FPR, the *receiver operating characteristic* (ROC). A perfect classifier reaches the top-left corner (all positives caught, no false alarms); random guessing lies on the diagonal; and

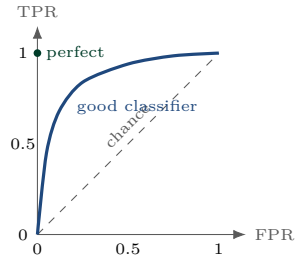


Figure 8.3. Sliding the decision threshold traces true positive rate against false positive rate. The diagonal is chance; the top-left corner is perfect; the area under the curve (AUC) scores all thresholds at once. Overlapping classes keep the curve off the corner, the Bayes-error limit of Chapter 1.

the *area under the curve* (AUC) summarizes performance across all thresholds in one number. Figure 8.3 shows the shape. The curve also makes concrete a limit from Chapter 1: when the class-conditional densities overlap, no threshold achieves both perfect TPR and zero FPR, and the best possible accuracy is the Bayes-optimal ceiling, an information limit that no classifier, however clever, can exceed.

Example 8.6 (Reading a confusion matrix). At one threshold a classifier on 100 test points produces TP = 40, FN = 10, FP = 5, TN = 45 (so 50 actual positives and 50 actual negatives). Then the true positive rate is $\text{TPR} = 40/(40+10) = 0.80$, the false positive rate is $\text{FPR} = 5/(5+45) = 0.10$, the precision is $40/(40+5) = 0.889$, and the recall equals the TPR, 0.80. The F1 score is the harmonic mean $2(0.889)(0.80)/(0.889+0.80) = 1.422/1.689 = 0.842$. Raising the threshold would lower both TPR and FPR (fewer flags, catching fewer positives but raising fewer false alarms), moving down-left along the ROC curve; lowering it moves up-right. The right operating point depends on whether a miss or a false alarm is costlier. ◊

Notes and References

The generative and discriminative routes, the sigmoid and softmax posteriors, logistic regression, and iterative reweighted least squares follow Sections 4.2 and 4.3 of Bishop [3]. Gaussian discriminant analysis and naive Bayes in the form given here are the staples of the CS229 lectures of Andrew Ng; the GDA-implies-logistic- regression relationship and the parameter-count comparison are developed there. Laplace smoothing is the Dirichlet pseudo-count of Chapter 2 applied to counts. The convexity of the cross-entropy error (Proposition 8.3) and the resulting positive semidefinite Hessian are what make logistic regression reliable, and the same Hessian is the reweighting matrix of IRLS. The ROC curve and the confusion-matrix metrics are standard tools of applied classification. Logistic regression as the single-neuron base case of a network is the bridge to Chapter 10; the Bayesian treatment of this same model, with a posterior over $\boldsymbol{w}$ and the Laplace approximation, is Chapter 9.

Exercises

Exercise 8.1. Show that for two classes with priors $p(\mathcal{C}_1), p(\mathcal{C}_2)$ and class-conditional densities $p(\boldsymbol{x} \mid \mathcal{C}_1), p(\boldsymbol{x} \mid \mathcal{C}_2)$, the posterior $p(\mathcal{C}_1 \mid \boldsymbol{x})$ equals $\sigma(a)$ with $a = \log[p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1)/(p(\boldsymbol{x} \mid \mathcal{C}_2)p(\mathcal{C}_2))]$.

Solution. By Bayes' theorem, $p(\mathcal{C}_1 \mid \boldsymbol{x}) = p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1)/[p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x} \mid \mathcal{C}_2)p(\mathcal{C}_2)]$. Divide numerator and denominator by the numerator: $p(\mathcal{C}_1 \mid \boldsymbol{x}) = 1/[1 + p(\boldsymbol{x} \mid \mathcal{C}_2)p(\mathcal{C}_2)/(p(\boldsymbol{x} \mid \mathcal{C}_1)p(\mathcal{C}_1))]$. The ratio in the denominator is e^{-a} with a as defined, since a is the log of its reciprocal, so $p(\mathcal{C}_1 \mid \boldsymbol{x}) = 1/(1 + e^{-a}) = \sigma(a)$. □

Exercise 8.2. One-dimensional data has class $\mathcal{C}_1 = \{1, 3\}$ and class $\mathcal{C}_2 = \{-2, -4\}$. Assuming Gaussian classes with equal priors and a shared variance, find the maximum likelihood parameters and the decision boundary.

Solution. There are $N_1 = N_2 = 2$ points, $N = 4$, so the prior is $\hat{\phi} = 2/4 = 0.5$. The class means are $\hat{\mu}_1 = (1+3)/2 = 2$ and $\hat{\mu}_2 = (-2-4)/2 = -3$. The pooled variance is $\hat{\sigma}^2 = \frac{1}{4}[(1-2)^2+(3-2)^2+(-2+3)^2+(-4+3)^2] = \frac{1}{4}(1+1+1+1) = 1$. With equal priors the log-odds is $a(x) = (\mu_1 - \mu_2)x/\sigma^2 - (\mu_1^2 - \mu_2^2)/(2\sigma^2) = 5x - (4 - 9)/2 = 5x + 2.5$. The boundary $a(x) = 0$ gives $x = -0.5$, the midpoint of the means 2 and -3. □

Exercise 8.3. A naive Bayes spam filter is trained on four spam and four ham emails. The word "win" appears in 3 of 4 spam and 1 of 4 ham. Using Laplace smoothing, find $\hat{p}(\text{win} \mid \text{spam})$ and $\hat{p}(\text{win} \mid \text{ham})$, then classify an email containing "win" (assume equal priors).

Solution. Laplace smoothing adds one pseudo-count per outcome, so $\hat{p}(\text{win} \mid \text{spam}) = (3 + 1)/(4 + 2) = 4/6 = 2/3$ and $\hat{p}(\text{win} \mid \text{ham}) = (1 + 1)/(4 + 2) = 2/6 = 1/3$. With equal priors $p(\text{spam}) = p(\text{ham}) = 1/2$, the unnormalized posteriors are $p(\text{spam} \mid \text{win}) \propto (1/2)(2/3) = 1/3$ and $p(\text{ham} \mid \text{win}) \propto (1/2)(1/3) = 1/6$. Normalizing, $p(\text{spam} \mid \text{win}) = (1/3)/(1/3 + 1/6) = (1/3)/(1/2) = 2/3 \approx 0.67$: about two-thirds likely spam. □

Exercise 8.4. For logistic regression with the cross-entropy loss, two examples $\boldsymbol{x}_1 = (1, 1)$ with $t_1 = 1$ and $\boldsymbol{x}_2 = (2, 0)$ with $t_2 = 0$, and current weights $\boldsymbol{w} = (0, 0)$, compute the predictions, the gradient $\nabla E = \sum_n (y_n - t_n)\boldsymbol{x}_n$, and one gradient-descent step with $\eta = 0.5$.

Solution. At $\boldsymbol{w} = (0, 0)$ both scores are zero, $\boldsymbol{w}^\top \boldsymbol{x}_1 = 0$ and $\boldsymbol{w}^\top \boldsymbol{x}_2 = 0$, so $y_1 = y_2 = \sigma(0) = 0.5$. The errors are $y_1 - t_1 = 0.5 - 1 = -0.5$ and $y_2 - t_2 = 0.5 - 0 = 0.5$. The gradient is $\nabla E = (-0.5)(1, 1) + (0.5)(2, 0) = (-0.5, -0.5) + (1.0, 0) = (0.5, -0.5)$. The step is $\boldsymbol{w} \leftarrow (0, 0) - 0.5(0.5, -0.5) = (-0.25, 0.25)$, which raises the score of $\boldsymbol{x}_1$ (label 1) and lowers that of $\boldsymbol{x}_2$ (label 0), as it should. □

Exercise 8.5. For the one-parameter logistic model $y(x) = \sigma(wx)$ with data $(x_1, t_1) = (2, 1)$ and $(x_2, t_2) = (-2, 0)$, starting at $w = 0$, compute one Newton/IRLS step. Report the gradient, the Hessian, and the updated weight.

Solution. At $w = 0$, $y_1 = y_2 = \sigma(0) = 0.5$. Gradient $\nabla E = \sum_n (y_n - t_n)x_n = (0.5-1)(2)+(0.5-0)(-2) = -1-1 = -2$. Hessian $H = \sum_n x_n^2 y_n(1 - y_n) = (4)(0.25) + (4)(0.25) = 2$. Newton step $w^{\text{new}} = 0 - H^{-1}\nabla E = 0 - \frac{1}{2}(-2) = 1$. After it, $y_1 = \sigma(2) = 0.881$ and $y_2 = \sigma(-2) = 0.119$, both correctly placed: one second-order step did what many gradient steps would. □

Exercise 8.6. Show that the logistic-regression Hessian $H = \sum_n y_n(1 - y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$ is positive semidefinite, and explain why this guarantees gradient descent finds the global optimum.

Solution. For any vector $\boldsymbol{z}$, $\boldsymbol{z}^\top H \boldsymbol{z} = \sum_n y_n(1-y_n)\boldsymbol{z}^\top \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{z} = \sum_n y_n(1-y_n)(\boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{z})^2$. Each $y_n(1-y_n) > 0$ since $y_n \in (0, 1)$, and each $(\boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{z})^2 \geq 0$, so the sum is nonnegative and $H \succeq 0$. A twice-differentiable function whose Hessian is positive semidefinite everywhere is convex, and a convex function has no local minima other than its global minimum, so gradient descent (with a suitable step size) converges to the global optimum. □

Exercise 8.7. A classifier produces TP = 80, FN = 20, FP = 30, TN = 70. Compute the true positive rate, false positive rate, precision, and F1 score.

Solution. There are $\text{TP} + \text{FN} = 100$ actual positives and $\text{FP} + \text{TN} = 100$ actual negatives. The true positive rate (recall) is $\text{TPR} = 80/(80 + 20) = 0.80$; the false positive rate is $\text{FPR} = 30/(30 + 70) = 0.30$; the precision is $80/(80 + 30) = 80/110 = 0.727$; and the F1 score is $2(0.727)(0.80)/(0.727 + 0.80) = 1.164/1.527 = 0.762$. The high false positive rate (0.30) shows the threshold is permissive, catching most positives but raising many false alarms. □

CHAPTER 9

Bayesian Logistic Regression and the Laplace Approximation

When you are not sure of the weights, you should not be sure of the prediction either. The honest classifier hedges toward the coin flip.

after Professor Peter Chin, ENGS 106

Chapter 6 gave Bayesian linear regression a posterior over its weights and a predictive distribution that widened where data was scarce. We would like the same for classification, and for the same reason: a logistic-regression point estimate reports a probability with false confidence, treating a weight vector pinned down by a million examples the same as one barely constrained by ten. The obstacle is that the posterior over the weights of a logistic-regression model is *not* Gaussian, and unlike the regression case there is no closed-form posterior to write down. The remedy is the first of the approximate-inference tools in this book, the *Laplace approximation*, which fits a Gaussian to any peaked distribution by matching its mode and its curvature. We develop the approximation in general, apply it to logistic regression, and find that the resulting predictive probability is *moderated* toward $\frac{1}{2}$ in proportion to how uncertain the weights are, the classification analogue of the widening band.

Roadmap

Suggested reading: Bishop [3], Sections 4.4–4.5; MacKay [26] for the evidence framework.

We cover: the Laplace approximation as a general tool (Section 9.1); the Gaussian approximation to the logistic-regression posterior (Section 9.2); the predictive distribution and its moderation by uncertainty, via the probit approximation (Section 9.3); and the bridge to Bayesian neural networks (Section 9.4).

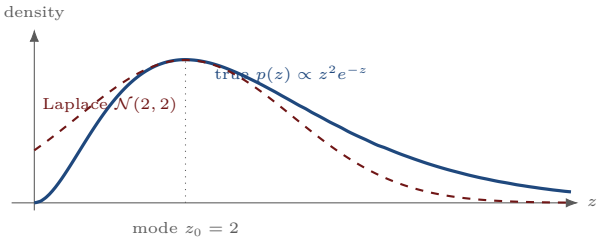


Figure 9.1. The true skewed density $p(z) \propto z^2 e^{-z}$ (solid) and its Laplace Gaussian $\mathcal{N}(2, 2)$ (dashed). The two agree at the mode $z_0 = 2$ and in local curvature, but the symmetric Gaussian misses the long right tail, underestimating the variance (2 versus the true 3).

9.1 The Laplace approximation

Many distributions we cannot integrate are nonetheless sharply peaked around a single mode, and near a peak almost anything looks like a Gaussian. The Laplace approximation makes this precise: it replaces a distribution by the Gaussian that has the same mode and the same curvature there.

Let $p(z) = \frac{1}{Z} f(z)$ be a distribution we know only up to its normalizer Z, with $f(z)$ the unnormalized density. Find the mode z_0, the point where f is largest, equivalently where $\frac{d}{dz} \ln f(z_0) = 0$. Expand $\ln f$ in a Taylor series about z_0. The first-order term vanishes at the mode, so to second order

$$\ln f(z) \approx \ln f(z_0) - \tfrac{1}{2} A (z - z_0)^2, \qquad A = - \left. \frac{d^2}{dz^2} \ln f(z) \right|_{z=z_0},$$

where $A > 0$ because z_0 is a maximum (the second derivative of $\ln f$ is negative, so its negative A is positive). Exponentiating, $f(z) \approx f(z_0) \exp(-\frac{1}{2} A (z - z_0)^2)$, which is an unnormalized Gaussian. Normalizing gives the Laplace approximation.

Laplace approximation

Approximate $p(z)$ by the Gaussian centered at the mode z_0 with variance the inverse curvature of $-\ln f$ there,

$$q(z) = \mathcal{N}\big(z \,\big|\, z_0,\ A^{-1}\big), \qquad A = - \left. \frac{d^2}{dz^2} \ln f(z) \right|_{z_0}. \tag{9.1}$$

In M dimensions z_0 is the mode and A is the Hessian matrix $A = -\nabla^2 \ln f(\boldsymbol{z})|_{\boldsymbol{z}_0}$, giving $q(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{z}_0, A^{-1})$.

The approximation matches the true distribution exactly at the mode and to second order around it; it is good when the distribution is unimodal and not too skewed, and it is the workhorse behind Bayesian treatment of models without conjugate priors.

Example 9.1 (Laplace approximation of a skewed density). Approximate the distribution $p(z) \propto z^2 e^{-z}$ on $z > 0$ (the kernel of a $\Gamma(3, 1)$ density). The unnormalized log-density is $\ln f(z) = 2 \ln z - z$. Its first derivative is $\frac{d}{dz} \ln f = \frac{2}{z} - 1$, which is zero at the mode $z_0 = 2$. Its second derivative is $\frac{d^2}{dz^2} \ln f = -\frac{2}{z^2}$, so the curvature at the mode is $A = -(-2/z_0^2) = 2/4 = 0.5$, and the Laplace variance is $A^{-1} = 2$. The Gaussian approximation is therefore $q(z) = \mathcal{N}(z \mid 2, 2)$. Comparing with the true distribution, whose mode is 2, mean 3, and variance 3: the Laplace approximation nails the mode and gives a variance of 2 against the true 3, underestimating the spread because the true density is skewed with a long right tail that a symmetric Gaussian cannot capture. The mode and local shape are right; the tail is not. ◊

9.2 The Gaussian approximation to the posterior

Now apply this to logistic regression. Put a Gaussian prior on the weights, $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{m}_0, \boldsymbol{S}_0)$, and recall the logistic likelihood from Chapter 8: with $y_n = \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n))$ and labels $t_n \in \{0, 1\}$, the data probability is $\prod_n y_n^{t_n}(1 - y_n)^{1-t_n}$. The log-posterior is, up to a constant,

$$\ln p(\boldsymbol{w} \mid \mathcal{D}) = -\tfrac{1}{2}(\boldsymbol{w} - \boldsymbol{m}_0)^\top \boldsymbol{S}_0^{-1}(\boldsymbol{w} - \boldsymbol{m}_0) + \sum_{n=1}^{N} \big[t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\big] + \text{const.} \tag{9.2}$$

Unlike the regression case of Chapter 6, this is not a quadratic in $\boldsymbol{w}$: the log-sigmoid terms are not quadratic, so the posterior is not Gaussian and has no closed form. We Laplace-approximate it. The mode $\boldsymbol{w}_{\text{MAP}}$ is the maximizer of Eq. (9.2), found by the same convex optimization (IRLS or gradient ascent) as in Chapter 8, now with the prior's contribution added. The covariance is the inverse of the Hessian of the negative log-posterior at the mode. Differentiating Eq. (9.2) twice, the prior contributes $\boldsymbol{S}_0^{-1}$ and the likelihood contributes the logistic Hessian $\sum_n y_n(1 - y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$ of Chapter 8, so

$$\boldsymbol{S}_N^{-1} = \boldsymbol{S}_0^{-1} + \sum_{n=1}^{N} y_n(1 - y_n)\, \boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top, \tag{9.3}$$

and the Laplace posterior is $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{w}_{\text{MAP}}, \boldsymbol{S}_N)$. Each term of the likelihood Hessian is weighted by $y_n(1-y_n)$, largest for points near the boundary (where $y_n \approx \frac{1}{2}$); these are the points that most sharpen the posterior, just as the support vectors will be in Chapter 15.

Example 9.2 (A Laplace posterior for logistic regression, by hand). Take a one-weight logistic model $p(\mathcal{C}_1 \mid x) = \sigma(wx)$ with a Gaussian prior $p(w) = \mathcal{N}(w \mid 0, \tau^2)$ of variance $\tau^2 = 1$, so prior precision $1/\tau^2 = 1$. The negative log-posterior is $E(w) = -\ln \sigma(wx) + \frac{1}{2\tau^2} w^2$ for a single positive training point. Suppose the design point is $x = 1$ with label $t = 1$, and that maximizing the posterior (one IRLS solve, Chapter 8) gives the mode $w_0 = 1$.

The Laplace approximation needs the curvature of E at w_0. Differentiating twice, the data term contributes $x^2\, y(1-y)$ and the prior contributes $1/\tau^2$:

$$A = \underbrace{x^2\, y(1-y)}_{\text{likelihood}} + \underbrace{\frac{1}{\tau^2}}_{\text{prior}}, \qquad y = \sigma(w_0 x) = \sigma(1) = 0.731.$$

So $A = (1)^2(0.731)(0.269) + 1 = 0.197 + 1 = 1.197$, and the posterior variance is the inverse curvature $S = A^{-1} = 0.836$. The Laplace approximation to the posterior is therefore

$$q(w) = \mathcal{N}\big(w \,\big|\, 1,\ 0.836\big).$$

The prior precision 1 and the data precision 0.197 add: more data, or a point further from the boundary (larger $y(1-y)$), would sharpen the posterior. This Gaussian over the weight is what the next section pushes through to a prediction. ◊

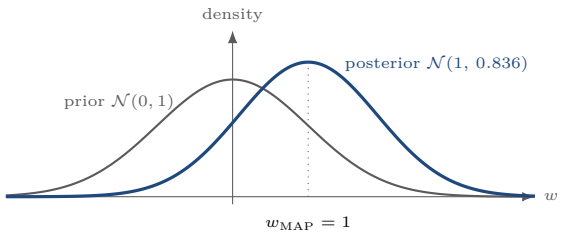


Figure 9.2. The Laplace posterior $\mathcal{N}(1, 0.836)$ of Example 9.2 (solid) against the prior $\mathcal{N}(0, 1)$ (gray). The single data point shifted the mode from 0 to 1 and sharpened the spread; more data would tighten it further. This Gaussian is averaged over to make the moderated prediction.

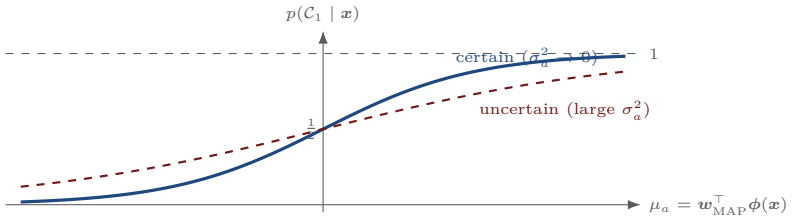


Figure 9.3. The point-estimate classifier uses the steep sigmoid $\sigma(\mu_a)$ (solid). The Bayesian predictive uses $\sigma(\kappa\mu_a)$ with $\kappa = (1+\pi\sigma_a^2/8)^{-1/2} < 1$ (dashed), flatter when the weights are uncertain, so confident predictions are pulled toward $\frac{1}{2}$. The boundary at $\mu_a = 0$ is unchanged.

9.3 The predictive distribution and its moderation

The point of all this is a better prediction. The Bayesian predictive probability for a new input $\boldsymbol{x}$ averages the logistic output over the posterior weights,

$$p(\mathcal{C}_1 \mid \boldsymbol{x}, \mathcal{D}) = \int \sigma\big(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})\big)\, q(\boldsymbol{w})\, d\boldsymbol{w}. \tag{9.4}$$

This integral, a sigmoid against a Gaussian, has no closed form, but two observations tame it. First, the integrand depends on $\boldsymbol{w}$ only through the scalar $a = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$, and a linear function of a Gaussian is Gaussian, so a is Gaussian with mean and variance

$$\mu_a = \boldsymbol{w}_{\mathrm{MAP}}^\top \boldsymbol{\phi}(\boldsymbol{x}), \qquad \sigma_a^2 = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{S}_N\, \boldsymbol{\phi}(\boldsymbol{x}). \tag{9.5}$$

The predictive probability collapses to a one-dimensional integral, $p(\mathcal{C}_1 \mid \boldsymbol{x}, \mathcal{D}) = \int \sigma(a)\, \mathcal{N}(a \mid \mu_a, \sigma_a^2)\, da$. Notice μ_a is exactly the score the point-estimate classifier would use, and $\sigma_a^2 = \boldsymbol{\phi}^\top \boldsymbol{S}_N \boldsymbol{\phi}$ is the predictive variance, the same quadratic form that gave the widening band in Chapter 6, now measuring uncertainty in the decision score.

Second, the one-dimensional integral has an accurate closed-form approximation. The logistic sigmoid is close to the probit (the Gaussian cumulative distribution function) Φ, and matching their slopes at the origin gives $\sigma(a) \approx \Phi(\lambda a)$ with $\lambda^2 = \pi/8$. Because a probit integrated against a Gaussian is again a probit, the integral evaluates to a sigmoid of a rescaled mean:

$$p(\mathcal{C}_1 \mid \boldsymbol{x}, \mathcal{D}) \approx \sigma\big(\kappa(\sigma_a^2)\, \mu_a\big), \qquad \kappa(\sigma_a^2) = \left(1 + \frac{\pi\sigma_a^2}{8}\right)^{-1/2}. \tag{9.6}$$

The factor $\kappa(\sigma_a^2) \le 1$ shrinks the score toward zero, and the more uncertain the decision (σ_a^2 large), the more it shrinks, pulling the probability toward $\frac{1}{2}$. This is the chapter's punchline: *uncertainty moderates the prediction.* A point-estimate classifier ignores σ_a^2 (it uses $\kappa = 1$) and so reports the same confident probability whether the weights are well determined or barely constrained; the Bayesian predictive hedges toward the coin flip exactly when it should. The decision boundary itself, where the probability is $\frac{1}{2}$, is unchanged, because $\kappa\mu_a = 0$ iff $\mu_a = 0$; only the confidence away from the boundary is tempered. Figure 9.3 shows the effect.

Example 9.3 (Moderation by hand). Suppose at a test point the posterior-mean score is $\mu_a = 2$ and the predictive variance of the score is $\sigma_a^2 = 4$. The point-estimate classifier ignores the variance and reports $\sigma(\mu_a) = \sigma(2) = 1/(1 + e^{-2}) = 0.881$, confidently class 1. The Bayesian predictive applies the moderation factor $\kappa(\sigma_a^2) = (1+\pi\sigma_a^2/8)^{-1/2} = (1+\pi\cdot 4/8)^{-1/2} = (1+1.571)^{-1/2} = (2.571)^{-1/2} = 0.624$, giving $\sigma(\kappa\mu_a) = \sigma(0.624\cdot 2) = \sigma(1.248) = 1/(1+e^{-1.248}) = 0.777$. The large uncertainty in the weights has pulled the probability from 0.881 down toward the coin flip, to 0.777: the model is still calling class 1, but with appropriately tempered confidence. Had the weights been well determined, say $\sigma_a^2 = 0.1$, then $\kappa = (1 + \pi\cdot 0.1/8)^{-1/2} = 0.981$ and the prediction $\sigma(1.96) = 0.876$ would be barely moved from the point estimate. ◊

The two halves now join. Example 9.2 gave the Laplace posterior $q(w) = \mathcal{N}(1, 0.836)$; the next example pushes it through to a moderated prediction, the whole Bayesian pipeline on one set of numbers.

Example 9.4 (From posterior to moderated prediction). Continue Example 9.2, where the one-weight logistic model had Laplace posterior $q(w) = \mathcal{N}(w \mid 1, 0.836)$, so $w_{\mathrm{MAP}} = 1$ and $S_N = 0.836$. Predict at a new input $x_* = 2$. The decision score $a = wx_*$ is Gaussian with

$$\mu_a = w_{\mathrm{MAP}}\, x_* = (1)(2) = 2, \qquad \sigma_a^2 = x_*^2\, S_N = (2)^2(0.836) = 3.34.$$

The plug-in classifier would report $\sigma(\mu_a) = \sigma(2) = 0.881$. The Bayesian predictive moderates it by $\kappa(\sigma_a^2) = (1 + \pi\sigma_a^2/8)^{-1/2} = (1 + \pi(3.34)/8)^{-1/2} = (1 + 1.31)^{-1/2} = 0.658$, giving

$$p(\mathcal{C}_1 \mid x_*, \mathcal{D}) \approx \sigma(\kappa\mu_a) = \sigma(0.658 \cdot 2) = \sigma(1.32) = 0.789.$$

The single training point left the weight uncertain ($\sigma_a^2 = 3.34$ is sizeable), so the prediction is pulled from 0.881 down to 0.789, honestly hedged. With more data the posterior would tighten ($S_N \to 0$), $\kappa \to 1$, and the Bayesian and plug-in predictions would agree. ◊

The single-weight example kept the algebra to one number, but the Hessian of Eq. (9.3) is a matrix, and the predictive variance of Eq. (9.5) is a quadratic form. Both deserve to be seen on real entries. We work a model with a bias and one feature, $\boldsymbol{\phi}(x) = (1, x)$, so the weight is $\boldsymbol{w} = (w_0, w_1)$ and the posterior is a genuine two-dimensional Gaussian whose covariance has off-diagonal correlation between the bias and the slope.

Example 9.5 (A two-parameter Laplace posterior, by hand). Fit $p(\mathcal{C}_1 \mid x) = \sigma(w_0 + w_1 x)$ with the isotropic Gaussian prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \mathbf{0}, \alpha^{-1} I)$ at precision $\alpha = 1$, so $\boldsymbol{S}_0^{-1} = I$. Two training points carry features $\boldsymbol{\phi}_1 = (1, 0)$ (label $t_1 = 0$) and $\boldsymbol{\phi}_2 = (1, 2)$ (label $t_2 = 1$). Suppose the convex MAP solve of Chapter 8, now with the prior added, returns the mode $\boldsymbol{w}_{\text{MAP}} = (0, 0.5)$.

The Laplace posterior is $q(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{w}_{\text{MAP}}, \boldsymbol{S}_N)$ with $\boldsymbol{S}_N^{-1}$ the Hessian of Eq. (9.3). Build it piece by piece. Each linear score at the mode is $a_n = \boldsymbol{w}_{\text{MAP}}^\top \boldsymbol{\phi}_n$, so $a_1 = 0$ and $a_2 = 0 \cdot 1 + 0.5 \cdot 2 = 1$, giving fitted probabilities

$$\begin{aligned} y_1 &= \sigma(0) = 0.5, & y_1(1 - y_1) &= 0.25, \\ y_2 &= \sigma(1) = 0.731, & y_2(1 - y_2) &= 0.197. \end{aligned}$$

The two outer products are

$$\boldsymbol{\phi}_1 \boldsymbol{\phi}_1^\top = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \boldsymbol{\phi}_2 \boldsymbol{\phi}_2^\top = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix},$$

each weighted by its $y_n(1 - y_n)$. The weighted data Hessian is therefore

$$0.25 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + 0.197 \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix} = \begin{bmatrix} 0.447 & 0.394 \\ 0.394 & 0.788 \end{bmatrix}.$$

Adding the prior precision $\alpha I = I$ gives the full Hessian, which is the posterior precision:

$$\boldsymbol{S}_N^{-1} = I + \begin{bmatrix} 0.447 & 0.394 \\ 0.394 & 0.788 \end{bmatrix} = \begin{bmatrix} 1.447 & 0.394 \\ 0.394 & 1.788 \end{bmatrix}.$$

Invert the 2×2 matrix. Its determinant is $1.447 \cdot 1.788 - 0.394^2 = 2.587 - 0.155 = 2.432$, so

$$\boldsymbol{S}_N = \frac{1}{2.432} \begin{bmatrix} 1.788 & -0.394 \\ -0.394 & 1.447 \end{bmatrix} = \begin{bmatrix} 0.735 & -0.162 \\ -0.162 & 0.595 \end{bmatrix},$$

and the Laplace posterior is $q(\boldsymbol{w}) = \mathcal{N}\big(\boldsymbol{w} \mid (0, 0.5),\ \boldsymbol{S}_N\big)$. The negative off-diagonal -0.162 says the bias and the slope are anticorrelated under the posterior: a fit that raises w_0 can compensate by lowering w_1 and still explain the two points, so the data pin down a ridge in weight space rather than a single point. The point near the boundary, $\boldsymbol{\phi}_1$ with $y_1 = 0.5$, contributes the most curvature ($y(1 - y) = 0.25$ is its maximum), exactly as Eq. (9.3) predicts. ◊

This two-dimensional posterior now feeds a prediction, and the predictive variance is the quadratic form $\sigma_a^2 = \boldsymbol{\phi}(x_*)^\top \boldsymbol{S}_N \boldsymbol{\phi}(x_*)$ of Eq. (9.5) evaluated on the matrix we just built.

Example 9.6 (Predictive variance and moderation in two dimensions). Continue Example 9.5, with $\boldsymbol{w}_{\text{MAP}} = (0, 0.5)$ and

$$\boldsymbol{S}_N = \begin{bmatrix} 0.735 & -0.162 \\ -0.162 & 0.595 \end{bmatrix}.$$

Predict at the test input $x_* = 3$, whose feature vector is $\boldsymbol{\phi}(x_*) = (1, 3)$. The posterior-mean score is the plug-in score,

$$\mu_a = \boldsymbol{w}_{\text{MAP}}^\top \boldsymbol{\phi}(x_*) = 0 \cdot 1 + 0.5 \cdot 3 = 1.5.$$

For the variance, first apply the covariance to the feature vector,

$$\boldsymbol{S}_N\boldsymbol{\phi}(x_*) = \begin{bmatrix} 0.735 & -0.162 \\ -0.162 & 0.595 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.735 - 0.486 \\ -0.162 + 1.785 \end{bmatrix} = \begin{bmatrix} 0.249 \\ 1.623 \end{bmatrix},$$

then close the quadratic form by the second multiplication,

$$\sigma_a^2 = \boldsymbol{\phi}(x_*)^\top \big(\boldsymbol{S}_N\boldsymbol{\phi}(x_*)\big) = 1 \cdot 0.249 + 3 \cdot 1.623 = 0.249 + 4.869 = 5.118.$$

The decision score at $x_* = 3$ is therefore Gaussian, $a \sim \mathcal{N}(1.5,\ 5.118)$: a sizeable variance, because $x_* = 3$ sits well outside the cluster of training inputs at 0 and 2, where the slope w_1 is poorly constrained and its uncertainty is amplified by the large feature value. Feed this into the moderation factor of Eq. (9.6),

$$\kappa(\sigma_a^2) = \left(1 + \frac{\pi\sigma_a^2}{8}\right)^{-1/2} = \left(1 + \frac{\pi \cdot 5.118}{8}\right)^{-1/2} = (1 + 2.010)^{-1/2} = (3.010)^{-1/2} = 0.576.$$

The plug-in classifier would report $\sigma(\mu_a) = \sigma(1.5) = 0.818$. The Bayesian predictive moderates it,

$$p(\mathcal{C}_1 \mid x_*, \mathcal{D}) \approx \sigma(\kappa\mu_a) = \sigma(0.576 \cdot 1.5) = \sigma(0.864) = 0.704.$$

Extrapolating to $x_* = 3$ inflates the predictive variance, the moderation factor drops to 0.576, and a confident 0.818 is pulled most of the way back toward the coin flip, to 0.704. The off-diagonal of $\boldsymbol{S}_N$ mattered: had we dropped the -0.162 correlation and kept only the diagonal, the quadratic form would have read $0.735 + 9 \cdot 0.595 = 6.09$, overstating the variance and over-moderating the prediction. Bias-slope correlation is part of the honest answer. ◊

How good is the Laplace normalizer?

The Laplace approximation does more than supply a Gaussian posterior; in M dimensions it also estimates the normalizing constant $Z = \int f(\boldsymbol{z})\, d\boldsymbol{z}$, since the Gaussian it fits integrates to $f(\boldsymbol{z}_0)$ times the Gaussian normalizer. In one dimension,

$$Z = \int f(z)\, dz \approx f(z_0)\sqrt{\frac{2\pi}{A}}, \qquad A = -\frac{d^2}{dz^2}\ln f(z)\Big|_{z_0}. \tag{9.7}$$

This is the quantity that becomes the model evidence when f is the unnormalized posterior, so it is worth checking against a case we can integrate exactly.

Example 9.7 (Laplace normalizer against an exact integral). Return to the density of Example 9.1, $f(z) = z^2 e^{-z}$ on $z > 0$, whose exact integral is a standard one,

$$Z = \int_0^\infty z^2 e^{-z}\, dz = \Gamma(3) = 2! = 2.$$

The Laplace approximation reuses the mode and curvature already found there, $z_0 = 2$ and $A = 0.5$. The peak value is $f(z_0) = z_0^2 e^{-z_0} = 4e^{-2} = 4 \cdot 0.135 = 0.541$, and the Gaussian width factor is $\sqrt{2\pi/A} = \sqrt{2\pi/0.5} = \sqrt{4\pi} = 3.545$. Multiplying through Eq. (9.7),

$$Z \approx f(z_0)\sqrt{\frac{2\pi}{A}} = 0.541 \cdot 3.545 = 1.919.$$

Against the exact $Z = 2$, the estimate 1.919 is low by about 4%. The miss has the same cause as the variance miss in Example 9.1: the density is right-skewed, so the symmetric Gaussian, by ignoring the long right tail, leaves out a sliver of mass. The accuracy is still good for a one-line approximation, and it improves as the peak sharpens. For a posterior of N near-Gaussian data terms the relative error of Eq. (9.7) shrinks like $1/N$, which is why the Laplace evidence is trustworthy in the data-rich regime where models are usually compared. ◊

9.4 Toward Bayesian neural networks

The Laplace approximation is not special to logistic regression; it is a general recipe for any model whose log-posterior is peaked, and that includes neural networks. A Bayesian neural network places a prior over the network weights, and its posterior, hopelessly non-Gaussian and high-dimensional, can be Laplace-approximated around a mode found by ordinary training: the mode is the trained weights, and the curvature is the Hessian of the loss. The same machinery then gives predictive uncertainty for the network and, through the evidence (the marginal likelihood of Chapter 6), a way to set hyperparameters such as the weight-decay strength and to compare architectures. This is the framework of MacKay [26], and it is the principled answer to a question Part IV will otherwise answer with heuristics: how confident should a deep network be, and how complex should it be allowed to grow.

The evidence carries an automatic Occam's razor worth naming. The marginal likelihood of a model is the prior-weighted average of the likelihood, $p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{w})\, p(\boldsymbol{w})\, d\boldsymbol{w}$, and the Laplace approximation factors it into the best-fit likelihood times an *Occam factor*, the ratio of posterior to prior width. A more complex model spreads its prior over more hypotheses, so each gets less mass; it is rewarded only if the extra flexibility buys enough fit to overcome that penalty. Complexity is thus paid for automatically, with no validation set, which is why the evidence can rank architectures and tune the weight-decay strength directly. The exact and approximate inference needed to scale these ideas beyond the Laplace approximation, variational methods and sampling, are previewed in Chapter 17 and Chapter 20.

Notes and References

The Laplace approximation, the Gaussian approximation to the logistic-regression posterior, and the probit-moderated predictive distribution follow Sections 4.4 and 4.5 of Bishop [3]. The probit approximation $\sigma(a) \approx \Phi(\lambda a)$ with $\lambda^2 = \pi/8$ matches the slopes of the logistic sigmoid and the Gaussian CDF at the origin, and the moderation factor $\kappa(\sigma^2) = (1 + \pi\sigma^2/8)^{-1/2}$ follows from integrating a probit against a Gaussian. The evidence framework and its application to neural networks are due to MacKay [26]. The Laplace approximation is the simplest of a family of approximate-inference methods; the variational and sampling alternatives appear in Chapters 17 to 20. With this chapter Part III is complete: we have the geometry of Chapter 7, the generative and discriminative probabilistic models of Chapter 8, and now their Bayesian treatment.

Exercises

Exercise 9.1. Find the Laplace approximation to the distribution $p(z) \propto z^4 e^{-2z}$ on $z > 0$. Give the mode, the curvature, and the approximating Gaussian.

Solution. The unnormalized log-density is $\ln f(z) = 4 \ln z - 2z$. Its first derivative is $4/z - 2$, zero at the mode $z_0 = 2$. Its second derivative is $-4/z^2$, so the curvature is $A = -(-4/z_0^2) = 4/4 = 1$, and the Laplace variance is $A^{-1} = 1$. The approximation is $q(z) = \mathcal{N}(z \mid 2, 1)$. (The true distribution is the kernel of a $\Gamma(5, 2)$, with mode 2, mean 2.5, variance 1.25; Laplace matches the mode and gives variance 1, again a slight underestimate from the right skew.) □

Exercise 9.2. Write the Laplace posterior covariance for Bayesian logistic regression with prior $\mathcal{N}(\mathbf{0}, \alpha^{-1} I)$, and explain which data points contribute most to sharpening the posterior.

Solution. With prior precision $\boldsymbol{S}_0^{-1} = \alpha I$, the Laplace posterior covariance is the inverse of $\boldsymbol{S}_N^{-1} = \alpha I + \sum_n y_n(1 - y_n)\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$. Each data point contributes a term weighted by $y_n(1 - y_n)$, which is largest when $y_n = \frac{1}{2}$, that is for points lying on or near the decision boundary; points the model already classifies confidently (y_n near 0 or 1) contribute almost nothing. So boundary points sharpen the posterior most, the same points that will be the support vectors of Chapter 15. □

Exercise 9.3. At a test point the posterior-mean score is $\mu_a = 3$ and the predictive variance is $\sigma_a^2 = 8$. Compare the point-estimate probability $\sigma(\mu_a)$ with the Bayesian predictive $\sigma(\kappa \mu_a)$, where $\kappa = (1 + \pi \sigma_a^2/8)^{-1/2}$.

Solution. The point estimate is $\sigma(3) = 1/(1 + e^{-3}) = 1/(1 + 0.0498) = 0.953$. The moderation factor is $\kappa = (1 + \pi \cdot 8/8)^{-1/2} = (1 + \pi)^{-1/2} = (4.142)^{-1/2} = 0.491$, so the Bayesian predictive is $\sigma(0.491 \cdot 3) = \sigma(1.474) = 1/(1 + e^{-1.474}) = 1/(1 + 0.229) = 0.814$. The large weight uncertainty pulls the confident 0.953 down to 0.814, a substantial hedge toward $\frac{1}{2}$. □

Exercise 9.4. A one-weight logistic model $\sigma(wx)$ has a Gaussian prior $\mathcal{N}(w \mid 0, 2)$ (prior precision 1/2). Suppose the MAP weight is $w_0 = 0.5$ and the single design point is $x = 2$. Compute the Laplace posterior precision A and variance S_N, and write the approximate posterior.

Solution. At the mode, $y = \sigma(w_0 x) = \sigma(1) = 0.731$, so $y(1 - y) = 0.197$. The posterior precision adds the data curvature and the prior precision: $A = x^2 y(1 - y) + 1/\tau^2 = (4)(0.197) + 0.5 = 0.787 + 0.5 = 1.287$. The variance is $S_N = A^{-1} = 0.777$, so $q(w) = \mathcal{N}(w \mid 0.5, 0.777)$. The data point, being away from the boundary, adds 0.787 of precision on top of the prior's 0.5. □

Exercise 9.5. A Bayesian logistic classifier gives, at a test point, a score with mean $\mu_a = 1.5$ and variance $\sigma_a^2 = 2$. Compute the plug-in probability $\sigma(\mu_a)$ and the moderated Bayesian probability $\sigma(\kappa \mu_a)$, and explain the difference.

Solution. The plug-in classifier reports $\sigma(1.5) = 1/(1 + e^{-1.5}) = 0.818$. The moderation factor is $\kappa = (1 + \pi \sigma_a^2/8)^{-1/2} = (1 + \pi(2)/8)^{-1/2} = (1 + 0.785)^{-1/2} = 0.748$, so the Bayesian probability is $\sigma(0.748 \cdot 1.5) = \sigma(1.122) = 0.754$. The weight uncertainty ($\sigma_a^2 = 2$) pulls the confident 0.818 toward the coin flip, to 0.754; the boundary at $\mu_a = 0$ is unmoved, only the confidence away from it is tempered. □

Exercise 9.6. A two-weight logistic model $\sigma(w_0 + w_1 x)$ has the isotropic Gaussian prior $\mathcal{N}(\boldsymbol{w} \mid \mathbf{0}, \alpha^{-1} I)$ with $\alpha = 2$. Two training points have features $\boldsymbol{\phi}_1 = (1, 0)$ and $\boldsymbol{\phi}_2 = (1, 1)$, and the MAP weight is $\boldsymbol{w}_{\text{MAP}} = (0, 0)$. Build the posterior precision $\boldsymbol{S}_N^{-1}$, invert it to get $\boldsymbol{S}_N$, and write the Laplace posterior $q(\boldsymbol{w})$.

Solution. At the mode every score is $a_n = \boldsymbol{w}_{\text{MAP}}^\top \boldsymbol{\phi}_n = 0$, so $y_n = \sigma(0) = 0.5$ and $y_n(1 - y_n) = 0.25$ for both points. The outer products are $\boldsymbol{\phi}_1 \boldsymbol{\phi}_1^\top = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right]$ and $\boldsymbol{\phi}_2 \boldsymbol{\phi}_2^\top = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$, each weighted by 0.25. The weighted data Hessian is $0.25 \left[\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0.5 & 0.25 \\ 0.25 & 0.25 \end{smallmatrix}\right]$. Adding the prior precision $\alpha I = 2I$ gives

$$\boldsymbol{S}_N^{-1} = 2I + \begin{bmatrix} 0.5 & 0.25 \\ 0.25 & 0.25 \end{bmatrix} = \begin{bmatrix} 2.5 & 0.25 \\ 0.25 & 2.25 \end{bmatrix}.$$

Its determinant is $2.5 \cdot 2.25 - 0.25^2 = 5.625 - 0.0625 = 5.5625$, so $\boldsymbol{S}_N = \frac{1}{5.5625}\left[\begin{smallmatrix} 2.25 & -0.25 \\ -0.25 & 2.5 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0.404 & -0.045 \\ -0.045 & 0.449 \end{smallmatrix}\right]$, and $q(\boldsymbol{w}) = \mathcal{N}\big(\boldsymbol{w} \mid (0,0),\ \boldsymbol{S}_N\big)$. The strong prior ($\alpha = 2$) keeps the posterior tight, and the small off-diagonal -0.045 reflects a weak anticorrelation between the bias and the slope. □

Exercise 9.7. A Bayesian logistic classifier has posterior mean $\boldsymbol{w}_{\text{MAP}} = (0.5, 1)$ and posterior covariance $\boldsymbol{S}_N = \left[\begin{smallmatrix} 0.5 & 0.1 \\ 0.1 & 0.3 \end{smallmatrix}\right]$. At the test point with feature vector $\boldsymbol{\phi}(x_*) = (1, 2)$, compute the score mean μ_a, the predictive variance $\sigma_a^2 = \boldsymbol{\phi}(x_*)^\top \boldsymbol{S}_N \boldsymbol{\phi}(x_*)$, the moderation factor κ, and compare the plug-in and moderated probabilities.

Solution. The score mean is $\mu_a = \boldsymbol{w}_{\text{MAP}}^\top \boldsymbol{\phi}(x_*) = 0.5 \cdot 1 + 1 \cdot 2 = 2.5$. For the variance, first $\boldsymbol{S}_N \boldsymbol{\phi}(x_*) = \left[\begin{smallmatrix} 0.5 & 0.1 \\ 0.1 & 0.3 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0.5+0.2 \\ 0.1+0.6 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0.7 \\ 0.7 \end{smallmatrix}\right]$, then $\sigma_a^2 = \boldsymbol{\phi}(x_*)^\top \left[\begin{smallmatrix} 0.7 \\ 0.7 \end{smallmatrix}\right] = 1 \cdot 0.7 + 2 \cdot 0.7 = 2.1$. The moderation factor is $\kappa = (1 + \pi\sigma_a^2/8)^{-1/2} = (1 + \pi \cdot 2.1/8)^{-1/2} = (1 + 0.825)^{-1/2} = (1.825)^{-1/2} = 0.740$. The plug-in probability is $\sigma(2.5) = 1/(1 + e^{-2.5}) = 0.924$, while the moderated probability is $\sigma(\kappa\mu_a) = \sigma(0.740 \cdot 2.5) = \sigma(1.851) = 0.864$. The predictive variance of 2.1 pulls a confident 0.924 down to 0.864, a moderate hedge; the boundary at $\mu_a = 0$ would be unmoved. □

Exercise 9.8. Explain why the posterior over the weights in logistic regression is not Gaussian, whereas in Bayesian linear regression (Chapter 6) it is exactly Gaussian, and what this forces us to do.

Solution. In Bayesian linear regression the log-likelihood is a quadratic in $\boldsymbol{w}$ (squared error from the Gaussian noise model), so adding the quadratic log-prior keeps the log-posterior quadratic, and the exponential of a quadratic is exactly Gaussian. In logistic regression the log-likelihood contains $\ln \sigma(\boldsymbol{w}^\top \boldsymbol{\phi})$ and $\ln(1 - \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}))$ terms, which are not quadratic in $\boldsymbol{w}$, so the log-posterior is not quadratic and the posterior is not Gaussian and has no closed form. This forces an approximation; the Laplace approximation fits a Gaussian at the posterior mode using the local curvature, which is the simplest accurate choice. □

PART IV

Neural Networks and Deep Learning

CHAPTER 10

Feed-Forward Neural Networks

A linear model learns the weights on features you chose. A neural network learns the features too.

after Professor Peter Chin, ENGS 106

Every model so far has been linear in a *fixed* set of features: pick basis functions $\boldsymbol{\phi}(\boldsymbol{x})$ by hand, then learn the weights $\boldsymbol{w}$. The power and the limit both come from that choice of features. Choose them well and a linear model is unbeatable; choose them poorly and no amount of weight-fitting can recover, as the perceptron's failure on XOR showed in Chapter 7. A *neural network* removes the bottleneck by learning the features themselves. It builds them in layers, each layer a bank of simple units that transform the previous layer's output, and the features in the final layer, the ones a linear classifier then acts on, are discovered from data rather than dictated. This chapter is about what such networks can represent: how they are wired, why a single hidden layer already solves XOR and in fact approximates any continuous function, and how the output layer is just the generalized linear model of Chapter 3 sitting on top of learned features. How to train them, the backpropagation algorithm, is Chapter 11.

Roadmap

Suggested reading: Bishop [3], Sections 5.1–5.2; Goodfellow et al. [13], Chapter 6.

We cover: the move from fixed to learned features and the artificial neuron (Section 10.1); feed-forward architecture, the matrix form, and counting parameters (Section 10.2); activation functions and the saturation that motivates the ReLU (Section 10.3); a forward pass worked entirely by hand (Section 10.4); how a hidden layer solves XOR by reshaping the feature space (Section 10.5); the universal approximation theorem and why depth helps (Section 10.6); and output layers and loss functions as exponential-family models (Section 10.7).

10.1 From fixed features to learned features

A linear model computes $y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$, a weighted sum of fixed features. Logistic regression added a nonlinearity at the end, $y(\boldsymbol{x}) = \sigma(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}))$, and we noted in Chapter 8 that this is one artificial *neuron*: a weighted sum of inputs passed through an activation function. A neuron with inputs $\boldsymbol{x}$, weights $\boldsymbol{w}$, bias b, and activation g computes

$$z = g(\boldsymbol{w}^\top \boldsymbol{x} + b) = g\Big(\textstyle\sum_i w_i x_i + b\Big), \tag{10.1}$$

which is logistic regression when g is the sigmoid. The weighted sum $\boldsymbol{w}^\top \boldsymbol{x} + b$ is the *pre-activation*; the bias b shifts the threshold at which the unit turns on, exactly the role of the intercept w_0 with a constant input $x_0 = 1$. The artificial neuron dates to McCulloch and Pitts [29], the perceptron learning rule to Rosenblatt [32], and the very name "artificial intelligence" to the 1956 summer workshop held here at Dartmouth [28]; the brain that inspired the metaphor has on the order of 10^{11} neurons, each wired to thousands of others.

One neuron, by the numbers. Let $\boldsymbol{x} = (2, -1, 3)^\top$, weights $\boldsymbol{w} = (0.5, 1.0, -0.5)^\top$, bias $b = 0.4$, and $g = \sigma$. The pre-activation is

$$a = \boldsymbol{w}^\top \boldsymbol{x} + b = (0.5)(2) + (1.0)(-1) + (-0.5)(3) + 0.4 = 1.0 - 1.0 - 1.5 + 0.4 = -1.1,$$

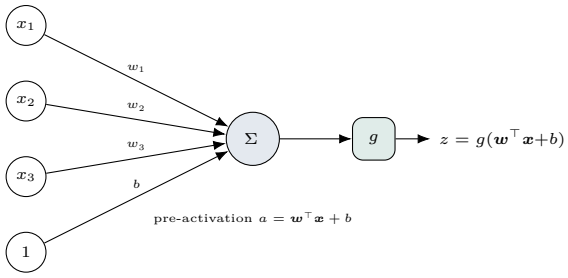


Figure 10.1. One neuron: it forms the weighted sum (pre-activation) $a = \boldsymbol{w}^\top \boldsymbol{x} + b$, with the bias carried as a weight on a constant input 1, then passes it through the activation g. Logistic regression is the single neuron with $g = \sigma$.

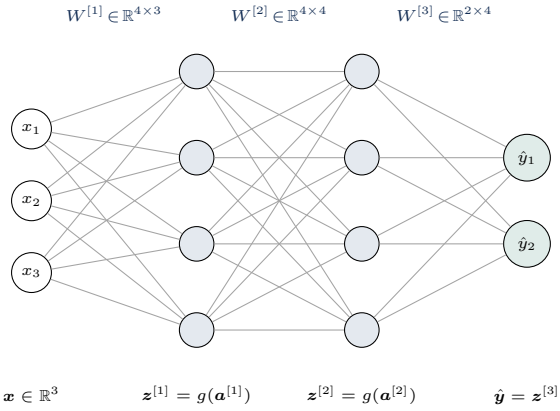


Figure 10.2. A $3-4-4-2$ network. Layer ℓ maps the previous layer's activations through $\boldsymbol{a}^{[\ell]} = W^{[\ell]}\boldsymbol{z}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}$ then $\boldsymbol{z}^{[\ell]} = g(\boldsymbol{a}^{[\ell]})$. The weight matrix $W^{[\ell]}$ has one row per unit in layer ℓ and one column per unit in layer $\ell-1$, so its shape is (units in ℓ) $\times$ (units in $\ell-1$).

so the neuron outputs $z = \sigma(-1.1) = 1/(1+e^{1.1}) = 1/(1+3.004) = 0.250$. A second neuron reading the same inputs with different weights would compute a different feature of $\boldsymbol{x}$; a *layer* is a bank of such neurons, and a *network* feeds one layer's outputs into the next.

The idea of a neural network is to use the outputs of one bank of neurons as the inputs to the next. The first layer's neurons compute features of the raw input; the second layer computes features of those features; and so on. Crucially, the weights of *every* layer are learned, so the features are not chosen in advance but shaped by the data to make the final task easy. A linear model is the special case of no hidden layers; everything new comes from stacking.

10.2 Feed-forward architecture

A feed-forward network of L layers computes its output by a chain of these transformations. Writing $\boldsymbol{z}^{[0]} = \boldsymbol{x}$ for the input, each layer ℓ forms a *pre-activation* (an affine map) and then applies the activation elementwise:

$$\boldsymbol{a}^{[\ell]} = W^{[\ell]}\boldsymbol{z}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}, \qquad \boldsymbol{z}^{[\ell]} = g(\boldsymbol{a}^{[\ell]}), \qquad \ell = 1, \dots, L, \tag{10.2}$$

where $W^{[\ell]}$ is the weight matrix of layer ℓ (its rows are the weight vectors of that layer's neurons), $\boldsymbol{b}^{[\ell]}$ the bias vector, and g the activation. The final layer's output $\boldsymbol{z}^{[L]}$ is the network's prediction, and the intermediate $\boldsymbol{z}^{[\ell]}$ are the learned features. This forward sweep, input to output, is the *forward pass*.

Figure 10.2 is the anatomy of a four-layer network with the dimensions marked. Reading it left to right: three inputs feed a hidden layer of four units, which feeds a second hidden layer of four units, which feeds two output units. Every arrow is one learned weight; every unit also has a learned bias.

Shapes and parameter counts. The single most useful habit when reading or writing network code is to track the shape of every array. If layer ℓ has d_ℓ units and layer $\ell-1$ has $d_{\ell-1}$, then

$$W^{[\ell]} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \qquad \boldsymbol{b}^{[\ell]} \in \mathbb{R}^{d_\ell}, \qquad \boldsymbol{a}^{[\ell]}, \boldsymbol{z}^{[\ell]} \in \mathbb{R}^{d_\ell}.$$

Layer ℓ therefore holds $d_\ell d_{\ell-1}$ weights and d_ℓ biases. Table 10.1 counts them for the network of Fig. 10.2: 46 parameters in all. The same arithmetic on a small image classifier with layer sizes 784–100–100–10 gives $100{\cdot}784 + 100{\cdot}100 + 10{\cdot}100 = 89{,}400$ weights and $100 + 100 + 10 = 210$ biases, about 89,610 parameters, all learned from data.

layer ℓ	shape of $W^{[\ell]}$	weights $d_\ell d_{\ell-1}$	biases d_ℓ	parameters
1	4×3	12	4	16
2	4×4	16	4	20
3	2×4	8	2	10
	total	36	10	**46**

Table 10.1. Parameter count for the $3{-}4{-}4{-}2$ network of Fig. 10.2. Each layer contributes $d_\ell d_{\ell-1}$ weights and d_ℓ biases.

The vectorized and batched forms. Equation (10.2) is already vectorized: one matrix-vector product computes a whole layer at once. To process a *batch* of m inputs, stack them as the columns of a matrix $Z^{[0]} = X \in \mathbb{R}^{d_0 \times m}$ and run

$$A^{[\ell]} = W^{[\ell]}Z^{[\ell-1]} + \boldsymbol{b}^{[\ell]}\mathbf{1}^\top, \qquad Z^{[\ell]} = g\big(A^{[\ell]}\big), \tag{10.3}$$

where $\boldsymbol{b}^{[\ell]}\mathbf{1}^\top$ adds the bias to every column ("broadcasting") and g applies elementwise. The whole network is then a handful of matrix multiplies, which is exactly what makes GPUs so well suited to neural networks: a forward pass is dense linear algebra.

The activation g must be nonlinear, or the whole network would collapse. A composition of affine maps is itself affine: with $g(a) = a$, two layers give $W^{[2]}(W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]} = (W^{[2]}W^{[1]})\boldsymbol{x} + \tilde{\boldsymbol{b}}$, a single linear layer in disguise (Exercise 10.9). Depth buys nothing without a nonlinearity between the layers.

10.3 Activation functions

Three activations recur (Fig. 10.3):

- the *logistic sigmoid* $g(a) = 1/(1+e^{-a})$, smooth and bounded in $(0,1)$, with derivative $g'(a) = g(a)\big(1-g(a)\big)$;
- the *hyperbolic tangent* $g(a) = \tanh(a)$, a rescaled sigmoid bounded in $(-1,1)$ and centered at zero, with $g'(a) = 1 - \tanh^2(a)$;
- the *rectified linear unit* $g(a) = \max(0, a)$, the ReLU, with derivative 1 for $a > 0$ and 0 for $a < 0$, the default in modern deep networks.

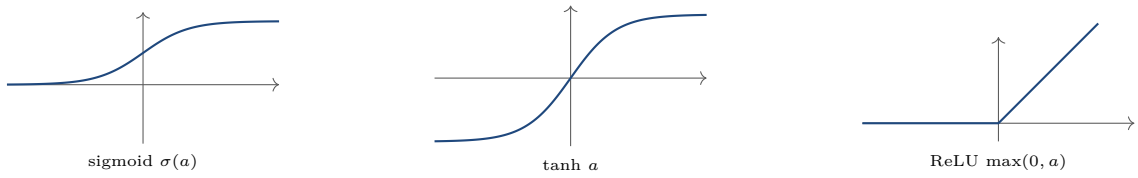


Figure 10.3. The sigmoid and tanh are smooth and bounded but saturate at the extremes; the ReLU has slope 1 on its positive half and does not saturate there.

The sigmoid and tanh *saturate*: for large positive or negative input their output flattens and the derivative goes to zero. Table 10.2 makes this concrete. At $a = 5$ the sigmoid derivative is $\sigma(5)(1 - \sigma(5)) = 0.993 \times 0.007 = 0.0066$, smaller than its peak value 0.25 at $a = 0$ by a factor of nearly forty. A neuron driven into saturation therefore passes almost no gradient backward during training, so it learns slowly; stack many such layers and the gradient all but vanishes by the time it reaches the early layers (Chapter 11 makes this precise). The ReLU sidesteps the problem on its active half, where the derivative is exactly 1 no matter how large a grows, which is the main reason it replaced the sigmoid in deep networks.

a	$\sigma(a)$	$\sigma'(a) = \sigma(1-\sigma)$	$\tanh a$	$1 - \tanh^2 a$	$\max(0, a)$	slope
-2	0.119	0.105	-0.964	0.071	0	0
0	0.500	0.250	0.000	1.000	0	undef.
2	0.881	0.105	0.964	0.071	2	1
5	0.993	0.0066	0.9999	0.0002	5	1

Table 10.2. Values and derivatives of the three activations. The sigmoid and tanh derivatives collapse toward zero as $|a|$ grows (saturation); the ReLU slope stays at 1 on its active half. The ReLU is not differentiable at $a = 0$, where either one-sided value is used in practice.

10.4 A forward pass by hand

Nothing fixes the mechanics like running real numbers through a real network. We take a two-input network with three ReLU hidden units and one sigmoid output, evaluate it on a single input, and write out every entry. This same network is trained by hand, backward pass and all, in Chapter 11.

Example 10.1 (Forward pass through a $2-3-1$ network). The network has

$$W^{[1]} = \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & 0.2 \\ -0.4 & 0.6 \end{bmatrix} \in \mathbb{R}^{3\times 2}, \quad \boldsymbol{b}^{[1]} = \begin{bmatrix} -0.2 \\ -0.5 \\ 0.2 \end{bmatrix}, \quad W^{[2]} = \begin{bmatrix} 1.0 & -2.0 & 0.5 \end{bmatrix} \in \mathbb{R}^{1\times 3}, \quad b^{[2]} = 0.3,$$

with a ReLU on the hidden layer and a sigmoid on the output. Evaluate it at $\boldsymbol{x} = (1, 2)^\top$.

Step 1: hidden pre-activation $\boldsymbol{a}^{[1]} = W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}$. Multiply the matrix by the input, then add the bias:

$$W^{[1]}\boldsymbol{x} = \begin{bmatrix} 0.5(1) + (-0.3)(2) \\ 0.8(1) + 0.2(2) \\ -0.4(1) + 0.6(2) \end{bmatrix} = \begin{bmatrix} -0.1 \\ 1.2 \\ 0.8 \end{bmatrix}, \qquad \boldsymbol{a}^{[1]} = \begin{bmatrix} -0.1 \\ 1.2 \\ 0.8 \end{bmatrix} + \begin{bmatrix} -0.2 \\ -0.5 \\ 0.2 \end{bmatrix} = \begin{bmatrix} -0.3 \\ 0.7 \\ 1.0 \end{bmatrix}.$$

Step 2: hidden activation $\boldsymbol{z}^{[1]} = \max(0, \boldsymbol{a}^{[1]})$. Apply the ReLU elementwise. The first unit's pre-activation is negative, so the ReLU switches it off:

$$\boldsymbol{z}^{[1]} = \big(\max(0, -0.3),\ \max(0, 0.7),\ \max(0, 1.0)\big)^\top = (0,\ 0.7,\ 1.0)^\top.$$

Step 3: output pre-activation $a^{[2]} = W^{[2]}\boldsymbol{z}^{[1]} + b^{[2]}$.

$$a^{[2]} = (1.0)(0) + (-2.0)(0.7) + (0.5)(1.0) + 0.3 = 0 - 1.4 + 0.5 + 0.3 = -0.6.$$

Step 4: output activation. The sigmoid turns the score into a probability,

$$\hat{y} = \sigma(-0.6) = \frac{1}{1+e^{0.6}} = \frac{1}{1+1.822} = 0.354.$$

So the network predicts $\hat{y} \approx 0.354$. Read off what the dead first hidden unit cost us: because $z_1^{[1]} = 0$, the first column of $W^{[2]}$ and the first row of $W^{[1]}$ played no part in this particular prediction, and (as we will see) receive no gradient from this example either. ◊

10.5 How a hidden layer solves XOR

Chapter 7 left the perceptron defeated by XOR, the four points $(0,0)$ and $(1,1)$ labeled 0 and $(0,1)$ and $(1,0)$ labeled 1, because no single line separates the two classes. A network with one hidden layer solves it outright, and working the construction by hand shows exactly how a hidden layer manufactures the feature a linear classifier needs.

Example 10.2 (A network that computes XOR). We build a two-layer network with two ReLU hidden units that computes XOR exactly. Let both hidden units read the sum of the inputs, with different biases:

$$a_1 = x_1 + x_2, \qquad a_2 = x_1 + x_2 - 1, \qquad z_j = \max(0, a_j),$$

that is hidden weights $W^{[1]} = \left[\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right]$ and biases $\boldsymbol{b}^{[1]} = (0, -1)$. The output unit takes the linear combination

$$\hat{y} = z_1 - 2z_2 = \max(0, x_1 + x_2) - 2\max(0, x_1 + x_2 - 1),$$

that is output weights $W^{[2]} = (1, -2)$ and bias 0. Now evaluate it at all four inputs, restating the computation in full each time:

(x_1, x_2)	$a_1 = x_1 + x_2$	$a_2 = x_1 + x_2 - 1$	$z_1 = \max(0, a_1)$	$z_2 = \max(0, a_2)$	$\hat{y} = z_1 - 2z_2$
$(0,0)$	0	-1	0	0	0
$(1,0)$	1	0	1	0	1
$(0,1)$	1	0	1	0	1
$(1,1)$	2	1	2	1	0

The output is $0, 1, 1, 0$, exactly XOR. ◊

Why did it work? The hidden layer moved the four points to new coordinates (z_1, z_2) in which the classes *are* linearly separable (Fig. 10.4). In the original (x_1, x_2) plane the two classes sit on opposite diagonals and no line divides them. After the hidden layer, the points land at $(0,0)$, $(1,0)$, $(1,0)$, and $(2,1)$; the two class-1 inputs collapse onto the single point $(1,0)$, and the line $z_1 - 2z_2 = \frac{1}{2}$ now cleanly separates class 1 (below the line) from class 0 (above it). The output unit is just a linear classifier reading that line. This is the whole idea of deep learning in miniature: each layer re-represents the data so the next layer's job is easier, until a final linear readout suffices.

The XOR construction also makes concrete why the activation *must* be nonlinear; with linear hidden units the composition stays linear and the points never move off their diagonals.

10.6 Universal approximation, and why depth helps

XOR is a special case of a sweeping fact: a single hidden layer can approximate *any* continuous function, given enough units.

Theorem 10.3 (Universal approximation). Let f be a continuous function on a compact subset of $\mathbb{R}^n$. For any tolerance $\varepsilon > 0$ there is a single-hidden-layer network $h(\boldsymbol{x}) = \sum_{j=1}^{M} \alpha_j \, g(\boldsymbol{w}_j^\top \boldsymbol{x} + b_j)$, with a nonpolynomial activation g and enough hidden units M, such that $\sup_{\boldsymbol{x}} |h(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon$.

The result is due to Cybenko [8] for sigmoidal g and Hornik et al. [19] more generally. The intuition is a construction. A single sigmoid $g(a(x - c))$ with large slope a is nearly a step at $x = c$. Subtracting two such steps, $g(a(x - c_1)) -$

Figure 10.4. Left: in the input plane the XOR classes lie on opposite diagonals and no line separates them. Right: the ReLU hidden layer maps the four points to (z_1, z_2); the two class-1 points coincide at $(1, 0)$ and the line $z_1 - 2z_2 = \frac{1}{2}$ separates the classes. The output neuron is a linear classifier on these learned coordinates.

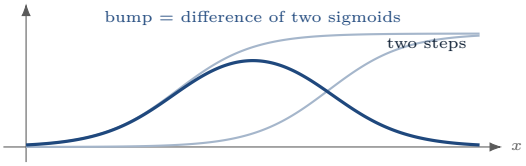


Figure 10.5. A steep sigmoid is nearly a step; subtracting two steps at $c_1 = 2$ and $c_2 = 4$ makes a localized bump. A sum of such bumps, with learned heights and positions, approximates any continuous function, the constructive heart of the universal approximation theorem.

$g(a(x - c_2))$, makes a *bump*: a function near 1 between c_1 and c_2 and near 0 outside (Fig. 10.5). With enough bumps of adjustable height and position one can approximate any continuous curve to any accuracy, the way a sum of narrow rectangles approximates an integral. In several input dimensions the same idea builds localized bumps and tiles the domain.

Universal approximation is reassuring but not the whole story, because it says nothing about *how many* units are needed, and the answer for a single hidden layer can be astronomically large. This is where *depth* earns its keep. Many functions that require an exponential number of units in one wide hidden layer can be represented with only polynomially many units when the network is deep, because depth lets the network *compose* features: early layers detect simple patterns (edges), later layers combine them into complex ones (shapes, then objects), so a complicated function is factored into many simple steps instead of approximated in one gigantic sum. Figure 10.6 sketches that hierarchy. Depth trades width for the ability to reuse and build on intermediate features, and it is the reason the field is called deep learning. The convolutional and recurrent architectures of the next chapters are structured ways of being deep.

Figure 10.6. Depth as composition: each layer builds features from the previous layer's, turning raw pixels into edges, edges into parts, parts into objects, and objects into a label. A shallow network must approximate the whole map in one step.

10.7 Output layers and loss functions

A network produces learned features in its last hidden layer; the output layer turns them into a prediction, and which output layer to use is exactly the generalized linear model question of Chapter 3. The output layer is a GLM on the learned features $\boldsymbol{z}^{[L-1]}$, and the three standard cases match the three exponential-family outputs:

- **Regression**: a linear output $\hat{y} = \boldsymbol{w}^\top \boldsymbol{z}^{[L-1]} + b$ with the *squared-error* loss $\frac{1}{2}\sum_n (\hat{y}_n - t_n)^2$, the Gaussian negative log-likelihood of Chapter 5.
- **Binary classification**: a sigmoid output $\hat{y} = \sigma(\cdot)$ with the *cross-entropy* loss $-\sum_n [t_n \log \hat{y}_n + (1-t_n)\log(1-\hat{y}_n)]$, the Bernoulli negative log-likelihood of Chapter 8.
- **Multiclass classification**: a *softmax* output $\hat{y}_k = e^{a_k} / \sum_j e^{a_j}$ with the categorical *cross-entropy* loss $-\sum_n \sum_k t_{nk} \log \hat{y}_{nk}$.

In each case the loss is the negative log-likelihood of the matching exponential-family distribution, and the output nonlinearity is that distribution's inverse link, exactly as derived in Chapter 3.

Example 10.4 (Softmax and its gradient, by the numbers). A three-class output layer produces logits $\boldsymbol{a}^{[L]} = (2.0, 1.0, 0.1)^\top$. Exponentiate and normalize:

$$e^{\boldsymbol{a}} = (e^{2.0}, e^{1.0}, e^{0.1}) = (7.389, 2.718, 1.105), \qquad \sum_j e^{a_j} = 11.212,$$

$$\hat{\boldsymbol{y}} = \frac{1}{11.212}(7.389, 2.718, 1.105) = (0.659, 0.242, 0.099).$$

The probabilities are positive and sum to 1, and the largest logit gets the largest share. Suppose the true class is the first, a one-hot target $\boldsymbol{t} = (1, 0, 0)$. The cross-entropy loss is $-\log \hat{y}_1 = -\log 0.659 = 0.417$. Its gradient with respect to the logits is the clean "prediction minus target" form

$$\frac{\partial E}{\partial \boldsymbol{a}^{[L]}} = \hat{\boldsymbol{y}} - \boldsymbol{t} = (0.659 - 1,\ 0.242 - 0,\ 0.099 - 0) = (-0.341,\ 0.242,\ 0.099),$$

which is negative on the true class (push that logit up) and positive on the rest (push them down). ◊

The pattern of Example 10.4 is general and is the reason training is tractable: for each matched output-and-loss pair the gradient of the loss with respect to the output pre-activation collapses to

$$\frac{\partial E}{\partial \boldsymbol{a}^{[L]}} = \hat{\boldsymbol{y}} - \boldsymbol{t}, \tag{10.4}$$

prediction minus target. We met this for linear regression, logistic regression, and softmax regression separately; in a network it is the seed from which backpropagation grows the gradients for every earlier layer, the subject of Chapter 11.

Notes and References

The artificial neuron is due to McCulloch and Pitts [29] and the perceptron learning rule to Rosenblatt [32]; the term "artificial intelligence" was coined for the 1956 Dartmouth workshop [28]. The feed-forward architecture, activation functions, and output-layer choices follow Sections 5.1 and 5.2 of Bishop [3] and Chapter 6 of Goodfellow et al. [13]. The universal approximation theorem is due to Cybenko [8] for sigmoidal activations and Hornik et al. [19] for general ones; the bump-construction intuition is standard. The advantage of depth over width, that deep networks can represent with polynomially many units functions requiring exponentially many in a shallow network, is discussed at length in Goodfellow et al. [13]. The XOR construction of Example 10.2 resolves the perceptron's limitation from Chapter 7, and the matched output-layer losses tie back to the exponential family of Chapter 3. The algorithm that trains all these weights, backpropagation, is Chapter 11.

Exercises

Exercise 10.1. Take the $2-3-1$ network of Example 10.1 with the same weights and biases, but change the input to $\boldsymbol{x} = (-1, 1)^\top$. Compute $\boldsymbol{a}^{[1]}$, apply the ReLU to get $\boldsymbol{z}^{[1]}$, then compute $a^{[2]}$ and $\hat{y} = \sigma(a^{[2]})$, showing every entry.

Solution. $W^{[1]}\boldsymbol{x} = (0.5(-1) + (-0.3)(1),\ 0.8(-1) + 0.2(1),\ -0.4(-1) + 0.6(1))^\top = (-0.8, -0.6, 1.0)^\top$. Adding $\boldsymbol{b}^{[1]} = (-0.2, -0.5, 0.2)^\top$ gives $\boldsymbol{a}^{[1]} = (-1.0, -1.1, 1.2)^\top$. The ReLU zeroes the first two: $\boldsymbol{z}^{[1]} = (0, 0, 1.2)^\top$. Then $a^{[2]} = (1.0)(0) + (-2.0)(0) + (0.5)(1.2) + 0.3 = 0.6 + 0.3 = 0.9$, and $\hat{y} = \sigma(0.9) = 1/(1 + e^{-0.9}) = 1/(1 + 0.407) = 0.711$. Two of the three hidden units are dead for this input, so only the third row of $W^{[1]}$ and third column of $W^{[2]}$ shaped the prediction. □

Exercise 10.2. A fully connected network has layer sizes $10-64-64-32-1$. How many weights, how many biases, and how many parameters total does it have?

Solution. Weights per layer are $d_\ell d_{\ell-1}$: $64{\cdot}10 = 640$, $64{\cdot}64 = 4096$, $32{\cdot}64 = 2048$, $1{\cdot}32 = 32$, totalling 6816 weights. Biases are d_ℓ: $64 + 64 + 32 + 1 = 161$. The network has $6816 + 161 = 6977$ parameters. □

Exercise 10.3. For the logistic sigmoid, compute the derivative $\sigma'(a) = \sigma(a)(1 - \sigma(a))$ at $a = 0$ and at $a = 4$, and state the ratio. What does the result imply for a deep stack of sigmoid layers?

Solution. $\sigma(0) = 0.5$, so $\sigma'(0) = 0.5 \times 0.5 = 0.25$. $\sigma(4) = 1/(1 + e^{-4}) = 1/(1 + 0.0183) = 0.982$, so $\sigma'(4) = 0.982 \times 0.018 = 0.0177$. The ratio is $0.25/0.0177 \approx 14$: a saturated unit passes roughly fourteen times less gradient than one at its midpoint. In a deep stack these small factors multiply, so the gradient reaching the early layers can be vanishingly small (the vanishing-gradient problem), which is why ReLU activations and architectural fixes like residual connections are used in deep networks. □

Exercise 10.4. A softmax output layer produces logits $\boldsymbol{a} = (0, 1, 2)^\top$. Compute the predicted probabilities $\hat{\boldsymbol{y}}$, the cross-entropy loss if the true class is the third, and the gradient $\hat{\boldsymbol{y}} - \boldsymbol{t}$.

Solution. $e^{\boldsymbol{a}} = (1, 2.718, 7.389)$ summing to 11.107, so $\hat{\boldsymbol{y}} = (0.090, 0.245, 0.665)$. With $\boldsymbol{t} = (0, 0, 1)$ the loss is $-\log 0.665 = 0.408$. The gradient is $\hat{\boldsymbol{y}} - \boldsymbol{t} = (0.090, 0.245, 0.665 - 1) = (0.090, 0.245, -0.335)$: push the two wrong logits down, the correct one up. □

Exercise 10.5. Verify that the two-ReLU-hidden-unit network $\hat{y} = \max(0, x_1 + x_2) - 2\max(0, x_1 + x_2 - 1)$ computes XOR by evaluating it at all four binary inputs, and explain in one sentence why a single neuron cannot.

Solution. At $(0, 0)$: $\max(0, 0) - 2\max(0, -1) = 0 - 0 = 0$. At $(1, 0)$ and $(0, 1)$: $\max(0, 1) - 2\max(0, 0) = 1 - 0 = 1$. At $(1, 1)$: $\max(0, 2) - 2\max(0, 1) = 2 - 2 = 0$. The outputs $0, 1, 1, 0$ are XOR. A single neuron computes σ or sign of a linear function, whose decision boundary is one hyperplane, and XOR's two classes (the two diagonals) cannot be separated by any single line, so one neuron cannot represent it. □

Exercise 10.6. Show that two ReLU units can represent $f(x, y) = |2x - y|$ exactly, and give the weights.

Solution. Use the identity $|t| = \max(0, t) + \max(0, -t)$. Setting $t = 2x - y$, take two hidden units $z_1 = \max(0, 2x - y)$ and $z_2 = \max(0, -(2x - y)) = \max(0, y - 2x)$, that is hidden weights $W^{[1]} = \left[\begin{smallmatrix} 2 & -1 \\ -2 & 1 \end{smallmatrix}\right]$ and zero biases. The output is $z_1 + z_2 = \max(0, 2x - y) + \max(0, y - 2x) = |2x - y|$, with output weights $(1, 1)$ and zero bias. This holds exactly for all (x, y). □

Exercise 10.7. Explain how to build an approximate bump function (near 1 on an interval $[c_1, c_2]$, near 0 outside) from two sigmoids, and how a sum of bumps approximates an arbitrary continuous function.

Solution. A sigmoid $\sigma(a(x - c))$ with large slope a rises sharply from 0 to 1 near $x = c$, so it is approximately a step at c. The difference $\sigma(a(x - c_1)) - \sigma(a(x - c_2))$ with $c_1 < c_2$ is near 1 for x between c_1 and c_2 (where the first step is on and the second still off) and near 0 outside, an approximate bump on $[c_1, c_2]$. A weighted sum of such

bumps, $\sum_j \alpha_j\,[\text{bump}_j]$, can match the height α_j of a target function on each small interval, so as the bumps are made narrow and numerous the sum approximates any continuous function uniformly, which is the universal approximation theorem. □

Exercise 10.8. A single hidden layer can approximate any continuous function (Theorem 10.3), so why are deep networks used at all? Answer in terms of the number of units required.

Solution. Universal approximation guarantees a shallow network *exists* but says nothing about its size, and for many functions the required number of hidden units grows exponentially with the input dimension or the function's complexity. A deep network can represent the same functions with far fewer units, polynomially many rather than exponentially, because depth lets it compose features: early layers compute simple intermediate features that later layers reuse and combine, factoring a complex function into many simple steps. Depth trades width for compositional reuse, which is why deep architectures are preferred in practice. □

Exercise 10.9. Show that a feed-forward network with *linear* activations $g(a) = a$ in every layer computes only a linear function of its input, no matter how many layers it has.

Solution. With $g(a) = a$ each layer is the affine map $\boldsymbol{z}^{[\ell]} = W^{[\ell]}\boldsymbol{z}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}$. Composing two such maps gives $W^{[2]}(W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]}) + \boldsymbol{b}^{[2]} = (W^{[2]}W^{[1]})\boldsymbol{x} + (W^{[2]}\boldsymbol{b}^{[1]} + \boldsymbol{b}^{[2]})$, again an affine map with weight matrix $W^{[2]}W^{[1]}$ and a combined bias. By induction, any number of linear layers composes to a single affine map $\tilde{W}\boldsymbol{x} + \tilde{\boldsymbol{b}}$, no more expressive than one linear layer. This is why the activation must be nonlinear: without it, depth buys nothing. □

CHAPTER 11

Backpropagation and Network Training

Backpropagation is just the chain rule, organized so that you compute every derivative once and reuse it. That bookkeeping is the whole trick.

after Professor Peter Chin, ENGS 106

Chapter 10 built networks but left them untrained. Training means minimizing a loss $E(\boldsymbol{w})$ over all the weights, and the tool is gradient descent from Chapter 5: step downhill, $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla E$. The only hard part is computing the gradient, because a deep network has the loss buried under many nested layers, and the weight of an early layer affects the loss only through everything downstream of it. *Backpropagation* is the algorithm that computes all these derivatives at once, and it is nothing more than the chain rule applied with care, sweeping backward through the network and reusing partial results so that the entire gradient costs about as much as a single forward pass. This chapter derives it, traces it by hand on a small network with real numbers, explains why it is so cheap, and then surveys the practical machinery that makes training deep networks actually work: the optimizers that improve on plain gradient descent, the vanishing-gradient problem that long stalled deep learning, and the regularizers that keep large networks from overfitting.

Roadmap

Suggested reading: Bishop [3], Section 5.3; Goodfellow et al. [13], Chapters 6 and 8.

We cover: the chain rule and the credit-assignment problem (Section 11.1); the backpropagation equations, derived in full (Section 11.2); a complete hand-traced example, first scalar and then in full matrix form, checked against finite differences (Section 11.3); backprop as automatic differentiation and why it is cheap (Section 11.4); gradient-descent variants from SGD to Adam (Section 11.5); vanishing and exploding gradients, with the initialization and normalization that tame them (Section 11.6); and regularization (Section 11.7).

11.1 The chain rule and credit assignment

Recall the forward pass of Chapter 10: with $\boldsymbol{z}^{[0]} = \boldsymbol{x}$, each layer computes a pre-activation and an activation,

$$\boldsymbol{a}^{[\ell]} = W^{[\ell]} \boldsymbol{z}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}, \qquad \boldsymbol{z}^{[\ell]} = g(\boldsymbol{a}^{[\ell]}),$$

ending in the output $\boldsymbol{z}^{[L]}$ and a scalar loss E. We need $\partial E / \partial W^{[\ell]}$ and $\partial E / \partial \boldsymbol{b}^{[\ell]}$ for every layer. The difficulty is credit assignment: changing an early weight nudges its layer's output, which nudges the next layer's, and so on until the loss changes at the very end, so the derivative is a long chain of nested dependencies. The chain rule handles exactly this, and the key realization is that the chains for different layers share their tail ends. If we compute the shared parts once, starting from the output and moving backward, every layer's gradient comes almost for free. The quantity worth computing and reusing is the sensitivity of the loss to each layer's pre-activation,

$$\boldsymbol{\delta}^{[\ell]} = \frac{\partial E}{\partial \boldsymbol{a}^{[\ell]}}, \tag{11.1}$$

the *error signal* at layer ℓ. Backpropagation is the recursion that computes $\boldsymbol{\delta}^{[\ell]}$ from $\boldsymbol{\delta}^{[\ell+1]}$.

11.2 The backpropagation equations

Start at the output. For the matched output-layer-and-loss pairs of Chapter 10 (linear output with squared error, sigmoid with cross-entropy, softmax with cross-entropy), we showed the output error signal collapses to prediction minus target,

$$\boldsymbol{\delta}^{[L]} = \frac{\partial E}{\partial \boldsymbol{a}^{[L]}} = \boldsymbol{z}^{[L]} - \boldsymbol{t} = \hat{\boldsymbol{y}} - \boldsymbol{t}. \tag{11.2}$$

Now push the signal back one layer. The loss depends on $\boldsymbol{a}^{[\ell]}$ only through the next layer's pre-activation $\boldsymbol{a}^{[\ell+1]} = W^{[\ell+1]}\boldsymbol{z}^{[\ell]} + \boldsymbol{b}^{[\ell+1]}$, and $\boldsymbol{z}^{[\ell]} = g(\boldsymbol{a}^{[\ell]})$, so by the chain rule, component by component,

$$\delta_i^{[\ell]} = \frac{\partial E}{\partial a_i^{[\ell]}} = \sum_j \frac{\partial E}{\partial a_j^{[\ell+1]}} \frac{\partial a_j^{[\ell+1]}}{\partial a_i^{[\ell]}} = \sum_j \delta_j^{[\ell+1]} W_{ji}^{[\ell+1]} g'(a_i^{[\ell]}),$$

because $\partial a_j^{[\ell+1]}/\partial z_i^{[\ell]} = W_{ji}^{[\ell+1]}$ and $\partial z_i^{[\ell]}/\partial a_i^{[\ell]} = g'(a_i^{[\ell]})$. In vector form, with $\odot$ the elementwise (Hadamard) product, this is the backward recursion

$$\boldsymbol{\delta}^{[\ell]} = \left(W^{[\ell+1]\top}\boldsymbol{\delta}^{[\ell+1]}\right) \odot g'(\boldsymbol{a}^{[\ell]}). \tag{11.3}$$

The matrix $W^{[\ell+1]\top}$ propagates the error from layer $\ell+1$ back to layer ℓ (the forward weights, transposed), and the $\odot\, g'(\boldsymbol{a}^{[\ell]})$ accounts for the activation. Once the error signals are known, the weight and bias gradients fall out, because E depends on $W^{[\ell]}$ and $\boldsymbol{b}^{[\ell]}$ only through $\boldsymbol{a}^{[\ell]} = W^{[\ell]}\boldsymbol{z}^{[\ell-1]} + \boldsymbol{b}^{[\ell]}$:

$$\frac{\partial E}{\partial W^{[\ell]}} = \boldsymbol{\delta}^{[\ell]}\, \boldsymbol{z}^{[\ell-1]\top}, \qquad \frac{\partial E}{\partial \boldsymbol{b}^{[\ell]}} = \boldsymbol{\delta}^{[\ell]}. \tag{11.4}$$

The weight gradient is an outer product: the error at a layer times the activation that fed into it, the same "error times input" structure seen in every model since Chapter 5.

Backpropagation

Forward pass: compute and store $\boldsymbol{a}^{[\ell]}, \boldsymbol{z}^{[\ell]}$ for $\ell = 1, \dots, L$.
Backward pass: set $\boldsymbol{\delta}^{[L]} = \hat{\boldsymbol{y}} - \boldsymbol{t}$; for $\ell = L-1, \dots, 1$ recurse $\boldsymbol{\delta}^{[\ell]} = (W^{[\ell+1]\top}\boldsymbol{\delta}^{[\ell+1]}) \odot g'(\boldsymbol{a}^{[\ell]})$.
Gradients: $\dfrac{\partial E}{\partial W^{[\ell]}} = \boldsymbol{\delta}^{[\ell]}\boldsymbol{z}^{[\ell-1]\top}$, $\dfrac{\partial E}{\partial \boldsymbol{b}^{[\ell]}} = \boldsymbol{\delta}^{[\ell]}$.

A dimensional check confirms the wiring. If layer ℓ has d_ℓ units, then $W^{[\ell]}$ is $d_\ell \times d_{\ell-1}$, $\boldsymbol{\delta}^{[\ell]}$ is $d_\ell \times 1$, and $\boldsymbol{z}^{[\ell-1]}$ is $d_{\ell-1} \times 1$, so the outer product $\boldsymbol{\delta}^{[\ell]}\boldsymbol{z}^{[\ell-1]\top}$ is $d_\ell \times d_{\ell-1}$, matching $W^{[\ell]}$ exactly; and $W^{[\ell+1]\top}\boldsymbol{\delta}^{[\ell+1]}$ is $(d_\ell \times d_{\ell+1})(d_{\ell+1} \times 1) = d_\ell \times 1$, matching $\boldsymbol{\delta}^{[\ell]}$. Everything lines up.

11.3 A backpropagation pass by hand

Nothing makes the recursion concrete like running it on real numbers, so we trace a tiny network end to end.

Example 11.1 (Forward and backward by hand). Consider a network with one input, one sigmoid hidden unit, and one linear output unit, trained with the squared-error loss $E = \frac{1}{2}(\hat{y} - t)^2$. The parameters are the input-to-hidden weight $w_1 = 0.5$ and bias $b_1 = 0$, and the hidden-to-output weight $w_2 = 0.5$ and bias $b_2 = 0$. The training example is the input $x = 1$ with target $t = 0$.

Forward pass. The hidden pre-activation is $a_1 = w_1 x + b_1 = (0.5)(1) + 0 = 0.5$. The hidden activation is $z_1 = \sigma(a_1) = 1/(1+e^{-0.5}) = 1/(1+0.6065) = 0.6225$. The output pre-activation is $a_2 = w_2 z_1 + b_2 = (0.5)(0.6225) = 0.3112$, and with a linear output $\hat{y} = a_2 = 0.3112$. The loss is $E = \frac{1}{2}(0.3112 - 0)^2 = \frac{1}{2}(0.0969) = 0.0484$.

Backward pass. The output error signal is $\delta_2 = \partial E/\partial a_2 = \hat{y} - t = 0.3112$. The output-weight gradient is $\partial E/\partial w_2 = \delta_2\, z_1 = (0.3112)(0.6225) = 0.1937$, and $\partial E/\partial b_2 = \delta_2 = 0.3112$. Propagating back to the hidden unit, the recursion Eq. (11.3) reads $\delta_1 = (w_2\, \delta_2)\, \sigma'(a_1)$. The sigmoid derivative is $\sigma'(a_1) = \sigma(a_1)(1-\sigma(a_1)) = (0.6225)(0.3775) = 0.2350$, so $\delta_1 = (0.5)(0.3112)(0.2350) = (0.1556)(0.2350) = 0.0366$. The hidden-weight gradient is $\partial E/\partial w_1 = \delta_1\, x = (0.0366)(1) = 0.0366$, and $\partial E/\partial b_1 = \delta_1 = 0.0366$.

Update. A gradient-descent step with learning rate $\eta = 1$ gives $w_2 \leftarrow 0.5 - 0.1937 = 0.306$ and $w_1 \leftarrow 0.5 - 0.0366 = 0.463$, both reduced so the next forward pass will push $\hat{y}$ closer to the target 0. Notice the output weight changed far more than the input weight (0.194 versus 0.037): the error signal shrank by the factor $w_2\, \sigma'(a_1) = (0.5)(0.235) = 0.118$ as it crossed the sigmoid, a first glimpse of the vanishing-gradient effect of Section 11.6. ◊

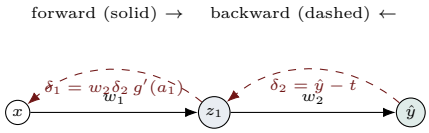


Figure 11.1. The network of Example 11.1. The forward pass (solid) carries activations left to right; the backward pass (dashed) carries error signals $\boldsymbol{\delta}$ right to left, each weight gradient being the local error times the incoming activation. The error shrinks by $w_2 g'(a_1)$ as it crosses the sigmoid.

11.3.1 The same recursion in matrix form

The scalar trace shows the mechanics, but real layers are vectors and the gradients are matrices, so we now run the backward pass on the multi-unit network we already pushed *forward* in Example 10.1, writing every vector and matrix out in full. This is the single most important worked example in the chapter: follow each number and the abstract recursion (11.3) becomes concrete.

Example 11.2 (Backpropagation through a $2-3-1$ network). The network has a ReLU hidden layer and a sigmoid output, with

$$W^{[1]} = \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & 0.2 \\ -0.4 & 0.6 \end{bmatrix}, \quad \boldsymbol{b}^{[1]} = \begin{bmatrix} -0.2 \\ -0.5 \\ 0.2 \end{bmatrix}, \quad W^{[2]} = \begin{bmatrix} 1.0 & -2.0 & 0.5 \end{bmatrix}, \quad b^{[2]} = 0.3.$$

The training example is $\boldsymbol{x} = (1, 2)^\top$ with target class $t = 1$, and the loss is the binary cross-entropy $E = -[t \log \hat{y} + (1 - t) \log(1 - \hat{y})]$.

Forward pass (from Example 10.1). The hidden pre-activation, activation, output pre-activation, and prediction are

$$\boldsymbol{a}^{[1]} = \begin{bmatrix} -0.3 \\ 0.7 \\ 1.0 \end{bmatrix}, \quad \boldsymbol{z}^{[1]} = \max(0, \boldsymbol{a}^{[1]}) = \begin{bmatrix} 0 \\ 0.7 \\ 1.0 \end{bmatrix}, \quad a^{[2]} = -0.6, \quad \hat{y} = \sigma(-0.6) = 0.354.$$

With $t = 1$ the loss is $E = -\log \hat{y} = -\log 0.354 = 1.038$.

Step 1: output error signal. For the matched sigmoid-and-cross-entropy pair, $\boldsymbol{\delta}^{[2]} = \hat{y} - t$ (this is Eq. (11.2), and Exercise 11.6 derives it):

$$\delta^{[2]} = \hat{y} - t = 0.354 - 1 = -0.646.$$

It is negative because the network predicted 0.354 when the target was 1; the backward pass will move every weight to push $\hat{y}$ upward.

Step 2: output-layer gradients. By Eq. (11.4), the weight gradient is the outer product of the error with the activation that fed the layer, and the bias gradient is the error itself:

$$\frac{\partial E}{\partial W^{[2]}} = \delta^{[2]} \, \boldsymbol{z}^{[1]\top} = -0.646 \begin{bmatrix} 0 & 0.7 & 1.0 \end{bmatrix} = \begin{bmatrix} 0 & -0.452 & -0.646 \end{bmatrix}, \qquad \frac{\partial E}{\partial b^{[2]}} = \delta^{[2]} = -0.646.$$

Step 3: propagate the error back through the layer. First send it through the transposed weights, $W^{[2]\top} \delta^{[2]}$, then gate it by the ReLU derivative $g'(\boldsymbol{a}^{[1]})$, which is 1 where the pre-activation was positive and 0 where it was negative:

$$W^{[2]\top} \delta^{[2]} = \begin{bmatrix} 1.0 \\ -2.0 \\ 0.5 \end{bmatrix} (-0.646) = \begin{bmatrix} -0.646 \\ 1.292 \\ -0.323 \end{bmatrix}, \qquad g'(\boldsymbol{a}^{[1]}) = \begin{bmatrix} \mathbf{1}[-0.3 > 0] \\ \mathbf{1}[0.7 > 0] \\ \mathbf{1}[1.0 > 0] \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}.$$

Their Hadamard product is the hidden error signal:

$$\boldsymbol{\delta}^{[1]} = \left(W^{[2]\top} \delta^{[2]} \right) \odot g'(\boldsymbol{a}^{[1]}) = \begin{bmatrix} -0.646 \\ 1.292 \\ -0.323 \end{bmatrix} \odot \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1.292 \\ -0.323 \end{bmatrix}.$$

The first component is zero: the first hidden unit was dead on the forward pass ($z_1^{[1]} = 0$), and it is dead on the backward pass too. A unit that contributed nothing to the prediction receives no blame for the error, and (next step) its weights do not move.

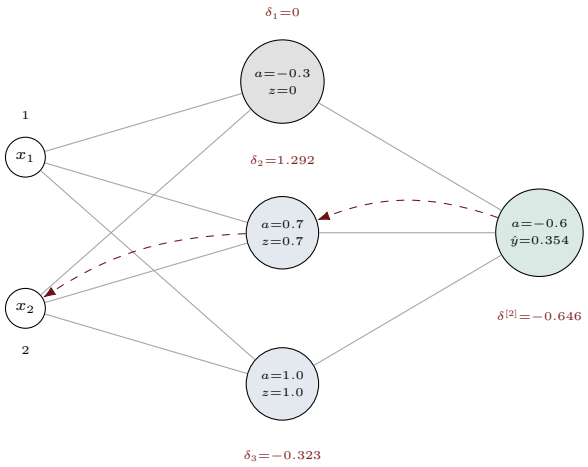


Figure 11.2. The $2-3-1$ network of Example 11.2. Each unit shows its forward pre-activation a and activation z; the red labels are the backward error signals δ. The top hidden unit is dead (gray): its ReLU output and its error signal are both zero, so its incoming weights receive no update.

Step 4: hidden-layer gradients. Again an outer product, now with the network input $\boldsymbol{z}^{[0]} = \boldsymbol{x}$:

$$\frac{\partial E}{\partial W^{[1]}} = \boldsymbol{\delta}^{[1]}\,\boldsymbol{x}^{\top} = \begin{bmatrix} 0 \\ 1.292 \\ -0.323 \end{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1.292 & 2.584 \\ -0.323 & -0.646 \end{bmatrix}, \qquad \frac{\partial E}{\partial \boldsymbol{b}^{[1]}} = \boldsymbol{\delta}^{[1]} = \begin{bmatrix} 0 \\ 1.292 \\ -0.323 \end{bmatrix}.$$

The first row is all zeros, confirming that the dead unit's incoming weights get no update.

Step 5: the gradient-descent update. With learning rate $\eta = 0.1$, subtract η times each gradient:

$$W^{[2]} \leftarrow \begin{bmatrix} 1.0 & -1.955 & 0.565 \end{bmatrix}, \qquad W^{[1]} \leftarrow \begin{bmatrix} 0.5 & -0.3 \\ 0.671 & -0.058 \\ -0.368 & 0.665 \end{bmatrix},$$

with biases $b^{[2]} \leftarrow 0.365$ and $\boldsymbol{b}^{[1]} \leftarrow (-0.2, -0.629, 0.232)^{\top}$. The first row of $W^{[1]}$ is unchanged, as promised. Re-running the forward pass with the updated weights gives $\hat{y} = 0.739$, up from 0.354 and closer to the target 1: one step of training did its job. ◊

Figure 11.2 draws this network with the forward values and backward error signals side by side, so the flow of Example 11.2 can be read off the picture.

11.3.2 Gradient checking

How do you know the backward pass is correct? Compare it against the definition of the derivative. A central finite difference estimates any partial $\partial E/\partial w$ by perturbing that one weight up and down and dividing,

$$\frac{\partial E}{\partial w} \approx \frac{E(w+\epsilon) - E(w-\epsilon)}{2\epsilon}, \tag{11.5}$$

which is accurate to order ϵ^2. Take the weight $W_2^{[2]} = -2.0$ from Example 11.2, whose backprop gradient we found to be $\partial E/\partial W_2^{[2]} = \delta^{[2]} z_2^{[1]} = (-0.646)(0.7) = -0.452$. As a function of this one weight the output pre-activation is $a^{[2]} = 0.7\,W_2^{[2]} + 0.8$ and the loss is $E = \log(1 + e^{-a^{[2]}})$. With $\epsilon = 10^{-2}$,

$$E(W_2^{[2]} + \epsilon) = E(-1.99) = 1.03298, \qquad E(W_2^{[2]} - \epsilon) = E(-2.01) = 1.04201,$$

$$\frac{1.03298 - 1.04201}{2(0.01)} = -0.451,$$

which matches the backprop value -0.452 to the precision shown. Run on every weight, this check catches almost any bug in a hand-written backward pass; it is too slow to use for actual training (one pair of forward passes per weight, the very cost backprop avoids), but indispensable for verifying an implementation once.

11.4 Backpropagation is automatic differentiation

Backpropagation is one instance of a general technique, *reverse-mode automatic differentiation*, which computes the derivatives of any function defined by a computation graph. A computation graph is the network drawn as a DAG of elementary operations (add, multiply, sigmoid), each with a known local derivative. The forward pass evaluates the graph and stores intermediate values; the backward pass applies the chain rule through the local derivatives, accumulating the sensitivity of the output to every intermediate quantity. Backpropagation is exactly this run on a neural network. Figure 11.3 draws the scalar network of Example 11.1 as such a graph: each box is an elementary operation, the forward pass flows left to right computing values, and the backward pass flows right to left multiplying by each operation's local derivative, which is the chain rule made mechanical.

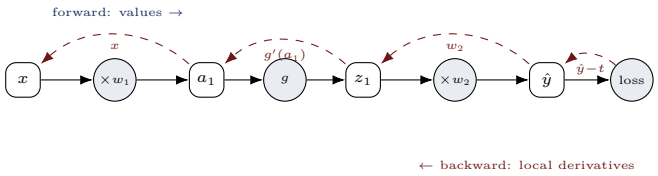


Figure 11.3. The network of Example 11.1 as a computation graph. The forward pass (top, solid) evaluates each operation; the backward pass (bottom, dashed) multiplies by each operation's local derivative, accumulating $\partial E/\partial(\cdot)$ at every node. Reverse-mode automatic differentiation is this procedure for any computation graph, and a deep-learning framework's `backward()` call is exactly this run automatically.

What makes it indispensable is its cost. Suppose evaluating the loss (one forward pass) takes m operations. There are several ways to get the gradient with respect to the n weights, and they differ enormously:

- **Finite differences** perturb each weight in turn, $\partial E/\partial w_i \approx (E(\boldsymbol{w}+\epsilon \boldsymbol{e}_i) - E(\boldsymbol{w}))/\epsilon$, costing one forward pass per weight, $\mathcal{O}(nm)$ operations, and suffering numerical error.
- **Symbolic or by-hand** differentiation re-derives each partial separately, also $\mathcal{O}(nm)$ and error-prone.
- **Reverse-mode automatic differentiation** (backpropagation) computes *all* n partials in a single backward pass whose cost is a small constant times the forward pass, about $4m$ operations, *independent of* n.

The cheap-gradient principle

If the loss costs m operations to evaluate, backpropagation computes the gradient with respect to *all* parameters in about $4m$ operations, regardless of how many parameters there are. If you can afford one forward pass, you can afford the entire gradient.

This is why networks with millions or billions of weights can be trained at all: the gradient is no more expensive than the prediction. It is also why deep-learning frameworks feel like magic. When you call `backward()` on a loss, the framework simply runs the recorded computation graph in reverse, applying the local derivative rules; the user never writes a gradient. The whole apparatus rests on the principle above.

11.5 Gradient-descent variants

Plain gradient descent uses the full-data gradient for each step, which is accurate but expensive when the data set is large. The practical variants trade accuracy for speed and stability.

Stochastic gradient descent (SGD) estimates the gradient from a single example, or a small *mini-batch*, and takes many cheap, noisy steps. The noise is not only tolerable but often helpful, because it lets the iterate escape shallow local minima and saddle points. Convergence is guaranteed under the Robbins–Monro conditions on the learning-rate schedule η_t, namely $\sum_t \eta_t = \infty$ (the steps can travel any distance) and $\sum_t \eta_t^2 < \infty$ (the noise is eventually damped).

Momentum accelerates descent by accumulating a velocity, averaging recent gradients so that consistent directions build up speed and oscillations cancel:

$$\boldsymbol{v}_t = \beta\, \boldsymbol{v}_{t-1} + \nabla E(\boldsymbol{w}_t), \qquad \boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta\, \boldsymbol{v}_t, \tag{11.6}$$

with $\beta \approx 0.9$. *RMSProp* divides each coordinate's step by a running root-mean-square of its recent gradients, so steep and flat directions are rescaled to a common footing. *Adam* [21] combines both ideas, keeping running averages of the

gradient (a momentum term) and of its square (an RMSProp term) and correcting them for initialization bias; it is the default optimizer for most deep networks. All of these still rely on backpropagation for the gradient itself; they only change how the gradient is turned into a step.

11.6 Vanishing and exploding gradients

The backward recursion Eq. (11.3) multiplies the error signal by a weight matrix and an activation derivative at every layer. Unrolled across L layers, the gradient at the first layer is a product of L such factors,

$$\frac{\partial E}{\partial \boldsymbol{a}^{[1]}} \sim \prod_{\ell=2}^{L} W^{[\ell]\top} \, g'(\boldsymbol{a}^{[\ell]}).$$

A product of many numbers is unstable. If the typical factor has magnitude less than one, the product shrinks geometrically and the gradient *vanishes*: early layers receive almost no learning signal, and a deep network trains its first layers agonizingly slowly or not at all. If the typical factor exceeds one, the product grows geometrically and the gradient *explodes*, producing wild updates and numerical overflow. We already saw the seed of this in Example 11.1, where the error shrank by 0.118 crossing a single sigmoid; across thirty layers a factor like that becomes $0.118^{30} \approx 10^{-28}$.

The sigmoid is a chief culprit, because its derivative $\sigma'(a) = \sigma(a)(1-\sigma(a))$ never exceeds $\frac{1}{4}$ and is near zero whenever the unit is saturated. Several remedies tame the product. The *ReLU* activation has derivative exactly 1 on its active side, so it neither shrinks nor inflates the signal there, which is the main reason it replaced the sigmoid in deep networks. *Careful initialization* scales the initial weights so the typical factor is near one. *Normalization* layers rescale activations between layers to keep them in a healthy range. And *residual connections*, which add a layer's input to its output, $\boldsymbol{z}^{[\ell+1]} = \boldsymbol{z}^{[\ell]} + F(\boldsymbol{z}^{[\ell]})$, give the gradient an unimpeded path backward through the identity term, the innovation behind the very deep ResNet of Chapter 12.

11.6.1 Initialization and normalization

Two of those remedies deserve formulas, because they are what makes a deep network train from scratch. *Initialization* sets the scale of the random starting weights so that signals neither blow up nor die out as they pass through the layers. If a unit sums n_{in} inputs each of variance 1 through independent zero-mean weights of variance σ_w^2, its pre-activation has variance $n_{\text{in}}\sigma_w^2$. To keep that at 1 layer after layer we need $\sigma_w^2 = 1/n_{\text{in}}$, the *Xavier* (Glorot) rule for tanh-like activations. A ReLU zeroes half its inputs and so halves the variance, which is repaired by doubling the weight variance: the *He* rule

$$\sigma_w^2 = \frac{2}{n_{\text{in}}}, \tag{11.7}$$

the standard initialization for ReLU networks. For a layer with $n_{\text{in}} = 256$ inputs this is a standard deviation of $\sqrt{2/256} = 0.088$: small weights, scaled to the fan-in.

Normalization does at run time what initialization does at the start. *Batch normalization* standardizes each pre-activation across the current mini-batch, subtracting the batch mean and dividing by the batch standard deviation, then restoring flexibility with a learned scale γ and shift β:

$$\hat{a} = \frac{a - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}, \qquad a_{\text{out}} = \gamma\,\hat{a} + \beta. \tag{11.8}$$

By holding every layer's inputs to a stable distribution it keeps units out of saturation, lets larger learning rates be used, and noticeably speeds training; *layer normalization*, which standardizes across features within a single example rather than across the batch, plays the same role in the sequence models of Chapter 13.

11.7 Regularization

A network large enough to be useful is large enough to overfit, the bias-variance tension of Chapter 4, so training includes a regularizer. The simplest is *weight decay*, adding $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$ to the loss, which is the ridge penalty of Chapter 5 and, as Chapter 9 showed, a Gaussian prior on the weights; it shrinks weights toward zero and discourages the network from relying on any one connection too heavily. *Early stopping* halts training when validation error starts to rise, before the network has fit the training noise; it can be shown to act like weight decay. *Dropout* randomly zeroes a fraction of units on each training step, forcing the network to spread its representation across many units rather than depend on a few, which behaves like training an ensemble of subnetworks and sharing their weights. *Data augmentation* enlarges the training set with label-preserving transformations of the inputs (rotations and crops of images, for instance), building in the invariances the task should respect. Each of these is a different way to lower effective capacity, and the right amount of each is chosen by the cross-validation of Chapter 4.

Notes and References

Backpropagation was popularized for neural networks by Rumelhart et al. [33], and the derivation here follows Section 5.3 of Bishop [3]. The view of backpropagation as reverse-mode automatic differentiation, and the cheap-gradient principle that the full gradient costs a small constant times the forward pass, is developed in Goodfellow et al. [13], Chapter 6, which also treats the optimizers of Section 11.5, the vanishing-gradient problem of Section 11.6, and the regularizers of Section 11.7 at length. Adam is due to Kingma and Ba [21]. The residual connection that addresses vanishing gradients is the subject of the ResNet architecture in Chapter 12, and the Bayesian view of weight decay as a Gaussian prior is Chapter 9.

Exercises

Exercise 11.1. A network has one input, one sigmoid hidden unit, and one linear output, with squared-error loss $E = \frac{1}{2}(\hat{y} - t)^2$ and weights $w_1 = 1$, $b_1 = 0$, $w_2 = 2$, $b_2 = 0$. For the example $x = 1$, $t = 1$, carry out the forward pass and the backward pass, reporting all four gradients.

Solution. *Forward:* $a_1 = w_1 x + b_1 = 1$; $z_1 = \sigma(1) = 1/(1 + e^{-1}) = 0.731$; $a_2 = w_2 z_1 = 2(0.731) = 1.462$; $\hat{y} = a_2 = 1.462$; $E = \frac{1}{2}(1.462 - 1)^2 = \frac{1}{2}(0.213) = 0.107$. *Backward:* $\delta_2 = \hat{y} - t = 0.462$; $\partial E/\partial w_2 = \delta_2 z_1 = 0.462(0.731) = 0.338$; $\partial E/\partial b_2 = \delta_2 = 0.462$. For the hidden unit, $\sigma'(1) = \sigma(1)(1 - \sigma(1)) = 0.731(0.269) = 0.197$, so $\delta_1 = w_2 \delta_2\, \sigma'(1) = 2(0.462)(0.197) = 0.182$; $\partial E/\partial w_1 = \delta_1 x = 0.182$; $\partial E/\partial b_1 = \delta_1 = 0.182$. □

Exercise 11.2. Derive the backward recursion $\boldsymbol{\delta}^{[\ell]} = (W^{[\ell+1]\top}\boldsymbol{\delta}^{[\ell+1]}) \odot g'(\boldsymbol{a}^{[\ell]})$ from the chain rule, stating clearly each partial derivative used.

Solution. By definition $\delta_i^{[\ell]} = \partial E/\partial a_i^{[\ell]}$. The loss depends on $a_i^{[\ell]}$ only through the next layer's pre-activations $a_j^{[\ell+1]} = \sum_k W_{jk}^{[\ell+1]} z_k^{[\ell]} + b_j^{[\ell+1]}$, where $z_k^{[\ell]} = g(a_k^{[\ell]})$. The chain rule gives $\delta_i^{[\ell]} = \sum_j (\partial E/\partial a_j^{[\ell+1]})(\partial a_j^{[\ell+1]}/\partial a_i^{[\ell]})$. The first factor is $\delta_j^{[\ell+1]}$. For the second, $a_j^{[\ell+1]}$ depends on $a_i^{[\ell]}$ through $z_i^{[\ell]}$ only, so $\partial a_j^{[\ell+1]}/\partial a_i^{[\ell]} = W_{ji}^{[\ell+1]}\, g'(a_i^{[\ell]})$. Thus $\delta_i^{[\ell]} = g'(a_i^{[\ell]}) \sum_j W_{ji}^{[\ell+1]} \delta_j^{[\ell+1]}$, which in vector form is $(W^{[\ell+1]\top}\boldsymbol{\delta}^{[\ell+1]}) \odot g'(\boldsymbol{a}^{[\ell]})$. □

Exercise 11.3. A loss takes m operations to evaluate, and the network has n weights. Compare the cost of computing the gradient by finite differences versus by backpropagation, and explain why the difference matters for large networks.

Solution. Finite differences perturb each of the n weights and re-evaluate the loss, costing one forward pass (m operations) per weight, so $\mathcal{O}(nm)$ in total. Backpropagation computes all n partial derivatives in a single backward pass costing about $4m$ operations, independent of n. For a network with $n = 10^7$ weights the finite- difference cost is about 10^7 times the forward pass, while backpropagation costs about four times the forward pass; only the latter is feasible. This is why the gradient is no more expensive than the prediction, and why large networks can be trained at all. □

Exercise 11.4. Using the fact that the sigmoid derivative satisfies $\sigma'(a) \le \frac{1}{4}$, bound the factor by which an error signal can shrink as it crosses 20 sigmoid layers (taking the weight factors to be 1 for simplicity), and explain the consequence for training.

Solution. Crossing one sigmoid layer multiplies the error signal by at most $\sigma'(a) \le \frac{1}{4}$ (times the weight factor, here 1). Across 20 layers the signal is multiplied by at most $(\frac{1}{4})^{20} = 4^{-20} = 2^{-40} \approx 9 \times 10^{-13}$. The gradient reaching the first layer is therefore essentially zero, so the early layers receive almost no learning signal and train extremely slowly, the vanishing-gradient problem. Replacing the sigmoid with a ReLU (derivative 1 on its active side) or adding residual connections (an identity path for the gradient) prevents this geometric decay. □

Exercise 11.5. Show that adding the weight-decay penalty $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$ to the loss is equivalent to placing a zero-mean Gaussian prior on the weights and taking the MAP estimate, and identify the prior variance in terms of λ.

Solution. A zero-mean Gaussian prior $p(\boldsymbol{w}) = \mathcal{N}(\boldsymbol{w} \mid \mathbf{0}, \tau^2 I)$ has log-density $\log p(\boldsymbol{w}) = -\frac{1}{2\tau^2}\|\boldsymbol{w}\|^2 + \text{const}$. The MAP objective is the log-likelihood plus the log-prior; writing the loss E as the negative log-likelihood, minimizing $E - \log p(\boldsymbol{w}) = E + \frac{1}{2\tau^2}\|\boldsymbol{w}\|^2 + \text{const}$. This is the loss plus weight decay with $\lambda = 1/\tau^2$, so weight decay is MAP estimation under a Gaussian prior of variance $\tau^2 = 1/\lambda$: a stronger penalty (larger λ) is a tighter prior (smaller variance) pulling the weights more firmly toward zero. □

Exercise 11.6. For a sigmoid output $\hat{y} = \sigma(a)$ and the binary cross-entropy loss $E = -[t \log \hat{y} + (1-t)\log(1-\hat{y})]$, show that the output error signal collapses to $\delta = \partial E/\partial a = \hat{y} - t$.

Solution. By the chain rule $\partial E/\partial a = (\partial E/\partial \hat{y})(\partial \hat{y}/\partial a)$. The loss derivative is $\partial E/\partial \hat{y} = -t/\hat{y} + (1-t)/(1-\hat{y})$, and the sigmoid derivative is $\partial \hat{y}/\partial a = \hat{y}(1-\hat{y})$. Multiplying,

$$\frac{\partial E}{\partial a} = \Big(-\frac{t}{\hat{y}} + \frac{1-t}{1-\hat{y}}\Big)\hat{y}(1-\hat{y}) = -t(1-\hat{y}) + (1-t)\hat{y} = -t + t\hat{y} + \hat{y} - t\hat{y} = \hat{y} - t.$$

The factor $\hat{y}(1-\hat{y})$ from the sigmoid cancels the $1/\hat{y}$ and $1/(1-\hat{y})$ from the loss, which is exactly why the sigmoid and cross-entropy are matched: together they give the clean gradient $\hat{y} - t$ and avoid the saturated-sigmoid slowdown that a squared-error loss on a sigmoid output would suffer. □

Exercise 11.7. A $2-2-1$ network has a tanh hidden layer and a linear output, trained with the squared-error loss $E = \frac{1}{2}(\hat{y} - t)^2$. The parameters are

$$W^{[1]} = \begin{bmatrix} 0.5 & 0.5 \\ 1 & -1 \end{bmatrix},\ \boldsymbol{b}^{[1]} = \mathbf{0}, \qquad W^{[2]} = \begin{bmatrix} 1 & 2 \end{bmatrix},\ b^{[2]} = 0.$$

For the example $\boldsymbol{x} = (1, 0)^\top$, $t = 1$, carry out the forward pass and then the full matrix backward pass, reporting $\boldsymbol{\delta}^{[2]}$, $\boldsymbol{\delta}^{[1]}$, and the gradients $\partial E/\partial W^{[2]}$ and $\partial E/\partial W^{[1]}$. Use $\tanh(0.5) = 0.462$, $\tanh(1) = 0.762$.

Solution. *Forward.* $\boldsymbol{a}^{[1]} = W^{[1]}\boldsymbol{x} = (0.5, 1.0)^\top$; $\boldsymbol{z}^{[1]} = \tanh \boldsymbol{a}^{[1]} = (0.462, 0.762)^\top$; $a^{[2]} = (1)(0.462) + (2)(0.762) = 1.986$; $\hat{y} = 1.986$ (linear); $E = \frac{1}{2}(1.986 - 1)^2 = \frac{1}{2}(0.986)^2 = 0.486$. *Backward.* $\delta^{[2]} = \hat{y} - t = 0.986$. Output gradient $\partial E/\partial W^{[2]} = \delta^{[2]}\boldsymbol{z}^{[1]\top} = 0.985\,(0.462, 0.762) = (0.455, 0.751)$. Send back: $W^{[2]\top}\delta^{[2]} = (0.986, 1.972)^\top$; the tanh derivative is $1 - \tanh^2$, namely $1 - 0.462^2 = 0.786$ and $1 - 0.762^2 = 0.420$, so

$$\boldsymbol{\delta}^{[1]} = (0.985, 1.970)^\top \odot (0.786, 0.420)^\top = (0.775, 0.828)^\top.$$

Hidden gradient $\partial E/\partial W^{[1]} = \boldsymbol{\delta}^{[1]}\boldsymbol{x}^\top = (0.775, 0.828)^\top(1, 0) = \begin{bmatrix} 0.775 & 0 \\ 0.828 & 0 \end{bmatrix}$. The second input is zero, so the second column of $W^{[1]}$ gets no gradient from this example. □

Exercise 11.8. Explain the central finite-difference gradient check $\partial E/\partial w \approx [E(w+\epsilon) - E(w-\epsilon)]/(2\epsilon)$: why is it used, why central rather than one-sided, and why is it too slow to replace backpropagation for training?

Solution. It verifies a hand-written backward pass by comparing each analytic gradient against the definition of the derivative. The central difference is accurate to order ϵ^2, whereas the one-sided difference $[E(w+\epsilon) - E(w)]/\epsilon$ is only order ϵ, so for the same ϵ the central version is far more accurate and catches subtler bugs. It is too slow for training

because it needs a pair of full forward passes *per weight*, costing $\mathcal{O}(nm)$ for n weights and a length-m forward pass, exactly the cost backpropagation avoids by getting all n gradients in one backward pass. Use it once to validate the implementation, then train with backprop. □

Exercise 11.9. A ReLU layer has $n_{\text{in}} = 512$ inputs. Using He initialization $\sigma_w^2 = 2/n_{\text{in}}$, what standard deviation should the initial weights have, and why is the factor 2 (rather than 1) appropriate for ReLU?

Solution. $\sigma_w = \sqrt{2/512} = \sqrt{0.00391} = 0.0625$. The reasoning: a unit summing n_{in} unit-variance inputs through zero-mean weights of variance σ_w^2 has pre-activation variance $n_{\text{in}}\sigma_w^2$, and keeping it at 1 needs $\sigma_w^2 = 1/n_{\text{in}}$ (Xavier). A ReLU sets the negative half of its inputs to zero, roughly halving the variance that propagates, so the weight variance is doubled to $2/n_{\text{in}}$ to compensate, keeping activation variance stable across many layers and preventing the signal from decaying with depth. □

CHAPTER 12

Convolutional Networks

An edge is an edge wherever it appears. A convolutional network is the architecture that takes that sentence seriously.

after Professor Peter Chin, ENGS 106

A fully connected network applied to an image throws away everything we know about images. A modest 224×224 color image has about 150,000 inputs, so a single fully connected hidden layer of a thousand units would need 150 million weights, and worse, it would treat every pixel as unrelated to its neighbors and would have to learn separately that an edge in the top-left corner and an identical edge in the bottom-right are the same kind of thing. Images have structure that begs to be exploited: features are *local* (a small patch determines an edge) and *translation invariant* (a cat is a cat wherever it sits in the frame). The *convolutional network* builds both facts into its architecture through one operation, convolution, applied with shared weights. The payoff is a network with orders of magnitude fewer parameters that generalizes far better on images, and it is the architecture behind the deep-learning revolution in computer vision. This chapter develops the convolution operation, shows by hand how it detects features, explains why weight sharing is the key, and traces the architectures from LeNet to ResNet.

Roadmap

Suggested reading: Goodfellow et al. [13], Chapter 9; Bishop [3], Section 5.5.6.

We cover: the convolution operation, worked with numbers and over channels (Section 12.1); feature detection by hand (Section 12.2); why weight sharing wins (Section 12.3); convolution as a banded Toeplitz matrix (Section 12.4); pooling, padding, stride, and the output-size formula (Section 12.5); the architecture lineage from LeNet to ResNet, traced layer by layer (Section 12.6); backpropagation through a convolution (Section 12.7); receptive-field growth in a deep stack (Section 12.8); and the 1×1 bottleneck (Section 12.9).

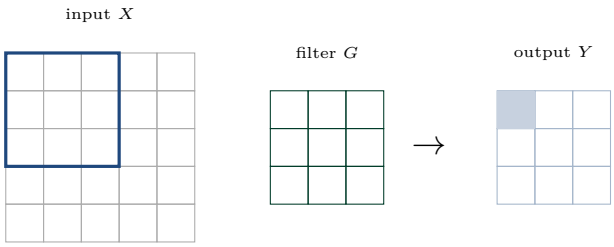


Figure 12.1. A 3×3 filter slides over the input; at each position the overlapping entries are multiplied and summed to produce one output value. A 5×5 input with a 3×3 filter and no padding yields a 3×3 output, one value per filter placement.

12.1 The convolution operation

A convolution slides a small array of weights, the *filter* or *kernel*, across the input, computing at each position a weighted sum of the local patch it covers. In one dimension, with input $\boldsymbol{x}$ and filter $\boldsymbol{k}$ of size K, the output is $y_i = \sum_{u=0}^{K-1} k_u \, x_{i+u}$. In two dimensions, the workhorse for images, a $K \times K$ filter G slides over an image X and produces

$$Y_{ij} = \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} G_{uv} \, X_{i+u,\, j+v}. \tag{12.1}$$

At each location the filter is laid over the image, corresponding entries are multiplied, and the products are summed to give one output number; the filter then shifts by one pixel and the process repeats (Fig. 12.1).

Remark 12.1 *(Convolution or cross-correlation?).* Strictly, the mathematical convolution flips the filter before sliding it, $Y_{ij} = \sum_{u,v} G_{uv} \, X_{i-u,\, j-v}$. The operation in Eq. (12.1), with no flip, is *cross-correlation*. Deep-learning practice, and this chapter, call it "convolution" regardless, because the filter weights are learned and the network is free to learn the flipped filter if it wants one; the distinction makes no difference to what the network can represent. We adopt the no-flip convention used by every deep learning framework.

It is worth doing one convolution entirely by hand, with numbers, before trusting the formula. Figure 12.2 sets up a 4×4 input and a 3×3 filter; we compute all four output values.

Example 12.2 (A 4×4 convolution, every number). Convolve (no padding, stride 1)

$$X = \begin{bmatrix} 3 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 1 & 2 & 3 & 0 \\ 0 & 1 & 2 & 3 \end{bmatrix} \quad \text{with} \quad G = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}.$$

The output is $4 - 3 + 1 = 2$ on each side, so 2×2. The top-left output lays G over the top-left 3×3 block of X, multiplies entry by entry, and sums:

$$Y_{11} = \underbrace{(1{\cdot}3 + 0{\cdot}0 + 1{\cdot}1)}_{\text{row 1}} + \underbrace{(0{\cdot}2 + 1{\cdot}1 + 0{\cdot}0)}_{\text{row 2}} + \underbrace{(1{\cdot}1 + 0{\cdot}2 + 1{\cdot}3)}_{\text{row 3}} = 4 + 1 + 4 = 9.$$

Sliding the filter one column right, then down, then both, gives the other three: $Y_{12} = 2+0+2 = 4$, $Y_{21} = 2+2+2 = 6$, and $Y_{22} = 2 + 3 + 4 = 9$, so

$$Y = \begin{bmatrix} 9 & 4 \\ 6 & 9 \end{bmatrix}.$$

Each number is one placement of the filter: nine multiplies and eight adds, repeated at every position the filter fits.$\diamond$

More than one channel. Real images have channels (red, green, blue), and inner layers have one channel per filter in the layer before. A filter then has a slice for every input channel, and the convolution sums over channels as well as over space,

$$Y_{ij} = \sum_{c=1}^{C_{\text{in}}} \sum_{u=0}^{K-1} \sum_{v=0}^{K-1} G_{c,u,v} \, X_{c,\, i+u,\, j+v} + b, \tag{12.2}$$

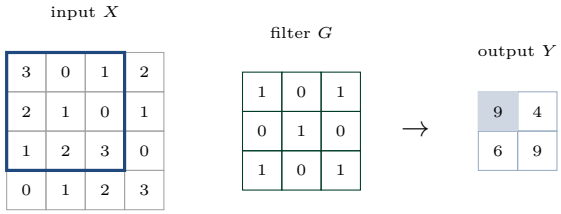


Figure 12.2. The convolution of Example 12.2. The highlighted top-left 3×3 patch of X, multiplied entrywise by G and summed, gives the highlighted output $Y_{11} = 9$. Each of the four output cells is one placement of the filter.

producing one output map. A *bank* of C_{out} such filters produces C_{out} output maps, so the layer maps a C_{in}-channel input to a C_{out}-channel output, and the number of output channels is simply the number of filters. This is why CNNs grow narrower in space but deeper in channels as they go.

Example 12.3 (A two-channel convolution). Take a 3×3 input with two channels and a single 3×3 filter that also has two channels (valid padding, so the output is one number, $b = 0$):

$$X_1 = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix},\ X_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}; \qquad G_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix},\ G_2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

Convolution sums the per-channel responses. Channel 1: the entrywise product $G_1 \odot X_1$ keeps the corners and center, $1+0+0+0+1+0+2+0+1 = 5$. Channel 2: $G_2 \odot X_2$ keeps the edge-midpoints, $0+1+0+1+0+0+0+2+0 = 4$. The output is the sum across channels,

$$Y = \underbrace{5}_{\text{channel 1}} + \underbrace{4}_{\text{channel 2}} = 9.$$

A single filter spanning all input channels produces one output map; stacking C_{out} such filters produces C_{out} maps. The filter here holds $2 \times 3 \times 3 = 18$ weights, one 3×3 slice per input channel. ◊

A convolution also *smooths* or *spreads* a signal, which the classic "box meets box" calculation shows. Convolving the length-3 box $\boldsymbol{k} = (1, 1, 1)$ with the length-3 box $\boldsymbol{x} = (1, 1, 1)$ in full (letting the filter hang off both ends) gives

$$(1, 1, 1) * (1, 1, 1) = (1,\ 2,\ 3,\ 2,\ 1),$$

a discrete *triangle* (Fig. 12.3): the overlap grows to a peak of 3 when the boxes align fully, then falls off symmetrically. The same effect in two dimensions is why a box filter blurs an image, and it is the discrete echo of the fact that convolving two boxes in continuous time gives a triangular pulse.

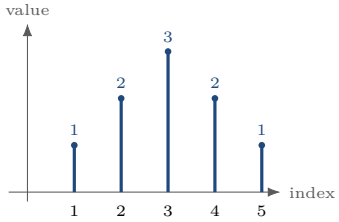


Figure 12.3. The full convolution $(1, 1, 1) * (1, 1, 1) = (1, 2, 3, 2, 1)$: as the two boxes slide across each other the overlap rises to a peak and falls symmetrically, the discrete triangle that explains why a box filter blurs.

12.2 Feature detection, worked by hand

The reason convolution is useful is that a well-chosen filter *detects a feature*. The cleanest case is an edge.

Example 12.4 (Vertical edge detection). Take a 6×6 grayscale image that is bright on the left and dark on the right: every pixel in columns 1 through 3 has value 10, and every pixel in columns 4 through 6 has value 0. There is a single vertical edge down the middle. Convolve it (no padding, stride 1) with the 3×3 vertical-edge filter

$$G = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix},$$

which subtracts the right column of a patch from its left column. The output is $6 - 3 + 1 = 4$ wide and 4 tall. Because every row of the image is identical, every row of the output is identical, so we compute one row by sliding the filter across the four horizontal positions, each covering three columns of the image:

- Columns 1–3 (all 10): each filter row gives $1{\cdot}10 + 0{\cdot}10 - 1{\cdot}10 = 0$, and summing the three rows gives 0.
- Columns 2–4 (values $10, 10, 0$): each row gives $1{\cdot}10 + 0{\cdot}10 - 1{\cdot}0 = 10$, times three rows $= 30$.
- Columns 3–5 (values $10, 0, 0$): each row gives $1{\cdot}10 - 1{\cdot}0 = 10$, times three $= 30$.
- Columns 4–6 (all 0): every term is 0, giving 0.

So each output row is $[\,0,\ 30,\ 30,\ 0\,]$, and the full 4×4 output has its two middle columns lit to 30 while the flat regions are 0. The filter has found the edge: the output is large exactly where the image transitions from bright to dark, and zero where the image is uniform. A horizontal-edge filter (the transpose of G) applied to this same image would give all zeros, because there is no horizontal edge. This is the whole principle: each filter responds to one kind of local pattern, anywhere it occurs. ◊

The historical point is that filters like Sobel's were once designed by hand. A convolutional network instead *learns* its filters from data: the entries of G are weights, fit by backpropagation, so the network discovers for itself which local patterns matter for the task, edges in early layers, then textures, then object parts.

12.3 Why weight sharing wins

The same filter is used at every position, and this *weight sharing* is the source of every advantage a convolutional layer has over a fully connected one.

The first advantage is parameter count. A convolutional layer's parameters are just the filter entries, independent of the image size. A layer with C_{out} filters of size $K \times K$ over an input with C_{in} channels has $K^2\, C_{\text{in}}\, C_{\text{out}} + C_{\text{out}}$ parameters. For a 3×3 filter bank producing 64 channels from a 3-channel image, that is $3^2 \cdot 3 \cdot 64 + 64 = 1{,}792$ weights, whatever the image's height and width. A fully connected layer mapping a $224 \times 224 \times 3$ image to even 64 outputs would need $224 \cdot 224 \cdot 3 \cdot 64 \approx 9.6$ million weights. The convolutional layer is smaller by a factor of thousands, and it does not grow when the image does.

The second advantage is *translation equivariance*. Because the same filter is applied everywhere, shifting the input shifts the output identically: a feature detector trained to find an edge finds it in any location automatically, with no need to relearn it per position. Combined with pooling (next section), this gives approximate translation *invariance*, the property that the network's verdict does not depend on where in the frame the object sits. A fully connected network has to learn this separately for every position, wasting both parameters and data; the convolutional network has it built in. The third advantage, *sparse connectivity*, is that each output depends only on a small local patch of the input, matching the locality of image features and further cutting computation.

12.4 Convolution as matrix multiplication

A convolution is a linear map, so it can be written as a matrix times the (flattened) input, $\boldsymbol{y} = W\boldsymbol{x}$. Seeing that matrix explains exactly what weight sharing and sparse connectivity mean and connects convolution back to the plain linear layers of Chapter 10.

Example 12.5 (A convolution is a banded Toeplitz matrix). Take the 1-D convolution of Exercise 12.1: filter $\boldsymbol{k} = (1, 0, -1)$, input $\boldsymbol{x} \in \mathbb{R}^5$, valid padding, so the output is in $\mathbb{R}^3$. Writing each output as a row, $y_i = x_i - x_{i+2}$, stacks into $\boldsymbol{y} = W\boldsymbol{x}$ with

$$W = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & -1 \end{bmatrix} \in \mathbb{R}^{3\times 5}.$$

The filter $(1, 0, -1)$ appears in every row, shifted one step right each time: this is a *banded Toeplitz* matrix (constant along diagonals). On the input $\boldsymbol{x} = (10, 10, 10, 0, 0)^\top$,

$$W\boldsymbol{x} = \begin{bmatrix} 10 - 10 \\ 10 - 0 \\ 10 - 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \\ 10 \end{bmatrix},$$

the same answer the sliding filter gave. ◊

Two things stand out in that matrix. It is *sparse*: most entries are zero, because each output touches only $K = 3$ inputs (sparse connectivity). And it has only $K = 3$ *distinct* values, repeated down the diagonals, however large the matrix grows (weight sharing). A fully connected layer of the same shape would have $3 \times 5 = 15$ free parameters; the convolution has 3. The same picture holds in two dimensions, where the matrix is doubly-block-Toeplitz, and it explains why backpropagation through a convolution is itself a convolution: the transpose $W^\top$ that the backward pass applies (Eq. (11.4)) is again banded and shares weights, so the gradient computation reuses the same fast machinery as the forward pass.

12.5 Pooling, padding, stride, and output size

A convolutional network interleaves convolution layers with *pooling* layers that shrink the spatial size, summarizing each small region by a single number. *Max pooling* takes the largest value in each region, keeping the strongest feature response and discarding its exact location, which is what builds in translation invariance.

Example 12.6 (Max pooling). Apply 2×2 max pooling with stride 2 to the 4×4 feature map

$$\begin{bmatrix} 12 & 20 & 30 & 0 \\ 8 & 12 & 2 & 0 \\ 34 & 70 & 37 & 4 \\ 112 & 100 & 25 & 12 \end{bmatrix}.$$

The map is divided into four non-overlapping 2×2 blocks, and each is replaced by its maximum. The top-left block $\{12, 20, 8, 12\}$ has max 20; the top-right $\{30, 0, 2, 0\}$ has max 30; the bottom-left $\{34, 70, 112, 100\}$ has max 112; the bottom-right $\{37, 4, 25, 12\}$ has max 37. The pooled output is the 2×2 map $\left[\begin{smallmatrix} 20 & 30 \\ 112 & 37 \end{smallmatrix}\right]$, halving each spatial dimension while keeping the strongest response in each region. *Average pooling* would instead report each block's mean. ◊

The spatial size after a convolution or pooling layer is fixed by three knobs: the filter size K, the zero-*padding* P added around the border, and the *stride* S by which the filter hops. Counting the number of filter placements along one dimension of a length-n input gives the output-size formula.

Output size

A length-n input convolved with a length-K filter, with padding P and stride S, produces an output of length

$$n_{\text{out}} = \left\lfloor \frac{n - K + 2P}{S} \right\rfloor + 1. \tag{12.3}$$

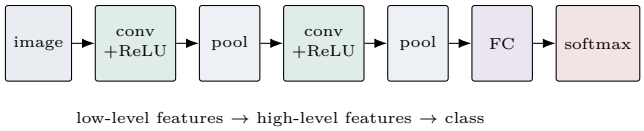


Figure 12.4. The canonical pipeline: alternating convolution-plus- ReLU and pooling layers build a hierarchy of features from edges to object parts, then fully connected layers and a softmax produce the class. Each stage shrinks space and grows the number of feature channels.

Two padding choices are standard. *Valid* padding uses $P = 0$, so the output shrinks to $n - K + 1$ (with stride 1): a 6×6 image with a 3×3 filter gives $6 - 3 + 1 = 4$, as in Example 12.4. *Same* padding uses $P = (K - 1)/2$, chosen so the output matches the input size: for $K = 3$, $P = 1$ gives $n_{\text{out}} = n$. A stride $S > 1$ downsamples: a 6×6 image, 3×3 filter, no padding, stride 2 gives $\lfloor (6 - 3)/2 \rfloor + 1 = \lfloor 1.5 \rfloor + 1 = 2$. The *receptive field* of a unit, the region of the original image that can influence it, grows as layers stack, so deep units see large parts of the image even though each filter is small.

12.6 Architectures: from LeNet to ResNet

A convolutional network stacks these pieces: convolution and ReLU to extract features, pooling to shrink and build invariance, repeated to build a hierarchy from edges to object parts, and finally a few fully connected layers and a softmax to classify (Fig. 12.4). The history of these architectures is the history of modern computer vision.

LeNet [25], from the 1990s, was the first successful convolutional network, built to read handwritten digits: two convolution-and-pooling stages followed by fully connected layers, small by today's standards but the template for everything since. It is worth tracing all the way through, because following the shape and the parameter count layer by layer is the surest way to understand a convolutional network. Figure 12.5 shows the shapes shrinking in space while growing in channels, and Table 12.1 counts the parameters at each step, using the output-size formula (12.3) and the channel parameter count $(K^2 C_{\text{in}} + 1)C_{\text{out}}$ from Section 12.3.

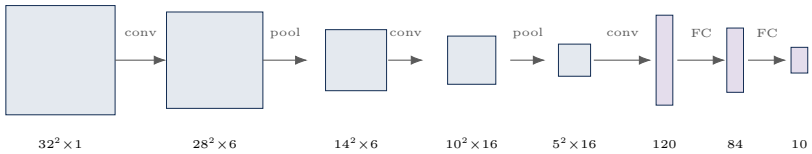


Figure 12.5. The feature maps of LeNet-5 shrink in spatial size ($32 \to 28 \to 14 \to 10 \to 5$) while the channel count grows ($1 \to 6 \to 16$), then flatten into fully connected layers ($120 \to 84 \to 10$). Convolution and pooling alternate; the final layers are dense.

layer	operation	output shape	parameters
input	n/a	$32 \times 32 \times 1$	0
C1	conv 5×5, 6 filters, valid	$28 \times 28 \times 6$	$(25{\cdot}1{+}1){\cdot}6 = 156$
S2	2×2 pool, stride 2	$14 \times 14 \times 6$	0
C3	conv 5×5, 16 filters, valid	$10 \times 10 \times 16$	$(25{\cdot}6{+}1){\cdot}16 = 2416$
S4	2×2 pool, stride 2	$5 \times 5 \times 16$	0
C5	conv 5×5, 120 filters	$1 \times 1 \times 120$	$(25{\cdot}16{+}1){\cdot}120 = 48120$
F6	fully connected	84	$(120{+}1){\cdot}84 = 10164$
out	fully connected	10	$(84{+}1){\cdot}10 = 850$
		total	**61,706**

Table 12.1. Layer-by-layer shapes and parameter counts for LeNet-5 on a 32×32 input. Pooling has no parameters; the convolution layers are cheap and most weights sit in the dense layers $C5$ and $F6$. (The original network wired $C3$ partially, trimming its 2416 to 1516 and the total to about 60,000; we count full connectivity here.)

Network	Year	Key idea	Scale
LeNet	1998	first working CNN (digits)	small
AlexNet	2012	ReLU, GPUs, dropout; won ImageNet	∼60M params, top-5 15.3%
VGG	2014	deep stacks of 3×3 convs	∼138M params
GoogLeNet	2014	Inception, 1×1 bottlenecks	deeper, fewer params
ResNet	2015	residual connections, 100+ layers	solves vanishing gradient

Table 12.2. Each generation went deeper by fixing the obstacle the last one hit, culminating in the residual connection that made very deep networks trainable. Years mark each architecture's first public appearance; the bibliography cites the archival publications, dated a year later for VGG, GoogLeNet, and ResNet.

The lesson of the table outlives LeNet: convolution layers are parameter-cheap because of weight sharing, and the dense layers dominate the count, which is exactly why later designs (Table 12.2) work to shrink or remove the fully connected layers.

AlexNet [23] launched the deep-learning era by winning the 2012 ImageNet competition. It had five convolution layers and three fully connected layers, about 60 million parameters, and it cut the top-five error rate on ImageNet to 15.3% far ahead of the runner-up's 26.2%, a margin so large it ended the debate about whether deep networks worked. Its innovations were practical: ReLU activations (for the gradient-flow reasons of Chapter 11), training on GPUs, dropout in the fully connected layers, and data augmentation.

VGG [36] showed that depth and uniformity help: it used only 3×3 convolutions stacked deep, with 2×2 max pooling, doubling the number of channels after each pooling stage. Its sixteen-layer version has about 138 million parameters, most of them in the fully connected layers.

GoogLeNet [38] introduced the Inception module, which runs several filter sizes in parallel and uses 1×1 convolutions as cheap channel-mixing bottlenecks to control the parameter count; the result was far deeper than VGG with far fewer parameters.

ResNet [16] solved the problem that had capped depth: the vanishing gradient of Chapter 11. Its *residual connection* adds a layer's input to its output, $\boldsymbol{z}^{[\ell+1]} = \boldsymbol{z}^{[\ell]} + F(\boldsymbol{z}^{[\ell]})$, so the layer learns only a *residual* correction F and the gradient has an identity path straight back through the addition. With this fix, networks of over a hundred layers became trainable, and residual connections are now standard in architectures far beyond vision, including the Transformers of Chapter 13. The progression, summarized in Table 12.2, is one of steadily greater depth made possible by architectural fixes to the training problems each previous generation hit.

12.7 Backpropagation through a convolution

Section 12.4 noted that the backward pass through a convolution is itself a convolution, because the matrix W is banded and shares weights. It is worth seeing this in full on small numbers, because two distinct gradients come out of one convolution layer: the gradient with respect to the *filter*, which the optimizer uses to update the weights, and the gradient with respect to the *input*, which the chain rule passes back to the layer below. Both are convolutions, but of different things.

Take the 1-D layer $y_i = w_1 x_i + w_2 x_{i+1}$, the valid convolution of the length-3 input $\boldsymbol{x} = (1, 2, 3)$ with the length-2 filter $\boldsymbol{w} = (1, -1)$. The output has length $3 - 2 + 1 = 2$:

$$\begin{aligned} y_1 &= w_1 x_1 + w_2 x_2 = (1)(1) + (-1)(2) = -1, \\ y_2 &= w_1 x_2 + w_2 x_3 = (1)(2) + (-1)(3) = -1. \end{aligned} \tag{12.4}$$

Suppose the rest of the network hands back the upstream gradient $\delta = \partial L / \partial \boldsymbol{y} = (\delta_1, \delta_2) = (1, 2)$, one number per output. Everything the layer needs follows from the chain rule applied to the two lines of Eq. (12.4).

Example 12.7 (Both convolution gradients, every number). Gradient with respect to the filter. Each weight w_u appears in *every* output, so its gradient sums over all output positions, $\partial L / \partial w_u = \sum_i \delta_i \, \partial y_i / \partial w_u$. From Eq. (12.4), $\partial y_i / \partial w_1 = x_i$ and $\partial y_i / \partial w_2 = x_{i+1}$, so

$$\begin{aligned} \frac{\partial L}{\partial w_1} &= \delta_1 x_1 + \delta_2 x_2 = (1)(1) + (2)(2) = 5, \\ \frac{\partial L}{\partial w_2} &= \delta_1 x_2 + \delta_2 x_3 = (1)(2) + (2)(3) = 8. \end{aligned}$$

So $\partial L / \partial \boldsymbol{w} = (5, 8)$. Look at what produced it: slide the upstream gradient $\delta = (1, 2)$ across the input $\boldsymbol{x} = (1, 2, 3)$ as if δ were the filter. Position 1 covers $(1, 2)$ and gives $(1)(1) + (2)(2) = 5$; position 2 covers $(2, 3)$ and gives $(1)(2) + (2)(3) = 8$. The filter gradient is the convolution of the input with the upstream gradient, exactly the same sliding-and-summing the forward pass does, only with δ in the filter's role.

Gradient with respect to the input. Each input x_j feeds the outputs whose window covers it, so $\partial L / \partial x_j = \sum_i \delta_i \, \partial y_i / \partial x_j$. Reading the two lines of Eq. (12.4), x_1 enters only y_1 (with weight w_1), x_2 enters both y_1 (weight w_2) and y_2 (weight w_1), and x_3 enters only y_2 (weight w_2):

$$\begin{aligned} \frac{\partial L}{\partial x_1} &= \delta_1 w_1 = (1)(1) = 1, \\ \frac{\partial L}{\partial x_2} &= \delta_1 w_2 + \delta_2 w_1 = (1)(-1) + (2)(1) = 1, \\ \frac{\partial L}{\partial x_3} &= \delta_2 w_2 = (2)(-1) = -2, \end{aligned}$$

so $\partial L / \partial \boldsymbol{x} = (1, 1, -2)$, one number per input, the same length as $\boldsymbol{x}$. This too is a convolution. Pad the upstream gradient with $K - 1 = 1$ zero on each side, $\delta_{\text{pad}} = (0, 1, 2, 0)$, flip the filter to $\boldsymbol{w}^{\text{flip}} = (-1, 1)$, and slide it across:

$$\begin{aligned} &\text{covers } (0, 1) : (-1)(0) + (1)(1) = 1, \\ &\text{covers } (1, 2) : (-1)(1) + (1)(2) = 1, \\ &\text{covers } (2, 0) : (-1)(2) + (1)(0) = -2, \end{aligned}$$

reproducing $(1, 1, -2)$. The input gradient is a *full* convolution of the upstream gradient with the flipped filter: full because the padding lets the window hang off both ends (as the box-meets-box calculation did in Fig. 12.3), and flipped because running the chain rule backward through the window reverses the weight order. ◊

The two convolution gradients

For a convolution layer $\boldsymbol{y} = \boldsymbol{w} * \boldsymbol{x}$ with upstream gradient $\delta = \partial L/\partial \boldsymbol{y}$,

$$\frac{\partial L}{\partial \boldsymbol{w}} = \boldsymbol{x} * \delta \quad \text{(valid)}, \qquad \frac{\partial L}{\partial \boldsymbol{x}} = \delta * \boldsymbol{w}^{\text{flip}} \quad \text{(full)}. \tag{12.5}$$

The filter gradient convolves the input with the upstream gradient and has the filter's shape; the input gradient convolves the upstream gradient (zero-padded) with the flipped filter and has the input's shape. Both reuse the forward pass's sliding machinery, which is why a convolution layer trains as cheaply as it runs, and why the shared weights accumulate one contribution from every spatial position.

Two consequences of Eq. (12.5) are worth stating. First, the filter gradient is a single small array no matter how large the feature map is, because the sum over output positions collapses every position's contribution onto the same K weights: this is weight sharing seen from the backward pass, the gradient counterpart of the forward pass's parameter saving. Second, the input gradient has the same shape as the input, so it slots straight into the layer below as its upstream gradient, and the whole backward sweep is convolutions all the way down.

12.8 Receptive fields in a deep stack

Section 12.5 introduced the *receptive field*, the patch of the original image that can influence one unit, and Exercise 12.7 showed that two stacked 3×3 convolutions reach a 5×5 patch. The growth is worth tracking through a deeper stack, because it is the reason a network of small filters can eventually see a whole object: every layer widens the window by a fixed amount, and stride multiplies the steps.

Let r_ℓ be the receptive field (side length) of a unit after layer ℓ, let K_ℓ be that layer's filter size, and let S_ℓ be its stride. Adding one more layer extends the reach by $(K_\ell - 1)$ units of the layer below, but each of those units already stands for $\prod_{i<\ell} S_i$ pixels of the input (the strides compound), so

$$r_0 = 1, \qquad r_\ell = r_{\ell-1} + (K_\ell - 1) \prod_{i=1}^{\ell-1} S_i. \tag{12.6}$$

With every stride equal to 1 the product is 1 and each layer simply adds $K_\ell - 1$, so a stack of L filters of size K reaches $1 + L(K-1)$.

Example 12.8 (Receptive field of three 3×3 layers). Stack three 3×3 convolutions, all stride 1, no pooling. Apply Eq. (12.6) with $K_\ell = 3$ and every $S_i = 1$, so the product is always 1 and each layer adds $K - 1 = 2$:

$$\begin{aligned} r_1 &= 1 + (3-1)(1) = 3, \\ r_2 &= 3 + (3-1)(1) = 5, \\ r_3 &= 5 + (3-1)(1) = 7. \end{aligned}$$

A unit three layers deep sees a 7×7 patch of the input, matching the closed form $1 + L(K-1) = 1 + 3(2) = 7$. The first layer sees 3×3, as it must, and each further 3×3 layer widens the window by two, one pixel on each side. This is the VGG bargain of Exercise 12.7 extended: three 3×3 layers reach the same 7×7 field as one 7×7 filter, at $3 \times 9 = 27$ weights against 49, with two extra nonlinearities folded in. ◊

Stride changes the picture sharply, because a stride-S layer makes every unit above it stand for S pixels below, and that factor multiplies the reach of all later layers through the product in Eq. (12.6). A single pooling layer therefore enlarges the receptive field of everything downstream, which is the other half of why pooling is in the architecture: it shrinks the map (Section 12.5) and, at the same time, lets later filters see more of the image per step. Exercise 12.9 works a stack with a pooling layer in the middle so the stride factor appears explicitly.

Remark 12.9 *(Receptive field versus effective receptive field).* The count in Eq. (12.6) is the *theoretical* receptive field, the set of input pixels that *can* influence a unit. In a trained network the influence is far from uniform across that patch: pixels near the center pass through many more paths than pixels at the rim, so the *effective* receptive field, weighted by how much each pixel actually moves the unit, is smaller and roughly bell-shaped, often a fraction of the theoretical size. The theoretical field is the ceiling; the effective field is what the unit truly responds to. This is why very deep stacks still benefit from explicit downsampling rather than relying on depth alone to cover the image.

12.9 The 1×1 convolution bottleneck

Section 12.6 credited GoogLeNet with using 1×1 convolutions as cheap channel-mixing bottlenecks. A 1×1 convolution has no spatial extent: it touches one pixel at a time and mixes only across channels, so a layer of C_{out} filters of size 1×1 over a C_{in}-channel input is a per-pixel linear map from C_{in} channels to C_{out}, with $C_{\text{in}}\,C_{\text{out}}$ weights (plus C_{out} biases). Choosing $C_{\text{out}} < C_{\text{in}}$ *reduces* the channel count, and doing that before an expensive spatial convolution is what saves the parameters.

Example 12.10 (Parameters saved by a bottleneck). A feature map with $C_{\text{in}} = 256$ channels is to be sent through a 5×5 convolution producing $C_{\text{out}} = 64$ channels. Count the weights two ways (biases, 64 in each case, are negligible and omitted).

Direct. One 5×5 filter spans all 256 input channels, so the bank costs

$$K^2\,C_{\text{in}}\,C_{\text{out}} = 5^2 \cdot 256 \cdot 64 = 25 \cdot 16{,}384 = 409{,}600 \text{ weights.}$$

With a bottleneck. First a 1×1 convolution squeezes 256 channels down to a thin 32, then the 5×5 convolution acts on those 32 channels and produces the same 64:

$$\begin{aligned} 1 \times 1\ (256 \to 32): &\quad 1^2 \cdot 256 \cdot 32 = 8{,}192, \\ 5 \times 5\ (32 \to 64): &\quad 5^2 \cdot 32 \cdot 64 = 25 \cdot 2{,}048 = 51{,}200, \end{aligned}$$

for a total of $8{,}192 + 51{,}200 = 59{,}392$ weights. The bottleneck does the same job (256-channel input, 64-channel output, 5×5 spatial reach) with 59,392 weights against 409,600, a factor of roughly 6.9 fewer. The squeeze works because the costly 5×5 filters now span 32 channels instead of 256, and the 1×1 that does the squeezing is itself cheap. ◊

This is the lesson Table 12.1 foreshadowed, applied to channels rather than to the dense layers: a great deal of a convolutional network's cost is spent multiplying wide channel stacks against large spatial filters, and a 1×1 convolution buys a way to thin the stack first. Inception modules place such bottlenecks before each expensive branch and concatenate the results; ResNet's deeper variants wrap each 3×3 convolution between a 1×1 that reduces channels and a 1×1 that restores them, the "bottleneck block" that makes hundred-layer networks affordable.

Remark 12.11 *(Batch normalization inside a convolution).* Deep convolutional networks are almost always trained with *batch normalization*, inserted between the convolution and its ReLU. For a convolution layer it normalizes each *channel* separately: over a training batch it computes that channel's mean and variance across both the batch and all spatial positions, subtracts the mean, divides by the standard deviation, then rescales by two learned per-channel parameters γ and β. Treating a whole feature map as one unit (rather than normalizing each pixel independently) keeps the statistics consistent with weight sharing: the same filter runs at every position, so its outputs should be normalized by one shared statistic. The effect is to keep each layer's inputs in a stable range as training shifts the weights below it, which lets larger learning rates be used and speeds up training, the convolutional echo of the gradient-flow concerns of Chapter 11.

Notes and References

The convolution operation, weight sharing, pooling, and the output-size formula follow Chapter 9 of Goodfellow et al. [13] and Section 5.5.6 of Bishop [3]. The distinction between convolution and cross-correlation of Remark 12.1 is discussed in Goodfellow et al. [13]. The architectures are LeNet [25], AlexNet [23], VGG [36], GoogLeNet [38], and ResNet [16]; the AlexNet ImageNet figures and parameter counts are from the original paper. The residual connection of ResNet is the architectural answer to the vanishing-gradient problem of Chapter 11 and reappears in the Transformer of Chapter 13. Convolutional networks were trained by the backpropagation of Chapter 11, specialized so that the shared filter weights accumulate gradient contributions from every spatial position; the explicit gradient calculation of Section 12.7, in which both gradients are themselves convolutions, follows Chapter 9 of Goodfellow et al. [13]. The receptive-field recurrence of Section 12.8 is the standard bookkeeping behind the VGG design, and the distinction between theoretical and effective receptive fields in Remark 12.9 is a later refinement. The 1×1 bottleneck of Section 12.9 is the parameter-saving device of GoogLeNet [38] and the ResNet bottleneck block [16]; batch normalization (Remark 12.11) is the now-standard training aid layered between convolution and activation.

Exercises

Exercise 12.1. Convolve the 1×5 signal $x = [10, 10, 10, 0, 0]$ with the 1×3 filter $k = [1, 0, -1]$ (no padding, stride 1), and explain what the output detects.

Solution. The output has length $5-3+1 = 3$. Sliding the filter: position 1 covers $[10, 10, 10]$, giving $1{\cdot}10+0{\cdot}10-1{\cdot}10 = 0$; position 2 covers $[10, 10, 0]$, giving $1{\cdot}10 + 0{\cdot}10 - 1{\cdot}0 = 10$; position 3 covers $[10, 0, 0]$, giving $1{\cdot}10 - 1{\cdot}0 = 10$. The output is $[0, 10, 10]$, large where the signal steps down from 10 to 0 and zero in the flat regions: the filter detects the edge, the location of change, rather than the absolute level. □

Exercise 12.2. Apply 2×2 max pooling with stride 2 to $\left[\begin{smallmatrix} 1 & 3 & 2 & 4 \\ 5 & 6 & 1 & 2 \\ 7 & 2 & 3 & 0 \\ 1 & 0 & 4 & 8 \end{smallmatrix}\right]$, and state the output size from Eq. (12.3).

Solution. The four 2×2 blocks have maxima: top-left $\{1, 3, 5, 6\} \to 6$; top-right $\{2, 4, 1, 2\} \to 4$; bottom-left $\{7, 2, 1, 0\} \to 7$; bottom-right $\{3, 0, 4, 8\} \to 8$. The pooled map is $\left[\begin{smallmatrix} 6 & 4 \\ 7 & 8 \end{smallmatrix}\right]$. By the output-size formula with $n = 4$, $K = 2$, $P = 0$, $S = 2$: $n_{\text{out}} = \lfloor (4 - 2 + 0)/2 \rfloor + 1 = \lfloor 1 \rfloor + 1 = 2$, a 2×2 output, as found. □

Exercise 12.3. An input image is 32×32. Give the output spatial size after (a) a 5×5 convolution with no padding and stride 1; (b) the same with "same" padding; (c) a 3×3 convolution with stride 2 and no padding.

Solution. Use $n_{\text{out}} = \lfloor (n - K + 2P)/S \rfloor + 1$ with $n = 32$. (a) $K = 5$, $P = 0$, $S = 1$: $\lfloor (32 - 5)/1 \rfloor + 1 = 28$, so 28×28. (b) Same padding sets $P = (K - 1)/2 = 2$: $\lfloor (32 - 5 + 4)/1 \rfloor + 1 = 32$, so 32×32, unchanged. (c) $K = 3$, $P = 0$, $S = 2$: $\lfloor (32 - 3)/2 \rfloor + 1 = \lfloor 14.5 \rfloor + 1 = 15$, so 15×15. □

Exercise 12.4. A convolution layer has 32 filters of size 5×5 acting on an input with 3 channels. How many parameters does it have, including biases? Compare with a fully connected layer mapping a $32 \times 32 \times 3$ input to the same number of output units as the convolution produces over a 32×32 "same"-padded output.

Solution. The convolution has $K^2 C_{\text{in}} C_{\text{out}} + C_{\text{out}} = 5^2 \cdot 3 \cdot 32 + 32 = 2400 + 32 = 2432$ parameters, independent of image size. A "same"-padded output is $32 \times 32 \times 32 = 32{,}768$ units; a fully connected layer producing that many outputs from the $32 \cdot 32 \cdot 3 = 3072$ inputs would need $3072 \cdot 32768 \approx 1.0 \times 10^8$ weights. The convolution uses about 2,400 weights against the fully connected layer's hundred million, a factor of roughly 40,000, the saving that weight sharing buys. □

Exercise 12.5. Explain how a residual connection $\boldsymbol{z}^{[\ell+1]} = \boldsymbol{z}^{[\ell]} + F(\boldsymbol{z}^{[\ell]})$ helps with the vanishing-gradient problem of Chapter 11.

Solution. Differentiating $\boldsymbol{z}^{[\ell+1]} = \boldsymbol{z}^{[\ell]} + F(\boldsymbol{z}^{[\ell]})$ with respect to $\boldsymbol{z}^{[\ell]}$ gives $I + \partial F/\partial \boldsymbol{z}^{[\ell]}$: the Jacobian has an identity term added to the usual layer Jacobian. In backpropagation the gradient is multiplied by this Jacobian, and the identity term provides a path along which the gradient passes *unattenuated*, so even if $\partial F/\partial \boldsymbol{z}^{[\ell]}$ is small (the situation that causes vanishing gradients), the gradient still flows back through the I term. The product of Jacobians across many layers no longer decays geometrically toward zero, which is why residual connections let networks of a hundred or more layers be trained. □

Exercise 12.6. Write the 1-D valid convolution of the filter $\boldsymbol{k} = (2, -1)$ with an input $\boldsymbol{x} \in \mathbb{R}^4$ as a matrix $\boldsymbol{y} = W\boldsymbol{x}$. Give W explicitly and state how many distinct values it contains.

Solution. Valid convolution of a length-4 input with a length-2 filter gives length $4 - 2 + 1 = 3$, with $y_i = 2x_i - x_{i+1}$. Stacking the rows,

$$W = \begin{bmatrix} 2 & -1 & 0 & 0 \\ 0 & 2 & -1 & 0 \\ 0 & 0 & 2 & -1 \end{bmatrix} \in \mathbb{R}^{3 \times 4},$$

a banded Toeplitz matrix. It contains only the $K = 2$ distinct filter values 2 and -1 (besides the structural zeros), repeated along the diagonals: weight sharing in matrix form. □

Exercise 12.7. Two 3×3 convolution layers (stride 1, no pooling) are stacked. What is the receptive field of a unit in the second layer, that is, how large a patch of the original input can influence it? Compare the parameter count of two stacked 3×3 filters with one 5×5 filter (single channel), and state the practical lesson.

Solution. A unit in the first conv layer sees a 3×3 input patch. A unit in the second layer sees a 3×3 patch of the first layer, and those nine units' receptive fields, each 3×3 and offset by one, together span a 5×5 region of the input: the receptive field is 5×5. Parameter-wise, two 3×3 filters cost $2 \times 9 = 18$ weights, against 25 for one 5×5 filter, while covering the same 5×5 receptive field, and the two-layer stack adds an extra nonlinearity in between. The lesson, which is the VGG design principle, is that deep stacks of small 3×3 convolutions match the receptive field of larger filters with fewer parameters and more nonlinearity. □

Exercise 12.8. A 1-D convolution layer computes the valid convolution $y_i = w_1 x_i + w_2 x_{i+1}$ of the input $\boldsymbol{x} = (2, 1, 3)$ with the filter $\boldsymbol{w} = (1, 2)$. The upstream gradient is $\delta = \partial L/\partial \boldsymbol{y} = (1, -1)$. Compute the forward output $\boldsymbol{y}$, the filter gradient $\partial L/\partial \boldsymbol{w}$, and the input gradient $\partial L/\partial \boldsymbol{x}$, and identify each gradient as a convolution.

Solution. Forward: $y_1 = w_1 x_1 + w_2 x_2 = (1)(2) + (2)(1) = 4$ and $y_2 = w_1 x_2 + w_2 x_3 = (1)(1) + (2)(3) = 7$, so $\boldsymbol{y} = (4, 7)$. Filter gradient: $\partial L/\partial w_u = \sum_i \delta_i \, \partial y_i/\partial w_u$ with $\partial y_i/\partial w_1 = x_i$ and $\partial y_i/\partial w_2 = x_{i+1}$, giving $\partial L/\partial w_1 = \delta_1 x_1 + \delta_2 x_2 = (1)(2) + (-1)(1) = 1$ and $\partial L/\partial w_2 = \delta_1 x_2 + \delta_2 x_3 = (1)(1) + (-1)(3) = -2$, so $\partial L/\partial \boldsymbol{w} = (1, -2)$, the valid convolution of the input $\boldsymbol{x}$ with the upstream gradient δ. Input gradient: $\partial L/\partial x_1 = \delta_1 w_1 = (1)(1) = 1$; $\partial L/\partial x_2 = \delta_1 w_2 + \delta_2 w_1 = (1)(2) + (-1)(1) = 1$; $\partial L/\partial x_3 = \delta_2 w_2 = (-1)(2) = -2$. So $\partial L/\partial \boldsymbol{x} = (1, 1, -2)$, the full convolution of δ (zero-padded to $(0, 1, -1, 0)$) with the flipped filter $\boldsymbol{w}^{\text{flip}} = (2, 1)$: the windows $(0, 1)$, $(1, -1)$, $(-1, 0)$ give $(2)(0)+(1)(1) = 1$, $(2)(1)+(1)(-1) = 1$, $(2)(-1)+(1)(0) = -2$, matching Eq. (12.5). □

Exercise 12.9. A stack applies a 3×3 convolution (stride 1), then 2×2 max pooling (stride 2), then another 3×3 convolution (stride 1). Using the receptive-field recurrence (12.6), give the receptive field after each layer, and explain why the pooling layer enlarges the field of the final convolution beyond what a third stride-1 convolution would.

Solution. Apply $r_\ell = r_{\ell-1} + (K_\ell - 1) \prod_{i<\ell} S_i$ with $r_0 = 1$. First convolution ($K = 3$, $S = 1$, product of earlier strides $= 1$): $r_1 = 1 + (3 - 1)(1) = 3$. Pooling ($K = 2$, $S = 2$, earlier strides still 1): $r_2 = 3 + (2 - 1)(1) = 4$, and the running stride product becomes $1 \cdot 2 = 2$. Second convolution ($K = 3$, $S = 1$, product of earlier strides $= 2$): $r_3 = 4 + (3 - 1)(2) = 8$. So the fields are 3×3, 4×4, 8×8. The final convolution adds $(3 - 1)(2) = 4$ rather than the 2 a stride-1-only stack would add, because the pooling stride of 2 makes each unit it feeds stand for two input pixels, doubling the reach of every layer after it. The pooling layer thus shrinks the map and widens later receptive fields at once. □

Exercise 12.10. A feature map with 192 channels is sent through a 3×3 convolution producing 128 channels. Count the weights (a) directly, and (b) with a 1×1 bottleneck that first reduces 192 channels to 48. State the saving factor.

Solution. (a) Direct: $K^2 \, C_{\text{in}} \, C_{\text{out}} = 3^2 \cdot 192 \cdot 128 = 9 \cdot 24{,}576 = 221{,}184$ weights. (b) Bottleneck: the 1×1 reduction costs $1^2 \cdot 192 \cdot 48 = 9{,}216$ weights, and the 3×3 convolution on the thinned 48 channels costs $3^2 \cdot 48 \cdot 128 = 9 \cdot 6{,}144 = 55{,}296$ weights, for a total of $9{,}216 + 55{,}296 = 64{,}512$. The saving factor is $221{,}184/64{,}512 \approx 3.4$: the costly 3×3 filters now span 48 channels instead of 192, and the 1×1 squeeze that thins them is itself cheap. □

Exercise 12.11. In the LeNet trace of Table 12.1, verify the output shape and parameter count of layer $C3$: a 5×5 convolution with 16 filters and valid padding applied to the $14 \times 14 \times 6$ output of $S2$.

Solution. Spatial size by (12.3) with $n = 14$, $K = 5$, $P = 0$, $S = 1$: $n_{\text{out}} = \lfloor (14 - 5)/1 \rfloor + 1 = 10$, so 10×10. With 16 filters the output is $10 \times 10 \times 16$. Each filter spans all 6 input channels, so it has $5 \times 5 \times 6 = 150$ weights plus 1 bias, 151 parameters; sixteen filters give $151 \times 16 = 2416$ parameters, matching the table. □

CHAPTER 13

Sequence Models and Transformers

For a sentence, the order is the meaning. Ignore the order and a sentence collapses into a bag of words.

after Professor Peter Chin, ENGS 106

The networks of the last three chapters take a fixed-size input and produce a fixed-size output. Much of the most interesting data is not like that. A sentence, a stretch of audio, a strand of DNA, a stock price over time, all are *sequences*: ordered, variable in length, and meaningful only because of their order. "Batman and Robin went to the coop" and "coop the to went Robin and Batman" contain the same words and almost nothing of the same meaning. This chapter develops the models built for sequences. The recurrent network processes a sequence one element at a time, carrying a hidden state that summarizes the past; its gated descendants, the LSTM and GRU, fix the recurrent network's difficulty with long-range memory; and the attention mechanism, which lets a model look directly at any part of the sequence, culminates in the Transformer, the architecture behind modern language models. The chapter is the modern frontier of the course, and it rests on everything before it: the chain rule of Chapter 11, the softmax of Chapter 3, and the residual connections of Chapter 12.

Roadmap

Suggested reading: Goodfellow et al. [13], Chapter 10; Vaswani et al. [42] for the Transformer.

We cover: recurrent networks and weight sharing across time (Section 13.1); backpropagation through time and its gradient problem (Section 13.2); the gated units LSTM and GRU (Section 13.3); language models and machine translation (Section 13.4); attention (Section 13.5); and self-attention and the Transformer (Section 13.6).

13.1 Recurrent neural networks

A recurrent neural network (RNN) reads a sequence $\boldsymbol{x}^{\langle 1\rangle}, \dots, \boldsymbol{x}^{\langle T\rangle}$ one element at a time, maintaining a *hidden state* $\boldsymbol{a}^{\langle t\rangle}$ that acts as a memory of everything seen so far. At each step it combines the new input with the previous state to form the new state, and optionally emits an output:

$$\boldsymbol{a}^{\langle t\rangle} = g\big(W_{aa}\,\boldsymbol{a}^{\langle t-1\rangle} + W_{ax}\,\boldsymbol{x}^{\langle t\rangle} + \boldsymbol{b}_a\big), \tag{13.1}$$

$$\hat{\boldsymbol{y}}^{\langle t\rangle} = g_y\big(W_{ya}\,\boldsymbol{a}^{\langle t\rangle} + \boldsymbol{b}_y\big), \tag{13.2}$$

starting from $\boldsymbol{a}^{\langle 0\rangle} = \mathbf{0}$. The activation g is usually tanh, whose output in $(-1, 1)$ keeps the state bounded and signed, a memory that can hold both positive and negative evidence. The single most important feature is that the *same* weight matrices W_{aa}, W_{ax}, W_{ya} are used at every time step. This weight sharing across time is the temporal analogue of the spatial weight sharing of Chapter 12: it lets one set of parameters process a sequence of *any* length, and it builds in the assumption that the rule for updating memory is the same at every position, just as a convolution applies the same filter everywhere. Figure 13.1 shows the network unrolled in time.

Running two steps of the recurrence on numbers shows how the hidden state threads the past into the present.

Example 13.1 (An RNN forward pass, two steps). Take a hidden state of size two and a scalar input, with tanh activation and

$$W_{aa} = \begin{bmatrix} 0.1 & -0.2 \\ 0.3 & 0.1 \end{bmatrix}, \quad W_{ax} = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix}, \quad \boldsymbol{b}_a = \mathbf{0}, \quad \boldsymbol{a}^{\langle 0\rangle} = \mathbf{0},$$

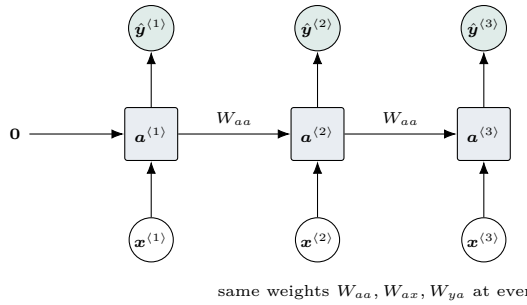


Figure 13.1. The RNN carries a hidden state forward, $\boldsymbol{a}^{\langle t\rangle} = g(W_{aa}\boldsymbol{a}^{\langle t-1\rangle} + W_{ax}\boldsymbol{x}^{\langle t\rangle} + \boldsymbol{b}_a)$, reusing the same weights at every step. The unrolled view shows it as a deep network whose depth is the sequence length.

on the input sequence $x^{\langle 1\rangle} = 1$, $x^{\langle 2\rangle} = 2$.

Step 1. With $\boldsymbol{a}^{\langle 0\rangle} = \mathbf{0}$ the recurrent term drops out:

$$\boldsymbol{a}^{\langle 1\rangle} = \tanh\big(W_{ax}\, x^{\langle 1\rangle}\big) = \tanh \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix} = \begin{bmatrix} 0.462 \\ -0.462 \end{bmatrix}.$$

Step 2. Now both terms contribute. First the recurrent part,

$$W_{aa}\boldsymbol{a}^{\langle 1\rangle} = \begin{bmatrix} 0.1(0.462) + (-0.2)(-0.462) \\ 0.3(0.462) + 0.1(-0.462) \end{bmatrix} = \begin{bmatrix} 0.139 \\ 0.092 \end{bmatrix},$$

then add the input term $W_{ax}\, x^{\langle 2\rangle} = (1.0, -1.0)^\top$ and apply tanh:

$$\boldsymbol{a}^{\langle 2\rangle} = \tanh \begin{bmatrix} 0.139 + 1.0 \\ 0.092 - 1.0 \end{bmatrix} = \tanh \begin{bmatrix} 1.139 \\ -0.908 \end{bmatrix} = \begin{bmatrix} 0.814 \\ -0.720 \end{bmatrix}.$$

With an output weight $W_{ya} = (1, 1)$ and a sigmoid (a name-detector emitting a probability), the step-2 output is $\hat{y}^{\langle 2\rangle} = \sigma(0.814 - 0.720) = \sigma(0.094) = 0.523$. The state $\boldsymbol{a}^{\langle 2\rangle}$ depends on $x^{\langle 1\rangle}$ *through* $\boldsymbol{a}^{\langle 1\rangle}$, which is the whole point of the recurrence: the memory carries the first input's influence into the second step. ◊

13.2 Backpropagation through time

Training an RNN means computing the gradient of the total loss, summed over time steps, with respect to the shared weights. Unrolling the network in time turns it into a deep feed-forward network, one layer per time step, and the chain rule applies exactly as in Chapter 11; run on the unrolled network it is called *backpropagation through time* (BPTT). Because the weights are shared, each weight's gradient is the *sum* of its contributions at every time step, just as a convolutional filter sums gradients over spatial positions.

The trouble is that the unrolled network is as deep as the sequence is long, so the vanishing- and exploding-gradient problem of Chapter 11 is acute. The gradient flowing from the loss at step T back to the state at step 1 is multiplied by the recurrent Jacobian once per step, a product of T factors. For a long sequence, T can be in the hundreds, so if the typical factor is even slightly below one the gradient vanishes and the network cannot learn long-range dependencies, the very thing sequences are about; if slightly above one it explodes. Exploding gradients are patched by *gradient clipping*, rescaling the gradient when its norm exceeds a threshold. Vanishing gradients need an architectural fix, and that is what the gated units provide.

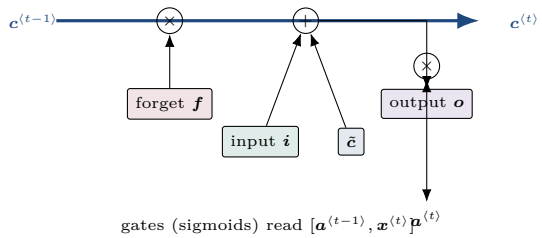


Figure 13.2. The cell state $\boldsymbol{c}$ runs along the top as a memory highway. The forget gate scales what is kept, the input gate scales the candidate $\tilde{\boldsymbol{c}}$ added, and the output gate scales what is exposed as $\boldsymbol{a}$. With forget open and input closed the cell passes unchanged, giving the gradient an unobstructed path back through time.

13.3 Gated units: the LSTM and GRU

Remark 13.2 *(An aside on checking intuitions).* Professor Chin opens this topic with a puzzle. Imagine a ribbon wrapped snugly around the Earth's equator, then lengthened by one meter and held at a uniform height h above the ground all the way around. How big is h? The circumference is $2\pi R$, the new ribbon is $2\pi(R+h) = 2\pi R + 1$, so $2\pi h = 1$ and $h = 1/(2\pi) \approx 0.16$ meters, about sixteen centimeters, *independent of the Earth's radius.* The same gap would open above a basketball. The lesson, and the reason it precedes the gated units: intuition about how quantities scale is often wrong, and a careful derivation beats a confident guess. The plain RNN's intuition, that a hidden state will simply remember the past, fails for exactly such a scaling reason, the geometric decay of the gradient, and the gates are the careful fix.

The *long short-term memory* unit (LSTM) of Hochreiter and Schmidhuber [18] adds a separate *cell state* $\boldsymbol{c}^{\langle t\rangle}$, a memory conveyor belt, and three *gates* that control it. Each gate is a sigmoid-activated layer producing values in $(0,1)$ that multiply a quantity elementwise, acting as a soft switch. With $[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}]$ denoting the concatenation of the previous state and the current input, the LSTM computes a forget gate, an input gate, an output gate, and a candidate update:

$$\begin{aligned} \boldsymbol{f}^{\langle t\rangle} &= \sigma(W_f[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_f), & \boldsymbol{i}^{\langle t\rangle} &= \sigma(W_i[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_i), \\ \boldsymbol{o}^{\langle t\rangle} &= \sigma(W_o[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_o), & \tilde{\boldsymbol{c}}^{\langle t\rangle} &= \tanh(W_c[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_c), \end{aligned} \tag{13.3}$$

and then updates the cell and the hidden state by

$$\boldsymbol{c}^{\langle t\rangle} = \boldsymbol{f}^{\langle t\rangle} \odot \boldsymbol{c}^{\langle t-1\rangle} + \boldsymbol{i}^{\langle t\rangle} \odot \tilde{\boldsymbol{c}}^{\langle t\rangle}, \qquad \boldsymbol{a}^{\langle t\rangle} = \boldsymbol{o}^{\langle t\rangle} \odot \tanh(\boldsymbol{c}^{\langle t\rangle}). \tag{13.4}$$

The forget gate $\boldsymbol{f}$ decides what to erase from the cell, the input gate $\boldsymbol{i}$ decides what new information to write, and the output gate $\boldsymbol{o}$ decides what to expose as the hidden state. The reason this fixes vanishing gradients is the cell-update equation: if the forget gate is open ($\boldsymbol{f} = \mathbf{1}$) and the input gate is closed ($\boldsymbol{i} = \mathbf{0}$), then $\boldsymbol{c}^{\langle t\rangle} = \boldsymbol{c}^{\langle t-1\rangle}$ exactly, the memory is carried forward unchanged, and the gradient flows back through that addition undiminished, just as a residual connection does in Chapter 12. The cell state is a protected highway for gradients across many time steps. The *gated recurrent unit* (GRU) of Cho et al. [6] is a streamlined cousin with two gates instead of three, merging the cell and hidden states; it is cheaper and often as effective. Figure 13.2 sketches the cell.

The GRU writes the same idea with two gates and no separate cell. An *update* gate Γ_u and a *reset* gate Γ_r control a candidate state, and the new state interpolates between the old one and the candidate:

$$\begin{aligned} \Gamma_u &= \sigma(W_u[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_u), & \Gamma_r &= \sigma(W_r[\boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}_r), \\ \tilde{\boldsymbol{a}}^{\langle t\rangle} &= \tanh(W[\Gamma_r \odot \boldsymbol{a}^{\langle t-1\rangle}, \boldsymbol{x}^{\langle t\rangle}] + \boldsymbol{b}), & \boldsymbol{a}^{\langle t\rangle} &= \Gamma_u \odot \tilde{\boldsymbol{a}}^{\langle t\rangle} + (\mathbf{1} - \Gamma_u) \odot \boldsymbol{a}^{\langle t-1\rangle}. \end{aligned} \tag{13.5}$$

When $\Gamma_u = \mathbf{0}$ the state is copied forward, $\boldsymbol{a}^{\langle t\rangle} = \boldsymbol{a}^{\langle t-1\rangle}$, the same gradient highway the LSTM's open forget gate gives; the reset gate Γ_r lets the unit ignore the past when forming a fresh candidate.

Example 13.3 (An LSTM step, by the numbers). Take a scalar cell with previous cell $c^{\langle t-1\rangle} = 0.8$ and previous hidden $a^{\langle t-1\rangle} = 0.5$, and suppose the gate layers produce the pre-activations $z_f = 2.0$, $z_i = -1.0$, $z_o = 0.5$ and candidate pre-activation $z_{\tilde{c}} = 1.0$. Apply the sigmoid to the gates and the tanh to the candidate:

$$f = \sigma(2.0) = 0.881, \quad i = \sigma(-1.0) = 0.269, \quad o = \sigma(0.5) = 0.622, \quad \tilde{c} = \tanh(1.0) = 0.762.$$

The cell update keeps most of the old memory and adds a little new,

$$c^{\langle t\rangle} = f\, c^{\langle t-1\rangle} + i\, \tilde{c} = (0.881)(0.8) + (0.269)(0.762) = 0.705 + 0.205 = 0.910,$$

and the hidden state is the output gate times the squashed cell,

$$a^{\langle t\rangle} = o\, \tanh(c^{\langle t\rangle}) = (0.622)\tanh(0.910) = (0.622)(0.721) = 0.449.$$

The forget gate at 0.881 preserved 0.705 of the old cell while the input gate at 0.269 admitted only a fraction of the candidate; had the gates been $f = 1$, $i = 0$, the cell would have passed unchanged at 0.8, the memory-preserving limit that protects the gradient. ◊

13.4 Language models and machine translation

A *language model* assigns a probability to a sequence of words, and the chain rule of probability factors it into a product of next-word predictions,

$$p(y_1, \ldots, y_T) = \prod_{t=1}^{T} p(y_t \mid y_1, \ldots, y_{t-1}). \tag{13.6}$$

For instance $p(\text{“tom and jerry”}) = p(\text{tom})\, p(\text{and} \mid \text{tom})\, p(\text{jerry} \mid \text{tom, and})$. An RNN models each factor: at step t it reads the words so far through its hidden state and outputs a softmax over the vocabulary for the next word, $\hat{\boldsymbol{y}}^{\langle t\rangle} = \text{softmax}(W_{ya}\boldsymbol{a}^{\langle t\rangle} + \boldsymbol{b}_y)$. Training minimizes the cross-entropy of the predicted next word against the true next word, summed over the sequence, the categorical loss of Chapter 3. To *generate* text, one samples a word from the softmax, feeds it back in as the next input, and repeats, the model writing one word at a time conditioned on its own output.

Conditioning the language model on an input sequence gives *machine translation*: $p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_t p(y_t \mid y_{<t}, \boldsymbol{x})$, the probability of an output sentence given an input sentence. The classic architecture is the *encoder-decoder*: one RNN (the encoder) reads the source sentence and compresses it into a context vector, and a second RNN (the decoder) generates the translation from that context. Decoding searches for a high-probability output sequence; greedy decoding takes the most likely word at each step, while *beam search* keeps the few best partial sequences at each step and is less likely to be trapped by a bad early choice. The weakness of the basic encoder-decoder, that the whole source sentence must be squeezed through one fixed-size context vector, is what attention was invented to fix.

13.5 Attention

Attention lets the decoder look back at *all* of the encoder's states, not just a single summary, and decide at each step which source words to focus on. For each decoder position it forms a *context vector* as a weighted average of the encoder states $\boldsymbol{h}^{\langle t'\rangle}$,

$$\boldsymbol{c}^{\langle t\rangle} = \sum_{t'} \alpha^{\langle t,t'\rangle}\, \boldsymbol{h}^{\langle t'\rangle}, \qquad \alpha^{\langle t,t'\rangle} = \frac{\exp(e^{\langle t,t'\rangle})}{\sum_{s} \exp(e^{\langle t,s\rangle})}, \tag{13.7}$$

where the *attention weights* $\alpha^{\langle t,t'\rangle}$ are a softmax over *scores* $e^{\langle t,t'\rangle}$ that measure how relevant source position t' is to the current decoder state. The weights sum to one and are large for the source words the decoder should attend to, so the context is a focused summary that changes from step to step. Attention turns the fixed bottleneck into a dynamic, learned lookup over the whole input, and it improved translation immediately. It also suggested a more radical idea: if attention can replace the recurrence's role of moving information across positions, perhaps the recurrence is not needed at all.

13.6 Self-attention and the Transformer

The Transformer of Vaswani et al. [42] dispenses with recurrence entirely, building a sequence model out of attention alone. The key operation is *self-attention*, in which a sequence attends to itself: every position gathers information from every other position in one step, with no need to pass it hand-to-hand through a hidden state.

The mechanism casts attention as a soft dictionary lookup. From the input, packed as a matrix X with one row per token, three linear projections produce a *query*, a *key*, and a *value* for each token,

$$Q = XW^Q, \qquad K = XW^K, \qquad V = XW^V.$$

The query of a token asks "what am I looking for," each key advertises "what do I contain," and each value carries "what I will contribute." A token attends to others by matching its query against their keys, the match score being a dot product, and the scaled dot-product attention is

$$\text{Attention}(Q, K, V) = \text{softmax}\Big(\frac{QK^\top}{\sqrt{d_k}}\Big)V. \tag{13.8}$$

The matrix $QK^\top$ holds the score of every query against every key, the softmax (applied row by row) turns each token's scores into attention weights summing to one, and multiplying by V averages the values accordingly. The division by $\sqrt{d_k}$, where d_k is the key dimension, is not cosmetic: the dot product of two d_k-dimensional vectors has variance growing with d_k, so without the scaling the scores would be large, the softmax would saturate into a near one-hot vector, and its gradient would vanish; the $\sqrt{d_k}$ keeps the scores at a healthy scale.

Tracking the shapes is the cleanest way to understand the operation, and it is a favorite exam question. With T tokens and key dimension d_k: X is $T \times d_{\text{model}}$; each projection matrix W^Q, W^K, W^V is $d_{\text{model}} \times d_k$; so Q, K, V are each $T \times d_k$; the product $QK^\top$ is $(T \times d_k)(d_k \times T) = T \times T$, one score per ordered pair of tokens; the row-wise softmax leaves it $T \times T$; and multiplying by V gives $(T \times T)(T \times d_k) = T \times d_k$, one output vector per token. The output is the same shape as the input, so self-attention layers stack.

Example 13.4 (Self-attention by hand). Take two tokens with embeddings $\boldsymbol{x}_1 = (1, 0)$ and $\boldsymbol{x}_2 = (0, 1)$, so $X = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, and for clarity let the projection matrices be the identity, $W^Q = W^K = W^V = I$ (in practice they are learned). Then $Q = K = V = X$. The score matrix is $QK^\top = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$, scaled by $\sqrt{d_k} = \sqrt{2}$ to $\left[\begin{smallmatrix} 0.707 & 0 \\ 0 & 0.707 \end{smallmatrix}\right]$. Take the softmax of the first row $[0.707, 0]$: the exponentials are $e^{0.707} = 2.028$ and $e^0 = 1$, summing to 3.028, so the attention weights are $[2.028/3.028,\ 1/3.028] = [0.670, 0.330]$. By symmetry the second row's weights are $[0.330, 0.670]$. The output of token 1 is therefore $0.670\,\boldsymbol{v}_1 + 0.330\,\boldsymbol{v}_2 = 0.670(1, 0) + 0.330(0, 1) = (0.670, 0.330)$: each token attends mostly to itself and partly to the other, blending their values. Learned, non-identity projections let the model choose which tokens attend to which. ◊

A full Transformer wraps self-attention in three more ideas (Fig. 13.3). *Multi-head attention* runs several self-attention operations in parallel with different learned projections and concatenates them, so different heads can specialize, one tracking syntax, another long-range reference. *Positional encodings* are added to the token embeddings because self-attention is otherwise permutation-invariant, treating the sequence as a set; the encodings (sinusoids of different frequencies, or learned vectors) inject the order that Section 13.1 got for free from the recurrence. And each sublayer is wrapped in a *residual connection* (Chapter 12) and *layer normalization*, which together keep gradients flowing in the very deep stacks that Transformers use. The advantage over the RNN is parallelism and reach: self-attention connects any two positions in a single step (path length $O(1)$ versus the RNN's $O(T)$) and processes all positions at once rather than sequentially, which is why Transformers train so much faster and scale to the enormous models behind modern language systems.

Remark 13.5 *(Beyond the architecture).* The Transformer is the engine of modern large language models, which are Transformers with billions of parameters trained on enormous text corpora to predict the next token, the language-modeling objective of Eq. (13.6) at scale. The same attention machinery, applied to image patches, audio frames, or other modalities, now underlies much of deep learning beyond text. A parallel line of generative models, *diffusion models* [17], generates images by learning to reverse a gradual noising process, training a network to predict and remove the noise added to data; they are the basis of modern image generators. These developments are beyond the scope of the course, but they are direct descendants of the ideas in this chapter and the last.

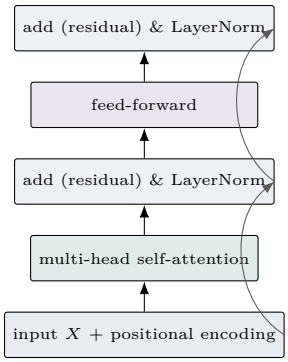


Figure 13.3. Self-attention lets every token gather from every other in one step; multi-head attention runs several in parallel; residual connections and layer normalization (the curved skip arcs) keep deep stacks trainable; and a position-wise feed-forward layer follows. Positional encodings restore the order that dropping the recurrence discarded.

13.7 The gradient through time, on numbers

Section 13.2 argued that the gradient flowing back through an RNN is multiplied by the recurrent Jacobian once per step, so a product of many factors decides whether it survives. The cleanest way to see that product is to differentiate a tiny scalar RNN by hand. The mechanism that emerges, a chain of factors each smaller than one, is exactly what the gated units of Section 13.3 were built to defeat.

Example 13.6 (Backpropagation through time by hand). Take a scalar RNN with recurrent weight $w = 0.5$, input weight $u = 0.5$, no bias, and tanh activation, so $a^{\langle t\rangle} = \tanh\big(w\,a^{\langle t-1\rangle} + u\,x^{\langle t\rangle}\big)$. Start from $a^{\langle 0\rangle} = 0.5$, feed the constant input $x^{\langle 1\rangle} = x^{\langle 2\rangle} = x^{\langle 3\rangle} = 1$, and use the loss $L = \frac{1}{2}\big(a^{\langle 3\rangle}\big)^2$ at the final step only. We want $\partial L/\partial w$.

Forward pass. Running the recurrence three steps,

$$\begin{aligned} a^{\langle 1\rangle} &= \tanh(0.5 \cdot 0.5 + 0.5) = \tanh(0.75) = 0.635, \\ a^{\langle 2\rangle} &= \tanh(0.5 \cdot 0.635 + 0.5) = \tanh(0.818) = 0.674, \\ a^{\langle 3\rangle} &= \tanh(0.5 \cdot 0.674 + 0.5) = \tanh(0.837) = 0.684. \end{aligned}$$

The recurrent Jacobian. Since $\tanh'(z) = 1 - \tanh^2(z)$, the derivative of one state with respect to the previous one is $J^{\langle t\rangle} = \partial a^{\langle t\rangle}/\partial a^{\langle t-1\rangle} = \big(1 - (a^{\langle t\rangle})^2\big)\,w$. Numerically,

$$\begin{aligned} J^{\langle 1\rangle} &= (1 - 0.635^2)(0.5) = (0.597)(0.5) = 0.298, \\ J^{\langle 2\rangle} &= (1 - 0.674^2)(0.5) = (0.546)(0.5) = 0.273, \\ J^{\langle 3\rangle} &= (1 - 0.684^2)(0.5) = (0.532)(0.5) = 0.266. \end{aligned}$$

Each factor sits well below one, because $\tanh' \le 1$ and the weight is 0.5.

Backward pass. The weight w enters at every step, so by the chain rule its gradient is a sum of one contribution per step, and the contribution from step k travels forward to the loss through every Jacobian after it:

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a^{\langle 3\rangle}} \sum_{k=1}^{3} r_k\,\ell_k, \qquad r_k = \prod_{j=k+1}^{3} J^{\langle j\rangle}, \quad \ell_k = \big(1 - (a^{\langle k\rangle})^2\big)\,a^{\langle k-1\rangle}, \tag{13.9}$$

where r_k collects the Jacobians *after* step k and ℓ_k is the local derivative of $a^{\langle k\rangle}$ with respect to w. Here $\partial L/\partial a^{\langle 3\rangle} = a^{\langle 3\rangle} = 0.684$, and the local terms are $\ell_1 = 0.298$, $\ell_2 = 0.347$, $\ell_3 = 0.358$ (each $\ell_k = (1 - (a^{\langle k\rangle})^2)\,a^{\langle k-1\rangle}$). Summing the three contributions,

$$\begin{aligned} k = 3 &: 0.684\,(0.358) = 0.245, \\ k = 2 &: 0.684\,(J^{\langle 3\rangle})(0.347) = 0.684\,(0.266)(0.347) = 0.063, \\ k = 1 &: 0.684\,(J^{\langle 3\rangle} J^{\langle 2\rangle})(0.298) = 0.684\,(0.0726)(0.298) = 0.0148, \end{aligned}$$

so $\partial L/\partial w = 0.245 + 0.063 + 0.0148 = 0.323$. The earliest step's contribution is the smallest, and the reason is visible in the arithmetic: it carries the *product* $J^{\langle 3\rangle} J^{\langle 2\rangle} = 0.0726$ of two recurrent Jacobians, while the latest step carries no such product at all. ◊

The example is short, so the suppression is mild. Stretch the same product over a long sequence and it becomes fatal. If each recurrent factor is about 0.27, as it was above, then the gradient reaching back 20 steps is scaled by roughly $0.27^{20} \approx 4 \times 10^{-12}$: the loss at step 20 learns essentially nothing about the input at step 1. This is the *vanishing gradient*. The opposite case is just as real. Replace the contractive factor with an expansive one, say a Jacobian near 1.5, and the same product becomes $1.5^{20} \approx 3.3 \times 10^{3}$, an *exploding gradient* that throws the weights to nonsense. Gradient clipping (Section 13.2) caps the explosion; the gated cell of the LSTM and GRU cures the vanishing case by replacing the repeated multiplicative factor with an additive carry, whose Jacobian is 1.

Proposition 13.7 (Why the product governs the gradient). For a recurrent state $a^{\langle t \rangle} = g(w\, a^{\langle t-1 \rangle} + \cdots)$, the sensitivity of a late state to a much earlier one is the product of the intervening single-step Jacobians,

$$\frac{\partial a^{\langle T \rangle}}{\partial a^{\langle t \rangle}} = \prod_{s=t+1}^{T} \frac{\partial a^{\langle s \rangle}}{\partial a^{\langle s-1 \rangle}} = \prod_{s=t+1}^{T} g'(\cdot)\, w. \tag{13.10}$$

A product of $T - t$ numbers shrinks geometrically when the typical factor is below one and grows geometrically when it is above one, so the gradient at long range is controlled by whether $|g'(\cdot)\, w|$ tends to sit under or over 1.

The additive cell update of Eq. (13.4) sidesteps Eq. (13.10) entirely. When the forget gate is open and the input gate is closed, $\partial \boldsymbol{c}^{\langle t \rangle} / \partial \boldsymbol{c}^{\langle t-1 \rangle} = \mathbf{1}$, so the product in Eq. (13.10) is a product of ones and never decays. That single observation, seen here on numbers, is the whole reason gates work.

13.8 Self-attention with learned projections

The worked example of Example 13.4 used identity projections and two tokens so the softmax stayed simple. Real attention learns the projections W^Q, W^K, W^V, and the softmax runs over every token in the sequence. The next example does both: three tokens and genuinely non-identity projections, with one query's softmax computed in full.

Example 13.8 (Three-token self-attention with non-identity projections). Take three tokens with embeddings $\boldsymbol{x}_1 = (1, 0)$, $\boldsymbol{x}_2 = (0, 1)$, $\boldsymbol{x}_3 = (1, 1)$, stacked as the rows of

$$X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad W^Q = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \quad W^K = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad W^V = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix},$$

with key dimension $d_k = 2$. Project the rows of X through each matrix (each row times the matrix), giving

$$Q = XW^Q = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 2 \end{bmatrix}, \qquad K = XW^K = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 2 & 1 \end{bmatrix},$$

$$V = XW^V = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & 2 \end{bmatrix}.$$

The score matrix $QK^\top$ holds every query-key dot product. Its first row is the query $\boldsymbol{q}_1 = (1, 1)$ against the three keys: $\boldsymbol{q}_1 \cdot \boldsymbol{k}_1 = 1$, $\boldsymbol{q}_1 \cdot \boldsymbol{k}_2 = 1 + 1 = 2$, $\boldsymbol{q}_1 \cdot \boldsymbol{k}_3 = 2 + 1 = 3$. Doing the same for the other queries,

$$QK^\top = \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 1 \\ 1 & 3 & 4 \end{bmatrix}, \qquad \frac{QK^\top}{\sqrt{d_k}} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & 1 \\ 1 & 3 & 4 \end{bmatrix}.$$

Softmax of the first row. The scaled scores are $[1, 2, 3]/\sqrt{2} = [0.707, 1.414, 2.121]$. Exponentiating, $e^{0.707} = 2.028$, $e^{1.414} = 4.113$, $e^{2.121} = 8.342$, which sum to 14.484. Dividing,

$$\operatorname{softmax}\big([0.707, 1.414, 2.121]\big) = \frac{[2.028,\ 4.113,\ 8.342]}{14.484} = [0.140,\ 0.284,\ 0.576],$$

and these weights sum to 1, as a softmax must. Token 1 attends most to token 3, whose key scored highest against its query.

Output of token 1. The output is the weighted sum of the value rows,

$$\begin{aligned}\boldsymbol{o}_1 &= 0.140\,\boldsymbol{v}_1 + 0.284\,\boldsymbol{v}_2 + 0.576\,\boldsymbol{v}_3 \\ &= 0.140\,(1,0) + 0.284\,(0,2) + 0.576\,(1,2) = (0.716,\ 1.720).\end{aligned}$$

The second coordinate is large because both $\boldsymbol{v}_2$ and $\boldsymbol{v}_3$ carry a 2 there and together hold most of the attention weight. Running the same softmax on rows 2 and 3 gives weights $[0.198, 0.401, 0.401]$ and $[0.074, 0.306, 0.620]$, and stacking the three output rows produces the 3×2 result, the same shape as X, so the layer can feed another just like it. ◊

13.9 Multi-head attention and positional encodings, on numbers

Figure 13.3 listed multi-head attention and positional encodings as two of the pieces wrapped around self-attention. Both are easiest to pin down with a shape count and a small numeric table, the kind of check an exam rewards.

Example 13.9 (Multi-head attention: shapes and parameters). A multi-head layer splits the model dimension across h heads. Take $d_{\text{model}} = 8$ and $h = 4$ heads, so each head works in dimension $d_k = d_v = d_{\text{model}}/h = 2$. Each head i owns its own three projections,

$$W_i^Q,\ W_i^K,\ W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k} = \mathbb{R}^{8\times 2},$$

so one head holds $3\,(8 \times 2) = 48$ weights, and all four heads together hold $4 \times 48 = 192$. Inside head i, with T tokens, Q_i, K_i, V_i are each $T \times 2$, the score matrix $Q_i K_i^\top$ is $T \times T$ (the per-token-pair count does not depend on d_k), and the head output is $T \times 2$. Concatenating the four head outputs along the feature axis restores the width,

$$\underbrace{T \times 2}_{\text{head 1}} \,\|\, \cdots \,\|\, \underbrace{T \times 2}_{\text{head 4}} \;=\; T \times 8 = T \times d_{\text{model}},$$

and a final output projection $W^O \in \mathbb{R}^{8\times 8}$ mixes the concatenated heads, adding 64 weights. The layer's total is $192 + 64 = 256$ weights (ignoring biases). That equals $4\,d_{\text{model}}^2 = 4 \cdot 8^2$, the same count a single full-width attention with $d_k = d_{\text{model}}$ would use: multi-head attention buys several independent attention patterns at no extra parameter cost, because each head is correspondingly narrower. ◊

Positional encodings restore the order that self-attention discards. The sinusoidal scheme of Vaswani et al. [42] assigns position pos the vector

$$\text{PE}(\text{pos}, 2i) = \sin\!\Big(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\Big), \qquad \text{PE}(\text{pos}, 2i+1) = \cos\!\Big(\frac{\text{pos}}{10000^{2i/d_{\text{model}}}}\Big), \tag{13.11}$$

pairing each even dimension's sine with the next dimension's cosine at the same frequency. Low dimensions (i small) use high frequencies that change fast across positions; high dimensions use slow ones, so a position is written in something like binary across scales.

Example 13.10 (Sinusoidal positional encoding by hand). Take $d_{\text{model}} = 4$, so there are two frequency pairs, $i = 0$ and $i = 1$, with denominators $10000^{0/4} = 1$ and $10000^{2/4} = 100$. The encoding of a position is $\big(\sin(\text{pos}/1),\ \cos(\text{pos}/1),\ \sin(\text{pos}/100),\ \cos(\text{pos}/100)\big)$. For the first three positions,

$$\begin{aligned}&\text{pos} = 0 : (\sin 0,\ \cos 0,\ \sin 0,\ \cos 0) = (0,\ 1,\ 0,\ 1),\\ &\text{pos} = 1 : (\sin 1,\ \cos 1,\ \sin 0.01,\ \cos 0.01) = (0.841,\ 0.540,\ 0.010,\ 1.000),\\ &\text{pos} = 2 : (\sin 2,\ \cos 2,\ \sin 0.02,\ \cos 0.02) = (0.909,\ -0.416,\ 0.020,\ 1.000).\end{aligned}$$

The first pair (dimensions 0 and 1) swings noticeably from one position to the next, giving the model a fast clock; the second pair (dimensions 2 and 3) barely moves over these positions, a slow clock that distinguishes far-apart positions. Adding this fixed vector to each token's embedding before the projections lets a token's query and key depend on where it sits, so reordering the sequence changes the attention scores. ◊

Notes and References

Recurrent networks, backpropagation through time, the gated units, and attention follow Chapter 10 of Goodfellow et al. [13]. The LSTM is due to Hochreiter and Schmidhuber [18] and the GRU to Cho et al. [6]. The Transformer and scaled dot-product attention are from Vaswani et al. [42], whose architecture (multi-head attention, positional encodings, residual connections, and layer normalization) is described here in outline; the original paper gives the hyperparameters and the machine-translation results that established the model. The scaling by $\sqrt{d_k}$ and the query-key-value framing are from that paper, and the residual connections trace back to the ResNet of Chapter 12. Diffusion models are due to Ho et al. [17]. This chapter closes Part IV; the kernel methods of Chapter 14 return to a classical, non-network route to nonlinearity, and the inner-product view of attention in Eq. (13.8) is a foretaste of the kernel idea there.

Exercises

Exercise 13.1. Explain why a recurrent network can process sequences of varying length with a fixed number of parameters, and why this is the temporal analogue of weight sharing in a convolutional network.

Solution. The RNN uses the same weight matrices W_{aa}, W_{ax}, W_{ya} at every time step, applying the update $\boldsymbol{a}^{\langle t\rangle} = g(W_{aa}\boldsymbol{a}^{\langle t-1\rangle} + W_{ax}\boldsymbol{x}^{\langle t\rangle} + \boldsymbol{b}_a)$ for as many steps as the sequence has elements. Because the parameters do not depend on t or on the total length T, the same fixed set processes any-length input: a longer sequence just means more applications of the same rule. This mirrors a convolutional layer, which applies the same filter at every spatial position, so a CNN handles any image size with fixed parameters; both share weights across the index (time or space) to encode the assumption that the same operation is appropriate everywhere along it. □

Exercise 13.2. Show that when the LSTM forget gate is fully open ($\boldsymbol{f}^{\langle t\rangle} = \mathbf{1}$) and the input gate is fully closed ($\boldsymbol{i}^{\langle t\rangle} = \mathbf{0}$), the cell state is carried forward unchanged, and explain why this addresses the vanishing-gradient problem.

Solution. The cell update is $\boldsymbol{c}^{\langle t\rangle} = \boldsymbol{f}^{\langle t\rangle} \odot \boldsymbol{c}^{\langle t-1\rangle} + \boldsymbol{i}^{\langle t\rangle} \odot \tilde{\boldsymbol{c}}^{\langle t\rangle}$. With $\boldsymbol{f} = \mathbf{1}$ and $\boldsymbol{i} = \mathbf{0}$ this becomes $\boldsymbol{c}^{\langle t\rangle} = \mathbf{1} \odot \boldsymbol{c}^{\langle t-1\rangle} + \mathbf{0} \odot \tilde{\boldsymbol{c}}^{\langle t\rangle} = \boldsymbol{c}^{\langle t-1\rangle}$, the cell unchanged. For the gradient, $\partial \boldsymbol{c}^{\langle t\rangle}/\partial \boldsymbol{c}^{\langle t-1\rangle} = \mathbf{1}$ (an identity, elementwise), so the gradient passes back through this step undiminished rather than being multiplied by a small factor. Over many steps the cell provides a path along which the gradient does not decay geometrically, the same mechanism as a residual connection, so long-range dependencies can be learned. □

Exercise 13.3. In scaled dot-product self-attention with T tokens, model dimension d_{model}, and key dimension d_k, give the shapes of X, W^Q, Q, $QK^\top$, the softmax output, and the final result. In particular, what is the shape of $QK^\top$?

Solution. X is $T \times d_{\text{model}}$ (one row per token). Each projection W^Q (and W^K, W^V) is $d_{\text{model}} \times d_k$, so $Q = XW^Q$ (and K, V) is $T \times d_k$. The score matrix $QK^\top$ is $(T \times d_k)(d_k \times T) = T \times T$, one score per ordered pair of tokens; the row-wise softmax preserves this, $T \times T$. Multiplying by V gives $(T \times T)(T \times d_k) = T \times d_k$, one output vector per token, the same number of rows as the input. So $QK^\top$ is $T \times T$. □

Exercise 13.4. Two tokens have embeddings $\boldsymbol{x}_1 = (2, 0)$ and $\boldsymbol{x}_2 = (0, 2)$, with $W^Q = W^K = W^V = I$ and $d_k = 2$. Compute the first row of the scaled-attention weight matrix and the output for token 1.

Solution. With identity projections, $Q = K = V = X$. The score of token 1 against token 1 is $\boldsymbol{x}_1 \cdot \boldsymbol{x}_1 = 4$ and against token 2 is $\boldsymbol{x}_1 \cdot \boldsymbol{x}_2 = 0$. Scaling by $\sqrt{d_k} = \sqrt{2}$ gives scaled scores $[4/\sqrt{2}, 0] = [2.83, 0]$. The softmax: $e^{2.83} = 16.9$, $e^0 = 1$, sum 17.9, so the weights are $[16.9/17.9,\ 1/17.9] = [0.944, 0.056]$. The output of token 1 is $0.944\,\boldsymbol{v}_1 + 0.056\,\boldsymbol{v}_2 = 0.944(2, 0) + 0.056(0, 2) = (1.89, 0.11)$. Token 1 attends almost entirely to itself here, because its query matches its own key far more strongly than the orthogonal key of token 2. □

Exercise 13.5. Self-attention with no positional information gives the same output set for a sequence and any reordering of it. Show informally why, and explain how positional encodings fix it.

Solution. Self-attention computes each output as a weighted sum of values, with weights from softmax of dot products of queries and keys; all of these depend only on the token embeddings rather than on their positions. If the tokens are permuted, the same set of queries, keys, and values is produced (just relabeled), the same pairwise scores arise, and the same set of outputs results, permuted accordingly: the operation is equivariant to permutation and so cannot tell "dog bites man" from "man bites dog." Adding a distinct positional encoding to each token's embedding before the projections breaks this symmetry: now a token's query and key depend on *where* it sits, so reordering changes the scores and the outputs, letting the model use word order. □

Exercise 13.6. A scalar RNN has $W_{aa} = 0.5$, $W_{ax} = 1.0$, $b_a = 0$, tanh activation, and $a^{\langle 0\rangle} = 0$. For the input sequence $x^{\langle 1\rangle} = 1$, $x^{\langle 2\rangle} = -1$, compute $a^{\langle 1\rangle}$ and $a^{\langle 2\rangle}$. Use $\tanh(1) = 0.762$, $\tanh(-0.619) = -0.550$.

Solution. $a^{\langle 1\rangle} = \tanh(0.5\cdot 0 + 1.0\cdot 1) = \tanh(1) = 0.762$. Then $a^{\langle 2\rangle} = \tanh(0.5\cdot 0.762 + 1.0\cdot(-1)) = \tanh(0.381 - 1) = \tanh(-0.619) = -0.550$. The second state depends on the first through the recurrent term $0.5 \cdot a^{\langle 1\rangle}$, so the memory of $x^{\langle 1\rangle}$ persists into step 2. □

Exercise 13.7. A scalar RNN has recurrent weight $w = 0.5$, input weight $u = 1$, no bias, tanh activation, and $a^{\langle 0\rangle} = 0.5$. With inputs $x^{\langle 1\rangle} = x^{\langle 2\rangle} = 1$ and loss $L = \frac{1}{2}(a^{\langle 2\rangle})^2$, compute $\partial L/\partial w$, showing the two per-step contributions and which recurrent Jacobian suppresses the earlier one. Use $\tanh(1.25) = 0.848$ and $\tanh(1.424) = 0.891$.

Solution. Forward: $a^{\langle 1\rangle} = \tanh(0.5 \cdot 0.5 + 1) = \tanh(1.25) = 0.848$, then $a^{\langle 2\rangle} = \tanh(0.5 \cdot 0.848 + 1) = \tanh(1.424) = 0.891$. The recurrent Jacobian is $J^{\langle 2\rangle} = \partial a^{\langle 2\rangle}/\partial a^{\langle 1\rangle} = (1 - 0.891^2)(0.5) = (0.207)(0.5) = 0.104$. The weight enters at both steps, so $\partial L/\partial w = a^{\langle 2\rangle}\, \partial a^{\langle 2\rangle}/\partial w$ with $\partial a^{\langle 2\rangle}/\partial w$ summing a step-2 term and a step-1 term:

$$\text{step 2: } (1 - 0.891^2)\, a^{\langle 1\rangle} = (0.207)(0.848) = 0.176,$$
$$\text{step 1: } J^{\langle 2\rangle}\, (1 - 0.848^2)\, a^{\langle 0\rangle} = 0.104\, (0.280)(0.5) = 0.0146.$$

So $\partial a^{\langle 2\rangle}/\partial w = 0.176 + 0.0146 = 0.190$ and $\partial L/\partial w = 0.891\,(0.190) = 0.169$. The step-1 contribution (0.0146) is the smaller because it is multiplied by the recurrent Jacobian $J^{\langle 2\rangle} = 0.104$, the single factor that, repeated over a long sequence, makes the gradient vanish. □

Exercise 13.8. A multi-head attention layer has $d_{\text{model}} = 12$ and $h = 3$ heads, with $d_k = d_v = d_{\text{model}}/h$. Give d_k, the shape of each head's W_i^Q, the shape of one head's $Q_i K_i^\top$ for a sequence of T tokens, the shape after concatenating the heads, the shape of the output projection W^O, and the total weight count (ignore biases).

Solution. Each head works in dimension $d_k = d_{\text{model}}/h = 12/3 = 4$. Each head projection W_i^Q (and W_i^K, W_i^V) is $d_{\text{model}} \times d_k = 12 \times 4$. Within a head, Q_i, K_i, V_i are $T \times 4$, so $Q_i K_i^\top$ is $(T \times 4)(4 \times T) = T \times T$, independent of d_k. The three projections per head hold $3(12 \times 4) = 144$ weights, and all three heads hold $3 \times 144 = 432$. Concatenating the head outputs gives $T \times (3 \cdot 4) = T \times 12 = T \times d_{\text{model}}$, and the output projection W^O is $12 \times 12 = 144$ weights. The total is $432 + 144 = 576$, which equals $4\, d_{\text{model}}^2 = 4 \cdot 12^2$: splitting into heads keeps the parameter count of one full-width attention while giving three independent attention patterns. □

Exercise 13.9. For the GRU state update $\boldsymbol{a}^{\langle t\rangle} = \Gamma_u \odot \tilde{\boldsymbol{a}}^{\langle t\rangle} + (\mathbf{1} - \Gamma_u) \odot \boldsymbol{a}^{\langle t-1\rangle}$ with scalar values, an update gate $\Gamma_u = 0.2$, candidate $\tilde{a}^{\langle t\rangle} = 0.9$, and previous state $a^{\langle t-1\rangle} = 0.4$, compute $a^{\langle t\rangle}$. What value of Γ_u would copy the previous state unchanged, and why does that matter for gradients?

Solution. $a^{\langle t\rangle} = 0.2(0.9) + (1 - 0.2)(0.4) = 0.18 + 0.32 = 0.50$, a blend weighted mostly toward the retained old state because the update gate is small. Setting $\Gamma_u = 0$ gives $a^{\langle t\rangle} = a^{\langle t-1\rangle} = 0.4$ exactly, copying the state forward; then $\partial a^{\langle t\rangle}/\partial a^{\langle t-1\rangle} = 1$, so the gradient passes back through the step undiminished, the gated highway that lets GRUs learn long-range dependencies where a plain RNN's gradient would vanish. □

PART V

Kernel Methods

CHAPTER 14

Kernel Methods

If the only thing your model needs from the data is inner products, you can swap the inner product for a kernel and work in a space you never have to visit.

after Professor Peter Chin, ENGS 106

A neural network earns its power by learning features. Kernel methods earn theirs a different way: they fix an enormous, even infinite, set of features in advance, and then arrange never to compute them. The trick rests on a recurring observation. Many linear models, when you look closely, depend on the data only through inner products between examples. If that is so, then mapping the data into a high-dimensional feature space $\boldsymbol{\phi}(\boldsymbol{x})$ and working there requires only the inner products $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$, and if those can be computed cheaply by a *kernel function* $k(\boldsymbol{x}, \boldsymbol{x}')$ without ever forming $\boldsymbol{\phi}$, the high-dimensional space comes for free. This is the kernel trick, and it turns linear methods into powerful nonlinear ones at almost no cost. This chapter develops the dual view that exposes the inner products, the kernel trick that exploits it, the condition (Mercer's) for a function to be a valid kernel, and the standard kernels, setting up the support vector machine of Chapter 15.

Roadmap

Suggested reading: Bishop [3], Sections 6.1–6.2; Deisenroth et al. [9], Section 12.4 for the kernel view.

We cover: feature maps and the cost problem (Section 14.1); the dual representation and the representer theorem (Section 14.2); the kernel trick (Section 14.3); Mercer's theorem and valid kernels (Section 14.4); and the standard kernel catalogue (Section 14.5).

14.1 Feature maps and the cost of going high-dimensional

A linear model is powerful only if its features are rich. We can always enrich them by a *feature map* $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^M$ that adds nonlinear combinations of the inputs, then fit a linear model $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x})$ in the enlarged space, which is nonlinear in $\boldsymbol{x}$. For example, the degree-two map in two dimensions,

$$\boldsymbol{\phi}(\boldsymbol{x}) = (x_1,\ x_2,\ x_1^2,\ x_2^2,\ x_1 x_2),$$

lets a linear model fit any quadratic boundary. The difficulty is that M grows fast: a degree-d polynomial map in n dimensions has $\mathcal{O}(n^d)$ features, and some useful maps are infinite-dimensional. Forming $\boldsymbol{\phi}(\boldsymbol{x})$ explicitly, storing it, and computing with it becomes impossible. The kernel trick sidesteps this entirely, but to see how, we first need to rewrite the model so that $\boldsymbol{\phi}$ appears only inside inner products.

14.2 The dual representation

Take ridge regression in feature space, minimizing $\frac{1}{2}\sum_n (\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$. Setting the gradient to zero, $\sum_n (\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n) + \lambda \boldsymbol{w} = \boldsymbol{0}$, and solving for $\boldsymbol{w}$,

$$\boldsymbol{w} = -\frac{1}{\lambda}\sum_n (\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)\boldsymbol{\phi}(\boldsymbol{x}_n) = \sum_n a_n\, \boldsymbol{\phi}(\boldsymbol{x}_n),$$

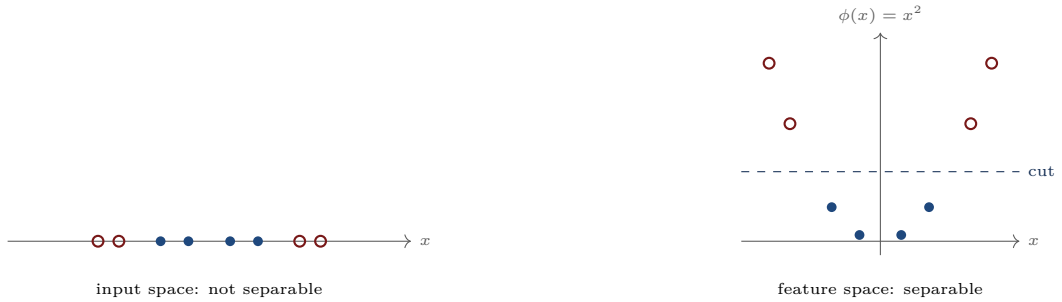


Figure 14.1. Two classes on the line (inner blue, outer red) that no single threshold separates. The feature map $\phi(x) = x^2$ sends them to a space where a single cut does separate them. A linear method in the feature space is a nonlinear method in the input space, and the kernel computes the needed inner products without forming ϕ.

where we have named the coefficient $a_n = -\frac{1}{\lambda}(\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n)$. The content of this line is the *representer theorem*: the optimal weight vector lies in the span of the training features, $\boldsymbol{w} = \sum_n a_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^\top \boldsymbol{a}$, where $\boldsymbol{\Phi}$ is the design matrix with row n equal to $\boldsymbol{\phi}(\boldsymbol{x}_n)^\top$. Even if the feature space has a million dimensions, the solution lives in the at-most-N-dimensional subspace the data spans, so we may as well solve for the N coefficients $\boldsymbol{a}$ instead of the M weights $\boldsymbol{w}$. This is the *dual representation*.

Substitute $\boldsymbol{w} = \boldsymbol{\Phi}^\top \boldsymbol{a}$ back into the objective. Every appearance of $\boldsymbol{\phi}$ now sits inside an inner product, which assembles into the *Gram matrix* (or kernel matrix)

$$K = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top, \qquad K_{nm} = \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m), \tag{14.1}$$

an $N \times N$ matrix of inner products between all pairs of training examples. The objective becomes $\frac{1}{2}\|K\boldsymbol{a} - \boldsymbol{t}\|^2 + \frac{\lambda}{2}\boldsymbol{a}^\top K \boldsymbol{a}$ (a function of $\boldsymbol{a}$ alone), and setting its gradient to zero gives the dual solution

$$\boldsymbol{a} = (K + \lambda I)^{-1} \boldsymbol{t}. \tag{14.2}$$

A prediction at a new point $\boldsymbol{x}$ is $\boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{a}^\top \boldsymbol{\Phi} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_n a_n \, \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x})$, which again uses $\boldsymbol{\phi}$ only through inner products: collecting them into the vector $\boldsymbol{k}(\boldsymbol{x})$ with entries $k_n(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x})$,

$$h(\boldsymbol{x}) = \boldsymbol{k}(\boldsymbol{x})^\top (K + \lambda I)^{-1} \boldsymbol{t}. \tag{14.3}$$

The feature map has disappeared. Both fitting Eq. (14.2) and predicting Eq. (14.3) require only inner products of feature vectors, never the feature vectors themselves.

Representer theorem

For any model that fits $\boldsymbol{w}$ by minimizing a sum of per-example losses plus a penalty $\frac{\lambda}{2}\|\boldsymbol{w}\|^2$, the optimum lies in the span of the training features,

$$\boldsymbol{w}^\star = \sum_{n=1}^{N} a_n \, \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^\top \boldsymbol{a}. \tag{14.4}$$

The weight vector, however high-dimensional, is a combination of the N training feature vectors, so the search collapses from the M weights to the N coefficients $\boldsymbol{a}$. This is what makes the kernel trick possible: predictions $\boldsymbol{w}^{\star\top} \boldsymbol{\phi}(\boldsymbol{x}) = \sum_n a_n \, k(\boldsymbol{x}_n, \boldsymbol{x})$ then need only the kernel.

14.3 The kernel trick

Since the model needs only inner products, we define a *kernel function* that computes them,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}'), \tag{14.5}$$

and replace every inner product by a call to k. The Gram matrix is $K_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ and the prediction vector is $k_n(\boldsymbol{x}) = k(\boldsymbol{x}_n, \boldsymbol{x})$. The *kernel trick* is the realization that if k can be computed directly, in time that does not depend on the dimension of $\boldsymbol{\phi}$, then we never form $\boldsymbol{\phi}$ at all and the high-dimensional space costs nothing. The payoff is largest when $\boldsymbol{\phi}$ is huge or infinite but k is a simple formula, which, remarkably, is the common case.

Example 14.1 (Dual equals primal: kernel ridge by hand). Use the linear kernel $k(x, x') = xx'$ (the feature map is the identity, $\phi(x) = x$) on one-dimensional data with two training points $x_1 = 1$, $x_2 = 2$ and targets $t_1 = 1$, $t_2 = 3$, with regularization $\lambda = 1$. The Gram matrix collects the kernel values, $K = \left[\begin{smallmatrix} k(1,1) & k(1,2) \\ k(2,1) & k(2,2) \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 2 \\ 2 & 4 \end{smallmatrix}\right]$. Then $K + \lambda I = \left[\begin{smallmatrix} 2 & 2 \\ 2 & 5 \end{smallmatrix}\right]$, with determinant $2 \cdot 5 - 2 \cdot 2 = 6$ and inverse $\frac{1}{6}\left[\begin{smallmatrix} 5 & -2 \\ -2 & 2 \end{smallmatrix}\right]$. The dual coefficients are

$$\boldsymbol{a} = (K + \lambda I)^{-1}\boldsymbol{t} = \tfrac{1}{6}\begin{bmatrix} 5 & -2 \\ -2 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 3 \end{bmatrix} = \tfrac{1}{6}\begin{bmatrix} 5-6 \\ -2+6 \end{bmatrix} = \tfrac{1}{6}\begin{bmatrix} -1 \\ 4 \end{bmatrix} = (-0.167,\ 0.667).$$

To predict at $x_* = 1.5$, the kernel vector is $\boldsymbol{k}(x_*) = (k(1, 1.5), k(2, 1.5)) = (1.5, 3)$, so $h(1.5) = \boldsymbol{k}(x_*)^\top \boldsymbol{a} = (1.5)(-0.167) + (3)(0.667) = -0.25 + 2.0 = 1.75$. As a check, solve the same problem in the primal: ridge with feature x gives weight $w = (\sum_n x_n t_n)/(\sum_n x_n^2 + \lambda) = (1 \cdot 1 + 2 \cdot 3)/(1 + 4 + 1) = 7/6 = 1.167$, and $h(1.5) = (1.167)(1.5) = 1.75$. The dual and primal agree exactly, as they must; the dual just reached the answer through inner products. ◊

14.4 Mercer's theorem and valid kernels

The kernel trick is usually run in reverse: rather than choosing a feature map and deriving its kernel, we choose a kernel directly and trust that *some* feature map underlies it. When is a function $k(\boldsymbol{x}, \boldsymbol{x}')$ a legitimate inner product in some feature space? The answer is Mercer's condition.

Theorem 14.2 (Mercer's condition). A symmetric function $k(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel, meaning $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$ for some feature map $\boldsymbol{\phi}$, if and only if for every finite set of points the Gram matrix K with $K_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ is symmetric and positive semidefinite.

The necessity is easy and worth seeing: if $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$, then for any vector $\boldsymbol{c}$,

$$\boldsymbol{c}^\top K \boldsymbol{c} = \sum_{n,m} c_n c_m\, \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m) = \Big\| \sum_n c_n \boldsymbol{\phi}(\boldsymbol{x}_n) \Big\|^2 \geq 0,$$

because it is the squared norm of the vector $\sum_n c_n \boldsymbol{\phi}(\boldsymbol{x}_n)$, so K is positive semidefinite. Mercer's theorem supplies the converse: positive semidefiniteness is also sufficient for a feature map to exist. The practical consequence is that we may invent kernels freely, checking only that the Gram matrix is always positive semidefinite, and a feature space is guaranteed without our ever constructing it. New valid kernels can also be built from old ones: sums of kernels, products of kernels, and a kernel scaled by a positive constant are all valid kernels, which lets one compose complex similarity measures from simple parts.

14.5 The kernel catalogue

A few kernels do most of the work.

The *linear kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$ is the trivial case, with $\boldsymbol{\phi}$ the identity; it recovers ordinary linear methods.

The *polynomial kernel* $k(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$ corresponds to a feature map of all monomials up to degree d, and computing it costs one inner product and a power, regardless of how many monomial features that implies. The correspondence is worth seeing explicitly.

Example 14.3 (The polynomial kernel's feature map). Take the homogeneous degree-two kernel $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ in two dimensions. Expanding the square,

$$(\boldsymbol{x}^\top \boldsymbol{z})^2 = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2.$$

Group the terms as an inner product by matching each x-factor with its z-factor:

$$= (x_1^2)(z_1^2) + (\sqrt{2}\, x_1 x_2)(\sqrt{2}\, z_1 z_2) + (x_2^2)(z_2^2) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z}), \quad \boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2,\ \sqrt{2}\, x_1 x_2,\ x_2^2).$$

So $(\boldsymbol{x}^\top \boldsymbol{z})^2$ is exactly the inner product of the three-dimensional feature vectors $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$, with the $\sqrt{2}$ appearing to make the cross term's coefficient come out right. Computing k takes a two-dimensional dot product and a square; computing $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$ directly takes a three-dimensional dot product. The saving is tiny here but becomes decisive at higher degree and dimension, where the feature space is vast and the kernel is still one dot product and one power. ◊

Kernel	$k(\boldsymbol{x}, \boldsymbol{x}')$	**Feature space**
Linear	$\boldsymbol{x}^\top \boldsymbol{x}'$	the inputs themselves
Polynomial	$(\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$	monomials up to degree d
Gaussian (RBF)	$\exp(-\lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2 / 2\sigma^2)$	infinite-dimensional

Table 14.1. Each computes an inner product in a feature space without forming it. The polynomial and RBF kernels give cheap access to very high or infinite dimensional spaces, the source of their power.

The *Gaussian* or *radial basis function* (RBF) kernel,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp\Big(-\frac{\lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2}{2\sigma^2}\Big), \tag{14.6}$$

is the most widely used. It measures similarity: near 1 when $\boldsymbol{x}$ and $\boldsymbol{x}'$ are close and decaying to 0 as they separate, with the bandwidth σ setting the reach. Its feature map is *infinite-dimensional* (expanding the exponential produces all degrees of monomials), so it could never be used in the primal, yet the kernel is a one-line formula. The RBF kernel turns any of the dual algorithms into a flexible nonlinear method that compares each new point to the training points by proximity.

The RBF kernel is worth seeing on numbers. Its Gram matrix has ones on the diagonal (every point is identical to itself) and entries that decay with distance off it. For the one-dimensional points $0, 1, 2$ with $\sigma = 1$, so $k(x, x') = \exp(-(x - x')^2/2)$,

$$K = \begin{bmatrix} 1 & e^{-1/2} & e^{-2} \\ e^{-1/2} & 1 & e^{-1/2} \\ e^{-2} & e^{-1/2} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0.607 & 0.135 \\ 0.607 & 1 & 0.607 \\ 0.135 & 0.607 & 1 \end{bmatrix},$$

neighbours scoring 0.607 and the far pair only 0.135. Plugging this kind of K into the dual solution (14.2) gives a genuinely nonlinear regressor.

Example 14.4 (RBF kernel ridge by hand). Fit kernel ridge with the RBF kernel ($\sigma = 1$) and $\lambda = 0.5$ to two points $x_1 = 0$, $x_2 = 2$ with targets $t_1 = 1$, $t_2 = -1$. The Gram matrix is $K = \left[\begin{smallmatrix} 1 & e^{-2} \\ e^{-2} & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 0.135 \\ 0.135 & 1 \end{smallmatrix}\right]$, so

$$K + \lambda I = \begin{bmatrix} 1.5 & 0.135 \\ 0.135 & 1.5 \end{bmatrix}, \quad \det = 1.5^2 - 0.135^2 = 2.232, \quad \boldsymbol{a} = (K + \lambda I)^{-1} \boldsymbol{t} = \begin{bmatrix} 0.733 \\ -0.733 \end{bmatrix}.$$

To predict at $x_* = 0.5$, form the kernel vector $\boldsymbol{k}(x_*) = (k(0, 0.5), k(2, 0.5)) = (e^{-0.125}, e^{-1.125}) = (0.882, 0.325)$, so

$$h(0.5) = \boldsymbol{k}(x_*)^\top \boldsymbol{a} = (0.882)(0.733) + (0.325)(-0.733) = 0.647 - 0.238 = 0.41.$$

The prediction leans positive because $x_* = 0.5$ is closer to the $+1$ point at 0 than to the -1 point at 2. At the midpoint $x_* = 1$, symmetry gives $\boldsymbol{k} = (0.607, 0.607)$ and $h(1) = 0.607(0.733) + 0.607(-0.733) = 0$: equidistant from the two opposite targets, the regressor sits exactly on the fence. The kernel never formed the (infinite-dimensional) feature map. ◊

The RBF kernel is also what makes a linear method handle data wrapped in circles (Fig. 14.2), the case no straight boundary in the input plane can solve.

Remark 14.5 *(Where kernels go next).* The dual view applies to far more than ridge regression. The perceptron of Chapter 7 kernelizes (its weight vector is a sum of training features, so its predictions are sums of kernels), and most importantly the support vector machine of Chapter 15 is naturally a dual, kernelized method whose decision depends only on inner products. There is also a Bayesian end of this story: a Gaussian process is what Bayesian linear regression (Chapter 6) becomes when its equivalent kernel is replaced by an arbitrary kernel, giving nonparametric Bayesian regression with calibrated uncertainty. The inner-product-as-similarity idea even echoes in the attention of Chapter 13, where $\boldsymbol{q}^\top \boldsymbol{k}$ scores how much one token attends to another.

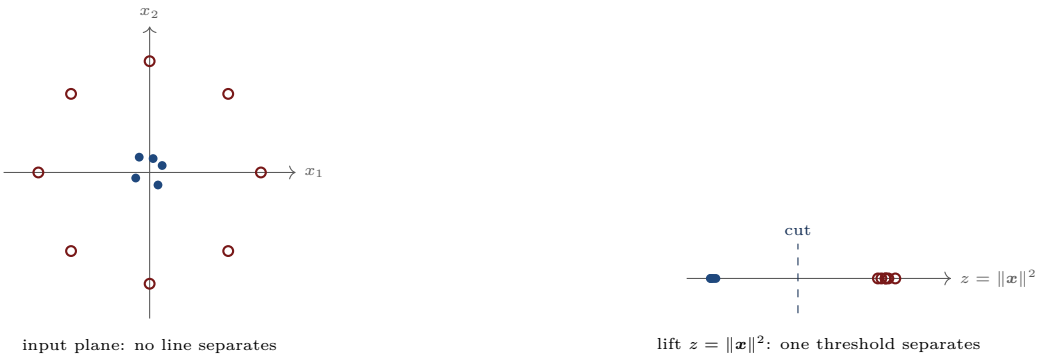


Figure 14.2. An inner cluster (blue) ringed by an outer class (red): no straight line separates them in the plane. Lifting each point to $z = \|\boldsymbol{x}\|^2$ (which the RBF and polynomial kernels do implicitly) sends the inner class to small z and the outer class to large z, where a single threshold separates them.

14.6 The kernel perceptron

Ridge regression is not the only method the dual view kernelizes. The perceptron of Chapter 7 updates $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$ on each misclassified point, so after training $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{\phi}(\boldsymbol{x}_n)$, where α_n counts how many times point n was used to correct the weights. The weight vector never needs to be formed: both the prediction and the update reach $\boldsymbol{\phi}$ only through inner products, hence through a kernel.

The kernel perceptron

Initialize $\alpha_n = 0$ for all n. Cycle through the data; for each point n, if it is misclassified, $y_n \sum_m \alpha_m y_m \, k(\boldsymbol{x}_m, \boldsymbol{x}_n) \le 0$, increment $\alpha_n \leftarrow \alpha_n + 1$. Repeat until no point is misclassified. Predict with $g(\boldsymbol{x}) = \operatorname{sign}\big(\sum_n \alpha_n y_n \, k(\boldsymbol{x}_n, \boldsymbol{x})\big)$.

With a nonlinear kernel this learns boundaries the plain perceptron cannot. The quadratic kernel $(\boldsymbol{x}^\top \boldsymbol{x}' + 1)^2$, whose feature map includes the product $x_1 x_2$, separates the XOR data that defeated the linear perceptron in Chapter 7, because XOR is linear in that one extra feature. The kernel turned an unsolvable problem into a solvable one without the algorithm changing at all.

Example 14.6 (A kernel-perceptron decision). Three training points $x_1 = 0$ $(y = +1)$, $x_2 = 3$ $(y = -1)$, $x_3 = 1$ $(y = +1)$, with the RBF kernel $(\sigma = 1)$ and, after training, counts $\boldsymbol{\alpha} = (1, 1, 1)$. Classify $x_* = 0.5$. The kernels are $k(0, 0.5) = e^{-0.125} = 0.882$, $k(3, 0.5) = e^{-3.125} = 0.044$, and $k(1, 0.5) = e^{-0.125} = 0.882$, so

$$g(0.5) = (1)(+1)(0.882) + (1)(-1)(0.044) + (1)(+1)(0.882) = 1.72 > 0,$$

classifying $x_* = 0.5$ as $+1$. It lies near the two positive points and far from the lone negative one, and the RBF kernel, weighting by proximity, agrees. ◊

Notes and References

The dual representation, the representer theorem, the kernel trick, Mercer's condition, and the kernel catalogue follow Sections 6.1 and 6.2 of Bishop [3]. The kernel idea entered machine learning through the support vector machine of Boser et al. [4] and Cortes and Vapnik [7], the subject of Chapter 15. Mercer's theorem, on the conditions under which a symmetric function is an inner product in some space, is the classical result underlying Theorem 14.2. The Gaussian-process view mentioned in Remark 14.5 is the Bayesian counterpart, developed at length in Bishop [3], Chapter 6. The next chapter applies all of this to the maximum-margin classifier, where the kernel trick is at its most powerful.

Exercises

Exercise 14.1. Using the linear kernel $k(x, x') = xx'$, fit kernel ridge regression with $\lambda = 1$ to the points $x_1 = 0$, $x_2 = 2$ with targets $t_1 = 1$, $t_2 = 5$, and predict at $x_* = 1$. Verify against the primal ridge solution.

Solution. The Gram matrix is $K = \left[\begin{smallmatrix} 0\cdot 0 & 0\cdot 2 \\ 2\cdot 0 & 2\cdot 2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0 & 0 \\ 0 & 4 \end{smallmatrix}\right]$, so $K + \lambda I = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 5 \end{smallmatrix}\right]$ with inverse $\left[\begin{smallmatrix} 1 & 0 \\ 0 & 1/5 \end{smallmatrix}\right]$. The dual coefficients are $\boldsymbol{a} = (K + \lambda I)^{-1}\boldsymbol{t} = (1 \cdot 1,\ (1/5) \cdot 5) = (1, 1)$. The kernel vector at $x_* = 1$ is $\boldsymbol{k} = (k(0, 1), k(2, 1)) = (0, 2)$, so $h(1) = \boldsymbol{k}^\top \boldsymbol{a} = 0 \cdot 1 + 2 \cdot 1 = 2$. Primal check: $w = (\sum x_n t_n)/(\sum x_n^2 + \lambda) = (0 + 10)/(0 + 4 + 1) = 10/5 = 2$, and $h(1) = 2 \cdot 1 = 2$. They agree. □

Exercise 14.2. Find the explicit feature map for the kernel $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z} + 1)^2$ in two dimensions, and state its dimension.

Solution. Expand $(\boldsymbol{x}^\top \boldsymbol{z} + 1)^2 = (x_1 z_1 + x_2 z_2 + 1)^2$. The terms are $x_1^2 z_1^2 + x_2^2 z_2^2 + 1 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2$. Matching x-factors with z-factors gives $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$ with $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2, x_2^2, \sqrt{2}\, x_1 x_2, \sqrt{2}\, x_1, \sqrt{2}\, x_2, 1)$, a six-dimensional feature map that includes all monomials of degree 0, 1, and 2 (the $+1$ inside the kernel is what adds the lower-degree terms). One can verify each squared-magnitude coefficient matches: the $\sqrt{2}$ factors reproduce the 2's on the cross and linear terms. □

Exercise 14.3. Prove that any valid kernel's Gram matrix is positive semidefinite, and explain why this makes positive semidefiniteness the criterion for inventing new kernels.

Solution. If $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$, then for any coefficients $\boldsymbol{c}$, $\boldsymbol{c}^\top K \boldsymbol{c} = \sum_{n,m} c_n c_m\, \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m) = \|\sum_n c_n \boldsymbol{\phi}(\boldsymbol{x}_n)\|^2 \geq 0$, the squared norm of a vector, so K is positive semidefinite (and it is symmetric since k is). Mercer's theorem gives the converse: any symmetric function whose Gram matrix is always positive semidefinite is the inner product of some feature map. So to invent a kernel we need only guarantee its Gram matrices are positive semidefinite; the feature space then exists automatically and need never be constructed. □

Exercise 14.4. Fit kernel ridge with the RBF kernel ($\sigma = 1$) and $\lambda = 1$ to the points $x_1 = -1$, $x_2 = 1$ with targets $t_1 = 2$, $t_2 = 0$. Give the Gram matrix, the dual coefficients $\boldsymbol{a}$, and the prediction at $x_* = 0$. Use $e^{-2} = 0.135$, $e^{-1/2} = 0.607$.

Solution. The distance between the points is 2, so $K = \left[\begin{smallmatrix} 1 & e^{-2} \\ e^{-2} & 1 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 0.135 \\ 0.135 & 1 \end{smallmatrix}\right]$ and $K + \lambda I = \left[\begin{smallmatrix} 2 & 0.135 \\ 0.135 & 2 \end{smallmatrix}\right]$ with determinant $4 - 0.018 = 3.982$. Then $\boldsymbol{a} = (K + \lambda I)^{-1}\boldsymbol{t} = \frac{1}{3.982}\left[\begin{smallmatrix} 2 & -0.135 \\ -0.135 & 2 \end{smallmatrix}\right]\left[\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right] = \frac{1}{3.982}(4, -0.27) = (1.005, -0.068)$. At $x_* = 0$ both points are distance 1 away, so $\boldsymbol{k} = (e^{-1/2}, e^{-1/2}) = (0.607, 0.607)$ and $h(0) = 0.607(1.005) + 0.607(-0.068) = 0.610 - 0.041 = 0.57$, leaning toward the $+2$ point. □

Exercise 14.5. Given two valid kernels k_1 and k_2, prove that $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$ is a valid kernel, and that $c\, k_1$ is valid for any constant $c > 0$. Use these rules to argue that $(\boldsymbol{x}^\top \boldsymbol{x}' + 1)^3$ is a valid kernel.

Solution. By Mercer (Theorem 14.2) a kernel is valid iff every Gram matrix is positive semidefinite. If K_1, K_2 are the Gram matrices of k_1, k_2 on any point set, the Gram matrix of $k_1 + k_2$ is $K_1 + K_2$, and for any $\boldsymbol{c}$, $\boldsymbol{c}^\top (K_1 + K_2)\boldsymbol{c} = \boldsymbol{c}^\top K_1 \boldsymbol{c} + \boldsymbol{c}^\top K_2 \boldsymbol{c} \geq 0$ since each term is ≥ 0; so the sum is valid. For $c > 0$, $\boldsymbol{c}^\top (cK_1)\boldsymbol{c} = c(\boldsymbol{c}^\top K_1 \boldsymbol{c}) \geq 0$, so ck_1 is valid. Products of kernels are also valid (the Gram matrix is the Hadamard product, which preserves positive semidefiniteness). The linear kernel $\boldsymbol{x}^\top \boldsymbol{x}'$ is valid, so by the product rule $(\boldsymbol{x}^\top \boldsymbol{x}')^j$ is valid for each j, and $(\boldsymbol{x}^\top \boldsymbol{x}' + 1)^3 = \sum_{j=0}^{3} \binom{3}{j} (\boldsymbol{x}^\top \boldsymbol{x}')^j$ is a nonnegative combination of valid kernels, hence valid. □

Exercise 14.6. For the Gaussian kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2 / 2\sigma^2)$ with $\sigma = 1$, compute $k(\boldsymbol{x}, \boldsymbol{x}')$ for $\boldsymbol{x} = (0, 0)$ and $\boldsymbol{x}' = (1, 1)$, and describe how the value changes as the points move apart or as σ grows.

Solution. The squared distance is $\|\boldsymbol{x} - \boldsymbol{x}'\|^2 = (0-1)^2 + (0-1)^2 = 2$, so with $\sigma = 1$, $k = \exp(-2/2) = \exp(-1) = 0.368$. As the points move farther apart, $\|\boldsymbol{x} - \boldsymbol{x}'\|^2$ grows and k decays toward 0 (dissimilar points have near-zero kernel); as the points coincide, $k \to 1$ (maximal similarity). Increasing the bandwidth σ flattens the decay, so distant points

retain higher similarity and the kernel becomes smoother, while a small σ makes the kernel a sharp spike that treats all but the nearest points as dissimilar. □

CHAPTER 15

Support Vector Machines

Of all the lines that separate the data, take the one with the most room to spare on either side. Fit the widest street, not just any fence.

after Professor Peter Chin, ENGS 106

The perceptron of Chapter 7 stops at the first separating hyperplane it finds, and there are infinitely many. Some are reckless, skimming right past a data point; others leave a comfortable buffer. The *support vector machine* (SVM) picks the safest one: the hyperplane that maximizes the *margin*, the distance to the nearest point on either side. Maximizing the margin is the most robust choice, it generalizes best (a wide margin means a low-capacity classifier, in the sense of Chapter 4), and it leads to a beautiful optimization problem whose solution depends only on a handful of critical points and only through inner products, so the kernel trick of Chapter 14 applies in full. This chapter derives the margin, sets up the maximum-margin problem, solves it through Lagrangian duality and the KKT conditions, identifies the support vectors, extends to non-separable data with soft margins, and finally kernelizes the whole thing into one of the most effective classifiers in machine learning.

Roadmap

Suggested reading: Bishop [3], Section 7.1; Boyd and Vandenberghe [5] for the duality, summarized in Appendix C.

We cover: the margin and the maximum-margin problem (Section 15.1); the hard-margin primal (Section 15.2); Lagrangian duality and the dual problem (Section 15.3); support vectors and the KKT conditions (Section 15.4); soft margins and hinge loss (Section 15.5); and the kernel SVM (Section 15.6).

15.1 The margin

Take two linearly separable classes with labels $y_n \in \{-1, +1\}$ and a discriminant $y(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$, classifying by its sign. From Chapter 7, the signed distance from a point $\boldsymbol{x}_n$ to the boundary is $y(\boldsymbol{x}_n)/\|\boldsymbol{w}\|$, and the point is correctly classified when $y_n\, y(\boldsymbol{x}_n) > 0$, so the (positive) distance of a correctly classified point to the boundary is $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)/\|\boldsymbol{w}\|$. The *margin* is the distance from the boundary to the closest point, $\min_n y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)/\|\boldsymbol{w}\|$, and we want to choose $\boldsymbol{w}, b$ to make it as large as possible.

This looks awkward because $\boldsymbol{w}$ and b can be rescaled freely without moving the boundary: multiplying both by any positive constant leaves $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ unchanged. We use that freedom to fix a convenient scale, the *canonical* form, by requiring the closest points to satisfy $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) = 1$. Then every point obeys $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$, the two classes are bounded by the hyperplanes $\boldsymbol{w}^\top \boldsymbol{x} + b = +1$ and $\boldsymbol{w}^\top \boldsymbol{x} + b = -1$, and the closest points sit exactly on them. The distance from the boundary to each of these is $1/\|\boldsymbol{w}\|$, so the full width of the margin, the street, is $2/\|\boldsymbol{w}\|$ (Fig. 15.1). Maximizing $2/\|\boldsymbol{w}\|$ is the same as minimizing $\|\boldsymbol{w}\|$, or, more conveniently, $\frac{1}{2}\|\boldsymbol{w}\|^2$.

Maximum margin equals minimum norm

With the canonical scaling $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$, the margin is $2/\|\boldsymbol{w}\|$, so maximizing the margin is minimizing $\frac{1}{2}\|\boldsymbol{w}\|^2$.

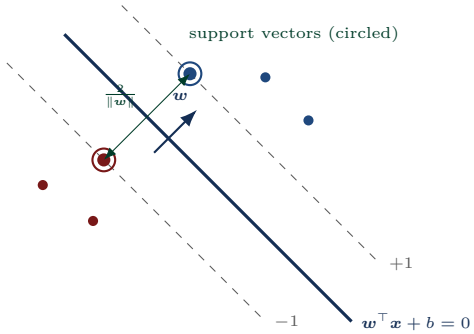


Figure 15.1. The SVM chooses the boundary with the widest street. With canonical scaling the margin hyperplanes are $\boldsymbol{w}^\top \boldsymbol{x} + b = \pm 1$, the street width is $2/\|\boldsymbol{w}\|$, and the few points touching the margin (circled) are the support vectors, the only points that matter.

15.2 The hard-margin primal

The maximum-margin classifier for separable data is therefore the solution of a constrained optimization problem, the *hard-margin SVM*:

$$\min_{\boldsymbol{w},b} \ \tfrac{1}{2}\|\boldsymbol{w}\|^2 \quad \text{subject to} \quad y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1, \quad n = 1, \dots, N. \tag{15.1}$$

The objective is convex (a quadratic) and the constraints are linear, so this is a convex quadratic program with a unique solution. We could hand it to a generic solver, but deriving its *dual* is far more illuminating, because the dual exposes the support vectors and the inner-product structure that makes kernels possible.

15.3 Lagrangian duality and the dual problem

Constrained optimization is handled by Lagrange multipliers, developed in full in Appendix C; we use the essentials here. Introduce a multiplier $\alpha_n \geq 0$ for each inequality constraint and form the *Lagrangian*

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \tfrac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{n=1}^{N} \alpha_n \big[y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) - 1 \big]. \tag{15.2}$$

The *dual function* is the minimum of $\mathcal{L}$ over the primal variables $\boldsymbol{w}, b$. Minimizing is easy because $\mathcal{L}$ is convex in them; set the derivatives to zero:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_n \alpha_n y_n \boldsymbol{x}_n = \boldsymbol{0} \implies \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n, \tag{15.3}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_n \alpha_n y_n = 0 \implies \sum_{n=1}^{N} \alpha_n y_n = 0. \tag{15.4}$$

The first equation is the representer theorem again: the optimal weight vector is a linear combination of the training inputs, weighted by $\alpha_n y_n$. Substitute Eqs. (15.3) and (15.4) back into Eq. (15.2). The norm term becomes $\frac{1}{2}\sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m$, the constraint term's b part vanishes by Eq. (15.4), and after collecting (the algebra is carried out in Exercise 15.2) the Lagrangian reduces to a function of $\boldsymbol{\alpha}$ alone. This is the dual problem.

The SVM dual

$$\max_{\boldsymbol{\alpha}} \ \sum_{n=1}^{N} \alpha_n - \tfrac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m \, y_n y_m \, \boldsymbol{x}_n^\top \boldsymbol{x}_m \quad \text{subject to} \quad \alpha_n \geq 0, \ \sum_{n=1}^{N} \alpha_n y_n = 0. \tag{15.5}$$

The dual is again a convex quadratic program, now in the N multipliers $\boldsymbol{\alpha}$, and crucially the data enters *only* through the inner products $\boldsymbol{x}_n^\top \boldsymbol{x}_m$. This is the opening the kernel trick needs (Section 15.6). The primal and dual have the same optimal value (strong duality holds, since the problem is convex and feasible), so solving either gives the margin; we solve the dual because of its inner-product form and because its solution reveals the support vectors.

15.4 Support vectors and the KKT conditions

At the optimum the Karush–Kuhn–Tucker (KKT) conditions hold, and the decisive one is *complementary slackness*: for every n,

$$\alpha_n \big[y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) - 1 \big] = 0. \tag{15.6}$$

This product is zero, so for each point either $\alpha_n = 0$ or $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) = 1$. A point strictly outside the margin has $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) > 1$, which forces $\alpha_n = 0$: it contributes nothing to $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$. Only the points *on* the margin, where $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) = 1$, can have $\alpha_n > 0$. These are the *support vectors*, and they alone determine the solution:

$$\boldsymbol{w} = \sum_{n \in \mathrm{SV}} \alpha_n y_n \boldsymbol{x}_n. \tag{15.7}$$

The classifier is *sparse*: typically a small fraction of the data are support vectors, and the rest could be deleted without changing the boundary, the geometric reason being that only the closest points pin down the widest street. The bias b is recovered from any support vector by solving $y_s(\boldsymbol{w}^\top \boldsymbol{x}_s + b) = 1$, and a new point is classified by

$$h(\boldsymbol{x}) = \operatorname{sign}\Big(\sum_{n \in \mathrm{SV}} \alpha_n y_n \, \boldsymbol{x}_n^\top \boldsymbol{x} + b \Big), \tag{15.8}$$

again only through inner products.

Example 15.1 (A two-point SVM solved completely). Two points, $\boldsymbol{x}_1 = (1, 1)$ with $y_1 = +1$ and $\boldsymbol{x}_2 = (-1, -1)$ with $y_2 = -1$. By the symmetry of the configuration the boundary passes through the origin ($b = 0$) and $\boldsymbol{w}$ points along $(1, 1)$, so write $\boldsymbol{w} = (w, w)$. The canonical constraint on the positive point, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) = 1$, reads $(1)\big[(w)(1) + (w)(1) + 0\big] = 2w = 1$, so $w = 0.5$ and $\boldsymbol{w} = (0.5, 0.5)$. Check the negative point: $y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) = (-1)\big[(0.5)(-1) + (0.5)(-1)\big] = (-1)(-1) = 1$, satisfied with equality, so both points are support vectors. The margin is $2/\|\boldsymbol{w}\| = 2/\sqrt{0.5^2 + 0.5^2} = 2/\sqrt{0.5} = 2.83$, and $\tfrac{1}{2}\|\boldsymbol{w}\|^2 = \tfrac{1}{2}(0.5) = 0.25$.

Now the dual. By symmetry $\alpha_1 = \alpha_2 = \alpha$, and the constraint $\sum_n \alpha_n y_n = \alpha(1) + \alpha(-1) = 0$ holds automatically. From $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n = \alpha(1)(1, 1) + \alpha(-1)(-1, -1) = 2\alpha(1, 1)$, matching $\boldsymbol{w} = (0.5, 0.5)$ gives $2\alpha = 0.5$, so $\alpha = 0.25$. The dual objective Eq. (15.5) evaluates, with $\boldsymbol{x}_1^\top \boldsymbol{x}_1 = 2$, $\boldsymbol{x}_2^\top \boldsymbol{x}_2 = 2$, $\boldsymbol{x}_1^\top \boldsymbol{x}_2 = -2$, to

$$\sum_n \alpha_n - \tfrac{1}{2} \sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m = (0.25 + 0.25) - \tfrac{1}{2}\big[(0.0625)(2) + (0.0625)(2) + 2(0.0625)(-1)(-2)\big],$$

where the last bracket is $0.125 + 0.125 + 0.25 = 0.5$, so the dual value is $0.5 - \tfrac{1}{2}(0.5) = 0.5 - 0.25 = 0.25$. This equals the primal value $\tfrac{1}{2}\|\boldsymbol{w}\|^2 = 0.25$, confirming strong duality: primal and dual optima coincide. The boundary is $x_1 + x_2 = 0$, both points are support vectors, and the multipliers are $\alpha_1 = \alpha_2 = 0.25$. ◊

15.5 Soft margins and hinge loss

Real data is rarely perfectly separable, and even when it is, a single outlier can force a absurdly narrow margin. The *soft-margin* SVM allows violations, charging for them. Introduce a *slack* variable $\xi_n \geq 0$ for each point, measuring how far it intrudes past its margin, and relax the constraint to $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$:

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \ \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{n=1}^{N} \xi_n \quad \text{subject to} \quad y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n, \ \ \xi_n \geq 0. \tag{15.9}$$

A point with $\xi_n = 0$ is outside its margin (fine); $0 < \xi_n < 1$ is inside the margin but still correctly classified; $\xi_n > 1$ is misclassified. The constant C tunes the tradeoff: large C punishes violations heavily and approaches the hard margin, while small C tolerates violations for a wider, simpler margin, a regularization knob like λ of Chapter 5 (in fact $\tfrac{1}{2}\|\boldsymbol{w}\|^2$ is exactly an ℓ_2 penalty). The dual is the same as Eq. (15.5) with one change: the multipliers gain an upper bound, $0 \leq \alpha_n \leq C$, a *box constraint* that caps how much any one point can influence the solution, limiting the damage an outlier can do. Figure 15.2 shows what the slack variables measure.

The soft-margin objective has an illuminating rewrite. The optimal slack is $\xi_n = \max(0, 1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b))$: zero if the margin is met, and the shortfall otherwise. Substituting, the SVM minimizes

$$\tfrac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{n=1}^{N} \max\big(0, \ 1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)\big), \tag{15.10}$$

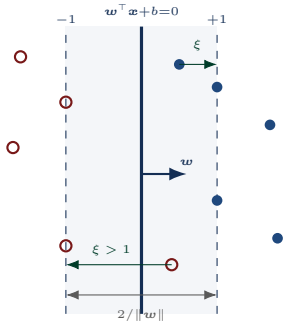


Figure 15.2. The boundary and its ± 1 margins. Points beyond their margin have $\xi = 0$ (the support vectors sit exactly on the dashed lines). A correctly classified point inside the margin has $0 < \xi < 1$ (top, blue), and a misclassified point on the wrong side has $\xi > 1$ (bottom, red); each ξ measures the shortfall to that point's own margin. The cost $C \sum_n \xi_n$ charges for these violations.

Figure 15.3. Both are convex upper bounds on the 0–1 misclassification loss, plotted against the margin $y\, f(\boldsymbol{x})$. The hinge loss is exactly zero once a point is correctly classified beyond the margin ($yf \geq 1$), which makes the SVM solution sparse; the logistic loss never quite reaches zero, so logistic regression uses every point.

a regularizer plus the *hinge loss* $\max(0, 1 - y\, f(\boldsymbol{x}))$. The hinge loss is zero once a point is correctly classified *with margin* (beyond the $+1$ line) and rises linearly as the point falls short. It is the SVM's answer to the cross-entropy loss of logistic regression (Chapter 8): both are convex upper bounds on the misclassification rate, but the hinge loss is exactly zero for confident correct points (giving the sparse support-vector solution), whereas the logistic loss is always slightly positive (Fig. 15.3). This is the loss-function view of the difference between the two classifiers.

15.6 The kernel SVM

Here the chapters converge. The SVM dual Eq. (15.5) and the classifier Eq. (15.8) use the data only through inner products $\boldsymbol{x}_n^\top \boldsymbol{x}_m$ and $\boldsymbol{x}_n^\top \boldsymbol{x}$. By the kernel trick of Chapter 14, replace every inner product with a kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$, and the SVM operates in the high-dimensional feature space without ever entering it. The dual becomes

$$\max_{\boldsymbol{\alpha}} \sum_n \alpha_n - \tfrac{1}{2} \sum_{n,m} \alpha_n \alpha_m y_n y_m \, k(\boldsymbol{x}_n, \boldsymbol{x}_m), \quad 0 \le \alpha_n \le C, \; \sum_n \alpha_n y_n = 0, \tag{15.11}$$

and the classifier is $h(\boldsymbol{x}) = \operatorname{sign}\big(\sum_{n \in \mathrm{SV}} \alpha_n y_n \, k(\boldsymbol{x}_n, \boldsymbol{x}) + b\big)$. With a polynomial or Gaussian kernel the decision boundary becomes nonlinear in the original space, a linear boundary in the feature space pulled back into a curve, and the SVM can separate classes no hyperplane could. The kernel SVM combines three ideas at their best: the maximum margin for robustness, the dual's sparsity for efficient prediction (only support vectors appear), and the kernel trick for nonlinearity at the cost of a single linear classifier. It was, for over a decade, the most accurate general-purpose classifier available.

Solving the SVM dual as a quadratic program

The dual (15.11) is a quadratic program in $\boldsymbol{\alpha}$, which a standard solver (the course uses `cvxopt`) accepts in the form

$$\min_{\boldsymbol{\alpha}} \; \tfrac{1}{2} \boldsymbol{\alpha}^\top P \boldsymbol{\alpha} + q^\top \boldsymbol{\alpha} \quad \text{s.t.} \quad G\boldsymbol{\alpha} \le h, \quad A\boldsymbol{\alpha} = b.$$

Match the SVM dual term by term:

- $P_{nm} = y_n y_m \, k(\boldsymbol{x}_n, \boldsymbol{x}_m)$, the label-signed Gram matrix; $q = -\mathbf{1}$, turning the dual's $+\sum_n \alpha_n$ into a minimization;
- $G = \left[\begin{smallmatrix} -I \\ I \end{smallmatrix}\right]$, $h = \left[\begin{smallmatrix} \mathbf{0} \\ C\mathbf{1} \end{smallmatrix}\right]$, which encode $0 \le \alpha_n \le C$;
- $A = \boldsymbol{y}^\top$, $b = 0$, which encode $\sum_n \alpha_n y_n = 0$.

The solver returns $\boldsymbol{\alpha}$; the support vectors are the entries with $\alpha_n > 0$, and b is recovered from any support vector with $0 < \alpha_n < C$ via $y_s(\sum_m \alpha_m y_m k(\boldsymbol{x}_m, \boldsymbol{x}_s) + b) = 1$.

A final connection ties the chapter back to learning theory. A wider margin is a simpler classifier: one can show the VC dimension of margin-γ separators of data within a ball of radius R is bounded by R^2/γ^2, independent of the dimension of the feature space. So maximizing the margin directly minimizes a capacity bound, which is why the SVM generalizes well even when the kernel feature space is infinite-dimensional: it is the geometric realization of the bias-variance control of Chapter 4.

Notes and References

The maximum-margin derivation, the dual, the KKT conditions, support vectors, soft margins, and the kernel SVM follow Section 7.1 of Bishop [3]. The support vector machine is due to Boser et al. [4] (the maximum-margin classifier with kernels) and Cortes and Vapnik [7] (the soft margin). The Lagrangian-duality machinery is the subject of Boyd and Vandenberghe [5] and is summarized in Appendix C, where weak and strong duality and the KKT conditions are developed. The margin-based VC bound R^2/γ^2 connects this chapter to the learning theory of Chapter 4; the kernels are those of Chapter 14. In practice the dual quadratic program is solved by specialized methods such as sequential minimal optimization. With this chapter Part V is complete, and the inner-product view that runs through it returns once more, unexpectedly, in the attention mechanism of Chapter 13.

Exercises

Exercise 15.1. With the canonical scaling $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$, derive that the margin (the distance from the boundary to the nearest point) is $1/\|\boldsymbol{w}\|$ and the full street width is $2/\|\boldsymbol{w}\|$.

Solution. The signed distance from a point $\boldsymbol{x}$ to the hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ is $(\boldsymbol{w}^\top \boldsymbol{x} + b)/\|\boldsymbol{w}\|$. A nearest correctly classified point satisfies $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) = 1$ under canonical scaling, so its distance to the boundary is $|\boldsymbol{w}^\top \boldsymbol{x}_n + b|/\|\boldsymbol{w}\| = 1/\|\boldsymbol{w}\|$ (the numerator is 1 in magnitude since $y_n = \pm 1$). The nearest points of each class are at distance $1/\|\boldsymbol{w}\|$ on opposite sides, so the street between the margin hyperplanes $\boldsymbol{w}^\top \boldsymbol{x} + b = \pm 1$ has width $2/\|\boldsymbol{w}\|$. □

Exercise 15.2. Starting from the Lagrangian Eq. (15.2) and the stationarity conditions $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$ and $\sum_n \alpha_n y_n = 0$, derive the dual objective Eq. (15.5).

Solution. Substitute $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$ into $\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_n \alpha_n [y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) - 1]$. The norm term is $\frac{1}{2}\boldsymbol{w}^\top \boldsymbol{w} = \frac{1}{2}\sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m$. In the constraint term, $\sum_n \alpha_n y_n \boldsymbol{w}^\top \boldsymbol{x}_n = \boldsymbol{w}^\top \sum_n \alpha_n y_n \boldsymbol{x}_n = \boldsymbol{w}^\top \boldsymbol{w} = \sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m$; the term $\sum_n \alpha_n y_n b = b \sum_n \alpha_n y_n = 0$ by stationarity in b; and $\sum_n \alpha_n$ remains. So $\mathcal{L} = \frac{1}{2}\sum_{n,m}(\cdots) - \sum_{n,m}(\cdots) + \sum_n \alpha_n = \sum_n \alpha_n - \frac{1}{2}\sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m$, the dual objective, maximized subject to $\alpha_n \geq 0$ and $\sum_n \alpha_n y_n = 0$. □

Exercise 15.3. Solve the SVM for $\boldsymbol{x}_1 = (2, 0)$ with $y_1 = +1$ and $\boldsymbol{x}_2 = (0, 0)$ with $y_2 = -1$. Find $\boldsymbol{w}$, b, the margin, and the multipliers, and verify strong duality.

Solution. By geometry the boundary is vertical, midway between the points, so $\boldsymbol{w} = (w, 0)$. The canonical constraints $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) = 1$ and $y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) = 1$ read $2w + b = 1$ and $-(0 + b) = 1$, so $b = -1$ and $2w - 1 = 1$, giving $w = 1$, $\boldsymbol{w} = (1, 0)$. The margin is $2/\|\boldsymbol{w}\| = 2/1 = 2$ (the points are distance 2 apart, each 1 from the boundary $x_1 = 1$). Both points are support vectors. For the multipliers, $\sum_n \alpha_n y_n = \alpha_1 - \alpha_2 = 0$ so $\alpha_1 = \alpha_2 = \alpha$, and $\boldsymbol{w} = \alpha_1 y_1 \boldsymbol{x}_1 + \alpha_2 y_2 \boldsymbol{x}_2 = \alpha(2, 0) + 0 = (2\alpha, 0) = (1, 0)$, so $\alpha = 0.5$. The primal value is $\frac{1}{2}\|\boldsymbol{w}\|^2 = 0.5$. The dual value is $\sum \alpha_n - \frac{1}{2}\sum_{n,m} \alpha_n \alpha_m y_n y_m \boldsymbol{x}_n^\top \boldsymbol{x}_m = (0.5 + 0.5) - \frac{1}{2}(0.25)(1)(4) = 1 - 0.5 = 0.5$ (only the $\boldsymbol{x}_1^\top \boldsymbol{x}_1 = 4$ term is nonzero since $\boldsymbol{x}_2 = \boldsymbol{0}$). Primal equals dual at 0.5, confirming strong duality. □

Exercise 15.4. Compute the hinge loss $\max(0, 1 - yf)$ for a point with margin $yf(\boldsymbol{x}) = 1.5$, for one with $yf(\boldsymbol{x}) = 0.3$, and for one with $yf(\boldsymbol{x}) = -0.5$. Explain why points of the first kind do not affect the SVM solution.

Solution. For $yf = 1.5$: $\max(0, 1 - 1.5) = \max(0, -0.5) = 0$. For $yf = 0.3$: $\max(0, 1 - 0.3) = 0.7$. For $yf = -0.5$ (misclassified): $\max(0, 1 - (-0.5)) = 1.5$. A point with $yf \geq 1$, like the first, incurs zero loss and zero gradient, so it exerts no force on the solution; in the dual its multiplier is $\alpha_n = 0$ and it is not a support vector. Only points with $yf \leq 1$ (on or inside the margin) contribute, which is why the SVM solution depends on just the support vectors. □

Exercise 15.5. A trained SVM gives four points the signed scores $f(\boldsymbol{x}) = \boldsymbol{w}^\top \boldsymbol{x} + b$ equal to 1.0, 0.5, −0.5, 0.4, with labels $y = +1, +1, -1, -1$ respectively. Compute the margin quantity yf, the slack $\xi = \max(0, 1 - yf)$, and the total hinge loss $\sum_n \xi_n$. Which point is misclassified?

Solution. Compute yf then $\xi = \max(0, 1 - yf)$ for each: point 1, $yf = (1)(1.0) = 1.0$, $\xi = 0$; point 2, $yf = (1)(0.5) = 0.5$, $\xi = 0.5$; point 3, $yf = (-1)(-0.5) = 0.5$, $\xi = 0.5$; point 4, $yf = (-1)(0.4) = -0.4$, $\xi = 1 - (-0.4) = 1.4$. The total hinge loss is $0 + 0.5 + 0.5 + 1.4 = 2.4$. Point 4 has $\xi > 1$, so it lies on the wrong side of the boundary and is misclassified; points 2 and 3 are correct but inside the margin ($0 < \xi < 1$); point 1 sits exactly on its margin ($\xi = 0$). □

Exercise 15.6. Explain the role of the parameter C in the soft-margin SVM Eq. (15.9), and what happens to the solution as $C \to \infty$ and as $C \to 0$.

Solution. C weights the total slack $\sum_n \xi_n$ (the margin violations) against the margin width term $\frac{1}{2}\|\boldsymbol{w}\|^2$. As $C \to \infty$, violations are punished so heavily that the optimizer drives all $\xi_n \to 0$ if possible, recovering the hard-margin SVM that

tolerates no violations (and fails if the data is not separable). As $C \to 0$, violations are nearly free, so the optimizer minimizes $\|\boldsymbol{w}\|^2$ almost unconstrained, giving a very wide margin that ignores the data and underfits. Intermediate C, chosen by cross-validation, balances margin width against training errors, exactly the regularization tradeoff of Chapter 4. □

PART VI

Graphical Models

CHAPTER 16

Probabilistic Graphical Models

Draw the dependencies as a graph, and the graph tells you what you can ignore. That is the whole economy of graphical models.

after Professor Peter Chin, ENGS 106

A joint distribution over many variables is, in the raw, hopeless. With n binary variables the joint $p(x_1, \dots, x_n)$ has $2^n - 1$ free numbers; at $n = 100$ that exceeds the number of atoms in a person. Yet we reason about such distributions all the time, because the variables are rarely all entangled: most pairs are conditionally independent given a few others. A *probabilistic graphical model* draws those dependencies as a graph and uses the graph's structure to factor the joint into small, local pieces, collapsing the parameter count from exponential to manageable and making the conditional independencies readable at a glance. There are two languages. *Directed* graphs (Bayesian networks) encode a generative story, each variable produced from its parents, and are natural for causal or sequential models. *Undirected* graphs (Markov random fields) encode soft mutual constraints and are natural for spatial models like images. This chapter develops both, the independence rules that let you read a graph, and an image-denoising application; Chapter 17 then computes with them.

Roadmap

Suggested reading: Bishop [3], Sections 8.1–8.3; Koller and Friedman [22] for the full theory.

We cover: factorization and the parameter saving (Section 16.1); Bayesian networks (Section 16.2); the three canonical structures and explaining-away (Section 16.3); d-separation (Section 16.4); Markov random fields and Hammersley–Clifford (Section 16.5); and image denoising (Section 16.6).

16.1 Factorization and the parameter saving

The product rule lets us always factor a joint into a chain of conditionals, $p(x_1, \dots, x_n) = \prod_i p(x_i \mid x_1, \dots, x_{i-1})$, but with no structure each conditional depends on all earlier variables and nothing is saved. Structure is conditional independence: if x_i depends on only a few of its predecessors, its conditional shrinks. A graphical model is a graph whose nodes are the variables and whose edges record exactly these dependencies, so that the joint factors into small local terms. The saving can be enormous: a chain of n binary variables in which each depends only on the previous one needs $1 + 2(n-1)$ numbers instead of $2^n - 1$. The art is choosing the graph that matches the problem's true dependencies.

16.2 Bayesian networks

A *Bayesian network* is a directed acyclic graph in which each node x_i has a set of parents $\mathrm{pa}(x_i)$, the nodes with arrows into it, and the joint distribution factors as a product of each variable conditioned on its parents:

$$p(x_1, \dots, x_n) = \prod_{i=1}^{n} p\big(x_i \mid \mathrm{pa}(x_i)\big). \tag{16.1}$$

Each factor is a small conditional distribution, a table or a parametric form, attached to its node. The arrows carry a generative reading: to sample from the model, draw the parentless nodes first, then each node from its conditional given its already-sampled parents.

Example 16.1 (A small Bayesian network). Consider three binary variables: R (it rained), S (the sprinkler ran), and W (the grass is wet). Rain and the sprinkler each make the grass wet, and suppose rain also makes the sprinkler less likely to run. The graph has edges $R \to S$, $R \to W$, $S \to W$, so $\mathrm{pa}(S) = \{R\}$ and $\mathrm{pa}(W) = \{R, S\}$, and the joint factors as

$$p(R, S, W) = p(R)\, p(S \mid R)\, p(W \mid R, S).$$

Counting parameters: $p(R)$ needs 1 number, $p(S \mid R)$ needs 2 (one per value of R), and $p(W \mid R, S)$ needs 4 (one per joint value of its two parents), so 7 in all, which here equals the full joint's $2^3 - 1 = 7$ because every variable is connected. The saving appears when variables are *not* all connected: if W depended on R alone (no $S \to W$ edge), $p(W \mid R)$ would need only 2 numbers and the total would drop. The missing edges are the savings. ◊

To do more than store the model, we answer queries from it. The next example puts numbers on the rain-sprinkler-wet network and computes a marginal and a posterior by enumeration.

Example 16.2 (Inference by enumeration). Give the network of Example 16.1 the conditional probability tables

$$p(R{=}1) = 0.2; \quad p(S{=}1 \mid R{=}0) = 0.4,\ p(S{=}1 \mid R{=}1) = 0.1;$$

(R, S)	$p(W{=}1 \mid R, S)$
$(0, 0)$	0.0
$(0, 1)$	0.8
$(1, 0)$	0.7
$(1, 1)$	0.95

(rain makes the sprinkler less likely; either one tends to wet the grass). *Is the grass wet?* Sum the joint over all four parent configurations:

$$p(W{=}1) = \sum_{R,S} p(W{=}1 \mid R, S)\, p(S \mid R)\, p(R) = 0 + \underbrace{0.8(0.4)(0.8)}_{0.256} + \underbrace{0.7(0.9)(0.2)}_{0.126} + \underbrace{0.95(0.1)(0.2)}_{0.019} = 0.401.$$

Given wet grass, did it rain? By Bayes' rule, keeping only the $R{=}1$ terms in the numerator, $p(R{=}1 \mid W{=}1) = (0.126 + 0.019)/0.401 = 0.145/0.401 = 0.36$: wet grass raises the chance of rain from the prior 0.2 to 0.36. *Now also learn the sprinkler ran* ($S{=}1$). Then

$$p(R{=}1 \mid W{=}1, S{=}1) = \frac{0.95(0.1)(0.2)}{0.8(0.4)(0.8) + 0.95(0.1)(0.2)} = \frac{0.019}{0.256 + 0.019} = 0.069.$$

Knowing the sprinkler ran drops the probability of rain from 0.36 back to 0.069: the sprinkler *explains away* the wet grass. W is a collider on the path $R \to W \leftarrow S$, so observing it couples its two parents. ◊

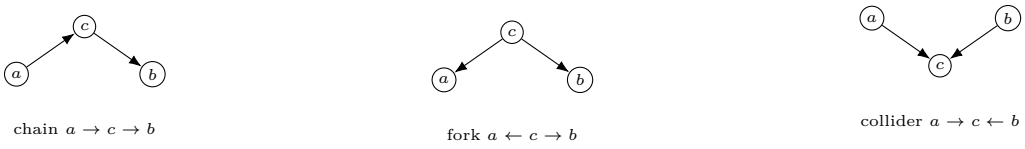


Figure 16.1. In the chain and fork, a and b are dependent but become independent once c is observed (conditioning blocks the path). In the collider, a and b are independent but become dependent once c is observed (conditioning opens the path), the explaining-away effect.

16.3 The three canonical structures

All conditional-independence reading of a Bayesian network is built from three three-node patterns (Fig. 16.1). Understanding each is understanding the whole language.

Chain $a \to c \to b$: the joint is $p(a)p(c \mid a)p(b \mid c)$. Here a and b are marginally dependent (information flows a to b through c), but conditioning on c makes them independent, $a \perp\!\!\!\perp b \mid c$: once you know the middle variable, the ends carry no further information about each other. Observing c *blocks* the path.

Fork $a \leftarrow c \to b$: the joint is $p(c)p(a \mid c)p(b \mid c)$, a common cause c of two effects. Again a and b are marginally dependent (both echo c) but conditionally independent given c, $a \perp\!\!\!\perp b \mid c$: the common cause explains their correlation, and fixing it decouples them. Observing c blocks the path.

Collider $a \to c \leftarrow b$: the joint is $p(a)p(b)p(c \mid a, b)$, two causes of a common effect. This one is opposite. Here a and b are marginally *independent* ($p(a, b) = p(a)p(b)$), but conditioning on the common effect c makes them *dependent*. Observing c *opens* the path. This is the famous *explaining-away* effect, and it is worth a worked example because it is the most counterintuitive fact in the subject.

Example 16.3 (Explaining away). Let X and Y be independent fair coins, $X, Y \in \{0, 1\}$ with $p(X{=}1) = p(Y{=}1) = \frac{1}{2}$, and let $Z = X \vee Y$ be their logical OR (the collider $X \to Z \leftarrow Y$), so $Z = 1$ unless both are 0. We compare $p(X{=}1 \mid Z{=}1)$ with $p(X{=}1 \mid Z{=}1, Y{=}1)$.

First $p(X{=}1 \mid Z{=}1)$. The event $Z = 1$ has probability $1 - p(X{=}0, Y{=}0) = 1 - \frac{1}{4} = \frac{3}{4}$. If $X = 1$ then $Z = 1$ automatically, so $p(X{=}1, Z{=}1) = p(X{=}1) = \frac{1}{2}$. Hence $p(X{=}1 \mid Z{=}1) = \frac{1}{2}/\frac{3}{4} = \frac{2}{3}$. Knowing the grass is wet (something turned Z on) raises the chance it was X from $\frac{1}{2}$ to $\frac{2}{3}$.

Now also learn $Y = 1$. Given $Y = 1$, the OR is satisfied no matter what X is, so $Z = 1$ tells us nothing more about X, and $p(X{=}1 \mid Z{=}1, Y{=}1) = p(X{=}1 \mid Y{=}1) = p(X{=}1) = \frac{1}{2}$, using $X \perp\!\!\!\perp Y$. The probability dropped from $\frac{2}{3}$ back to $\frac{1}{2}$. The second cause Y has *explained away* the evidence $Z = 1$: once we know Y alone accounts for the wet grass, the wet grass no longer implicates X. Two causes that were independent became negatively coupled once their shared effect was observed. ◊

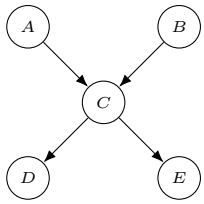


Figure 16.2. A network with a collider at C (from A and B) and two forks out of C (to D and E). Whether information flows between two nodes depends on which nodes are observed, through the three-structure rules applied along each path.

16.4 d-separation

The three structures combine into a general rule, *d-separation*, for reading any conditional independence off a Bayesian network. A path between two nodes is *blocked* by a set of observed nodes C if it passes through a chain or fork node that is in C, or through a collider node that is *not* in C and none of whose descendants are in C. Two sets of nodes A and B are *d-separated* given C, which implies $A \perp\!\!\!\perp B \mid C$, if every path between them is blocked. The rule simply applies the three-structure logic edge by edge: chains and forks transmit information unless their middle is observed, while colliders block information unless their bottom (or a descendant) is observed. A useful companion notion is the *Markov blanket* of a node, the set of its parents, children, and children's co-parents; conditioned on its Markov blanket, a node is independent of all other nodes in the network, so the blanket is everything you need to predict it.

Figure 16.2 and the example below put the rule to work on a five-node network.

Example 16.4 (d-separation queries). Read several independencies off Fig. 16.2, where $A \to C \leftarrow B$ is a collider and $D \leftarrow C \to E$ a fork.

- $A \perp\!\!\!\perp B$? The only path $A \to C \leftarrow B$ passes through the collider C, which is unobserved (and has no observed descendant), so the path is *blocked*: A and B are marginally independent.
- $A \perp\!\!\!\perp B \mid C$? Observing the collider C *opens* the path, so the independence fails: conditioning on C makes A and B dependent (explaining away).
- $D \perp\!\!\!\perp E \mid C$? The path $D \leftarrow C \to E$ runs through the fork C, and observing the fork node blocks it, so $D \perp\!\!\!\perp E \mid C$ holds. (Marginally, D and E are dependent, both echoing their common cause C.)
- $A \perp\!\!\!\perp E$? The path $A \to C \to E$ is a chain through the unobserved C, which transmits, so A and E are dependent; but $A \perp\!\!\!\perp E \mid C$, since observing the chain's middle blocks it.

The same node C helps or hurts depending on its role: observing it *opens* the collider path A–B but *closes* the fork path D–E and the chain path A–E. ◊

16.5 Markov random fields

When dependencies are mutual rather than causal, undirected graphs are the better language. A *Markov random field* (MRF) is an undirected graph in which the joint distribution factors over the graph's *cliques* (fully connected subsets of nodes). To each maximal clique C we attach a nonnegative *potential function* $\psi_C(\boldsymbol{x}_C)$, scoring how compatible the values of the clique's variables are, and the joint is the normalized product

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C} \psi_C(\boldsymbol{x}_C), \qquad Z = \sum_{\boldsymbol{x}} \prod_{C} \psi_C(\boldsymbol{x}_C), \tag{16.2}$$

where Z, the *partition function*, is the sum over all configurations that makes the distribution normalize. Unlike the directed case, the potentials are not conditional probabilities, just nonnegative compatibility scores, which is why the global normalizer Z is needed and why it is often the hard part to compute. Conditional independence in an MRF is read by simple *separation*: $A \perp\!\!\!\perp B \mid C$ whenever every path from A to B passes through C, with none of the collider subtlety of the directed case. The link between the factorization and the independence is the *Hammersley–Clifford theorem*: for strictly positive distributions, a distribution factors over the cliques of a graph if and only if it satisfies the conditional independencies that graph's separation encodes. The graph structure and the factorization are two views of the same thing.

16.6 Application: image denoising

MRFs shine where variables are laid out in space, and the cleanest example is cleaning up a noisy image. Suppose we observe a black-and-white image $\boldsymbol{y}$ in which each pixel $y_i \in \{-1, +1\}$ has been independently flipped with some probability, and we want to recover the clean image $\boldsymbol{x} \in \{-1, +1\}^N$. Two beliefs guide us: each recovered pixel should agree with its observation, and neighboring pixels should usually agree with each other (real images are mostly smooth). An MRF on the pixel grid encodes both through an *energy* that the recovered image should make small,

$$E(\boldsymbol{x}; \boldsymbol{y}) = -h \sum_i x_i - \beta \sum_{(i,j)\in\mathcal{E}} x_i x_j - \eta \sum_i x_i y_i, \tag{16.3}$$

with the distribution $p(\boldsymbol{x} \mid \boldsymbol{y}) \propto e^{-E(\boldsymbol{x};\boldsymbol{y})}$ (low energy means high probability). The middle term, summed over neighboring pixel pairs $\mathcal{E}$, rewards agreement between neighbors with strength β (the smoothness prior); the last term rewards agreement with the observation with strength η (the data fidelity); the first is an optional bias. Recovering the image means finding the $\boldsymbol{x}$ of lowest energy. Figure 16.3 draws the model: a grid of hidden pixels tied to their neighbours and to their noisy observations.

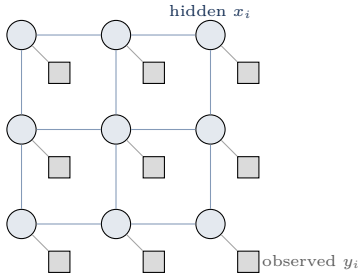


Figure 16.3. Each hidden pixel x_i (circle) connects to its grid neighbours through the smoothness term $\beta x_i x_j$ and to its noisy observation y_i (square) through the data term $\eta x_i y_i$. Denoising finds the hidden configuration of lowest energy, balancing agreement with neighbours against agreement with the data.

A simple and effective optimizer is *coordinate descent*, here called iterated conditional modes: start from $\boldsymbol{x} = \boldsymbol{y}$, then repeatedly visit each pixel and flip it if flipping lowers the total energy, sweeping until no flip helps. The change in energy from flipping pixel i depends only on i and its neighbors, so each step is cheap: x_i should be set to $+1$ or -1 according to whichever sign agrees better with the weighted vote of its neighbors and its observation. Concretely, flipping x_i changes the energy by

$$\Delta E_i = 2x_i \Big(h + \beta \sum_{j\in\mathcal{N}(i)} x_j + \eta\, y_i \Big), \tag{16.4}$$

because only the terms touching x_i change and each flips sign. The flip lowers the energy ($\Delta E_i < 0$) exactly when x_i disagrees with the neighbour-and-data vote $S_i = h + \beta \sum_j x_j + \eta y_i$. For instance, with $h = 0$, $\beta = \eta = 1$, a pixel $x_i = +1$ whose observation $y_i = -1$ (a noise flip) and whose four neighbours are all -1 has $S_i = (-4) + (-1) = -5$ and $\Delta E_i = 2(+1)(-5) = -10 < 0$, so it flips to -1, snapping back into agreement with its neighbours and shedding the noise. Each accepted flip strictly lowers the energy, and since there are finitely many configurations the process terminates at a local minimum, typically a clean image in which isolated noise flips have been smoothed away by their agreeing neighbors. This is a graphical model doing real work, and Chapter 17 takes up the general problem of inference in such models.

16.7 Cliques and the MRF factorization, worked

The factorization (16.2) is abstract until you carry a small graph through it, so this section does exactly that: read the maximal cliques off a graph, write the product of potentials, and then put numbers on a two-node field to compute the partition function Z and a marginal by hand. The recipe never changes. List the maximal cliques, attach one potential to each, multiply, and divide by the sum over all configurations.

A *clique* is a set of nodes that are all pairwise connected, and a clique is *maximal* if no node can be added without losing that property. The MRF factorizes over maximal cliques because any potential on a smaller clique can be folded into a potential on a maximal clique that contains it, so the maximal cliques are the coarsest useful pieces. Figure 16.4 shows a four-node graph whose maximal cliques are a triangle and a single edge.

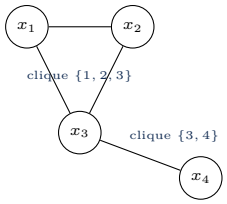


Figure 16.4. An undirected graph with edges x_1–x_2, x_2–x_3, x_3–x_1 (a triangle) and x_3–x_4. The triangle $\{x_1, x_2, x_3\}$ is one maximal clique; the edge $\{x_3, x_4\}$ is the other. Node x_3 sits in both, which is how the field couples the triangle to the pendant node.

The triangle $\{x_1, x_2, x_3\}$ is maximal: every pair inside it is joined, and adding x_4 would fail because there is no x_1–x_4 or x_2–x_4 edge. The pair $\{x_3, x_4\}$ is the only other maximal clique. With one potential per maximal clique, the joint of Fig. 16.4 factorizes as

$$p(\boldsymbol{x}) = \frac{1}{Z}\,\psi_{123}(x_1, x_2, x_3)\,\psi_{34}(x_3, x_4), \qquad Z = \sum_{\boldsymbol{x}} \psi_{123}(x_1, x_2, x_3)\,\psi_{34}(x_3, x_4). \tag{16.5}$$

The single edge $\{x_3, x_4\}$ gets its own potential rather than being absorbed into the triangle because x_4 is outside the triangle; only cliques that genuinely contain a pair share a potential.

Example 16.5 (A two-node MRF: Z and a marginal by hand). Take the simplest field that still needs a partition function: two binary nodes $x_1, x_2 \in \{0, 1\}$ joined by one edge. The graph has a single maximal clique $\{x_1, x_2\}$, so $p(x_1, x_2) = \frac{1}{Z}\,\psi_{12}(x_1, x_2)$ with the potential table

(x_1, x_2)	ψ_{12}
$(0, 0)$	2
$(0, 1)$	1
$(1, 0)$	1
$(1, 1)$	4

(the field prefers the two nodes to agree, and prefers the joint state $(1, 1)$ most). These are bare compatibility scores rather than probabilities, so they need not sum to one and we must normalize.

Partition function. Sum the potential over all four configurations:

$$Z = \psi(0, 0) + \psi(0, 1) + \psi(1, 0) + \psi(1, 1) = 2 + 1 + 1 + 4 = 8.$$

The normalized joint is each entry over 8: $p(0, 0) = \frac{2}{8} = 0.25$, $p(0, 1) = p(1, 0) = \frac{1}{8} = 0.125$, $p(1, 1) = \frac{4}{8} = 0.5$, and these four do sum to 1.

Marginal $p(x_1)$. Sum the joint over x_2:

$$p(x_1{=}0) = \frac{\psi(0, 0) + \psi(0, 1)}{Z} = \frac{2 + 1}{8} = \frac{3}{8} = 0.375, \qquad p(x_1{=}1) = \frac{\psi(1, 0) + \psi(1, 1)}{Z} = \frac{1 + 4}{8} = \frac{5}{8} = 0.625.$$

Check. The marginal sums to one, $\frac{3}{8} + \frac{5}{8} = \frac{8}{8} = 1$, as a marginal must. Node x_1 leans toward 1 because the configuration $(1, 1)$ carries the heaviest potential and drags probability mass with it. The whole computation is the meaning of (16.2): multiply the clique potentials, divide by their total Z, and sum out whatever you do not want. ◊

16.8 The Markov blanket, worked

Section 16.4 named the *Markov blanket* of a node, its parents, its children, and its children's other parents (the co-parents), and claimed that conditioning on it renders the node independent of everything else. This section identifies a blanket on a concrete network and shows why the claim holds, straight from the factorization.

Example 16.6 (Identifying a Markov blanket). Figure 16.5 has six binary variables with edges $A \to B$, $B \to D$, $C \to D$, $D \to F$, $E \to F$, so the joint factorizes as

$$p(A, B, C, D, E, F) = p(A)\,p(B \mid A)\,p(C)\,p(D \mid B, C)\,p(E)\,p(F \mid D, E).$$

Read the Markov blanket of D off the graph in three pieces.

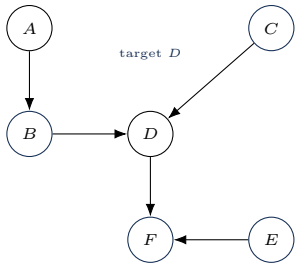


Figure 16.5. A six-node Bayesian network. The target D has parents $\{B, C\}$, one child F, and one co-parent E (the other parent of F). Its Markov blanket is $\{B, C, E, F\}$, circled; the grandparent A lies outside the blanket and reaches D only through B.

- *Parents* of D: the nodes with arrows into D, namely $\{B, C\}$.
- *Children* of D: the nodes D points to, namely $\{F\}$.
- *Co-parents*: the other parents of D's children. The child F has parents $\{D, E\}$, so the co-parent is $\{E\}$.

The Markov blanket is the union, $\mathrm{MB}(D) = \{B, C, E, F\}$. The only node left out is the grandparent A.

Why conditioning on the blanket isolates D. Group the factorization by which terms mention D. Only $p(D \mid B, C)$ and $p(F \mid D, E)$ contain D; the rest, namely $p(A)\,p(B \mid A)\,p(C)\,p(E)$, do not. Hence the conditional of D given everything else is

$$p\big(D \mid A, B, C, E, F\big) = \frac{p(D \mid B, C)\,p(F \mid D, E)}{\sum_{D'} p(D' \mid B, C)\,p(F \mid D', E)},$$

where every factor without D cancels between numerator and the normalizing sum. The right-hand side depends only on D and on $\{B, C, E, F\}$, the blanket; the value of A has disappeared. So once the blanket is observed, $D \perp\!\!\!\perp A \mid \{B, C, E, F\}$, and the same cancellation removes any node outside the blanket. Equivalently in d-separation terms, the path $A \to B \to D$ is a chain through the observed middle B, which blocks it, and no other path from A reaches D. The blanket is the smallest set that shields a node: it is everything you must know to predict D, and nothing past it adds information. ◊

Remark 16.7 *(The blanket in both graph languages).* The Markov blanket is the bridge between the directed and undirected pictures. In an MRF the blanket is simply a node's graph neighbors, since separation there has no collider subtlety. In a Bayesian network the blanket must also include the co-parents, because conditioning on a shared child opens the collider between a node and that child's other parents (the explaining-away of Example 16.3). The co-parents are exactly the price the directed language pays for colliders. A sampler that resamples one variable at a time, such as the Gibbs sampler of Chapter 20, needs only the blanket to compute each local update, which is why the blanket is the unit of computation in practice.

16.9 Variable elimination on a chain

Computing Z or a marginal by summing over every configuration costs $\mathcal{O}(2^N)$ for N binary nodes, which is hopeless past small N. *Variable elimination* exploits the clique factorization to push each sum inward past the factors that do not depend on the summed variable, turning the global sum into a sequence of small local sums. On a chain it is especially clean, and it previews the message passing of Chapter 17.

Example 16.8 (Eliminating variables on a four-node chain MRF). Take the chain MRF $x_1 - x_2 - x_3 - x_4$ on binary nodes $x_i \in \{0, 1\}$. The maximal cliques are the three edges, so

$$p(\boldsymbol{x}) = \frac{1}{Z}\,\psi_{12}(x_1, x_2)\,\psi_{23}(x_2, x_3)\,\psi_{34}(x_3, x_4).$$

Use the attractive pairwise potential M on the last two edges and a skewed potential P on the first, written as 2×2 tables indexed by $0, 1$:

$$\psi_{23} = \psi_{34} = M = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \qquad \psi_{12} = P = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}$$

(M rewards agreement; P further favors $x_1 = x_2 = 0$).

Partition function by elimination. Write $Z = \sum_{x_1}\sum_{x_2}\sum_{x_3}\sum_{x_4} \psi_{12}\psi_{23}\psi_{34}$ and sum from the far end inward, each time forming a *message* that summarizes the eliminated node.

Eliminate x_4. Only ψ_{34} touches x_4, so

$$m_4(x_3) = \sum_{x_4} \psi_{34}(x_3, x_4) = \big(\psi(0,0) + \psi(0,1),\ \psi(1,0) + \psi(1,1)\big) = (2{+}1,\ 1{+}2) = (3,3).$$

Eliminate x_3. Now combine ψ_{23} with the incoming message m_4:

$$m_3(x_2) = \sum_{x_3} \psi_{23}(x_2, x_3)\, m_4(x_3), \quad m_3(0) = 2(3) + 1(3) = 9, \quad m_3(1) = 1(3) + 2(3) = 9,$$

so $m_3 = (9, 9)$. *Eliminate x_2.* Combine ψ_{12} with m_3:

$$m_2(x_1) = \sum_{x_2} \psi_{12}(x_1, x_2)\, m_3(x_2), \quad m_2(0) = 3(9) + 1(9) = 36, \quad m_2(1) = 1(9) + 1(9) = 18.$$

Finally $Z = \sum_{x_1} m_2(x_1) = 36 + 18 = 54$.

Marginal of the endpoint. The remaining message already is the unnormalized marginal of x_1, since summing out x_2, x_3, x_4 is exactly what m_2 did:

$$p(x_1{=}0) = \frac{m_2(0)}{Z} = \frac{36}{54} = \frac{2}{3} \approx 0.667, \qquad p(x_1{=}1) = \frac{m_2(1)}{Z} = \frac{18}{54} = \frac{1}{3} \approx 0.333,$$

which sum to 1. The skew in P toward $x_1 = 0$ survives the chain and tilts the marginal.

Check by direct matrix product. For a chain, $Z = \mathbf{1}^\top P M M \mathbf{1}$ with $\mathbf{1} = (1,1)^\top$, treating the potentials as transfer matrices. Compute $M^2 = \left[\begin{smallmatrix} 5 & 4 \\ 4 & 5 \end{smallmatrix}\right]$, then $M^2\mathbf{1} = (9,9)^\top$ (this is m_3), then $P\,(M^2\mathbf{1}) = \left[\begin{smallmatrix} 3 & 1 \\ 1 & 1 \end{smallmatrix}\right](9,9)^\top = (36,18)^\top$ (this is m_2), and $\mathbf{1}^\top(36,18)^\top = 54 = Z$. The messages of variable elimination are precisely these intermediate matrix-vector products, which is why elimination costs $\mathcal{O}(N)$ matrix products on a chain instead of the $\mathcal{O}(2^N)$ of brute enumeration. ◊

Variable elimination in one line

To marginalize or normalize, sum out variables one at a time, each time multiplying together only the factors that mention the variable and replacing them with a single new factor (the message) over the remaining variables. The cost is set by the largest intermediate factor, which on a chain or tree stays small, so elimination is exponential only in the graph's *treewidth* rather than in the number of nodes. Chapter 17 turns this into the sum-product algorithm.

Notes and References

The factorization of Bayesian networks, the three canonical structures, d-separation, Markov random fields, and the Hammersley–Clifford theorem follow Sections 8.1–8.3 of Bishop [3]; the comprehensive reference is Koller and Friedman [22], and Barber [1] gives an accessible treatment. The MRF image-denoising model of Section 16.6 and its iterated-conditional-modes optimizer are from Bishop [3], and the MRF formulation of lattice systems traces to Besag [2]. The explaining-away effect of Example 16.3 is the signature of the collider and the reason directed and undirected graphs encode genuinely different independence structures. Computing marginals and most probable configurations in these models, by message passing, is the subject of Chapter 17.

Exercises

Exercise 16.1. A Bayesian network on four binary variables has edges $A \to B$, $A \to C$, $B \to D$, $C \to D$. Write the joint factorization and count the parameters, comparing with the full joint.

Solution. The parents are $\mathrm{pa}(A) = \emptyset$, $\mathrm{pa}(B) = \{A\}$, $\mathrm{pa}(C) = \{A\}$, $\mathrm{pa}(D) = \{B, C\}$, so $p(A, B, C, D) = p(A)\, p(B \mid A)\, p(C \mid A)\, p(D \mid B, C)$. Parameter counts: $p(A)$ needs 1; $p(B \mid A)$ and $p(C \mid A)$ need 2 each; $p(D \mid B, C)$ needs 4. Total $1 + 2 + 2 + 4 = 9$, against the full joint's $2^4 - 1 = 15$. The two missing edges (no $A \to D$, no $B \to C$, etc.) save six parameters. □

Exercise 16.2. For the collider $X \to Z \leftarrow Y$ with X, Y independent fair coins and $Z = X \vee Y$, compute $p(X{=}1 \mid Z{=}1)$ and $p(X{=}1 \mid Z{=}1, Y{=}0)$, and interpret.

Solution. As in the text, $p(Z{=}1) = 1 - \frac{1}{4} = \frac{3}{4}$ and $p(X{=}1, Z{=}1) = p(X{=}1) = \frac{1}{2}$, so $p(X{=}1 \mid Z{=}1) = \frac{1}{2} / \frac{3}{4} = \frac{2}{3}$. Now condition on $Y = 0$: then $Z = X \vee 0 = X$, so $Z = 1$ forces $X = 1$, giving $p(X{=}1 \mid Z{=}1, Y{=}0) = 1$. Learning the other cause is *absent* ($Y = 0$) makes the observed effect $Z = 1$ point entirely at X, raising its probability from $\frac{2}{3}$ to 1, the opposite of the explaining-away in the text where $Y = 1$ lowered it to $\frac{1}{2}$. Conditioning on the collider couples the two causes. □

Exercise 16.3. In the chain $a \to c \to b$, show from the factorization that $a \perp\!\!\!\perp b \mid c$ but a and b are not marginally independent.

Solution. The joint is $p(a, b, c) = p(a)p(c \mid a)p(b \mid c)$. Conditioning on c: $p(a, b \mid c) = p(a, b, c)/p(c) = p(a)p(c \mid a)p(b \mid c)/p(c)$. Using $p(a)p(c \mid a) = p(a, c) = p(a \mid c)p(c)$, this is $p(a \mid c)p(b \mid c)$, a product, so $a \perp\!\!\!\perp b \mid c$. Marginally, $p(a, b) = \sum_c p(a)p(c \mid a)p(b \mid c) = p(a) \sum_c p(c \mid a)p(b \mid c) = p(a)p(b \mid a)$, which equals $p(a)p(b)$ only if $p(b \mid a) = p(b)$, not true in general (information flows from a to b through c). So a and b are marginally dependent but conditionally independent given c. □

Exercise 16.4. In the denoising energy $E(\boldsymbol{x}; \boldsymbol{y}) = -\beta \sum_{(i,j)} x_i x_j - \eta \sum_i x_i y_i$ (taking $h = 0$), with $\beta = 1$, $\eta = 1$, consider a pixel x_i with observation $y_i = -1$ whose four neighbors are all $+1$. Should iterated conditional modes set x_i to $+1$ or -1?

Solution. The terms involving x_i are $-\beta x_i \sum_{j \in N(i)} x_j - \eta x_i y_i = -x_i(4) - x_i(-1) = -x_i(4 - 1) = -3x_i$ (the four neighbors sum to $+4$, and $y_i = -1$). The energy contribution is -3 when $x_i = +1$ and $+3$ when $x_i = -1$, so the lower energy is at $x_i = +1$. ICM sets $x_i = +1$: the four agreeing neighbors outweigh the single noisy observation $y_i = -1$, so the pixel is corrected to match its smooth surroundings, which is exactly the denoising we want. □

Exercise 16.5. Consider a three-node Ising chain $x_1 - x_2 - x_3$ with spins $x_i \in \{-1, +1\}$ and energy $E(\boldsymbol{x}) = -J(x_1 x_2 + x_2 x_3)$, so $p(\boldsymbol{x}) \propto e^{-E(\boldsymbol{x})} = e^{J(x_1 x_2 + x_2 x_3)}$. With coupling $J = 1$, compute the partition function $Z = \sum_{\boldsymbol{x}} e^{x_1 x_2 + x_2 x_3}$. Use $e^2 = 7.389$ and $e^{-2} = 0.135$.

Solution. Enumerate the eight spin configurations and the exponent $s = x_1 x_2 + x_2 x_3$. The two all-aligned states $(+,+,+)$ and $(-,-,-)$ give $s = 1{+}1 = 2$. The two fully alternating states $(+,-,+)$ and $(-,+,-)$ give $s = -1{-}1 = -2$. The remaining four states each have one aligned bond and one anti-aligned bond, so $s = 0$. Hence

$$Z = 2\, e^{2} + 2\, e^{-2} + 4\, e^{0} = 2(7.389) + 2(0.135) + 4 = 14.778 + 0.270 + 4 = 19.05.$$

As a check, use the transfer matrix $T(a, b) = e^{Jab}$, that is $T = \left[\begin{smallmatrix} e & e^{-1} \\ e^{-1} & e \end{smallmatrix}\right]$, so $Z = \mathbf{1}^\top T^2 \mathbf{1}$. Squaring, $T^2 = \left[\begin{smallmatrix} e^2 + e^{-2} & 2 \\ 2 & e^2 + e^{-2} \end{smallmatrix}\right]$, whose four entries sum to $2(e^2 + e^{-2}) + 4 = 19.05$, matching the enumeration. The aligned configurations dominate Z because the positive coupling rewards agreeing neighbors. □

Exercise 16.6. An undirected graph on five nodes $\{1, 2, 3, 4, 5\}$ has edges 1–2, 1–3, 2–3, 2–4, 3–4, 3–5, 4–5. List all maximal cliques and write the MRF factorization $p(\boldsymbol{x}) = \frac{1}{Z} \prod_C \psi_C(\boldsymbol{x}_C)$ over them.

Solution. Check each candidate triangle for all three edges, then test whether it extends. The set $\{1, 2, 3\}$ has edges 1–2, 1–3, 2–3, so it is a clique; it cannot grow, since adding 4 needs the missing 1–4 edge and adding 5 needs the missing 1–5 edge, so $\{1, 2, 3\}$ is maximal. The set $\{2, 3, 4\}$ has edges 2–3, 2–4, 3–4, a clique; adding 1 needs 1–4 and adding 5 needs 2–5, both missing, so it is maximal. The set $\{3, 4, 5\}$ has edges 3–4, 3–5, 4–5, a clique; adding 2 needs 2–5, missing, so it is maximal. No four-node set is fully connected (any such set would need an edge like 1–4 or 2–5 that is absent). The maximal cliques are

$$\{1, 2, 3\}, \quad \{2, 3, 4\}, \quad \{3, 4, 5\},$$

and the field factorizes as $p(\boldsymbol{x}) = \frac{1}{Z}\,\psi_{123}(x_1, x_2, x_3)\,\psi_{234}(x_2, x_3, x_4)\,\psi_{345}(x_3, x_4, x_5)$. The edges 2–3 and 3–4 each lie in two of the maximal cliques, but they are not maximal cliques themselves, so they get no separate potential; their effect is carried inside the triangles that contain them. □

Exercise 16.7. Explain why a Markov random field needs a partition function Z to normalize, whereas a Bayesian network does not.

Solution. A Bayesian network's joint is a product of conditional probabilities $p(x_i \mid \mathrm{pa}(x_i))$, each of which already sums to one over x_i for every setting of its parents; multiplying properly normalized conditionals yields a properly normalized joint, so no extra constant is needed. An MRF's joint is a product of *potentials* $\psi_C(\boldsymbol{x}_C)$, which are merely nonnegative compatibility scores rather than probabilities, and need not sum to anything in particular. Their product is therefore unnormalized, and dividing by $Z = \sum_{\boldsymbol{x}} \prod_C \psi_C(\boldsymbol{x}_C)$ is required to make it a distribution. Computing Z, a sum over all configurations, is the central computational difficulty of MRFs. □

CHAPTER 17

Inference in Graphical Models

The graph that made the model compact also makes the arithmetic compact.
Push the sums inside the products and the exponential cost collapses.

after Professor Peter Chin, ENGS 106

A graphical model defines a joint distribution; *inference* is the act of answering questions about it. The two questions that matter are the *marginal*, the distribution of one variable with all others summed out, $p(x_i) = \sum_{\text{rest}} p(\boldsymbol{x})$, and the *most probable configuration*, the joint assignment of largest probability, $\arg\max_{\boldsymbol{x}} p(\boldsymbol{x})$. Done naively, both are hopeless: summing or maximizing over all configurations of n K-valued variables costs K^n operations. But the same graph structure that let the joint factorize compactly lets the computation factorize too. The trick is the distributive law, pushing sums inside products so that each is done over one variable at a time, and organized as *message passing* along the graph it gives exact inference in time linear in the number of variables for trees. This chapter develops it: the idea on a chain, factor graphs as the general stage, the sum-product algorithm for marginals, and the max-sum algorithm for the most probable configuration.

Roadmap

Suggested reading: Bishop [3], Section 8.4; Koller and Friedman [22] for the general theory.

We cover: the inference problem and its naive cost (Section 17.1); inference on a chain by message passing (Section 17.2); factor graphs (Section 17.3); the sum-product algorithm for marginals (Section 17.4); and the max-sum algorithm for the most probable configuration (Section 17.5).

17.1 The inference problem

Given a joint $p(x_1, \dots, x_n)$ from a graphical model, a marginal query asks for the distribution of one variable, obtained by summing the joint over all the others,

$$p(x_i) = \sum_{x_1} \cdots \sum_{x_{i-1}} \sum_{x_{i+1}} \cdots \sum_{x_n} p(x_1, \dots, x_n).$$

There are $n-1$ nested sums, each over K values, so a direct evaluation touches K^{n-1} terms, exponential in the number of variables and impossible beyond a handful. A most-probable-configuration query replaces every sum with a max, and is equally expensive done directly. The graph rescues both, because the joint is not an arbitrary function of all variables at once but a product of small factors, and a product can be summed far more cheaply than its size suggests.

17.2 Inference on a chain

The idea is clearest on a Markov chain $x_1 \to x_2 \to \cdots \to x_N$, whose joint factors as $p(\boldsymbol{x}) = p(x_1) \prod_{t=2}^{N} p(x_t \mid x_{t-1})$. To find the marginal $p(x_n)$ we must sum out every other variable, but we need not do it all at once. Consider the sum over x_1: it appears only in the first two factors, so

$$\sum_{x_1} p(x_1)\, p(x_2 \mid x_1) = \mu_\alpha(x_2),$$

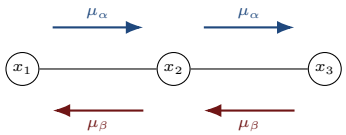


Figure 17.1. Forward messages μ_α (blue) accumulate everything to the left as they sweep right; backward messages μ_β (red) accumulate everything to the right as they sweep left. The marginal at any node is the normalized product of the two messages that meet there, computed in $\mathcal{O}(NK^2)$ rather than $\mathcal{O}(K^N)$.

a function of x_2 alone, a *message* passed from x_1 to x_2. Now x_2 can be summed the same way, $\mu_\alpha(x_3) = \sum_{x_2} \mu_\alpha(x_2)\, p(x_3 \mid x_2)$, and so on: a *forward* sweep accumulates a message μ_α that absorbs everything to the left. Symmetrically, a *backward* sweep accumulates a message μ_β absorbing everything to the right. The marginal is then the (normalized) product of the two messages meeting at the query node, $p(x_n) \propto \mu_\alpha(x_n)\, \mu_\beta(x_n)$. The crucial accounting: each message is a vector of K numbers, and computing the next message from the previous one is a sum of K terms for each of K values, K^2 work, repeated N times. Inference costs $\mathcal{O}(NK^2)$ instead of $\mathcal{O}(K^N)$, exponential cost traded for linear.

Example 17.1 (Forward message passing on a chain). Take a three-node Markov chain $x_1 \to x_2 \to x_3$ of binary variables (states 1 and 2). Let the initial distribution be uniform, $p(x_1) = (0.5, 0.5)$, and let both transitions share the matrix

$$T = \begin{bmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{bmatrix}, \qquad T_{ij} = p(x_{t+1} = j \mid x_t = i),$$

so the chain tends to stay in its current state. The forward message to x_2 sums out x_1:

$$\mu_\alpha(x_2) = \sum_{x_1} p(x_1)\, p(x_2 \mid x_1) = (0.5, 0.5)\, T = (0.5 \cdot 0.8 + 0.5 \cdot 0.3,\ 0.5 \cdot 0.2 + 0.5 \cdot 0.7) = (0.55, 0.45),$$

which (already normalized) is the marginal $p(x_2) = (0.55, 0.45)$. Passing the message one more step sums out x_2:

$$p(x_3) = \mu_\alpha(x_3) = \sum_{x_2} \mu_\alpha(x_2)\, p(x_3 \mid x_2) = (0.55, 0.45)\, T = (0.55 \cdot 0.8 + 0.45 \cdot 0.3,\ 0.55 \cdot 0.2 + 0.45 \cdot 0.7) = (0.575, 0.425).$$

Each step is one multiplication of a length-2 vector by the 2×2 matrix, $K^2 = 4$ multiplications, rather than summing the full joint over the other two variables. (Here the backward message is uniform, because with no observation downstream, $\sum_{x_3} p(x_3 \mid x_2) = 1$ for every x_2; had we observed x_3, the backward message would carry that evidence leftward and change $p(x_2)$.) ◊

Evidence is what makes the backward message earn its keep. Suppose we now *observe* $x_3 = 1$ and ask for $p(x_2 \mid x_3 = 1)$. The backward message carries the observation leftward: $\mu_\beta(x_2) = p(x_3 = 1 \mid x_2)$, which is the first column of T, namely $(0.8, 0.3)$. The forward message is unchanged, $\mu_\alpha(x_2) = (0.55, 0.45)$ from Example 17.1, so

$$p(x_2 \mid x_3 = 1) \propto \mu_\alpha(x_2)\, \mu_\beta(x_2) = (0.55 \cdot 0.8,\ 0.45 \cdot 0.3) = (0.44, 0.135),$$

which normalizes (dividing by 0.575) to $(0.765, 0.235)$. Observing that the chain ended in state 1 raised $p(x_2 = 1)$ from 0.55 to 0.765: the evidence flowed backward through μ_β and updated the belief about the middle node, exactly what the product of the two messages accomplishes.

17.3 Factor graphs

To run message passing on any model, not just a chain, we use a representation that makes the factors explicit: the *factor graph*. It is a bipartite graph with two kinds of node, one *variable node* for each variable and one *factor node* for each factor in the joint, with an edge connecting a factor to each variable it involves. A Bayesian network's factors are its conditionals $p(x_i \mid \mathrm{pa}(x_i))$; an MRF's are its clique potentials ψ_C; either way the factor graph lays bare the product structure that message passing exploits. On a factor graph that is a tree (no loops), messages flow along the edges and the algorithms below are exact (Fig. 17.2).

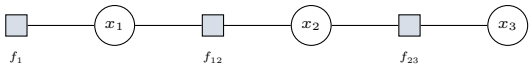


Figure 17.2. The chain $x_1 \to x_2 \to x_3$ drawn as a factor graph: circles are variables, shaded squares are factors ($f_1 = p(x_1)$, $f_{12} = p(x_2 \mid x_1)$, $f_{23} = p(x_3 \mid x_2)$). Each factor connects to the variables it involves, and messages pass in both directions along the edges.

17.4 The sum-product algorithm

The *sum-product algorithm* generalizes the chain's forward and backward sweeps to any tree-structured factor graph. Messages pass in both directions along every edge, of two types. A *variable-to-factor* message is the product of the messages that variable receives from its other neighboring factors,

$$\mu_{x \to f}(x) = \prod_{g \in \mathrm{ne}(x) \setminus f} \mu_{g \to x}(x), \tag{17.1}$$

collecting all the evidence about x except what comes through f. A *factor-to-variable* message multiplies in the factor and sums out the factor's other variables,

$$\mu_{f \to x}(x) = \sum_{\text{other vars of } f} f(\cdots) \prod_{y \in \mathrm{ne}(f) \setminus x} \mu_{y \to f}(y), \tag{17.2}$$

the "sum" (of) "products" that names the algorithm. Once messages have flowed in both directions across every edge, the marginal of any variable is the product of all messages arriving at its node, normalized. On a tree two sweeps suffice, one inward to a chosen root and one back out, so every marginal in the entire model is obtained in $\mathcal{O}(NK^2)$ total, the same linear cost as the chain but now for any tree. The chain's μ_α and μ_β are exactly this algorithm specialized to a line.

The most important special case is the *hidden Markov model*, a chain of hidden states each emitting an observation. Sum-product there is the *forward-backward* algorithm, and it is worth one full numerical pass.

Example 17.2 (Forward-backward in an HMM). A two-state hidden chain has prior $\boldsymbol{\pi} = (0.5, 0.5)$ and transition $T = \left[\begin{smallmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{smallmatrix}\right]$. At three times we observe data whose emission likelihoods $b_t(i) = p(\mathrm{obs}_t \mid x_t = i)$ are $b_1 = (0.9, 0.2)$, $b_2 = (0.6, 0.5)$, $b_3 = (0.1, 0.8)$. The *forward* messages $\alpha_t(i) = p(\mathrm{obs}_{1:t}, x_t = i)$ start from $\alpha_1 = \boldsymbol{\pi} \odot b_1 = (0.45, 0.10)$ and recurse $\alpha_t(j) = b_t(j) \sum_i \alpha_{t-1}(i) T_{ij}$:

$$\alpha_2 = \big(0.6(0.45 \cdot 0.8 + 0.10 \cdot 0.3),\ 0.5(0.45 \cdot 0.2 + 0.10 \cdot 0.7)\big) = (0.234, 0.080),$$

$$\alpha_3 = \big(0.1(0.234 \cdot 0.8 + 0.080 \cdot 0.3),\ 0.8(0.234 \cdot 0.2 + 0.080 \cdot 0.7)\big) = (0.0211, 0.0822).$$

The data likelihood is $p(\mathrm{obs}) = \sum_i \alpha_3(i) = 0.0211 + 0.0822 = 0.1034$, and the filtered belief at the last step is $p(x_3 \mid \mathrm{obs}) \propto \alpha_3 = (0.0211, 0.0822) \to (0.20, 0.80)$.

To *smooth* the middle state we need the backward message $\beta_t(i) = p(\mathrm{obs}_{t+1:T} \mid x_t = i)$, with $\beta_3 = (1, 1)$ and $\beta_t(i) = \sum_j T_{ij} b_{t+1}(j) \beta_{t+1}(j)$:

$$\beta_2 = \big(0.8(0.1) + 0.2(0.8),\ 0.3(0.1) + 0.7(0.8)\big) = (0.24, 0.59).$$

The smoothed posterior multiplies the two messages, $p(x_2 \mid \mathrm{obs}) \propto \alpha_2 \odot \beta_2 = (0.234 \cdot 0.24,\ 0.080 \cdot 0.59) = (0.0562, 0.0472)$, which sums to 0.1034 (the likelihood again, a useful check) and normalizes to $(0.54, 0.46)$. The future observations have pulled the belief about x_2, which is exactly what forward messages alone cannot do. $\lozenge$

17.5 The max-sum algorithm

To find the single most probable configuration $\arg\max_{\boldsymbol{x}} p(\boldsymbol{x})$ rather than marginals, two changes turn sum-product into the *max-sum* algorithm. First, replace every sum in Eq. (17.2) with a max: instead of accumulating total probability, each message carries the best achievable probability of the variable's value given the subtree behind it. Second, work with logarithms, so that products of many small probabilities become sums and do not underflow to zero, which turns "max of products" into "max of sums," the algorithm's name. The forward pass computes, for each value of each variable, the best score attainable, and records which predecessor value achieved it; the most probable configuration is then recovered by *backtracking* from the best final value through the stored choices, exactly the dynamic program known as the Viterbi algorithm for hidden Markov models. As with sum-product, the cost is $\mathcal{O}(NK^2)$ on a tree, an exponential search reduced to a linear sweep by the same push-the-operation-inside-the-product trick. The two algorithms are twins: sum-product answers "how probable is each value on average," max-sum answers "what is the single best joint explanation."

Example 17.3 (Most probable path by max-sum). Take the chain of Example 17.1: $p(x_1) = (0.5, 0.5)$ and transition $T = \left[\begin{smallmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{smallmatrix}\right]$. Find the single most probable sequence (x_1, x_2, x_3). We run max-sum in the equivalent max-*product* form (taking logs would turn the products into sums; the bookkeeping is identical). Pass a max-message forward, recording the best predecessor at each step. From x_1 to x_2,

$$m_1(x_2{=}1) = \max\big(0.5(0.8),\, 0.5(0.3)\big) = 0.40 \text{ (from } x_1{=}1), \quad m_1(x_2{=}2) = \max\big(0.5(0.2),\, 0.5(0.7)\big) = 0.35 \text{ (from } x_1{=}2).$$

From x_2 to x_3, using $m_1 = (0.40, 0.35)$,

$$m_2(x_3{=}1) = \max\big(0.40(0.8),\, 0.35(0.3)\big) = 0.32 \text{ (from } x_2{=}1),$$
$$m_2(x_3{=}2) = \max\big(0.40(0.2),\, 0.35(0.7)\big) = 0.245 \text{ (from } x_2{=}2).$$

The best final value is $x_3 = 1$ with score 0.32. *Backtrack* through the recorded choices: $x_3 = 1$ came from $x_2 = 1$, which came from $x_1 = 1$. The most probable configuration is $(1, 1, 1)$ with probability 0.32, found without scoring all $2^3 = 8$ sequences. This forward-score-then-backtrack is exactly the Viterbi algorithm, the classic decoder for the most probable state sequence of a hidden Markov model. ◊

Remark 17.4 *(Beyond trees).* Exactness rests on the factor graph being a tree. When the graph has loops, as a grid-shaped image MRF does, the message-passing recursion has no well-defined order and the algorithms are no longer exact. Three responses are standard. *Loopy belief propagation* runs the sum-product messages anyway, iterating to a fixed point, often with good empirical results. The *junction tree* algorithm groups variables into clusters to restore a tree at the cost of larger messages. And when even that is too expensive, one abandons exactness for the *sampling* methods of Chapter 20 or variational approximations, which trade a provably correct answer for a tractable one. The coordinate-descent denoiser of Chapter 16 was one such practical compromise on a loopy grid.

17.6 Sum-product on a tree

The chain was a line, so its messages had only one direction to come from. A genuine tree branches, and a branch is where the variable-to-factor rule of Eq. (17.1) starts to matter: a variable sitting at a junction must combine the messages from several factors before forwarding their product onward. We run the algorithm on the smallest tree that shows this, a four-variable model with a junction at x_2.

The factor graph has four variable nodes and three factor nodes (Fig. 17.3). The variable x_2 is the hub; factor f_a joins x_1 to x_2, factor f_b joins x_2 to x_3, and factor f_c joins x_2 to x_4. The unnormalized joint is the product of the three factors,

$$\tilde{p}(x_1, x_2, x_3, x_4) = f_a(x_1, x_2)\, f_b(x_2, x_3)\, f_c(x_2, x_4),$$

and the three leaves x_1, x_3, x_4 each hang off the hub through one factor. All four variables are binary with states 1 and 2.

Choose the factor tables (each row indexed by the first variable, each column by the second):

$$f_a = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix}, \qquad f_b = \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix}, \qquad f_c = \begin{bmatrix} 1 & 1 \\ 2 & 1 \end{bmatrix}.$$

The factors need not be normalized; sum-product normalizes the marginal at the end. We compute $p(x_2)$ by treating x_2 as the root and sending one message inward from each leaf.

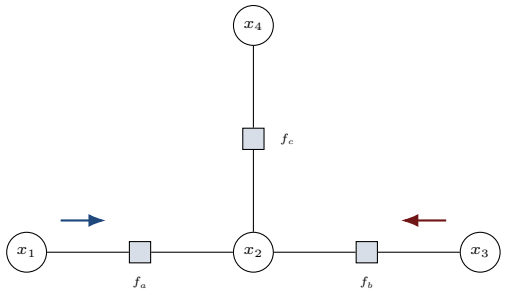


Figure 17.3. A four-variable tree factor graph with a junction at the hub x_2: circles are variables, shaded squares are factors. The three leaves x_1, x_3, x_4 each connect to x_2 through one factor. Finding $p(x_2)$ collects one message from each of the three factors and multiplies them, the branching version of the chain's two sweeps.

Example 17.5 (Sum-product on a four-variable tree). Take the tree of Fig. 17.3 with the factor tables above. We want the marginal $p(x_2)$. Each leaf variable sends the trivial message 1 into its factor (a leaf has no other evidence), so by Eq. (17.2) each factor-to-hub message is just that factor summed over its leaf variable.

Message from f_a. Sum $f_a(x_1, x_2)$ over x_1, leaving a function of x_2,

$$\mu_{f_a \to x_2}(x_2) = \sum_{x_1} f_a(x_1, x_2) = \big(3 + 1,\ 1 + 2\big) = (4,\ 3),$$

where the first entry sums column 1 of f_a and the second sums column 2.

Message from f_b. Sum $f_b(x_2, x_3)$ over x_3 (the leaf here is the second index, so we sum each *row*),

$$\mu_{f_b \to x_2}(x_2) = \sum_{x_3} f_b(x_2, x_3) = \big(2 + 1,\ 1 + 4\big) = (3,\ 5).$$

Message from f_c. Likewise sum $f_c(x_2, x_4)$ over x_4,

$$\mu_{f_c \to x_2}(x_2) = \sum_{x_4} f_c(x_2, x_4) = \big(1 + 1,\ 2 + 1\big) = (2,\ 3).$$

Combine at the hub. The marginal is the product of all three incoming messages, normalized,

$$\tilde{p}(x_2) = \mu_{f_a \to x_2} \odot \mu_{f_b \to x_2} \odot \mu_{f_c \to x_2} = (4 \cdot 3 \cdot 2,\ 3 \cdot 5 \cdot 3) = (24,\ 45),$$

so $p(x_2) = (24, 45)/69 = (8/23,\ 15/23) \approx (0.348,\ 0.652)$, the normalizer $Z = 24 + 45 = 69$ falling out of the same numbers. As a check, the full joint has $2^4 = 16$ configurations; summing $\tilde{p}$ over x_1, x_3, x_4 by brute force gives $(24, 45)$ and $Z = 69$, matching exactly. The message form did the same arithmetic over one variable at a time. To get $p(x_1)$ instead, the hub would forward to f_a the product of the *other* two messages, $\mu_{x_2 \to f_a} = (3, 5) \odot (2, 3) = (6, 15)$, and then $\mu_{f_a \to x_1} = \sum_{x_2} f_a(x_1, x_2)\, \mu_{x_2 \to f_a}(x_2) = (3 \cdot 6 + 1 \cdot 15,\ 1 \cdot 6 + 2 \cdot 15) = (33, 36)$, giving $p(x_1) = (33, 36)/69 = (11/23, 12/23) \approx (0.478, 0.522)$. One inward pass and one outward pass deliver every marginal in the tree. ◊

The branch did the work the chain hid. At the hub, x_2 multiplied three messages together; on a line there were only ever two. That product is exactly Eq. (17.1), and the sums inside each factor message are exactly Eq. (17.2). Nothing else changes, which is the point: the same two rules run on any tree.

17.7 Variable elimination

Message passing computes *all* marginals at once. When only one marginal is wanted, *variable elimination* is the leaner route: sum out the unwanted variables one at a time, pushing each sum as far inside the product as it will go. Each elimination removes a variable and replaces the factors that mentioned it with a single new *intermediate factor* over the variables that remain. The marginal of the surviving variable is the product of whatever factors are left, normalized. Sum-product is variable elimination organized to share work across all queries; elimination is the bare distributive law applied once.

Example 17.6 (Variable elimination on a four-node chain). Take the pairwise Markov network on a chain $x_1 - x_2 - x_3 - x_4$ of binary variables, with unnormalized potentials

$$\phi_{12} = \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix}, \quad \phi_{23} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}, \quad \phi_{34} = \begin{bmatrix} 3 & 1 \\ 1 & 2 \end{bmatrix},$$

and joint $\tilde{p}(\boldsymbol{x}) = \phi_{12}(x_1, x_2)\, \phi_{23}(x_2, x_3)\, \phi_{34}(x_3, x_4)$. We want $p(x_2)$, eliminating in the order x_1, x_4, x_3.

Eliminate x_1. Only ϕ_{12} mentions x_1, so the sum slips inside past ϕ_{23} and ϕ_{34},

$$\tau_1(x_2) = \sum_{x_1} \phi_{12}(x_1, x_2) = (2 + 1,\ 1 + 3) = (3,\ 4),$$

a new factor over x_2 alone. The model now reads $\tau_1(x_2)\, \phi_{23}(x_2, x_3)\, \phi_{34}(x_3, x_4)$.

Eliminate x_4. Only ϕ_{34} mentions x_4, and summing it out (over the second index, so over each row) leaves a factor over x_3,

$$\tau_2(x_3) = \sum_{x_4} \phi_{34}(x_3, x_4) = (3 + 1,\ 1 + 2) = (4,\ 3).$$

The model is now $\tau_1(x_2)\, \phi_{23}(x_2, x_3)\, \tau_2(x_3)$.

Eliminate x_3. Now x_3 appears in ϕ_{23} and in τ_2, so multiply those and sum,

$$\tau_3(x_2) = \sum_{x_3} \phi_{23}(x_2, x_3)\, \tau_2(x_3) = \big(1 \cdot 4 + 2 \cdot 3,\ 2 \cdot 4 + 1 \cdot 3\big) = (10,\ 11),$$

a factor over x_2. The only factors left are $\tau_1(x_2)$ and $\tau_3(x_2)$, both over the query variable.

Read off the marginal. Multiply the surviving factors and normalize,

$$\tilde{p}(x_2) = \tau_1(x_2) \odot \tau_3(x_2) = (3 \cdot 10,\ 4 \cdot 11) = (30,\ 44),$$

so $p(x_2) = (30, 44)/74 = (15/37,\ 22/37) \approx (0.405,\ 0.595)$, with normalizer $Z = 30 + 44 = 74$. Brute-force summation of the joint over the $2^3 = 8$ settings of x_1, x_3, x_4 gives the same $(30, 44)$, confirming the elimination. ◊

Every intermediate factor in that run had a single argument, so each elimination cost only a sum over one variable. That is the best case, and it is no accident: eliminating along the chain never forced two variables into the same new factor. A careless order can. Suppose on the same chain we eliminated x_2 first. It sits in both ϕ_{12} and ϕ_{23}, so the new factor $\tau(x_1, x_3) = \sum_{x_2} \phi_{12}(x_1, x_2)\, \phi_{23}(x_2, x_3)$ has *two* arguments, a $K \times K$ table rather than a length-K vector. The largest factor created during elimination, its number of arguments called the *induced width*, sets the cost: elimination runs in time exponential in that width rather than in N. For a chain the best orders keep the width at one and the cost linear; a poor order inflates it. Finding the order that minimizes the induced width is itself hard in general, which is why structured models (chains, trees) that admit an obvious good order are so much easier to work with.

Remark 17.7 *(Elimination order and the induced width).* On a tree, eliminating leaves before their parents always keeps every intermediate factor small, and the cost stays $\mathcal{O}(NK^2)$. On a loopy graph no order avoids large intermediate factors entirely; the smallest achievable largest-factor size is the treewidth, and exact inference costs $\mathcal{O}(NK^{w+1})$ where w is the induced width of the order used. A grid of side $\sqrt{N}$ has treewidth $\mathcal{O}(\sqrt{N})$, so even the best order costs K to a power that grows with the grid, which is why the loopy methods of Remark 17.4 exist.

17.8 The cost, counted

The whole chapter rests on one comparison: $\mathcal{O}(NK^2)$ for message passing against $\mathcal{O}(K^N)$ for brute force. The exponents make the gap enormous even at modest sizes, and it is worth substituting numbers to feel it.

Example 17.8 (Linear versus exponential, on numbers). Consider a chain of $N = 6$ variables, each taking $K = 5$ values. Brute force forms the marginal of one variable by summing the joint over the other five, a table with

$$K^N = 5^6 = 15{,}625$$

entries to add. Message passing instead computes each forward message from the previous one by a $K \times K$ matrix-vector product, $K^2 = 25$ multiplications, and does so $N = 6$ times,

$$NK^2 = 6 \cdot 5^2 = 6 \cdot 25 = 150$$

operations (a backward pass doubles this to 300, still negligible). The ratio is $15{,}625/150 \approx 104$, so brute force does about a hundred times the work here. The gap widens without bound: at $N = 20$ and $K = 5$, brute force needs $5^{20} \approx 9.5 \times 10^{13}$ operations while message passing needs $20 \cdot 25 = 500$, a ratio near 2×10^{11}. Linear in N beats exponential in N by a margin that grows with every variable added; that single fact is what makes inference on large structured models possible at all. ◊

The same accounting governs variable elimination through Remark 17.7: a good order on a chain or tree keeps the largest intermediate factor at K^2 and the total at $\mathcal{O}(NK^2)$, matching message passing, while a width-w order costs $\mathcal{O}(NK^{w+1})$. The exponent is the induced width, and keeping it small is the entire game.

Notes and References

Inference on chains, factor graphs, and the sum-product and max-sum algorithms follow Section 8.4 of Bishop [3]; the comprehensive treatment, including the junction tree algorithm and the complexity of exact inference, is Koller and Friedman [22]. The max-sum algorithm specialized to a chain is the Viterbi algorithm, the workhorse of hidden Markov models. The distributive-law view, summing a product by pushing sums inside, is the unifying principle, and it is the same idea that makes dynamic programming efficient. For the loopy graphs where exact inference fails, the sampling methods of Chapter 20 provide an approximate alternative, closing the inference toolkit.

Exercises

Exercise 17.1. For the three-node chain of Example 17.1 with $p(x_1) = (0.7, 0.3)$ and the same transition matrix $T = \left[\begin{smallmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{smallmatrix}\right]$, compute the marginals $p(x_2)$ and $p(x_3)$ by forward message passing.

Solution. The forward message to x_2 is $\mu_\alpha(x_2) = (0.7, 0.3)\,T = (0.7{\cdot}0.8{+}0.3{\cdot}0.3,\ 0.7{\cdot}0.2{+}0.3{\cdot}0.7) = (0.56{+}0.09,\ 0.14{+}0.21) = (0.65, 0.35) = p(x_2)$. Passing again, $p(x_3) = (0.65, 0.35)\,T = (0.65 \cdot 0.8 + 0.35 \cdot 0.3,\ 0.65 \cdot 0.2 + 0.35 \cdot 0.7) = (0.52 + 0.105,\ 0.13 + 0.245) = (0.625, 0.375)$. Each marginal is one vector-matrix product, $K^2 = 4$ multiplications, rather than a sum over the other two variables. □

Exercise 17.2. A Markov chain has $N = 100$ variables each taking $K = 10$ values. Compare the number of operations to compute a marginal by brute-force summation versus by message passing.

Solution. Brute force sums the joint over the other $N - 1 = 99$ variables, $K^{99} = 10^{99}$ terms, utterly infeasible. Message passing computes each forward message from the previous one with a $K \times K$ matrix-vector product, $K^2 = 100$ multiplications, repeated about $N = 100$ times, for $NK^2 = 100 \cdot 100 = 10^4$ operations, plus a similar backward pass. The graph structure turns 10^{99} into about 10^4, the exponential-to-linear collapse that makes inference possible. □

Exercise 17.3. Explain the two changes that turn the sum-product algorithm into the max-sum algorithm, and what question each algorithm answers.

Solution. First, every sum over a factor's other variables is replaced by a maximum, so a message carries the best attainable probability for each value rather than the total probability. Second, the computation is done in logarithms, turning products of probabilities into sums (which avoids numerical underflow), so "max of products" becomes "max of sums." One also records, at each max, which choice achieved it, to backtrack the optimizing configuration. Sum-product answers "what is the marginal probability of each value of each variable," averaging over the others; max-sum answers "what single joint assignment of all variables is most probable," the most likely explanation. □

Exercise 17.4. Take the four-variable tree factor graph of Fig. 17.3: hub x_2 joined to leaves x_1, x_3, x_4 through factors $f_a(x_1, x_2)$, $f_b(x_2, x_3)$, $f_c(x_2, x_4)$, all binary. With the tables

$$f_a = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 1 \end{smallmatrix}\right], \quad f_b = \left[\begin{smallmatrix} 1 & 3 \\ 2 & 1 \end{smallmatrix}\right], \quad f_c = \left[\begin{smallmatrix} 2 & 2 \\ 1 & 3 \end{smallmatrix}\right]$$

(rows the first variable, columns the second), compute the marginal $p(x_2)$ by sum-product: pass one message from each factor into the hub and multiply.

Solution. Each leaf sends the message 1 into its factor, so each factor-to-hub message is the factor summed over its leaf variable. From f_a, sum over x_1 (down each column): $\mu_{f_a \to x_2} = (2+1,\ 1+1) = (3, 2)$. From f_b, sum over x_3 (across each row): $\mu_{f_b \to x_2} = (1+3,\ 2+1) = (4, 3)$. From f_c, sum over x_4: $\mu_{f_c \to x_2} = (2+2,\ 1+3) = (4, 4)$. Multiply the three at the hub: $\tilde{p}(x_2) = (3 \cdot 4 \cdot 4,\ 2 \cdot 3 \cdot 4) = (48, 24)$, so $p(x_2) = (48, 24)/72 = (2/3,\ 1/3) \approx (0.667,\ 0.333)$, with normalizer $Z = 48 + 24 = 72$. The branch contributed three messages where a chain would have had two. □

Exercise 17.5. On the pairwise chain $x_1 - x_2 - x_3 - x_4$ of binary variables with potentials

$$\phi_{12} = \left[\begin{smallmatrix} 3 & 1 \\ 1 & 2 \end{smallmatrix}\right], \quad \phi_{23} = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 3 \end{smallmatrix}\right], \quad \phi_{34} = \left[\begin{smallmatrix} 1 & 2 \\ 2 & 1 \end{smallmatrix}\right],$$

compute the marginal $p(x_3)$ by variable elimination in the order x_1, x_2, x_4, showing the intermediate factor at each step. Then state how many arguments the intermediate factor would have if you instead eliminated x_2 first while x_1 and x_3 are both still present.

Solution. Eliminate x_1 (only ϕ_{12} mentions it), summing down each column: $\tau_1(x_2) = \sum_{x_1} \phi_{12} = (3{+}1,\ 1{+}2) = (4, 3)$. Eliminate x_2, which now sits in ϕ_{23} and τ_1: $\tau_2(x_3) = \sum_{x_2} \phi_{23}(x_2, x_3)\, \tau_1(x_2) = (2 \cdot 4 + 1 \cdot 3,\ 1 \cdot 4 + 3 \cdot 3) = (11, 13)$. Eliminate x_4 (only ϕ_{34} mentions it), summing across each row: $\tau_3(x_3) = \sum_{x_4} \phi_{34} = (1{+}2,\ 2{+}1) = (3, 3)$. The factors left over x_3 are τ_2 and τ_3, so $\tilde{p}(x_3) = (11{\cdot}3,\ 13{\cdot}3) = (33, 39)$ and $p(x_3) = (33, 39)/72 = (11/24,\ 13/24) \approx (0.458,\ 0.542)$, with $Z = 72$. Every intermediate factor here had one argument, the cheap case. Eliminating x_2 first instead, while x_1 and x_3 remain, multiplies $\phi_{12}(x_1, x_2)$ and $\phi_{23}(x_2, x_3)$ before summing out x_2, producing a factor $\tau(x_1, x_3)$ with *two* arguments, a 2×2 table. That larger intermediate factor is what a poor elimination order costs. □

Exercise 17.6. Draw (describe) the factor graph for the Markov chain $x_1 \to x_2 \to x_3$, identifying the variable nodes and factor nodes.

Solution. The joint is $p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_2)$, three factors. The factor graph has three variable nodes x_1, x_2, x_3 and three factor nodes: f_1 for $p(x_1)$ connected to x_1 only; f_2 for $p(x_2 \mid x_1)$ connected to x_1 and x_2; and f_3 for $p(x_3 \mid x_2)$ connected to x_2 and x_3. The result is a line $f_1 - x_1 - f_2 - x_2 - f_3 - x_3$, a tree, so sum-product gives exact marginals in two sweeps along it. □

PART VII

Unsupervised Learning

CHAPTER 18

Mixture Models and the EM Algorithm

When you cannot see the labels, guess them softly, fit the model to your guesses, and repeat. That loop is the expectation-maximization algorithm.

after Professor Peter Chin, ENGS 106

The supervised chapters had labels to learn from. Unsupervised learning has none: we see inputs $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ and must find structure unaided. The most common structure is *clustering*, the data falling into groups, and the natural probabilistic model for it is a *mixture*, a weighted sum of simple distributions, one per group. Fitting a mixture is harder than fitting a single distribution, because we do not know which group each point came from, a hidden label. The *expectation-maximization* (EM) algorithm solves exactly this kind of problem with hidden variables, by a loop that alternates between softly guessing the hidden labels (the E step) and fitting the model to those guesses (the M step). This chapter builds up to it: k-means as the hard-assignment special case, the Gaussian mixture model, the EM algorithm and the elegant lower-bound argument that proves it works, and the sense in which k-means is EM with the temperature turned to zero.

Roadmap

Suggested reading: Bishop [3], Chapter 9; the original EM paper is Dempster et al. [10].

We cover: k-means clustering (Section 18.1); the Gaussian mixture model and responsibilities (Section 18.2); the EM algorithm (Section 18.3); why EM works, via Jensen and the ELBO (Section 18.4); k-means as the zero-variance limit of EM (Section 18.5); and choosing the number of clusters (Section 18.6).

18.1 K-means clustering

The simplest clustering method partitions the data into K groups, each represented by a center $\boldsymbol{\mu}_k$, so as to minimize the total squared distance from points to their assigned centers. Writing $r_{nk} = 1$ if point n is assigned to cluster k and 0 otherwise, the *distortion* objective is

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \, \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2. \tag{18.1}$$

We minimize over both the assignments r_{nk} and the centers $\boldsymbol{\mu}_k$, and the trouble is they depend on each other: the best assignment needs the centers, and the best centers need the assignment. *Lloyd's algorithm* breaks the deadlock by alternating, fixing one and optimizing the other:

- **Assignment step:** with the centers fixed, assign each point to its nearest center, $r_{nk} = 1$ for $k = \arg\min_j \|\boldsymbol{x}_n - \boldsymbol{\mu}_j\|^2$. This is the best assignment because each term of J is minimized independently.
- **Update step:** with the assignments fixed, set each center to the mean of its points, $\boldsymbol{\mu}_k = (\sum_n r_{nk}\boldsymbol{x}_n)/(\sum_n r_{nk})$. This is the best center because the mean minimizes the sum of squared distances (setting the derivative of $\sum_n r_{nk}\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$ to zero gives the mean).

Each step decreases J or leaves it unchanged, and since there are only finitely many possible assignments, the algorithm must terminate at a local minimum. A vivid picture, due to Professor Chin, is to imagine dropping K magnets onto the data: each point rolls to its nearest magnet (assignment), then each magnet slides to the balance point of its points (update), and the two repeat until nothing moves.

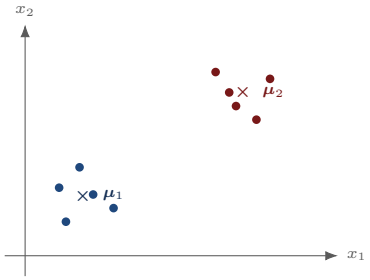


Figure 18.1. Two clusters with their centroids (×). The assignment step sends each point to the nearer centroid; the update step moves each centroid to the mean of its points. Iterating until neither changes minimizes the total squared distance, the hard-assignment limit of the Gaussian mixture below.

Example 18.1 (K-means by hand). Cluster the one-dimensional points $\{1, 2, 3, 10, 11, 12\}$ into $K = 2$ groups, starting from a deliberately poor initialization $\mu_1 = 1$, $\mu_2 = 12$.

Assignment step. Assign each point to the nearer center. Distances put $1, 2, 3$ closer to $\mu_1 = 1$ (e.g., point 3 is 2 from μ_1 but 9 from μ_2) and $10, 11, 12$ closer to $\mu_2 = 12$. So cluster $1 = \{1, 2, 3\}$ and cluster $2 = \{10, 11, 12\}$.

Update step. Move each center to its cluster's mean: $\mu_1 = (1 + 2 + 3)/3 = 2$ and $\mu_2 = (10 + 11 + 12)/3 = 11$.

Second iteration. Reassign with the new centers $\mu_1 = 2$, $\mu_2 = 11$: the same two clusters result, so the assignment no longer changes and the algorithm has converged. The final distortion is $J = [(1-2)^2 + (2-2)^2 + (3-2)^2] + [(10-11)^2 + (11-11)^2 + (12-11)^2] = (1+0+1) + (1+0+1) = 4$. One reassignment was enough to pull the centers from the bad start $(1, 12)$ to the sensible $(2, 11)$. ◊

K-means is fast and simple, but it makes hard, all-or-nothing assignments and implicitly assumes round, equal-sized clusters (it uses plain Euclidean distance). A point halfway between two clusters is forced entirely into one. The probabilistic generalization relaxes both limitations.

18.2 Gaussian mixture models

A *Gaussian mixture model* (GMM) supposes the data was generated by first picking one of K Gaussian components at random, then drawing a point from it. With *mixing coefficients* π_k (the prior probability of each component, $\pi_k \geq 0$, $\sum_k \pi_k = 1$) and component densities $\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, the marginal density of a point is the weighted sum

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k). \tag{18.2}$$

The component a point came from is a hidden *latent variable* $z \in \{1, \dots, K\}$ with $p(z = k) = \pi_k$. We never observe z, but we can compute its posterior given a point, which is the soft cluster assignment we wanted. By Bayes' theorem, the *responsibility* of component k for point $\boldsymbol{x}_n$ is

$$\gamma(z_{nk}) = p(z = k \mid \boldsymbol{x}_n) = \frac{\pi_k \, \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \, \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{18.3}$$

Unlike k-means' hard $r_{nk} \in \{0, 1\}$, the responsibility $\gamma(z_{nk}) \in [0, 1]$ is a soft assignment summing to one over components, so a point can belong 70% to one cluster and 30% to another.

Example 18.2 (Computing a responsibility). Two one-dimensional Gaussian components with equal weights $\pi_1 = \pi_2 = 0.5$, unit variance, and means $\mu_1 = 0$ and $\mu_2 = 4$. What is the responsibility of each for a point at $x = 1$? The component densities are $\mathcal{N}(1 \mid 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-(1-0)^2/2} = \frac{1}{2.5066}(0.6065) = 0.242$ and $\mathcal{N}(1 \mid 4, 1) = \frac{1}{\sqrt{2\pi}} e^{-(1-4)^2/2} = \frac{1}{2.5066} e^{-4.5} = \frac{1}{2.5066}(0.0111) = 0.00443$. The responsibilities are

$$\gamma(z_1) = \frac{0.5(0.242)}{0.5(0.242) + 0.5(0.00443)} = \frac{0.242}{0.242 + 0.00443} = \frac{0.242}{0.2464} = 0.982,$$

and $\gamma(z_2) = 1 - 0.982 = 0.018$. The point at $x = 1$ is assigned with 98% responsibility to the component centered at 0 and only 2% to the one at 4, a soft but nearly decisive verdict that reflects how much closer it sits to the first mean. ◊

Fitting a GMM by maximum likelihood is where the difficulty bites. The log-likelihood is $\sum_n \log\big[\sum_k \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\big]$, with the sum over components trapped *inside* the logarithm. We cannot push the log through the sum, so setting the derivative to zero does not give a closed form, the parameters of every component are tangled together through the responsibilities. EM untangles them.

18.3 The EM algorithm

The expectation-maximization algorithm fits the GMM by alternating, just as k-means did, but with soft responsibilities in place of hard assignments. The observation that makes it work: if we *knew* the responsibilities, fitting each component would be a simple weighted maximum-likelihood problem; and if we knew the components, computing the responsibilities would be the Bayes calculation Eq. (18.3). So we alternate.

EM for a Gaussian mixture

E step. With the parameters fixed, compute the responsibilities $\gamma(z_{nk})$ from Eq. (18.3).

M step. With the responsibilities fixed, update the parameters by responsibility-weighted maximum likelihood. Writing $N_k = \sum_n \gamma(z_{nk})$ for the effective number of points in component k,

$$\pi_k = \frac{N_k}{N}, \quad \boldsymbol{\mu}_k = \frac{1}{N_k} \sum_n \gamma(z_{nk})\, \boldsymbol{x}_n, \quad \boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_n \gamma(z_{nk})\, (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top. \tag{18.4}$$

Repeat until the log-likelihood stops increasing.

The M-step formulas are the ordinary maximum-likelihood estimates of a Gaussian's parameters from Chapter 2, except that each point $\boldsymbol{x}_n$ contributes with weight $\gamma(z_{nk})$ rather than fully: the mean of component k is the responsibility-weighted average of the data, its covariance the responsibility-weighted scatter, and its mixing coefficient the fraction of total responsibility it claims. The two steps are run from an initialization (often the output of k-means) until convergence. Each iteration is cheap, and the next section proves that each one increases the likelihood. Figure 18.2 shows the fitted mixture as a sum of weighted bumps, and the next example runs one full E-and-M iteration on four points.

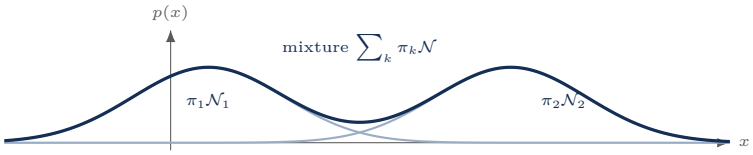


Figure 18.2. A two-component mixture (solid) is the weighted sum of its component Gaussians (dashed). Where the components overlap, points get split responsibilities; far into one component, responsibility is nearly certain.

Example 18.3 (One full EM iteration). Fit a two-component 1-D Gaussian mixture to the four points $x = 0, 1, 4, 5$, starting from $\pi = (0.5, 0.5)$, means $\mu_1 = 1$, $\mu_2 = 4$, and unit variances.

E step. Since the weights and variances are equal, the responsibility ratio reduces to the ratio of $e^{-(x-\mu_k)^2/2}$. For $x = 0$: $e^{-(0-1)^2/2} = e^{-0.5} = 0.607$ and $e^{-(0-4)^2/2} = e^{-8} = 0.0003$, so $\gamma_1 = 0.607/(0.607 + 0.0003) = 0.999$. Repeating for all four points,

$$\gamma(z_{n1}) = (0.999,\ 0.989,\ 0.011,\ 0.001), \qquad \gamma(z_{n2}) = (0.001,\ 0.011,\ 0.989,\ 0.999).$$

M step. The effective counts are $N_1 = \sum_n \gamma_{n1} = 2.00$ and $N_2 = 2.00$, so $\pi_1 = \pi_2 = 2/4 = 0.5$. The means become responsibility-weighted averages:

$$\mu_1 = \frac{0.999(0) + 0.989(1) + 0.011(4) + 0.001(5)}{2.00} = \frac{1.04}{2.00} = 0.52,$$
$$\mu_2 = \frac{0.001(0) + 0.011(1) + 0.989(4) + 0.999(5)}{2.00} = \frac{8.96}{2.00} = 4.48,$$

and the variances follow as the weighted scatter, $\sigma_1^2 = \frac{1}{2}\sum_n \gamma_{n1}(x_n - 0.52)^2 = 0.33$ and likewise $\sigma_2^2 = 0.33$. One iteration pulled the means from $(1, 4)$ to $(0.52, 4.48)$, snapping onto the two obvious clusters $\{0, 1\}$ and $\{4, 5\}$, and tightened the variances. A second iteration would barely move them: the fit has nearly converged. ◊

18.4 Why EM works: Jensen and the lower bound

EM is not an ad hoc heuristic; it provably never decreases the likelihood, and the argument is one of the most elegant in the subject. The obstacle was the log of a sum, $\log p(\boldsymbol{x}) = \log \sum_z p(\boldsymbol{x}, z)$. Introduce any distribution $q(z)$ over the latent variable and rewrite the sum as an expectation, then apply *Jensen's inequality*, which for the concave logarithm states $\log \mathbb{E}[\,\cdot\,] \geq \mathbb{E}[\log\,\cdot\,]$:

$$\log p(\boldsymbol{x}) = \log \sum_z q(z)\, \frac{p(\boldsymbol{x}, z)}{q(z)} \geq \sum_z q(z) \log \frac{p(\boldsymbol{x}, z)}{q(z)} =: \mathcal{L}(q, \boldsymbol{\theta}). \tag{18.5}$$

The quantity $\mathcal{L}(q, \boldsymbol{\theta})$ is the *evidence lower bound* (ELBO): a function of both the auxiliary distribution q and the parameters $\boldsymbol{\theta}$ that never exceeds the log-likelihood. The gap between them is exactly a Kullback-Leibler divergence, as a short calculation shows,

$$\log p(\boldsymbol{x}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q(z) \,\|\, p(z \mid \boldsymbol{x}))\,, \tag{18.6}$$

and since $\mathrm{KL}(\cdot \,\|\, \cdot) \geq 0$ (Gibbs' inequality, Chapter 1), the bound is tight, $\mathcal{L} = \log p(\boldsymbol{x})$, exactly when $q(z) = p(z \mid \boldsymbol{x})$, the posterior, which is the responsibility.

Now EM is transparent. The **E step** maximizes the bound over q with $\boldsymbol{\theta}$ fixed: the best q is the posterior $p(z \mid \boldsymbol{x})$, which drives the KL gap to zero and makes the bound touch the log-likelihood, $\mathcal{L} = \log p$. That is exactly computing the responsibilities. The **M step** maximizes the bound over $\boldsymbol{\theta}$ with q fixed: holding the responsibilities, it increases $\mathcal{L}$, which (since $\mathcal{L} \leq \log p$) pushes the log-likelihood up too. Because the E step makes $\mathcal{L} = \log p$ and the M step increases $\mathcal{L}$, each full iteration increases the log-likelihood, or leaves it fixed at a stationary point.

Theorem 18.4 (EM monotonically increases the likelihood). Each EM iteration satisfies $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{new}}) \geq \log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{old}})$.

Proof. After the E step at $\boldsymbol{\theta}^{\mathrm{old}}$, we set $q = p(z \mid \boldsymbol{x}, \boldsymbol{\theta}^{\mathrm{old}})$, making the bound tight: $\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}}) = \log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{old}})$. The M step chooses $\boldsymbol{\theta}^{\mathrm{new}}$ to maximize $\mathcal{L}(q, \cdot)$, so $\mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}})$. Finally the bound still holds at the new parameters, $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{new}})$. Chaining the three, $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\mathrm{old}}) = \log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\mathrm{old}})$. ■

The likelihood climbs every iteration and is bounded above, so it converges; like k-means, EM reaches a local optimum, and random restarts are used to find a good one. This lower-bound view is more than a proof: it is the template for *variational inference*, where the posterior $p(z \mid \boldsymbol{x})$ is intractable and one maximizes the ELBO over a restricted family of q instead, the workhorse of modern approximate Bayesian methods.

18.5 K-means as the zero-variance limit of EM

K-means and EM are not merely analogous; k-means is a limiting case of EM. Take a GMM in which every component shares a spherical covariance $\boldsymbol{\Sigma}_k = \epsilon I$ and let $\epsilon \to 0$. The responsibility Eq. (18.3) involves $\exp(-\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2/2\epsilon)$, and as $\epsilon \to 0$ this exponential becomes infinitely peaked: for each point, the component with the nearest mean gets responsibility approaching 1 and all others approaching 0. The soft responsibilities harden into the all-or-nothing assignments of k-means, the E step becomes "assign to the nearest center," and the M step's responsibility-weighted mean becomes the plain cluster mean. So k-means is EM for a GMM with vanishing, equal, spherical variances: the hard-assignment, zero-temperature limit of the soft probabilistic algorithm. This explains both k-means' speed (no

soft bookkeeping) and its limitations (it cannot represent clusters of different sizes, shapes, or orientations, which a full GMM with general $\boldsymbol{\Sigma}_k$ can).

18.6 Choosing the number of clusters

Both k-means and the GMM require the number of clusters K in advance, and choosing it is a model-selection problem of the kind Chapter 4 warned about: more clusters always fit the training data better (in the limit $K = N$, every point is its own cluster with zero distortion), so we cannot pick K by fit alone. Several criteria balance fit against complexity. The *Akaike* and *Bayesian information criteria*, $\mathrm{AIC} = 2p - 2\hat{\ell}$ and $\mathrm{BIC} = p \log N - 2\hat{\ell}$, add a penalty for the number of parameters p to the maximized log-likelihood $\hat{\ell}$, with BIC penalizing complexity more heavily as the data grows; the preferred K minimizes the criterion. The *elbow method* plots distortion against K and looks for the kink where adding clusters stops helping much. The *silhouette score*, $s(i) = (b(i) - a(i))/\max(a(i), b(i))$ averaged over points (where $a(i)$ is the mean distance to a point's own cluster and $b(i)$ to the nearest other cluster), measures how well-separated the clusters are, ranging from -1 to 1 with higher better. All are practical stand-ins for the cross-validation of Chapter 4, adapted to the unlabeled setting where there is no held-out error to measure directly.

18.7 A two-iteration k-means run in full

The one-iteration example earlier converged immediately because its initialization was already close to the answer. The more instructive case is one where a point changes hands between iterations, which is where the alternation does real work. We run Lloyd's algorithm to convergence on six two-dimensional points and watch a point migrate from one cluster to the other on the second pass.

Example 18.5 (K-means over two iterations). Cluster the six points

$$\boldsymbol{x}_1 = (1,1),\ \boldsymbol{x}_2 = (1,2),\ \boldsymbol{x}_3 = (3,2),\ \boldsymbol{x}_4 = (5,4),\ \boldsymbol{x}_5 = (5,5),\ \boldsymbol{x}_6 = (6,5)$$

into $K = 2$ groups, starting from $\boldsymbol{\mu}_1 = (1,1)$ and $\boldsymbol{\mu}_2 = (3,2)$. We use squared Euclidean distance throughout, so a point goes to the center with the smaller $\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$.

Iteration 1, assignment. Compute both squared distances for each point and keep the smaller (the assigned cluster is in bold):

$\boldsymbol{x}_n$	$\|\boldsymbol{x}_n - \boldsymbol{\mu}_1\|^2$	$\|\boldsymbol{x}_n - \boldsymbol{\mu}_2\|^2$	cluster
$(1,1)$	0	$4+1=5$	**1**
$(1,2)$	$0+1=1$	$4+0=4$	**1**
$(3,2)$	$4+1=5$	0	**2**
$(5,4)$	$16+9=25$	$4+4=8$	**2**
$(5,5)$	$16+16=32$	$4+9=13$	**2**
$(6,5)$	$25+16=41$	$9+9=18$	**2**

So cluster $1 = \{\boldsymbol{x}_1, \boldsymbol{x}_2\}$ and cluster $2 = \{\boldsymbol{x}_3, \boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$. The seed $\boldsymbol{\mu}_2 = (3,2)$ pulled the nearby point $(3,2)$ to itself, an artifact of the lopsided start.

Iteration 1, update. Each center moves to the mean of its assigned points:

$$\boldsymbol{\mu}_1 = \tfrac{1}{2}\big[(1,1) + (1,2)\big] = (1,\ 1.5), \qquad \boldsymbol{\mu}_2 = \tfrac{1}{4}\big[(3,2) + (5,4) + (5,5) + (6,5)\big] = \left(\tfrac{19}{4},\ 4\right) = (4.75,\ 4).$$

Iteration 2, assignment. Reassign with the moved centers. The only point in doubt is $(3,2)$, which now sits much closer to $\boldsymbol{\mu}_1 = (1, 1.5)$:

$$\|(3,2) - (1,1.5)\|^2 = 4 + 0.25 = 4.25, \qquad \|(3,2) - (4.75,4)\|^2 = 3.0625 + 4 = 7.0625,$$

so $(3,2)$ switches to cluster 1. Checking the rest confirms no other point moves: $(1,1)$ and $(1,2)$ stay with cluster 1 (distances 0.25 and 0.25 versus 23.06 and 18.06), and $(5,4), (5,5), (6,5)$ stay with cluster 2 (distances $0.0625, 1.0625, 2.5625$ versus $22.25, 28.25, 37.25$). The new partition is cluster $1 = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3\}$, cluster $2 = \{\boldsymbol{x}_4, \boldsymbol{x}_5, \boldsymbol{x}_6\}$.

Iteration 2, update. Recompute the means of the corrected clusters:

$$\boldsymbol{\mu}_1 = \tfrac{1}{3}\big[(1,1) + (1,2) + (3,2)\big] = \left(\tfrac{5}{3},\ \tfrac{5}{3}\right) \approx (1.667,\ 1.667),$$

$$\boldsymbol{\mu}_2 = \tfrac{1}{3}\big[(5,4)+(5,5)+(6,5)\big] = \Big(\tfrac{16}{3},\ \tfrac{14}{3}\Big) \approx (5.333,\ 4.667).$$

Convergence check. A third assignment with these centers reproduces the same two clusters (each point is plainly nearest its own center now), so the assignments stop changing and the algorithm has converged. The final distortion sums the squared distances of points to their centers:

$$J_1 = \tfrac{8}{9}+\tfrac{5}{9}+\tfrac{17}{9} = \tfrac{30}{9} = \tfrac{10}{3}, \qquad J_2 = \tfrac{5}{9}+\tfrac{2}{9}+\tfrac{5}{9} = \tfrac{12}{9} = \tfrac{4}{3},$$

where, for instance, $\|(1,1)-(\tfrac{5}{3},\tfrac{5}{3})\|^2 = (\tfrac{2}{3})^2+(\tfrac{2}{3})^2 = \tfrac{8}{9}$. The total is $J = \tfrac{10}{3}+\tfrac{4}{3} = \tfrac{14}{3} \approx 4.667$. The second iteration was the one that mattered: it corrected the point the bad initialization had misassigned, and the distortion fell from 11.25 (after iteration 1) to 4.667. ◊

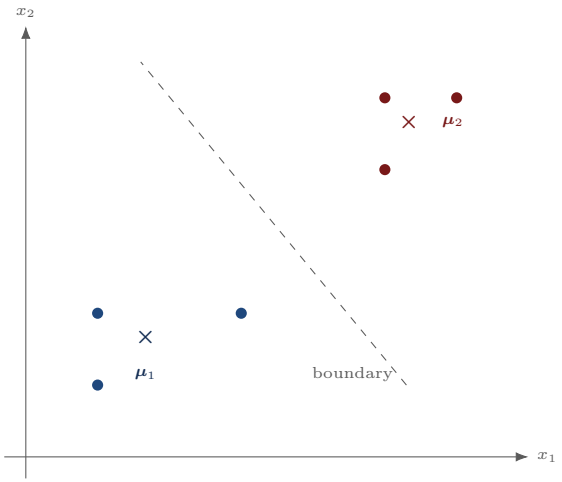


Figure 18.3. The six points of Example 18.5 at convergence, colored by final cluster, with the two centroids (×) at $(\tfrac{5}{3},\tfrac{5}{3})$ and $(\tfrac{16}{3},\tfrac{14}{3})$. The dashed line is the cluster boundary, the perpendicular bisector of the segment joining the centroids: every point on it is equidistant from the two centers, and each point is assigned to the center on its own side. Point $(3,2)$ started in the right-hand cluster and crossed to the left one on the second iteration.

18.8 The lower bound on numbers: log-likelihood, ELBO, and the KL gap

The decomposition $\log p(\boldsymbol{x}) = \mathcal{L}(q,\boldsymbol{\theta}) + \mathrm{KL}(q(z)\,\|\,p(z\mid\boldsymbol{x}))$ from Eq. (18.6) is the engine of the convergence proof, but it is easy to read it as abstract. A tiny example with explicit numbers makes the three quantities concrete: the log-likelihood, the evidence lower bound for a chosen q, and the gap that separates them, which is exactly a Kullback-Leibler divergence.

Example 18.6 (The ELBO and the Jensen gap on numbers). Take a latent-variable model with one observed value $\boldsymbol{x}$ and a binary latent $z \in \{1,2\}$. Suppose the joint probabilities are

$$p(\boldsymbol{x}, z=1) = 0.2, \qquad p(\boldsymbol{x}, z=2) = 0.1.$$

The marginal likelihood of the observation is the sum over the latent,

$$p(\boldsymbol{x}) = \sum_z p(\boldsymbol{x}, z) = 0.2 + 0.1 = 0.3, \qquad \log p(\boldsymbol{x}) = \log 0.3 = -1.204.$$

The true posterior over the latent is $p(z\mid\boldsymbol{x}) = p(\boldsymbol{x},z)/p(\boldsymbol{x})$, namely

$$p(z=1\mid\boldsymbol{x}) = \tfrac{0.2}{0.3} = \tfrac{2}{3}, \qquad p(z=2\mid\boldsymbol{x}) = \tfrac{0.1}{0.3} = \tfrac{1}{3}.$$

The ELBO for a chosen q. Pick the deliberately wrong (uniform) auxiliary distribution $q(z=1) = q(z=2) = 0.5$. The evidence lower bound from Eq. (18.5) is the expectation under q of $\log\big(p(\boldsymbol{x},z)/q(z)\big)$:

$$\mathcal{L}(q,\boldsymbol{\theta}) = \sum_z q(z)\log\frac{p(\boldsymbol{x},z)}{q(z)} = 0.5\log\frac{0.2}{0.5} + 0.5\log\frac{0.1}{0.5}.$$

The two terms are $0.5\log 0.4 = 0.5(-0.9163) = -0.4581$ and $0.5\log 0.2 = 0.5(-1.6094) = -0.8047$, so

$$\mathcal{L}(q, \boldsymbol{\theta}) = -0.4581 - 0.8047 = -1.2629.$$

This sits below the log-likelihood, $-1.2629 \le -1.204$, exactly as the bound promises: $\mathcal{L} \le \log p(\boldsymbol{x})$.

The gap is the KL divergence. The difference $\log p(\boldsymbol{x}) - \mathcal{L}(q, \boldsymbol{\theta}) = -1.204 - (-1.2629) = 0.0589$ should equal $\mathrm{KL}(q(z) \,\|\, p(z \mid \boldsymbol{x}))$. Compute the KL directly:

$$\mathrm{KL}(q \,\|\, p(z \mid \boldsymbol{x})) = \sum_z q(z) \log \frac{q(z)}{p(z \mid \boldsymbol{x})} = 0.5 \log \frac{0.5}{2/3} + 0.5 \log \frac{0.5}{1/3}.$$

The terms are $0.5\log 0.75 = 0.5(-0.2877) = -0.1438$ and $0.5\log 1.5 = 0.5(0.4055) = 0.2027$, summing to 0.0589. The gap and the KL match to the digit, confirming $\log p(\boldsymbol{x}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q \,\|\, p(z \mid \boldsymbol{x}))$.

Tightening the bound. Now set q to the posterior itself, $q(z) = p(z \mid \boldsymbol{x}) = (\frac{2}{3}, \frac{1}{3})$. The KL term becomes $\mathrm{KL}(p \,\|\, p) = 0$, so the bound is tight, and indeed

$$\mathcal{L} = \tfrac{2}{3} \log \frac{0.2}{2/3} + \tfrac{1}{3} \log \frac{0.1}{1/3} = \tfrac{2}{3} \log 0.3 + \tfrac{1}{3} \log 0.3 = \log 0.3 = -1.204 = \log p(\boldsymbol{x}).$$

This is precisely what the E step does: choosing $q = p(z \mid \boldsymbol{x})$ drives the KL gap to zero and lifts the bound up to touch the log-likelihood, after which the M step can push the likelihood higher by optimizing $\boldsymbol{\theta}$ with this q held fixed. ◊

Remark 18.7 *(Why the bound is enough).* The numbers expose the logic of Theorem 18.4. A poor q leaves a positive KL gap, so $\mathcal{L}$ underestimates $\log p(\boldsymbol{x})$; the worse q matches the posterior, the larger the gap. Because the gap is a KL divergence it can never be negative, which is why $\mathcal{L}$ is a genuine lower bound and never an overestimate. The E step removes the gap entirely, so after it the surrogate $\mathcal{L}$ and the true objective $\log p(\boldsymbol{x})$ agree, and any increase the M step makes to $\mathcal{L}$ is an increase to $\log p(\boldsymbol{x})$ as well.

18.9 Model selection by AIC and BIC: a worked choice

Section 18.6 introduced the information criteria $\mathrm{AIC} = 2p - 2\hat{\ell}$ and $\mathrm{BIC} = p \log m - 2\hat{\ell}$, where p is the number of free parameters, $\hat{\ell}$ the maximized log-likelihood, and m the number of data points. Both reward fit (through $-2\hat{\ell}$) and punish complexity (through the parameter term), and the chosen K is the one that minimizes the criterion. The two can disagree, and seeing them disagree on numbers is the clearest way to understand the difference between them.

Example 18.8 (Choosing K with AIC and BIC). We fit a one-dimensional Gaussian mixture with $K = 1, 2, 3$ components to $m = 100$ points and record the maximized log-likelihoods. Each component contributes a mean, a variance, and a mixing weight, but the weights must sum to one, so a K-component model has $p = 3K - 1$ free parameters: $p = 2, 5, 8$ for $K = 1, 2, 3$. Suppose the fits give

$$\hat{\ell}_1 = -330, \qquad \hat{\ell}_2 = -300, \qquad \hat{\ell}_3 = -295,$$

as expected, more components fit better, so the log-likelihood rises with K. With $\log m = \log 100 = 4.605$, the criteria are

$$\mathrm{AIC} = 2p - 2\hat{\ell}, \qquad \mathrm{BIC} = p(4.605) - 2\hat{\ell}.$$

Tabulating both for each K:

K	p	$-2\hat{\ell}$	$\mathrm{AIC} = 2p - 2\hat{\ell}$	$\mathrm{BIC} = 4.605\,p - 2\hat{\ell}$
1	2	660	$4 + 660 = 664.00$	$9.21 + 660 = 669.21$
2	5	600	$10 + 600 = 610.00$	$23.03 + 600 = 623.03$
3	8	590	$16 + 590 = 606.00$	$36.84 + 590 = 626.84$

The two criteria split. **AIC** keeps falling, $664.00 \to 610.00 \to 606.00$, so it prefers $K = 3$. **BIC** falls then rises, $669.21 \to 623.03 \to 626.84$, so it prefers $K = 2$.

The disagreement comes from the per-parameter penalty. Moving from $K = 2$ to $K = 3$ adds three parameters and improves $-2\hat{\ell}$ by $600 - 590 = 10$. AIC charges 2 per parameter, a penalty of $2 \times 3 = 6$, which is less than the gain of 10, so AIC still drops by 4 and rewards the extra component. BIC charges $\log m = 4.605$ per parameter, a penalty of

$4.605 \times 3 = 13.82$, which exceeds the gain of 10, so BIC rises by 3.82 and rejects the third component. With $m = 100$ points the BIC penalty per parameter (4.605) is more than double the AIC penalty (2), so BIC is the more conservative choice, and the gap only widens as m grows. Here BIC's verdict of $K = 2$ matches the two visibly separated clusters; AIC's $K = 3$ is the familiar tendency of AIC to err toward slightly richer models. ◊

Notes and References

The k-means algorithm, the Gaussian mixture model, the EM algorithm, and the lower-bound derivation follow Chapter 9 of Bishop [3]. The EM algorithm in its general form is due to Dempster et al. [10]. The ELBO decomposition Eq. (18.6) and the view of the E step as tightening the bound are the foundation of the variational inference that scales these ideas to intractable posteriors, treated further in Bishop [3], Chapter 10. The k-means-as-EM-limit connection and the model-selection criteria of Section 18.6 are standard. EM appears throughout machine learning wherever there are hidden variables, from mixture models to hidden Markov models to the latent-variable models of Chapter 19, which is where we turn next.

Exercises

Exercise 18.1. Run k-means with $K = 2$ on the points $\{0, 1, 8, 9\}$ starting from $\mu_1 = 0$, $\mu_2 = 9$. Give the assignments, the updated centers, and the final distortion.

Solution. *Assignment:* 0 and 1 are nearer $\mu_1 = 0$ (distances $0, 1$ versus $9, 8$), and $8, 9$ are nearer $\mu_2 = 9$, so cluster $1 = \{0, 1\}$, cluster $2 = \{8, 9\}$. *Update:* $\mu_1 = (0 + 1)/2 = 0.5$, $\mu_2 = (8 + 9)/2 = 8.5$. *Recheck:* with centers 0.5 and 8.5 the same assignment holds, so it has converged. The distortion is $J = [(0-0.5)^2+(1-0.5)^2]+[(8-8.5)^2+(9-8.5)^2] = (0.25 + 0.25) + (0.25 + 0.25) = 1$. □

Exercise 18.2. A two-component GMM has $\pi_1 = \pi_2 = 0.5$, unit variances, and means $\mu_1 = 0$, $\mu_2 = 2$. Compute the responsibilities of each component for a point at $x = 2$.

Solution. The densities at $x = 2$ are $\mathcal{N}(2 \mid 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-(2)^2/2} = \frac{1}{2.5066} e^{-2} = \frac{0.1353}{2.5066} = 0.0540$ and $\mathcal{N}(2 \mid 2, 1) = \frac{1}{\sqrt{2\pi}} e^{0} = \frac{1}{2.5066} = 0.3989$. The responsibilities are $\gamma(z_1) = \frac{0.5(0.0540)}{0.5(0.0540)+0.5(0.3989)} = \frac{0.0540}{0.4529} = 0.119$ and $\gamma(z_2) = 0.881$. The point sits on the second mean, so component 2 claims 88% responsibility and component 1 only 12%. □

Exercise 18.3. Using the ELBO decomposition $\log p(\boldsymbol{x}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q(z) \,\|\, p(z \mid \boldsymbol{x}))$, explain why the E step (setting $q = p(z \mid \boldsymbol{x})$) and the M step together cannot decrease the log-likelihood.

Solution. The E step sets $q = p(z \mid \boldsymbol{x}, \boldsymbol{\theta}^{\text{old}})$, making the KL term zero, so the bound is tight: $\mathcal{L}(q, \boldsymbol{\theta}^{\text{old}}) = \log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\text{old}})$. The M step maximizes $\mathcal{L}(q, \cdot)$ over $\boldsymbol{\theta}$, so $\mathcal{L}(q, \boldsymbol{\theta}^{\text{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\text{old}})$. Since $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \geq \mathcal{L}(q, \boldsymbol{\theta})$ always (KL is nonnegative), we have $\log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\text{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\text{new}}) \geq \mathcal{L}(q, \boldsymbol{\theta}^{\text{old}}) = \log p(\boldsymbol{x} \mid \boldsymbol{\theta}^{\text{old}})$, so the log-likelihood does not decrease. □

Exercise 18.4. Explain why k-means is the limit of EM for a Gaussian mixture as the (shared, spherical) component variance $\epsilon \to 0$.

Solution. With $\boldsymbol{\Sigma}_k = \epsilon I$, the responsibility of component k for point $\boldsymbol{x}_n$ is proportional to $\pi_k \exp(-\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2/2\epsilon)$. As $\epsilon \to 0$ the exponent's differences are magnified without bound, so the component with the smallest distance $\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$ dominates: its responsibility tends to 1 and all others to 0. The soft E step becomes the hard "assign each point to its nearest center," and the responsibility-weighted M-step mean becomes the ordinary cluster mean. These are exactly the two steps of k-means, so k-means is the zero-variance (hard-assignment) limit of EM. □

Exercise 18.5. Run k-means with $K = 2$ on the one-dimensional points $\{0, 1, 2, 7, 8, 9\}$ starting from $\mu_1 = 0$, $\mu_2 = 2$. Carry out two full assignment-and-update iterations, identify the point that changes clusters between them, give the final centers, and compute the final distortion.

Solution. *Iteration 1, assignment.* Each point goes to the nearer center. Distances to $(\mu_1, \mu_2) = (0, 2)$ are 0 and 2 for the point 0, 1 and 1 for the point 1 (a tie, broken to μ_1), 2 and 0 for 2, and so on; the result is cluster $1 = \{0, 1\}$ and cluster $2 = \{2, 7, 8, 9\}$. *Update:* $\mu_1 = (0 + 1)/2 = 0.5$ and $\mu_2 = (2 + 7 + 8 + 9)/4 = 26/4 = 6.5$.

Iteration 2, assignment. With centers $(0.5, 6.5)$ the point 2 is now nearer μ_1, since $|2 - 0.5| = 1.5 < |2 - 6.5| = 4.5$, so it switches to cluster 1. The points $0, 1$ stay with cluster 1 and $7, 8, 9$ stay with cluster 2, giving cluster $1 = \{0, 1, 2\}$ and cluster $2 = \{7, 8, 9\}$. *Update:* $\mu_1 = (0 + 1 + 2)/3 = 1$ and $\mu_2 = (7 + 8 + 9)/3 = 8$. The migrating point is 2, which moved from cluster 2 to cluster 1 on the second pass.

Convergence and distortion. A third assignment with centers $(1, 8)$ reproduces the same clusters, so the algorithm has converged. The distortion is

$$J = \left[(0-1)^2 + (1-1)^2 + (2-1)^2\right] + \left[(7-8)^2 + (8-8)^2 + (9-8)^2\right] = (1 + 0 + 1) + (1 + 0 + 1) = 4.$$

□

Exercise 18.6. A latent-variable model with one observed $\boldsymbol{x}$ and binary latent $z \in \{1, 2\}$ has joint probabilities $p(\boldsymbol{x}, z = 1) = 0.3$ and $p(\boldsymbol{x}, z = 2) = 0.1$. (a) Find $\log p(\boldsymbol{x})$ and the posterior $p(z \mid \boldsymbol{x})$. (b) For the uniform $q(z) = (0.5, 0.5)$, compute the ELBO $\mathcal{L}(q, \boldsymbol{\theta})$ and verify $\mathcal{L} \le \log p(\boldsymbol{x})$. (c) Check that the gap $\log p(\boldsymbol{x}) - \mathcal{L}$ equals $\mathrm{KL}(q \,\|\, p(z \mid \boldsymbol{x}))$.

Solution. (a) $p(\boldsymbol{x}) = 0.3 + 0.1 = 0.4$, so $\log p(\boldsymbol{x}) = \log 0.4 = -0.9163$. The posterior is $p(z = 1 \mid \boldsymbol{x}) = 0.3/0.4 = 0.75$ and $p(z = 2 \mid \boldsymbol{x}) = 0.1/0.4 = 0.25$.

(b) The ELBO is $\mathcal{L} = \sum_z q(z) \log \frac{p(\boldsymbol{x}, z)}{q(z)} = 0.5 \log \frac{0.3}{0.5} + 0.5 \log \frac{0.1}{0.5}$. The terms are $0.5 \log 0.6 = 0.5(-0.5108) = -0.2554$ and $0.5 \log 0.2 = 0.5(-1.6094) = -0.8047$, so $\mathcal{L} = -1.0601$. Since $-1.0601 \le -0.9163$, the bound holds, $\mathcal{L} \le \log p(\boldsymbol{x})$.

(c) The gap is $\log p(\boldsymbol{x}) - \mathcal{L} = -0.9163 - (-1.0601) = 0.1438$. Computing the KL directly, $\mathrm{KL}(q \,\|\, p(z \mid \boldsymbol{x})) = 0.5 \log \frac{0.5}{0.75} + 0.5 \log \frac{0.5}{0.25} = 0.5(-0.4055) + 0.5(0.6931) = -0.2027 + 0.3466 = 0.1438$. The gap equals the KL, confirming $\log p(\boldsymbol{x}) = \mathcal{L}(q, \boldsymbol{\theta}) + \mathrm{KL}(q \,\|\, p(z \mid \boldsymbol{x}))$. Setting q to the posterior $(0.75, 0.25)$ would make the KL zero and the bound tight. □

Exercise 18.7. Explain why maximizing the GMM log-likelihood $\sum_n \log \sum_k \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ directly is hard, and how EM circumvents the difficulty.

Solution. The sum over components sits inside the logarithm, so the log cannot be distributed over it and the derivative with respect to one component's parameters depends on all the others through the shared denominator (the mixture density), giving coupled equations with no closed-form solution. EM circumvents this by introducing the latent assignment: given the responsibilities (the E step), the expected complete-data log-likelihood has the log *inside* the sum over components removed (each point is softly assigned), so the M step decouples into independent weighted Gaussian fits with closed forms. Alternating the two solves the original problem without ever differentiating the log-of-a-sum. □

CHAPTER 19

Continuous Latent Variables: PCA and ICA

High-dimensional data usually lives on a low-dimensional shadow. Principal component analysis finds the shadow.

after Professor Peter Chin, ENGS 106

The mixture models of Chapter 18 had a *discrete* latent variable, the cluster label. This chapter treats *continuous* latent variables, the case where the data is governed by a few hidden real-valued factors. The premise, the *manifold hypothesis*, is that high-dimensional data rarely fills its space; a thousand-pixel image of a handwritten digit varies along only a handful of meaningful directions (slant, thickness, size), so it lies near a low-dimensional surface in the thousand-dimensional pixel space. Finding that surface is *dimensionality reduction*, and the workhorse is *principal component analysis* (PCA), which finds the low-dimensional subspace capturing the most variance. PCA turns out to be nothing but the eigendecomposition of the covariance matrix from Chapter 2, seen in a new light, and it connects to the SVD, to a probabilistic latent-variable model, and even to a linear autoencoder. We close with independent component analysis, which seeks not the highest-variance directions but the most independent ones.

Roadmap

Suggested reading: Bishop [3], Chapter 12; Deisenroth et al. [9], Chapter 10.

We cover: PCA from two views, maximum variance and minimum reconstruction error (Section 19.1); the eigendecomposition derivation (Section 19.2); choosing the number of components (Section 19.3); PCA via the SVD and probabilistic PCA (Section 19.4); the linear autoencoder as PCA (Section 19.5); and independent component analysis (Section 19.6).

19.1 Two views of PCA

Principal component analysis can be defined two ways that turn out to coincide. The *maximum-variance* view seeks the direction along which the projected data varies most: project every point onto a unit vector $\boldsymbol{u}$, and choose $\boldsymbol{u}$ to maximize the variance of the projections, on the grounds that a direction of large variance carries the most information and a direction of tiny variance is nearly constant and can be discarded. The *minimum-reconstruction-error* view seeks the subspace onto which the data can be projected with the least loss: choose the k-dimensional subspace minimizing the total squared distance from the points to their projections, so that compressing to the subspace and decompressing back loses as little as possible. Remarkably, both views give the same answer, the subspace spanned by the top eigenvectors of the data's covariance matrix, which is why PCA has such a central place.

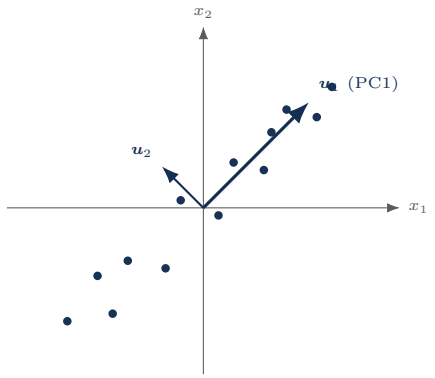


Figure 19.1. The first principal component $\boldsymbol{u}_1$ is the direction of greatest variance (the long axis of the cloud); the second $\boldsymbol{u}_2$ is orthogonal to it, along the short axis. Projecting onto $\boldsymbol{u}_1$ alone keeps most of the variance, the eigenvector of the covariance with the largest eigenvalue.

19.2 The eigendecomposition derivation

Take the maximum-variance view and assume the data is centered (mean subtracted), with covariance matrix $\boldsymbol{\Sigma} = \frac{1}{N}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$. The variance of the data projected onto a unit vector $\boldsymbol{u}$ is

$$\frac{1}{N}\sum_n (\boldsymbol{u}^\top \boldsymbol{x}_n)^2 = \boldsymbol{u}^\top \Big(\frac{1}{N}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top\Big)\boldsymbol{u} = \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u},$$

so we maximize $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$ subject to $\|\boldsymbol{u}\| = 1$ (the constraint prevents the trivial blow-up of scaling $\boldsymbol{u}$). Introduce a Lagrange multiplier λ for the constraint and set the gradient of $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} + \lambda(1 - \boldsymbol{u}^\top \boldsymbol{u})$ to zero:

$$2\boldsymbol{\Sigma}\boldsymbol{u} - 2\lambda \boldsymbol{u} = \boldsymbol{0} \implies \boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}. \tag{19.1}$$

So the optimal direction $\boldsymbol{u}$ is an *eigenvector* of the covariance matrix, and the variance it captures is $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} = \boldsymbol{u}^\top (\lambda \boldsymbol{u}) = \lambda$, its eigenvalue. To maximize variance we take the eigenvector of the *largest* eigenvalue, the *first principal component*; the next-largest gives the second principal component (orthogonal to the first, since $\boldsymbol{\Sigma}$ is symmetric), and so on. The top k eigenvectors span the k-dimensional PCA subspace, and projecting the data onto them is the dimensionality reduction. The maximized quantity $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u} / \boldsymbol{u}^\top \boldsymbol{u}$ is the *Rayleigh quotient*, whose maximum over all $\boldsymbol{u}$ is exactly the largest eigenvalue, a fact worth remembering independently.

Example 19.1 (PCA on the covariance from Chapter 2). We already did the hard part in Chapter 2. There we took the five points $(1,2), (2,3), (3,5), (4,4), (5,6)$, centered them about their mean $(3,4)$, and computed the sample covariance

$$\boldsymbol{\Sigma} = \begin{bmatrix} 2.5 & 2.25 \\ 2.25 & 2.5 \end{bmatrix},$$

with eigenvalues $\lambda_1 = 4.75$ and $\lambda_2 = 0.25$ and eigenvectors $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1,1)$ and $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1,-1)$. That eigendecomposition *is* principal component analysis. The first principal component is the 45° direction $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1,1)$, along which the data is most spread, and it captures a fraction $\lambda_1/(\lambda_1+\lambda_2) = 4.75/(4.75+0.25) = 4.75/5 = 0.95$, that is 95%, of the total variance. Reducing these two-dimensional points to one dimension by keeping only their coordinate along $\boldsymbol{u}_1$ discards just 5% of the variance, the small amount of spread in the perpendicular direction $\boldsymbol{u}_2$. To project a point, take its inner product with $\boldsymbol{u}_1$: the centered point $(2,2)$ (which is $(5,6)$ minus the mean) projects to $\boldsymbol{u}_1^\top (2,2) = \frac{1}{\sqrt{2}}(2+2) = \frac{4}{\sqrt{2}} = 2.83$, its single PCA coordinate. ◊

For a self-contained drill, the next example runs PCA end to end on a fresh data set, computing the covariance, solving the characteristic equation by hand, and projecting.

Example 19.2 (PCA from scratch). Take the four points $(1,1), (2,3), (3,2), (4,4)$. *Center:* the mean is $\big(\frac{1+2+3+4}{4}, \frac{1+3+2+4}{4}\big) = (2.5, 2.5)$, so the centered points are $(-1.5,-1.5), (-0.5, 0.5), (0.5,-0.5), (1.5, 1.5)$. *Covariance* $\boldsymbol{\Sigma} = \frac{1}{N}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$:

$$\Sigma_{11} = \tfrac{1}{4}(1.5^2 + 0.5^2 + 0.5^2 + 1.5^2) = \tfrac{5}{4} = 1.25 = \Sigma_{22}, \quad \Sigma_{12} = \tfrac{1}{4}(2.25 - 0.25 - 0.25 + 2.25) = \tfrac{4}{4} = 1.0,$$

so $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 1.25 & 1.0 \\ 1.0 & 1.25 \end{smallmatrix}\right]$. *Eigenvalues* from $\det(\boldsymbol{\Sigma} - \lambda I) = (1.25 - \lambda)^2 - 1 = 0$, i.e. $1.25 - \lambda = \pm 1$, giving $\lambda_1 = 2.25$ and $\lambda_2 = 0.25$. *Eigenvectors:* for $\lambda_1 = 2.25$, $(1.25 - 2.25)u_1 + u_2 = 0 \Rightarrow u_2 = u_1$, so $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1, 1)$; for $\lambda_2 = 0.25$, $u_2 = -u_1$, so $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1, -1)$. *Variance explained* by the first component is $\lambda_1/(\lambda_1 + \lambda_2) = 2.25/2.5 = 0.90$, so one dimension keeps 90% of the spread. *Projecting* the centered point $(1.5, 1.5)$ onto $\boldsymbol{u}_1$ gives the single PCA coordinate $\boldsymbol{u}_1^\top(1.5, 1.5) = \frac{1}{\sqrt{2}}(3) = 2.12$. The data lies almost entirely along the 45° line, exactly what $\boldsymbol{u}_1$ captures. ◊

19.3 Choosing the number of components

How many principal components to keep is decided by how much variance they explain. Because the kth eigenvalue is the variance along the kth component, the fraction of total variance captured by the top k components is $(\sum_{i=1}^{k} \lambda_i)/(\sum_{i=1}^{D} \lambda_i)$. A *scree plot* graphs the eigenvalues in decreasing order, and one looks for the "elbow" where they level off into a flat tail of negligible directions, keeping the components before it. A common rule is to keep enough components to explain a target fraction, say 90% or 95%, of the variance. In the examples above, the first component already explains at least 90%, so a single dimension suffices; in a real image data set, a few dozen components out of thousands of pixels often capture nearly all the variance, the quantitative form of the manifold hypothesis.

19.4 PCA via the SVD, and probabilistic PCA

In practice PCA is computed not by forming the covariance and diagonalizing it but through the *singular value decomposition* of the (centered) data matrix $X = U\Sigma V^\top$. The right singular vectors V are the eigenvectors of $X^\top X$, hence (up to the $1/N$) of the covariance, so they *are* the principal components, and the squared singular values are proportional to the eigenvalues. The SVD route is more accurate numerically (it avoids squaring the data into $X^\top X$, which worsens conditioning) and is how every software library computes PCA. It also ties PCA back to the matrix factorizations that underlie so much of the subject.

PCA also has a generative, probabilistic form, *probabilistic PCA* [39]. It models each data point as a linear function of a low-dimensional Gaussian latent variable plus Gaussian noise, $\boldsymbol{x} = W\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}$ with $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, I)$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$. The maximum-likelihood solution recovers the principal subspace, so classical PCA is the zero-noise limit of this model. The probabilistic version brings the usual Bayesian advantages: it handles missing data, it can be fit by the EM algorithm of Chapter 18 (the latent $\boldsymbol{z}$ is the hidden variable), and it gives a likelihood for comparing models and detecting outliers, turning the geometric PCA into a full latent-variable model.

19.5 The linear autoencoder is PCA

There is a striking bridge to the neural networks of Part IV. An *autoencoder* is a network trained to reproduce its own input through a narrow bottleneck: an encoder maps the input down to a small code, a decoder maps the code back up, and the network minimizes the reconstruction error $\sum_n \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|^2$. Consider the simplest case, a *linear* autoencoder with a k-unit bottleneck, encoder $\boldsymbol{z} = W_e\boldsymbol{x}$ and decoder $\hat{\boldsymbol{x}} = W_e^\top \boldsymbol{z}$ (tied weights), and no nonlinearities. Minimizing the reconstruction error over W_e turns out to recover exactly the PCA subspace: the optimal encoder projects onto the span of the top k eigenvectors of the data covariance.

The reason is the minimum-reconstruction-error view of Section 19.1. With $P = W_e^\top W_e$ a rank-k projection, the reconstruction error is $\sum_n \|\boldsymbol{x}_n - P\boldsymbol{x}_n\|^2 = N\ \mathrm{tr}[(I - P)\boldsymbol{\Sigma}] = N(\mathrm{tr}\,\boldsymbol{\Sigma} - \mathrm{tr}(P\boldsymbol{\Sigma}))$, so minimizing it maximizes $\mathrm{tr}(P\boldsymbol{\Sigma})$ over rank-k projections, which is maximized when P projects onto the top-k eigenvectors of $\boldsymbol{\Sigma}$, the same answer as Eq. (19.1). So a linear autoencoder *is* PCA. The value of the autoencoder framing is what happens when the linearity is dropped: a deep autoencoder with nonlinear activations can model a *curved* low-dimensional manifold rather than a flat subspace, learning a nonlinear dimensionality reduction that PCA cannot. PCA is the linear skeleton of the autoencoder, and the autoencoder is PCA set free to bend.

19.6 Independent component analysis

PCA finds *uncorrelated* directions of maximum variance, which is the right goal for compression but not always for separation. The motivating problem is the *cocktail party*: several people speak at once, several microphones each record a different mixture of all the voices, and the task is to recover the individual voices from the mixtures. The voices are statistically *independent* sources, and the recordings are linear mixtures of them. *Independent component analysis* (ICA) seeks the unmixing that makes the recovered signals as independent as possible, going beyond mere uncorrelatedness.

The distinction is crucial. Uncorrelatedness (what PCA delivers) is only the absence of *linear* relationship, the second-moment condition $\mathbb{E}[s_i s_j] = \mathbb{E}[s_i]\mathbb{E}[s_j]$, whereas independence is the much stronger condition that the joint factorizes, $p(s_1, \dots, s_m) = \prod_i p(s_i)$. ICA achieves the stronger condition by exploiting *non-Gaussianity*: a fundamental result says that sums of independent variables tend toward Gaussian (the central limit theorem of Chapter 2), so a mixture of independent sources is "more Gaussian" than the individual sources, and therefore the unmixing directions that make the recovered signals *maximally non-Gaussian* are the ones that recover the original independent sources. This is why ICA cannot work if the sources are themselves Gaussian: with Gaussian sources, uncorrelated already implies independent, the non-Gaussianity signal vanishes, and the mixing direction cannot be identified. PCA decorrelates by variance; ICA separates by independence; and the difference is exactly the difference between second-order and higher-order structure in the data.

19.7 Whitening: PCA as a change of units

Projecting onto the principal components is a rotation that lines the axes up with the directions of variance. *Whitening* (or *sphering*) takes one further step: after rotating, it rescales each axis by the inverse square root of its variance, so that the transformed data has covariance equal to the identity. The result is a cloud that is round and unit-scaled in every direction, with all the elliptical structure removed. Whitening is the standard preprocessing for independent component analysis, and it is a clean way to see what the eigendecomposition buys us.

Write the spectral decomposition of the covariance as $\mathbf{\Sigma} = U\Lambda U^\top$, where the columns of U are the orthonormal eigenvectors and $\Lambda = \mathrm{diag}(\lambda_1, \dots, \lambda_D)$ holds the eigenvalues. The whitening transform is

$$W = \Lambda^{-1/2} U^\top, \qquad \boldsymbol{z} = W\boldsymbol{x}, \tag{19.2}$$

with $\Lambda^{-1/2} = \mathrm{diag}(\lambda_1^{-1/2}, \dots, \lambda_D^{-1/2})$. The $U^\top$ rotates the data into the eigenbasis (the PCA step), and the $\Lambda^{-1/2}$ divides each new coordinate by its standard deviation $\sqrt{\lambda_i}$. The covariance of the whitened data is the identity: using $U^\top U = I$,

$$\mathrm{Cov}(\boldsymbol{z}) = W\mathbf{\Sigma}W^\top = \Lambda^{-1/2}U^\top (U\Lambda U^\top) U\Lambda^{-1/2} = \Lambda^{-1/2}\Lambda\,\Lambda^{-1/2} = I. \tag{19.3}$$

Every direction now carries unit variance, and all correlations are gone. The whitening matrix is not unique: any rotation RW with $R^\top R = I$ also whitens, since $(RW)\mathbf{\Sigma}(RW)^\top = R\,I\,R^\top = I$. This freedom to rotate after whitening is exactly the gap that ICA later fills, because second-order statistics alone cannot fix the rotation.

Example 19.3 (Whitening a 2×2 covariance). Take the centered covariance

$$\mathbf{\Sigma} = \begin{bmatrix} 2.5 & 1.5 \\ 1.5 & 2.5 \end{bmatrix},$$

whose eigenvalues are $\lambda_1 = 4$ and $\lambda_2 = 1$ (from $\det(\mathbf{\Sigma} - \lambda I) = (2.5-\lambda)^2 - 1.5^2 = 0$, so $2.5 - \lambda = \pm 1.5$) with eigenvectors $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1,1)$ and $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1,-1)$. Assemble $U = [\boldsymbol{u}_1\ \boldsymbol{u}_2]$ and $\Lambda = \mathrm{diag}(4,1)$, so $\Lambda^{-1/2} = \mathrm{diag}(\frac{1}{2}, 1)$. The whitening transform of Eq. (19.2) is

$$W = \Lambda^{-1/2}U^\top = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ 1 & -1 \end{bmatrix}.$$

Verify the whitened covariance is the identity. First $\mathbf{\Sigma}W^\top$: writing $W^\top = \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & 1 \\ \frac{1}{2} & -1 \end{bmatrix}$,

$$\mathbf{\Sigma}W^\top = \begin{bmatrix} 2.5 & 1.5 \\ 1.5 & 2.5 \end{bmatrix} \frac{1}{\sqrt{2}} \begin{bmatrix} \frac{1}{2} & 1 \\ \frac{1}{2} & -1 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 2 & 1 \\ 2 & -1 \end{bmatrix},$$

since the top-left entry is $2.5(\frac{1}{2}) + 1.5(\frac{1}{2}) = 2$ and the top-right is $2.5(1) + 1.5(-1) = 1$, and likewise for the second row. Then

$$W\boldsymbol{\Sigma}W^{\top} = \frac{1}{\sqrt{2}}\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\\ 1 & -1\end{bmatrix}\frac{1}{\sqrt{2}}\begin{bmatrix}2 & 1\\ 2 & -1\end{bmatrix} = \frac{1}{2}\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = I,$$

because the top-left is $\frac{1}{2}(\frac{1}{2}\cdot 2 + \frac{1}{2}\cdot 2) = \frac{1}{2}(2) = 1$, the off-diagonal is $\frac{1}{2}(\frac{1}{2}\cdot 1 + \frac{1}{2}(-1)) = 0$, and the bottom-right is $\frac{1}{2}(1\cdot 1 + (-1)(-1)) = \frac{1}{2}(2) = 1$. Whitening worked: the transformed covariance is the identity. *Whitening a point.* The centered point $\boldsymbol{x} = (2, 2)$ lies purely along $\boldsymbol{u}_1$, at distance $\boldsymbol{u}_1^{\top}\boldsymbol{x} = \frac{1}{\sqrt{2}}(4) = 2\sqrt{2} \approx 2.83$. Applying W gives $\boldsymbol{z} = W(2, 2) = (\sqrt{2}, 0) \approx (1.41, 0)$: the high-variance $\boldsymbol{u}_1$ coordinate 2.83 has been divided by $\sqrt{4} = 2$ to read 1.41, while the $\boldsymbol{u}_2$ coordinate is 0 as before. The long axis has been shrunk to match the short one, and the ellipse has become a circle. ◊

19.8 Variance explained and the scree plot

Section 19.3 stated the rule for choosing the number of components: keep enough top eigenvalues to reach a target fraction of the total variance. This section runs that calculation on numbers and draws the scree plot that goes with it. Recall the two quantities. The fraction of variance carried by component i alone is $\lambda_i / \sum_j \lambda_j$, and the fraction carried by the top k components together is the cumulative ratio

$$R(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{D} \lambda_j}. \tag{19.4}$$

The scree plot graphs the eigenvalues in decreasing order; the cumulative curve $R(k)$ climbs from the first value to 1. One reads off the "elbow" where the eigenvalues flatten into a tail of negligible directions.

Example 19.4 (A scree calculation). A five-dimensional data set has covariance eigenvalues, sorted in decreasing order,

$$\lambda_1 = 4, \quad \lambda_2 = 1, \quad \lambda_3 = 0.5, \quad \lambda_4 = 0.3, \quad \lambda_5 = 0.2.$$

The total variance is the sum (equivalently $\operatorname{tr}\boldsymbol{\Sigma}$), $\sum_j \lambda_j = 4 + 1 + 0.5 + 0.3 + 0.2 = 6$. The individual fractions are

$$\tfrac{4}{6} = 0.667, \quad \tfrac{1}{6} = 0.167, \quad \tfrac{0.5}{6} = 0.083, \quad \tfrac{0.3}{6} = 0.050, \quad \tfrac{0.2}{6} = 0.033.$$

The cumulative curve of Eq. (19.4) accumulates these:

$$R(1) = \tfrac{4}{6} = 0.667, \quad R(2) = \tfrac{5}{6} = 0.833, \quad R(3) = \tfrac{5.5}{6} = 0.917, \quad R(4) = \tfrac{5.8}{6} = 0.967, \quad R(5) = \tfrac{6}{6} = 1.$$

So one component keeps 66.7% of the variance, two keep 83.3%, and three keep 91.7%. To reach a 90% target we need $k = 3$ components, cutting the dimension from five to three while discarding under a tenth of the spread. The scree plot (Fig. 19.2) shows the elbow: $\lambda_1 = 4$ towers over the rest, $\lambda_2 = 1$ steps down, and from λ_3 on the bars form a low flat tail. The drop from λ_2 to λ_3 is the elbow, and keeping the two or three components before the tail is the natural choice. ◊

19.9 A worked cocktail party: why uncorrelated is not enough

Section 19.6 argued in words that PCA's uncorrelated directions cannot separate mixed sources and that ICA needs non-Gaussianity. Numbers make the gap concrete. Suppose two independent sources s_1, s_2 (two speakers) are mixed into two recordings by a 2×2 mixing matrix, $\boldsymbol{x} = A\boldsymbol{s}$, with

$$A = \begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}.$$

Each microphone hears a blend of both voices: $x_1 = 2s_1 + s_2$ and $x_2 = s_1 + 2s_2$. For a frame where the sources take values $\boldsymbol{s} = (3, -1)$,

$$\boldsymbol{x} = A\boldsymbol{s} = \begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}\begin{bmatrix}3\\ -1\end{bmatrix} = \begin{bmatrix}2\cdot 3 + 1\cdot(-1)\\ 1\cdot 3 + 2\cdot(-1)\end{bmatrix} = \begin{bmatrix}5\\ 1\end{bmatrix},$$

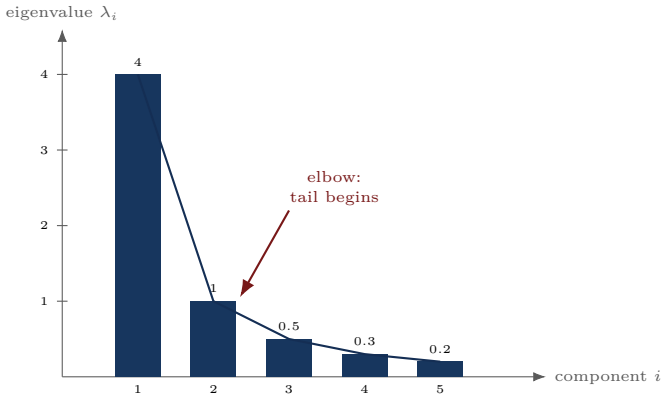


Figure 19.2. The eigenvalues $4, 1, 0.5, 0.3, 0.2$ in decreasing order. The first dominates and the second steps down; from the third on the bars form a low flat tail. The elbow at the drop into the tail marks where extra components stop paying for themselves: keeping the first two or three components captures 83% or 92% of the variance, and the rest is negligible spread.

and for $\boldsymbol{s} = (1, -2)$ we get $\boldsymbol{x} = (0, -3)$. The separation task is to recover $\boldsymbol{s}$ from the recordings $\boldsymbol{x}$. Since A is invertible ($\det A = 4 - 1 = 3$), an unmixing matrix exists,

$$A^{-1} = \frac{1}{3}\begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix},$$

and indeed $A^{-1}(5, 1) = \frac{1}{3}(2 \cdot 5 - 1, -5 + 2) = \frac{1}{3}(9, -3) = (3, -1) = \boldsymbol{s}$, the original sources. The whole problem is that we are handed only $\boldsymbol{x}$, never A, so the unmixing matrix has to be *learned* from the recordings alone.

Here is where PCA falls short. PCA whitens the recordings, producing signals that are uncorrelated and unit-variance (Section 19.7). But uncorrelatedness pins down the unmixing only up to a rotation: by Eq. (19.3), if W whitens then so does RW for any rotation R, and all of these rotated versions are equally uncorrelated and unit-variance. The second-order statistics, the covariance, are blind to R. So PCA narrows the candidate unmixings to a one-parameter family of rotations of the whitened data and then has no second-order reason to prefer one rotation over another. The true sources sit at one particular rotation, and PCA cannot find which.

Non-Gaussianity breaks the tie. By the central limit theorem of Chapter 2, a sum of independent variables drifts toward a Gaussian, so each mixed recording x_i, being a weighted sum of the independent sources, looks *more Gaussian* than the sources themselves. Turning that around: among all rotations of the whitened data, the directions that make the recovered signals *least* Gaussian (most peaked or most heavy-tailed, measured by kurtosis or negentropy) are the ones that undo the mixing and recover the sources. ICA searches the rotation R for maximal non-Gaussianity, which is exactly the higher-order information PCA never uses.

To turn "maximally non-Gaussian" into something a computer can optimize, we need a numerical measure of how far a distribution sits from Gaussian. The classic one is *kurtosis*. For a zero-mean variable y, the kurtosis is the fourth-order quantity

$$\operatorname{kurt}(y) = \mathbb{E}[y^4] - 3\big(\mathbb{E}[y^2]\big)^2, \tag{19.5}$$

defined so that it equals exactly 0 for a Gaussian. A positive kurtosis means the distribution is *heavy-tailed and peaked* (super-Gaussian, like a spiky speech signal); a negative kurtosis means it is *flat and bounded* (sub-Gaussian, like a uniform signal). Either sign is a departure from Gaussian, so ICA maximizes $|\operatorname{kurt}|$ (or a more robust surrogate, the negentropy) of the projected signal. The Gaussian sits alone at kurt $= 0$, which is precisely why a Gaussian source leaves ICA no signal to climb.

Example 19.5 (Kurtosis measures non-Gaussianity). Standardize each variable to zero mean and unit variance, so $\mathbb{E}[y^2] = 1$ and Eq. (19.5) reduces to $\operatorname{kurt}(y) = \mathbb{E}[y^4] - 3$. Compare three sources. *Gaussian.* For a standard Gaussian the fourth moment is $\mathbb{E}[y^4] = 3$, so kurt $= 3 - 3 = 0$, the baseline. *A symmetric binary source* taking ± 1 with equal probability (a crude two-level signal): $\mathbb{E}[y^2] = \frac{1}{2}(1) + \frac{1}{2}(1) = 1$ already, and $\mathbb{E}[y^4] = \frac{1}{2}(1) + \frac{1}{2}(1) = 1$, so kurt $= 1 - 3 = -2$. It is strongly sub-Gaussian: all its mass sits at the two extremes, the opposite of a Gaussian's central peak. *A uniform source* on $[-\sqrt{3}, \sqrt{3}]$ (chosen so the variance is 1): the fourth moment is $\mathbb{E}[y^4] = \frac{1}{2\sqrt{3}} \int_{-\sqrt{3}}^{\sqrt{3}} y^4 \, dy = \frac{1}{2\sqrt{3}} \cdot \frac{2(\sqrt{3})^5}{5} = \frac{(\sqrt{3})^4}{5} = \frac{9}{5} = 1.8$, so kurt $= 1.8 - 3 = -1.2$, again sub-Gaussian but less extreme than the binary

Figure 19.3. Both clouds are whitened, so both are uncorrelated and unit-variance, and PCA cannot tell them apart. Left: whitened Gaussians are rotationally symmetric, so the solid and dashed axis pairs are equally valid and no rotation is special; ICA cannot recover the sources. Right: a non-Gaussian distribution has visible structure (here heavy spikes along two directions), so exactly one pair of axes lines up with it. ICA finds that pair by maximizing non-Gaussianity, which is the higher-order information PCA never sees.

source. Both real sources land well away from 0, so ICA can detect them by their kurtosis. A Gaussian source would sit exactly at 0 with no direction to favour, and the separation collapses. The sign and size of the kurtosis are the higher-order structure PCA, which stops at the second moment $\mathbb{E}[y^2]$, can never see. ◊

PCA versus ICA

Both factor the data into a small set of directions, but they optimize different things. *PCA* rotates to the *uncorrelated* directions of maximum variance, using only the covariance (second-order statistics). It is the right tool for compression and denoising, where keeping the most variance is the goal, and its directions are orthogonal. *ICA* rotates to the *independent* directions, using higher-order statistics (non-Gaussianity, measured by kurtosis or negentropy). It is the right tool for blind source separation, and its directions need not be orthogonal. PCA is the second-order skeleton; ICA adds the higher-order rotation that separation demands. If the sources are Gaussian, the higher-order structure vanishes and ICA reduces to PCA, recovering the sources only up to an unidentifiable rotation.

Remark 19.6 *(The two-step recipe for ICA).* Most ICA algorithms split the work in two. First *whiten* the data with Eq. (19.2), which uses second-order statistics to reduce the unmixing to a pure rotation. Then *rotate* to maximize non-Gaussianity, which uses higher-order statistics to fix the rotation. PCA does the first step and stops; ICA adds the second. This is the precise sense in which PCA captures second-order structure (variance and correlation) while ICA reaches the higher-order structure that separation requires.

19.10 A numerical ICA unmixing, end to end

Remark 19.6 gave the two-step recipe in words: whiten, then rotate to maximize non-Gaussianity. This section carries it out on numbers small enough to check by hand, taking a whitened two-dimensional data set and finding the rotation that recovers the sources. The whitening step (the first half of the recipe) was worked in Example 19.3, so we start from data that is already whitened and concentrate on the rotation that whitening leaves undetermined.

Suppose whitening has already produced the four data points

$$\boldsymbol{x}_1 = (\sqrt{2},\, 0), \quad \boldsymbol{x}_2 = (0,\, -\sqrt{2}), \quad \boldsymbol{x}_3 = (0,\, \sqrt{2}), \quad \boldsymbol{x}_4 = (-\sqrt{2},\, 0). \tag{19.6}$$

These four points form a cross lying on the axes. First confirm the data really is whitened, that is, that its sample covariance is the identity. The points are already centered (they sum to $\mathbf{0}$), so the covariance is $\boldsymbol{\Sigma} = \frac{1}{4}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$, and

$$\Sigma_{11} = \tfrac{1}{4}\big(2 + 0 + 0 + 2\big) = 1, \qquad \Sigma_{22} = \tfrac{1}{4}\big(0 + 2 + 2 + 0\big) = 1, \tag{19.7}$$

while the off-diagonal $\Sigma_{12} = \frac{1}{4}(\sqrt{2}\cdot 0 + 0 + 0 + 0) = 0$. So $\boldsymbol{\Sigma} = I$: the data is white, every direction carries unit variance, and second-order statistics are exhausted. By the freedom of Eq. (19.3) any rotation of this cloud is equally

white, so PCA stops here with no way to choose among rotations. ICA chooses the rotation that makes the recovered coordinates maximally non-Gaussian.

Measure non-Gaussianity with the kurtosis of Eq. (19.5). Project the data onto the unit vector $\boldsymbol{u}(\theta) = (\cos\theta, \sin\theta)$, giving the recovered coordinate $y_n = \boldsymbol{u}(\theta)^\top \boldsymbol{x}_n$, and ask for the θ that maximizes $|\mathrm{kurt}(y)|$. Because the data is white, every projection already has unit variance, $\mathbb{E}[y^2] = 1$, so the kurtosis reduces to $\mathrm{kurt}(y) = \mathbb{E}[y^4] - 3$ and only the fourth moment varies with θ.

Example 19.7 (Finding the unmixing rotation by kurtosis). Take the whitened data (19.6) and sweep the rotation angle θ. The four projections onto $\boldsymbol{u}(\theta) = (\cos\theta, \sin\theta)$ are

$$y_1 = \sqrt{2}\cos\theta, \quad y_2 = -\sqrt{2}\sin\theta, \quad y_3 = \sqrt{2}\sin\theta, \quad y_4 = -\sqrt{2}\cos\theta. \tag{19.8}$$

Try the whitened axes, $\theta = 0$. Then $\boldsymbol{u} = (1, 0)$ and the projections are $(\sqrt{2},\, 0,\, 0,\, -\sqrt{2})$, so

$$\begin{aligned} \mathbb{E}[y^2] &= \tfrac{1}{4}(2 + 0 + 0 + 2) = 1, \\ \mathbb{E}[y^4] &= \tfrac{1}{4}(4 + 0 + 0 + 4) = 2, \end{aligned} \qquad \mathrm{kurt} = 2 - 3 = -1. \tag{19.9}$$

Try the diagonal, $\theta = 45^\circ$. Then $\boldsymbol{u} = \frac{1}{\sqrt{2}}(1, 1)$, and each projection $y_n = \frac{1}{\sqrt{2}}(x_{n,1} + x_{n,2})$ works out to $(1,\, -1,\, 1,\, -1)$, so $\mathbb{E}[y^2] = 1$ as before but $\mathbb{E}[y^4] = \frac{1}{4}(1 + 1 + 1 + 1) = 1$, giving $\mathrm{kurt} = 1 - 3 = -2$. The diagonal direction is *more* non-Gaussian (further from the Gaussian's kurt $= 0$) than the axis direction. To see that 45° is the true optimum and not just better than 0, write the fourth moment in closed form. Using $\cos^4\theta + \sin^4\theta = \frac{3}{4} + \frac{1}{4}\cos 4\theta$,

$$\mathbb{E}[y^4] = \tfrac{1}{4}\big(2(\sqrt{2}\cos\theta)^4 + 2(\sqrt{2}\sin\theta)^4\big) = 2(\cos^4\theta + \sin^4\theta) = \tfrac{3}{2} + \tfrac{1}{2}\cos 4\theta, \tag{19.10}$$

so the kurtosis is

$$\mathrm{kurt}(\theta) = \mathbb{E}[y^4] - 3 = -\tfrac{3}{2} + \tfrac{1}{2}\cos 4\theta. \tag{19.11}$$

This is most negative, $\mathrm{kurt} = -2$, exactly when $\cos 4\theta = -1$, that is $4\theta = 180^\circ$ and $\theta = 45^\circ$ (or equivalently -45°, the perpendicular source). The unmixing rotation is therefore $\theta^\star = 45^\circ$. Stack the two source directions $\boldsymbol{u}(45^\circ) = \frac{1}{\sqrt{2}}(1, 1)$ and its perpendicular $\boldsymbol{u}(135^\circ) = \frac{1}{\sqrt{2}}(-1, 1)$ as the rows of the unmixing matrix,

$$W = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \tag{19.12}$$

Applying it to the data recovers the sources $\boldsymbol{s}_n = W\boldsymbol{x}_n$:

$$\boldsymbol{s}_1 = \tfrac{1}{\sqrt{2}}(\sqrt{2},\, -\sqrt{2}) = (1,\, -1), \qquad \boldsymbol{s}_2 = \tfrac{1}{\sqrt{2}}(-\sqrt{2},\, -\sqrt{2}) = (-1,\, -1), \tag{19.13}$$

and likewise $\boldsymbol{s}_3 = (1, 1)$ and $\boldsymbol{s}_4 = (-1, 1)$. The four recovered points are the corners $(\pm 1, \pm 1)$. Each recovered coordinate now takes only the values ± 1: two clean sub-Gaussian binary sources, exactly the kind of two-level signal whose kurtosis is the most negative (-2, as in Example 19.5). Whitening rounded the cloud; the rotation found by maximizing $|\mathrm{kurt}|$ then turned it to the source axes. PCA would have left the cloud at $\theta = 0$, where the recovered coordinates are the less non-Gaussian mixtures, never reaching the sources. ◊

The closed form (19.11) also makes the Gaussian failure of Section 19.6 quantitative. If the whitened data were a round Gaussian blob, every projection would have the same kurtosis 0, so the $\cos 4\theta$ term would be absent and (19.11) would read $\mathrm{kurt}(\theta) = 0$ for all θ: a flat objective with no maximum to climb. The $\frac{1}{2}\cos 4\theta$ ripple is precisely the higher-order signal that non-Gaussian sources provide and Gaussian sources do not, and its amplitude is what ICA detects.

19.11 A probabilistic-PCA likelihood by hand

Section 19.4 introduced probabilistic PCA as the generative model $\boldsymbol{x} = W\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\varepsilon}$ with latent $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, I)$ and noise $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, and stated that its maximum-likelihood solution recovers the principal subspace. This section makes those claims concrete: it computes the model covariance, evaluates the data log-likelihood on a tiny case, and shows the maximum-likelihood parameters reading off the top eigenvalues.

Because $\boldsymbol{x}$ is a linear function of the independent Gaussians $\boldsymbol{z}$ and $\boldsymbol{\varepsilon}$, it is itself Gaussian, $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}, C)$, with mean $\boldsymbol{\mu}$ and covariance built from the two sources of randomness. The latent term contributes $W \operatorname{Cov}(\boldsymbol{z}) W^\top = WW^\top$ and the noise contributes $\sigma^2 I$, and the two are independent, so they add:

$$C = WW^\top + \sigma^2 I. \tag{19.14}$$

This is the central object. The latent factors carve out a low-rank piece $WW^\top$ (rank equal to the number of latent dimensions), and the noise inflates every direction equally by σ^2. The model thus says the data is a thin pancake (the column space of W) blurred isotropically by spherical noise, which is exactly the picture of data near a low-dimensional subspace.

Example 19.8 (The probabilistic-PCA covariance). Take a two-dimensional observation ($d = 2$) with a single latent factor ($q = 1$), loading matrix and noise

$$W = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, \qquad \sigma^2 = 1. \tag{19.15}$$

The low-rank piece is the outer product $WW^\top = \left[\begin{smallmatrix} 2 \\ 0 \end{smallmatrix}\right]\left[\begin{smallmatrix} 2 & 0 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 4 & 0 \\ 0 & 0 \end{smallmatrix}\right]$, so the model covariance (19.14) is

$$C = \begin{bmatrix} 4 & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}, \tag{19.16}$$

with determinant $\det C = 5 \cdot 1 - 0 = 5$ and inverse $C^{-1} = \left[\begin{smallmatrix} 1/5 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. The eigenvalues of C are 5 along the loading direction $(1, 0)$ and 1 along $(0, 1)$. The latent factor has stretched the variance along its direction from the noise floor $\sigma^2 = 1$ up to $\sigma^2 + \|W\|^2 = 1 + 4 = 5$, while the perpendicular direction, which the factor does not touch, sits at the noise floor 1. The eigenvalue gap, 5 versus 1, is the signature of one strong latent factor above isotropic noise. ◊

Now fit the model by maximum likelihood. For N centered points with sample covariance $S = \frac{1}{N}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$, the Gaussian log-likelihood, written with the trace identity $\sum_n \boldsymbol{x}_n^\top C^{-1} \boldsymbol{x}_n = N \operatorname{tr}(C^{-1} S)$, is

$$\log p(X \mid W, \sigma^2) = -\frac{N}{2}\Big[\, d \log(2\pi) + \log|C| + \operatorname{tr}(C^{-1} S) \,\Big]. \tag{19.17}$$

The data enters only through S and only through the single scalar $\operatorname{tr}(C^{-1} S)$, so the fit depends on S alone. This is why the maximum-likelihood solution can be written in closed form in terms of the eigenvalues of S.

Proposition 19.9 (Maximum-likelihood probabilistic PCA). Let the sample covariance S have eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$ with orthonormal eigenvectors $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_d$, and fit the model with q latent dimensions. The maximum-likelihood noise variance is the average of the discarded eigenvalues,

$$\sigma^2_{\mathrm{ML}} = \frac{1}{d - q} \sum_{i=q+1}^{d} \lambda_i, \tag{19.18}$$

and the maximum-likelihood loading matrix has columns built from the top q eigenvectors, $\boldsymbol{w}_i = \sqrt{\lambda_i - \sigma^2_{\mathrm{ML}}}\, \boldsymbol{u}_i$ for $i = 1, \ldots, q$ (up to an arbitrary rotation of the latent space). So the principal subspace of S is exactly the column space of W_{ML}, and classical PCA is the limit $\sigma^2 \to 0$.

Example 19.10 (Maximum-likelihood fit and a likelihood comparison). Suppose a data set of $N = 10$ points in $d = 2$ dimensions has sample covariance with eigenvalues $\lambda_1 = 5$ and $\lambda_2 = 1$ and eigenvectors $\boldsymbol{u}_1 = (1, 0)$ and $\boldsymbol{u}_2 = (0, 1)$, so that $S = \left[\begin{smallmatrix} 5 & 0 \\ 0 & 1 \end{smallmatrix}\right]$. Fit probabilistic PCA with one latent factor, $q = 1$. By Proposition 19.9 the noise variance is the lone discarded eigenvalue,

$$\sigma^2_{\mathrm{ML}} = \frac{1}{d - q} \sum_{i=2}^{2} \lambda_i = \frac{1}{2 - 1} \lambda_2 = 1, \tag{19.19}$$

and the single loading column is $\boldsymbol{w}_1 = \sqrt{\lambda_1 - \sigma^2_{\mathrm{ML}}}\,\boldsymbol{u}_1 = \sqrt{5-1}\,(1,0) = (2,0)$. These are exactly the W and σ^2 of Example 19.8, so the fitted model reconstructs $C = WW^\top + \sigma^2 I = \left[\begin{smallmatrix}5&0\\0&1\end{smallmatrix}\right] = S$ and matches the data covariance perfectly. *Evaluate the log-likelihood* (19.17) at this fit. With $C = S$ the trace term is $\mathrm{tr}(C^{-1}S) = \mathrm{tr}(I) = 2$, the log-determinant is $\log|C| = \log 5 = 1.609$, and $d\log(2\pi) = 2(1.838) = 3.676$, so

$$\begin{aligned}\log p &= -\tfrac{10}{2}\left[\,3.676 + 1.609 + 2\,\right]\\ &= -5\,(7.285) = -36.43.\end{aligned} \tag{19.20}$$

Compare a worse model. Fit instead a purely spherical Gaussian, $C = 3I$ (one shared variance for both axes, the average $\frac{5+1}{2} = 3$ of the eigenvalues). Then $\log|C| = \log 9 = 2.197$ and $\mathrm{tr}(C^{-1}S) = \frac{1}{3}\,\mathrm{tr}(S) = \frac{5+1}{3} = 2$, so

$$\log p_{\mathrm{sph}} = -5\left[\,3.676 + 2.197 + 2\,\right] = -5\,(7.873) = -39.36. \tag{19.21}$$

The probabilistic-PCA fit scores higher by $39.36 - 36.43 = 2.93$, which is exactly $5(\log 9 - \log 5) = 5\log 1.8$: the spherical model pays a log-determinant penalty for spreading variance into the low-variance direction that the data does not occupy, while probabilistic PCA puts the variance where the eigenvalues say it belongs. The likelihood (19.17) is what lets the model be compared and selected, the advantage Section 19.4 promised over geometric PCA. ◊

19.12 Reconstruction error equals the discarded eigenvalues

The minimum-reconstruction-error view of Section 19.1 said PCA chooses the subspace that loses the least when points are projected onto it and reconstructed. Section 19.5 turned that into the identity $\sum_n \|\boldsymbol{x}_n - P\boldsymbol{x}_n\|^2 = N(\mathrm{tr}\,\boldsymbol{\Sigma} - \mathrm{tr}(P\boldsymbol{\Sigma}))$ for a rank-k projection P. This section pins the loss to a single clean number: when P projects onto the top k principal components, the total squared reconstruction error equals the sum of the *discarded* eigenvalues, scaled by N.

The reason is short. Write the covariance in its eigenbasis, $\boldsymbol{\Sigma} = \sum_{i=1}^{d} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top$, and let P project onto the top k eigenvectors. Then $\mathrm{tr}\,\boldsymbol{\Sigma} = \sum_i \lambda_i$ is the total variance, and $\mathrm{tr}(P\boldsymbol{\Sigma}) = \sum_{i=1}^{k} \lambda_i$ keeps only the retained ones, because P annihilates the discarded eigenvectors. Subtracting,

$$\sum_{n=1}^{N} \|\boldsymbol{x}_n - P\boldsymbol{x}_n\|^2 = N\big(\mathrm{tr}\,\boldsymbol{\Sigma} - \mathrm{tr}(P\boldsymbol{\Sigma})\big) = N\sum_{i=k+1}^{d} \lambda_i. \tag{19.22}$$

Equivalently, the squared error per point is the variance the projection threw away, $\sum_{i>k} \lambda_i$, so the eigenvalues are not just "variance captured" but literally "error incurred" when their directions are dropped. The scree plot of Section 19.8 reads twice over: the bars above the elbow are the variance kept, and the bars below it are the reconstruction error paid.

Example 19.11 (Reconstruction error matches the dropped eigenvalue). Take the four centered points

$$\boldsymbol{x}_1 = (3,1), \quad \boldsymbol{x}_2 = (1,3), \quad \boldsymbol{x}_3 = (-1,-3), \quad \boldsymbol{x}_4 = (-3,-1), \tag{19.23}$$

which already have mean $\mathbf{0}$. The covariance $\boldsymbol{\Sigma} = \frac{1}{4}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$ has entries $\Sigma_{11} = \frac{1}{4}(9+1+1+9) = 5$, $\Sigma_{22} = \frac{1}{4}(1+9+9+1) = 5$, and $\Sigma_{12} = \frac{1}{4}(3+3+3+3) = 3$, so

$$\boldsymbol{\Sigma} = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}, \tag{19.24}$$

with eigenvalues from $(5-\lambda)^2 - 9 = 0$, i.e. $5 - \lambda = \pm 3$, giving $\lambda_1 = 8$ (eigenvector $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1,1)$) and $\lambda_2 = 2$ (eigenvector $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1,-1)$). *Keep the top $k = 1$ component* $\boldsymbol{u}_1$ and reconstruct each point as $\hat{\boldsymbol{x}}_n = (\boldsymbol{u}_1^\top \boldsymbol{x}_n)\,\boldsymbol{u}_1$. For $\boldsymbol{x}_1 = (3,1)$ the coordinate along $\boldsymbol{u}_1$ is $\boldsymbol{u}_1^\top \boldsymbol{x}_1 = \frac{1}{\sqrt{2}}(3+1) = 2\sqrt{2}$, so

$$\hat{\boldsymbol{x}}_1 = 2\sqrt{2}\cdot\tfrac{1}{\sqrt{2}}(1,1) = (2,2), \qquad \boldsymbol{x}_1 - \hat{\boldsymbol{x}}_1 = (1,-1), \tag{19.25}$$

a squared error $\|\boldsymbol{x}_1 - \hat{\boldsymbol{x}}_1\|^2 = 1 + 1 = 2$. By the symmetry of the four points each one reconstructs the same way: $\boldsymbol{x}_2 = (1,3) \to \hat{\boldsymbol{x}}_2 = (2,2)$ with residual $(-1,1)$, $\boldsymbol{x}_3 \to (-2,-2)$, $\boldsymbol{x}_4 \to (-2,-2)$, every residual of squared length 2. The total is

$$\sum_{n=1}^{4} \|\boldsymbol{x}_n - \hat{\boldsymbol{x}}_n\|^2 = 2 + 2 + 2 + 2 = 8. \tag{19.26}$$

Check against (19.22): the discarded eigenvalue is $\lambda_2 = 2$, and $N\lambda_2 = 4 \cdot 2 = 8$, matching exactly. One more cross-check: the discarded variance equals the spread along $\boldsymbol{u}_2$, and indeed $\sum_n (\boldsymbol{u}_2^\top \boldsymbol{x}_n)^2 = 4 \cdot (\sqrt{2})^2 = 8$ since each $\boldsymbol{u}_2^\top \boldsymbol{x}_n = \pm\sqrt{2}$. The error of dropping a component is the variance that component carried, no more and no less. $\lozenge$

Three readings of an eigenvalue

For a covariance $\boldsymbol{\Sigma}$ with eigenvalue λ_i and eigenvector $\boldsymbol{u}_i$, the number λ_i admits three equivalent readings, each used in this chapter. It is the *variance* of the data projected onto $\boldsymbol{u}_i$ (Section 19.2); it is the *reconstruction error* incurred per point if direction $\boldsymbol{u}_i$ is dropped from the projection (Eq. (19.22)); and in probabilistic PCA it is the *signal-plus-noise* power $\|\boldsymbol{w}_i\|^2 + \sigma^2$ along that direction, splitting into a latent part and the noise floor (Proposition 19.9). Variance kept, error paid, and signal above noise are the same spectrum seen three ways.

Notes and References

The two views of PCA, the eigendecomposition derivation, the SVD route, and probabilistic PCA follow Chapter 12 of Bishop [3]; probabilistic PCA is due to Tipping and Bishop [39]. The linear-autoencoder-equals-PCA result of Section 19.5 connects this chapter to the neural networks of Chapter 10 and is the linear special case of the autoencoders used for nonlinear dimensionality reduction. Independent component analysis and the non-Gaussianity principle are due to Hyvärinen and Oja [20]. The Rayleigh-quotient characterization of PCA and the eigendecomposition of the covariance tie this chapter back to Chapter 2, where the covariance and its spectral decomposition first appeared; PCA is that decomposition put to work.

Exercises

Exercise 19.1. A centered two-dimensional data set has sample covariance $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$. Find the principal components and the fraction of variance each captures.

Solution. The eigenvalues solve $\det(\boldsymbol{\Sigma} - \lambda I) = (2-\lambda)^2 - 1 = 0$, so $2 - \lambda = \pm 1$ and $\lambda_1 = 3$, $\lambda_2 = 1$. For $\lambda_1 = 3$: $(\boldsymbol{\Sigma} - 3I)\boldsymbol{u} = \left[\begin{smallmatrix} -1 & 1 \\ 1 & -1 \end{smallmatrix}\right]\boldsymbol{u} = \mathbf{0}$ gives $u_1 = u_2$, so $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1,1)$. For $\lambda_2 = 1$: $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1,-1)$. The first principal component $\boldsymbol{u}_1$ captures $\lambda_1/(\lambda_1+\lambda_2) = 3/4 = 75\%$ of the variance, the second $1/4 = 25\%$. Reducing to one dimension along $\boldsymbol{u}_1$ keeps 75% of the variance. □

Exercise 19.2. Show that the variance of data projected onto a unit vector $\boldsymbol{u}$ is $\boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$, and that maximizing it subject to $\|\boldsymbol{u}\| = 1$ leads to the eigenvalue equation $\boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}$.

Solution. For centered data the projection onto $\boldsymbol{u}$ is $\boldsymbol{u}^\top \boldsymbol{x}_n$, with mean zero, so its variance is $\frac{1}{N}\sum_n (\boldsymbol{u}^\top \boldsymbol{x}_n)^2 = \frac{1}{N}\sum_n \boldsymbol{u}^\top \boldsymbol{x}_n \boldsymbol{x}_n^\top \boldsymbol{u} = \boldsymbol{u}^\top (\frac{1}{N}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top)\boldsymbol{u} = \boldsymbol{u}^\top \boldsymbol{\Sigma} \boldsymbol{u}$. Maximizing this subject to $\boldsymbol{u}^\top \boldsymbol{u} = 1$ with a Lagrange multiplier λ means setting $\nabla[\boldsymbol{u}^\top \boldsymbol{\Sigma}\boldsymbol{u} + \lambda(1 - \boldsymbol{u}^\top \boldsymbol{u})] = 2\boldsymbol{\Sigma}\boldsymbol{u} - 2\lambda \boldsymbol{u} = \mathbf{0}$, giving $\boldsymbol{\Sigma}\boldsymbol{u} = \lambda \boldsymbol{u}$. So the maximizing direction is an eigenvector of $\boldsymbol{\Sigma}$, and its captured variance $\boldsymbol{u}^\top \boldsymbol{\Sigma}\boldsymbol{u} = \lambda$ is the eigenvalue, maximized by the largest one. □

Exercise 19.3. Explain why a linear autoencoder trained to minimize reconstruction error recovers the PCA subspace, and what changes when the autoencoder is made nonlinear.

Solution. A linear autoencoder with encoder W_e and tied decoder $W_e^\top$ reconstructs $\hat{\boldsymbol{x}} = P\boldsymbol{x}$ with $P = W_e^\top W_e$ a rank-k projection. The reconstruction error is $\sum_n \|\boldsymbol{x}_n - P\boldsymbol{x}_n\|^2 = N \operatorname{tr}[(I-P)\boldsymbol{\Sigma}] = N(\operatorname{tr}\boldsymbol{\Sigma} - \operatorname{tr}(P\boldsymbol{\Sigma}))$, so minimizing it maximizes $\operatorname{tr}(P\boldsymbol{\Sigma})$ over rank-k projections, which is achieved when P projects onto the top-k eigenvectors of $\boldsymbol{\Sigma}$, exactly the PCA subspace. So the linear autoencoder and PCA give the same subspace. Adding nonlinear activations lets the encoder and decoder represent a curved manifold instead of a flat subspace, so a nonlinear (deep) autoencoder can perform a nonlinear dimensionality reduction that captures structure PCA's linear projection cannot. □

Exercise 19.4. A centered two-dimensional data set has the diagonal sample covariance $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 9 & 0 \\ 0 & 4 \end{smallmatrix}\right]$. Construct the whitening transform $W = \Lambda^{-1/2} U^\top$, verify that the whitened covariance $W\boldsymbol{\Sigma}W^\top$ is the identity, and whiten the centered point $\boldsymbol{x} = (3,4)$.

Solution. The covariance is already diagonal, so its eigenvalues are $\lambda_1 = 9$, $\lambda_2 = 4$ and its eigenvectors are the standard axes, $U = I$. Then $\Lambda^{-1/2} = \operatorname{diag}(9^{-1/2}, 4^{-1/2}) = \operatorname{diag}(\frac{1}{3}, \frac{1}{2})$, and since $U^\top = I$,

$$W = \Lambda^{-1/2} U^\top = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}.$$

The whitened covariance is

$$W\boldsymbol{\Sigma}W^\top = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} \begin{bmatrix} 9 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{2} \end{bmatrix} = \begin{bmatrix} \frac{9}{9} & 0 \\ 0 & \frac{4}{4} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I,$$

so each axis has been rescaled to unit variance. Whitening the point gives $\boldsymbol{z} = W(3,4) = (\frac{1}{3}\cdot 3,\ \frac{1}{2}\cdot 4) = (1,2)$: the first coordinate is divided by $\sqrt{9} = 3$ and the second by $\sqrt{4} = 2$, the standard deviations along the two axes. □

Exercise 19.5. A data set has covariance eigenvalues $5, 3, 1, 1$ (in decreasing order). Compute the fraction of variance explained by the top k components for $k = 1, 2, 3$, and state how many components are needed to retain at least 85% of the variance.

Solution. The total variance is $5 + 3 + 1 + 1 = 10$. The cumulative fraction of Eq. (19.4) is

$$R(1) = \tfrac{5}{10} = 0.50, \qquad R(2) = \tfrac{8}{10} = 0.80, \qquad R(3) = \tfrac{9}{10} = 0.90.$$

So one component keeps 50%, two keep 80%, and three keep 90%. Two components fall short of the 85% target ($80\% < 85\%$), while three exceed it ($90\% \geq 85\%$), so $k = 3$ components are needed. The last two eigenvalues are equal, a flat tail with no elbow between them, so there is no reason to keep one without the other. □

Exercise 19.6. A probabilistic-PCA model in $d = 2$ dimensions with one latent factor has loading $W = \left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]$ and noise variance $\sigma^2 = 2$. Form the model covariance $C = WW^\top + \sigma^2 I$, give its determinant and inverse, and find its eigenvalues. Then state which sample covariance eigenvalues and eigenvectors would make this W and σ^2 the maximum-likelihood fit.

Solution. The outer product is $WW^\top = \left[\begin{smallmatrix}1\\2\end{smallmatrix}\right]\left[\begin{smallmatrix}1&2\end{smallmatrix}\right] = \left[\begin{smallmatrix}1&2\\2&4\end{smallmatrix}\right]$, so

$$C = \begin{bmatrix}1 & 2\\ 2 & 4\end{bmatrix} + \begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix} = \begin{bmatrix}3 & 2\\ 2 & 6\end{bmatrix}, \quad \det C = 3\cdot 6 - 2\cdot 2 = 14, \quad C^{-1} = \tfrac{1}{14}\begin{bmatrix}6 & -2\\ -2 & 3\end{bmatrix}.$$

The eigenvalues solve $(3-\lambda)(6-\lambda) - 4 = \lambda^2 - 9\lambda + 14 = 0$, so $\lambda = \frac{9 \pm \sqrt{81-56}}{2} = \frac{9\pm 5}{2}$, giving $\lambda_1 = 7$ and $\lambda_2 = 2$. The top eigenvector solves $(C - 7I)\boldsymbol{u} = \left[\begin{smallmatrix}-4 & 2\\ 2 & -1\end{smallmatrix}\right]\boldsymbol{u} = \mathbf{0}$, i.e. $u_2 = 2u_1$, so $\boldsymbol{u}_1 = \frac{1}{\sqrt{5}}(1,2)$. By Proposition 19.9 this W and σ^2 are the maximum-likelihood fit when the sample covariance has top eigenvalue $\lambda_1 = \|W\|^2 + \sigma^2 = 5 + 2 = 7$ along the eigenvector $\frac{1}{\sqrt{5}}(1,2)$ (matching $W = \sqrt{\lambda_1 - \sigma^2}\,\boldsymbol{u}_1 = \sqrt{5}\,\boldsymbol{u}_1 = (1,2)$) and discarded eigenvalue $\lambda_2 = \sigma^2 = 2$. The eigenvalues of the model covariance are the signal-plus-noise power 7 along the factor and the noise floor 2 across it. □

Exercise 19.7. A centered two-dimensional data set has the four points $(2,0), (0,2), (-2,0), (0,-2)$. Compute the covariance and its eigenvalues, project each point onto the top principal component, and verify that the total squared reconstruction error equals N times the discarded eigenvalue.

Solution. The points are centered (they sum to $\mathbf{0}$). The covariance $\boldsymbol{\Sigma} = \frac{1}{4}\sum_n \boldsymbol{x}_n \boldsymbol{x}_n^\top$ has $\Sigma_{11} = \frac{1}{4}(4+0+4+0) = 2$, $\Sigma_{22} = \frac{1}{4}(0+4+0+4) = 2$, and $\Sigma_{12} = 0$, so $\boldsymbol{\Sigma} = \left[\begin{smallmatrix}2&0\\0&2\end{smallmatrix}\right] = 2I$. Both eigenvalues equal 2, so the data is isotropic and *any* unit vector is a principal direction; take $\boldsymbol{u}_1 = (1,0)$. Projecting onto $\boldsymbol{u}_1$ and reconstructing $\hat{\boldsymbol{x}}_n = (\boldsymbol{u}_1^\top \boldsymbol{x}_n)\boldsymbol{u}_1$ keeps only the first coordinate: $\hat{\boldsymbol{x}}_1 = (2,0)$ with residual $\mathbf{0}$, $\hat{\boldsymbol{x}}_2 = (0,0)$ with residual $(0,2)$, $\hat{\boldsymbol{x}}_3 = (-2,0)$ with residual $\mathbf{0}$, $\hat{\boldsymbol{x}}_4 = (0,0)$ with residual $(0,-2)$. The squared errors are $0, 4, 0, 4$, summing to 8. The discarded eigenvalue is $\lambda_2 = 2$, and $N\lambda_2 = 4 \cdot 2 = 8$, so (19.22) holds. The lesson of the isotropic case is that no direction is better than another: dropping any one component costs the same $N\lambda = 8$, because all directions carry equal variance and PCA has no preferred axis. □

Exercise 19.8. Explain the difference between the directions PCA finds and those ICA finds, and why ICA fails if the source signals are Gaussian.

Solution. PCA finds orthogonal directions of maximum variance, which are mutually *uncorrelated* (zero second-order, linear, dependence). ICA finds directions that make the recovered signals *independent*, the stronger condition that the joint density factorizes, by seeking maximally non-Gaussian projections (since mixing independent sources makes them more Gaussian by the central limit theorem, unmixing maximizes non-Gaussianity). If the sources are Gaussian, then uncorrelated already implies independent, so there is no extra higher-order structure to exploit: any rotation of uncorrelated Gaussians is still uncorrelated Gaussians, the non-Gaussianity objective is flat, and ICA cannot identify the unmixing direction. ICA needs non-Gaussian sources. □

CHAPTER 20

Sampling and Monte Carlo Methods

When you cannot do the integral, roll dice. Enough dice, rolled the right way, compute anything.

after Professor Peter Chin, ENGS 106

Bayesian machine learning is full of integrals that cannot be done. The predictive distribution of Chapter 6 was a Gaussian integral only because everything was Gaussian; in general the posterior is intractable, and so is any expectation over it. When exact inference fails (the loopy graphs of Chapter 17, the non-conjugate posteriors of Chapter 9), the practical recourse is to *approximate* the integral by sampling: draw points from the distribution and average. This is the *Monte Carlo* idea, and its remarkable property is that its accuracy does not degrade with dimension. The catch is that drawing samples from a complicated distribution is itself hard, especially when the distribution is known only up to its normalizing constant, exactly the situation with posteriors. *Markov chain Monte Carlo* (MCMC) solves this by building a random walk whose long-run distribution is the target. This chapter, which rounds out the inference toolkit, covers Monte Carlo estimation, the basic samplers, and the two MCMC methods that made Bayesian computation practical: Metropolis-Hastings and Gibbs sampling.

Roadmap

Suggested reading: Bishop [3], Chapter 11; MacKay [27] for an information-theoretic treatment.

We cover: Monte Carlo estimation and its dimension-free error (Section 20.1); rejection and importance sampling (Section 20.2); Markov chain Monte Carlo (Section 20.3); the Metropolis-Hastings algorithm (Section 20.4); and Gibbs sampling (Section 20.5).

20.1 Monte Carlo estimation

The quantity we usually want is an expectation, $\mathbb{E}_p[f] = \int f(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x}$, a predictive mean, a marginal probability, a Bayes estimate. The Monte Carlo method replaces the integral with an average over samples: draw S independent points $\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(S)}$ from p and estimate

$$\mathbb{E}_p[f] \approx \frac{1}{S} \sum_{s=1}^{S} f(\boldsymbol{x}^{(s)}). \tag{20.1}$$

The estimator is *unbiased*, its expectation is exactly $\mathbb{E}_p[f]$, and by the law of large numbers it converges as $S \to \infty$. Its error is governed by the variance of the average: the standard deviation of the estimate is $\sigma_f/\sqrt{S}$, where σ_f^2 is the variance of f under p. Two features of this $1/\sqrt{S}$ rate matter. It is slow, quartering the error needs sixteen times the samples, but it is *independent of the dimension* of $\boldsymbol{x}$: a thousand-dimensional integral converges at the same rate as a one-dimensional one, which is why Monte Carlo is the method of choice in high dimensions where grid-based numerical integration is hopeless (a grid with 10 points per axis needs 10^{1000} points in a thousand dimensions).

Example 20.1 (Estimating π by Monte Carlo). Scatter S points uniformly in the unit square $[0,1]^2$ and count the fraction that land inside the quarter circle $x^2 + y^2 \le 1$. The quarter circle has area $\pi/4$ and the square has area 1, so the fraction of points inside estimates $\pi/4$, and four times the fraction estimates π. Each point is a Bernoulli trial (inside or out) with success probability $\pi/4 \approx 0.785$, so the standard error of the fraction is $\sqrt{p(1-p)/S} \approx \sqrt{0.785(0.215)/S} = 0.41/\sqrt{S}$, and the error in the estimate of π is four times that, about $1.64/\sqrt{S}$. With $S = 10{,}000$

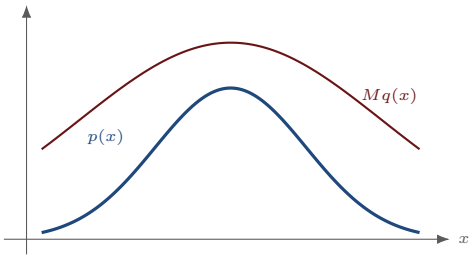


Figure 20.1. Draw a candidate from the easy envelope Mq (chosen so $Mq \geq p$ everywhere) and accept it with probability $p(x)/Mq(x)$. Accepted points (under the inner curve p) are exact draws from p; the rest (between the curves) are discarded. A loose envelope wastes most candidates, which is why the method struggles in high dimensions.

points the error is roughly $1.64/100 = 0.016$, so a few thousand random darts pin down π to a couple of decimals. To halve the error one must *quadruple* S, the price of the $1/\sqrt{S}$ rate. ◊

20.2 Rejection and importance sampling

Monte Carlo presumes we can draw from p. For standard distributions we can, but for a general p, often known only up to its normalizer, we cannot, and two classical tricks sample with the help of an easy *proposal* distribution q.

Rejection sampling uses q as an envelope. Choose a constant M so that $Mq(\boldsymbol{x}) \geq p(\boldsymbol{x})$ everywhere, draw a candidate $\boldsymbol{x} \sim q$, and accept it with probability $p(\boldsymbol{x})/(Mq(\boldsymbol{x}))$; accepted samples are exact draws from p. The picture is throwing darts under the envelope Mq and keeping those that also fall under p. It is correct but can be wildly inefficient: if the envelope is much larger than p, almost every candidate is rejected, and in high dimensions the required M grows so large that the acceptance rate collapses toward zero.

Importance sampling avoids rejection entirely by reweighting. Since $\mathbb{E}_p[f] = \int f(\boldsymbol{x}) \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} q(\boldsymbol{x})\, d\boldsymbol{x} = \mathbb{E}_q\big[f(\boldsymbol{x}) \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\big]$, we draw from q and weight each sample by the *importance weight* $w(\boldsymbol{x}) = p(\boldsymbol{x})/q(\boldsymbol{x})$:

$$\mathbb{E}_p[f] \approx \frac{1}{S} \sum_{s=1}^{S} w(\boldsymbol{x}^{(s)})\, f(\boldsymbol{x}^{(s)}), \qquad \boldsymbol{x}^{(s)} \sim q. \tag{20.2}$$

No samples are thrown away, but the estimate is only as good as the match between q and p: if q misses a region where p is large, the few samples that land there carry enormous weights and the estimate has huge variance. Both methods need a good proposal, and finding one in high dimensions is hard, which is what motivates a fundamentally different approach.

20.3 Markov chain Monte Carlo

The idea of *Markov chain Monte Carlo* is to give up on independent samples and instead build a *Markov chain*, a random walk in which each state depends only on the previous one, designed so that its long-run *stationary distribution* is the target p. Run the chain long enough and its state is distributed as p; continue running and the successive states, though correlated, can be averaged as in Eq. (20.1). The decisive advantage for Bayesian work is that the chain can be constructed using p only *up to its normalizing constant*, because the transition rule depends only on *ratios* $p(\boldsymbol{x}')/p(\boldsymbol{x})$ in which the unknown normalizer cancels. Since posteriors are precisely the distributions we know up to a constant (the evidence we cannot compute), MCMC is the tool that made Bayesian inference practical at scale.

20.4 The Metropolis-Hastings algorithm

The *Metropolis-Hastings* algorithm is the general recipe for building such a chain. From the current state $\boldsymbol{x}$, propose a candidate $\boldsymbol{x}'$ from a proposal distribution $q(\boldsymbol{x}' \mid \boldsymbol{x})$, and accept the move with probability

$$A(\boldsymbol{x}', \boldsymbol{x}) = \min\left(1,\ \frac{p(\boldsymbol{x}')\, q(\boldsymbol{x} \mid \boldsymbol{x}')}{p(\boldsymbol{x})\, q(\boldsymbol{x}' \mid \boldsymbol{x})}\right); \tag{20.3}$$

if the move is rejected, the chain stays at $\boldsymbol{x}$ (and that state is recorded again). The acceptance ratio compares the target density at the proposed and current states: moves to higher-density regions ($p(\boldsymbol{x}') > p(\boldsymbol{x})$) are favored, but moves to lower density are sometimes accepted too, which is what lets the chain explore rather than collapse onto the mode. Crucially, p appears only through the ratio $p(\boldsymbol{x}')/p(\boldsymbol{x})$, so the normalizer cancels and an unnormalized target suffices. When the proposal is symmetric, $q(\boldsymbol{x}' \mid \boldsymbol{x}) = q(\boldsymbol{x} \mid \boldsymbol{x}')$, the q-factors cancel and the rule simplifies to the original Metropolis form $A = \min(1, p(\boldsymbol{x}')/p(\boldsymbol{x}))$. One can show this transition leaves p invariant (it satisfies detailed balance, $p(\boldsymbol{x})\, T(\boldsymbol{x}' \mid \boldsymbol{x}) = p(\boldsymbol{x}')\, T(\boldsymbol{x} \mid \boldsymbol{x}')$), so p is the chain's stationary distribution.

Example 20.2 (A Metropolis acceptance step). Sample from the standard Gaussian, whose unnormalized density is $p(x) \propto e^{-x^2/2}$, using a symmetric proposal (say a small Gaussian step), so the acceptance probability is $A = \min(1, p(x')/p(x))$. Suppose the chain is at $x = 0$, the mode. If it proposes $x' = 1$, the ratio is $p(1)/p(0) = e^{-1/2}/e^{0} = e^{-0.5} = 0.607$, so the uphill-to-downhill move is accepted with probability 0.607, often but not always, allowing the chain to wander away from the mode. If it instead proposes the nearer $x' = 0.5$, the ratio is $e^{-0.125} = 0.882$, accepted more readily because it gives up less density. Conversely, starting from $x = 1$ and proposing $x' = 0$ (toward the mode), the ratio is $p(0)/p(1) = e^{0}/e^{-0.5} = e^{0.5} = 1.65 > 1$, so the move is accepted with probability 1: downhill-to-uphill moves are always taken. Over many steps the chain spends time at each x in proportion to $e^{-x^2/2}$, reproducing the Gaussian. ◊

Stringing acceptance steps together gives the chain. The next example traces five steps with the proposals and the uniform accept/reject draws written in, so the whole mechanism is visible.

Example 20.3 (Five steps of a Metropolis chain). Target $p(x) \propto e^{-x^2/2}$, symmetric proposals, start at $x_0 = 0$. The acceptance probability is $A = \min(1, e^{-(x'^2 - x^2)/2})$; accept if the uniform draw $u < A$.

step	x	proposal x'	$A = \min(1, e^{-(x'^2 - x^2)/2})$	u	decision	new x
1	0.0	0.8	$e^{-0.32} = 0.73$	0.50	accept	0.8
2	0.8	1.6	$e^{-0.96} = 0.38$	0.70	reject	0.8
3	0.8	0.2	$e^{+0.30} \to 1$	0.40	accept	0.2
4	0.2	−0.5	$e^{-0.105} = 0.90$	0.30	accept	−0.5
5	−0.5	−1.4	$e^{-0.855} = 0.43$	0.60	reject	−0.5

The chain visits $0 \to 0.8 \to 0.8 \to 0.2 \to -0.5 \to -0.5$ (Fig. 20.2). Step 3 moves toward the mode and is always accepted ($A = 1$); steps 1 and 4 move slightly downhill and are accepted by the coin flip; steps 2 and 5 propose far into the tail and are rejected, so the chain repeats its state. Repeated states are real samples rather than wasted draws: they are how the chain spends more time in high-density regions. ◊

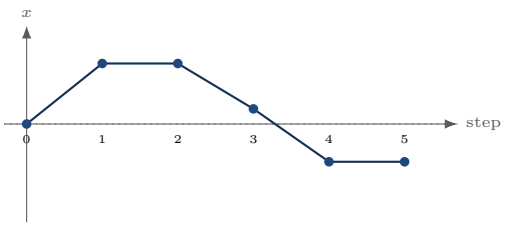


Figure 20.2. The five-step trace of Example 20.3. Flat segments are rejected proposals (the chain repeats its state); the walk wanders around the mode $x = 0$, spending time in proportion to the target density.

20.5 Gibbs sampling

Gibbs sampling is the special case of Metropolis-Hastings best suited to multi-dimensional distributions, and especially to graphical models. Instead of proposing a move in all variables at once, it updates one variable at a time, drawing each from its conditional distribution given the current values of all the others:

$$x_i^{\text{new}} \sim p(x_i \mid x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n), \tag{20.4}$$

sweeping through the variables in turn. These conditional distributions are often simple even when the full joint is intractable, and in a graphical model the conditional of a variable depends only on its Markov blanket (Chapter 16), so each update is local and cheap. Gibbs sampling is a Metropolis-Hastings chain in which the proposal is the exact conditional and the acceptance probability works out to exactly 1, so every proposed move is accepted, with no wasted steps. It is the natural sampler for the posteriors of hierarchical Bayesian models and for image MRFs, where the per-pixel conditional given the neighbors is trivial to sample.

Example 20.4 (Gibbs sampling a 2×2 joint). Sample from the joint $p(x, y)$ on two binary variables with $p(0,0) = 0.1$, $p(0,1) = 0.2$, $p(1,0) = 0.3$, $p(1,1) = 0.4$. The conditionals come from normalizing a row or column:

$$p(x{=}1 \mid y{=}0) = \tfrac{0.3}{0.1+0.3} = 0.75, \qquad p(x{=}1 \mid y{=}1) = \tfrac{0.4}{0.2+0.4} = 0.667,$$
$$p(y{=}1 \mid x{=}0) = \tfrac{0.2}{0.1+0.2} = 0.667, \qquad p(y{=}1 \mid x{=}1) = \tfrac{0.4}{0.3+0.4} = 0.571.$$

Start at $(0,0)$ and sweep, drawing each variable from its conditional (uniforms u shown): sample $x \mid y{=}0$ with $p(x{=}1) = 0.75$, $u = 0.50 < 0.75 \Rightarrow x = 1$, giving $(1,0)$; then $y \mid x{=}1$ with $p(y{=}1) = 0.571$, $u = 0.40 \Rightarrow y = 1$, giving $(1,1)$; then $x \mid y{=}1$ with $p(x{=}1) = 0.667$, $u = 0.80 > 0.667 \Rightarrow x = 0$, giving $(0,1)$. Each step moves only one coordinate and always along its exact conditional, so no step is ever rejected. Over many sweeps the chain visits the four states in proportion to $p(x, y)$, the $(1,1)$ cell most often. ◊

A few practical matters attend any MCMC method. The chain must be run for a *burn-in* period before its samples are used, to let it forget the arbitrary starting point and reach the stationary distribution. Successive samples are *correlated*, so the effective number of independent samples is smaller than the chain length, and a poorly designed proposal can make the chain *mix* slowly, exploring the distribution at a crawl. Diagnosing convergence and designing proposals that mix well are the practical art of MCMC, but the principle is the one above: a cleverly constructed random walk turns an intractable integral into a tractable average, and closes the inference toolkit this book has built, from the exact message passing of Chapter 17 to the approximate sampling here.

20.6 Worked estimates with numbers

The samplers of this chapter are easiest to trust once the arithmetic is on the page. This section carries three of them through on small, hand-checkable numbers: an importance-sampling estimate of an expectation, a rejection-sampling run with explicit accept tests, and the $1/\sqrt{S}$ shrinkage of the Monte Carlo standard error.

Importance sampling on numbers

We estimate $\mathbb{E}_p[f]$ for $f(x) = x^2$ under the standard Gaussian $p = \mathcal{N}(0,1)$, drawing the samples from a wider proposal $q = \mathcal{N}(0, 1.5^2)$. Since p has mean 0 and variance 1, the true value is $\mathbb{E}_p[x^2] = 1$, which we will recover. The wider proposal is the safe choice here: its tails are heavier than the target's, so no region of p is starved of samples, the failure mode flagged in Section 20.2. The importance weight is a ratio of two Gaussian densities, and the normalizers collapse into a single constant,

$$w(x) = \frac{p(x)}{q(x)} = \frac{(1)^{-1}\exp(-x^2/2)}{(1.5)^{-1}\exp(-x^2/(2\cdot 1.5^2))} = 1.5\,\exp\!\Big(-\tfrac{1}{2}\big(1-\tfrac{1}{2.25}\big)x^2\Big) = 1.5\,e^{-0.278\,x^2}. \tag{20.5}$$

The coefficient is $\frac{1}{2}(1 - 1/2.25) = \frac{1}{2}(0.5556) = 0.278$. The estimator is the sample average of $w(x)f(x)$ from Eq. (20.2), $\mathbb{E}_p[f] \approx \frac{1}{S}\sum_s w(x^{(s)})f(x^{(s)})$. Example 20.5 draws five points from q and runs the average.

Example 20.5 (An importance-sampling estimate). Five draws $x^{(s)} \sim q = \mathcal{N}(0, 1.5^2)$ come up as $-1.5, -0.5, 0.3, 0.9, 1.8$. With $f(x) = x^2$, target $p = \mathcal{N}(0,1)$, and the weight (20.5), the table collects each piece. The densities $p(x)$ and $q(x)$ are the Gaussian values; the weight w comes from the closed form $1.5\,e^{-0.278x^2}$ (a useful check on p/q).

x	$f(x)=x^2$	$p(x)$	$q(x)$	$w=p/q$	wf
-1.5	2.25	0.1295	0.1613	0.803	1.806
-0.5	0.25	0.3521	0.2516	1.399	0.350
0.3	0.09	0.3814	0.2607	1.463	0.132
0.9	0.81	0.2661	0.2221	1.198	0.970
1.8	3.24	0.0790	0.1295	0.610	1.976
				$\sum wf=$	5.234

The plain importance estimate is the average of the last column,

$$\mathbb{E}_p[x^2] \approx \frac{1}{S}\sum_{s=1}^{5} w(x^{(s)})f(x^{(s)}) = \frac{1.806+0.350+0.132+0.970+1.976}{5} = \frac{5.234}{5} = 1.047,$$

within 5% of the true value 1 on only five samples. Two checks confirm the mechanics. First, the weights average to $\frac{1}{5}(0.803+1.399+1.463+1.198+0.610) = \frac{1}{5}(5.473) = 1.095$, close to the value $\mathbb{E}_q[w]=1$ they must average to in the limit, since $\int (p/q)\,q = \int p = 1$. Second, dividing instead by that weight sum gives the *self-normalized* estimate $5.234/5.473 = 0.956$, which also brackets the truth and is the form one must use when p is known only up to its normalizer (the weights then carry an unknown common factor that cancels in the ratio). ◊

Rejection sampling on numbers

Take the unnormalized target $\tilde{p}(x) = x(2-x)$ on $[0,2]$, a downward parabola peaking at $\tilde{p}(1)=1$. The proposal is uniform on the interval, $q(x)=1/2$. To make $Mq(x) \geq \tilde{p}(x)$ everywhere we need $M\cdot\frac{1}{2}\geq 1$, so $M=2$ gives the flat envelope $Mq(x)=1$, exactly touching the peak. The accept test of Section 20.2 draws a candidate $x\sim q$ and a uniform $u\in[0,1]$, and accepts when

$$u \;\leq\; \frac{\tilde{p}(x)}{M\,q(x)} = \frac{x(2-x)}{1} = x(2-x). \tag{20.6}$$

Example 20.6 (A rejection-sampling run). Four candidates drawn from the uniform proposal, each with its own uniform u, run through the test (20.6). The accept threshold is just $\tilde{p}(x)=x(2-x)$ because the envelope equals 1.

candidate x	$\tilde{p}(x)=x(2-x)$	threshold $\tilde{p}/(Mq)$	u	decision
0.4	$0.4(1.6)=0.640$	0.640	0.30	accept
1.0	$1.0(1.0)=1.000$	1.000	0.55	accept
1.8	$1.8(0.2)=0.360$	0.360	0.50	reject
1.3	$1.3(0.7)=0.910$	0.910	0.40	accept

Three of the four candidates are accepted, an empirical acceptance rate of $3/4=0.75$. The lone rejection is the tail point $x=1.8$, where the target has dropped to 0.36 and the uniform $u=0.50$ lands above it (Fig. 20.3). The theoretical acceptance probability is the area under $\tilde{p}$ divided by the area under the envelope,

$$\mathbb{P}(\text{accept}) = \frac{\int_0^2 x(2-x)\,dx}{M\int_0^2 q(x)\,dx} = \frac{[x^2-x^3/3]_0^2}{2\cdot 1} = \frac{4-8/3}{2} = \frac{4/3}{2} = \frac{2}{3} \approx 0.667,$$

so 0.75 from four draws is a reasonable estimate of $2/3$. The gap between the target and the flat envelope, the wasted region in Fig. 20.3, is exactly the $1-2/3=1/3$ of candidates thrown away. A tighter envelope would raise the rate; a proposal shaped like a parabola rather than a flat line would discard almost nothing. ◊

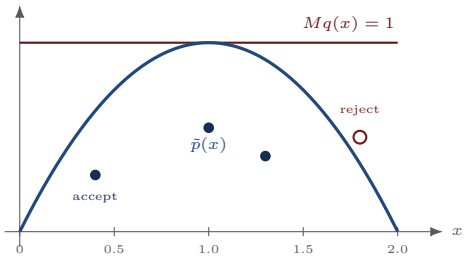


Figure 20.3. The four candidates of Example 20.6 under the flat envelope $Mq(x) = 1$ and the target $\tilde{p}(x) = x(2-x)$. Each dart sits at height $u\,Mq$ above its candidate x. The three filled darts fall under $\tilde{p}$ and are accepted; the open dart at $x = 1.8$ lands in the wasted gap between $\tilde{p}$ and the envelope and is rejected. The fraction of the envelope's area lying under $\tilde{p}$ is $2/3$, the long-run acceptance rate.

The Monte Carlo standard error

The error of a Monte Carlo average shrinks at the $1/\sqrt{S}$ rate of Section 20.1. In practice the population standard deviation σ_f is unknown, so we estimate it by the *sample* standard deviation s of the S values $f(x^{(s)})$ and report the *standard error*

$$\mathrm{SE} = \frac{s}{\sqrt{S}}. \tag{20.7}$$

The next example reads off the rate by computing the standard error at two sample sizes.

Example 20.7 (Standard error shrinks as $1/\sqrt{S}$). A Monte Carlo estimate of some expectation uses samples whose sample standard deviation is $s = 2.0$. At $S = 100$ samples the standard error is

$$\mathrm{SE}_{100} = \frac{2.0}{\sqrt{100}} = \frac{2.0}{10} = 0.200.$$

Quadrupling the sample size to $S = 400$ gives

$$\mathrm{SE}_{400} = \frac{2.0}{\sqrt{400}} = \frac{2.0}{20} = 0.100,$$

exactly half of SE_{100} (Fig. 20.4). This is the $1/\sqrt{S}$ rate in action: $\sqrt{400}/\sqrt{100} = \sqrt{4} = 2$, so four times the samples buys a factor of two in accuracy. Reaching $\mathrm{SE} \le 0.05$ would need $\sqrt{S} \ge 2.0/0.05 = 40$, that is $S \ge 1600$, sixteen times the original budget for a quarter of the error. The curve flattens fast, which is why Monte Carlo gives a couple of digits cheaply and further digits only at steep cost. ◊

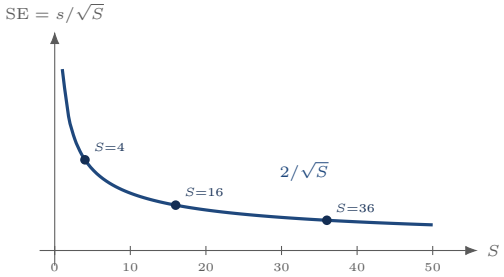


Figure 20.4. The standard error $\mathrm{SE} = s/\sqrt{S}$ for $s = 2$, plotted against the sample size S. The curve falls steeply at first and then flattens: doubling accuracy (halving SE) requires quadrupling S, the hallmark of the dimension-free but slow $1/\sqrt{S}$ Monte Carlo rate. Marked points sit at $S = 4, 16, 36$.

Notes and References

Monte Carlo estimation, rejection and importance sampling, and MCMC follow Chapter 11 of Bishop [3]; MacKay [27] gives a complementary information-theoretic treatment. The Metropolis algorithm is due to Metropolis et al. [30] and its generalization to Hastings [15]; Gibbs sampling was named and popularized for image models by Geman and Geman [12], whose work also underlies the MRF denoising of Chapter 16. The dimension-independent $1/\sqrt{S}$ convergence of Monte Carlo is the reason it dominates high-dimensional integration. MCMC is the computational engine of modern Bayesian statistics, making tractable the posteriors that Chapter 6 and Chapter 9 could only approximate analytically, and with it the inference toolkit of the book is complete: exact inference where the structure allows it, the Laplace approximation and variational bounds where it does not, and sampling as the general-purpose fallback.

Exercises

Exercise 20.1. You estimate π by the dartboard method of Example 20.1. Roughly how many samples are needed to get the estimate's standard error below 0.01?

Solution. The standard error of the π estimate is about $1.64/\sqrt{S}$ (four times the standard error $\sqrt{p(1-p)/S} \approx 0.41/\sqrt{S}$ of the inside-fraction, with $p = \pi/4$). Setting $1.64/\sqrt{S} \le 0.01$ gives $\sqrt{S} \ge 164$, so $S \ge 164^2 \approx 27{,}000$ samples. The slow $1/\sqrt{S}$ rate means two-decimal accuracy already costs tens of thousands of darts, and another decimal would cost a hundredfold more. □

Exercise 20.2. Explain why importance sampling can have very high variance when the proposal q is much narrower than the target p.

Solution. Importance sampling weights each sample drawn from q by $w(\boldsymbol{x}) = p(\boldsymbol{x})/q(\boldsymbol{x})$. If q is much narrower than p, then in the tails of p where q is tiny the weight p/q becomes enormous, but samples land there only rarely; the estimate is then dominated by a few huge-weight samples and is unreliable, with high variance, because the average is controlled by rare events. In the worst case $\mathbb{E}_q[w^2]$ is infinite and the estimator has no finite variance at all. A good proposal must have heavier tails than the target, covering wherever p has mass. □

Exercise 20.3. Sampling from $p(x) \propto e^{-x^2/2}$ with a symmetric proposal, give the Metropolis acceptance probability for a move from $x = 2$ to $x' = 1$, and for a move from $x = 1$ to $x' = 2$.

Solution. For a symmetric proposal the acceptance probability is $\min(1, p(x')/p(x))$. Moving from $x = 2$ to $x' = 1$ (toward the mode, higher density): $p(1)/p(2) = e^{-1/2}/e^{-2} = e^{2-0.5} = e^{1.5} = 4.48 > 1$, so accept with probability 1. Moving from $x = 1$ to $x' = 2$ (away from the mode, lower density): $p(2)/p(1) = e^{-2}/e^{-0.5} = e^{-1.5} = 0.223$, so accept with probability 0.223. The chain always moves toward higher density and sometimes toward lower, which is what lets it explore the whole distribution rather than stick at the peak. □

Exercise 20.4. You estimate $\mathbb{E}_p[f]$ for $f(x) = x$ by importance sampling, drawing three samples from a proposal q. The samples and their densities are $x = 1$ with $p = 0.2$, $q = 0.4$; $x = 2$ with $p = 0.5$, $q = 0.4$; and $x = 3$ with $p = 0.3$, $q = 0.2$. Compute the importance weights, the plain importance estimate $\frac{1}{S}\sum_s w(x^{(s)})f(x^{(s)})$, and the self-normalized estimate $\sum_s wf / \sum_s w$.

Solution. The weights are $w = p/q$: at $x = 1$, $w = 0.2/0.4 = 0.5$; at $x = 2$, $w = 0.5/0.4 = 1.25$; at $x = 3$, $w = 0.3/0.2 = 1.5$. The weighted values $w\,f$ are $0.5 \cdot 1 = 0.5$, $1.25 \cdot 2 = 2.5$, and $1.5 \cdot 3 = 4.5$, summing to 7.5. The plain estimate divides by $S = 3$,

$$\mathbb{E}_p[f] \approx \frac{0.5 + 2.5 + 4.5}{3} = \frac{7.5}{3} = 2.5.$$

The weights sum to $0.5 + 1.25 + 1.5 = 3.25$, so the self-normalized estimate is $7.5/3.25 = 2.31$. The two differ because the weights here average to $3.25/3 = 1.08$ rather than exactly 1; the self-normalized form rescales by the observed weight total and is the version to use when p is unnormalized. As a sanity check, p here sums to 1, so the exact answer is $\sum_x p(x)x = 0.2(1) + 0.5(2) + 0.3(3) = 2.1$, which both estimates bracket on only three samples. □

Exercise 20.5. Rejection-sample the unnormalized target $\tilde{p}(x) = 3x^2$ on $[0, 1]$ using the uniform proposal $q(x) = 1$. Find the smallest envelope constant M with $Mq(x) \geq \tilde{p}(x)$ everywhere, give the long-run acceptance probability, and run the accept test $u \leq \tilde{p}(x)/(Mq(x))$ on the three candidates $(x, u) = (0.5, 0.20),\ (0.9, 0.70),\ (0.3, 0.10)$.

Solution. On $[0, 1]$ the target $\tilde{p}(x) = 3x^2$ is largest at $x = 1$, where $\tilde{p}(1) = 3$. Since $q(x) = 1$, the envelope condition $M \cdot 1 \geq 3x^2$ needs $M \geq 3$, so the smallest envelope is $M = 3$ with $Mq(x) = 3$. The long-run acceptance probability is the area under $\tilde{p}$ over the area under the envelope,

$$\mathbb{P}(\text{accept}) = \frac{\int_0^1 3x^2\,dx}{M\int_0^1 1\,dx} = \frac{[x^3]_0^1}{3 \cdot 1} = \frac{1}{3} \approx 0.333.$$

The accept threshold is $\tilde{p}(x)/(Mq(x)) = 3x^2/3 = x^2$. Testing each candidate: at $x = 0.5$, threshold 0.25, and $u = 0.20 \leq 0.25$, accept; at $x = 0.9$, threshold 0.81, and $u = 0.70 \leq 0.81$, accept; at $x = 0.3$, threshold 0.09, and $u = 0.10 > 0.09$, reject. Two of the three are accepted, an empirical rate of $2/3$. With only three candidates this is a noisy estimate of the true rate $1/3$; the wasted candidates are the price of the loose flat envelope, which sits far above $\tilde{p}$ except near $x = 1$. □

Exercise 20.6. Explain why MCMC methods can sample from a distribution known only up to its normalizing constant, and why this matters for Bayesian inference.

Solution. The Metropolis-Hastings acceptance probability depends on the target only through the ratio $p(\boldsymbol{x}')/p(\boldsymbol{x})$. If $p(\boldsymbol{x}) = \tilde{p}(\boldsymbol{x})/Z$ with unknown normalizer Z, the ratio is $\tilde{p}(\boldsymbol{x}')/\tilde{p}(\boldsymbol{x})$, in which Z cancels, so the chain can be run knowing only the unnormalized $\tilde{p}$. This matters because a Bayesian posterior is $p(\boldsymbol{\theta} \mid \mathcal{D}) = p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})/p(\mathcal{D})$, where the evidence $p(\mathcal{D}) = \int p(\mathcal{D} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})\,d\boldsymbol{\theta}$ is exactly the intractable integral we cannot compute. MCMC samples the posterior using only the numerator (likelihood times prior), sidestepping the unknown evidence, which is what makes Bayesian computation feasible. □

PART VIII

The Problem Sets

About the Problem Sets

ENGS 106 was taught half as theory and half as code. The graded work was a programming assignment and a term project, but the real week-to-week problem solving happened in the X-hour sessions, where each week's theory was turned into worked problems and small programs. The chapters that follow collect that problem solving into five problem sets, one per part of the book, plus the programming assignment and the project. Each problem is *restated in full* so the set stands on its own, and each is then worked from the start: the proofs are written out, the numbers are carried through, and the computational problems are described with the algorithm, the expected behavior, and the figure each produces.

The arc of the five sets mirrors the arc of the book.

Set	Chapters	What you do
PS1	Chapters 1 and 2	probability foundations: prove the sample variance unbiased, the covariance identity, Bayes and total probability, a Gaussian by hand
PS2	Chapters 7 and 8	classification: the perceptron convergence theorem, a logistic gradient step, k-nearest-neighbors, naive Bayes, and the confusion-matrix metrics
PS3	Chapters 10 to 13	deep learning: multi-output regression with a shared covariance, a sparse autoencoder's KL penalty, and the forward/reverse diffusion process
PS4	Chapters 14 and 15	kernels and SVMs: the two-point max-margin solution, the kernel perceptron, and the dual quadratic program solved with `cvxopt`
PS5	Chapters 16 to 18	graphical models: marginal versus conditional independence, explaining-away, and image denoising by iterated conditional modes

The programming assignment (Chapter 26) builds a working classifier from scratch, and the term project (Chapter 27) is an open machine-learning problem of the student's choosing. Where a problem set asks for a proof that also appears in a chapter, the solution here is the short, exam-ready version and the chapter carries the long one; cross-references point both ways.

CHAPTER 21

Problem Set 1: Probability Foundations

This first set drills the probability that every later method rests on: expectation algebra, the covariance identity, Bayes' rule with total probability, and a Gaussian computation by hand. It collects the Week 1 and Week 2 X-hour problems and pairs with Chapters 1 and 2.

Problem 1: The sample variance is unbiased

Let $x_1, \dots, x_N$ be independent draws from a distribution with known mean $\mathbb{E}[x] = \mu$ and variance $\mathrm{Var}(x) = \sigma^2$, so that $\mathbb{E}[x^2] = \mu^2 + \sigma^2$. Show that the estimator $\hat{\sigma}^2 = \frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2$ is unbiased, that is $\mathbb{E}[\hat{\sigma}^2] = \sigma^2$.

Solution. Expectation is linear, so push it inside the sum and evaluate one term. Expanding the square and using $\mathbb{E}[x_n] = \mu$ and $\mathbb{E}[x_n^2] = \mu^2 + \sigma^2$,

$$\mathbb{E}\big[(x_n - \mu)^2\big] = \mathbb{E}[x_n^2] - 2\mu\,\mathbb{E}[x_n] + \mu^2 = (\mu^2 + \sigma^2) - 2\mu^2 + \mu^2 = \sigma^2.$$

Every term has expectation σ^2, so

$$\mathbb{E}[\hat{\sigma}^2] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\big[(x_n - \mu)^2\big] = \frac{1}{N}\cdot N\sigma^2 = \sigma^2,$$

and the estimator is unbiased. The catch is that μ was *known*. If instead the sample mean $\bar{x} = \frac{1}{N}\sum_n x_n$ replaces μ, the deviations are measured from a point fit to the same data, the sum of squares shrinks, and $\mathbb{E}[\frac{1}{N}\sum(x_n - \bar{x})^2] = \frac{N-1}{N}\sigma^2$, biased low by the factor $(N-1)/N$. Dividing by $N-1$ instead of N (Bessel's correction) restores unbiasedness.

Problem 2: The covariance identity

Prove the identity $\mathrm{Cov}(X,Y) = \mathbb{E}[XY] - \mathbb{E}[X]\,\mathbb{E}[Y]$ from the definition $\mathrm{Cov}(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$, then compute it for the random vector that takes $(X,Y) = (1,2)$ and $(3,4)$ each with probability $\frac{1}{2}$, and state the correlation.

Solution. Expand the product inside the expectation and use linearity, remembering that $\mathbb{E}[X]$ and $\mathbb{E}[Y]$ are constants:

$$\mathrm{Cov}(X,Y) = \mathbb{E}\big[XY - X\,\mathbb{E}[Y] - Y\,\mathbb{E}[X] + \mathbb{E}[X]\mathbb{E}[Y]\big] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y] - \mathbb{E}[Y]\mathbb{E}[X] + \mathbb{E}[X]\mathbb{E}[Y] = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y].$$

For the two-point distribution, $\mathbb{E}[X] = \frac{1}{2}(1+3) = 2$, $\mathbb{E}[Y] = \frac{1}{2}(2+4) = 3$, and $\mathbb{E}[XY] = \frac{1}{2}(1\cdot 2 + 3\cdot 4) = \frac{1}{2}(2+12) = 7$, so $\mathrm{Cov}(X,Y) = 7 - (2)(3) = 1$. The variances are $\mathrm{Var}(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2 = \frac{1}{2}(1+9) - 4 = 1$ and $\mathrm{Var}(Y) = \frac{1}{2}(4+16) - 9 = 1$, so the correlation is $\rho = \mathrm{Cov}(X,Y)/(\sigma_X\sigma_Y) = 1/(1\cdot 1) = 1$: the two points lie on a rising line, perfect positive correlation.

Problem 3: Bayes' rule and the base rate

A disease is present in 1% *of a population. A test is* 95% *sensitive (it is positive on* 95% *of those with the disease) and* 90% *specific (negative on* 90% *of those without it). A randomly chosen person tests positive. What is the probability they have the disease?*

Solution. Write D for disease and $+$ for a positive test. The given numbers are $P(D) = 0.01$, $P(+ \mid D) = 0.95$, and $P(+ \mid \neg D) = 1 - 0.90 = 0.10$. Total probability gives the unconditional positive rate,

$$P(+) = P(+ \mid D)P(D) + P(+ \mid \neg D)P(\neg D) = (0.95)(0.01) + (0.10)(0.99) = 0.0095 + 0.099 = 0.1085,$$

and Bayes' rule gives the posterior,

$$P(D \mid +) = \frac{P(+ \mid D)P(D)}{P(+)} = \frac{0.0095}{0.1085} = 0.088.$$

Even after a positive test the probability of disease is only about 9%, because the disease is rare: the 0.10 false-positive rate acting on the 99% healthy majority produces nearly ten times as many false positives as there are true positives. This is the base-rate lesson of Chapter 2, and the reason a single positive screening test is rarely conclusive.

Problem 4: A Gaussian by hand

Let $X \sim \mathcal{N}(\mu = 5, \sigma^2 = 4)$*. Compute* $P(X \leq 7)$ *and* $P(3 \leq X \leq 7)$*, using* $\Phi(1) = 0.841$ *for the standard normal CDF.*

Solution. Standardize by subtracting the mean and dividing by $\sigma = \sqrt{4} = 2$. For $P(X \leq 7)$, the z-score is $z = (7 - 5)/2 = 1$, so $P(X \leq 7) = \Phi(1) = 0.841$. For the interval, 3 and 7 are one standard deviation either side of the mean ($z = \pm 1$), so

$$P(3 \leq X \leq 7) = \Phi(1) - \Phi(-1) = \Phi(1) - (1 - \Phi(1)) = 2\Phi(1) - 1 = 2(0.841) - 1 = 0.682,$$

the familiar fact that about 68% of a Gaussian's mass lies within one standard deviation of the mean.

In the Problem Sets

The accompanying notebook draws N samples from $\mathcal{N}(5, 4)$, checks that the known-mean variance estimator of Problem 1 converges to $\sigma^2 = 4$ while the sample-mean version is biased low by $(N - 1)/N$, and verifies the 68% interval of Problem 4 empirically by counting samples in $[3, 7]$.

CHAPTER 22

Problem Set 2: Classification

This set works the classifiers of Part III by hand: the perceptron's convergence guarantee, a logistic-regression gradient step, k-nearest-neighbors, naive Bayes with smoothing, and the metrics that score a classifier. It collects the Week 3 and Week 4 X-hour problems and pairs with Chapters 7 and 8.

Problem 1: The perceptron converges

Suppose the training data is linearly separable: there is a unit vector $\boldsymbol{w}^*$ and a margin $\gamma > 0$ with $t_n\, \boldsymbol{w}^{*\top}\boldsymbol{\phi}(\boldsymbol{x}_n) \geq \gamma$ for every example, and let $R = \max_n \|\boldsymbol{\phi}(\boldsymbol{x}_n)\|$. Show that the perceptron, updating $\boldsymbol{w} \leftarrow \boldsymbol{w} + t_n\boldsymbol{\phi}(\boldsymbol{x}_n)$ on each misclassified point and starting from $\boldsymbol{w} = \mathbf{0}$, makes at most $(R/\gamma)^2$ updates.

Solution. Track two quantities across the M updates. First the alignment with the true separator: each update adds $t_n\boldsymbol{\phi}(\boldsymbol{x}_n)$ to $\boldsymbol{w}$, so $\boldsymbol{w}^{*\top}\boldsymbol{w}$ grows by $t_n\boldsymbol{w}^{*\top}\boldsymbol{\phi}(\boldsymbol{x}_n) \geq \gamma$, giving $\boldsymbol{w}^{*\top}\boldsymbol{w} \geq M\gamma$ after M updates. Second the length: an update happens only on a misclassified point, where $t_n\boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x}_n) \leq 0$, so

$$\|\boldsymbol{w} + t_n\boldsymbol{\phi}(\boldsymbol{x}_n)\|^2 = \|\boldsymbol{w}\|^2 + 2\, t_n\boldsymbol{w}^{\top}\boldsymbol{\phi}(\boldsymbol{x}_n) + \|\boldsymbol{\phi}(\boldsymbol{x}_n)\|^2 \leq \|\boldsymbol{w}\|^2 + R^2,$$

and after M updates $\|\boldsymbol{w}\|^2 \leq MR^2$. Combine through the Cauchy–Schwarz inequality $\boldsymbol{w}^{*\top}\boldsymbol{w} \leq \|\boldsymbol{w}^*\|\|\boldsymbol{w}\| = \|\boldsymbol{w}\|$:

$$M\gamma \leq \boldsymbol{w}^{*\top}\boldsymbol{w} \leq \|\boldsymbol{w}\| \leq \sqrt{M}\, R \quad\Longrightarrow\quad M\gamma \leq \sqrt{M}\, R \quad\Longrightarrow\quad M \leq \left(\frac{R}{\gamma}\right)^2.$$

The count is bounded by the data, with no dependence on the dimension: a wide margin and a small radius mean fast convergence. The bound says nothing when the data is not separable, and indeed there the perceptron never stops.

Problem 2: A logistic-regression gradient step

A logistic model $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sigma(\boldsymbol{\theta}^{\top}\boldsymbol{x})$ has parameters $\boldsymbol{\theta} = (0.5, -0.25)$. The data is $\boldsymbol{x}_1 = (1, 2)$ with $t_1 = 1$ and $\boldsymbol{x}_2 = (0, -1)$ with $t_2 = 0$. Compute the log-likelihood gradient $\nabla\ell = \sum_n (t_n - h_n)\boldsymbol{x}_n$ and take one gradient-ascent step with rate $\eta = 1$.

Solution. Predictions first: $\boldsymbol{\theta}^{\top}\boldsymbol{x}_1 = 0.5(1) - 0.25(2) = 0$, so $h_1 = \sigma(0) = 0.5$; $\boldsymbol{\theta}^{\top}\boldsymbol{x}_2 = 0.5(0) - 0.25(-1) = 0.25$, so $h_2 = \sigma(0.25) = 0.562$. The gradient is the error-weighted sum of inputs,

$$\nabla\ell = (t_1 - h_1)\boldsymbol{x}_1 + (t_2 - h_2)\boldsymbol{x}_2 = (0.5)(1, 2) + (-0.562)(0, -1) = (0.5, 1.0) + (0, 0.562) = (0.5, 1.562).$$

One ascent step $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \eta\,\nabla\ell = (0.5, -0.25) + (0.5, 1.562) = (1.0, 1.312)$. The update has the "error times input" form shared by every model in the book; it raises $\boldsymbol{\theta}^{\top}\boldsymbol{x}_1$ (pushing h_1 toward its target 1) and the recomputed h_2 would fall toward 0.

Problem 3: k-nearest-neighbors

Training points are class A at $(1,1)$ and $(2,1)$, and class B at $(5,5)$ and $(6,5)$. Classify the query $\boldsymbol{x} = (3,2)$ with $k = 1$ and $k = 3$ nearest neighbors under Euclidean distance.

Solution. Compute squared distances from $(3,2)$: to $(2,1)$ it is $(3-2)^2 + (2-1)^2 = 2$; to $(1,1)$ it is $4 + 1 = 5$; to $(5,5)$ it is $4 + 9 = 13$; to $(6,5)$ it is $9 + 9 = 18$. The ordering is $(2,1)_A < (1,1)_A < (5,5)_B < (6,5)_B$. For $k = 1$ the nearest point is $(2,1)$, class A. For $k = 3$ the three nearest are $(2,1)_A$, $(1,1)_A$, $(5,5)_B$, a vote of two A to one B, so class A. A practical note: k-NN compares raw distances, so features must be put on a common scale first (standardize each feature using the *training* mean and standard deviation, never the test set's), and k is usually chosen near $\sqrt{N}$ by cross-validation.

Problem 4: Naive Bayes with Laplace smoothing

From 4 spam and 6 ham emails, the word "free" appears in 3 of the spam and 1 of the ham, and the word "meeting" in 0 of the spam and 4 of the ham. Using a Bernoulli naive Bayes model with add-one (Laplace) smoothing, classify a new email that contains "free" but not "meeting".

Solution. The class priors are $P(\text{spam}) = 4/10 = 0.4$ and $P(\text{ham}) = 0.6$. Laplace-smoothed word probabilities add 1 to each count and 2 to each class total (one for each binary outcome):

$$\phi_{\text{free}|\text{spam}} = \tfrac{3+1}{4+2} = 0.667, \quad \phi_{\text{free}|\text{ham}} = \tfrac{1+1}{6+2} = 0.25, \quad \phi_{\text{meet}|\text{spam}} = \tfrac{0+1}{4+2} = 0.167, \quad \phi_{\text{meet}|\text{ham}} = \tfrac{4+1}{6+2} = 0.625.$$

For the email (free present, meeting absent) the unnormalized posteriors multiply the prior, the present-word probability, and one minus the absent-word probability:

$$P(\text{spam} \mid x) \propto 0.4\,(0.667)(1 - 0.167) = 0.222, \qquad P(\text{ham} \mid x) \propto 0.6\,(0.25)(1 - 0.625) = 0.056.$$

Normalizing, $P(\text{spam} \mid x) = 0.222/(0.222+0.056) = 0.80$, so the email is spam with about 80% probability. Smoothing earned its keep on "meeting": without it $\phi_{\text{meet}|\text{spam}} = 0/4 = 0$ would force the spam posterior of *any* email containing "meeting" to exactly zero, an overconfident verdict from a single unseen word.

Problem 5: Reading a confusion matrix

A binary classifier on 100 test examples produces 40 true positives, 10 false positives, 20 false negatives, and 30 true negatives. Compute accuracy, precision, recall, and the F_1 score.

Solution. Accuracy is the fraction correct, $(\text{TP} + \text{TN})/N = (40 + 30)/100 = 0.70$. Precision is the fraction of positive *predictions* that are right, $\text{TP}/(\text{TP} + \text{FP}) = 40/50 = 0.80$. Recall is the fraction of actual positives *caught*, $\text{TP}/(\text{TP} + \text{FN}) = 40/60 = 0.667$. The F_1 score is their harmonic mean,

$$F_1 = \frac{2\,(\text{precision})(\text{recall})}{\text{precision} + \text{recall}} = \frac{2(0.80)(0.667)}{0.80 + 0.667} = \frac{1.067}{1.467} = 0.727.$$

Precision and recall trade off as the decision threshold moves, and F_1 summarizes the balance; which matters more is set by the application (a spam filter prizes precision, a disease screen prizes recall), the decision-theoretic point of Chapter 1.

In the Problem Sets

The accompanying notebook implements the perceptron and logistic regression from scratch and runs them on a two-class data set, implements k-NN and Gaussian naive Bayes with log-probabilities to avoid underflow, and prints the confusion matrix and the four metrics of Problem 5 on a held-out test split.

CHAPTER 23

Problem Set 3: Deep Learning

The Week 5 X-hour pushed past the core into the methods that fill modern deep learning: multi-output regression in matrix form, the sparsity penalty that regularizes an autoencoder, and the forward process of a diffusion model. This set works each. It pairs with Chapters 10 to 13.

Problem 1: Multi-output least squares

A network's final linear layer maps a feature matrix $X \in \mathbb{R}^{N\times d}$ to a matrix of targets $Y \in \mathbb{R}^{N\times k}$ (k outputs at once) through weights $W \in \mathbb{R}^{d\times k}$, minimizing the squared Frobenius error $J(W) = \|XW - Y\|_F^2$. Find the W that minimizes J.

Solution. Write $J(W) = \operatorname{tr}\big[(XW-Y)^\top (XW-Y)\big]$ and differentiate with the matrix-calculus rule $\partial \operatorname{tr}(A^\top A)/\partial A = 2A$ together with the chain rule through $A = XW - Y$ (whose derivative in W contributes a left factor $X^\top$):

$$\frac{\partial J}{\partial W} = 2X^\top (XW - Y).$$

Setting this to zero gives the multi-output normal equations $X^\top XW = X^\top Y$, so

$$W^* = (X^\top X)^{-1} X^\top Y.$$

This is the single-output normal equation of Chapter 5 with a target *matrix* in place of a vector: each output column $\boldsymbol{y}_j$ is fit by its own least-squares weights $\boldsymbol{w}_j = (X^\top X)^{-1}X^\top \boldsymbol{y}_j$, and because the inverse $(X^\top X)^{-1}X^\top$ is shared, all k outputs are solved in one matrix multiply. The same expression is the gradient backpropagation delivers to a network's output layer.

Problem 2: An autoencoder's sparsity penalty

A sparse autoencoder pushes each hidden unit's average activation $\hat\rho_j$ toward a small target sparsity ρ by adding the penalty $\mathrm{KL}(\rho \,\|\, \hat\rho_j) = \rho \log \frac{\rho}{\hat\rho_j} + (1-\rho)\log\frac{1-\rho}{1-\hat\rho_j}$ to the reconstruction loss. For a target $\rho = 0.05$ and an observed average activation $\hat\rho_j = 0.10$, evaluate the penalty, and explain what it does.

Solution. Substitute the two values:

$$\mathrm{KL}(0.05 \,\|\, 0.10) = 0.05 \log \frac{0.05}{0.10} + 0.95 \log \frac{0.95}{0.90} = 0.05\log(0.5) + 0.95\log(1.056).$$

With $\log(0.5) = -0.693$ and $\log(1.056) = 0.054$, this is $0.05(-0.693) + 0.95(0.054) = -0.0347 + 0.0514 = 0.0167$. The penalty is a Bernoulli KL divergence between the desired firing rate ρ and the actual rate $\hat\rho_j$: it is zero only when $\hat\rho_j = \rho$ and grows as the unit fires too often (here 0.10 against a target 0.05), so adding it to the loss drives each hidden unit to be active on only a small fraction of inputs. The result is a sparse, more interpretable code, the autoencoder analogue of the ℓ_1 penalty.

Problem 3: The forward process of a diffusion model

A diffusion model corrupts data by adding Gaussian noise over T steps, $x_t = \sqrt{1-\beta_t}\, x_{t-1} + \sqrt{\beta_t}\, \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, I)$. Writing $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_s$, the marginal is $q(x_t \mid x_0) = \mathcal{N}(\sqrt{\bar{\alpha}_t}\, x_0,\ (1-\bar{\alpha}_t)I)$. For a constant schedule $\beta_t = 0.1$, find $\bar{\alpha}_{10}$ and write x_{10} in terms of x_0 and a single noise draw. What does the training objective minimize?

Solution. With $\beta_t = 0.1$ constant, $\alpha_t = 0.9$ and $\bar{\alpha}_{10} = 0.9^{10} = 0.349$. The marginal lets us jump straight from x_0 to step 10 with one Gaussian draw $\epsilon \sim \mathcal{N}(0, I)$:

$$x_{10} = \sqrt{\bar{\alpha}_{10}}\, x_0 + \sqrt{1-\bar{\alpha}_{10}}\, \epsilon = \sqrt{0.349}\, x_0 + \sqrt{0.651}\, \epsilon = 0.591\, x_0 + 0.807\, \epsilon.$$

By step 10 the signal has shrunk to 59% and noise dominates; as $t \to T$, $\bar{\alpha}_t \to 0$ and x_T is nearly pure noise. Training fits a network $\epsilon_\theta(x_t, t)$ to *predict the noise that was added*, minimizing $\mathcal{L} = \mathbb{E}_{x_0,t,\epsilon}\|\epsilon - \epsilon_\theta(x_t, t)\|^2$. Generation then runs the process backward: start from noise and repeatedly subtract the network's predicted noise to recover a sample. The forward process needs no learning at all; all the learning is in the reverse denoiser.

In the Problem Sets

The accompanying notebook fits the multi-output least-squares layer of Problem 1 on a toy data set, trains a small sparse autoencoder on image patches and plots the learned filters, and runs the diffusion forward process of Problem 3 on an image, showing it dissolve into noise as t grows and $\bar{\alpha}_t$ shrinks.

CHAPTER 24

Problem Set 4: Kernels and Support Vector Machines

This set works the kernel machinery of Part V: the closed-form max-margin solution for two points, the degree-two kernel as a disguised dot product, and the dual quadratic program in the form a solver expects. It collects the Week 8 X-hour problems and pairs with Chapters 14 and 15.

Problem 1: The two-point maximum-margin classifier

Find the hard-margin SVM separating the single positive point $\boldsymbol{x}_1 = (1, 1)$ from the single negative point $\boldsymbol{x}_2 = (-1, -1)$: give the weight vector $\boldsymbol{w}$, the bias b, the dual variables α_1, α_2, and verify the identity $\sum_i \alpha_i = 1/\gamma^2$, where γ is the geometric margin.

Solution. By symmetry the boundary passes through the origin, so $b = 0$ and $\boldsymbol{w}$ points along $\boldsymbol{x}_1 - \boldsymbol{x}_2 = (2, 2)$, say $\boldsymbol{w} = c(1, 1)$. The margin constraint is tight at each support vector, $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) = 1$: here $\boldsymbol{w}^\top \boldsymbol{x}_1 = 2c = 1$, so $c = 0.5$ and $\boldsymbol{w} = (0.5, 0.5)$, with $b = 0$. Check the negative point: $y_2(\boldsymbol{w}^\top \boldsymbol{x}_2) = (-1)(-1) = 1$, tight as well. The geometric margin is $\gamma = 1/\|\boldsymbol{w}\| = 1/\sqrt{0.5} = 1.414$.

For the duals, $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i = \alpha_1(1, 1) + \alpha_2(1, 1) = (\alpha_1 + \alpha_2)(1, 1)$, so $\alpha_1 + \alpha_2 = 0.5$; the constraint $\sum_i \alpha_i y_i = 0$ gives $\alpha_1 - \alpha_2 = 0$, hence $\alpha_1 = \alpha_2 = 0.25$. Then $\sum_i \alpha_i = 0.5$ and $1/\gamma^2 = 1/2 = 0.5$, so $\sum_i \alpha_i = 1/\gamma^2$, as claimed: the total dual mass is the inverse squared margin.

Problem 2: A kernel is a disguised dot product

Show that the degree-two polynomial kernel $k(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^\top \boldsymbol{z})^2$ on $\mathbb{R}^2$ equals $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z})$ for the feature map $\boldsymbol{\phi}(\boldsymbol{x}) = (x_1^2,\ \sqrt{2}\, x_1 x_2,\ x_2^2)$, and verify on $\boldsymbol{x} = (1, 2)$, $\boldsymbol{z} = (3, 1)$.

Solution. Expand the kernel:

$$(\boldsymbol{x}^\top \boldsymbol{z})^2 = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2.$$

Now the feature dot product:

$$\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z}) = x_1^2 z_1^2 + (\sqrt{2}\, x_1 x_2)(\sqrt{2}\, z_1 z_2) + x_2^2 z_2^2 = x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2,$$

identical, with the $\sqrt{2}$ chosen exactly so the cross term's coefficient matches. The kernel computes a dot product in the three-dimensional feature space at the cost of a two-dimensional dot product and a square, never forming $\boldsymbol{\phi}$. Numerically, $\boldsymbol{x}^\top \boldsymbol{z} = 1(3) + 2(1) = 5$, so $k = 25$; and $\boldsymbol{\phi}(\boldsymbol{x}) = (1, 2\sqrt{2}, 4)$, $\boldsymbol{\phi}(\boldsymbol{z}) = (9, 3\sqrt{2}, 1)$, so $\boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{z}) = 9 + 12 + 4 = 25$. They agree: this is the kernel trick.

Problem 3: The dual quadratic program for a solver

Write the hard-margin SVM dual for the two points of Problem 1 explicitly, solve it, and put it in the matrix form $\min_{\boldsymbol{\alpha}} \ \frac{1}{2}\boldsymbol{\alpha}^\top P\boldsymbol{\alpha} - \mathbf{1}^\top\boldsymbol{\alpha}$ *subject to* $\boldsymbol{\alpha} \geq 0$ *and* $\boldsymbol{y}^\top\boldsymbol{\alpha} = 0$ *that a quadratic-program solver such as* `cvxopt` *expects.*

Solution. The dual objective is $W(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j\, \boldsymbol{x}_i^\top\boldsymbol{x}_j$. The inner products are $\boldsymbol{x}_1^\top\boldsymbol{x}_1 = 2$, $\boldsymbol{x}_2^\top\boldsymbol{x}_2 = 2$, $\boldsymbol{x}_1^\top\boldsymbol{x}_2 = -2$, and $y_1y_2 = -1$, so

$$W = \alpha_1 + \alpha_2 - \tfrac{1}{2}\big[2\alpha_1^2 + 2\alpha_2^2 + 2\alpha_1\alpha_2(-1)(-2)\big] = \alpha_1 + \alpha_2 - (\alpha_1^2 + \alpha_2^2 + 2\alpha_1\alpha_2).$$

With $\alpha_1 = \alpha_2 = \alpha$ (from $\boldsymbol{y}^\top\boldsymbol{\alpha} = 0$), $W = 2\alpha - 4\alpha^2$, maximized at $dW/d\alpha = 2 - 8\alpha = 0$, so $\alpha = 0.25$, recovering $\alpha_1 = \alpha_2 = 0.25$ and $\boldsymbol{w} = (0.5, 0.5)$ from before. In solver form, the matrix is $P_{ij} = y_i y_j\, \boldsymbol{x}_i^\top\boldsymbol{x}_j$,

$$P = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix}, \quad \boldsymbol{q} = -\mathbf{1} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad G = -I,\ \boldsymbol{h} = \mathbf{0}\ (\text{for } \boldsymbol{\alpha} \geq 0), \quad A = \boldsymbol{y}^\top = \begin{bmatrix} 1 & -1 \end{bmatrix},\ b = 0.$$

The solver minimizes $\frac{1}{2}\boldsymbol{\alpha}^\top P\boldsymbol{\alpha} + \boldsymbol{q}^\top\boldsymbol{\alpha}$ subject to $G\boldsymbol{\alpha} \leq \boldsymbol{h}$ and $A\boldsymbol{\alpha} = b$, which is the negated dual; here P is rank-deficient (the two points are antipodal), and the equality constraint $\alpha_1 = \alpha_2$ pins down the solution $\boldsymbol{\alpha} = (0.25, 0.25)$.

In the Problem Sets

The accompanying notebook builds the Gram matrix $K_{ij} = y_i y_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ for a chosen kernel, hands $(P, \boldsymbol{q}, G, \boldsymbol{h}, A, b)$ to `cvxopt`'s quadratic-program solver, reads off the support vectors as the points with $\alpha_i > 0$, recovers $\boldsymbol{w}$ and b, and plots the margin and decision boundary for both a linear and an RBF kernel on a small data set.

CHAPTER 25

Problem Set 5: Graphical Models

This set works the graphical-model reasoning of Part VI: conditional independence from a chain, the explaining-away that makes a collider behave unlike a chain, and a Markov random field denoised one pixel at a time. It collects the Week 9 X-hour problems and pairs with Chapters 16 to 18.

Problem 1: Conditional independence in a chain

Three binary variables form the chain $X_1 \to X_2 \to X_3$, with joint $p(x_1, x_2, x_3) = p(x_1)\,p(x_2 \mid x_1)\,p(x_3 \mid x_2)$. Show that X_1 and X_3 are independent given X_2.

Solution. Conditional independence means $p(x_3 \mid x_2, x_1) = p(x_3 \mid x_2)$. Form the conditional from the joint:

$$p(x_3 \mid x_2, x_1) = \frac{p(x_1, x_2, x_3)}{p(x_1, x_2)} = \frac{p(x_1)\,p(x_2 \mid x_1)\,p(x_3 \mid x_2)}{p(x_1)\,p(x_2 \mid x_1)} = p(x_3 \mid x_2),$$

where $p(x_1, x_2) = p(x_1)p(x_2 \mid x_1)$ because X_3 has been summed out of the chain. The result is free of x_1, so $X_3 \perp X_1 \mid X_2$: once the middle variable is known, the past tells you nothing more about the future. Information in a chain flows only through the intermediate node, which is the Markov property that makes chains, and the message-passing of Chapter 17, tractable.

Problem 2: Explaining away at a collider

Two independent causes $X, Y \sim$ Bernoulli(0.5) feed a common effect through an OR gate, $Z = X \vee Y$ (the collider $X \to Z \leftarrow Y$). Compute $P(X = 1 \mid Z = 1)$ and $P(X = 1 \mid Z = 1, Y = 1)$, and explain the difference.

Solution. Since $X = 1$ forces $Z = 1$, and $P(Z = 1) = 1 - P(X{=}0, Y{=}0) = 1 - 0.25 = 0.75$,

$$P(X = 1 \mid Z = 1) = \frac{P(X = 1,\, Z = 1)}{P(Z = 1)} = \frac{P(X = 1)}{0.75} = \frac{0.5}{0.75} = \frac{2}{3}.$$

Observing the effect raised our belief in the cause from 0.5 to 2/3. Now also learn that the *other* cause fired, $Y = 1$. Then $Z = 1$ is fully accounted for, and because X and Y are independent a priori,

$$P(X = 1 \mid Z = 1, Y = 1) = P(X = 1 \mid Y = 1) = P(X = 1) = \frac{1}{2}.$$

Belief in X drops back from 2/3 to 1/2: the second cause has *explained away* the effect. This is the signature of a collider, the opposite of a chain: the two parents are marginally independent but become dependent once their shared child is observed, so conditioning here *opens* a path rather than blocking it.

Problem 3: Image denoising by iterated conditional modes

A binary image $\boldsymbol{x} \in \{-1, +1\}$ is recovered from a noisy observation $\boldsymbol{y}$ by minimizing the Markov-random-field energy $E(\boldsymbol{x}; \boldsymbol{y}) = -h \sum_i x_i - \beta \sum_{(i,j)} x_i x_j - \eta \sum_i x_i y_i$, where the sum over (i, j) is over neighboring pixels. Iterated conditional modes (ICM) visits each pixel and flips it if doing so lowers the energy. Show that flipping pixel i changes the energy by $\Delta E = 2x_i \left(h + \eta y_i + \beta \sum_{j \in N(i)} x_j\right)$, and decide the flip for a pixel with $x_i = -1$, $y_i = +1$, four neighbors summing to $\sum_j x_j = +2$, and parameters $h = 0$, $\eta = 1$, $\beta = 1$.

Solution. Only the terms containing x_i change. Collect them: E depends on x_i through $-x_i(h + \eta y_i + \beta \sum_{j \in N(i)} x_j) = -x_i F_i$, writing $F_i = h + \eta y_i + \beta \sum_{j \in N(i)} x_j$ for the local field. Flipping $x_i \to -x_i$ changes this term from $-x_i F_i$ to $+x_i F_i$, so

$$\Delta E = (+x_i F_i) - (-x_i F_i) = 2x_i F_i = 2x_i \Big(h + \eta y_i + \beta \sum_{j \in N(i)} x_j\Big),$$

and ICM flips when $\Delta E < 0$. For the given pixel, $F_i = 0 + (1)(+1) + (1)(+2) = 3$, so $\Delta E = 2(-1)(3) = -6 < 0$: flip it to $+1$. The flip makes sense, the pixel was -1 while both its data term ($y_i = +1$) and its neighbors (net $+2$) pulled toward $+1$. Sweeping ICM over all pixels until none flips settles into a low-energy image that agrees with the data (η term) while staying locally smooth (β term), typically cleaning well over ninety percent of the noise.

In the Problem Sets

The accompanying notebook builds the OR-gate joint of Problem 2 and checks both probabilities by enumeration, then implements ICM denoising of Problem 3 on a binary image corrupted with random bit flips, sweeping pixels and applying the ΔE rule until convergence, and reports the fraction of noise removed.

CHAPTER 26

The Programming Assignment: Beating Rock-Paper-Scissors

The course's programming lab is a small, complete machine-learning project disguised as a game. A truly random game of rock-paper-scissors is a statistical tie, each outcome a third of the time. People are not random, though, and neither are the bots you play online: they exploit patterns. The assignment is to turn the tables and *learn the opponent's pattern* from a recorded history of play, then choose the move that beats their most likely next throw.

The task

Collect at least one hundred rounds against an online opponent, recording both players' moves and the outcome to a text file. Then build a model that reads the game history and predicts your best next move, and play another hundred rounds using it, recording whether your score improves. The deliverable is the data, the code, and a short "design on paper" write-up. The last requirement is the real lesson Professor Chin presses: never write code without first designing the solution on paper, because code should be a translation of a finished design rather than a place to discover one. A clear paper design is what you debug against and what you communicate to a teammate or a grader.

A model that exploits a bias

The simplest learner counts the opponent's moves and bets they keep their habits. Suppose over a hundred recorded rounds the opponent threw

$$\text{rock } 45, \qquad \text{paper } 30, \qquad \text{scissors } 25,$$

so the estimated distribution is $\hat{p} = (0.45, 0.30, 0.25)$, a clear lean toward rock. The maximum-likelihood next move is rock, and the move that beats rock is *paper*, so the model plays paper every round. If the opponent's distribution holds, paper beats their rock 45% of the time, ties their paper 30%, and loses to their scissors 25%, for a win-minus-loss edge of $45 - 25 = +20$ points per hundred rounds, against the break-even of a random strategy. The model is exactly the maximum-likelihood estimate of a categorical distribution from Chapter 2, used to make a decision in the sense of Chapter 1: pick the action with the best expected outcome under the estimated probabilities.

Going conditional

A stronger model conditions on context, predicting the opponent's next move from their *previous* one, a first-order Markov model with a 3×3 table of transition counts. If after throwing rock the opponent has historically followed with paper most often, then whenever they have just played rock the model plays the counter to paper, namely scissors. This is the categorical naive-Bayes/frequency estimate of Chapter 8 applied per context, and it is the same next-token-from-history idea that the sequence models of Chapter 13 scale up to language. Two cautions from the rest of the book apply: estimate the transition probabilities with a little smoothing (Chapter 8) so an unseen context does not produce a degenerate prediction, and beware that a fixed model invites a counter-exploit, since the opponent is adapting to you as you adapt to it, a two-player dynamic that is the subject of a different Chin course on game theory.

In the Problem Sets

The notebook `assignment-1.ipynb` reads the recorded `training.txt`, builds the move-frequency and transition-count models above in NumPy, and outputs a recommended move given the running history. The grade rests less on the win rate than on the clarity of the paper design and the honest before-and-after comparison of scores.

CHAPTER 27

The Team Project: An End-to-End Study

The term project is where the course's methods meet a real data set. Teams pick a problem, from the UC Irvine or Kaggle repositories or elsewhere, survey what has been done, choose a machine-learning task, build a model, and report results in the format of a research paper. It is the capstone, and it runs the full arc of a study rather than a single technique.

The five sprints

The project is staged across the term, each sprint a piece of the scientific method made into a deliverable.

1. **Planning** (Week 2): a project-management document fixing the team's question, scope, and roles. Choosing a tractable question is half the battle.
2. **Literature review** (Weeks 3–4): an annotated bibliography in research-paper citation format, establishing what the state of the art on the chosen data set already is, so the project adds to it rather than repeating it.
3. **Data cleaning** (Weeks 5–6): the unglamorous majority of real machine learning, turning raw, messy records into the matrix X and targets the methods of this book assume. Handling missing values, encoding categories, and standardizing features (with statistics computed on the training split only) all live here.
4. **Independent model development** (Week 7): each member builds a model independently, so the team can compare approaches before committing, the spirit of the cross-validation and model-comparison ideas of Chapter 4.
5. **Final model** (Weeks 8–10, the bulk of the grade): a written report in research-paper format, the algorithm's code, and a recorded presentation.

Choosing methods from this book

The project is the natural place to choose among everything the course covered, guided by the task and the data. A labeled classification or regression target points to the supervised methods of Parts II–IV; the choice between a linear model, a kernel machine, and a neural network is the bias-variance and data-efficiency trade of Chapters 4, 10 and 15, with small or structured data favoring the simpler, more data-efficient models and large data favoring deep networks. An unlabeled data set points to the clustering and dimensionality reduction of Part VII. Throughout, the report should quantify uncertainty and compare models honestly on held-out data, the decision-theoretic and learning-theory discipline of Chapters 1 and 4, rather than reporting a single number on the training set. A good project is not the one with the fanciest model; it is the one whose question is clear, whose data is handled carefully, and whose claims are supported by the evaluation.

In the Problem Sets

The deliverables are a research-format report, the algorithm code, and a short slide deck and recording. The strongest projects treat the five sprints as one continuous argument: the planning question, the literature gap it fills, the cleaned data, the model comparison, and the final evaluation all telling a single story.

PART IX

Practice Examinations

CHAPTER 28

Practice Midterm Examination

Two hours. Closed book; one double-sided sheet of notes and a basic calculator are permitted. Five problems, one hundred points. The midterm covers Parts I–III: probability and estimation, regression, and classification. Show all work; method earns partial credit even when the arithmetic slips. Each problem is worked in full below; treat the boxed statement as the exam and the rest as the solution key.

Problem 1: Bayes' rule and the base rate (20 points)

A disease affects 2% of a population. A screening test is 90% sensitive (it is positive on 90% of those who have the disease) and 95% specific (negative on 95% of those who do not). (a) A person tests positive once; what is the probability they have the disease? (b) The test is repeated, independently given the true state, and is positive again; now what is the probability? (c) Explain in one or two sentences why the first answer is so far below the test's accuracy.

Solution.

(a) Write D for disease and $+$ for a positive test. The data are $P(D) = 0.02$, $P(+ \mid D) = 0.90$, and $P(+ \mid \neg D) = 1 - 0.95 = 0.05$. By total probability,

$$P(+) = P(+ \mid D)P(D) + P(+ \mid \neg D)P(\neg D) = (0.90)(0.02) + (0.05)(0.98) = 0.018 + 0.049 = 0.067,$$

and Bayes' rule gives $P(D \mid +) = (0.018)/(0.067) = \boxed{0.269}$.

(b) With the two tests conditionally independent given the true state, $P(+,+ \mid D) = 0.90^2 = 0.81$ and $P(+,+ \mid \neg D) = 0.05^2 = 0.0025$. Then

$$P(D \mid +,+) = \frac{(0.02)(0.81)}{(0.02)(0.81) + (0.98)(0.0025)} = \frac{0.0162}{0.0162 + 0.00245} = \frac{0.0162}{0.01865} = \boxed{0.869}.$$

A second independent positive moves the belief from 0.269 to 0.869.

(c) The disease is rare, so the 5% false-positive rate acting on the 98% healthy majority produces about 0.049 false positives against only 0.018 true positives: most positives are false alarms, and a single test cannot overcome a low base rate.

Problem 2: Estimation and the exponential family (20 points)

Let $x \in \{0, 1\}$ be Bernoulli with mean μ. (a) Write the Bernoulli in exponential-family form $p(x) = h(x)\exp(\eta\, T(x) - A(\eta))$, and identify the natural parameter η, sufficient statistic $T(x)$, and log-partition $A(\eta)$. (b) Derive the maximum likelihood estimate of μ from N independent samples. (c) With a Beta$(2, 2)$ *prior and a sample of 7 heads in 10 flips, give the posterior distribution and its mean.*

Solution.

(a) Write $p(x) = \mu^x(1-\mu)^{1-x} = \exp\big(x \ln \mu + (1-x)\ln(1-\mu)\big) = \exp\big(x \ln \frac{\mu}{1-\mu} + \ln(1-\mu)\big)$. So

$$T(x) = x, \qquad \eta = \ln \frac{\mu}{1-\mu}, \qquad A(\eta) = -\ln(1-\mu) = \ln(1+e^{\eta}), \qquad h(x) = 1.$$

Inverting the link gives $\mu = \sigma(\eta)$, the logistic function.

(b) With $k = \sum_n x_n$ successes, the log-likelihood is $\ell(\mu) = k \ln \mu + (N-k)\ln(1-\mu)$. Setting $\ell'(\mu) = k/\mu - (N-k)/(1-\mu) = 0$ gives $k(1-\mu) = (N-k)\mu$, so $\boxed{\hat{\mu} = k/N}$, the sample frequency.

(c) The Beta is conjugate to the Bernoulli: a Beta(a, b) prior with k successes and $N-k$ failures yields a Beta$(a+k,\ b+N-k)$ posterior. Here $a = b = 2$, $k = 7$, $N-k = 3$, so the posterior is $\boxed{\text{Beta}(9, 5)}$ with mean $9/(9+5) = 9/14 = 0.643$. The posterior mean 0.643 sits between the prior mean 0.5 and the MLE 0.7, pulled toward the data as evidence accumulates.

Problem 3: Least squares and ridge regression (20 points)

Fit $y = w_0 + w_1 x$ to the four points $(0,1), (1,2), (2,2), (3,5)$. (a) Solve the normal equations for the ordinary least-squares fit. (b) Refit with a ridge penalty $\lambda = 2$ added to $\boldsymbol{\Phi}^\top \boldsymbol{\Phi}$, and compare. (c) Show that the least-squares fit is the maximum likelihood estimate under Gaussian noise.

Solution.

(a) The design matrix has rows $(1, x_n)$, so $\boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \left[\begin{smallmatrix} 4 & 6 \\ 6 & 14 \end{smallmatrix}\right]$ (count 4, $\sum x = 6$, $\sum x^2 = 14$) and $\boldsymbol{\Phi}^\top \boldsymbol{t} = (\sum t,\ \sum xt) = (10, 21)$. With $\det = 4(14) - 36 = 20$,

$$\boldsymbol{w} = \tfrac{1}{20} \begin{bmatrix} 14 & -6 \\ -6 & 4 \end{bmatrix} \begin{bmatrix} 10 \\ 21 \end{bmatrix} = \tfrac{1}{20}(140 - 126,\ -60 + 84) = \tfrac{1}{20}(14, 24) = \boxed{(0.7, 1.2)},$$

the line $y = 0.7 + 1.2x$.

(b) Ridge solves $(\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \lambda I)\boldsymbol{w} = \boldsymbol{\Phi}^\top \boldsymbol{t}$. With $\lambda = 2$, $\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + 2I = \left[\begin{smallmatrix} 6 & 6 \\ 6 & 16 \end{smallmatrix}\right]$, $\det = 96 - 36 = 60$, so

$$\boldsymbol{w}_{\text{ridge}} = \tfrac{1}{60} \begin{bmatrix} 16 & -6 \\ -6 & 6 \end{bmatrix} \begin{bmatrix} 10 \\ 21 \end{bmatrix} = \tfrac{1}{60}(160 - 126,\ -60 + 126) = \tfrac{1}{60}(34, 66) = \boxed{(0.567, 1.10)}.$$

Both coefficients shrank toward zero ($0.7 \to 0.567$, $1.2 \to 1.10$): ridge trades a little fit for smaller, more stable weights.

(c) Model $t_n = \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n) + \varepsilon_n$ with $\varepsilon_n \sim \mathcal{N}(0, \sigma^2)$. The log-likelihood is $\ell(\boldsymbol{w}) = -\frac{1}{2\sigma^2} \sum_n (t_n - \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n))^2 + \text{const}$. Maximizing ℓ over $\boldsymbol{w}$ is the same as minimizing $\sum_n (t_n - \boldsymbol{w}^\top \boldsymbol{\phi}(\boldsymbol{x}_n))^2$, the least-squares objective, because $\sigma^2 > 0$ only rescales. So least squares is Gaussian-noise maximum likelihood, and ridge is the corresponding MAP estimate under a Gaussian prior on $\boldsymbol{w}$.

Problem 4: Logistic regression (20 points)

A logistic-regression model is $p(\mathcal{C}_1 \mid \boldsymbol{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ with weights $\boldsymbol{w} = (-1, 1, 1)$. (a) Give the decision boundary and classify the points $(1,1)$ and $(0,0)$ with their probabilities. (b) Starting from $w = 0$ for the one-feature model $p = \sigma(wx)$, take one gradient-ascent step (rate $\eta = 1$) on the single example $x = 2$, $t = 1$. (c) Explain why the cross-entropy loss (rather than squared error) is used with a sigmoid output.

Solution.

(a) The boundary is where the argument is zero, $-1 + x_1 + x_2 = 0$, the line $x_1 + x_2 = 1$. At $(1,1)$ the argument is $1 > 0$, so class $\mathcal{C}_1$ with probability $\sigma(1) = 0.731$; at $(0,0)$ it is $-1 < 0$, so class $\mathcal{C}_2$, with $p(\mathcal{C}_1) = \sigma(-1) = 0.269$.

(b) The log-likelihood gradient is $\partial \ell / \partial w = (t - \sigma(wx))x$. At $w = 0$, $\sigma(0) = 0.5$, so the gradient is $(1 - 0.5)(2) = 1$, and the ascent step is $w \leftarrow 0 + (1)(1) = \boxed{1}$. After it, $\sigma((1)(2)) = \sigma(2) = 0.881$, closer to the target $t = 1$.

(c) With a sigmoid output, the squared-error loss $\frac{1}{2}(\sigma(a) - t)^2$ has gradient proportional to $\sigma'(a) = \sigma(a)(1 - \sigma(a))$, which vanishes when the unit is saturated (confidently wrong), so learning stalls. The cross-entropy loss cancels that factor and gives the gradient $\partial E / \partial a = \hat{y} - t$, which is large exactly when the prediction is far from the target. Cross-entropy is also the correct negative log-likelihood for a Bernoulli target.

Problem 5: Learning theory and the bias–variance tradeoff (20 points)

(a) A finite hypothesis class has $k = 100$ members. Using the uniform-convergence bound, how many independent samples guarantee, with probability at least 0.95, that every hypothesis's training error is within $\gamma = 0.1$ of its true error? (b) Explain, in terms of bias and variance, why fitting a high-degree polynomial to a small noisy data set gives near-zero training error but poor test error.

Solution.

(a) The uniform-convergence sample complexity is $m \geq \frac{1}{2\gamma^2}\ln(2k/\delta)$ at confidence $1-\delta$. With $\delta = 0.05$, $\gamma = 0.1$, $k = 100$,

$$m \geq \frac{1}{2(0.1)^2}\ln\frac{2(100)}{0.05} = 50\,\ln(4000) = 50(8.294) \approx \boxed{415}.$$

The dependence on k and δ is only logarithmic, but halving γ quadruples m.

(b) A high-degree polynomial has many free parameters, enough to pass through every training point, so training error is near zero. That flexibility is high *variance*: small changes in the noisy data swing the fitted curve wildly, because the spare parameters fit the noise rather than the signal. The bias is low (the class can represent the truth), but expected test error is bias^2 plus variance plus irreducible noise, and the variance term dominates, so the curve oscillates between training points and predicts poorly there. A lower-degree model, or regularization, trades a little bias for a large drop in variance and generalizes better.

CHAPTER 29

A Full Practice Examination, Worked in Detail

Three hours, comprehensive, covering the whole book. One double-sided sheet of notes and a basic calculator permitted. Three parts: True or False with a one-line justification (Part I, 20 points), multi-part long questions worked in full (Part II, 60 points), and short proofs (Part III, 20 points). The statement of each question is in italic; the rest is the worked solution. Sit it closed-book first, then check against the solutions.

29.1 Part I: True or False

Decide whether each statement is true or false and give a one-sentence reason.

1. *A single-hidden-layer network with a nonpolynomial activation can approximate any continuous function on a compact set to any accuracy.*

   True. This is the universal approximation theorem; the catch it hides is that the required width can be enormous, which is why depth is used in practice.

2. *The perceptron learning algorithm converges on any training set.*

   False. It converges only when the data is linearly separable; on non-separable data (such as XOR) it cycles forever.

3. *Maximizing the SVM margin is equivalent to minimizing* $\frac{1}{2}\|\boldsymbol{w}\|^2$ *subject to* $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$.

   True. With the canonical scaling the margin is $2/\|\boldsymbol{w}\|$, so widening it is the same as shrinking $\|\boldsymbol{w}\|$ under the unit-margin constraints.

4. *The EM algorithm for a Gaussian mixture is guaranteed to converge to the global maximum of the likelihood.*

   False. EM never decreases the likelihood and converges to a stationary point, but that point is generally a local maximum; the result depends on the initialization, which is why several restarts are used.

5. *The Gaussian (RBF) kernel corresponds to an infinite-dimensional feature map.*

   True. Expanding the exponential produces all degrees of monomials, so the implicit feature space is infinite-dimensional, yet the kernel is a one-line formula.

6. *In the collider* $X \to Z \leftarrow Y$ *with* X, Y *independent, observing* Z *leaves* X *and* Y *independent.*

   False. Conditioning on a common effect *opens* the path and makes the causes dependent, the explaining-away effect; they are independent only *before* Z is observed.

7. *Compared with ordinary least squares, ridge regression has higher bias and lower variance.*

   True. The penalty shrinks the coefficients, which biases the estimate toward zero but reduces its sensitivity to the noise in any one data set.

8. *The first principal component is the eigenvector of the data covariance with the smallest eigenvalue.*

   False. It is the eigenvector of the *largest* eigenvalue, the direction of greatest variance; the smallest-eigenvalue direction is the one PCA discards first.

29.2 Part II: Long Questions

Long Question 1: A neural network, forward and backward

A two-layer network has a ReLU hidden layer and a sigmoid output, with

$$W^{[1]} = \begin{bmatrix} 0.5 & -0.3 \\ 0.8 & 0.2 \\ -0.4 & 0.6 \end{bmatrix},\ \boldsymbol{b}^{[1]} = \begin{bmatrix} -0.2 \\ -0.5 \\ 0.2 \end{bmatrix},\quad W^{[2]} = \begin{bmatrix} 1 & -2 & 0.5 \end{bmatrix},\ b^{[2]} = 0.3.$$

The training example is $\boldsymbol{x} = (1, 2)^\top$ with target class $t = 1$ and binary cross-entropy loss. (a) Compute the forward pass and the loss. (b) Compute the output error $\delta^{[2]}$ and the output-layer gradients. (c) Backpropagate to get $\boldsymbol{\delta}^{[1]}$ and the hidden-layer weight gradient. (d) Take one gradient step with rate $\eta = 0.1$. (e) Why does the first hidden unit receive no weight update?

Solution.

(a) Forward pass. $\boldsymbol{a}^{[1]} = W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]} = (-0.1, 1.2, 0.8)^\top + (-0.2, -0.5, 0.2)^\top = (-0.3, 0.7, 1.0)^\top$. The ReLU gives $\boldsymbol{z}^{[1]} = (0, 0.7, 1.0)^\top$ (the first unit is off). Then $a^{[2]} = (1)(0) + (-2)(0.7) + (0.5)(1.0) + 0.3 = -0.6$, and $\hat{y} = \sigma(-0.6) = 0.354$. With $t = 1$ the loss is $E = -\ln \hat{y} = -\ln 0.354 = \boxed{1.038}$.

(b) Output gradients. For the matched sigmoid-and-cross-entropy pair, $\delta^{[2]} = \hat{y} - t = 0.354 - 1 = -0.646$. The weight gradient is the outer product with the incoming activation, $\partial E/\partial W^{[2]} = \delta^{[2]}\boldsymbol{z}^{[1]\top} = -0.646\,(0, 0.7, 1.0) = (0, -0.452, -0.646)$, and $\partial E/\partial b^{[2]} = -0.646$.

(c) Backpropagate. Send the error through the transposed weights and gate by the ReLU derivative (1 where the pre-activation was positive):

$$W^{[2]\top}\delta^{[2]} = (1, -2, 0.5)^\top(-0.646) = (-0.646, 1.292, -0.323)^\top, \qquad g'(\boldsymbol{a}^{[1]}) = (0, 1, 1)^\top,$$

so $\boldsymbol{\delta}^{[1]} = (-0.646, 1.292, -0.323)^\top \odot (0, 1, 1)^\top = (0, 1.292, -0.323)^\top$. The hidden weight gradient is $\partial E/\partial W^{[1]} = \boldsymbol{\delta}^{[1]}\boldsymbol{x}^\top = (0, 1.292, -0.323)^\top(1, 2) = \begin{bmatrix} 0 & 0 \\ 1.292 & 2.584 \\ -0.323 & -0.646 \end{bmatrix}$.

(d) Update. With $\eta = 0.1$, $W^{[2]} \leftarrow (1, -1.955, 0.565)$ and the second and third rows of $W^{[1]}$ become $(0.671, -0.058)$ and $(-0.368, 0.665)$; the first row is unchanged, and the biases shift to $b^{[2]} = 0.365$ and $\boldsymbol{b}^{[1]} = (-0.2, -0.629, 0.232)$. Re-running the full forward pass gives $\hat{y} = 0.739$, up from 0.354 and closer to the target 1.

(e) The dead unit. The first hidden unit had pre-activation $a_1^{[1]} = -0.3 < 0$, so its ReLU output and its derivative are both zero. Its error signal $\delta_1^{[1]} = 0$, and its weight-gradient row $\delta_1^{[1]}\boldsymbol{x}^\top = \boldsymbol{0}$, so its incoming weights do not move: a unit that contributed nothing to the prediction gets no blame for the error.

Long Question 2: The support vector machine

Two training points are $\boldsymbol{x}_1 = (1, 0)$ with $y_1 = +1$ and $\boldsymbol{x}_2 = (-1, 0)$ with $y_2 = -1$. (a) State the hard-margin primal. (b) Solve for $\boldsymbol{w}$, b, and the margin. (c) Solve the dual for α_1, α_2. (d) Which points are support vectors, and how is a new point classified? (e) Write the kernelized decision rule and explain why it needs only inner products.

Solution.

(a) Primal. Minimize $\frac{1}{2}\|\boldsymbol{w}\|^2$ subject to $y_n(\boldsymbol{w}^\top\boldsymbol{x}_n + b) \geq 1$ for $n = 1, 2$.

(b) Solve. By symmetry the boundary is the vertical axis, so $\boldsymbol{w} = (w, 0)$. The canonical constraints $y_1(\boldsymbol{w}^\top\boldsymbol{x}_1 + b) = w + b = 1$ and $y_2(\boldsymbol{w}^\top\boldsymbol{x}_2 + b) = -(-w + b) = w - b = 1$ give $w = 1$, $b = 0$. Thus $\boldsymbol{w} = (1, 0)$, the boundary is $x_1 = 0$, and the margin is $2/\|\boldsymbol{w}\| = \boxed{2}$.

(c) Dual. The dual relation $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n = (\alpha_1 + \alpha_2, 0)$ must equal $(1, 0)$, and the constraint $\sum_n \alpha_n y_n = \alpha_1 - \alpha_2 = 0$ gives $\alpha_1 = \alpha_2$. Together $\alpha_1 = \alpha_2 = \boxed{0.5}$.

(d) Support vectors. Both $\alpha_n > 0$, so both points are support vectors (they sit on the margin). A new point is classified by $\hat{y} = \operatorname{sign}(\boldsymbol{w}^\top\boldsymbol{x} + b) = \operatorname{sign}(x_1)$.

(e) Kernelized rule. Substituting $\boldsymbol{w} = \sum_n \alpha_n y_n \boldsymbol{x}_n$ into the decision function gives $\hat{y}(\boldsymbol{x}) = \operatorname{sign}\big(\sum_n \alpha_n y_n\, \boldsymbol{x}_n^\top\boldsymbol{x} + b\big)$, and replacing each inner product $\boldsymbol{x}_n^\top\boldsymbol{x}$ by a kernel $k(\boldsymbol{x}_n, \boldsymbol{x})$ yields $\hat{y}(\boldsymbol{x}) = \operatorname{sign}\big(\sum_n \alpha_n y_n\, k(\boldsymbol{x}_n, \boldsymbol{x}) + b\big)$. Only inner products of data points appear, so the high-dimensional feature map is never formed; a nonlinear kernel then gives a nonlinear boundary at the cost of a linear one.

Long Question 3: Gaussian mixtures and EM

Fit a two-component one-dimensional Gaussian mixture to the points $x = 0, 1, 4, 5$, starting from $\pi = (0.5, 0.5)$, means $\mu_1 = 1$, $\mu_2 = 4$, and unit variances. (a) Write the responsibility formula. (b) Run the E step. (c) Run the M step for π and the means. (d) In what limit does this become k-means?

Solution.

(a) Responsibility. $\gamma(z_{nk}) = \dfrac{\pi_k \mathcal{N}(x_n \mid \mu_k, \sigma_k^2)}{\sum_j \pi_j \mathcal{N}(x_n \mid \mu_j, \sigma_j^2)}$, the posterior probability that point n came from component k.

(b) E step. With equal weights and variances the constants cancel, leaving the ratio of $e^{-(x-\mu_k)^2/2}$. For $x = 0$: $e^{-0.5} = 0.607$ versus $e^{-8} = 0.0003$, so $\gamma_1 = 0.999$. Repeating,

$$\gamma(z_{n1}) = (0.999,\ 0.989,\ 0.011,\ 0.001), \qquad \gamma(z_{n2}) = (0.001,\ 0.011,\ 0.989,\ 0.999).$$

(c) M step. The effective counts are $N_1 = \sum_n \gamma_{n1} = 2.00$ and $N_2 = 2.00$, so $\pi_1 = \pi_2 = 0.5$. The means are responsibility-weighted averages,

$$\mu_1 = \frac{0.999(0) + 0.989(1) + 0.011(4) + 0.001(5)}{2.00} = 0.52, \qquad \mu_2 = \frac{0.001(0) + 0.011(1) + 0.989(4) + 0.999(5)}{2.00} = 4.48.$$

One iteration pulled the means from $(1, 4)$ to $\boxed{(0.52, 4.48)}$, snapping onto the clusters $\{0, 1\}$ and $\{4, 5\}$.

(d) The k-means limit. As the component variances shrink to zero ($\sigma^2 \to 0$) with equal weights, each responsibility becomes hard (0 or 1, the nearest mean winning), the E step becomes the nearest-centroid assignment, and the M step becomes the cluster-mean update: EM degenerates to k-means.

Long Question 4: Principal component analysis

Take the four points $(1, 1), (2, 3), (3, 2), (4, 4)$. (a) Compute the mean and the covariance. (b) Find the eigenvalues and eigenvectors. (c) What fraction of variance does the first component capture? (d) Project the point $(4, 4)$ onto the first component. (e) What is the total squared reconstruction error if the second component is discarded?

Solution.

(a) Mean and covariance. The mean is $(2.5, 2.5)$, so the centered points are $(-1.5, -1.5), (-0.5, 0.5), (0.5, -0.5), (1.5, 1.5)$. Then $\Sigma_{11} = \Sigma_{22} = \frac{1}{4}(1.5^2 + 0.5^2 + 0.5^2 + 1.5^2) = 1.25$ and $\Sigma_{12} = \frac{1}{4}(2.25 - 0.25 - 0.25 + 2.25) = 1.0$, so $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 1.25 & 1.0 \\ 1.0 & 1.25 \end{smallmatrix}\right]$.

(b) Eigen-decomposition. From $(1.25 - \lambda)^2 - 1 = 0$, $\lambda_1 = 2.25$ and $\lambda_2 = 0.25$, with $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1, 1)$ and $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1, -1)$.

(c) Variance explained. $\lambda_1/(\lambda_1 + \lambda_2) = 2.25/2.5 = \boxed{90\%}$.

(d) Projection. The centered point $(4, 4) - (2.5, 2.5) = (1.5, 1.5)$ projects to $\boldsymbol{u}_1^\top (1.5, 1.5) = \frac{1}{\sqrt{2}}(3) = 2.12$.

(e) Reconstruction error. Discarding the second component costs the variance along it, summed over the $N = 4$ points: $\sum_n (\boldsymbol{u}_2^\top \tilde{\boldsymbol{x}}_n)^2 = N\lambda_2 = 4(0.25) = \boxed{1.0}$, where $\tilde{\boldsymbol{x}}_n$ are the centered points. The error equals the discarded eigenvalue times the number of points.

Long Question 5: Graphical models and inference

(a) For the Bayesian network $A \to C \leftarrow B$ with $C \to D$, state which of $A \perp\!\!\!\perp B$ and $A \perp\!\!\!\perp B \mid C$ hold, and why. (b) A hidden Markov chain has prior $\boldsymbol{\pi} = (0.5, 0.5)$, transition $T = \left[\begin{smallmatrix} 0.8 & 0.2 \\ 0.3 & 0.7 \end{smallmatrix}\right]$, and at three times the emission likelihoods $b_1 = (0.9, 0.2)$, $b_2 = (0.6, 0.5)$, $b_3 = (0.1, 0.8)$. Run the forward pass, give the data likelihood, and give the filtered belief $p(x_3 \mid \text{obs})$.

Solution.

(a) d-separation. C is a collider on the path $A \to C \leftarrow B$. Unobserved, it blocks the path, so $A \perp\!\!\!\perp B$ holds marginally. Observing C (or its descendant D) opens the collider, so $A \perp\!\!\!\perp B \mid C$ *fails*: the common effect couples the causes.

(b) Forward pass. The forward messages $\alpha_t(i) = p(\text{obs}_{1:t}, x_t = i)$ start from $\alpha_1 = \boldsymbol{\pi} \odot b_1 = (0.45, 0.10)$ and recurse $\alpha_t(j) = b_t(j) \sum_i \alpha_{t-1}(i) T_{ij}$:

$$\alpha_2 = \big(0.6(0.45(0.8) + 0.10(0.3)),\ 0.5(0.45(0.2) + 0.10(0.7))\big) = (0.234, 0.080),$$

$$\alpha_3 = \big(0.1(0.234(0.8)+0.080(0.3)),\ 0.8(0.234(0.2)+0.080(0.7))\big) = (0.0211, 0.0822).$$

The data likelihood is $p(\text{obs}) = \sum_i \alpha_3(i) = 0.0211 + 0.0822 = \boxed{0.1034}$, and the filtered belief is $p(x_3 \mid \text{obs}) \propto \alpha_3 = (0.0211, 0.0822) \to \boxed{(0.20, 0.80)}$. The chain most likely ends in state 2.

29.3 Part III: Proofs

Proof 1: The sigmoid–cross-entropy gradient

For a sigmoid output $\hat{y} = \sigma(a)$ and the binary cross-entropy loss $E = -[t \ln \hat{y} + (1-t)\ln(1-\hat{y})]$, prove that $\partial E/\partial a = \hat{y} - t$.

Proof. By the chain rule, $\partial E/\partial a = (\partial E/\partial \hat{y})(\partial \hat{y}/\partial a)$. The loss derivative is $\partial E/\partial \hat{y} = -t/\hat{y} + (1-t)/(1-\hat{y})$, and the sigmoid derivative is $\partial \hat{y}/\partial a = \hat{y}(1-\hat{y})$. Multiplying,

$$\frac{\partial E}{\partial a} = \Big(-\frac{t}{\hat{y}} + \frac{1-t}{1-\hat{y}}\Big)\hat{y}(1-\hat{y}) = -t(1-\hat{y}) + (1-t)\hat{y} = -t + t\hat{y} + \hat{y} - t\hat{y} = \hat{y} - t.$$

The sigmoid factor $\hat{y}(1-\hat{y})$ cancels the loss denominators, which is exactly why the pair is used together. ∎

Proof 2: The bias–variance decomposition

For a fixed input, let the target be $t = f + \varepsilon$ with $\mathbb{E}[\varepsilon] = 0$, $\mathrm{Var}(\varepsilon) = \sigma^2$, and let $\hat{y}$ be an estimate trained on a random data set (independent of ε). Prove that $\mathbb{E}[(t-\hat{y})^2] = (f - \mathbb{E}[\hat{y}])^2 + \mathrm{Var}(\hat{y}) + \sigma^2$.

Proof. Write $t - \hat{y} = (f - \hat{y}) + \varepsilon$. Since ε is independent of $\hat{y}$ and has mean zero, the cross term vanishes in expectation, so $\mathbb{E}[(t-\hat{y})^2] = \mathbb{E}[(f-\hat{y})^2] + \mathbb{E}[\varepsilon^2] = \mathbb{E}[(f-\hat{y})^2] + \sigma^2$. Now add and subtract $\mathbb{E}[\hat{y}]$ inside the first term:

$$\mathbb{E}[(f-\hat{y})^2] = \mathbb{E}\big[((f - \mathbb{E}[\hat{y}]) - (\hat{y} - \mathbb{E}[\hat{y}]))^2\big] = (f - \mathbb{E}[\hat{y}])^2 + \mathbb{E}[(\hat{y} - \mathbb{E}[\hat{y}])^2],$$

because the cross term $-2(f - \mathbb{E}[\hat{y}])\mathbb{E}[\hat{y} - \mathbb{E}[\hat{y}]] = 0$. The two pieces are the squared bias and the variance, so $\mathbb{E}[(t-\hat{y})^2] = (f - \mathbb{E}[\hat{y}])^2 + \mathrm{Var}(\hat{y}) + \sigma^2$. ∎

Proof 3: A valid kernel's Gram matrix is positive semidefinite

Show that if $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\phi}(\boldsymbol{x}')$ for some feature map $\boldsymbol{\phi}$, then for any points $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$ the Gram matrix K with $K_{nm} = k(\boldsymbol{x}_n, \boldsymbol{x}_m)$ is symmetric and positive semidefinite.

Proof. Symmetry is immediate: $K_{nm} = \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m) = \boldsymbol{\phi}(\boldsymbol{x}_m)^\top \boldsymbol{\phi}(\boldsymbol{x}_n) = K_{mn}$. For positive semidefiniteness, take any coefficients $\boldsymbol{c} \in \mathbb{R}^N$. Then

$$\boldsymbol{c}^\top K \boldsymbol{c} = \sum_{n,m} c_n c_m\, \boldsymbol{\phi}(\boldsymbol{x}_n)^\top \boldsymbol{\phi}(\boldsymbol{x}_m) = \Big(\sum_n c_n \boldsymbol{\phi}(\boldsymbol{x}_n)\Big)^\top \Big(\sum_m c_m \boldsymbol{\phi}(\boldsymbol{x}_m)\Big) = \Big\|\sum_n c_n \boldsymbol{\phi}(\boldsymbol{x}_n)\Big\|^2 \geq 0,$$

the squared norm of a vector. So $K \succeq 0$. By Mercer's theorem the converse also holds, which is why new kernels are validated by checking positive semidefiniteness rather than by exhibiting a feature map. ∎

PART X

Appendices

APPENDIX A

Linear Algebra Reference

This appendix collects the linear-algebra facts the main text leans on, stated for reference. A fuller development is Strang [37]; the machine-learning slant is in Deisenroth et al. [9].

A.1 Vectors, matrices, and the four operations of note

A vector $\boldsymbol{x} \in \mathbb{R}^n$ is a column of numbers; a matrix $A \in \mathbb{R}^{m \times n}$ is a linear map from $\mathbb{R}^n$ to $\mathbb{R}^m$. The *inner product* $\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\top \boldsymbol{y} = \sum_i x_i y_i$ measures alignment; the *Euclidean norm* $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\top \boldsymbol{x}}$ measures length; and the two are tied by the Cauchy–Schwarz inequality $|\boldsymbol{x}^\top \boldsymbol{y}| \le \|\boldsymbol{x}\| \|\boldsymbol{y}\|$, with equality only when the vectors are parallel. The *outer product* $\boldsymbol{x}\boldsymbol{y}^\top$ is the rank-one matrix with (i, j) entry $x_i y_j$. Matrix multiplication composes maps, $(AB)\boldsymbol{x} = A(B\boldsymbol{x})$, and is not commutative.

The *column space* $\mathrm{Col}(A)$ is the span of the columns (the set of reachable outputs $A\boldsymbol{x}$), the *null space* $\mathrm{Nul}(A)$ is the set of $\boldsymbol{x}$ with $A\boldsymbol{x} = \boldsymbol{0}$, and the *rank* is the dimension of the column space, the number of independent directions the map can reach. The transpose $A^\top$ has $(A^\top)_{ij} = A_{ji}$, and $(AB)^\top = B^\top A^\top$.

A.2 Special matrices

- **Symmetric**: $A = A^\top$. Covariance matrices, Gram matrices, and Hessians are symmetric, and symmetric matrices have real eigenvalues and orthogonal eigenvectors.
- **Orthogonal**: $Q^\top Q = I$, so Q's columns are orthonormal and Q preserves lengths and angles, $\|Q\boldsymbol{x}\| = \|\boldsymbol{x}\|$; it is a rotation or reflection.
- **Positive semidefinite (PSD)**: symmetric with $\boldsymbol{x}^\top A \boldsymbol{x} \ge 0$ for all $\boldsymbol{x}$, written $A \succeq 0$; equivalently all eigenvalues are nonnegative. *Positive definite* ($A \succ 0$) strengthens this to $\boldsymbol{x}^\top A \boldsymbol{x} > 0$ for $\boldsymbol{x} \neq \boldsymbol{0}$, all eigenvalues positive. Covariance matrices (Chapter 2) and the matrices $A^\top A$ are always PSD.

A.3 Eigenvalues and the spectral theorem

A scalar λ and nonzero vector $\boldsymbol{u}$ with $A\boldsymbol{u} = \lambda\boldsymbol{u}$ are an *eigenvalue* and *eigenvector*: directions the map only stretches, by the factor λ. For symmetric matrices the structure is as clean as possible.

Spectral theorem

Every symmetric matrix $A \in \mathbb{R}^{n\times n}$ can be written $A = Q\Lambda Q^\top = \sum_{i=1}^{n} \lambda_i \boldsymbol{u}_i \boldsymbol{u}_i^\top$, where Q is orthogonal (its columns are orthonormal eigenvectors $\boldsymbol{u}_i$) and $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ holds the real eigenvalues. A symmetric matrix is diagonal in its own eigenbasis.

This is the result behind the covariance geometry of Chapter 2 and principal component analysis of Chapter 19: the eigenvectors are the principal axes and the eigenvalues the variances along them. The trace and determinant read off the eigenvalues, $\operatorname{tr}(A) = \sum_i \lambda_i$ and $\det(A) = \prod_i \lambda_i$.

Example A.1 (An eigendecomposition by hand). Diagonalize $A = \left[\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right]$. The characteristic equation is $\det(A - \lambda I) = (2-\lambda)^2 - 1 = 0$, so $2 - \lambda = \pm 1$ and the eigenvalues are $\lambda_1 = 3$, $\lambda_2 = 1$. For $\lambda_1 = 3$, $(A - 3I)\boldsymbol{u} = \left[\begin{smallmatrix} -1 & 1 \\ 1 & -1 \end{smallmatrix}\right]\boldsymbol{u} = \boldsymbol{0}$ forces $u_1 = u_2$, giving the unit eigenvector $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1, 1)$; for $\lambda_2 = 1$, $u_1 = -u_2$, so $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1, -1)$. The spectral decomposition is

$$A = 3\,\boldsymbol{u}_1\boldsymbol{u}_1^\top + 1\,\boldsymbol{u}_2\boldsymbol{u}_2^\top = \tfrac{3}{2}\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} + \tfrac{1}{2}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},$$

recovering A. As a check, $\operatorname{tr}(A) = 4 = 3 + 1$ and $\det(A) = 3 = 3 \cdot 1$. ◊

A.4 The Rayleigh quotient

For a symmetric matrix A, the *Rayleigh quotient* $R(\boldsymbol{x}) = \boldsymbol{x}^\top A\boldsymbol{x} / \boldsymbol{x}^\top\boldsymbol{x}$ is bounded by the extreme eigenvalues, $\lambda_{\min} \le R(\boldsymbol{x}) \le \lambda_{\max}$, and attains $\lambda_{\max}$ at the top eigenvector and $\lambda_{\min}$ at the bottom one. Maximizing $\boldsymbol{x}^\top A\boldsymbol{x}$ subject to $\|\boldsymbol{x}\| = 1$ therefore returns the largest eigenvalue and its eigenvector, the optimization at the heart of PCA (Chapter 19).

A.5 The singular value decomposition

Every matrix, square or not, factors through the singular value decomposition (SVD).

Singular value decomposition

Any $A \in \mathbb{R}^{m\times n}$ can be written $A = U\Sigma V^\top$, where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal and Σ is diagonal with nonnegative *singular values* $\sigma_1 \ge \sigma_2 \ge \cdots \ge 0$. The columns of V are the eigenvectors of $A^\top A$, the columns of U are the eigenvectors of $AA^\top$, and σ_i^2 are their (shared nonzero) eigenvalues.

The SVD is the numerically stable route to PCA (Chapter 19), the basis of low-rank approximation (keeping the largest singular values gives the best rank-k approximation), and the cleanest definition of the pseudoinverse.

Example A.2 (A singular value decomposition by hand). Find the SVD of $A = \left[\begin{smallmatrix} 3 & 0 \\ 4 & 5 \end{smallmatrix}\right]$. Form $A^\top A = \left[\begin{smallmatrix} 25 & 20 \\ 20 & 25 \end{smallmatrix}\right]$, whose eigenvalues solve $(25-\lambda)^2 - 400 = 0$, giving $\lambda = 45, 5$; the singular values are their square roots, $\sigma_1 = \sqrt{45} = 3\sqrt{5}$ and $\sigma_2 = \sqrt{5}$. The eigenvectors of $A^\top A$ are the right singular vectors, $\boldsymbol{v}_1 = \frac{1}{\sqrt{2}}(1, 1)$ and $\boldsymbol{v}_2 = \frac{1}{\sqrt{2}}(1, -1)$. The left singular vectors come from $\boldsymbol{u}_i = A\boldsymbol{v}_i / \sigma_i$:

$$\boldsymbol{u}_1 = \frac{A\boldsymbol{v}_1}{3\sqrt{5}} = \frac{1}{\sqrt{10}}(1, 3), \qquad \boldsymbol{u}_2 = \frac{A\boldsymbol{v}_2}{\sqrt{5}} = \frac{1}{\sqrt{10}}(3, -1),$$

both unit length. As checks, $\sigma_1^2 + \sigma_2^2 = 50 = \|A\|_F^2$ (the sum of squared entries $9 + 0 + 16 + 25$) and $\sigma_1\sigma_2 = 15 = |\det A|$. ◊

The singular values also measure *conditioning*. The condition number $\kappa(A) = \sigma_{\max}/\sigma_{\min}$ is the factor by which A can amplify relative error when a system $A\boldsymbol{x} = \boldsymbol{b}$ is solved; a large κ (a singular value near zero) means the problem is

ill-conditioned, which is exactly the situation the ridge penalty of Chapter 5 and the $K + \lambda I$ of Chapter 14 stabilize. For the matrix above, $\kappa = 3\sqrt{5}/\sqrt{5} = 3$, well conditioned.

A.6 Projections and the pseudoinverse

The *orthogonal projection* of a vector $\boldsymbol{b}$ onto the column space of A (with A full column rank) is $P\boldsymbol{b}$ with projection matrix $P = A(A^\top A)^{-1}A^\top$, satisfying $P = P^\top = P^2$. The residual $\boldsymbol{b} - P\boldsymbol{b}$ is orthogonal to $\mathrm{Col}(A)$. The least-squares solution of $A\boldsymbol{x} \approx \boldsymbol{b}$, the $\boldsymbol{x}$ minimizing $\|A\boldsymbol{x} - \boldsymbol{b}\|^2$, is $\boldsymbol{x} = A^\dagger \boldsymbol{b}$ with the *Moore–Penrose pseudoinverse*

$$A^\dagger = (A^\top A)^{-1}A^\top \quad \text{(full column rank)}, \qquad A^\dagger = V\Sigma^\dagger U^\top \quad \text{(via SVD, in general)}, \tag{A.1}$$

where $\Sigma^\dagger$ inverts the nonzero singular values. This is the engine of the normal equations in Chapter 5 and the dual representation in Chapter 14.

Factorization	**Form**	**Used for**
Spectral (symmetric)	$A = Q\Lambda Q^\top$	covariance geometry, PCA
SVD (any matrix)	$A = U\Sigma V^\top$	PCA, low-rank approximation, pseudoinverse
Pseudoinverse	$A^\dagger = (A^\top A)^{-1}A^\top$	least squares, normal equations

Table A.1. The factorizations the book uses, and where.

APPENDIX B

Probability and Distributions Reference

A compact reference for the probability used throughout, developed in full in Chapters 1 to 3. The standard text is Bishop [3], Chapter 2.

B.1 The rules

For random variables X, Y: the *sum rule* $p(X) = \sum_Y p(X, Y)$ marginalizes; the *product rule* $p(X, Y) = p(X \mid Y)p(Y)$ factorizes; and *Bayes' theorem*

$$p(X \mid Y) = \frac{p(Y \mid X)\, p(X)}{p(Y)}, \qquad p(Y) = \sum_X p(Y \mid X)p(X),$$

inverts a conditional. Independence is $p(X, Y) = p(X)p(Y)$. The *change-of-variables* formula for a density under $y = g(x)$ is $p_y(y) = p_x(x)\,|dx/dy|$.

B.2 Moments

The expectation $\mathbb{E}[X] = \sum_x x\, p(x)$ (or $\int x\, p(x)\, dx$) is linear, $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$, always. The variance $\mathrm{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$ measures spread. The covariance $\mathrm{Cov}(X, Y) = \mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]$ measures linear association, zero if independent but not conversely. For a random vector, the covariance matrix $\boldsymbol{\Sigma} = \mathbb{E}[(\boldsymbol{x} - \boldsymbol{\mu})(\boldsymbol{x} - \boldsymbol{\mu})^\top]$ is symmetric PSD, and $\mathrm{Cov}(A\boldsymbol{x} + \boldsymbol{b}) = A\boldsymbol{\Sigma}A^\top$.

B.3 The distributions

Distribution	Support	Density / mass	Mean	Variance
Bernoulli(μ)	$\{0, 1\}$	$\mu^x(1-\mu)^{1-x}$	μ	$\mu(1-\mu)$
Binomial(N, μ)	$\{0, \dots, N\}$	$\binom{N}{m}\mu^m(1-\mu)^{N-m}$	$N\mu$	$N\mu(1-\mu)$
Beta(a, b)	$[0, 1]$	$\frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\mu^{a-1}(1-\mu)^{b-1}$	$\frac{a}{a+b}$	$\frac{ab}{(a+b)^2(a+b+1)}$
Categorical($\boldsymbol{\mu}$)	$\{1, \dots, K\}$	$\prod_k \mu_k^{x_k}$	μ_k	$\mu_k(1-\mu_k)$
Multinomial($N, \boldsymbol{\mu}$)	counts	$\binom{N}{\boldsymbol{m}}\prod_k \mu_k^{m_k}$	$N\mu_k$	$N\mu_k(1-\mu_k)$
Dirichlet($\boldsymbol{\alpha}$)	simplex	$\frac{\Gamma(\alpha_0)}{\prod_k \Gamma(\alpha_k)}\prod_k \mu_k^{\alpha_k - 1}$	$\frac{\alpha_k}{\alpha_0}$	$\frac{\alpha_k(\alpha_0-\alpha_k)}{\alpha_0^2(\alpha_0+1)}$
Gaussian(μ, σ^2)	$\mathbb{R}$	$\frac{1}{\sqrt{2\pi\sigma^2}}e^{-(x-\mu)^2/2\sigma^2}$	μ	σ^2
Gamma(a, b)	$(0, \infty)$	$\frac{b^a}{\Gamma(a)}x^{a-1}e^{-bx}$	a/b	a/b^2
Poisson(λ)	$\{0, 1, \dots\}$	$\frac{\lambda^x e^{-\lambda}}{x!}$	λ	λ

Table B.1. The distributions used in the book ($\alpha_0 = \sum_k \alpha_k$). The multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ has density $(2\pi)^{-d/2}|\boldsymbol{\Sigma}|^{-1/2}\exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu}))$.

B.4 Conjugacy

A conjugate prior keeps the posterior in the prior's family, so updating is arithmetic on the hyperparameters (Chapter 3 explains why this works for the whole exponential family).

Likelihood	**Conjugate prior**	**Posterior update**
Bernoulli / Binomial	Beta(a, b)	add successes to a, failures to b
Categorical / Multinomial	Dirichlet$(\boldsymbol{\alpha})$	add counts to $\boldsymbol{\alpha}$
Gaussian (known variance)	Gaussian on the mean	precision-weighted mean
Gaussian (known mean)	Gamma on the precision	add to shape and rate

Table B.2. Conjugate pairs used in the book.

B.5 Concentration inequalities

For nonnegative X, *Markov*: $\mathbb{P}(X \geq a) \leq \mathbb{E}[X]/a$. With a variance, *Chebyshev*: $\mathbb{P}(|X - \mu| \geq a) \leq \sigma^2/a^2$. For a sum of m independent bounded variables, *Hoeffding*: $\mathbb{P}(|\bar{X} - \mathbb{E}[\bar{X}]| \geq \varepsilon) \leq 2e^{-2m\varepsilon^2}$ (for $[0, 1]$ variables), an exponential tail. *Jensen*: for convex f, $\mathbb{E}[f(X)] \geq f(\mathbb{E}[X])$ (reversed for concave f, the form used for the logarithm in EM). These are the tools of Chapter 4.

B.6 Information theory

The *entropy* $\mathrm{H}[p] = -\sum_x p(x) \log p(x)$ measures uncertainty; the *Kullback–Leibler divergence* $\mathrm{KL}(p \,\|\, q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \geq 0$ measures the gap between distributions (zero iff $p = q$); the *cross-entropy* $\mathrm{H}(p, q) = -\sum_x p(x) \log q(x) = \mathrm{H}[p] + \mathrm{KL}(p \,\|\, q)$ is the classification loss; and the *mutual information* $I[X, Y] = \mathrm{KL}(p(x, y) \,\|\, p(x)p(y)) = \mathrm{H}[X] - \mathrm{H}[X \mid Y]$ measures dependence. These appear in Chapter 1 and as the cross-entropy losses throughout Parts III and IV.

APPENDIX C

Matrix Calculus and Lagrangian Duality

This appendix collects the two optimization tools the book uses repeatedly: matrix calculus, for the gradients of Chapters 5 to 11, and Lagrangian duality, for the support vector machine of Chapter 15. The reference for the latter is Boyd and Vandenberghe [5].

C.1 Gradients, Jacobians, and Hessians

For a scalar function $f(\boldsymbol{x})$ of a vector, the *gradient* ∇f is the vector of partial derivatives $(\partial f/\partial x_i)$, pointing in the direction of steepest ascent. The *Hessian* $\nabla^2 f$ is the matrix of second partials $(\partial^2 f/\partial x_i \partial x_j)$, symmetric for smooth f. For a vector-valued function $\boldsymbol{f}(\boldsymbol{x})$, the *Jacobian* is the matrix $J_{ij} = \partial f_i/\partial x_j$.

C.2 Matrix-calculus identities

The identities below, proved by differentiating componentwise, are the ones the text uses; here $\boldsymbol{a}$ is a constant vector and A a constant matrix.

Expression	Gradient / derivative
$\nabla_{\boldsymbol{x}}(\boldsymbol{a}^\top \boldsymbol{x})$	$\boldsymbol{a}$
$\nabla_{\boldsymbol{x}}(\boldsymbol{x}^\top A \boldsymbol{x})$	$(A + A^\top)\boldsymbol{x}$, $= 2A\boldsymbol{x}$ if A symmetric
$\nabla_{\boldsymbol{x}}\lVert\boldsymbol{x}\rVert^2$	$2\boldsymbol{x}$
$\nabla_{\boldsymbol{x}}\lVert A\boldsymbol{x} - \boldsymbol{b}\rVert^2$	$2A^\top(A\boldsymbol{x} - \boldsymbol{b})$
$\nabla_A \operatorname{tr}(AB)$	$B^\top$
$\nabla_A \log\lvert A\rvert$	$(A^{-1})^\top$
$\nabla_A(\boldsymbol{x}^\top A^{-1}\boldsymbol{x})$	$-(A^{-1})^\top \boldsymbol{x}\boldsymbol{x}^\top (A^{-1})^\top$

Table C.1. Matrix-calculus identities used in the book. The fourth gives the least-squares gradient (Chapter 5); the last two are used in fitting a Gaussian's covariance.

Example C.1 (The least-squares gradient). The sum-of-squares cost of Chapter 5 is $J(\boldsymbol{w}) = \frac{1}{2}\lVert\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}\rVert^2 = \frac{1}{2}(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})^\top(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t})$. Expanding, $J = \frac{1}{2}(\boldsymbol{w}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{w} - 2\boldsymbol{w}^\top\boldsymbol{\Phi}^\top\boldsymbol{t} + \boldsymbol{t}^\top\boldsymbol{t})$. Using $\nabla_{\boldsymbol{w}}(\boldsymbol{w}^\top A\boldsymbol{w}) = 2A\boldsymbol{w}$ for symmetric $A = \boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ and $\nabla_{\boldsymbol{w}}(\boldsymbol{w}^\top\boldsymbol{a}) = \boldsymbol{a}$ with $\boldsymbol{a} = \boldsymbol{\Phi}^\top\boldsymbol{t}$,

$$\nabla J = \boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{\Phi}^\top\boldsymbol{t} = \boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{w} - \boldsymbol{t}).$$

Setting this to zero gives the normal equations $\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{w} = \boldsymbol{\Phi}^\top\boldsymbol{t}$, and the Hessian $\nabla^2 J = \boldsymbol{\Phi}^\top\boldsymbol{\Phi} \succeq 0$ confirms convexity, so the stationary point is the global minimum. This is the derivation deferred from Chapter 5.

C.3 Convexity

A function f is *convex* if its graph lies below every chord, $f(\eta \boldsymbol{x} + (1-\eta)\boldsymbol{y}) \le \eta f(\boldsymbol{x}) + (1-\eta) f(\boldsymbol{y})$ for $\eta \in [0,1]$. For twice-differentiable f, convexity is equivalent to a positive semidefinite Hessian everywhere, $\nabla^2 f \succeq 0$. Convexity is the property that makes optimization easy: a convex function has no local minima other than its global minimum, so a zero gradient certifies the global optimum and gradient descent converges to it. The squared-error and cross-entropy losses (Chapters 5 to 8) are convex; neural-network losses (Chapter 11) are not.

C.4 Constrained optimization and Lagrange multipliers

To minimize $f_0(\boldsymbol{x})$ subject to inequality constraints $f_i(\boldsymbol{x}) \le 0$ and equality constraints $h_j(\boldsymbol{x}) = 0$, form the *Lagrangian*

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f_0(\boldsymbol{x}) + \sum_i \lambda_i f_i(\boldsymbol{x}) + \sum_j \nu_j h_j(\boldsymbol{x}), \qquad \lambda_i \ge 0, \tag{C.1}$$

with one multiplier per constraint. The *dual function* is the infimum over the primal variable, $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$, and the *dual problem* maximizes it, $\max_{\boldsymbol{\lambda} \ge 0, \boldsymbol{\nu}} g(\boldsymbol{\lambda}, \boldsymbol{\nu})$.

Weak and strong duality

The dual is always a lower bound on the primal optimum (*weak duality*): $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \le p^*$ for all feasible $\boldsymbol{\lambda} \ge 0, \boldsymbol{\nu}$, so $d^* = \max g \le p^*$. For convex problems satisfying a constraint qualification (such as Slater's condition, a strictly feasible point), the gap closes (*strong duality*): $d^* = p^*$.

Weak duality. For any primal-feasible $\boldsymbol{x}$ (so $f_i(\boldsymbol{x}) \le 0$ and $h_j(\boldsymbol{x}) = 0$) and any $\boldsymbol{\lambda} \ge 0$, each term $\lambda_i f_i(\boldsymbol{x}) \le 0$ and $\nu_j h_j(\boldsymbol{x}) = 0$, so $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) \le f_0(\boldsymbol{x})$. Taking the infimum over $\boldsymbol{x}$ on the left only lowers it, $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) = \inf_{\boldsymbol{x}} \mathcal{L} \le f_0(\boldsymbol{x})$, and since this holds for every feasible $\boldsymbol{x}$ it holds for the minimizer, $g(\boldsymbol{\lambda}, \boldsymbol{\nu}) \le p^*$. ■

C.5 The KKT conditions

At an optimum of a convex problem with strong duality, the primal $\boldsymbol{x}^*$ and dual $\boldsymbol{\lambda}^*, \boldsymbol{\nu}^*$ satisfy the *Karush–Kuhn–Tucker* (KKT) conditions:

- **Stationarity:** $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \boldsymbol{0}$.
- **Primal feasibility:** $f_i(\boldsymbol{x}^*) \le 0$ and $h_j(\boldsymbol{x}^*) = 0$.
- **Dual feasibility:** $\lambda_i^* \ge 0$.
- **Complementary slackness:** $\lambda_i^* \, f_i(\boldsymbol{x}^*) = 0$ for every i.

Complementary slackness is the consequential one: it forces $\lambda_i^* = 0$ unless the constraint is active ($f_i(\boldsymbol{x}^*) = 0$). In the support vector machine of Chapter 15 this is exactly why only the points on the margin (the active constraints) have nonzero multipliers and become support vectors, and why the solution is sparse. The KKT conditions are the multivariate, inequality-constrained generalization of "set the derivative to zero," and they are the optimality certificate behind the SVM dual.

APPENDIX D

Software, Computing, and the Assignments

The course is taught with code, and the programming assignments are where the mathematics becomes a working model. This appendix records the tools and the shape of the assignments, so the exposition stays self-contained.

D.1 Environment

The work is done in Python. A clean, reproducible setup uses a virtual environment (for example a `conda` environment pinned to a specific Python version) into which the scientific stack is installed: `numpy` for arrays and linear algebra, `matplotlib` for plotting, `pandas` for tabular data, and Jupyter for interactive notebooks. An editor with notebook support completes the toolchain. Pinning the environment, rather than installing into the system Python, keeps results reproducible across machines.

D.2 NumPy and pandas essentials

Nearly all the computation is vectorized. NumPy arrays support elementwise arithmetic, slicing, boolean masking, and *broadcasting* (operating on arrays of compatible but unequal shapes without explicit loops), which turns the sums and products of the book's equations into one-line, fast operations: a design matrix times a weight vector is a single matrix product, a whole batch of predictions is one call. The linear-algebra routines (matrix product, inverse, solve, eigendecomposition, SVD) implement the operations of Appendix A directly. Pandas adds labeled tables (data frames) for loading and cleaning real data sets before they are handed to NumPy as arrays.

D.3 Version control

Code is kept under version control with `git`: a local repository tracks changes through commits, and a remote hosts the shared history. The everyday cycle is to pull the latest changes, edit, stage and commit the edits with a message, and push. Committing in small, well-described steps makes the work auditable and recoverable, and is the professional default for any code, research included.

D.4 The assignments

The programming work runs in parallel with the theory and reinforces it.

- **A warm-up predictor.** Before any technique is taught, the first assignment builds a simple adaptive predictor from logged outcomes (recording many rounds of a repeated game, then learning the opponent's tendencies to predict the best next move). It is graded on the quality of the design and analysis rather than the win rate, and its purpose is to establish the workflow: state the problem, design on paper, implement, and reflect.
- **Regression and classification from scratch.** Implementing linear regression by both the normal equations and gradient descent (Chapter 5), k-nearest-neighbors with proper feature normalization computed from the training set only, and a naive Bayes classifier with the log-probability trick to avoid underflow (Chapter 8), then evaluating them with a confusion matrix and the accuracy, precision, recall, and F_1 metrics of Chapter 8.
- **Kernel machines.** Solving the support vector machine of Chapter 15 as a quadratic program with a standard convex-optimization solver, trying the polynomial and radial-basis kernels of Chapter 14, and handling multiple classes by the one-versus-rest and one-versus-one schemes.
- **Neural networks.** Implementing the forward and backward passes of Chapter 11 from scratch, checking the analytic gradient against a finite-difference estimate, and training by stochastic gradient descent.

D.5 Good practice

A few habits recur across the assignments and matter for honest results. Split the data into training, validation, and test sets, and touch the test set only once, at the very end (Chapter 4). Compute any normalization (means, standard deviations) on the training set alone and apply those same statistics to the validation and test sets, never peeking at them. Fix the random seed so runs are reproducible. And prefer vectorized array operations to explicit loops, both for speed and because they mirror the matrix equations of the text.

D.6 The course project

Beyond the assignments, the course culminates in a team project that runs the full machine learning pipeline on a real data set: framing a specific task, surveying the relevant literature, cleaning and exploring the data, developing and comparing models, and reporting the results in the format of a research paper. The project is where the parts of this book are assembled into a working system, the point of all the theory.

APPENDIX E

Practice Problem Bank

This appendix is a bank of practice problems grouped by topic, one or two per area, for quick targeted review. The two timed, full-length *practice examinations*, a midterm and a comprehensive final worked in complete detail, form the Practice Examinations part that precedes the appendices (Chapter 28 and Chapter 29). Every problem here is stated in full and solved completely; cover the solution and work each first.

Exam problems come in the three kinds named in the front matter, and it helps to recognize which you are facing: *compute* (carry out a procedure on numbers, where partial credit rewards showing every step), *prove* (establish a fact, where naming the definition or inequality you start from is half the work), and *explain* (justify a modeling choice in a sentence or two, where a concrete reason beats a vague one). A good exam strategy is to scan for the compute problems first, since they are the most mechanical, then return to the proofs and explanations.

E.1 A Bank of Practice Problems

Probability and estimation

Problem 1: Naive Bayes with smoothing

A spam filter is trained on five spam and five ham emails. The word "offer" appears in 3 of the 5 spam emails and in 1 of the 5 ham. Using Laplace smoothing and equal class priors, classify a new email that contains "offer."

Solution. Laplace smoothing adds one pseudo-count per outcome: $\hat{p}(\text{offer} \mid \text{spam}) = (3+1)/(5+2) = 4/7$ and $\hat{p}(\text{offer} \mid \text{ham}) = (1+1)/(5+2) = 2/7$. With equal priors $p(\text{spam}) = p(\text{ham}) = 1/2$, the unnormalized posteriors are $p(\text{spam} \mid \text{offer}) \propto \frac{1}{2} \cdot \frac{4}{7} = \frac{2}{7}$ and $p(\text{ham} \mid \text{offer}) \propto \frac{1}{2} \cdot \frac{2}{7} = \frac{1}{7}$. Normalizing, $p(\text{spam} \mid \text{offer}) = (2/7)/(2/7 + 1/7) = 2/3 \approx 0.67$: about two-thirds spam.

Problem 2: Maximum likelihood for the exponential

Independent samples $x_1, \dots, x_N$ are drawn from an exponential distribution with density $p(x \mid \lambda) = \lambda e^{-\lambda x}$ for $x \geq 0$. Derive the maximum likelihood estimate of λ, and evaluate it for the data $\{2, 4, 6\}$.

Solution. The log-likelihood is $\ell(\lambda) = \sum_n [\log \lambda - \lambda x_n] = N \log \lambda - \lambda \sum_n x_n$. Differentiating, $\ell'(\lambda) = N/\lambda - \sum_n x_n = 0$, so $\hat{\lambda} = N / \sum_n x_n = 1/\bar{x}$, the reciprocal of the sample mean. For $\{2, 4, 6\}$ the mean is $\bar{x} = 12/3 = 4$, so $\hat{\lambda} = 1/4 = 0.25$. (The second derivative $-N/\lambda^2 < 0$ confirms a maximum.)

Regression

Problem 3: Least-squares line fit

Fit a line $y = w_0 + w_1 x$ by least squares to the four points $(0,1), (1,2), (2,2), (3,5)$ using the normal equations.

Solution. With design matrix $\boldsymbol{\Phi} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix}$ and targets $\boldsymbol{t} = (1, 2, 2, 5)$, the normal-equation matrices are $\boldsymbol{\Phi}^\top \boldsymbol{\Phi} = \left[\begin{smallmatrix} 4 & 6 \\ 6 & 14 \end{smallmatrix}\right]$ (entries: count 4, $\sum x = 6$, $\sum x^2 = 14$) and $\boldsymbol{\Phi}^\top \boldsymbol{t} = (\sum t, \ \sum xt) = (10, 21)$. The determinant is $4 \cdot 14 - 36 = 20$, so $\boldsymbol{w} = \frac{1}{20} \left[\begin{smallmatrix} 14 & -6 \\ -6 & 4 \end{smallmatrix}\right] (10, 21) = \frac{1}{20}(140 - 126, \ -60 + 84) = \frac{1}{20}(14, 24) = (0.7, 1.2)$. The fitted line is $y = 0.7 + 1.2x$. (Check: predictions $0.7, 1.9, 3.1, 4.3$ give residuals $0.3, 0.1, -1.1, 0.7$ that sum to zero, as least squares requires.)

Classification

Problem 4: Logistic decision boundary

A logistic-regression model is $p(\mathcal{C}_1 \mid \boldsymbol{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2)$ with weights $\boldsymbol{w} = (-1, 1, 1)$. Give the decision boundary, and classify the points $(1, 1)$ and $(0, 0)$ with their probabilities.

Solution. The boundary is where the argument is zero, $-1 + x_1 + x_2 = 0$, that is the line $x_1 + x_2 = 1$. At $(1, 1)$: the argument is $-1 + 1 + 1 = 1 > 0$, so class $\mathcal{C}_1$ with probability $\sigma(1) = 1/(1 + e^{-1}) = 0.731$. At $(0, 0)$: the argument is $-1 < 0$, so class $\mathcal{C}_2$, with $p(\mathcal{C}_1) = \sigma(-1) = 0.269$. The two points lie on opposite sides of the line $x_1 + x_2 = 1$, equally far from it, so their probabilities are symmetric about $\frac{1}{2}$.

Neural networks

Problem 5: A backpropagation pass

A network has one input, one sigmoid hidden unit, and one linear output, with squared-error loss $E = \frac{1}{2}(\hat{y} - t)^2$ and weights $w_1 = 0.5$ (input to hidden), $w_2 = 1$ (hidden to output), zero biases. For the example $x = 2$, $t = 1$, compute the forward pass and the gradients $\partial E/\partial w_2$ and $\partial E/\partial w_1$.

Solution. Forward. $a_1 = w_1 x = (0.5)(2) = 1$; $z_1 = \sigma(1) = 0.731$; $\hat{y} = w_2 z_1 = (1)(0.731) = 0.731$; $E = \frac{1}{2}(0.731 - 1)^2 = \frac{1}{2}(0.0724) = 0.036$. *Backward.* The output error is $\delta_2 = \hat{y} - t = -0.269$, so $\partial E/\partial w_2 = \delta_2 z_1 = (-0.269)(0.731) = -0.197$. For the hidden unit, $\sigma'(1) = 0.731(1 - 0.731) = 0.197$, so $\delta_1 = w_2 \delta_2 \, \sigma'(1) = (1)(-0.269)(0.197) = -0.053$, giving $\partial E/\partial w_1 = \delta_1 x = (-0.053)(2) = -0.106$. Both gradients are negative, so a descent step increases both weights, pushing $\hat{y}$ from 0.731 up toward the target 1.

Kernel machines

Problem 6: A two-point SVM

Find the maximum-margin classifier for the two points $\boldsymbol{x}_1 = (0, 2)$ with $y_1 = +1$ and $\boldsymbol{x}_2 = (0, 0)$ with $y_2 = -1$: give $\boldsymbol{w}$, b, and the margin.

Solution. By symmetry the boundary is horizontal, midway between the points, so $\boldsymbol{w} = (0, w)$. The canonical margin constraints $y_1(\boldsymbol{w}^\top \boldsymbol{x}_1 + b) = 1$ and $y_2(\boldsymbol{w}^\top \boldsymbol{x}_2 + b) = 1$ read $2w + b = 1$ and $-(0 + b) = 1$. The second gives $b = -1$, and then $2w - 1 = 1$ gives $w = 1$, so $\boldsymbol{w} = (0, 1)$ and the boundary is $x_2 = 1$. The margin is $2/\|\boldsymbol{w}\| = 2/1 = 2$, with each point one unit from the boundary; both are support vectors.

Unsupervised learning

Problem 7: A k-means step

Run one full iteration of k-means with $K = 2$ on the points $\{1, 2, 9, 10\}$, starting from centers $\mu_1 = 1$, $\mu_2 = 10$, and report the final distortion.

Solution. Assignment: 1 and 2 are nearer $\mu_1 = 1$, and 9, 10 nearer $\mu_2 = 10$, so cluster $1 = \{1, 2\}$, cluster $2 = \{9, 10\}$. *Update:* $\mu_1 = (1 + 2)/2 = 1.5$, $\mu_2 = (9 + 10)/2 = 9.5$. A recheck gives the same assignment, so it has converged. The distortion is $J = [(1 - 1.5)^2 + (2 - 1.5)^2] + [(9 - 9.5)^2 + (10 - 9.5)^2] = (0.25 + 0.25) + (0.25 + 0.25) = 1$.

Problem 8: Principal components

A centered data set has covariance $\boldsymbol{\Sigma} = \left[\begin{smallmatrix} 5 & 4 \\ 4 & 5 \end{smallmatrix}\right]$. Find the principal components and the fraction of variance the first captures.

Solution. The eigenvalues solve $(5 - \lambda)^2 - 16 = 0$, so $5 - \lambda = \pm 4$ and $\lambda_1 = 9$, $\lambda_2 = 1$. For $\lambda_1 = 9$: $(\boldsymbol{\Sigma} - 9I)\boldsymbol{u} = \left[\begin{smallmatrix} -4 & 4 \\ 4 & -4 \end{smallmatrix}\right]\boldsymbol{u} = \boldsymbol{0}$ gives $u_1 = u_2$, so $\boldsymbol{u}_1 = \frac{1}{\sqrt{2}}(1, 1)$; similarly $\boldsymbol{u}_2 = \frac{1}{\sqrt{2}}(1, -1)$. The first principal component captures $\lambda_1/(\lambda_1 + \lambda_2) = 9/10 = 90\%$ of the variance, so reducing to one dimension along $\boldsymbol{u}_1$ loses only 10%.

Learning theory and graphical models

Problem 9: Sample complexity

A finite hypothesis class has $k = 100$ members. How many independent samples guarantee, with probability at least 0.95, that every hypothesis's training error is within $\gamma = 0.1$ of its true error?

Solution. The uniform-convergence sample complexity is $m \geq \frac{1}{2\gamma^2} \ln(2k/\delta)$ with confidence $1 - \delta$. Here $\delta = 0.05$, $\gamma = 0.1$, $k = 100$, so $m \geq \frac{1}{2(0.01)} \ln(2 \cdot 100/0.05) = 50 \ln(4000) = 50(8.29) \approx 415$. About 415 samples suffice. The dependence on k and δ is only logarithmic, but halving γ would quadruple m.

Problem 10: Explaining away

In the collider $X \to Z \leftarrow Y$ with X, Y independent fair coins and $Z = X \vee Y$, compute $p(X = 1 \mid Z = 1)$ and explain why observing $Y = 1$ changes it.

Solution. The event $Z = 1$ has probability $1 - p(X{=}0, Y{=}0) = 1 - \frac{1}{4} = \frac{3}{4}$, and $X = 1$ forces $Z = 1$, so $p(X{=}1, Z{=}1) = p(X{=}1) = \frac{1}{2}$, giving $p(X{=}1 \mid Z{=}1) = \frac{1}{2}/\frac{3}{4} = \frac{2}{3}$. Now also observe $Y = 1$: since $Y = 1$ already makes $Z = 1$, the observation $Z = 1$ tells us nothing more about X, so $p(X{=}1 \mid Z{=}1, Y{=}1) = p(X{=}1) = \frac{1}{2}$. Conditioning on the common effect Z coupled the two independent causes (raising X's probability to $\frac{2}{3}$); learning the other cause Y then explains away the evidence, dropping it back to $\frac{1}{2}$.

Bibliography

[1] David Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[2] Julian Besag. Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society: Series B*, 36(2):192–236, 1974.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

[4] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Workshop on Computational Learning Theory (COLT)*, 1992.

[5] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[6] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[7] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[8] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[9] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[10] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B*, 39(1):1–38, 1977.

[11] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.

[12] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[13] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2nd edition, 2009.

[15] W. Keith Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57 (1):97–109, 1970.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[17] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[20] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[21] Diederik P. Kingma and Jimmy Ba. Adam: a method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[22] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[23] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2012.

[24] Solomon Kullback and Richard A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22 (1):79–86, 1951.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[26] David J. C. MacKay. A practical bayesian framework for backpropagation networks. *Neural Computation*, 4(3): 448–472, 1992.

[27] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[28] John McCarthy, Marvin L. Minsky, Nathaniel Rochester, and Claude E. Shannon. A proposal for the Dartmouth summer research project on artificial intelligence. *AI Magazine*, 27(4):12–14, 2006. Proposal dated 1955; reprinted in AI Magazine, vol. 27, no. 4, 2006.

[29] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[30] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[31] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[32] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[33] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[34] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[35] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3):379–423, 1948.

[36] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[37] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 5th edition, 2016.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[39] Michael E. Tipping and Christopher M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society: Series B*, 61(3):611–622, 1999.

[40] Leslie G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, 1984.

[41] Vladimir N. Vapnik and Alexey Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and Its Applications*, 16(2):264–280, 1971.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.