Machine Learning

A Complete Course — Chapter 2: Mathematical Foundations

Pr. ABDELLAOUI ▪ www.abdellaoui.net
ENS-Meknes — Université Moulay Ismaïl ▪ 2026

Chapter outline

Linear Algebra I: Vectors, Matrices, Norms

Linear Algebra II: Eigendecomposition and SVD

Multivariate Calculus and Matrix Calculus

Probability I: Foundations

Probability II: Distributions and the Gaussian

Statistics and Estimation: MLE, MAP, Bayes

Information Theory

Linear Algebra I: Vectors, Matrices, Norms

Section 1 — Chapter 2

Learning objectives of this chapter

A toolbox chapter: know what each tool does and where to find it later.

By the end of this chapter you will be able to:

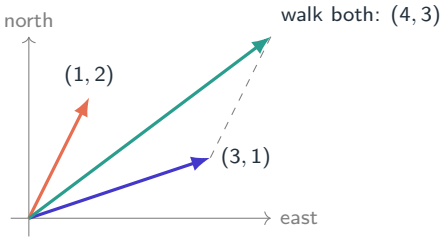
- Use vectors, matrices, norms, projections, eigen- and SVD
- Compute gradients and Hessians of ML objectives
- Use Bayes' rule, expectations, the Gaussian, the exponential family
- Derive maximum-likelihood and MAP estimators
- Use entropy, cross-entropy and KL divergence
- Run gradient descent, SGD, Newton; handle constraints (KKT)

Why a whole chapter of mathematics?

Every algorithm in Chapters 3–10 is *derived*, not just described. This chapter is the toolbox for those derivations.

Intuition: vectors are arrows and lists

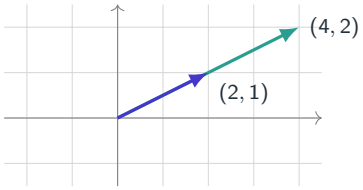
- A treasure map says "3 steps east, 1 step north". That is the vector $(3, 1)$: an arrow you can draw.
- A recipe card says "80 m^2, 3 rooms, 20 years old". That is also a vector: a *list* describing one house.
- In machine learning every sample is such a list; d features means a list of d numbers.
- Adding arrows = walking one after the other. Scaling an arrow = making it twice as long.
- The dot product asks "do these two arrows push the same way?" Same way: big positive number. Sideways: zero. Opposite: negative.



In one sentence

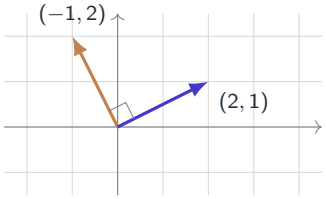
A vector is a list of numbers that you can also picture as an arrow.

Picture: dot product, three cases



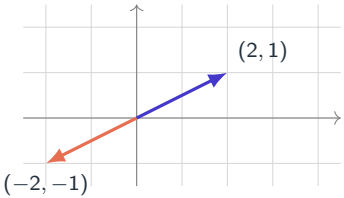
same way

$2\times4+1\times2=\mathbf{10}$



sideways (a right angle)

$2\times(-1)+1\times2=\mathbf{0}$



opposite

$2\times(-2)+1\times(-1)=\mathbf{-5}$

- The blue arrow $(2,1)$ is the same in all three pictures; only the second arrow changes.
- Recipe: multiply the first numbers, multiply the second numbers, add the two results.
- Same way: a big plus number. Sideways (a right angle): exactly 0. Opposite: a minus number.

In one sentence

The dot product is one number that says how much two arrows agree.

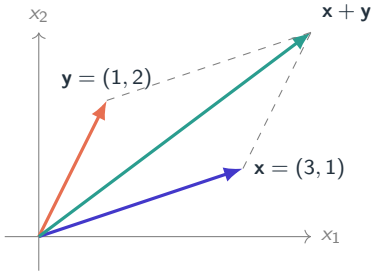
A vector is a list of numbers; the dot product says how much two vectors agree.

Vector

$\mathbf{x} = (x_1, \ldots, x_d)^\top \in \mathbb{R}^d$: a column of d real numbers. In ML, a sample is a vector of d features.

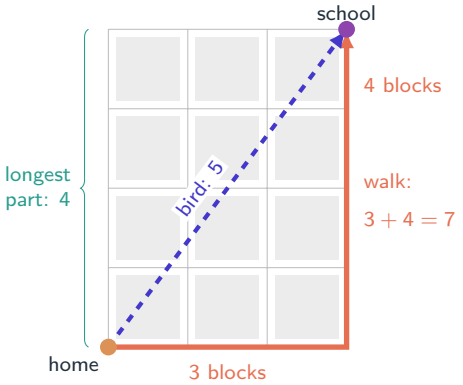
Operations:

- Addition: $(\mathbf{x} + \mathbf{y})_j = x_j + y_j$
- Scaling: $(\alpha \mathbf{x})_j = \alpha x_j$
- Linear combination: $\sum_k \alpha_k \mathbf{x}_k$
- Inner (dot) product: $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^\top \mathbf{y} = \sum_{j=1}^{d} x_j y_j$



$\mathbf{x}^\top \mathbf{y} = 3 \cdot 1 + 1 \cdot 2 = 5$

Picture: two ways to measure length



- From home to school the arrow is $(3, 4)$: 3 blocks east, 4 blocks north.
- A bird flies straight: 5 blocks (check: $3{\times}3 + 4{\times}4 = 25 = 5{\times}5$).
- A person walks along the streets: $3 + 4 = 7$ blocks. The longest straight part of the walk is 4 blocks.
- Three honest rulers for one arrow; maths calls them ℓ_2 (bird), ℓ_1 (walker), ℓ_∞ (longest part).

In one sentence

A norm is a ruler for arrows, and there is more than one honest ruler.

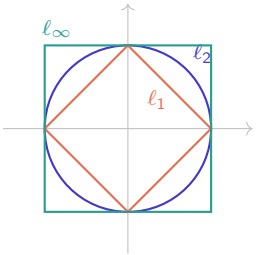
A norm measures length; the shape of its unit ball is why ℓ_1 gives zeros.

Norm

A function $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}_+$ with (i) $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$, (ii) $\|\alpha\mathbf{x}\| = |\alpha|\,\|\mathbf{x}\|$, (iii) $\|\mathbf{x}+\mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$ (triangle inequality).

$$\|\mathbf{x}\|_p = \Big(\sum_{j=1}^{d} |x_j|^p\Big)^{1/p}, \quad p \ge 1$$

- ℓ_2 (Euclidean): $\|\mathbf{x}\|_2 = \sqrt{\mathbf{x}^\top\mathbf{x}}$
- ℓ_1 (Manhattan): $\|\mathbf{x}\|_1 = \sum_j |x_j|$
- ℓ_∞: $\|\mathbf{x}\|_\infty = \max_j |x_j|$
- "ℓ_0": number of non-zeros (not a norm)



Unit balls $\{\mathbf{x} : \|\mathbf{x}\|_p = 1\}$ in $\mathbb{R}^2$.

ML relevance

Distance: how far apart; cosine: do they point the same way?

- Euclidean distance: $d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2$
- Angle: $\cos\theta = \dfrac{\mathbf{x}^\top \mathbf{y}}{\|\mathbf{x}\|\ \|\mathbf{y}\|}$
- Cosine similarity (text, embeddings): $\cos\theta \in [-1, 1]$
- Orthogonal: $\mathbf{x} \perp \mathbf{y} \iff \mathbf{x}^\top \mathbf{y} = 0$

Cauchy–Schwarz inequality

$$\left|\mathbf{x}^\top \mathbf{y}\right| \le \|\mathbf{x}\|_2\, \|\mathbf{y}\|_2\,,$$

with equality iff $\mathbf{x}$ and $\mathbf{y}$ are collinear.

Worked example

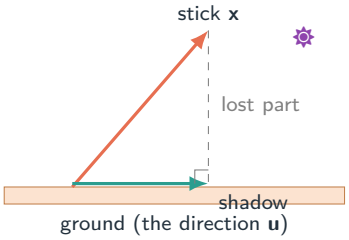
$\mathbf{x} = (3, 1)$, $\mathbf{y} = (1, 2)$:
$\|\mathbf{x}\| = \sqrt{10}$, $\|\mathbf{y}\| = \sqrt{5}$, $\mathbf{x}^\top \mathbf{y} = 5$
$\cos\theta = \dfrac{5}{\sqrt{50}} = 0.707 \Rightarrow \theta = 45°$
$d(\mathbf{x}, \mathbf{y}) = \|(2, -1)\| = \sqrt{5}$

Where it appears

k-NN and k-means use $d(\cdot, \cdot)$; kernels generalize $\mathbf{x}^\top \mathbf{y}$; Cauchy–Schwarz bounds appear in convergence proofs.

Intuition: a projection is a shadow

- Hold a stick above the ground under the midday sun. Its shadow is the *projection* of the stick onto the ground.
- The shadow keeps the part of the stick that lies "along the ground" and loses the part that points up.
- The piece that is lost (from the tip of the shadow up to the tip of the stick) stands straight up: it is *perpendicular* to the ground.
- Least squares (Chapter 3) is exactly this: the data vector $\mathbf{y}$ is the stick, the predictions a model can make are the ground, and the fitted values are the shadow.



In one sentence

Projecting means keeping the shadow and dropping what stands straight up from it.

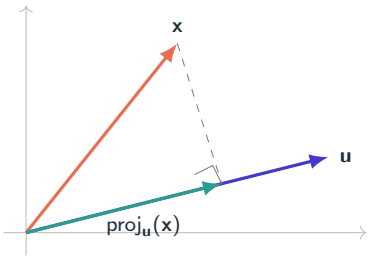
Projection onto a vector

Keep the part of $\mathbf{x}$ *along* $\mathbf{u}$*, drop the rest.*

Orthogonal projection of x onto u

$$\text{proj}_{\mathbf{u}}(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{u}}{\|\mathbf{u}\|^2}\,\mathbf{u} = (\mathbf{x}^\top \hat{\mathbf{u}})\hat{\mathbf{u}}, \quad \hat{\mathbf{u}} = \frac{\mathbf{u}}{\|\mathbf{u}\|}$$

- Residual $\mathbf{x} - \text{proj}_{\mathbf{u}}(\mathbf{x})$ is orthogonal to $\mathbf{u}$.
- PCA (Chapter 8) projects data on the directions of largest variance; least squares (Chapter 3) projects $\mathbf{y}$ onto the column space of X.



$\mathbf{x} = (2, 2.5)$, $\mathbf{u} = (4, 1)$: $\dfrac{\mathbf{x}^\top \mathbf{u}}{\|\mathbf{u}\|^2} = \dfrac{10.5}{17}$, proj $= (2.47, 0.62)$

Intuition: matrices are machines and tables

- **As a machine**: an arrow goes in, an arrow comes out. The machine can stretch, squash, rotate or flip. Two machines in a row = multiplying two matrices (and the order matters, like putting on socks and shoes).
- **As a table**: the data matrix X has one row per house and one column per feature. It is just the spreadsheet.
- **Scoring recipe**: multiplying the table by a weight vector $\mathbf{w}$ gives every house a score at once: "2 points per square metre, 10 points per room, minus 1 point per year of age". That score is the prediction.

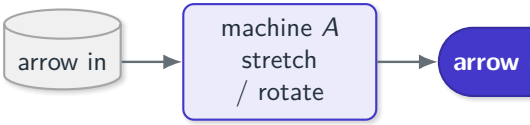
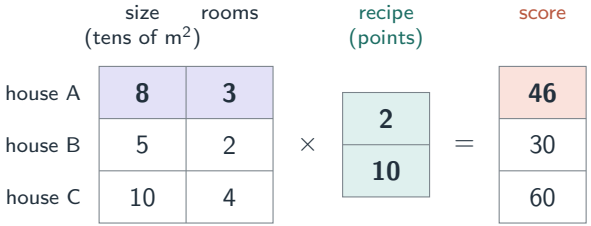


table $X \times$ recipe $\mathbf{w} =$
a score for every row

In one sentence

A matrix is a spreadsheet of numbers, and multiplying by it applies one recipe to every row.

Picture: a table times a recipe



house A: $8\times2 + 3\times10 = 46$

- One row per house, one column per feature: house A has 80 m^2 (8 tens) and 3 rooms. A matrix is just such a table.
- The recipe gives 2 points per ten m^2 and 10 points per room.
- Row A meets the recipe: $8\times2 + 3\times10 = 46$. Do the same for rows B and C: every score in one go.

In one sentence

A table times a recipe gives every row its score at once.

Matrices

A table of numbers; data matrix times weights gives all predictions at once.

Matrix

$A \in \mathbb{R}^{m \times n}$: m rows, n columns, entries A_{ij}. The design matrix $X \in \mathbb{R}^{n \times d}$ stacks the n samples as rows.

- Transpose: $(A^\top)_{ij} = A_{ji}$; $(AB)^\top = B^\top A^\top$
- Product: $(AB)_{ij} = \sum_k A_{ik} B_{kj}$ requires $A \in \mathbb{R}^{m \times k}$, $B \in \mathbb{R}^{k \times n}$; cost $O(mkn)$
- Not commutative: $AB \neq BA$ in general
- Symmetric: $A = A^\top$ (covariance matrices, kernels, Hessians)

Three views of $X\mathbf{w}$ ($X \in \mathbb{R}^{n \times d}$, $\mathbf{w} \in \mathbb{R}^d$):

1. Row view: $(X\mathbf{w})_i = \mathbf{x}_i^\top \mathbf{w}$ — one prediction per sample
2. Column view: $X\mathbf{w} = \sum_j w_j X_{:,j}$ — a linear combination of feature columns
3. Map view: $\mathbf{w} \mapsto X\mathbf{w}$ is a linear map $\mathbb{R}^d \to \mathbb{R}^n$

Matrix products that appear everywhere in ML

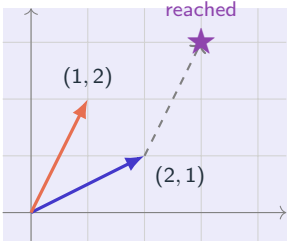
Learn the shapes and meanings, not the formulas; they return in every chapter.

Expression	Shape	Meaning	Appears in
$X\mathbf{w}$	$n \times 1$	predictions of a linear model	regression, logistic regression
$X^\top X$	$d \times d$	(unnormalized) feature Gram matrix, $\propto$ covariance	normal equations, PCA
$XX^\top$	$n \times n$	sample Gram matrix $G_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$	kernels, SVM dual
$X^\top \mathbf{y}$	$d \times 1$	feature–target correlations	normal equations, gradients
$\mathbf{x}\mathbf{x}^\top$	$d \times d$	rank-1 outer product	covariance, Hessian of squared loss
$\mathbf{w}^\top A\mathbf{w}$	scalar	quadratic form	Gaussian log-density, Mahalanobis, energy
$\mathrm{tr}(A)$	scalar	$\sum_i A_{ii} = \sum_i \lambda_i$	effective degrees of freedom

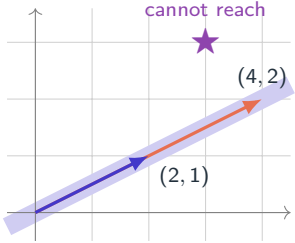
Useful identities

$$\mathrm{tr}(AB) = \mathrm{tr}(BA), \quad \mathbf{x}^\top A\mathbf{y} = \mathrm{tr}(A\mathbf{y}\mathbf{x}^\top), \quad \|\mathbf{x}\|_2^2 = \mathbf{x}^\top\mathbf{x} = \mathrm{tr}(\mathbf{x}\mathbf{x}^\top).$$

Picture: a copied arrow adds nothing



two directions: all the map
rank 2



a copy: only one line
rank 1

- An arrow is a direction you may walk in, as many steps as you like, forwards or backwards.
- Left: two different directions reach every point. The star: one blue step, then one red step.
- Right: the red arrow is the blue one, twice as long. No new direction: you stay on one line, so the star is out of reach.

In one sentence

Rank counts truly different directions; a copy adds none.

Linear independence, span, rank

Rank counts truly different directions; a duplicated feature adds none.

Definitions

- $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are **linearly independent** if $\sum \alpha_i \mathbf{x}_i = \mathbf{0} \Rightarrow \alpha_i = 0 \ \forall i$.
- $\text{span}\{\mathbf{x}_1, \ldots, \mathbf{x}_k\} = \{\sum \alpha_i \mathbf{x}_i\}$: a subspace.
- $\text{rank}(A)$ = number of linearly independent columns (= rows).

- $\text{rank}(A) \leq \min(m, n)$; full rank if equality.
- $\text{rank}(X^\top X) = \text{rank}(X)$.

Rank deficiency in ML

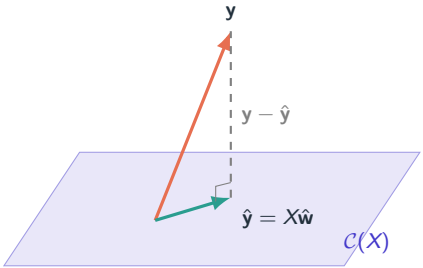
If X has rank $< d$ (e.g. a duplicated feature, one-hot with all m categories plus an intercept, or $n < d$), then $X^\top X$ is singular and the normal equations have infinitely many solutions.

Fixes: drop redundant columns, regularize ($X^\top X + \lambda I$ is always invertible for $\lambda > 0$), or use the pseudo-inverse.

Column space, null space, and least squares geometry

Least squares = the closest point to $\mathbf{y}$ *a linear model can reach.*

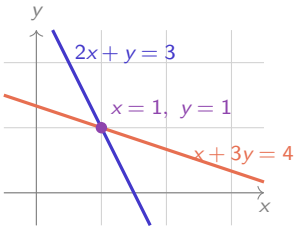
- Column space $\mathcal{C}(X) = \{X\mathbf{w} : \mathbf{w} \in \mathbb{R}^d\} \subseteq \mathbb{R}^n$: all vectors of predictions a linear model can produce.
- Null space $\mathcal{N}(X) = \{\mathbf{w} : X\mathbf{w} = \mathbf{0}\}$: parameter directions that change nothing.



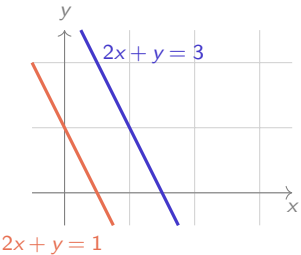
Least squares

$\hat{\mathbf{y}} = X\hat{\mathbf{w}}$ is the projection of $\mathbf{y}$ onto $\mathcal{C}(X)$; the residual is orthogonal to it: $X^\top(\mathbf{y} - X\hat{\mathbf{w}}) = \mathbf{0}$, the normal equations.

Picture: solving = two lines crossing



they cross once: one answer



parallel: no crossing, no answer

- Each line is one sentence about two unknown numbers: "two x plus one y makes 3".
- The crossing point obeys both sentences: $x = 1$, $y = 1$. Check: $2 + 1 = 3$ and $1 + 3 = 4$.
- The four numbers in front of x and y form the matrix; inverting it finds the crossing. No crossing: no inverse.

In one sentence

Solving two equations means finding where their lines cross.

Inverse and solving linear systems

Never compute the inverse; solve the system directly.

Inverse

$A \in \mathbb{R}^{n \times n}$ is invertible iff $\operatorname{rank}(A) = n$ iff $\det A \neq 0$; then $A^{-1}A = AA^{-1} = I$.

- $(AB)^{-1} = B^{-1}A^{-1}$, $(A^\top)^{-1} = (A^{-1})^\top$
- $A\mathbf{x} = \mathbf{b} \Rightarrow \mathbf{x} = A^{-1}\mathbf{b}$, but never compute A^{-1} explicitly: solve via LU / Cholesky ($O(n^3)$, better constants, better numerics).
- `np.linalg.solve(A, b)` not `inv(A) @ b`

2×2 example

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 3 \end{pmatrix}, \quad \det A = 6 - 1 = 5,$$

$$A^{-1} = \tfrac{1}{5}\begin{pmatrix} 3 & -1 \\ -1 & 2 \end{pmatrix}$$

Solve $A\mathbf{x} = (3, 4)^\top$: $\mathbf{x} = \frac{1}{5}(9 - 4, -3 + 8) = (1, 1)^\top$. Check: $2+1 = 3$, $1+3 = 4$ ✔

Determinant and trace

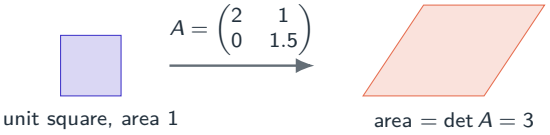
Determinant: how much areas stretch; trace: sum of the diagonal.

Determinant $\det A$ (square A):

- Signed volume scaling of the map $\mathbf{x} \mapsto A\mathbf{x}$
- $\det(AB) = \det A \det B$, $\det A^\top = \det A$, $\det A^{-1} = 1/\det A$
- $\det A = \prod_i \lambda_i$
- Appears in the Gaussian density: $|\Sigma|^{-1/2}$

Trace $\operatorname{tr} A = \sum_i A_{ii}$:

- Linear: $\operatorname{tr}(A + B) = \operatorname{tr} A + \operatorname{tr} B$
- Cyclic: $\operatorname{tr}(ABC) = \operatorname{tr}(CAB)$
- $\operatorname{tr} A = \sum_i \lambda_i$
- Total variance $= \operatorname{tr} \Sigma$ (PCA)



Orthogonal matrices

A pure rotation or reflection: lengths and angles do not change.

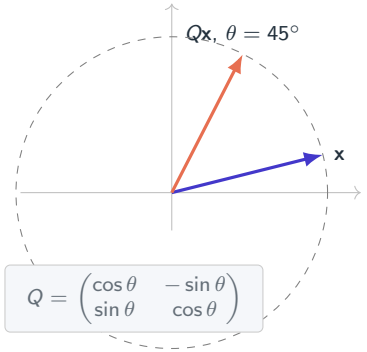
Orthogonal matrix

$Q \in \mathbb{R}^{n \times n}$ with $Q^\top Q = QQ^\top = I$, i.e. $Q^{-1} = Q^\top$. Columns form an orthonormal basis.

- Preserves lengths and angles: $\|Q\mathbf{x}\| = \|\mathbf{x}\|$, $(Q\mathbf{x})^\top (Q\mathbf{y}) = \mathbf{x}^\top \mathbf{y}$
- Rotations and reflections; $\det Q = \pm 1$

Why they matter

Eigendecomposition and SVD produce orthogonal matrices; PCA is such a rotation.



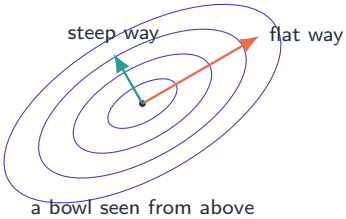
Special matrices cheat sheet

A reference table; come back when a later chapter names one of these.

Type	Definition	Property	ML example
Symmetric	$A = A^\top$	real eigenvalues, orthogonal eigenvectors	covariance, kernel, Hessian
Diagonal	$A_{ij} = 0,\ i \neq j$	trivial inverse/powers	scaling, Λ in decompositions
Positive semi-definite	$\mathbf{x}^\top A\mathbf{x} \geq 0\ \forall \mathbf{x}$	eigenvalues ≥ 0	covariance, Gram matrix $X^\top X$
Positive definite	$\mathbf{x}^\top A\mathbf{x} > 0\ \forall \mathbf{x} \neq 0$	invertible, Cholesky $A = LL^\top$	$X^\top X + \lambda I$, convex Hessian
Orthogonal	$Q^\top Q = I$	preserves norms	PCA rotation
Projection	$P^2 = P,\ P^\top = P$	idempotent	hat matrix $H = X(X^\top X)^{-1}X^\top$
Triangular	zeros above/below diagonal	solve by substitution $O(n^2)$	LU, Cholesky factors
Sparse	mostly zeros	memory/compute $\propto$ nnz	one-hot, bag-of-words

Intuition: bowls and saddles

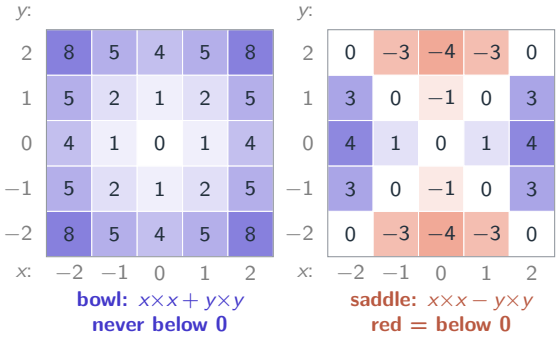
- Take a function of two numbers and look at its graph as a landscape.
- A **bowl**: wherever you drop a marble, it rolls to the single lowest point. This is what "positive definite" means: in every direction the ground goes up.
- A **saddle** (the shape of a potato crisp): it goes up one way and down the other way. A marble can roll off. This is "indefinite".
- Seen from above, the height lines of a bowl are ellipses, like an archery target that someone stretched. The long direction of the target is where the bowl is flat; the short one is where it is steep.



In one sentence

PSD means "bowl shaped"; the ellipses are the bowl's height lines, long where it is flat.

Picture: a bowl and a saddle in numbers



- Each square is a spot on the ground (x across, y up) and its number is the height. Check: $x = 2$, $y = 1$ gives $4 + 1 = 5$.
- Bowl: the numbers only go up, never below 0. A matrix (a table of numbers) that makes a bowl is called *positive semi-definite*, PSD.
- Saddle: up to 4 left or right, down to -4 up or down: not PSD.

In one sentence

PSD means bowl-shaped: the height is never below zero.

Positive semi-definite matrices

Never a negative square: $\mathbf{x}^\top A\mathbf{x} \geq 0$ always. Covariances are PSD.

PSD / PD

Symmetric A is **positive semi-definite** ($A \succeq 0$) if $\mathbf{x}^\top A\mathbf{x} \geq 0$ for all $\mathbf{x}$; **positive definite** ($A \succ 0$) if > 0 for $\mathbf{x} \neq \mathbf{0}$.

Equivalent: all eigenvalues ≥ 0 (resp. > 0).

Gram matrices are PSD

$\mathbf{x}^\top(X^\top X)\mathbf{x} = (X\mathbf{x})^\top(X\mathbf{x}) = \|X\mathbf{x}\|^2 \geq 0$. The same argument works for a covariance matrix.

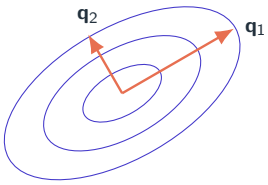
Consequences

- A twice-differentiable f is convex iff its Hessian is PSD everywhere (Section 8).
- A kernel function is valid iff every Gram matrix it produces is PSD (Chapter 7).

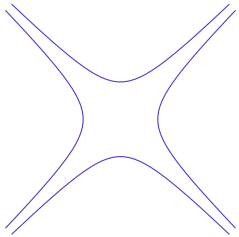
Quadratic forms and their geometry

A bowl whose contour lines are ellipses, long where the bowl is flat.

$$q(\mathbf{x}) = \mathbf{x}^\top A\mathbf{x} = \sum_{i,j} A_{ij}x_i x_j, \qquad A \text{ symmetric}$$



$A \succ 0$: ellipses



indefinite: hyperbolas (saddle)

- Level sets $\{q(\mathbf{x}) = c\}$ of a PD form are ellipsoids whose axes are the eigenvectors $\mathbf{q}_i$ and whose semi-axis lengths are $\propto 1/\sqrt{\lambda_i}$.
- Elongation $= \sqrt{\lambda_{\max}/\lambda_{\min}} = \sqrt{\kappa}$ — this is what slows gradient descent (Chapter 1, scaling slide).
- Gaussian log-density and squared loss are quadratic forms.

Worked example: the bottom of a bowl

The bottom of a bowl is where the slope is zero: one small linear system.

$$J(\mathbf{w}) = \tfrac{1}{2}\mathbf{w}^\top A\mathbf{w} - \mathbf{b}^\top \mathbf{w}, \qquad A = \begin{pmatrix} 4 & 1 \\ 1 & 2 \end{pmatrix}, \ \mathbf{b} = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$$

Step 1. Check it is a bowl: A is positive definite ($4 > 0$ and $\det A = 8 - 1 = 7 > 0$), so there is exactly one lowest point.

Step 2. Slope zero: the gradient (Section 3) is $\nabla J = A\mathbf{w} - \mathbf{b}$. Setting it to zero gives $A\mathbf{w}^* = \mathbf{b}$.

Step 3. Solve the 2×2 system $4w_1 + w_2 = 1$, $w_1 + 2w_2 = 3$:
$\mathbf{w}^* = \left(-\frac{1}{7}, \ \frac{11}{7}\right) \approx (-0.14, \ 1.57)$.

Step 4. Check: $A\mathbf{w}^* = \left(-\frac{4}{7} + \frac{11}{7}, \ -\frac{1}{7} + \frac{22}{7}\right) = (1, 3) = \mathbf{b}$ ✔

Why this matters

Least squares is exactly this bowl with $A = X^\top X$ and $\mathbf{b} = X^\top \mathbf{y}$ (Chapter 3). The equation $A\mathbf{w} = \mathbf{b}$ is then called the *normal equations*.

Matrix norms (optional)

The size of a matrix; Frobenius = length of all entries stacked up.

- **Frobenius**: $\|A\|_F = \sqrt{\sum_{ij} A_{ij}^2} = \sqrt{\mathrm{tr}(A^\top A)} = \sqrt{\sum_i \sigma_i^2}$
- **Spectral** (operator 2-norm): $\|A\|_2 = \max_{\|\mathbf{x}\|=1} \|A\mathbf{x}\| = \sigma_{\max}(A)$

Uses

- Weight decay on a weight matrix W: $\lambda \|W\|_F^2$
- Lipschitz constant of a linear layer: $\|W\|_2$ (spectral normalization in GANs)

Example

$A = \begin{pmatrix} 3 & 0 \\ 4 & 5 \end{pmatrix}$: $\|A\|_F = \sqrt{9+16+25} \approx 7.07$; largest singular value $\sqrt{45} \approx 6.71 = \|A\|_2$.

Computational costs you should know

Anything quadratic in n breaks around 10^5 samples; know your costs.

Operation	Cost	Comment
$\mathbf{x}^\top \mathbf{y}$, $\|\mathbf{x}\|$	$O(d)$	
$A\mathbf{x}$, $A \in \mathbb{R}^{m \times n}$	$O(mn)$	one prediction pass: $O(nd)$
AB, $m \times k$ times $k \times n$	$O(mkn)$	$X^\top X$: $O(nd^2)$
Solve $A\mathbf{x} = \mathbf{b}$ ($n \times n$)	$O(n^3)$	LU / Cholesky ($\frac{1}{3}n^3$ for SPD)
Inverse, determinant	$O(n^3)$	avoid when a solve suffices
Eigendecomposition (symmetric)	$O(n^3)$	PCA on $d \times d$ covariance
SVD of $n \times d$	$O(nd \min(n, d))$	truncated / randomized: $O(ndk)$
Pairwise distances, n points	$O(n^2 d)$	k-NN brute force, kernel matrix

Rule of thumb

$d = 10^3$: $d^3 = 10^9$ flops $\approx$ under a second. $n = 10^5$: $n^2 = 10^{10}$ entries $= 80$ GB in double precision — kernel methods need approximations.

Key ideas

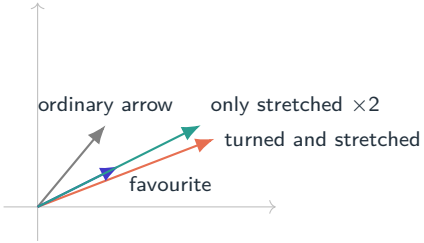
- Norms $\|\mathbf{x}\|_p$; unit balls explain ℓ_1 sparsity vs. ℓ_2 shrinkage. Cauchy–Schwarz: $|\mathbf{x}^\top \mathbf{y}| \leq \|\mathbf{x}\| \, \|\mathbf{y}\|$.
- Projection $\text{proj}_{\mathbf{u}}(\mathbf{x}) = \frac{\mathbf{x}^\top \mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u}$; least squares = projection of $\mathbf{y}$ onto $\mathcal{C}(X)$.
- Rank, invertibility, condition number $\kappa = \sigma_{\max}/\sigma_{\min}$; regularization restores invertibility.
- PSD matrices: $\mathbf{x}^\top A\mathbf{x} \geq 0$; Gram and covariance matrices are PSD; PD Hessian $\Leftrightarrow$ strictly convex.
- Quadratic forms have ellipsoidal level sets aligned with eigenvectors.
- Solve, don't invert; know the $O(\cdot)$ of what you run.

Linear Algebra II: Eigendecomposition and SVD

Section 2 — Chapter 2

Intuition: favourite directions

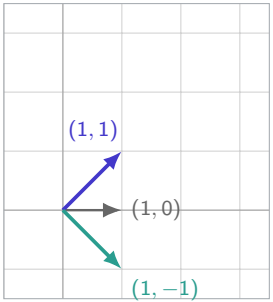
- Draw a face on a rubber sheet and pull the sheet: the face stretches, and most arrows drawn on it both turn and change length.
- A few special arrows do not turn at all; they only get longer or shorter. These are the **eigenvectors**, the sheet's favourite directions.
- How much they get longer is the **eigenvalue**: a factor 3 means three times longer, a factor 0.5 means half as long.
- Any matrix that is symmetric is just "pull the sheet along some favourite directions". Find those directions and the machine becomes easy to understand.



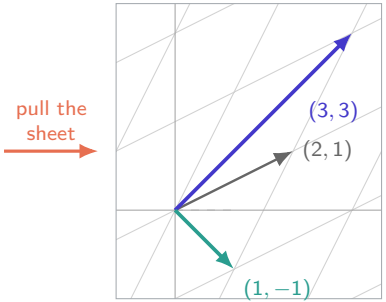
In one sentence

An eigenvector is a direction the matrix only stretches, and the eigenvalue says by how much.

Picture: arrows that do not turn



before



after the matrix $\begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$

- The matrix (a table of 4 numbers) pulls the rubber sheet; the squares become slanted boxes.
- Blue: $(1, 1)$ becomes $(3, 3)$. Same direction, 3 times longer.
- Teal: $(1, -1)$ stays $(1, -1)$. Same direction, same length: factor 1.
- Grey: $(1, 0)$ becomes $(2, 1)$. It turned, so it is not special.

In one sentence

An eigenvector is an arrow the matrix does not turn; its eigenvalue is the stretch factor.

Eigenvalues and eigenvectors

A direction the matrix only stretches, never turns; the eigenvalue is the factor.

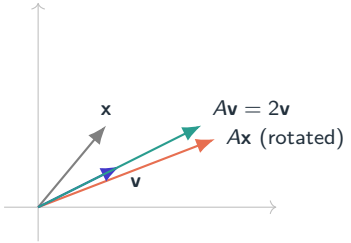
Definition

For square $A \in \mathbb{R}^{n \times n}$, $\mathbf{v} \neq \mathbf{0}$ is an **eigenvector** with **eigenvalue** λ if

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

A merely *stretches* $\mathbf{v}$ by λ without rotating it.

- Found from $\det(A - \lambda I) = 0$ (characteristic polynomial, degree n).
- $\sum_i \lambda_i = \operatorname{tr} A$, $\prod_i \lambda_i = \det A$.
- Eigenvectors are defined up to scale; normalize $\|\mathbf{v}\| = 1$.



Worked example: eigenvalues of a 2×2 matrix

Find the stretch factors, then the direction of each, then check.

$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

Step 1. Stretch factors: solve $\det(A - \lambda I) = 0$, i.e. $(2-\lambda)^2 - 1 = 0$, so $\lambda_1 = 3$ and $\lambda_2 = 1$.

Step 2. Direction for $\lambda_1 = 3$: solve $(A - 3I)\mathbf{v} = \mathbf{0}$, i.e. $-v_1 + v_2 = 0$, so $\mathbf{v}_1 = \frac{1}{\sqrt{2}}(1,1)^\top$.

Step 3. Direction for $\lambda_2 = 1$: $v_1 + v_2 = 0$, so $\mathbf{v}_2 = \frac{1}{\sqrt{2}}(1,-1)^\top$.

Step 4. Check: $\mathbf{v}_1 \perp \mathbf{v}_2$; $\operatorname{tr} A = 4 = 3 + 1$; $\det A = 3 = 3 \cdot 1$ ✔

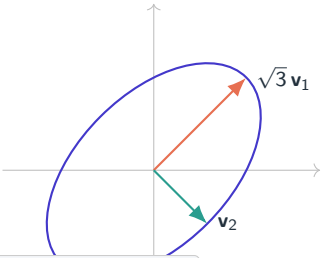
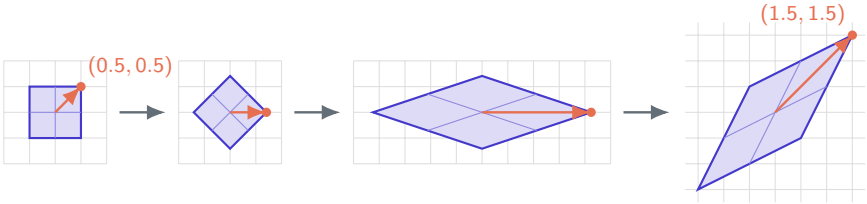


image of the unit circle under A

Picture: turn, stretch, turn back



1. start: a square with a red corner
2. turn 45°: the corner is now flat
3. stretch ×3 sideways, height unchanged
4. turn back 45°: done

- Same matrix $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ as before; it is symmetric: the top-right number equals the bottom-left one.
- The red corner $(0.5, 0.5)$ is on the favourite direction. Turn the sheet so it lies flat, stretch 3 times sideways (height unchanged), turn back.
- The matrix does all this in one go: $(0.5, 0.5)$ becomes $(1.5, 1.5)$, 3 times further out.

In one sentence

A symmetric matrix is only three easy moves: turn, stretch along the favourite directions, turn back.

Symmetric matrix = rotate to special axes, stretch, rotate back.

Spectral theorem

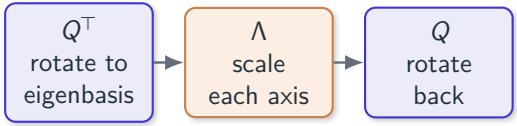
Every symmetric $A \in \mathbb{R}^{n \times n}$ can be written

$$A = Q \Lambda Q^\top = \sum_{i=1}^{n} \lambda_i \mathbf{q}_i \mathbf{q}_i^\top,$$

with $Q = [\mathbf{q}_1 \cdots \mathbf{q}_n]$ orthogonal and $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ real.

Consequences:

- $\mathbf{x}^\top A \mathbf{x} = \sum_i \lambda_i (\mathbf{q}_i^\top \mathbf{x})^2$: in the eigenbasis, a quadratic form is a weighted sum of squares.
- $A \succeq 0 \iff \lambda_i \geq 0 \ \forall i$.



$A\mathbf{x}$ = rotate, stretch, rotate back

The most-stretched direction is the top eigenvector; PCA asks this of Σ.

Rayleigh quotient

For symmetric A and a unit vector $\mathbf{x}$: $\lambda_{\min} \le \mathbf{x}^\top A\mathbf{x} \le \lambda_{\max}$, and the maximum $\lambda_{\max}$ is reached at $\mathbf{x} = \mathbf{q}_{\max}$.

Step 1. Write $\mathbf{x}$ in the eigenbasis: $\mathbf{x} = \sum_i c_i \mathbf{q}_i$ with $\sum_i c_i^2 = 1$.

Step 2. Then $\mathbf{x}^\top A\mathbf{x} = \sum_i \lambda_i c_i^2$: a weighted average of the eigenvalues.

Step 3. A weighted average can never exceed the largest value, so $\mathbf{x}^\top A\mathbf{x} \le \lambda_{\max}$, with equality when all the weight is on $\mathbf{q}_{\max}$.

This *is* PCA (Chapter 8)

The variance of the data projected on a unit direction $\mathbf{u}$ is $\mathbf{u}^\top \Sigma \mathbf{u}$. Maximizing it gives the top eigenvector of Σ; the next component is the best direction orthogonal to it, and so on.

Non-symmetric matrices (optional)

Non-symmetric matrices may rotate; luckily ML matrices are almost all symmetric.

- General $A = V\Lambda V^{-1}$ if A has n independent eigenvectors (V not orthogonal, λ possibly complex).
- Not all matrices are diagonalizable: $\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ has a single eigenvector.
- Rotation $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$: eigenvalues $\pm i$ (no real eigenvector).

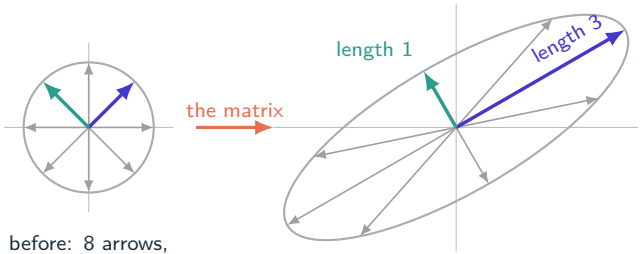
In ML you mostly meet symmetric matrices

covariance, Gram/kernel, Hessian, Laplacian (spectral clustering), $X^\top X$ — all symmetric, so the spectral theorem applies.

Power iteration

$\mathbf{v} \leftarrow A\mathbf{v}/\|A\mathbf{v}\|$ converges to $\mathbf{q}_{\max}$ at rate $|\lambda_2/\lambda_1|^k$. Basis of PageRank and of large-scale PCA.

Picture: circle in, ellipse out



before: 8 arrows, all of length 1

after: the tips lie on an ellipse

- 8 arrows of length 1: their tips make a circle. Push them all through one matrix: the tips now make an ellipse, a squashed circle.
- Blue became the longest: 3 times longer. Teal the shortest: still length 1. The others are in between.
- Blue and teal turned a little, but they still meet at a right angle.

In one sentence

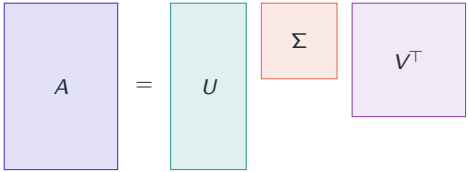
Any matrix turns a circle into an ellipse; its half-widths (here 3 and 1) are the singular values.

Rotation, stretch, rotation, for any rectangular matrix.

SVD

Every $A \in \mathbb{R}^{m \times n}$ factorizes as $A = U\Sigma V^\top$ with U, V orthogonal and Σ diagonal, $\sigma_1 \geq \cdots \geq \sigma_r > 0$, $r = \operatorname{rank} A$.

- $\mathbf{u}_i$: left singular vectors (basis of $\mathcal{C}(A)$)
- $\mathbf{v}_i$: right singular vectors (basis of $\mathcal{C}(A^\top)$)



$m \times n = (m \times r)(r \times r)(r \times n)$

SVD and eigendecomposition are cousins

Singular values of A = square roots of the eigenvalues of $A^\top A$.

$$A^\top A = V\Sigma^\top \Sigma V^\top = V\,\mathrm{diag}(\sigma_i^2)V^\top, \qquad AA^\top = U\Sigma\Sigma^\top U^\top = U\,\mathrm{diag}(\sigma_i^2)U^\top$$

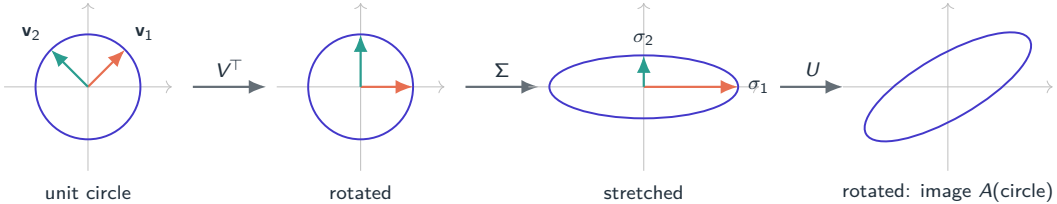
- Right singular vectors = eigenvectors of $A^\top A$; $\sigma_i = \sqrt{\lambda_i(A^\top A)}$.
- Left singular vectors = eigenvectors of $AA^\top$.
- For symmetric PSD A: SVD = eigendecomposition ($\sigma_i = \lambda_i$).

PCA via SVD of the data

Centred $X \in \mathbb{R}^{n\times d}$: $\Sigma = \frac{1}{n}X^\top X = V\,\mathrm{diag}(\sigma_i^2/n)V^\top$.
Principal directions = right singular vectors of X; scores = $XV = U\Sigma$.
Computing the SVD of X is more accurate than forming $X^\top X$.

Geometry of the SVD

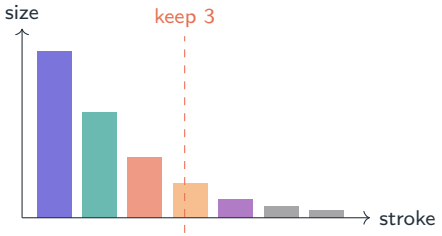
Any matrix turns the unit circle into an ellipse; the SVD reads off its axes.



- Any linear map = rotation ∘ axis-aligned scaling ∘ rotation. Singular values = lengths of the ellipse's semi-axes.

Intuition: the biggest brush strokes

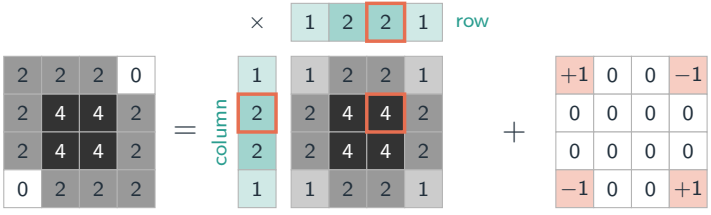
- Imagine painting a photo with brush strokes, biggest first: the sky, then the mountain, then the house, then the windows, then tiny details.
- The SVD does exactly this for a table of numbers: it splits it into "strokes" (rank-one pieces), each with a size (its singular value), from the biggest to the smallest.
- Keep only the first k strokes and you get a smaller description that still looks almost right. The strokes you dropped are the error, and they were small.
- Image compression, recommendations and PCA all do this: keep the big patterns, forget the noise.



In one sentence

The SVD sorts a matrix into patterns from biggest to smallest; keeping the first few is compression.

Picture: a tiny image in one stroke



the picture:
16 numbers

one stroke: column × row
8 numbers; red: $2 \times 2 = 4$

what is lost:
only 4 corners, by 1

- A tiny grey picture is just a table of 16 numbers: 0 is white, 4 is dark.
- One stroke is one column of 4 numbers times one row of 4 numbers: only $4 + 4 = 8$ numbers. Each cell is its column number times its row number; the red cell is $2 \times 2 = 4$.
- Not perfect: the 4 corners are off by 1. A small loss for half the numbers; the SVD (next slide) is the recipe that finds this best stroke.

In one sentence

One column times one row is one stroke; this picture is almost one stroke, so 8 numbers nearly replace 16.

Low-rank approximation (Eckart–Young theorem)

Keep the k largest stretches: the best compression with k patterns.

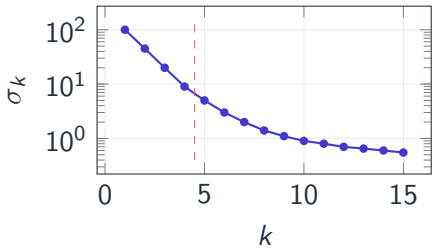
Truncated SVD

$A_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ is the best rank-k approximation of A; the error is $\|A - A_k\|_F = \sqrt{\sum_{i>k} \sigma_i^2}$ (the dropped stretches).

Uses:

- PCA / dimensionality reduction ($k \ll d$)
- Collaborative filtering (users $\times$ items)

typical spectrum: fast decay



Worked example: rank-1 approximation

Compress a 3×2 matrix into one column times one row; measure the loss.

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 3 \\ 1 & 1 \end{pmatrix}$$

Step 1. Form $A^\top A = \begin{pmatrix} 14 & 13 \\ 13 & 14 \end{pmatrix}$; its eigenvalues are 27 and 1 (same trick as the previous example).

Step 2. Singular values: $\sigma_1 = \sqrt{27} \approx 5.20$, $\sigma_2 = \sqrt{1} = 1$.

Step 3. Top right direction: $\mathbf{v}_1 = \frac{1}{\sqrt{2}}(1,1)^\top$. Top left direction: $\mathbf{u}_1 = A\mathbf{v}_1/\sigma_1 = \frac{1}{\sqrt{54}}(5,5,2)^\top$.

Step 4. Best rank-1 version:

$$A_1 = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^\top = \begin{pmatrix} 2.5 & 2.5 \\ 2.5 & 2.5 \\ 1 & 1 \end{pmatrix}.$$

Step 5. What is lost:

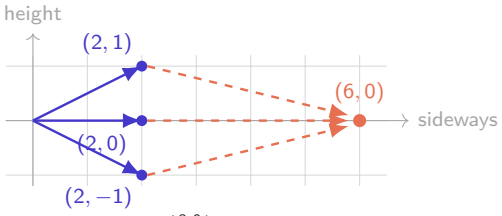
$$A - A_1 = \begin{pmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \\ 0 & 0 \end{pmatrix}, \text{ whose}$$

Frobenius norm is $1 = \sigma_2$ ✔

Interpretation

A_1 says "every row is proportional to $(1,1)$" and stores 5 numbers instead of 6. It keeps $27/28 \approx 96\%$ of the energy $\sum_i \sigma_i^2$.

Picture: a best-effort undo



- This matrix stretches 3 times sideways and flattens every height to 0: $(2, 1)$, $(2, 0)$ and $(2, -1)$ all land on $(6, 0)$.
- A true undo (an inverse) would have to send $(6, 0)$ back to three different points. Impossible: the height is lost.
- The pseudo-inverse is the best-effort undo: divide by 3 sideways, and answer 0 for the lost height. It returns $(2, 0)$, the shortest of the three arrows.

In one sentence

The pseudo-inverse undoes every stretch it can and answers 0 for what was flattened.

Pseudo-inverse and least squares for any X

A best-effort inverse: invert the used directions, ignore the rest.

Moore–Penrose pseudo-inverse

From the thin SVD $A = U\Sigma V^\top$: $A^+ = V\Sigma^{-1}U^\top$ (invert only non-zero σ_i).

- If A has full column rank: $A^+ = (A^\top A)^{-1}A^\top$.
- If A is square invertible: $A^+ = A^{-1}$.
- $\mathbf{x}^* = A^+\mathbf{b}$ is the *minimum-norm least-squares* solution of $A\mathbf{x} \approx \mathbf{b}$.

Least squares, three ways

1. Normal equations: $(X^\top X)\hat{\mathbf{w}} = X^\top \mathbf{y}$ — $O(nd^2)$, needs full rank
2. QR: $X = QR$, $R\hat{\mathbf{w}} = Q^\top \mathbf{y}$ — stabler
3. SVD: $\hat{\mathbf{w}} = V\Sigma^{-1}U^\top \mathbf{y}$ — handles rank deficiency, costliest

`np.linalg.lstsq` uses SVD.

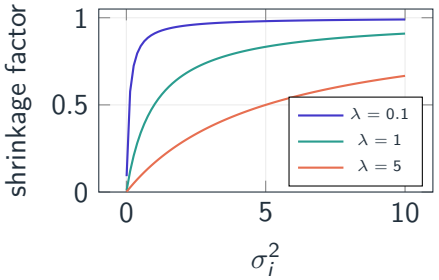
Ridge regression through the SVD lens (optional)

Ridge keeps strong directions and shrinks weak, noisy ones towards zero.

With $X = U\Sigma V^\top$ (thin), the ridge solution $\hat{\mathbf{w}}_\lambda = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$ becomes

$$\hat{\mathbf{w}}_\lambda = V \operatorname{diag}\Big(\frac{\sigma_i}{\sigma_i^2 + \lambda}\Big) U^\top \mathbf{y}, \qquad \hat{\mathbf{y}}_\lambda = X\hat{\mathbf{w}}_\lambda = \sum_i \underbrace{\frac{\sigma_i^2}{\sigma_i^2 + \lambda}}_{\text{shrinkage} \in (0,1)} \mathbf{u}_i(\mathbf{u}_i^\top \mathbf{y}).$$

- Directions with large σ_i (high variance) are barely shrunk; directions with small σ_i (noisy, ill-conditioned) are shrunk strongly.
- Effective degrees of freedom: $\mathrm{df}(\lambda) = \sum_i \frac{\sigma_i^2}{\sigma_i^2 + \lambda}$.



Matrix factorizations used in ML

Split a matrix into simpler pieces; each factorization fits one job.

Factorization	Form	Requires	Used for
LU	$A = LU$	square	general linear solves
Cholesky	$A = LL^\top$	symmetric PD	fast solves with $X^\top X + \lambda I$, Gaussian sampling ($\mathbf{x} = \boldsymbol{\mu} + L\mathbf{z}$), GPs
QR	$A = QR$	any	stable least squares, Gram–Schmidt
Eigen	$A = Q\Lambda Q^\top$	symmetric	PCA, spectral clustering, Hessian analysis
SVD	$A = U\Sigma V^\top$	any	PCA, low rank, pseudo-inverse, LSA
NMF	$A \approx WH,\ W, H \geq 0$	non-negative	topic models, parts-based decompositions

Sampling from $\mathcal{N}(\mu, \Sigma)$ with Cholesky

$\Sigma = LL^\top$, $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, I) \Rightarrow \mathbf{x} = \boldsymbol{\mu} + L\mathbf{z}$ has covariance $L\,I\,L^\top = \Sigma$.

Numerical stability (optional)

Computers keep 16 digits; ill-conditioning eats them, $X^\top X$ twice as fast.

- Floating point has ≈ 16 significant digits (double).
- Solving $A\mathbf{x} = \mathbf{b}$ loses about $\log_{10} \kappa(A)$ digits.
- Forming $X^\top X$ *squares* the condition number: $\kappa(X^\top X) = \kappa(X)^2$.
- Hence QR/SVD on X directly are preferred for ill-conditioned problems.

Example

Two features with correlation 0.9999 $\Rightarrow$ $\kappa(X) \approx 140$, $\kappa(X^\top X) \approx 2 \cdot 10^4$; with single precision (7 digits) the normal equations give garbage coefficients, though predictions may still look fine.

Practical rules

Standardize features; add λI; prefer library solvers (`lstsq`, `solve`, `cho_solve`) to hand-written inverses.

Tensors (brief)

An array with more than two indices, like a stack of images.

- Order-0: scalar; order-1: vector; order-2: matrix; order-k: k-dimensional array.
- A colour image: $H \times W \times 3$; a batch of images: $B \times H \times W \times 3$.
- Deep learning frameworks operate on tensors with *broadcasting* and *einsum*-style contractions.



Einstein summation

$C_{ij} = \sum_k A_{ik} B_{kj} \rightarrow$ `np.einsum("ik,kj->ij", A, B)`

Linear algebra in scikit-learn / NumPy: quick reference

Every object of this section is one NumPy call.

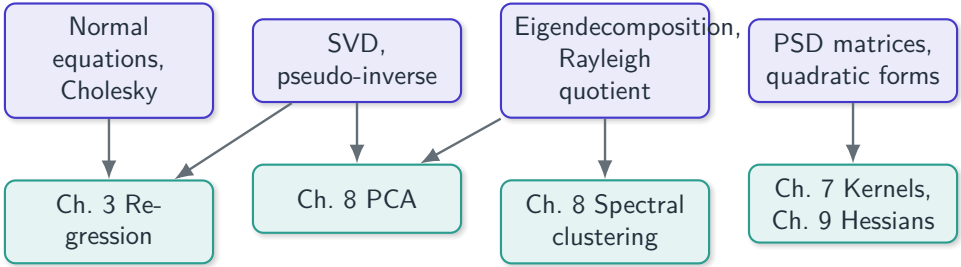
Python

```
import numpy as np
X = np.random.randn(100, 5)
Xc = X - X.mean(axis=0)                        # centre columns
Sigma = Xc.T @ Xc / len(X)                     # covariance (d x d)
lam, Q = np.linalg.eigh(Sigma)                  # symmetric eigendecomposition (ascending)
U, s, Vt = np.linalg.svd(Xc, full_matrices=False) # thin SVD
assert np.allclose(s**2 / len(X), lam[::-1])     # sigma_i^2 / n = eigenvalues
w = np.linalg.solve(X.T @ X + 0.1*np.eye(5), X.T @ y) # ridge via solve
w2, *_ = np.linalg.lstsq(X, y, rcond=None)      # least squares via SVD
L = np.linalg.cholesky(Sigma + 1e-6*np.eye(5))    # Cholesky factor
kappa = np.linalg.cond(X)                       # condition number
```

- `eigh` for symmetric matrices (faster, real output); `eig` for general.

Where each tool is used later

Each tool has a home chapter; this map says when you will need it.



Key formulas

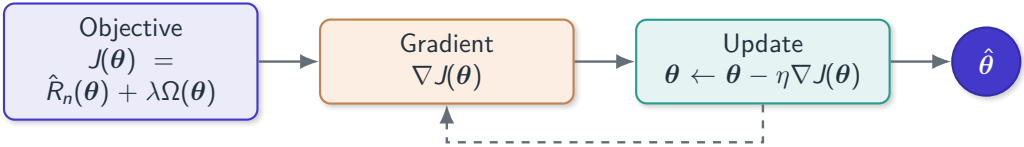
- $A\mathbf{v} = \lambda\mathbf{v}$; symmetric $A = Q\Lambda Q^\top$ with orthogonal Q; $\max_{\|\mathbf{x}\|=1} \mathbf{x}^\top A\mathbf{x} = \lambda_{\max}$.
- SVD $A = U\Sigma V^\top$; $\sigma_i = \sqrt{\lambda_i(A^\top A)}$; any linear map = rotate, scale, rotate.
- Eckart–Young: $A_k = \sum_{i \le k} \sigma_i \mathbf{u}_i \mathbf{v}_i^\top$ is the best rank-k approximation.
- Pseudo-inverse $A^+ = V\Sigma^{-1}U^\top$; ridge shrinks direction i by $\sigma_i^2/(\sigma_i^2 + \lambda)$.
- $\kappa(X^\top X) = \kappa(X)^2$: use QR/SVD or regularize.

Multivariate Calculus and Matrix ~~Calculus~~

Section 3 — Chapter 2

Why calculus? Learning is optimization

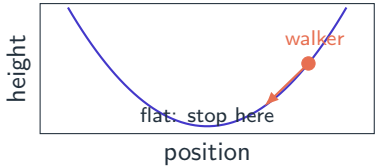
Training = walking downhill on the loss; the gradient points the way.



- Almost every training algorithm needs the gradient of a scalar loss with respect to a vector or matrix of parameters.
- Second derivatives (the Hessian) tell us about curvature, convexity, and how fast gradient descent converges.
- Backpropagation (Chapter 9) is nothing but the multivariate chain rule organized efficiently.

Intuition: the slope under your feet

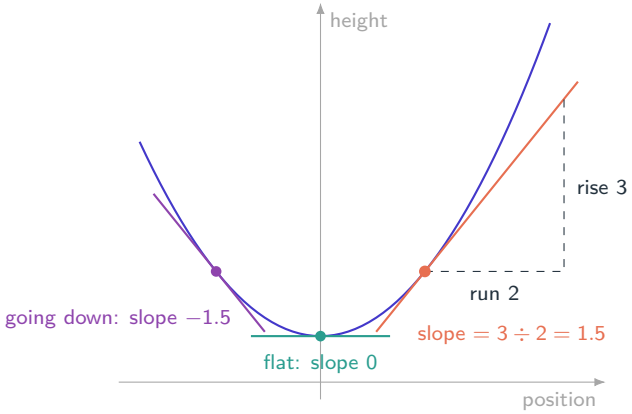
- You stand on a hill in thick fog: you cannot see the valley, but you feel the slope under your feet.
- The **gradient** is the steepest-uphill direction and how steep it is; walk the other way to go down fastest.
- **Gradient descent**: feel the slope, step downhill, repeat. The bottom is where the ground is flat (gradient zero).
- **Step size**: tiny steps take forever; giant steps jump over the valley.
- **Curvature** (the Hessian): narrow canyon versus wide meadow; in a canyon, take small steps.



In one sentence

The gradient is the steepest-uphill direction; learning means walking the other way, step by step.

Picture: slope = rise over run

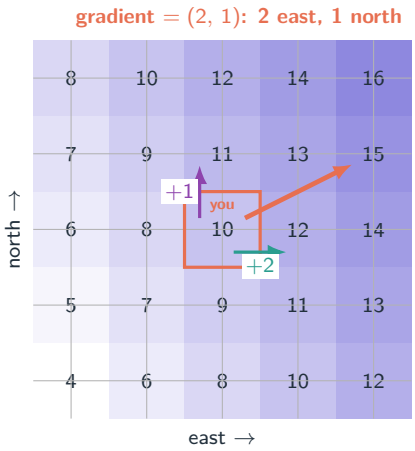


- The **slope** at a point says how steep the curve is there. Draw the straight line that just touches the curve at that point (its tangent) and measure how steep that line is.
- Orange point: go 2 to the right, the line rises 3. Slope $= 3 \div 2 = 1.5$.
- Green point: the line is flat, slope 0. That is the bottom. Purple point: the line goes down, so the slope is negative: -1.5.

In one sentence

The derivative of a curve at a point is just its slope there: rise divided by run.

Picture: gradient from a height map



- A hill drawn as a table of heights (in metres). You stand on the 10.
- Step one cell east: $12 - 10 = +2$. Step one cell north: $11 - 10 = +1$. Each number is the slope in one direction: a **partial derivative**.
- Put the two slopes together as one arrow, (2, 1): two cells east, one cell north. This arrow is the **gradient**. It points where the hill climbs fastest, straight towards the 15.

In one sentence

The gradient is the list of slopes, one per direction, drawn as one arrow that points straight uphill.

The slopes along each axis, stacked into one vector; it points uphill.

Gradient

For $f : \mathbb{R}^d \to \mathbb{R}$, the gradient stacks the d partial derivatives (the slope of f when only x_j moves):

$$\nabla f(\mathbf{x}) = \Big(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d}\Big)^\top \in \mathbb{R}^d.$$

Example

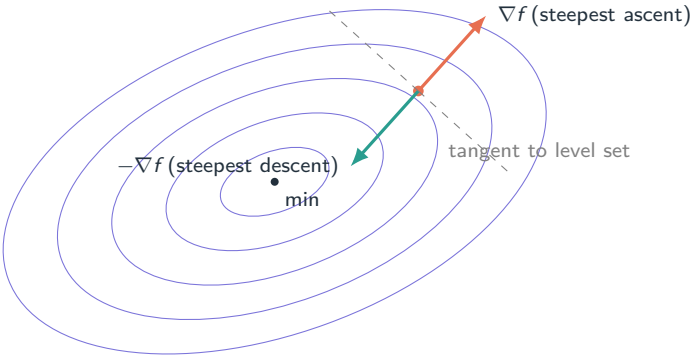
$f(x_1, x_2) = x_1^2 + 3x_1x_2 + \sin x_2$
$\partial f/\partial x_1 = 2x_1 + 3x_2$
$\partial f/\partial x_2 = 3x_1 + \cos x_2$
$\nabla f(1, 0) = (2,\ 4)^\top$

Properties:

- $\nabla f(\mathbf{x})$ points in the direction of *steepest ascent*.
- $\nabla f(\mathbf{x}) \perp$ level set $\{f = c\}$ through $\mathbf{x}$.
- At a local min/max (interior): $\nabla f = \mathbf{0}$ (stationary point).

The gradient is perpendicular to level sets

Steepest ascent crosses the contour lines at right angles.



- Directional derivative along unit $\mathbf{u}$: $D_{\mathbf{u}} f(\mathbf{x}) = \nabla f(\mathbf{x})^\top \mathbf{u} = \|\nabla f\| \cos\theta$, maximal when $\mathbf{u} \parallel \nabla f$ (Cauchy–Schwarz).
- Gradient descent walks perpendicular to the contours — which is why elongated contours cause zig-zagging.

One gradient per output, stacked as rows.

Jacobian

For $\mathbf{f}: \mathbb{R}^n \to \mathbb{R}^m$, $J_{\mathbf{f}}(\mathbf{x}) \in \mathbb{R}^{m \times n}$ with $(J_{\mathbf{f}})_{ij} = \dfrac{\partial f_i}{\partial x_j}$. Row i is $\nabla f_i^\top$.

- Best linear approximation: $\mathbf{f}(\mathbf{x} + \boldsymbol{\delta}) \approx \mathbf{f}(\mathbf{x}) + J_{\mathbf{f}}(\mathbf{x})\boldsymbol{\delta}$.
- Linear map $\mathbf{f}(\mathbf{x}) = A\mathbf{x}$: $J = A$.
- Elementwise $\mathbf{f}(\mathbf{x}) = (g(x_1), \ldots, g(x_n))$: $J = \mathrm{diag}(g'(x_i))$ — activations in neural nets.

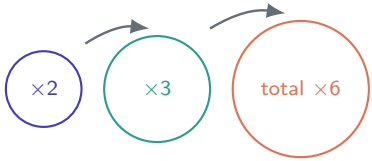
Softmax Jacobian

$s_i(\mathbf{z}) = \dfrac{e^{z_i}}{\sum_k e^{z_k}}$:

$$\frac{\partial s_i}{\partial z_j} = s_i(\delta_{ij} - s_j), \quad J = \mathrm{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top$$

Intuition: a chain of gears

- Three gears in a row. Turning the first by one notch turns the second by 2 notches, and each notch of the second turns the third by 3 notches.
- So one notch at the start moves the last gear by $2 \times 3 = 6$ notches. Effects along a chain *multiply*.
- A neural network is a chain of simple operations. "If I nudge this weight, how much does the error change?" is answered by multiplying the local effects along the chain.
- **Backpropagation** just reads the chain from the end: start at the error, multiply by each local effect going backwards, and you get the answer for every weight in one pass.

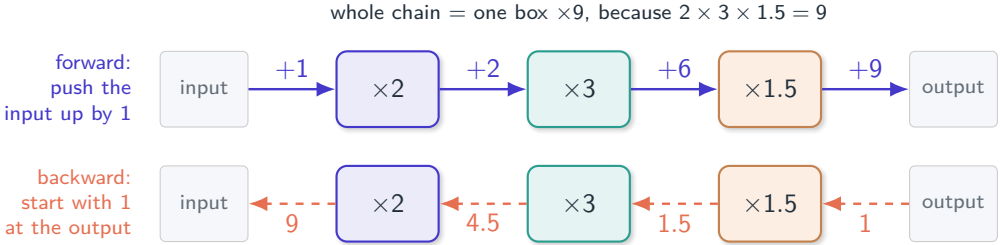


one notch in → six notches out

In one sentence

Along a chain, small effects multiply; backpropagation multiplies them from the end backwards.

Picture: chain rule as three boxes



- Each box multiplies a small change by its number: that number is its slope. Push the input up by 1: it becomes 2, then 6, then 9.
- So the whole chain acts like one box ×9: slopes along a chain **multiply**.
- Backward: start with 1 at the output, multiply by each box going left: 1.5, 4.5, 9. Same answer, and every middle arrow gets its number too.

In one sentence

The slope of a chain is the slopes of its boxes multiplied together; doing this from the end backwards is called back-propagation.

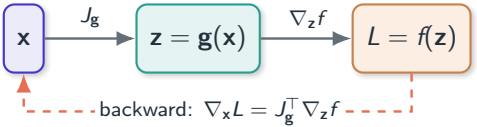
Multiply the derivatives of the links; backprop does it from the end.

Chain rule (vector form)

If $\mathbf{z} = \mathbf{g}(\mathbf{x})$ and $\mathbf{y} = \mathbf{f}(\mathbf{z})$, then

$$J_{\mathbf{f}\circ\mathbf{g}}(\mathbf{x}) = J_{\mathbf{f}}(\mathbf{g}(\mathbf{x}))\, J_{\mathbf{g}}(\mathbf{x}) \quad \text{(matrix product)}.$$

For a scalar output $L = f(\mathbf{g}(\mathbf{x}))$: $\nabla_{\mathbf{x}} L = J_{\mathbf{g}}(\mathbf{x})^{\top} \nabla_{\mathbf{z}} f$.



- Gradients flow *backwards* through transposed Jacobians: this is backpropagation.
- For a chain of K layers, $\nabla_{\mathbf{x}} L = J_1^{\top} J_2^{\top} \cdots J_K^{\top} \nabla L$: products of many Jacobians can vanish or explode (Chapter 9).

Worked example: chain rule on a tiny network

One neuron: differentiate each link, then multiply along the chain.

$L = \frac{1}{2}(y-\hat{y})^2$, $\hat{y} = \sigma(z)$, $z = wx + b$, with $\sigma(z) = \dfrac{1}{1+e^{-z}}$ and $\sigma'(z) = \sigma(z)(1-\sigma(z))$.

Step 1. Local derivative of the loss:
$\partial L/\partial \hat{y} = -(y-\hat{y})$.

Step 2. Local derivative of the activation:
$\partial \hat{y}/\partial z = \hat{y}(1-\hat{y})$.

Step 3. Local derivatives of the linear part:
$\partial z/\partial w = x$, $\partial z/\partial b = 1$.

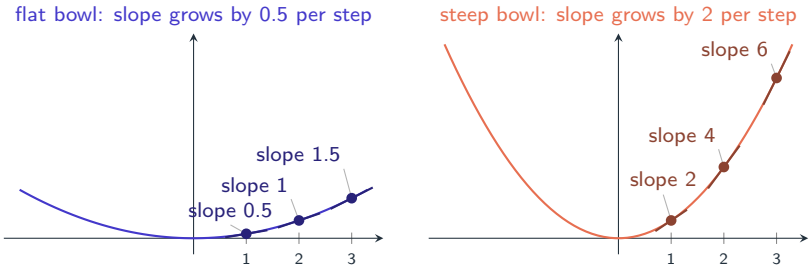
Step 4. Multiply along the chain:

$$\frac{\partial L}{\partial w} = -(y-\hat{y})\,\hat{y}(1-\hat{y})\,x, \qquad \frac{\partial L}{\partial b} = -(y-\hat{y})\,\hat{y}(1-\hat{y})$$

Numbers

$x = 2$, $y = 1$, $w = 0.5$, $b = -1$: $z = 0$, $\hat{y} = 0.5$.
Step 1: -0.5. Step 2: 0.25. Step 3: 2 and 1.
Step 4: $\partial L/\partial w = -0.5 \cdot 0.25 \cdot 2 = -0.25$, $\partial L/\partial b = -0.125$.
One gradient step ($\eta = 1$): $w \leftarrow 0.75$, $b \leftarrow -0.875$; now $z = 0.625$, $\hat{y} = 0.65$: closer to $y = 1$ ✔

Picture: how fast the slope changes



Same scale in both pictures; the short dark pieces show the slope at positions 1, 2 and 3.

- Walk one step to the right. Flat bowl: the slope grows by 0.5 each step (0.5, 1, 1.5). Steep bowl: it grows by 2 each step (2, 4, 6).
- This growth of the slope is the **curvature**. The Hessian is the curvature in every direction, written as a table of numbers.

In one sentence

Slope says how steep you are now; curvature says how fast the slope changes as you walk.

Curvature: how fast the slope changes. Positive everywhere means a bowl.

Hessian

For $f : \mathbb{R}^d \to \mathbb{R}$ twice differentiable, $H_f(\mathbf{x}) \in \mathbb{R}^{d \times d}$ with $(H_f)_{ij} = \dfrac{\partial^2 f}{\partial x_i \partial x_j}$; it is symmetric.

Second-order Taylor expansion

$$f(\mathbf{x} + \boldsymbol{\delta}) \approx f(\mathbf{x}) + \nabla f^\top \boldsymbol{\delta} + \tfrac{1}{2} \boldsymbol{\delta}^\top H \boldsymbol{\delta}$$

Classification of stationary points ($\nabla f = \mathbf{0}$):

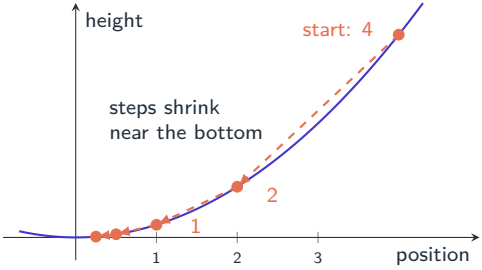
- $H \succ 0$: local minimum
- $H \prec 0$: local maximum
- indefinite: saddle point

Example

$f(x_1, x_2) = x_1^2 - x_2^2$: $\nabla f = (2x_1, -2x_2)$, $H = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$: saddle at origin.

Squared loss $\frac{1}{n} \|X\mathbf{w} - \mathbf{y}\|^2$: $H = \frac{2}{n} X^\top X \succeq 0$, a bowl.

Picture: five steps down a bowl



step	position	slope	move $= -0.25 \times$ slope
0	4	8	-2
1	2	4	-1
2	1	2	-0.5
3	0.5	1	-0.25
4	0.25	0.5	-0.125

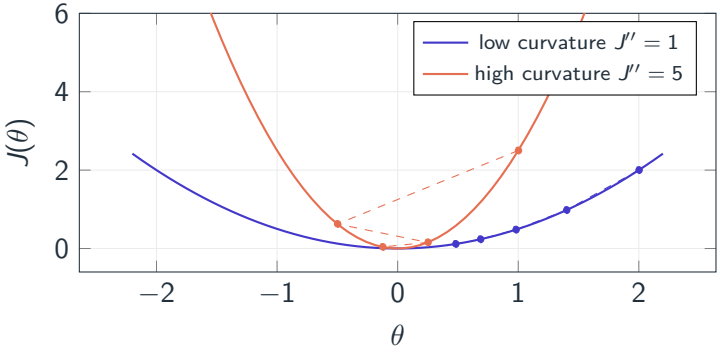
- Rule at every step: read the slope, move the *other* way by a quarter of it. First step: $4 - 0.25 \times 8 = 2$.
- Here the slope is twice the position (the steep bowl from before). So each move halves the distance to the bottom: 4, 2, 1, 0.5, 0.25, 0.125.
- Near the bottom the slope is small, so the steps shrink by themselves.

In one sentence

Measure the slope, step the other way, repeat: that is gradient descent, and it slows down by itself as the ground gets flat.

Curvature and step size: a 1-D preview

Steep bowl: small steps or you overshoot. Flat bowl: the same step crawls.



- GD on $J = \frac{c}{2}\theta^2$: $\theta_{t+1} = (1 - \eta c)\theta_t$. Converges iff $\eta < 2/c$; fastest at $\eta = 1/c$.
- Same $\eta = 0.3$: slow on the flat curve, oscillating on the steep one. In d dimensions, $c \to$ eigenvalues of H: the step must suit the *largest* λ, but progress along small-λ directions is slow $\Rightarrow$ the condition number governs speed.

Matrix calculus: conventions

A gradient has the same shape as the thing you differentiate with respect to.

Denominator layout (used in this course)

The derivative of a scalar w.r.t. a vector $\mathbf{x} \in \mathbb{R}^d$ is a column vector $\nabla_{\mathbf{x}} f \in \mathbb{R}^d$; w.r.t. a matrix $W \in \mathbb{R}^{m \times n}$ it is a matrix $\nabla_W f \in \mathbb{R}^{m \times n}$ with the same shape — so that a gradient step $W \leftarrow W - \eta \nabla_W f$ type-checks.

- Strategy for deriving gradients: (1) write the scalar as a sum of products or with traces, (2) differentiate w.r.t. one entry, (3) reassemble into matrix form, *or* use the small table on the next slide.
- Always check shapes: $\nabla_{\mathbf{x}} f$ must have the shape of $\mathbf{x}$.
- Sanity check numerically: $\dfrac{f(\mathbf{x} + h\mathbf{e}_j) - f(\mathbf{x} - h\mathbf{e}_j)}{2h} \approx (\nabla f)_j$ with $h = 10^{-5}$ (*gradient checking*).

Matrix calculus: six identities

Six rules plus the chain rule cover every gradient in this course.

f	gradient	where it is used
$\mathbf{a}^\top \mathbf{x}$	$\mathbf{a}$	linear terms
$\mathbf{x}^\top A\mathbf{x}$ (A symmetric)	$2A\mathbf{x}$	quadratic terms, Gaussians
$\|\mathbf{x}\|_2^2$	$2\mathbf{x}$	ℓ_2 regularization
$\|A\mathbf{x} - \mathbf{b}\|_2^2$	$2A^\top(A\mathbf{x} - \mathbf{b})$	least squares
$\mathbf{a}^\top W\mathbf{b}$	$\nabla_W = \mathbf{a}\mathbf{b}^\top$	gradient of a linear layer
$\|W\|_F^2$	$\nabla_W = 2W$	weight decay

How to read them

Each rule is the one-dimensional rule you know ($\frac{d}{dx}ax = a$, $\frac{d}{dx}ax^2 = 2ax$) with vectors in place of numbers. When in doubt, differentiate one entry at a time and reassemble, as on the next slide.

Derivation: $\nabla_{\mathbf{x}}(\mathbf{x}^\top A\mathbf{x})$

Differentiate one coordinate at a time, then stack the answers.

Step 1. Write the quadratic form as a double sum: $f(\mathbf{x}) = \sum_i \sum_j A_{ij} x_i x_j$.

Step 2. Differentiate with respect to one coordinate x_k. Only the terms containing x_k survive (those with $i = k$ or $j = k$):

$$\frac{\partial f}{\partial x_k} = \sum_j A_{kj} x_j + \sum_i A_{ik} x_i = (A\mathbf{x})_k + (A^\top \mathbf{x})_k.$$

Step 3. Stack the d coordinates: $\nabla f = (A + A^\top)\mathbf{x}$, which is $2A\mathbf{x}$ when A is symmetric.

Check with $d = 2$

$A = \begin{pmatrix} 1 & 2 \\ 0 & 3 \end{pmatrix}$, $f = x_1^2 + 2x_1x_2 + 3x_2^2$.

Direct: $\partial_1 f = 2x_1 + 2x_2$, $\partial_2 f = 2x_1 + 6x_2$.

Formula: $(A + A^\top)\mathbf{x} = \begin{pmatrix} 2 & 2 \\ 2 & 6 \end{pmatrix}\mathbf{x}$ ✔

Bonus: the Hessian

Differentiating once more gives $H = A + A^\top$, a constant: a quadratic form has the same curvature everywhere.

One table rule, gradient set to zero: the normal equations appear.

Step 1. The objective: $J(\mathbf{w}) = \frac{1}{2n}\|X\mathbf{w} - \mathbf{y}\|^2$.

Step 2. Apply the least-squares rule of the table ($\nabla \|A\mathbf{x} - \mathbf{b}\|^2 = 2A^\top(A\mathbf{x} - \mathbf{b})$) with $A = X$:

$$\nabla J(\mathbf{w}) = \tfrac{1}{2n} \cdot 2X^\top(X\mathbf{w} - \mathbf{y}) = \tfrac{1}{n}X^\top(X\mathbf{w} - \mathbf{y}).$$

Step 3. Set the gradient to zero: $X^\top X\mathbf{w} = X^\top\mathbf{y}$. These are the *normal equations*.

Step 4. Curvature: $H = \frac{1}{n}X^\top X$ is PSD, so J is a bowl; the solution is unique when X has full column rank.

Reading the gradient

$\nabla J = \frac{1}{n}\sum_i(\mathbf{x}_i^\top\mathbf{w} - y_i)\mathbf{x}_i$: each sample pushes $\mathbf{w}$ along its own features, in proportion to its error. Stochastic gradient descent uses one such push at a time.

Derivation: gradient of the logistic loss

Loss derivative times sigmoid derivative: again "error times features".

Binary labels $y_i \in \{0, 1\}$, probabilities $p_i = \sigma(\mathbf{w}^\top \mathbf{x}_i)$, loss $J(\mathbf{w}) = -\frac{1}{n}\sum_i \big[y_i \log p_i + (1 - y_i)\log(1 - p_i)\big]$.

Step 1. Derivative of the loss with respect to one probability:

$$\frac{\partial J}{\partial p_i} = -\frac{1}{n}\left(\frac{y_i}{p_i} - \frac{1 - y_i}{1 - p_i}\right) = -\frac{1}{n} \cdot \frac{y_i - p_i}{p_i(1 - p_i)}.$$

Step 2. Derivative of the sigmoid: $\dfrac{\partial p_i}{\partial \mathbf{w}} = p_i(1 - p_i)\,\mathbf{x}_i$ (because $\sigma' = \sigma(1 - \sigma)$).

Step 3. Multiply and sum over samples; the factor $p_i(1 - p_i)$ cancels:

$$\nabla J(\mathbf{w}) = \frac{1}{n}\sum_i (p_i - y_i)\,\mathbf{x}_i = \frac{1}{n} X^\top (\mathbf{p} - \mathbf{y}).$$

Remarkable

Same form as least squares, residual times features, with p_i in place of the linear prediction. Its Hessian is PSD: logistic regression is a bowl (Chapter 4).

Gradient w.r.t. W = error signal times input; backprop rests on this.

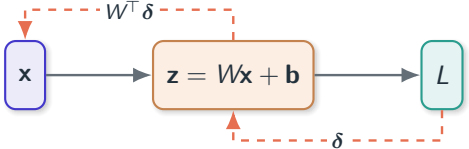
Suppose the loss L depends on $\mathbf{z} = W\mathbf{x} + \mathbf{b}$ and we already know the error signal $\boldsymbol{\delta} = \nabla_{\mathbf{z}} L \in \mathbb{R}^m$.

Step 1. One entry W_{ij} only affects z_i, and $\partial z_i / \partial W_{ij} = x_j$.

Step 2. Chain rule: $\partial L / \partial W_{ij} = \delta_i x_j$.

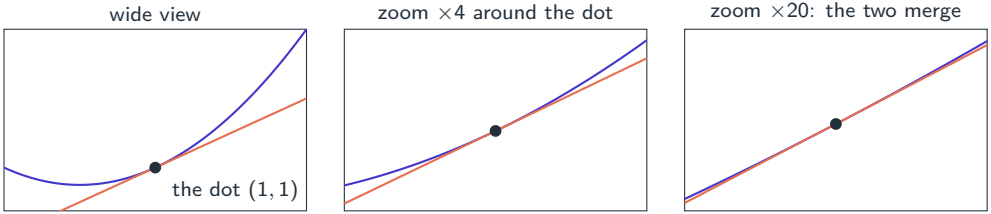
Step 3. Stack all entries: $\nabla_W L = \boldsymbol{\delta}\, \mathbf{x}^\top$ (an $m \times n$ outer product). Likewise $\nabla_{\mathbf{b}} L = \boldsymbol{\delta}$.

Step 4. Pass the error to the previous layer: $\nabla_{\mathbf{x}} L = W^\top \boldsymbol{\delta}$.



$\nabla_W L = \boldsymbol{\delta}\mathbf{x}^\top$, $\nabla_{\mathbf{b}} L = \boldsymbol{\delta}$; for a mini-batch, sum over samples.

Picture: zoom in, a curve is a line



- Blue: the curve (height = position squared). Orange: the straight line that just touches it at the dot (its tangent), slope 2.
- Zoom in on the dot: the two become hard to tell apart. Ten times closer, the gap is one hundred times smaller.
- Near a point, the slope alone predicts the curve well, so a small step against the slope always goes downhill.

position	distance from dot	curve	line	gap
1.5	0.5	2.25	2	0.25
1.1	0.1	1.21	1.2	0.01
1.01	0.01	1.0201	1.02	0.0001

In one sentence

Close to a point, every smooth curve (no corners) looks like a straight line: the slope is all you need nearby.

Near a point a function looks like a line, then a parabola.

First order (linearization):

$$f(\mathbf{x}+\boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|)$$

Second order:

$$f(\mathbf{x}+\boldsymbol{\delta}) = f(\mathbf{x}) + \nabla f^\top \boldsymbol{\delta} + \tfrac{1}{2}\boldsymbol{\delta}^\top H \boldsymbol{\delta} + o(\|\boldsymbol{\delta}\|^2)$$

Used to:

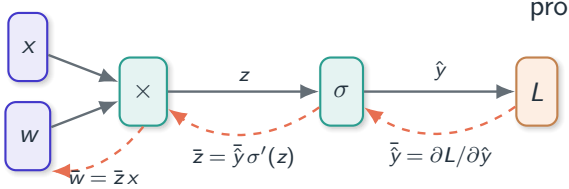
- Justify gradient descent: with $\boldsymbol{\delta} = -\eta \nabla f$, f decreases by $\approx \eta \|\nabla f\|^2$ for small η.
- Derive Newton's method: minimize the quadratic model $\Rightarrow \boldsymbol{\delta} = -H^{-1}\nabla f$.
- Derive XGBoost's objective (Chapter 6): second-order expansion of the loss around the current prediction.
- Analyse the Laplace approximation and Fisher information (Section 6).

Automatic differentiation (autodiff)

You write the forward pass; the library applies the chain rule backwards.

- **Symbolic**: manipulate formulas (expression swell).
- **Numerical**: finite differences ($O(d)$ evaluations, rounding error).
- **Automatic**: record the computation graph of elementary ops, apply the chain rule exactly.

- *Forward mode*: one input direction per pass ($O(d)$ passes for a gradient).
- *Reverse mode*: one output per pass — gradient of a scalar loss in *one* backward pass, cost $\approx$ 2–3 forward passes. This is backprop.
- `torch.autograd` / `jax.grad`: you write the forward pass, the framework produces the gradient.



forward: solid
backward (adjoints): dashed

Worked example: reverse-mode by hand

Forward for values, backward: incoming error times local derivative.

$L = (\sigma(wx) - y)^2$ with $x = 1$, $w = 2$, $y = 0$.

Step 1. Forward pass: $z = wx = 2$,
$\hat{y} = \sigma(2) = 0.881$, $L = 0.776$.

Step 2. Backward, start at the output: $\bar{L} = 1$.

Step 3. Through the square:
$\bar{\hat{y}} = 2(\hat{y} - y) = 1.762$.

Step 4. Through the sigmoid:
$\bar{z} = \bar{\hat{y}}\sigma'(z) = 1.762 \cdot 0.881 \cdot 0.119 = 0.185$.

Step 5. Through the product:
$\bar{w} = \bar{z}x = 0.185$ and $\bar{x} = \bar{z}w = 0.370$.

Numerical check

$L(w = 2.001) = 0.7762$, $L(w = 1.999) = 0.7758$

$$\frac{0.7762 - 0.7758}{0.002} \approx 0.185 \checkmark$$

Pattern

Each node: incoming error $\times$ local derivative, passed to the left. The cost is proportional to the number of operations, however many parameters there are.

Integrals you will meet (optional)

Expectations are integrals; most are replaced by averages over samples.

- Expectations are integrals: $\mathbb{E}[g(\mathbf{x})] = \int g(\mathbf{x}) p(\mathbf{x})\, d\mathbf{x}$.
- Risk $R(h) = \int \ell(y, h(\mathbf{x}))\, dP(\mathbf{x}, y)$.
- Marginalization: $p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z})\, d\mathbf{z}$ (latent variable models, VAEs).
- Normalizing constants: $Z = \int e^{-E(\mathbf{x})} d\mathbf{x}$.

Useful facts

- Gaussian integral: $\int_{-\infty}^{\infty} e^{-x^2/2} dx = \sqrt{2\pi}$.
- Leibniz rule (differentiate under the integral): $\frac{d}{d\theta} \int f(x, \theta) dx = \int \partial_\theta f\, dx$ (under regularity).
- Change of variables: $p_{\mathbf{y}}(\mathbf{y}) = p_{\mathbf{x}}(g^{-1}(\mathbf{y}))\, \left|\det J_{g^{-1}}(\mathbf{y})\right|$ (normalizing flows).
- Most integrals in ML are intractable $\Rightarrow$ Monte Carlo: $\mathbb{E}[g] \approx \frac{1}{S} \sum_s g(\mathbf{x}_s)$.

Key formulas

- Gradient ∇f: steepest ascent, orthogonal to level sets; Jacobian $J_{\mathbf{f}}$; Hessian H (curvature, convexity test).
- Chain rule: $J_{\mathbf{f}\circ\mathbf{g}} = J_{\mathbf{f}} J_{\mathbf{g}}$; gradients flow back through $J^\top$ (backprop).
- $\nabla_{\mathbf{x}} \mathbf{a}^\top \mathbf{x} = \mathbf{a}$, $\nabla_{\mathbf{x}} \mathbf{x}^\top A \mathbf{x} = (A + A^\top)\mathbf{x}$, $\nabla_{\mathbf{w}} \|X\mathbf{w} - \mathbf{y}\|^2 = 2X^\top(X\mathbf{w} - \mathbf{y})$, $\nabla_W \mathbf{a}^\top W \mathbf{b} = \mathbf{a}\mathbf{b}^\top$.
- Least squares: $\nabla J = \frac{1}{n} X^\top (X\mathbf{w} - \mathbf{y})$; logistic: $\nabla J = \frac{1}{n} X^\top (\mathbf{p} - \mathbf{y})$.
- Taylor: $f(\mathbf{x} + \boldsymbol{\delta}) \approx f + \nabla f^\top \boldsymbol{\delta} + \frac{1}{2} \boldsymbol{\delta}^\top H \boldsymbol{\delta}$.
- Reverse-mode autodiff computes the full gradient in one backward pass.

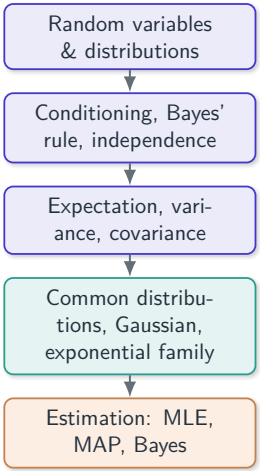
Probability I: Foundations

Section 4 — Chapter 2

Why probability?

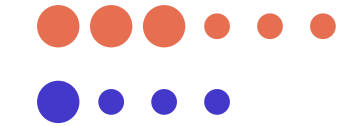
Data are noisy; probability is the language for how sure we are.

- Data are noisy: $y = f(\mathbf{x}) + \varepsilon$.
- Labels are uncertain: we want $P(y \mid \mathbf{x})$, not just a guess.
- Models are estimated from finite samples: parameters have uncertainty.
- Many algorithms *are* probabilistic models: Naive Bayes, logistic regression, GMM.



Intuition: a bag of marbles

- A bag holds 10 marbles: 6 red and 4 blue; among the red ones, 3 are big; among the blue, 1 is big.
- **Probability**: pick one without looking; the chance of red is 6/10.
- **Joint**: the chance of "red *and* big" is 3/10.
- **Marginal**: forget the size; the chance of red is 6/10 (add up big red and small red).
- **Conditional**: you felt that the marble is big (4 big marbles in total); now the chance it is red is 3/4.
- **Independent** would mean: knowing the size tells you nothing about the colour. Here it does, so they are *not* independent.

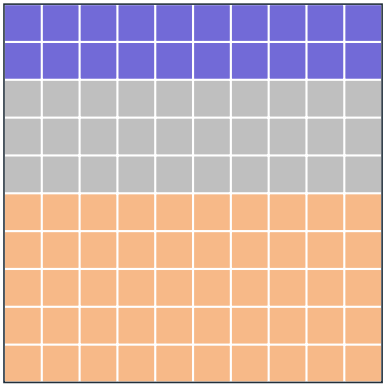


red: 3 big, 3 small blue: 1 big, 3 small

In one sentence

Joint = both things at once; marginal = ignore one thing; conditional = you already know one thing.

Picture: chance as 100 small squares



100 squares = 100%

- Tomorrow's weather is unknown. Cut the chance into 100 equal squares: the forecast gives sun 50, clouds 30, rain 20.
- The chance of rain is the share of squares it gets: 20 out of 100, that is 20%.
- "Clouds or rain" is $30 + 20 = 50$ squares: chances of different things add up.
- All 100 squares together make 100%: something must happen tomorrow.

In one sentence

A probability is the share of the 100 squares one outcome gets; the whole coloured grid is called the distribution.

Probability space and random variables

A random variable: a number not yet known; its distribution says what is likely.

Axioms (Kolmogorov)

Sample space Ω, events $A \subseteq \Omega$, and $\mathbb{P}$ with (i) $\mathbb{P}(A) \geq 0$, (ii) $\mathbb{P}(\Omega) = 1$, (iii) $\mathbb{P}(\bigcup_i A_i) = \sum_i \mathbb{P}(A_i)$ for disjoint A_i.

Random variable

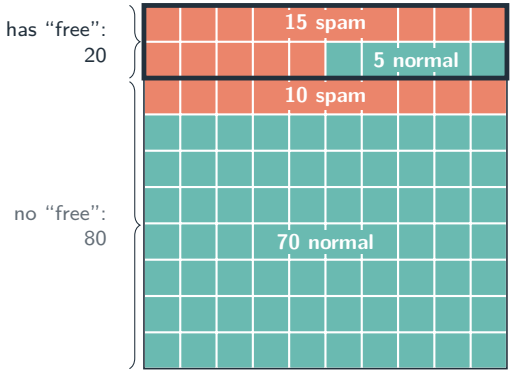
$X : \Omega \to \mathbb{R}$. Discrete: probability mass function $p(x) = \mathbb{P}(X = x)$. Continuous: density $p(x)$ with $\mathbb{P}(a \leq X \leq b) = \int_a^b p(x)dx$.

- CDF: $F(x) = \mathbb{P}(X \leq x)$, $p = F'$ for continuous X.
- Random *vector* $\mathbf{x} \in \mathbb{R}^d$: joint density $p(x_1, \ldots, x_d)$.

Two interpretations

Frequentist: long-run frequency. *Bayesian*: degree of belief. Same calculus, different use (Section 6).

Picture: 100 emails in a grid



- 100 emails: 25 spam (junk mail, red), 75 normal (green). The 20 with the word “free” sit in the thick box.
- **Joint**: “free” *and* spam together: 15 red squares in the box, so 15%.
- **Marginal**: “free” alone, spam or not: the whole box, $15 + 5 = 20$, so 20%.
- **Conditional**: you see “free”, so only the box counts: 15 of its 20 squares are red, $15/20 = 75\%$.

In one sentence

Knowing a fact shrinks the world to the matching squares; then count again inside that smaller world.

Joint, marginal and conditional distributions

Joint: both at once. Marginal: forget one. Conditional: one is fixed.

- Joint: $p(x, y)$
- Marginal (sum/integrate out): $p(x) = \sum_y p(x, y)$
- Conditional: $p(y \mid x) = \dfrac{p(x, y)}{p(x)}$
- Product rule: $p(x, y) = p(y \mid x)p(x) = p(x \mid y)p(y)$
- Chain rule: $p(x_1, \ldots, x_d) = \prod_j p(x_j \mid x_1, \ldots, x_{j-1})$

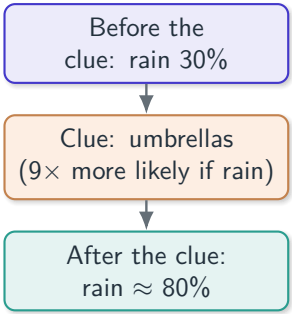
Worked example

$p(x, y)$	$y =$spam	$y =$ham	$p(x)$
$x =$"free"	0.15	0.05	0.20
$x =$no "free"	0.10	0.70	0.80
$p(y)$	0.25	0.75	1

$p(\text{spam} \mid \text{"free"}) = 0.15/0.20 = 0.75$
$p(\text{"free"} \mid \text{spam}) = 0.15/0.25 = 0.60$

Intuition: updating a guess with a clue

- You wake up and wonder: is it raining? From the season you guess 30%. This is your **prior**: what you believe before any clue.
- You look out and see people with umbrellas. Umbrellas are very common when it rains (say 90%) and rare when it does not (say 10%). This is the **likelihood**: how well each explanation fits the clue.
- Bayes' rule combines the two: your belief in rain jumps to about 80%. This is the **posterior**.
- If it almost never rains here (prior 1%), the umbrellas only raise your belief to 8%: rare things stay rare after one clue.



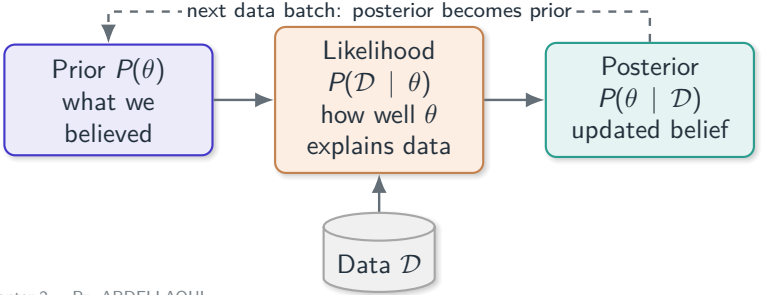
In one sentence

New belief = old belief, boosted by how much better the clue fits one explanation than the other.

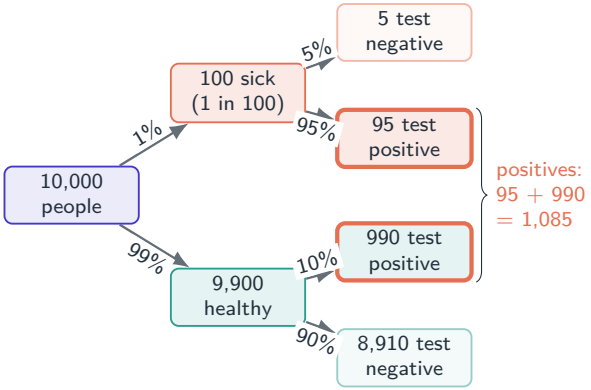
Flip the conditional: from data-given-cause to cause-given-data.

Bayes' rule

$$\underbrace{P(\theta \mid \mathcal{D})}_{\text{posterior}} = \frac{\overbrace{P(\mathcal{D} \mid \theta)}^{\text{likelihood}} \overbrace{P(\theta)}^{\text{prior}}}{\underbrace{P(\mathcal{D})}_{\text{evidence}}}, \qquad P(\mathcal{D}) = \sum_{\theta'} P(\mathcal{D} \mid \theta') P(\theta')$$



Picture: the test on 10,000 people



- 10,000 people. The disease is rare: 100 are sick, 9,900 are healthy.
- A positive test means the test says "sick". It is right for 95 of the 100 sick. But it also says "sick" to 10 of every 100 healthy people: 990 false alarms.
- Positive tests: $95 + 990 = 1{,}085$. Only 95 of them are really sick: about 9 in 100.

In one sentence

Rare stays rare: the few true alarms are drowned by the many false alarms from the healthy crowd.

A rare disease plus a good test still gives mostly false alarms.

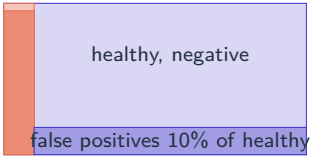
Prevalence $P(D) = 1\%$; sensitivity $P(+ \mid D) = 0.95$; specificity $P(- \mid \neg D) = 0.90$. You test positive. $P(D \mid +)$?

Step 1. True positives:
$P(+ \mid D)P(D) = 0.95 \cdot 0.01 = 0.0095$.

Step 2. False positives:
$P(+ \mid \neg D)P(\neg D) = 0.10 \cdot 0.99 = 0.099$.

Step 3. All positives:
$P(+) = 0.0095 + 0.099 = 0.1085$.

Step 4. Bayes: $P(D \mid +) = \dfrac{0.0095}{0.1085} \approx \mathbf{8.8}\%$.



ML lesson

Classifiers on rare classes must account for the prior: this is the class-imbalance problem, and why calibrated probabilities matter.

Independence and conditional independence

Independent = knowing one tells nothing about the other.

Independence

$X \perp Y \iff p(x, y) = p(x)p(y) \iff p(y \mid x) = p(y)$.

Conditional independence

$X \perp Y \mid Z \iff p(x, y \mid z) = p(x \mid z)p(y \mid z)$.

- i.i.d. samples: $p(\mathcal{D}) = \prod_i p(\mathbf{x}_i)$ — the assumption behind every likelihood in this course.

Naive Bayes (Chapter 4)

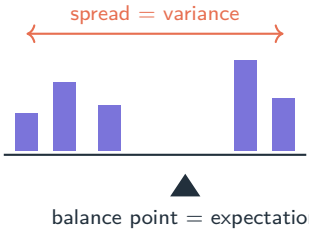
Assumes features are conditionally independent given the class:

$$p(\mathbf{x} \mid y) = \prod_{j=1}^{d} p(x_j \mid y).$$

Reduces a d-dimensional density to d one-dimensional ones. Wrong in general, surprisingly effective.

Intuition: balance point and spread

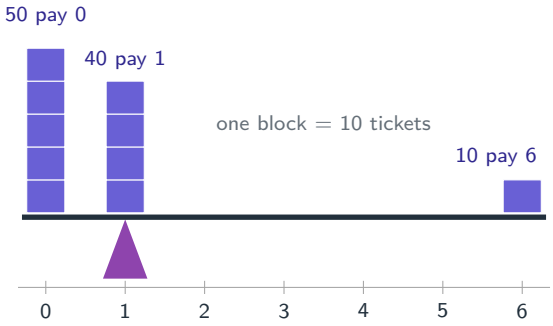
- **Expectation** is the average you would get in the long run. Picture the values as weights on a seesaw: the expectation is where the seesaw balances.
- **Variance** is how spread out the values are around that balance point. Small variance: everyone is about the same height. Large variance: a basketball team mixed with children.
- **Why averaging helps**: measure a table with a shaky ruler ten times. Each measurement is a bit off, but the errors go in both directions and partly cancel. The average of n measurements is $\sqrt{n}$ times more precise.



In one sentence

The mean is the balance point, the variance is the spread, and averaging shrinks the spread.

Picture: the seesaw balance point



balances at 1 = expectation

- 100 lottery tickets: 50 pay nothing, 40 pay 1 dirham, 10 pay 6 dirhams.
- All together they pay $0 + 40 + 60 = 100$ dirhams: 1 dirham per ticket on average. This average is the expectation.
- Same with shares (0.5 means 50 of 100): $0 \times 0.5 + 1 \times 0.4 + 6 \times 0.1 = 1$. Each value times its share, added up.
- Put the tickets as weights on a plank: it balances exactly at 1.

In one sentence

The expectation is the fair price of one ticket: the balance point of the plank.

The long-run average; always linear; estimated by averaging samples.

Expectation

$\mathbb{E}[X] = \sum_x x\,p(x)$ (discrete), $\mathbb{E}[X] = \int x\,p(x)\,dx$ (continuous). For a function: $\mathbb{E}[g(X)] = \int g(x)p(x)dx$.

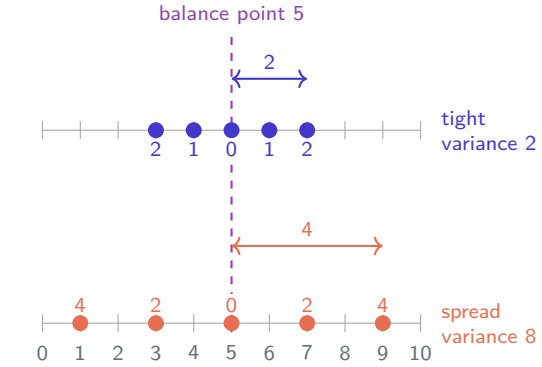
Properties:

- Linearity: $\mathbb{E}[aX+bY] = a\mathbb{E}[X]+b\mathbb{E}[Y]$ — *always*, even if dependent.
- $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$ if independent.
- Tower rule: $\mathbb{E}[X] = \mathbb{E}\big[\mathbb{E}[X \mid Y]\big]$.
- Jensen: g convex $\Rightarrow g(\mathbb{E}[X]) \le \mathbb{E}[g(X)]$.

Monte Carlo estimation

$\mathbb{E}[g(X)] \approx \frac{1}{S}\sum_{s=1}^{S} g(x_s)$, $x_s \sim p$. Unbiased, error $O(1/\sqrt{S})$ regardless of dimension. Used for: expected loss on a test set, dropout, stochastic gradients, VAEs.

Picture: tight dots vs spread dots



- Both rows balance at 5, but the red dots sit twice as far away.
- Variance: take the small numbers (distances to 5); multiply each by itself (square it). Add up, divide by the 5 dots: blue $10/5 = 2$, red $40/5 = 8$.
- Twice as far: 4 times the variance. The standard deviation, its square root, only doubles: 1.4 vs 2.8.

In one sentence

Variance: how far the dots usually sit from the balance point. Standard deviation: the same, in plain distance.

Variance = spread around the mean; averaging n values divides it by n.

Variance

$\mathrm{Var}(X) = \mathbb{E}[(X-\mathbb{E}X)^2] = \mathbb{E}[X^2]-(\mathbb{E}X)^2$; $\sigma = \sqrt{\mathrm{Var}(X)}$.

- $\mathrm{Var}(aX+b) = a^2\,\mathrm{Var}(X)$
- $\mathrm{Var}(X+Y) = \mathrm{Var}\,X + \mathrm{Var}\,Y + 2\,\mathrm{Cov}(X,Y)$
- Independent: $\mathrm{Var}(\sum_i X_i) = \sum_i \mathrm{Var}\,X_i$
- Mean of n i.i.d.: $\mathrm{Var}(\bar{X}) = \sigma^2/n$

Covariance and correlation

$\mathrm{Cov}(X,Y) = \mathbb{E}[(X-\mathbb{E}X)(Y-\mathbb{E}Y)] = \mathbb{E}[XY] - \mathbb{E}X\mathbb{E}Y$

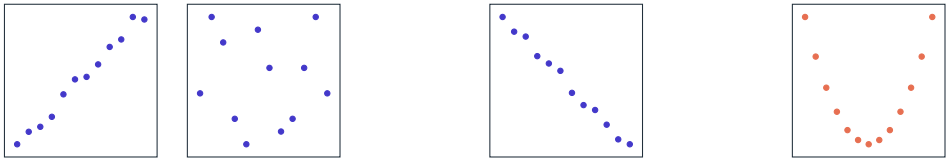
$\rho = \dfrac{\mathrm{Cov}(X,Y)}{\sigma_X \sigma_Y} \in [-1,1]$ (Cauchy–Schwarz)

- ρ measures *linear* dependence only.
- $\rho = 0 \not\Rightarrow$ independent ($Y = X^2$ with symmetric X).

Correlation gallery

Correlation only sees straight lines; a perfect curve can score zero.

$\rho \approx 0.95$ $\rho \approx 0$ $\rho \approx -0.9$ $\rho = 0$, dependent



- Right panel: $Y = X^2$ — perfectly dependent, zero correlation. Mutual information (Section 7) detects it.

Variances on the diagonal, pairwise covariances off it.

Moments of $\mathbf{x} \in \mathbb{R}^d$

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] \in \mathbb{R}^d, \qquad \Sigma = \mathrm{Cov}(\mathbf{x}) = \mathbb{E}\big[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^\top\big] \in \mathbb{R}^{d\times d}, \quad \Sigma_{jk} = \mathrm{Cov}(x_j, x_k)$$

- Symmetric PSD; diagonal = variances.
- Linear map: $\mathbb{E}[A\mathbf{x}+\mathbf{b}] = A\boldsymbol{\mu}+\mathbf{b}$, $\mathrm{Cov}(A\mathbf{x}+\mathbf{b}) = A\Sigma A^\top$.
- Variance along unit direction $\mathbf{u}$: $\mathrm{Var}(\mathbf{u}^\top\mathbf{x}) = \mathbf{u}^\top\Sigma\mathbf{u}$.
- Sample estimates: $\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_i \mathbf{x}_i$, $\hat{\Sigma} = \frac{1}{n-1}\sum_i(\mathbf{x}_i-\hat{\boldsymbol{\mu}})(\mathbf{x}_i-\hat{\boldsymbol{\mu}})^\top = \frac{1}{n-1}X_c^\top X_c$.

Example

$\Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$: $\sigma_1 = 2$, $\sigma_2 = \sqrt{3}$, $\rho = 2/(2\sqrt{3}) = 0.577$.

Eigenvalues $\lambda = \frac{7\pm\sqrt{17}}{2} = \{5.56, 1.44\}$: most variance along $\mathbf{q}_1 \approx (0.79, 0.61)$ — the first principal component.

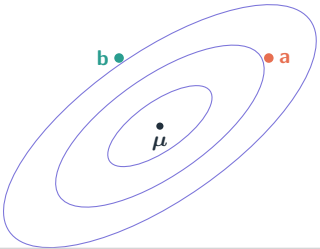
How many standard deviations away, accounting for scale and correlation.

Mahalanobis distance

$$d_M(\mathbf{x}, \boldsymbol{\mu}) = \sqrt{(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})}$$

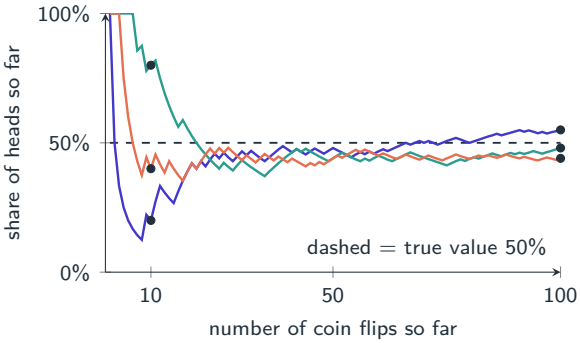
Euclidean distance after *whitening* $\mathbf{z} = \Sigma^{-1/2}(\mathbf{x} - \boldsymbol{\mu})$, which has $\mathrm{Cov}(\mathbf{z}) = I$.

- Accounts for scale *and* correlation between features.
- Level sets are the ellipsoids of the Gaussian.
- Used in: LDA/QDA, outlier detection, GMM E-step.



$\|\mathbf{a} - \boldsymbol{\mu}\|_2 > \|\mathbf{b} - \boldsymbol{\mu}\|_2$ but $d_M(\mathbf{a}) < d_M(\mathbf{b})$: $\mathbf{a}$ lies along the main axis of variation.

Picture: averages settle down



- Three friends each flip a fair coin (half heads) 100 times. Each line is one friend's share of heads so far.
- After 10 flips (left dots): 20%, 40% and 80%. Luck still rules.
- After 100 flips (right dots): 44% to 55%, close to the true 50%. After 10,000 flips: within about 1%.

In one sentence

Average many random outcomes and the average stops wobbling: it settles on the true value.

Averages settle to the true mean (LLN) and fluctuate like a Gaussian (CLT).

LLN

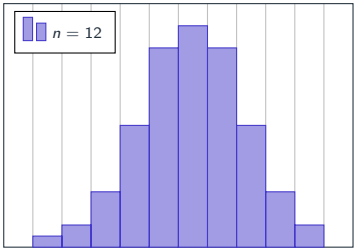
$X_1, \dots, X_n$ i.i.d. with mean μ: $\bar{X}_n = \frac{1}{n}\sum X_i \to \mu$ (in probability / almost surely).

CLT

With variance $\sigma^2 < \infty$:

$$\sqrt{n}\,\frac{\bar{X}_n - \mu}{\sigma} \xrightarrow{d} \mathcal{N}(0,1).$$

Justifies confidence intervals $\bar{X} \pm 1.96\,\sigma/\sqrt{n}$ and Gaussian noise models.



$\bar{X}_n$ (uniform summands)

Sum of 12 uniforms is already almost Gaussian.

An average of n samples is rarely off by more than about $1/\sqrt{n}$.

- **Markov**: $X \geq 0 \Rightarrow \mathbb{P}(X \geq a) \leq \dfrac{\mathbb{E}X}{a}$
- **Chebyshev**: $\mathbb{P}(|X - \mu| \geq a) \leq \dfrac{\sigma^2}{a^2}$
- **Hoeffding**: $X_i \in [a, b]$ i.i.d.:

$$\mathbb{P}\big(|\bar{X}_n - \mu| \geq \epsilon\big) \leq 2\exp\Big(-\frac{2n\epsilon^2}{(b-a)^2}\Big)$$

Why they matter

They give *finite-sample* guarantees, unlike the asymptotic CLT. Hoeffding is the seed of every generalization bound in Chapter 5: for a $[0,1]$-valued loss, the training error of a fixed h is within ϵ of its true risk with probability $\geq 1 - 2e^{-2n\epsilon^2}$.

Numbers

$n = 1000$, $\epsilon = 0.05$: bound $= 2e^{-5} \approx 0.013$. To halve ϵ, quadruple n.

Transform a variable and its density is rescaled by how much space stretches.

Change of variables

$Y = g(X)$ with g invertible and differentiable:

$$p_Y(y) = p_X(g^{-1}(y)) \left| \frac{d}{dy} g^{-1}(y) \right|$$

Vector case: multiply by $\left|\det J_{g^{-1}}(\mathbf{y})\right|$.

Example

$X \sim \mathcal{N}(0,1)$, $Y = \mu + \sigma X$: $p_Y(y) = \frac{1}{\sigma}\phi\left(\frac{y-\mu}{\sigma}\right)$, i.e. $\mathcal{N}(\mu, \sigma^2)$.

Reparameterization trick (VAE, Chapter 10)

To take gradients of $\mathbb{E}_{z\sim\mathcal{N}(\mu,\sigma^2)}[f(z)]$ w.r.t. μ, σ, write $z = \mu + \sigma\epsilon$, $\epsilon \sim \mathcal{N}(0,1)$, and differentiate inside the expectation.

Normalizing flows

Stack invertible maps with cheap Jacobian determinants to turn a Gaussian into a complex density.

Key formulas

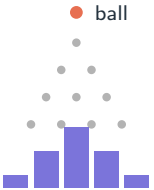
- Product rule $p(x, y) = p(y \mid x)p(x)$; marginalization $p(x) = \sum_y p(x, y)$; Bayes $P(\theta \mid \mathcal{D}) \propto P(\mathcal{D} \mid \theta)P(\theta)$.
- Independence $p(x, y) = p(x)p(y)$; conditional independence powers Naive Bayes and Markov models.
- $\mathbb{E}$ is linear; $\mathrm{Var}(X) = \mathbb{E}X^2 - (\mathbb{E}X)^2$; $\mathrm{Var}(\bar{X}) = \sigma^2/n$; $\mathrm{Cov}(A\mathbf{x}) = A\Sigma A^\top$.
- Mahalanobis $d_M^2 = (\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})$; whitening $\Sigma^{-1/2}(\mathbf{x} - \boldsymbol{\mu})$.
- LLN, CLT, Hoeffding $\mathbb{P}(\left|\bar{X} - \mu\right| \geq \epsilon) \leq 2e^{-2n\epsilon^2}$.
- Change of variables $p_Y(y) = p_X(g^{-1}(y)) \left|\det J_{g^{-1}}\right|$.

Probability II: Distributions and ~~the Gau~~ssian

Section 5 — Chapter 2

Intuition: shapes of randomness

- A **coin**: two outcomes, one number says how biased it is. That is the Bernoulli.
- A **die** with K faces: the Categorical. A class label is a die roll whose faces are the classes.
- **Heights of people**: most are near the middle, few are very tall or very short. The bell curve, or Gaussian.
- Why bells are everywhere: drop a ball through a board of pegs (a Galton board). At each peg it bounces left or right at random; many small random pushes add up to a bell-shaped pile at the bottom. Measurement noise is the same: many tiny causes.



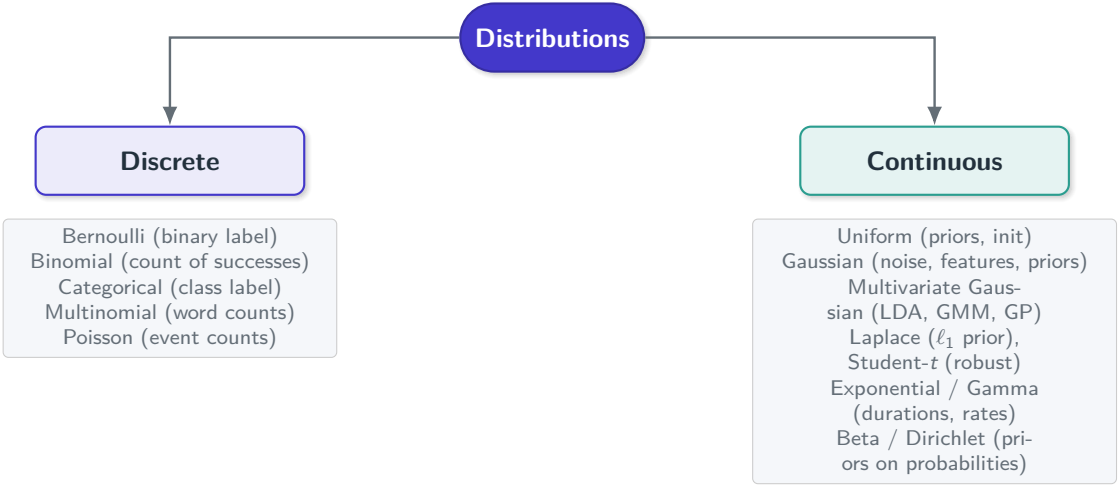
many random bounces $\rightarrow$ a bell

In one sentence

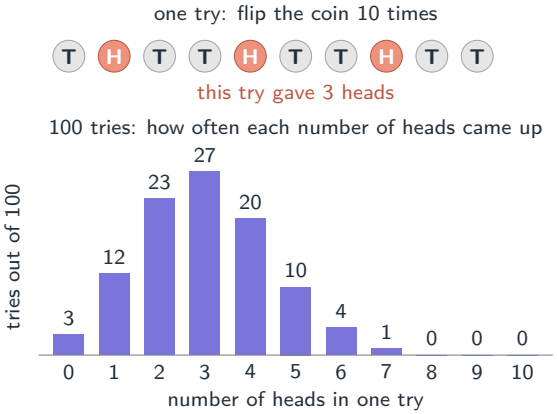
Coin, die, bell, bus: each distribution is a familiar kind of randomness with a few knobs.

Distributions used in this course

A distribution is a recipe for randomness; each fits a typical kind of data.



Picture: heads in 10 coin flips



- This coin is not fair: it lands heads about 3 times in 10. One flip is a **Bernoulli**: heads with chance 0.3.
- Flip it 10 times and count the heads. The count is most often 3, often 2 or 4, and 8 or more almost never. That count is a **Binomial**.
- Typical count: $10 \times 0.3 = 3$ heads.

In one sentence

Bernoulli is one flip of a coin; Binomial is how many heads you get in several flips.

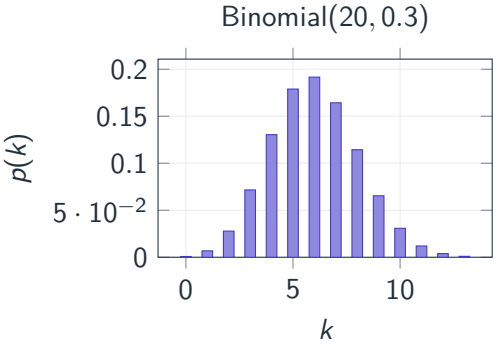
Bernoulli = one coin flip; Binomial = number of heads in n flips.

Bernoulli(θ)

$x \in \{0, 1\}$, $p(x \mid \theta) = \theta^x (1-\theta)^{1-x}$
$\mathbb{E}[x] = \theta$, $\mathrm{Var}(x) = \theta(1-\theta)$

Binomial(n, θ)

k successes in n trials: $p(k) = \binom{n}{k} \theta^k (1-\theta)^{n-k}$
$\mathbb{E}[k] = n\theta$, $\mathrm{Var}(k) = n\theta(1-\theta)$



In ML

Binary classification models $y \mid \mathbf{x} \sim \text{Bernoulli}(\sigma(\mathbf{w}^\top \mathbf{x}))$; its negative log-likelihood is the cross-entropy loss.

Categorical and Multinomial

Categorical = one roll of a K-sided die; Multinomial = counts after n rolls.

Categorical(π)

$y \in \{1, \dots, K\}$, $p(y = k) = \pi_k$, $\sum_k \pi_k = 1$.
One-hot form: $p(\mathbf{y}) = \prod_k \pi_k^{y_k}$.

Multinomial(n, π)

Counts $\mathbf{c}$ over K categories in n draws:
$p(\mathbf{c}) = \dfrac{n!}{\prod_k c_k!} \prod_k \pi_k^{c_k}$.

In ML

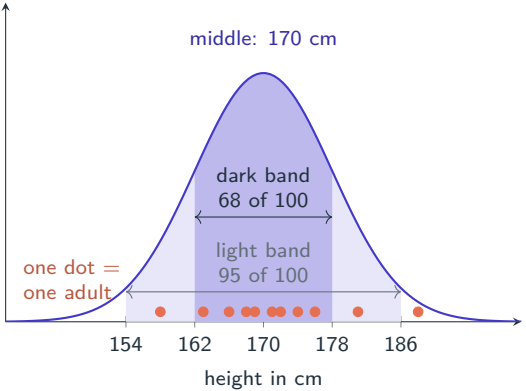
- Multiclass classification: $y \mid \mathbf{x} \sim \text{Cat}(\text{softmax}(W\mathbf{x}))$; NLL $= -\log \pi_y =$ multiclass cross-entropy.
- Multinomial Naive Bayes for text: word counts of a document given its class.
- Language models: next token $\sim$ Categorical over the vocabulary.

Poisson counts events, Exponential measures waiting times.

Distribution	Support	pmf / pdf	Mean, variance
Poisson(λ)	$k \in \mathbb{N}$	$\frac{\lambda^k e^{-\lambda}}{k!}$	λ, λ
Exponential(λ)	$x \geq 0$	$\lambda e^{-\lambda x}$	$1/\lambda$, $1/\lambda^2$
Gamma(α, β)	$x > 0$	$\frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$	α/β, α/β^2
Uniform(a, b)	$[a, b]$	$\frac{1}{b-a}$	$\frac{a+b}{2}$, $\frac{(b-a)^2}{12}$

- Poisson regression (a GLM, Chapter 3): counts $y \mid \mathbf{x} \sim \text{Poisson}(e^{\mathbf{w}^\top \mathbf{x}})$.
- Exponential: memoryless waiting times; Gamma: conjugate prior for a Poisson rate or a Gaussian precision.
- Uniform: weight initialization ranges, random search over hyperparameters.

Picture: the bell curve of heights



- Each dot is one adult. Heights pile up near 170 cm and thin out on both sides: a bell.
- Dark band: 68 of 100 people are within 8 cm of the middle (162 to 178). Light band: 95 of 100 are within 16 cm (154 to 186).
- Two numbers fix the whole curve: middle 170 and width 8.

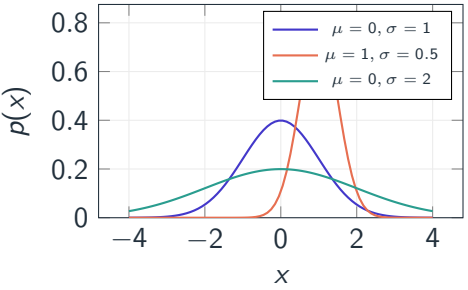
In one sentence

A bell is fixed by its centre and its width; two thirds of everyone sit within one width of the centre.

The bell curve; its negative log is a parabola, hence squared loss.

$\mathcal{N}(\mu, \sigma^2)$

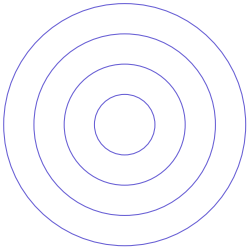
$$p(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big), \qquad \mathbb{E}[x] = \mu,\ \mathrm{Var}(x) = \sigma^2$$



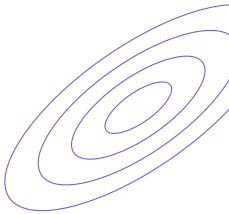
- 68–95–99.7% within $1, 2, 3\,\sigma$.
- Closed under linear maps and sums of independent Gaussians.
- Maximum-entropy distribution for given mean and variance.
- CLT makes it ubiquitous.
- $-\log p(x) = \frac{(x-\mu)^2}{2\sigma^2} + \text{const}$: Gaussian NLL = squared loss.

Intuition: the bell is a hill

- Pour sand on a table: it forms a round hill. The hill is the probability: high where points are likely, low far away.
- On a hiking map a hill is drawn with height lines. For a round hill they are circles; for a stretched hill they are ellipses.
- The **covariance matrix** is the shape of the hill: how wide it is in each direction and whether it is tilted. Tilted means the two variables move together (correlation).
- The **Mahalanobis distance** counts how many height lines you cross to reach a point: two points can be equally far in metres but one is far more surprising because the hill is narrow in its direction.



round hill

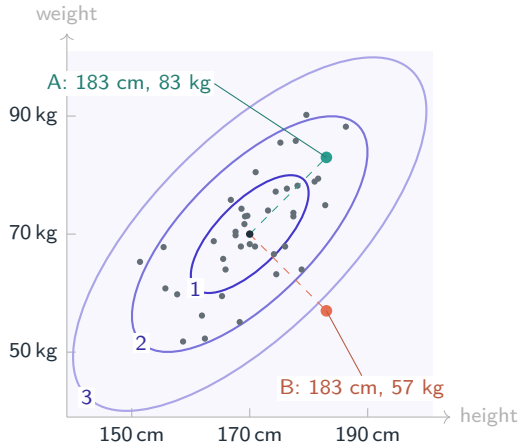


tilted hill: correlation

In one sentence

A 2-D Gaussian is a hill; its height lines are ellipses, and the covariance is the hill's shape.

Picture: a tilted hill of dots



- Each dot is one person: height across, weight up. Tilted cloud: taller people are usually heavier.
- The rings are the hill's height lines: nearly all dots are inside ring 2, none outside ring 3.
- A and B are both 13 cm and 13 kg from the centre (170 cm, 70 kg). A crosses 1 ring, B crosses 3: tall and light is rare.

In one sentence

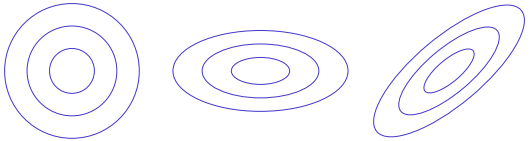
A 2-D bell is a tilted hill; count rings, not centimetres, to see how rare a point is.

A bell in d dimensions; its contours are the ellipses of Σ.

$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ on $\mathbb{R}^d$

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\,|\Sigma|^{1/2}} \exp\Big(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big)$$

- Exponent $= -\frac{1}{2} d_M^2$: level sets are ellipsoids with axes = eigenvectors of Σ, semi-axes $\propto \sqrt{\lambda_i}$.
- Σ diagonal $\Rightarrow$ independent coordinates, axis-aligned ellipses.
- $\Sigma = \sigma^2 I \Rightarrow$ spherical (isotropic).



Cut or slice a Gaussian and you get a Gaussian again.

Partition $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, $\boldsymbol{\mu} = (\boldsymbol{\mu}_a, \boldsymbol{\mu}_b)$, $\Sigma = \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}$.

- **Marginal**: $\mathbf{x}_a \sim \mathcal{N}(\boldsymbol{\mu}_a, \Sigma_{aa})$
- **Conditional**: $\mathbf{x}_a \mid \mathbf{x}_b \sim \mathcal{N}(\boldsymbol{\mu}_{a|b}, \Sigma_{a|b})$ with

$$\boldsymbol{\mu}_{a|b} = \boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$$
$$\Sigma_{a|b} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$$

- **Linear map**: $A\mathbf{x} + \mathbf{b} \sim \mathcal{N}(A\boldsymbol{\mu} + \mathbf{b}, A\Sigma A^\top)$

Why this is powerful

Everything stays Gaussian and closed-form. Gaussian processes (Chapter 7) are exactly the conditional formula: predict test outputs $\mathbf{x}_a$ given training outputs $\mathbf{x}_b$.

Note the conditional mean is *linear* in $\mathbf{x}_b$: linear regression is the Gaussian conditional.

Observing one correlated variable shifts and sharpens the other.

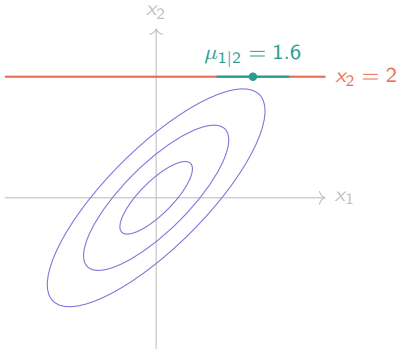
$\boldsymbol{\mu} = (0, 0)$, $\Sigma = \begin{pmatrix} 1 & 0.8 \\ 0.8 & 1 \end{pmatrix}$. We observe $x_2 = 2$. What do we now believe about x_1?

Step 1. Read off the pieces: $\Sigma_{11} = 1$, $\Sigma_{22} = 1$, $\Sigma_{12} = 0.8$.

Step 2. New mean: $\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) = 0.8 \cdot 2 = 1.6$.

Step 3. New variance: $\sigma^2_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = 1 - 0.64 = 0.36$, so $\sigma = 0.6$.

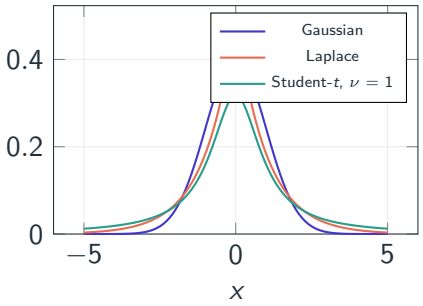
Step 4. Before: $x_1 \sim \mathcal{N}(0, 1)$. After: $x_1 \sim \mathcal{N}(1.6, 0.36)$, shifted towards x_2 and *sharper*.



Regression coefficient

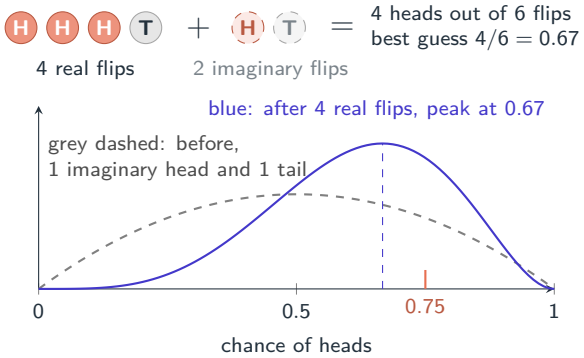
The factor $\Sigma_{12}\Sigma_{22}^{-1} = 0.8$ is the slope of the regression line of x_1 on x_2.

Expect occasional big surprises: this is what makes models robust.



- **Laplace**(μ, b): $p(x) = \frac{1}{2b} e^{-|x-\mu|/b}$. NLL $\propto |x - \mu|$: absolute loss. As a prior on weights: $-\log p(\mathbf{w}) \propto \|\mathbf{w}\|_1$ $\Rightarrow$ Lasso.
- **Student-t**(ν): polynomial tails; robust regression and robust mixtures — outliers get less influence.
- Gaussian tails decay as e^{-x^2}: an observation 10σ away is "impossible" under the Gaussian, mildly surprising under t.

Picture: imaginary coin flips



In one sentence

A few imaginary flips added to the real ones stop a handful of flips from fooling you; that is all a "Beta prior" is.

- A new coin: 4 flips give 3 heads, 1 tail. Trusting only that, heads has chance $3/4 = 0.75$ (red mark). But 4 flips is very little.
- Pretend you had already seen 1 head and 1 tail. Now it is 4 heads out of 6: best guess 0.67, pulled a bit towards one half.
- The blue curve shows which chances are still believable; its peak is the best guess. More real flips make it narrower, and the imaginary flips matter less.

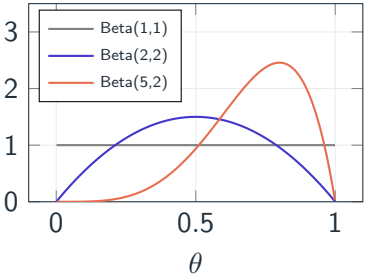
A Beta prior = a few imaginary observations added to the data.

Beta(α, β) on $\theta \in [0, 1]$

$$p(\theta) = \frac{\theta^{\alpha-1}(1-\theta)^{\beta-1}}{B(\alpha, \beta)}, \ \mathbb{E}\theta = \frac{\alpha}{\alpha+\beta}$$

Dirichlet(α) on the simplex

$$p(\boldsymbol{\pi}) \propto \prod_k \pi_k^{\alpha_k - 1}, \ \mathbb{E}\pi_k = \alpha_k / \sum_j \alpha_j$$



Conjugacy

Beta prior + Bernoulli likelihood ⇒ Beta posterior: Beta$(\alpha + \#\text{ones}, \beta + \#\text{zeros})$. Dirichlet is the same for Categorical. This is what *Laplace smoothing* in Naive Bayes computes (Chapter 4).

One template covering Bernoulli, Poisson, Gaussian; every GLM follows.

Exponential family

$$p(\mathbf{x} \mid \boldsymbol{\eta}) = h(\mathbf{x}) \exp\big(\boldsymbol{\eta}^\top T(\mathbf{x}) - A(\boldsymbol{\eta})\big)$$

$\boldsymbol{\eta}$: natural parameter; $T(\mathbf{x})$: sufficient statistic; $A(\boldsymbol{\eta})$: normalizer.

Distribution	η	$T(x)$	$A(\eta)$	Link to ML
Bernoulli(θ)	$\log \frac{\theta}{1-\theta}$	x	$\log(1+e^\eta)$	logistic regression
Poisson(λ)	$\log \lambda$	x	e^η	Poisson regression
Gaussian (known σ^2=1)	μ	x	$\eta^2/2$	linear regression

Why bother

Set $\eta = \mathbf{w}^\top \mathbf{x}$ and you get linear, logistic and Poisson regression as one algorithm with three choices of A (Chapter 3).

Derivatives of A give mean and variance; MLE becomes convex.

Moments from derivatives of A:

$$\nabla A(\boldsymbol{\eta}) = \mathbb{E}[T(\mathbf{x})], \qquad \nabla^2 A(\boldsymbol{\eta}) = \mathrm{Cov}(T(\mathbf{x})) \succeq 0$$

$\Rightarrow A$ is convex $\Rightarrow$ the negative log-likelihood

$$-\log p(\mathcal{D} \mid \boldsymbol{\eta}) = -\boldsymbol{\eta}^\top \textstyle\sum_i T(\mathbf{x}_i) + nA(\boldsymbol{\eta}) + \mathrm{c}$$

is convex in $\boldsymbol{\eta}$: MLE is a convex problem.

MLE = moment matching: setting the gradient to zero gives

$$\nabla A(\hat{\boldsymbol{\eta}}) = \frac{1}{n} \sum_i T(\mathbf{x}_i)$$

the model's expected sufficient statistics equal the empirical ones.

Generalized linear models (Chapter 3)

Put $\boldsymbol{\eta} = \mathbf{w}^\top \mathbf{x}$: linear, logistic, Poisson and softmax regression are the *same* algorithm with different A. Their NLL is convex in $\mathbf{w}$.

Rewrite the coin flip as an exponential and read off η, T, A.

Step 1. Start from $p(x \mid \theta) = \theta^x(1-\theta)^{1-x}$ and take exp log: $p = \exp\big(x\log\theta + (1-x)\log(1-\theta)\big)$.

Step 2. Group the terms in x: $p = \exp\Big(x\log\frac{\theta}{1-\theta} + \log(1-\theta)\Big)$.

Step 3. Name the pieces: $\eta = \log\frac{\theta}{1-\theta}$ (the log-odds), $T(x) = x$, and $-A(\eta) = \log(1-\theta)$.

Step 4. Express A in η: since $\theta = \sigma(\eta)$, $1-\theta = \frac{1}{1+e^{\eta}}$, so $A(\eta) = \log(1+e^{\eta})$.

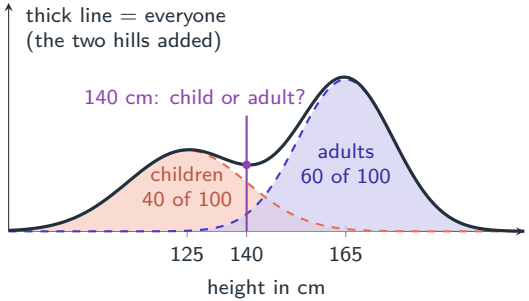
Step 5. Check the mean rule $A'(\eta) = \mathbb{E}[x]$:

$$A'(\eta) = \frac{e^{\eta}}{1+e^{\eta}} = \sigma(\eta) = \theta \checkmark$$

Logistic regression falls out

Put $\eta = \mathbf{w}^\top\mathbf{x}$: then $\theta = \sigma(\mathbf{w}^\top\mathbf{x})$ and the negative log-likelihood is $\sum_i\big[\log(1+e^{\mathbf{w}^\top\mathbf{x}_i}) - y_i\mathbf{w}^\top\mathbf{x}_i\big]$, a convex function of $\mathbf{w}$.

Picture: two hills added together



In one sentence

A mixture is a few bells added with weights; a point in the middle belongs partly to each bell.

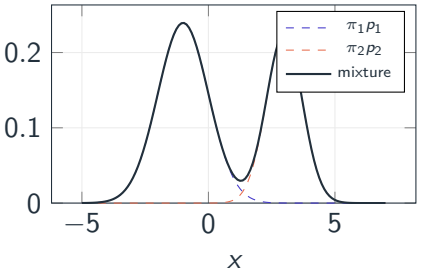
- A town: 40 of 100 people are children, about 125 cm tall; 60 are adults, about 165 cm.
- Pick someone at random: first a coin that says "child" 40% of the time, then a height from that group's bell.
- Everyone's curve is the two bells added: $0.4 \times$children $+ 0.6 \times$adults, two bumps. At 140 cm the orange bell is about 3 times higher than the blue one, so that person is probably a child (about 3 chances in 4).

Pick a component at random, then draw from it: several overlapping bumps.

Mixture

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \, p_k(\mathbf{x} \mid \boldsymbol{\theta}_k), \qquad \pi_k \geq 0, \ \sum_k \pi_k = 1$$

Generative story: draw a component $z \sim \mathrm{Cat}(\boldsymbol{\pi})$, then $\mathbf{x} \sim p_z$.

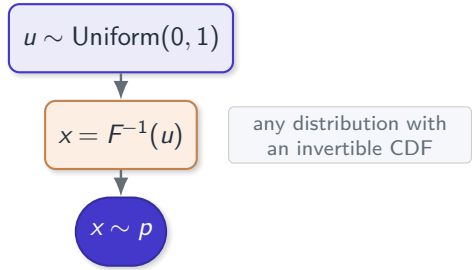


- Gaussian mixture model (GMM): $p_k = \mathcal{N}(\boldsymbol{\mu}_k, \Sigma_k)$ — soft clustering, density estimation (Chapter 8).
- Responsibility $\gamma_k(\mathbf{x}) = p(z = k \mid \mathbf{x}) = \dfrac{\pi_k p_k(\mathbf{x})}{\sum_j \pi_j p_j(\mathbf{x})}$ (Bayes' rule).

Sampling from distributions

Start from uniform random numbers and transform them.

- **Inverse CDF**: $u \sim \mathrm{U}(0,1)$, $x = F^{-1}(u)$. Exponential: $x = -\log(1-u)/\lambda$.
- **Gaussian**: Box–Muller, or $\mathbf{x} = \boldsymbol{\mu} + L\mathbf{z}$ with $\Sigma = LL^\top$.
- **Categorical**: cumulative sums + one uniform.
- **Rejection / importance sampling**: when only an unnormalized density is available.
- **MCMC**: Metropolis–Hastings, Gibbs — for posteriors.



Seeds

Pseudo-random generators are deterministic given a seed — fix it for reproducibility (`np.random.default_rng(42)`).

Key formulas

- Bernoulli/Categorical NLL = cross-entropy; Gaussian NLL = squared loss; Laplace NLL = absolute loss; Laplace prior = ℓ_1 penalty.
- $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$: $p \propto |\Sigma|^{-1/2} \exp(-\frac{1}{2} d_M^2)$; closed under marginals, conditionals ($\boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$) and linear maps.
- Exponential family $h(\mathbf{x}) \exp(\boldsymbol{\eta}^\top T(\mathbf{x}) - A(\boldsymbol{\eta}))$; $\nabla A = \mathbb{E}[T]$, A convex $\Rightarrow$ convex MLE; GLMs set $\boldsymbol{\eta} = \mathbf{w}^\top \mathbf{x}$.
- Beta/Dirichlet are conjugate to Bernoulli/Categorical: smoothing = pseudo-counts.
- Mixtures $\sum_k \pi_k p_k$; responsibilities by Bayes' rule.

Statistics and Estimation: MLE, MAP, Bayes

Section 6 — Chapter 2

The estimation problem

Guess the unknown parameters behind the data; four classic ways.

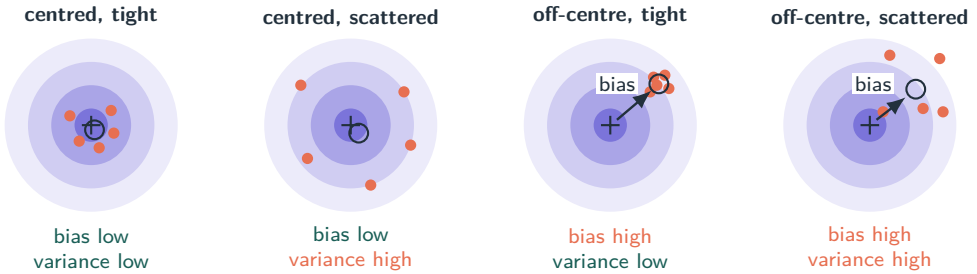
Setting

Data $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ i.i.d. from $p(\mathbf{x} \mid \boldsymbol{\theta}^*)$ with unknown $\boldsymbol{\theta}^*$. An **estimator** $\hat{\boldsymbol{\theta}}(\mathcal{D})$ is any function of the data. Goal: $\hat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^*$.

Maximum likelihood	**MAP**	**Full Bayesian**	**Method of moments**
$\arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta})$	$\arg\max_{\boldsymbol{\theta}} p(\mathcal{D} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})$	keep $p(\boldsymbol{\theta} \mid \mathcal{D})$, integrate	match sample moments
point estimate, no prior	point estimate + regularization	distribution: uncertainty	simple, sometimes inefficient

- This section: how each is derived, how they relate to losses and regularizers, and how good they are (bias, variance, consistency).

Picture: bias and variance as darts



- Each dart is a guess made from a different data set. The centre of the board is the true value.
- **Bias**: how far the *average* of the darts (the hollow circle) is from the centre. **Variance**: how *spread out* the darts are.

In one sentence

A good guessing recipe (an estimator) is both centred and tight; the total error adds up the two faults.

Bias, variance and MSE of an estimator

Bias = systematically off; variance = jumping around. Error is the sum.

Definitions

$$\text{Bias}(\hat{\theta}) = \mathbb{E}_{\mathcal{D}}[\hat{\theta}] - \theta^*$$

$$\text{Var}(\hat{\theta}) = \mathbb{E}_{\mathcal{D}}\big[(\hat{\theta} - \mathbb{E}\hat{\theta})^2\big]$$

$$\text{MSE}(\hat{\theta}) = \mathbb{E}_{\mathcal{D}}\big[(\hat{\theta} - \theta^*)^2\big] = \text{Bias}^2 + \text{Var}$$

- **Consistent**: $\hat{\theta} \to \theta^*$ as $n \to \infty$.
- Unbiased is not always best: a biased estimator with much smaller variance can have lower MSE (shrinkage, ridge).

Sample mean and variance

$\bar{x} = \frac{1}{n}\sum x_i$: unbiased, $\text{Var} = \sigma^2/n$, consistent.

$\hat{\sigma}^2_{\text{MLE}} = \frac{1}{n}\sum(x_i - \bar{x})^2$: $\mathbb{E} = \frac{n-1}{n}\sigma^2$, *biased* (downwards) but consistent.

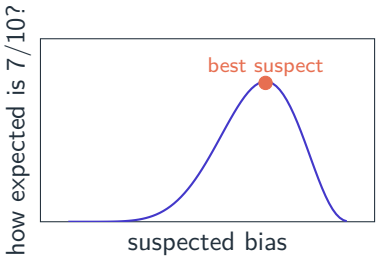
$s^2 = \frac{1}{n-1}\sum(x_i - \bar{x})^2$: unbiased (Bessel's correction).

Same decomposition as Chapter 1

Bias–variance for predictors is this identity applied to $\hat{f}(\mathbf{x})$.

Intuition: detective work

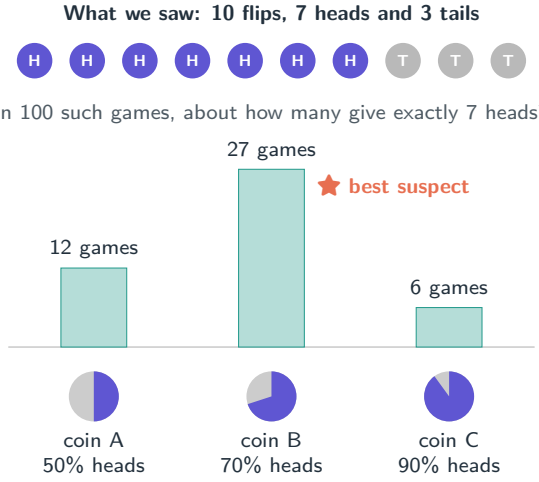
- Someone flips a coin ten times and you see 7 heads. You do not know the coin. What is your best guess for its bias?
- Play detective: if the coin were fair, 7 heads is a bit unusual. If it were "70% heads", 7 heads is the most expected result. If it were "100% heads", 7 heads is impossible.
- **Maximum likelihood** chooses the suspect under which what you saw is the least surprising: here, bias 0.7.
- The same idea fits a line through points: among all lines, choose the one under which the observed points are the most probable. With bell-shaped noise, that is the line with the smallest squared errors.



In one sentence

Pick the explanation that makes what you actually saw the most likely.

Picture: three suspect coins



- Someone flipped a coin 10 times: 7 heads, 3 tails. Which coin was it?
- Three suspects: coin A lands heads 50% of the time, coin B 70%, coin C 90%.
- Play 100 games of 10 flips with each coin. Count the games with exactly 7 heads: A about 12, B about 27, C only 6.

In one sentence

Pick the coin that makes what we saw the most common: coin B. That is maximum likelihood.

Pick the parameters that make the observed data most probable.

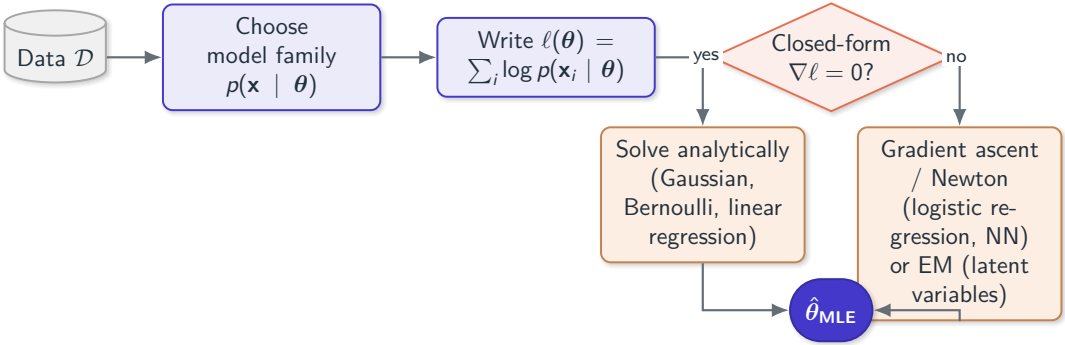
Likelihood and log-likelihood

$$L(\boldsymbol{\theta}) = p(\mathcal{D} \mid \boldsymbol{\theta}) = \prod_{i=1}^{n} p(\mathbf{x}_i \mid \boldsymbol{\theta}), \qquad \ell(\boldsymbol{\theta}) = \log L(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p(\mathbf{x}_i \mid \boldsymbol{\theta})$$

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}} \ \ell(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{\theta}} \underbrace{-\frac{1}{n} \sum_{i} \log p(\mathbf{x}_i \mid \boldsymbol{\theta})}_{\text{negative log-likelihood (NLL)}}$$

- Pick the parameters under which the observed data are most probable.
- Log turns products into sums (numerical stability, gradients decompose per sample).
- NLL is a loss: MLE is ERM with $\ell = -\log p$. Choosing a noise model = choosing a loss.

Log-likelihood, gradient, set to zero; otherwise use an optimizer.



Worked example: MLE for a coin

The most likely bias of a coin is the fraction of heads you saw.

Flips $x_1, \dots, x_n \in \{0, 1\}$ with $k = \sum_i x_i$ heads.

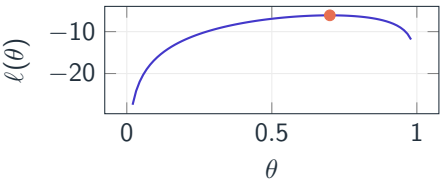
Step 1. Log-likelihood: each head contributes $\log \theta$, each tail $\log(1-\theta)$:
$\ell(\theta) = k \log \theta + (n-k) \log(1-\theta)$.

Step 2. Derivative: $\ell'(\theta) = \dfrac{k}{\theta} - \dfrac{n-k}{1-\theta}$.

Step 3. Set to zero: $k(1-\theta) = (n-k)\theta$, hence $\hat{\theta}_{\text{MLE}} = k/n$.

Step 4. Check it is a maximum: $\ell''(\theta) = -\dfrac{k}{\theta^2} - \dfrac{n-k}{(1-\theta)^2} < 0$ ✔

$n = 10,\ k = 7$



Small-sample pathology

$n = 3,\ k = 3$ gives $\hat{\theta} = 1$: the MLE claims tails are impossible. Naive Bayes hits this with unseen words. Fix: a prior (MAP) or pseudo-counts.

Worked example: MLE for a Gaussian

Most likely centre = sample mean; most likely width = sample spread.

$x_i \sim \mathcal{N}(\mu, \sigma^2)$ i.i.d.

Step 1. Log-likelihood (sum of the log of the bell curve over samples):

$$\ell(\mu, \sigma^2) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (x_i - \mu)^2.$$

Step 2. Derivative in μ: $\dfrac{\partial \ell}{\partial \mu} = \dfrac{1}{\sigma^2}\sum_i (x_i - \mu) = 0$, hence $\hat{\mu} = \dfrac{1}{n}\sum_i x_i = \bar{x}$.

Step 3. Derivative in σ^2: $-\dfrac{n}{2\sigma^2} + \dfrac{1}{2\sigma^4}\sum_i (x_i - \mu)^2 = 0$, hence $\hat{\sigma}^2 = \dfrac{1}{n}\sum_i (x_i - \bar{x})^2$.

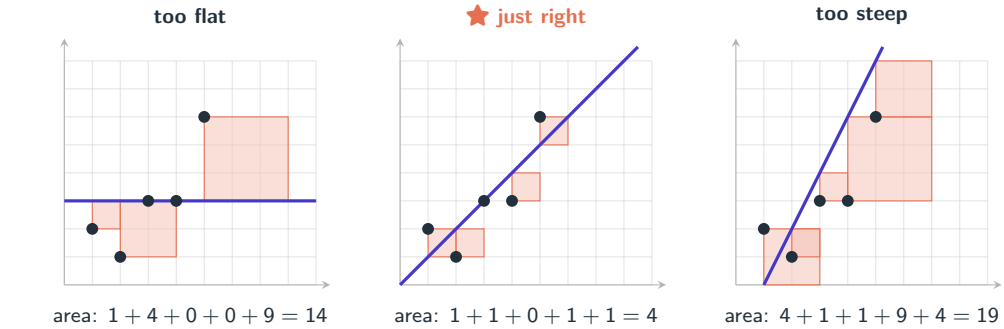
Multivariate version

Same answers with vectors: $\hat{\boldsymbol{\mu}} = \frac{1}{n}\sum_i \mathbf{x}_i$ and $\hat{\Sigma} = \frac{1}{n}\sum_i (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^\top$.

Used by

LDA/QDA and Gaussian Naive Bayes (one $\hat{\boldsymbol{\mu}}_k, \hat{\Sigma}_k$ per class), the M-step of GMMs, anomaly detection.

Picture: which line fits best?



- Five points, three lines. The up-or-down gap from a point to the line is the side of a square (gap 3, area 9).
- Add the areas: flat line 14, steep line 19, middle line 4. The smallest total wins.

In one sentence

The best line is the one whose squares have the smallest total area: the least-squares line. When the misses follow a bell curve, it is also the most likely line.

Gaussian noise around a line, most likely line: least squares.

Model: $y_i = \mathbf{w}^\top \mathbf{x}_i + \varepsilon_i$ with $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$, i.e. $y_i \mid \mathbf{x}_i \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \sigma^2)$.

Step 1. Log-likelihood: $\ell(\mathbf{w}) = -\frac{n}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$.

Step 2. The first term does not depend on $\mathbf{w}$; maximizing ℓ means minimizing the sum of squared errors.

Step 3. So $\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \|\mathbf{y} - X\mathbf{w}\|^2$, whose solution (Section 3) is $\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}$.

Noise model ↔ loss

Gaussian → squared; Laplace → absolute; Student-t → robust; Bernoulli → cross-entropy; Poisson → Poisson deviance.

Bonus

$\hat{\sigma}^2 = \frac{1}{n}\|\mathbf{y} - X\hat{\mathbf{w}}\|^2$ is the residual variance, and $\hat{\mathbf{w}} \sim \mathcal{N}(\mathbf{w}^*, \sigma^2 (X^\top X)^{-1})$ gives standard errors for the coefficients.

Properties of the MLE (optional)

With enough data the MLE is right, Gaussian, and unbeatable.

Under regularity conditions, as $n \to \infty$:

- **Consistent**: $\hat{\theta}_{\text{MLE}} \to \theta^*$.
- **Asymptotically normal**: $\sqrt{n}(\hat{\theta} - \theta^*) \xrightarrow{d} \mathcal{N}\big(\mathbf{0}, I(\theta^*)^{-1}\big)$.
- **Asymptotically efficient**: attains the Cramér–Rao lower bound $\text{Var}(\hat{\theta}) \geq \frac{1}{n} I(\theta^*)^{-1}$.

Fisher information

$$I(\boldsymbol{\theta}) = \mathbb{E}\big[\nabla \log p(\mathbf{x} \mid \boldsymbol{\theta}) \nabla \log p(\mathbf{x} \mid \boldsymbol{\theta})^\top\big] = -\mathbb{E}\big[\nabla^2 \log p(\mathbf{x} \mid \boldsymbol{\theta})\big]$$

Curvature of the log-likelihood: sharp peak $\Rightarrow$ precise estimate.

Bernoulli

$I(\theta) = \dfrac{1}{\theta(1-\theta)}$, so $\text{Var}(\hat{\theta}) \approx \dfrac{\theta(1-\theta)}{n}$ — matches the Binomial variance exactly.

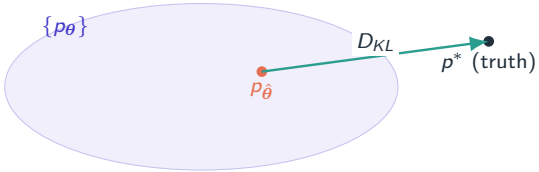
MLE picks the model closest to the data distribution.

Let $\hat{p}_{\mathcal{D}}$ be the empirical distribution ($\frac{1}{n}$ mass on each $\mathbf{x}_i$). Then

$$-\frac{1}{n}\ell(\boldsymbol{\theta}) = -\mathbb{E}_{\hat{p}_{\mathcal{D}}}[\log p(\mathbf{x} \mid \boldsymbol{\theta})] = D_{KL}\big(\hat{p}_{\mathcal{D}} \,\|\, p_{\boldsymbol{\theta}}\big) + \underbrace{H(\hat{p}_{\mathcal{D}})}_{\text{const}}$$

Interpretation

MLE = choosing the model closest to the data distribution in KL divergence (Section 7). As $n \to \infty$, it minimizes $D_{KL}(p^* \| p_{\boldsymbol{\theta}})$: if the model family is misspecified, MLE finds the *best approximation*, not the truth.



Intuition: what you believed before

- A pocket coin, three flips, three heads. Pure detective work says "this coin always lands heads". You do not believe it: pocket coins are usually nearly fair.
- That belief is the **prior**. Mixing it with the data gives a sensible guess, around 70–80% heads, not 100%.
- The simple way to add a prior: pretend you also saw one imaginary head and one imaginary tail; three real heads then give $4/5 = 0.8$. This is *Laplace smoothing*.
- For a model with many weights, the prior "weights are probably small" becomes a penalty on large weights: **regularization**.

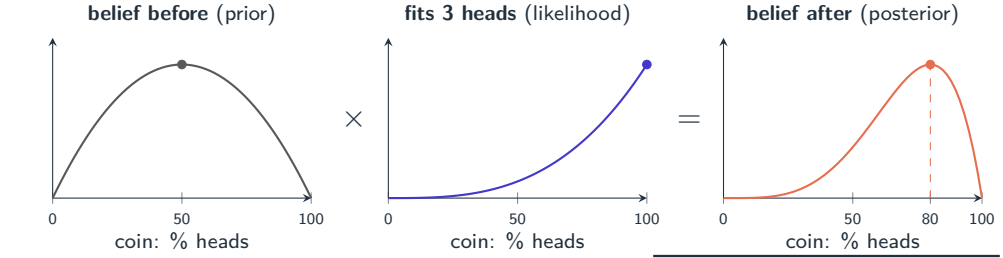
In one sentence

MAP = detective work plus what you believed beforehand; the prior is the regularizer.

Three guesses for the same three heads

- Data only (MLE): $3/3 = 1.00$
- One imaginary head and tail: $4/5 = 0.80$
- Strong belief in fairness: close to 0.5

Picture: prior times likelihood



- Before any flip, I believe a pocket coin is near 50% heads: the grey hump.
- How well does a coin explain 3 heads in a row? An 80% coin: $0.8 \times 0.8 \times 0.8 = 0.51$. A 100% coin: 1. The blue curve climbs.
- Multiply the two curves point by point: the red curve. It peaks at 80%, not 100%.

coin	50%	80%	100%
belief before	1	0.64	0
fits 3 heads	0.13	0.51	1
multiply	0.13	**0.33**	0

In one sentence

New belief is old belief times how well each coin fits the data. Its peak is the new best guess (MAP).

Maximum a posteriori (MAP)

MLE plus a prior; the prior becomes the regularizer.

MAP

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}} \; p(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \; \big[\log p(\mathcal{D} \mid \boldsymbol{\theta}) + \log p(\boldsymbol{\theta})\big]$$

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\min_{\boldsymbol{\theta}} \; \underbrace{-\sum_i \log p(\mathbf{x}_i \mid \boldsymbol{\theta})}_{\text{data fit (NLL)}} \; \underbrace{-\log p(\boldsymbol{\theta})}_{\text{regularizer}}$$

Prior $\leftrightarrow$ regularizer

Prior $p(\mathbf{w})$	$-\log p(\mathbf{w})$	Name
$\mathcal{N}(\mathbf{0}, \tau^2 I)$	$\frac{1}{2\tau^2}\|\mathbf{w}\|_2^2 + \text{c}$	ridge / weight decay, $\lambda = \sigma^2/\tau^2$
Laplace$(0, b)$	$\frac{1}{b}\|\mathbf{w}\|_1 + \text{c}$	Lasso

A Beta prior = a few imaginary flips; this is Laplace smoothing.

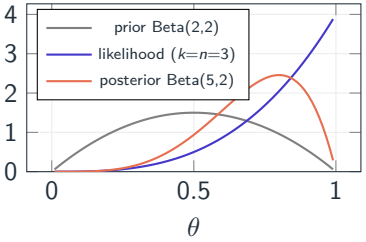
Prior $\theta \sim \text{Beta}(\alpha, \beta)$; data: k heads in n flips.

Step 1. Log-posterior = log-likelihood + log-prior:
$k \log\theta + (n-k)\log(1-\theta) + (\alpha-1)\log\theta + (\beta-1)\log(1-\theta)$.

Step 2. It looks like the coin log-likelihood with $k+\alpha-1$ heads and $n-k+\beta-1$ tails.

Step 3. So the same recipe gives $\hat{\theta}_{\text{MAP}} = \dfrac{k+\alpha-1}{n+\alpha+\beta-2}$.

- $\alpha = \beta = 1$ (flat prior): MAP = MLE $= k/n$.
- $\alpha = \beta = 2$: $\hat{\theta} = \frac{k+1}{n+2}$, Laplace smoothing: one imaginary head and one imaginary tail.
- $n = 3$, $k = 3$: MLE $= 1$, MAP $= 0.8$. Sensible.



A Gaussian prior on the weights adds a squared penalty: ridge.

Likelihood $y_i \mid \mathbf{x}_i \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \sigma^2)$; prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \tau^2 I)$.

Step 1. Negative log-posterior = data term + prior term:

$$-\log p(\mathbf{w} \mid \mathcal{D}) = \frac{1}{2\sigma^2} \|\mathbf{y} - X\mathbf{w}\|^2 + \frac{1}{2\tau^2} \|\mathbf{w}\|^2 + \text{const.}$$

Step 2. Gradient (two table rules): $-\frac{1}{\sigma^2} X^\top (\mathbf{y} - X\mathbf{w}) + \frac{1}{\tau^2}\mathbf{w}$.

Step 3. Set to zero and solve: $\hat{\mathbf{w}}_{\text{MAP}} = \left(X^\top X + \frac{\sigma^2}{\tau^2} I\right)^{-1} X^\top \mathbf{y} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$.

Reading $\lambda = \sigma^2/\tau^2$

Noisy data (large σ^2) or a tight prior (small τ^2) means strong regularization. The hyperparameter you tune by cross-validation has a probabilistic meaning.

Full Bayesian inference

Keep all plausible parameters and average their predictions.

Keep the whole posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ and *integrate* over it:

Posterior predictive

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

- Averages predictions of all plausible models, weighted by their posterior probability.
- Provides calibrated uncertainty: wide posterior $\Rightarrow$ wide predictive.

Cost

The integral is intractable except for conjugate models (Beta–Bernoulli, Gaussian–Gaussian). Approximations: Laplace, variational inference (ELBO), MCMC, deep ensembles, MC-dropout.

Bayesian linear regression

Gaussian prior + Gaussian likelihood $\Rightarrow$ Gaussian posterior $\mathcal{N}(\hat{\mathbf{w}}_{\text{MAP}}, (\frac{1}{\sigma^2} X^\top X + \frac{1}{\tau^2} I)^{-1})$; predictive variance grows away from the data. Gaussian processes (Chapter 7) generalize this.

MLE vs. MAP vs. Bayes: comparison

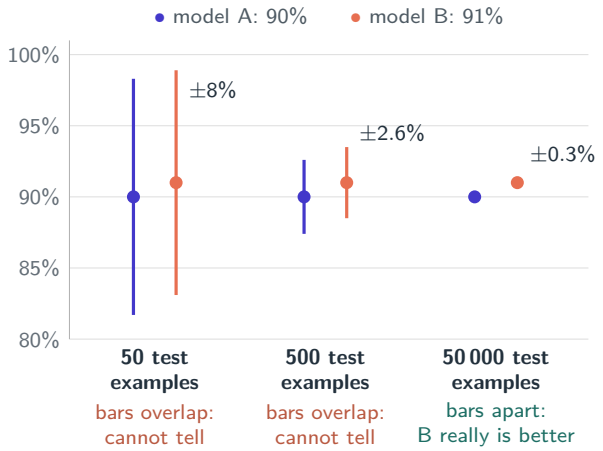
Little data: the three differ; lots of data: they agree.

	MLE	MAP	Bayesian
Output	point $\hat{\boldsymbol{\theta}}$	point $\hat{\boldsymbol{\theta}}$	distribution $p(\boldsymbol{\theta} \mid \mathcal{D})$
Prior	none (uniform)	yes	yes
Regularization	no	$-\log p(\boldsymbol{\theta})$	implicit (averaging)
Overfits with small n	yes	less	least
Uncertainty	asymptotic (Fisher)	no	yes, exact/approximate
Compute	optimization	optimization	integration (hard)
Large n	all three agree		

Practical stance in this course

Most algorithms are MLE/MAP (= regularized ERM). We use the Bayesian view when it gives insight (why ℓ_2? why smoothing?) and in Gaussian processes.

Picture: more data, shorter error bars



- Model A scores 90%, model B 91% on the same test examples. Is B really better?
- The bar shows where the true score (on endless examples) probably is.
- With 50 examples the bar is $\pm 8\%$: bars overlap, we cannot tell. With 50 000 it is $\pm 0.3\%$: bars apart, B really is better.

In one sentence

No error bar, no result. Ten times more data: a bar three times shorter.

Confidence intervals and tests

A score without an error bar is not a result; bars shrink like $1/\sqrt{n}$.

- 95% CI for a mean: $\bar{x} \pm 1.96\, s/\sqrt{n}$ (CLT) — e.g. for a CV score.
- CI for an accuracy $\hat{p}$: $\hat{p} \pm 1.96\sqrt{\hat{p}(1-\hat{p})/n}$.
- Bootstrap CI: resample the test set B times, recompute the metric, take the 2.5% and 97.5% quantiles — works for any metric (AUC, F1).
- Comparing two models on the same folds: paired t-test on per-fold differences, or a sign test.
- p-value < 0.05 is not proof; report effect sizes and intervals.
- Multiple comparisons: testing 50 configurations, some will look significant by chance — Bonferroni or a held-out confirmation.

Numbers

Two models score 0.90 and 0.91 on $n = 500$ test samples: SE ≈ 0.013 each, the difference is within noise. With $n = 50\,000$, SE ≈ 0.0013: the difference is real.

Guess the hidden part, refit, repeat: that is EM.

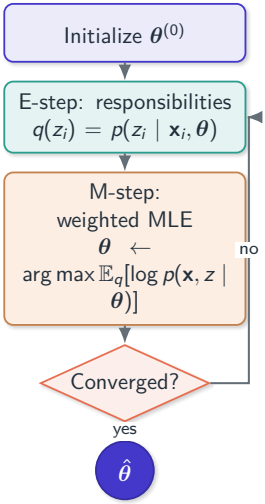
If each $\mathbf{x}_i$ has an unobserved z_i (cluster, topic, hidden state), the log-likelihood

$$\ell(\boldsymbol{\theta}) = \sum_i \log \sum_{z_i} p(\mathbf{x}_i, z_i \mid \boldsymbol{\theta})$$

has a *log of a sum*: no closed form.

Expectation–Maximization

- E-step: compute $q(z_i) = p(z_i \mid \mathbf{x}_i, \boldsymbol{\theta}^{(t)})$
- M-step: $\boldsymbol{\theta}^{(t+1)} = \arg\max_{\boldsymbol{\theta}} \sum_i \mathbb{E}_q[\log p(\mathbf{x}_i, z_i \mid \boldsymbol{\theta})]$



Key formulas

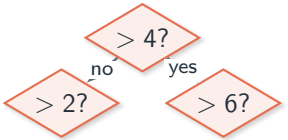
- $\mathrm{MSE}(\hat{\theta}) = \mathrm{Bias}^2 + \mathrm{Var}$; unbiased $\neq$ best.
- MLE: $\hat{\boldsymbol{\theta}} = \arg\max \sum_i \log p(\mathbf{x}_i \mid \boldsymbol{\theta})$; NLL is a loss; Gaussian noise $\Rightarrow$ least squares; Bernoulli $\Rightarrow$ cross-entropy.
- Bernoulli MLE k/n; Gaussian MLE $\bar{x}$, $\frac{1}{n}\sum(x_i - \bar{x})^2$; regression MLE $(X^\top X)^{-1} X^\top \mathbf{y}$.
- MLE is consistent, asymptotically normal with covariance $I(\boldsymbol{\theta})^{-1}/n$; MLE minimizes $D_{KL}(\hat{p}_{\mathcal{D}} \| p_{\boldsymbol{\theta}})$.
- MAP $= \arg\min$ NLL $- \log p(\boldsymbol{\theta})$: Gaussian prior $\Rightarrow$ ridge with $\lambda = \sigma^2/\tau^2$; Laplace prior $\Rightarrow$ Lasso; Beta prior $\Rightarrow$ pseudo-counts.
- Bayesian: integrate over the posterior; exact only for conjugate models.

Information Theory

Section 7 — Chapter 2

Intuition: information is surprise

- "The sun rose this morning" carries no information: you knew it would. "It snowed in the Sahara" carries a lot: it is surprising. **Information = surprise**, and rare events are more surprising.
- **Entropy** is the average surprise of a random thing, or equivalently how many yes/no questions you need, on average, to find out what happened.
- The game of 20 questions: to guess a number between 1 and 8 you need 3 questions ("is it bigger than 4?" and so on). Entropy of a fair 8-sided die: 3 bits.
- Decision trees ask the question that reduces the surprise the most.



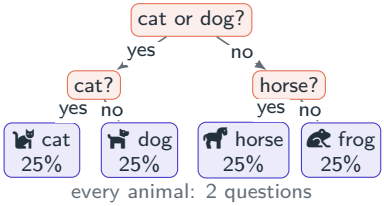
3 questions for 8 possibilities = 3 bits

In one sentence

Entropy counts how many yes/no questions, on average, you need to learn the outcome.

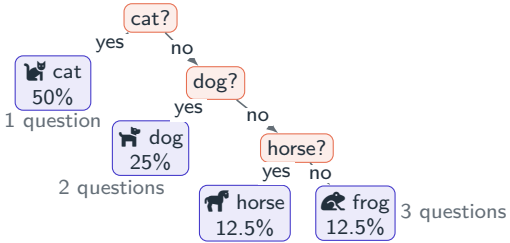
Picture: counting yes/no questions

Four animals, each 25%



average = 2 questions = 2 bits

The cat is very likely: ask "cat?" first



average = 1.75 questions = 1.75 bits

- Left: all four animals are equally likely, so you always need 2 questions. One yes/no question is one **bit**.
- Right: half of the time the animal is the cat, so ask "cat?" first. Over 8 rounds (4 cat, 2 dog, 1 horse, 1 frog) you ask $4\times1 + 2\times2 + 1\times3 + 1\times3 = 14$ questions: $14 \div 8 = 1.75$ per round.

In one sentence

Entropy is the average number of yes/no questions you need; when one outcome is very likely, you need fewer.

How unpredictable a variable is: zero when certain, maximal when uniform.

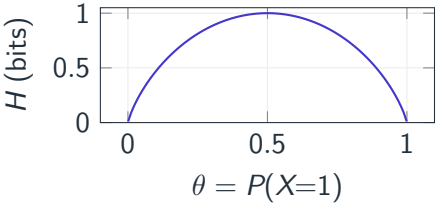
Self-information and entropy (Shannon, 1948)

$$I(x) = -\log p(x) \quad \text{(surprise of outcome } x\text{)}, \qquad H(X) = \mathbb{E}[I(X)] = -\sum_x p(x) \log p(x)$$

Units: bits ($\log_2$) or nats (ln).

- $H(X) \geq 0$; $H = 0$ iff deterministic; maximal $= \log K$ for the uniform over K outcomes.

Bernoulli entropy



Worked example: entropy of class proportions

A pure node has entropy 0; a 50/50 node has 1 bit.

Step 1. Take the proportions, e.g. $(0.9, 0.1)$.

Step 2. For each class compute $-p \log_2 p$: $-0.9 \log_2 0.9 = 0.137$ and $-0.1 \log_2 0.1 = 0.332$.

Step 3. Add them: $H = 0.469$ bits.

Class proportions	H (bits)	Gini $1 - \sum p_k^2$
$(1, 0)$	0	0
$(0.9, 0.1)$	0.469	0.18
$(0.5, 0.5)$	1	0.5
$(1/3, 1/3, 1/3)$	1.585	0.667

Decision trees (Chapter 6)

Entropy measures the *impurity* of a node. A split is chosen to maximize the information gain

$$\text{IG} = H(\text{parent}) - \sum_c \frac{n_c}{n} H(\text{child}_c),$$

the expected reduction in entropy of the label.

Differential entropy and the Gaussian (optional)

Entropy for continuous variables; the Gaussian is the most random for its spread.

Differential entropy

$h(X) = -\int p(x) \log p(x)\, dx$ — can be negative, not invariant to rescaling, but differences/KL remain meaningful.

Gaussian

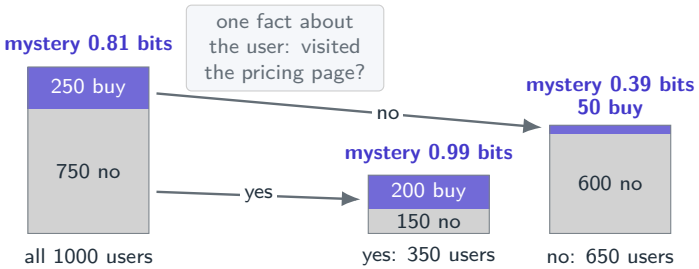
$h(\mathcal{N}(\mu, \sigma^2)) = \frac{1}{2} \log(2\pi e \sigma^2)$

$h(\mathcal{N}(\boldsymbol{\mu}, \Sigma)) = \frac{1}{2} \log\big((2\pi e)^d\, |\Sigma|\big)$

Maximum entropy principle

Among all densities with a given mean and covariance, the Gaussian has maximal entropy: it is the "least committed" choice. Likewise: uniform (bounded support), exponential (positive with given mean). This justifies default modelling choices.

Picture: a feature that saves questions



Mystery after the split
(650 of the 1000 users are in the "no" pile, 350 in the "yes" pile)
$0.65 \times 0.39 + 0.35 \times 0.99$
$= 0.60$ bits are left

Saved: $0.81 - 0.60 = \mathbf{0.21}$ bits
= the mutual information

- Mystery = entropy: how many yes/no questions, on average, you still need to know if a user buys. One pile, 250 of 1000 buy: 0.81 bits.
- Split the pile by one fact about the user (a **feature**). The "no" pile is almost pure, so little mystery is left; the "yes" pile is mixed. Average by pile size: 0.60 bits left, 0.21 saved.

In one sentence

Mutual information is the part of the mystery that disappears once you know the feature: here 0.21 bits out of 0.81.

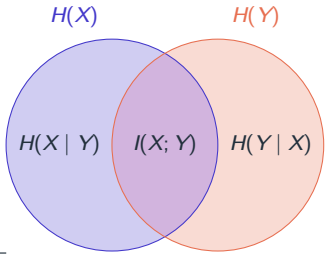
Joint, conditional entropy and mutual information

How much uncertainty about Y disappears once you know X.

- Conditional entropy $H(Y \mid X)$: the uncertainty left in Y after seeing X.
- Chain rule: $H(X, Y) = H(X) + H(Y \mid X)$.
- Mutual information:

$$I(X; Y) = H(Y) - H(Y \mid X) = \sum_{x,y} p(x, y) \log \frac{p(x, y)}{p(x)\, p(y)}$$

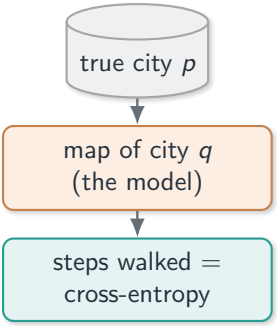
- $I(X; Y) \geq 0$; $= 0$ iff $X \perp Y$; symmetric; captures *any* dependence, not only linear.



The overlap of the two circles is the shared information.

Intuition: the wrong map

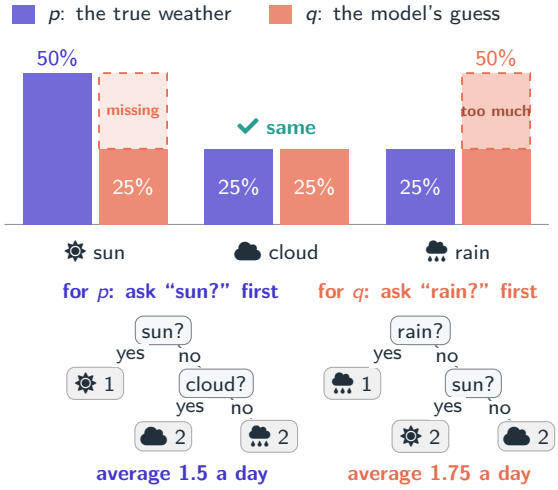
- You walk around city p using a map of city q. Some streets match, others do not, so you take extra steps.
- The **KL divergence** is the average number of extra steps caused by the wrong map: zero only if the maps agree, and not symmetric (a city map in a village wastes a different amount than the reverse).
- **Cross-entropy** is the total steps: the unavoidable steps of city p itself (its entropy) plus the extra ones (the KL).
- A classifier's probabilities are its map; the true label is the city. Training with cross-entropy redraws the map until the extra steps are as few as possible.



In one sentence

KL = extra steps caused by a wrong map; cross-entropy = all the steps; training shrinks the extra ones.

Picture: p and q as two bar charts



- Blue: the true weather p. Coral: the model's guess q; it bets on rain. They disagree on sun and on rain.
- Below: the question tree built for each chart; each leaf shows how many questions that kind of day costs.
- Over 4 true days (2 sun, 1 cloud, 1 rain), tree for p: $2\times1+1\times2+1\times2=6$ questions, 1.5 a day. Tree for q: $2\times2+1\times2+1\times1=7$, 1.75 a day. The extra 0.25 is the KL divergence.

In one sentence

KL counts the extra questions you pay because the model's chart differs from the true chart; zero only when they match.

How different two distributions are, seen from the first one; not symmetric.

Kullback–Leibler divergence

$$D_{KL}(p \,\|\, q) = \sum_x p(x) \log \frac{p(x)}{q(x)} = \mathbb{E}_p\Big[\log \frac{p(X)}{q(X)}\Big]$$

- $D_{KL} \geq 0$ (Gibbs' inequality, via Jensen), $= 0$ iff $p = q$.
- Not symmetric, no triangle inequality: a *divergence*, not a distance.
- Expected extra bits when coding p-data with a code optimized for q.
- Infinite if $q(x) = 0$ where $p(x) > 0$ (*support mismatch*).

Gaussians

$$D_{KL}\big(\mathcal{N}(\mu_1, \sigma_1^2) \| \mathcal{N}(\mu_2, \sigma_2^2)\big) = \log \tfrac{\sigma_2}{\sigma_1} + \tfrac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \tfrac{1}{2}$$

With $\mathcal{N}(\mu, \sigma^2)$ vs. $\mathcal{N}(0, 1)$: $\frac{1}{2}\big(\sigma^2 + \mu^2 - 1 - \log \sigma^2\big)$ — the VAE regularizer (Chapter 10).

Proof of $D_{KL} \geq 0$ (Gibbs' inequality)

One trick: the average of a log is at most the log of the average (Jensen).

Step 1. Write minus the KL as an expectation under p: $-D_{KL}(p\|q) = \mathbb{E}_p\big[\log \frac{q(X)}{p(X)}\big]$.

Step 2. Jensen for the concave log: $\mathbb{E}_p[\log Z] \leq \log \mathbb{E}_p[Z]$ with $Z = q/p$.

Step 3. Compute the right-hand side: $\mathbb{E}_p[q/p] = \sum_x p(x)\frac{q(x)}{p(x)} = \sum_x q(x) = 1$, and $\log 1 = 0$.

Step 4. Hence $-D_{KL} \leq 0$, i.e. $D_{KL}(p\|q) \geq 0$, with equality only when q/p is constant, i.e. $p = q$.

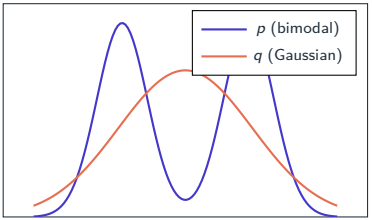
Three consequences

- Cross-entropy is never below entropy: $H(p, q) = H(p) + D_{KL}(p\|q) \geq H(p)$.
- Mutual information is never negative.
- The ELBO is a lower bound on the log-evidence (Chapters 8 and 10).

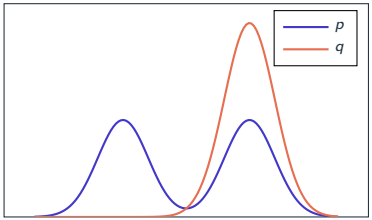
Forward vs. reverse KL (optional)

One direction makes q cover everything; the other makes it pick one bump.

$\min_q D_{KL}(p\|q)$: mean-seeking

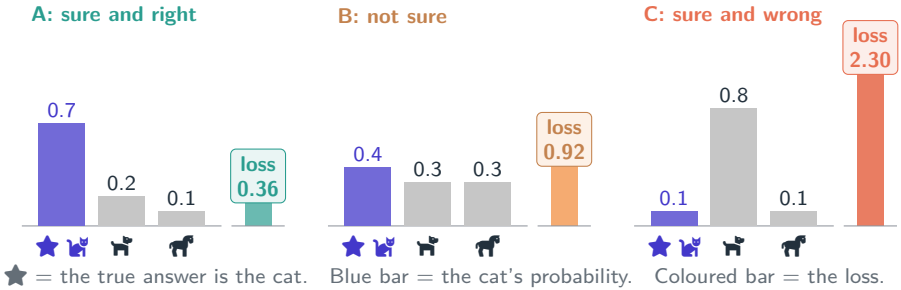


$\min_q D_{KL}(q\|p)$: mode-seeking



- $D_{KL}(p\|q)$ penalizes $q \approx 0$ where $p > 0 \Rightarrow q$ covers everything (MLE uses this direction).
- $D_{KL}(q\|p)$ penalizes $q > 0$ where $p \approx 0 \Rightarrow q$ locks onto one mode (variational inference uses this direction).

Picture: the cost of one prediction



- The model gives a probability to each possible answer (cat, dog, horse); the three bars add up to 1. The true answer is the cat.
- Only the cat bar matters: tall is cheap, short is expensive. The price list is a fixed curve called "minus the log": 0.7 costs 0.36, 0.4 costs 0.92, 0.1 costs 2.30 (six times more).

In one sentence

The cross-entropy loss is the price of the probability you gave to the true answer: cheap when it is high, very expensive when it is near zero.

Cross-entropy: the classification loss

Minus the log-probability of the true class; confident mistakes cost a lot.

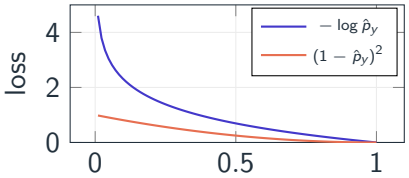
Cross-entropy

$$H(p, q) = -\sum_{x} p(x) \log q(x) = H(p) + D_{KL}(p \| q)$$

For one sample with true label y (one-hot $p = \mathbf{e}_y$) and predicted probabilities $q = \hat{\mathbf{p}}$:

$$H(\mathbf{e}_y, \hat{\mathbf{p}}) = -\log \hat{p}_y$$

- Averaged over the dataset = NLL = what MLE minimizes.



Cross-entropy punishes confident mistakes much harder than squared error: gradient does not vanish.

Worked example: cross-entropy and softmax

Softmax, then minus the log of the true class; gradient = prediction minus target.

Three classes, true label $y = 2$, scores (logits) $\mathbf{z} = (1.0,\ 2.0,\ 0.5)$.

Step 1. Exponentiate: $(e^1, e^2, e^{0.5}) = (2.718,\ 7.389,\ 1.649)$, sum $= 11.756$.

Step 2. Normalize (softmax): $\hat{\mathbf{p}} = (0.231,\ 0.629,\ 0.140)$.

Step 3. Loss: $L = -\log \hat{p}_2 = -\log 0.629 = 0.464$ nats.

Step 4. Gradient w.r.t. the scores: $\nabla_{\mathbf{z}} L = \hat{\mathbf{p}} - \mathbf{e}_y = (0.231,\ -0.371,\ 0.140)$.

The gradient is "prediction minus target"

Softmax followed by cross-entropy always gives $\partial L / \partial \mathbf{z} = \hat{\mathbf{p}} - \mathbf{e}_y$, the same pattern as least squares and logistic regression. Frameworks fuse the two (`CrossEntropyLoss` takes logits) for numerical stability.

Log-sum-exp and numerical stability

Subtract the largest score before exponentiating; nothing overflows.

$$\log \sum_k e^{z_k} = m + \log \sum_k e^{z_k - m}, \qquad m = \max_k z_k$$

- Computing e^{1000} overflows; subtracting the max keeps all exponents ≤ 0.
- $\log \mathrm{softmax}(\mathbf{z})_k = z_k - \mathrm{LSE}(\mathbf{z})$: compute log-probabilities directly, never log(softmax).
- Same trick for the E-step of GMMs and for HMM forward–backward.

Example

$\mathbf{z} = (1000, 1001)$: naive softmax $= \frac{e^{1000}}{e^{1000}+e^{1001}}$ overflows. With $m = 1001$: $\frac{e^{-1}}{e^{-1}+e^{0}} = 0.269$.

Also

$\log(1 + e^z)$ (softplus) for large z: use $\max(z, 0) + \log(1 + e^{-|z|})$.

Worked example: mutual information

Uncertainty of Y minus the uncertainty left after seeing X.

Target Y (buy / not) and feature X (visited pricing page), counts over 1000 users:

	$Y=0$	$Y=1$	total
$X=0$	600	50	650
$X=1$	150	200	350
total	750	250	1000

Step 1. $H(Y) = H(0.75, 0.25) = 0.811$ bits.

Step 2. Row $X=0$:
$H(Y \mid X{=}0) = H(0.923, 0.077) = 0.391$.

Step 3. Row $X=1$:
$H(Y \mid X{=}1) = H(0.429, 0.571) = 0.985$.

Step 4. Average with the row weights:
$H(Y \mid X) = 0.65 \cdot 0.391 + 0.35 \cdot 0.985 = 0.599$.

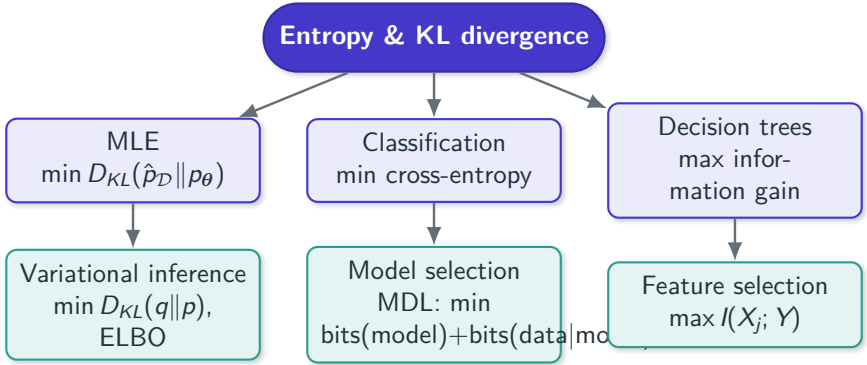
Step 5. $I(X; Y) = 0.811 - 0.599 = \mathbf{0.212}$ bits.

Interpretation

Knowing X removes 26% of the uncertainty about Y. Rank features by $I(X_j; Y)$ (`mutual_info_classif` in scikit-learn); unlike correlation it also detects non-linear relations.

Information-theoretic view of learning

Many methods are one idea: bring the model distribution close to the data.



Key formulas

- Entropy $H(X) = -\sum p \log p$; max $\log K$ at uniform; Gaussian maximizes entropy for given variance.
- $H(X, Y) = H(X) + H(Y \mid X)$;
  $I(X; Y) = H(Y) - H(Y \mid X) = D_{KL}(p(x, y) \| p(x)p(y)) \geq 0$.
- $D_{KL}(p\|q) = \sum p \log \frac{p}{q} \geq 0$, asymmetric; forward KL is mean-seeking, reverse KL mode-seeking.
- Cross-entropy $H(p, q) = H(p) + D_{KL}(p\|q)$; per-sample loss $-\log \hat{p}_y$; softmax + CE gradient $= \hat{\mathbf{p}} - \mathbf{e}_y$.
- Log-sum-exp trick for stability.
- Information gain drives trees; mutual information scores features.

Convex Optimization and Gradient Methods

Section 8 — Chapter 2

Every training algorithm minimizes something; is the landscape a bowl?

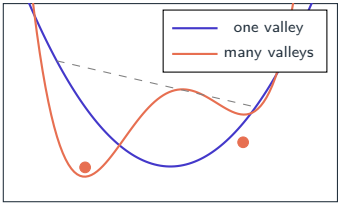
Generic form

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^d} J(\boldsymbol{\theta}) \qquad \text{or} \qquad \min_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \ \text{ s.t. } g_i(\boldsymbol{\theta}) \le 0,\ h_j(\boldsymbol{\theta}) = 0$$

Problem	J	Convex?	Solver
Linear regression	$\|X\mathbf{w}-\mathbf{y}\|^2$	yes (quadratic)	closed form / GD
Ridge	$\|X\mathbf{w}-\mathbf{y}\|^2+\lambda\|\mathbf{w}\|^2$	strongly convex	closed form / GD
Lasso	$\|X\mathbf{w}-\mathbf{y}\|^2+\lambda\|\mathbf{w}\|_1$	convex, non-smooth	coordinate descent, proximal GD
Logistic regression	cross-entropy	convex	GD, Newton / IRLS, L-BFGS
SVM	hinge $+\|\mathbf{w}\|^2$	convex, constrained dual	SMO, QP
Neural networks	any loss	no	SGD, Adam
k-means	within-cluster SS	no (combinatorial)	Lloyd (alternating)

Intuition: convex means one valley

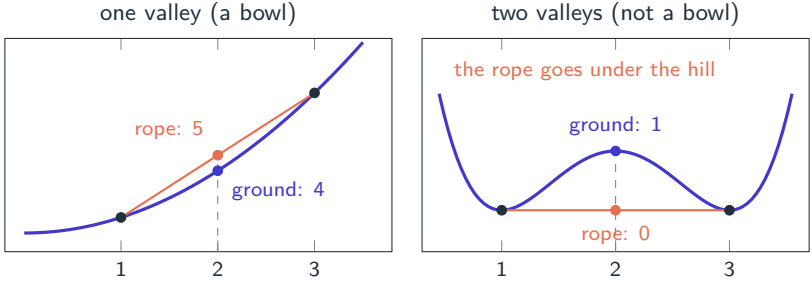
- A **convex** landscape is a single valley, like a bowl. Drop a ball anywhere: it rolls to the one lowest point. No wrong place to end up.
- A **non-convex** landscape is a mountain range with many small valleys. A ball can get stuck in a little dip that is not the lowest point.
- Test for a valley: pick any two points on the ground and stretch a rope between them. In a valley the rope stays *above* the ground everywhere. In a mountain range it cuts through a hill.
- Linear regression, logistic regression and SVMs are single valleys. Neural networks are mountain ranges, and still work surprisingly well.



In one sentence

Convex = one valley: a rope between any two points stays above the ground, and the ball always finds the bottom.

Picture: the rope test with numbers



- Pick two spots on the ground, here at 1 and 3, and stretch a rope between them.
- Bowl: in the middle (at 2) the ground is at height 4 and the rope at 5. The rope stays in the air.
- Two valleys: in the middle the ground is at height 1 and the rope at 0. The rope goes under the hill.

In one sentence

Convex means: a rope between any two points of the ground never goes under the ground. One valley, no hidden dips.

Bowl shaped: the chord between two points of the graph stays above it.

Convex set

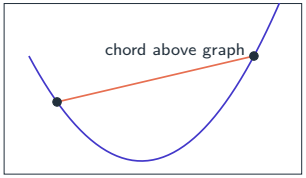
C is convex if $\mathbf{x}, \mathbf{y} \in C \Rightarrow t\mathbf{x} + (1-t)\mathbf{y} \in C$ for all $t \in [0,1]$.

Convex function

f is convex if for all $\mathbf{x}, \mathbf{y}$, $t \in [0,1]$:

$$f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y}).$$

Strictly convex if $<$ for $\mathbf{x} \neq \mathbf{y}$, $t \in (0,1)$.



Examples: $\|\mathbf{x}\|_p$, e^x, $-\log x$, $\mathbf{x}^\top A\mathbf{x}$ $(A \succeq 0)$, $\max_i(\mathbf{a}_i^\top \mathbf{x} + b_i)$, $\log \sum e^{x_i}$.

Tangent below the graph, or curvature never negative.

For differentiable f on a convex domain:

$$\text{First-order:} \quad f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) \quad \forall \mathbf{x}, \mathbf{y} \quad \text{(tangent plane lies below)}$$

$$\text{Second-order:} \quad \nabla^2 f(\mathbf{x}) \succeq 0 \quad \forall \mathbf{x} \quad (\text{PSD Hessian; } \succ 0 \Rightarrow \text{strictly convex})$$

Operations preserving convexity:

- Non-negative sums: $\alpha f + \beta g$
- Composition with affine map: $f(A\mathbf{x} + \mathbf{b})$
- Pointwise maximum: $\max(f, g)$
- Composition $h \circ g$ with h convex non-decreasing, g convex

Logistic loss is convex

$\ell(\mathbf{w}) = \log(1+e^{-y\mathbf{w}^\top \mathbf{x}})$: $s \mapsto \log(1+e^{-s})$ has second derivative $\sigma(s)(1-\sigma(s)) > 0$, composed with the affine $s = y\mathbf{w}^\top \mathbf{x}$. Sum over samples $+ \lambda \|\mathbf{w}\|^2$: convex.

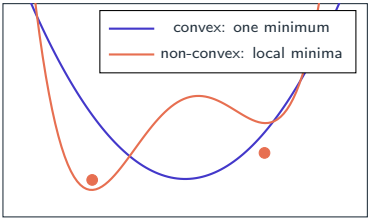
Why convexity matters

One bottom only: walking downhill from anywhere reaches the best solution.

Theorem

For convex f, every local minimum is a global minimum. If f is differentiable, $\nabla f(\boldsymbol{\theta}^*) = \mathbf{0} \iff \boldsymbol{\theta}^*$ is a global minimizer. If strictly convex, the minimizer is unique.

- No spurious local minima, no saddles to worry about.
- Any descent method converges to the optimum; guarantees on the rate.
- Initialization does not matter (for the solution).



Deep learning objectives are non-convex; SGD still works well in practice, for reasons partly understood (over-parameterization, benign landscapes).

Intuition: downhill in the fog, four ways

- **Gradient descent**: feel the slope, step down, repeat; each "feel" looks at *all* the data.
- **Stochastic gradient descent**: ask one random passer-by which way is down: noisy but instant, so a thousand small steps per careful one. A few passers-by averaged = a mini-batch.
- **Momentum**: a heavy ball instead of a walker; it keeps rolling, zig-zags cancel, long slopes speed up.
- **Adam**: adjustable shoes: shorter steps where it is always steep, longer where it rarely is.
- **Newton**: a walker who also feels the curvature and jumps to the bottom of the local bowl; great in small spaces, too costly in huge ones.

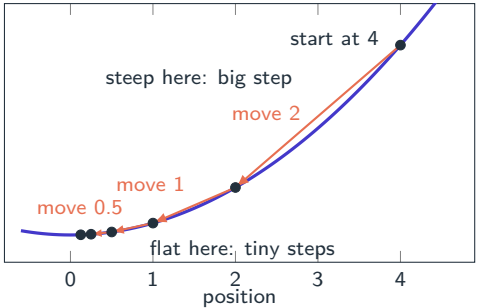
In one sentence

All optimizers walk downhill; they differ in how they feel the slope and how they choose the step.

Step size is everything

Tiny steps: you never arrive. Giant steps: you jump across the valley. Keep the largest step that still goes down.

Picture: five steps down a bowl



position	slope there	move (a quarter of the slope)
4	8	2
2	4	1
1	2	0.5
0.5	1	0.25
0.25	0.5	0.125

- Stand at 4: the ground is steep (slope 8, twice the position). Walk downhill by a quarter of the slope, 2, and land at 2.
- Repeat. Near the bottom the ground is flat, so the moves shrink by themselves: 1, 0.5, 0.25, ...

In one sentence

Measure the slope, step downhill by a fixed fraction of it, repeat. That fraction is the learning rate.

Compute the slope, step downhill, repeat; the step size is the key knob.

Gradient descent (GD)

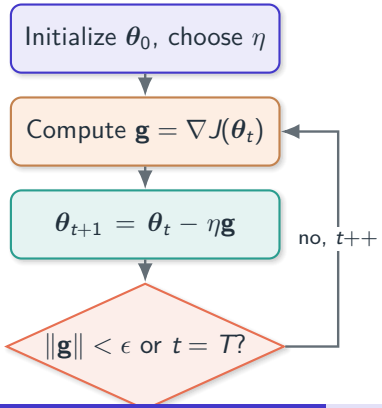
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla J(\boldsymbol{\theta}_t), \qquad \eta > 0 \text{ the learning rate (step size)}$$

Justification (first-order Taylor):

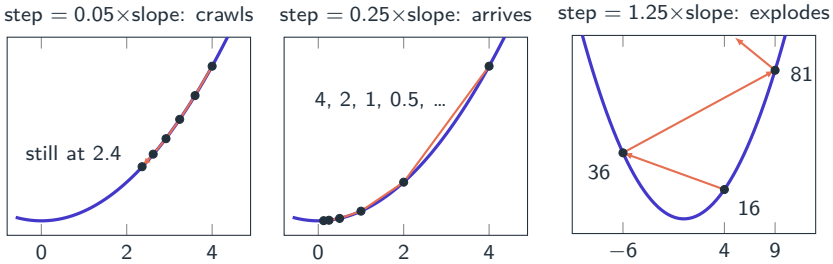
$$J(\boldsymbol{\theta} - \eta \nabla J) \approx J(\boldsymbol{\theta}) - \eta \|\nabla J\|^2 < J(\boldsymbol{\theta})$$

for small η — the steepest-descent direction decreases J fastest per unit step.

- Stop when $\|\nabla J\| < \epsilon$, or after T iterations, or when validation loss stops improving.



Picture: three step sizes



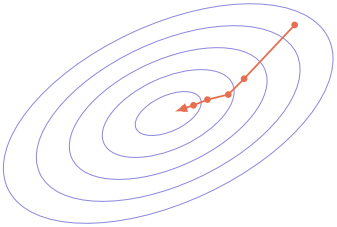
- Same bowl, same start at 4; the three walkers differ only in the fraction of the slope they walk.
- A twentieth (0.05): after 5 steps still at 2.4. A quarter (0.25): $4, 2, 1, 0.5, 0.25$, almost there. More than the whole slope (1.25): 4, then -6, then 9, then -13.5: every jump lands higher, on the other side.

In one sentence

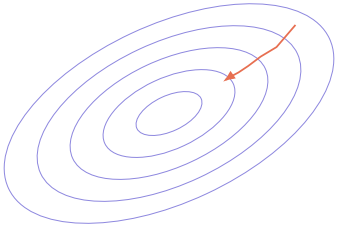
The learning rate is the fraction of the slope you walk: too small, you never arrive; too big, you fly out of the bowl.

Gradient descent in action

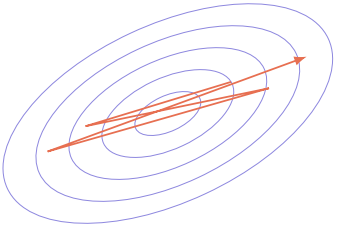
Too small: slow. Too large: bounces out. Right: steady progress.



η well chosen: steady convergence



η too small: slow



η too large: diverges

- The learning rate is the single most important hyperparameter of any gradient method. Scan it on a log grid (10^{-4} to 1) and watch the loss curve.

Steps needed grow with the condition number (steepest over flattest curvature).

Assumptions

J convex, L-smooth ($\|\nabla J(\mathbf{x}) - \nabla J(\mathbf{y})\| \le L\,\|\mathbf{x} - \mathbf{y}\|$, equivalently $\nabla^2 J \preceq L I$).

Theorem

With $\eta = 1/L$: $J(\boldsymbol{\theta}_T) - J^* \le \dfrac{L\,\|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|^2}{2T}$ — rate $O(1/T)$.

If moreover J is μ-strongly convex ($\nabla^2 J \succeq \mu I$): $\|\boldsymbol{\theta}_T - \boldsymbol{\theta}^*\|^2 \le \left(1 - \dfrac{\mu}{L}\right)^T \|\boldsymbol{\theta}_0 - \boldsymbol{\theta}^*\|^2$ — *linear* rate, geometric in T.

- $\kappa = L/\mu$ is the condition number of the problem ($= \lambda_{\max}/\lambda_{\min}$ of the Hessian for quadratics). Iterations to reach accuracy ϵ: $O(\kappa \log(1/\epsilon))$.
- Ridge with λ: $\mu \ge \lambda \Rightarrow$ regularization also improves conditioning.
- Feature standardization reduces κ — the reason behind Chapter 1's scaling advice.

Bound the loss by a parabola and step to its bottom: the loss must drop.

Step 1. L-smoothness means the curvature is at most L, so the function sits below a parabola:

$$J(\mathbf{y}) \leq J(\mathbf{x}) + \nabla J(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) + \frac{L}{2} \|\mathbf{y} - \mathbf{x}\|^2 .$$

Step 2. Plug in a gradient step $\mathbf{y} = \mathbf{x} - \eta \nabla J(\mathbf{x})$:

$$J(\mathbf{x} - \eta \nabla J) \leq J(\mathbf{x}) - \eta \|\nabla J\|^2 + \frac{L\eta^2}{2} \|\nabla J\|^2 = J(\mathbf{x}) - \eta \Big(1 - \frac{L\eta}{2}\Big) \|\nabla J\|^2 .$$

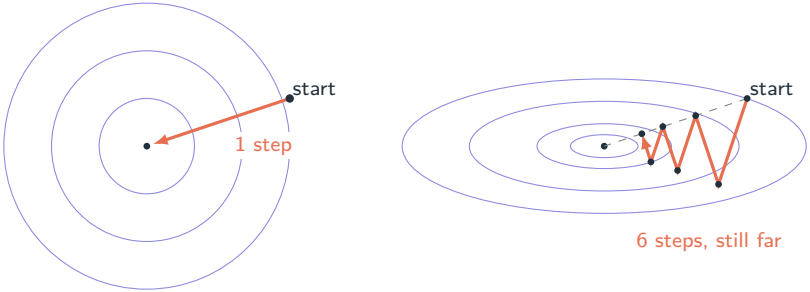
Step 3. The bracket is positive whenever $0 < \eta < 2/L$: the loss decreases. The best guaranteed drop is at $\eta = 1/L$.

Step 4. Summing these drops over iterations (and using convexity) gives the $O(1/T)$ rate.

Practical reading

You never know L, but the rule "if the loss goes up, halve η" is exactly a search for $\eta < 2/L$. Backtracking line search automates it.

Picture: round bowl, stretched bowl



round bowl: downhill points at the bottom **stretched bowl**: downhill points at the steep wall

- Round bowl: downhill points straight at the bottom. One big safe step (half the slope) and you are there.
- Stretched bowl: downhill points at the steep wall, not at the bottom. Small steps (a tenth of the slope) zigzag and crawl.

In one sentence

A stretched bowl makes the walk slow and wobbly; put all inputs on the same scale and the bowl turns round.

Worked example: GD on a 2-D quadratic

The step must suit the steep direction, so the flat one crawls.

$J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top A\mathbf{w}$ with $A = \mathrm{diag}(1, 10)$, start $\mathbf{w}_0 = (10, 1)$, step $\eta = 0.1$.

Step 1. Gradient:
$\nabla J = A\mathbf{w} = (w_1,\ 10w_2)$.

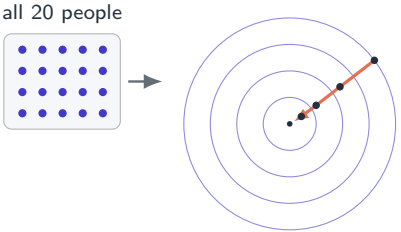
Step 2. One update:
$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta A\mathbf{w}_t = (0.9\, w_1,\ 0 \cdot w_2)$.

Step 3. Iterate: w_2 is exact after one step; w_1 shrinks by 0.9 each step.

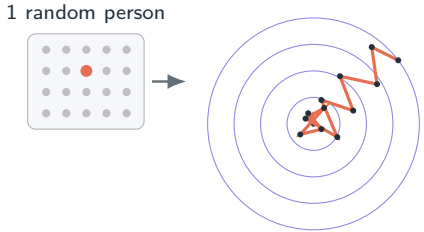
t	w_1	w_2
0	10	1
1	9	0
5	5.9	0
20	1.22	0
44	0.097	0

- $\eta = 0.1 = 1/\lambda_{\max}$ is the largest safe step; the flat direction then needs 44 steps for a $100\times$ reduction.
- Condition number $\kappa = 10$; time grows like κ. With $\eta = 0.25$ the w_2 coordinate would flip sign and grow: divergence.
- Rescaling the coordinates (standardization) makes $\kappa = 1$: one step.

Picture: ask everyone or one person?



gradient descent: ask all 20, then 1 careful step
4 steps = 80 questions



SGD: ask 1 random person, then 1 rough step
12 steps = 12 questions

- The slope is a vote of all 20 data points: asking everyone is exact but costs 20 questions per step.
- One random person gives a rough slope, so the path wobbles, but each step costs only 1 question.
- Near the bottom it keeps wobbling: take smaller steps, or ask a few people at once (a mini-batch).

In one sentence

SGD trades an exact slope for a cheap one: many rough steps beat a few perfect ones when the data is huge.

Stochastic gradient descent (SGD)

Estimate the slope from a few random samples: noisy but very cheap.

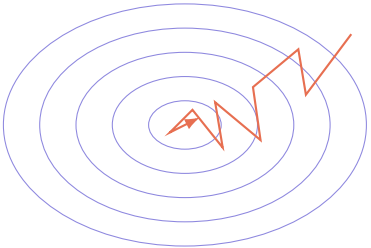
SGD

For $J(\boldsymbol{\theta}) = \frac{1}{n}\sum_i J_i(\boldsymbol{\theta})$, pick i_t uniformly at random and update

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta_t \nabla J_{i_t}(\boldsymbol{\theta}_t), \qquad \mathbb{E}[\nabla J_{i_t}] = \nabla J \text{ (unbiased)}.$$

Mini-batch: average over a batch B of size b: $\frac{1}{b}\sum_{i\in B}\nabla J_i$.

- Cost per step $O(bd)$ instead of $O(nd)$: for $n = 10^7$, one GD step $= 10^5$ SGD steps with $b = 100$.
- Noise means it never settles exactly: use a decaying step size η_t.



noisy but cheap steps

Batch size: trade-offs

Big batches: smooth steps, good GPU use; small: useful noise. Use 32–256.

- Gradient variance $\propto 1/b$: larger batches = smoother steps.
- GPU throughput: batches of 32–512 use hardware efficiently; $b = 1$ is wasteful.
- Very large batches generalize slightly worse in deep learning (less noise = less implicit regularization); scale η with b (linear scaling rule).
- One *epoch* = one pass over the n samples = n/b updates. Shuffle each epoch.

	GD ($b = n$)	mini-batch	SGD ($b = 1$)
cost / step	$O(nd)$	$O(bd)$	$O(d)$
gradient noise	none	moderate	high
steps / epoch	1	n/b	n
hardware use	good	best	poor
typical use	small n, convex	deep learning	online learning

Default

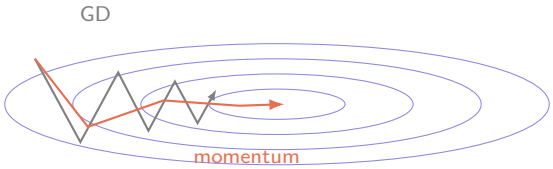
Mini-batch SGD with $b \in [32, 256]$ and a decaying or scheduled learning rate.

Keep rolling in the previous direction: oscillations cancel, progress adds up.

Momentum (heavy ball)

$$\mathbf{v}_{t+1} = \beta \mathbf{v}_t + \nabla J(\boldsymbol{\theta}_t)$$
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \mathbf{v}_{t+1}$$

$\beta \approx 0.9$: an average of past gradients.



- Damps oscillations across narrow valleys, accelerates along consistent directions.
- Nesterov: evaluate the gradient at the look-ahead point; provably faster on convex problems.

Each parameter gets its own step size from its gradient history.

- **AdaGrad**: divide each coordinate of the step by the square root of its accumulated squared gradients.
- **RMSProp**: same, but with a running average that forgets old gradients.
- **Adam**: RMSProp plus momentum. With $\mathbf{g}_t = \nabla J(\boldsymbol{\theta}_t)$:

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \quad \text{(average gradient)}$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \quad \text{(average squared gradient)}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t} + \epsilon} \quad \text{(hats: bias correction)}$$

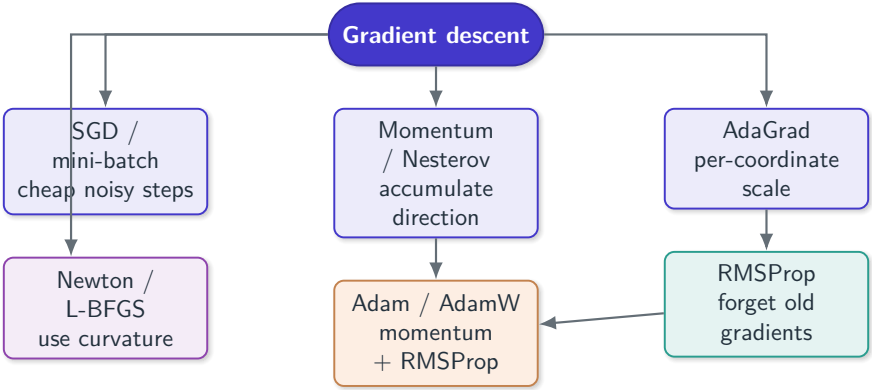
- Defaults: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$, $\eta = 10^{-3}$.
- Robust default for deep learning; SGD with momentum often generalizes slightly better when tuned.
- AdamW: weight decay applied separately, the correct way to combine ℓ_2 regularization with Adam.

Intuition

Coordinates with consistently large gradients get smaller steps; rare or sparse features get larger steps.

Optimizer family tree

Gradient descent plus one or two tricks: noise, memory, scaling, curvature.



plus a learning-rate schedule: step decay, cosine, warm-up (Chapter 9)

Fit a parabola and jump to its bottom: few steps, each one expensive.

Newton step

Minimize the second-order model $J(\boldsymbol{\theta}+\boldsymbol{\delta}) \approx J + \nabla J^\top \boldsymbol{\delta} + \frac{1}{2}\boldsymbol{\delta}^\top H \boldsymbol{\delta}$:

$$\boldsymbol{\delta} = -H^{-1}\nabla J, \qquad \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - H(\boldsymbol{\theta}_t)^{-1}\nabla J(\boldsymbol{\theta}_t)$$

- Exact in one step for quadratics; *quadratic* convergence near the optimum ($\epsilon_{t+1} \propto \epsilon_t^2$).
- Cost: $O(d^2)$ memory, $O(d^3)$ solve per step — fine for $d \leq 10^4$, impossible for deep nets.

Logistic regression: IRLS

$\nabla J = X^\top(\mathbf{p}-\mathbf{y})$, $H = X^\top S X$ with $S = \mathrm{diag}(p_i(1-p_i))$:

$$\mathbf{w} \leftarrow \mathbf{w} - (X^\top S X)^{-1} X^\top (\mathbf{p}-\mathbf{y})$$

converges in 5–10 iterations (scikit-learn's `newton-cg` solver).

Newton uses slope and curvature: three steps; GD uses slope only: ten.

$J'(\theta) = 2\theta + e^{\theta}$, $J''(\theta) = 2 + e^{\theta}$, start $\theta_0 = 1$; the minimizer is $\theta^* \approx -0.3517$.

Newton: $\theta \leftarrow \theta - J'(\theta)/J''(\theta)$

Step 1. $\theta_1 = 1 - \dfrac{2 + 2.718}{2 + 2.718} = 0.$

Step 2. $\theta_2 = 0 - \dfrac{0 + 1}{2 + 1} = -0.333.$

Step 3. $\theta_3 = -0.3517$; then it no longer moves ✔

Gradient descent: $\theta \leftarrow \theta - 0.2\, J'(\theta)$

Step 1. $\theta_1 = 1 - 0.2 \cdot 4.718 = 0.056.$

Step 2. $\theta_2 = -0.178$, **Step 3.** $\theta_3 = -0.274$.

Step 5. $\theta_5 = -0.335$, **Step 10.** $\theta_{10} = -0.3514$.

Trade-off

Newton: 3 iterations to machine precision, but each needs the second derivative (a $d \times d$ matrix to solve in general). GD: 10 iterations to 3 decimals, each one cheap. For large d, cheap wins.

At a kink: gradient step, then snap small weights to zero.

Subgradient

$\mathbf{g}$ is a subgradient of convex f at $\mathbf{x}$ if $f(\mathbf{y}) \geq f(\mathbf{x}) + \mathbf{g}^\top(\mathbf{y} - \mathbf{x})\ \forall \mathbf{y}$. At a kink there are many.

- $|x|$ at 0: any $g \in [-1, 1]$. Hinge at the kink: $g \in [-1, 0]$.

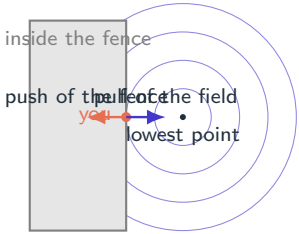
Proximal gradient (ISTA) for $J = f + \lambda \|\cdot\|_1$

$$\boldsymbol{\theta} \leftarrow \mathrm{soft}_{\eta\lambda}\big(\boldsymbol{\theta} - \eta \nabla f(\boldsymbol{\theta})\big)$$
$$\mathrm{soft}_\tau(z) = \mathrm{sign}(z)\max(|z| - \tau, 0)$$

- Gradient step on the smooth part, then *soft-thresholding*: coordinates smaller than $\eta\lambda$ become exactly zero $\Rightarrow$ sparsity.

Intuition: constraints are fences

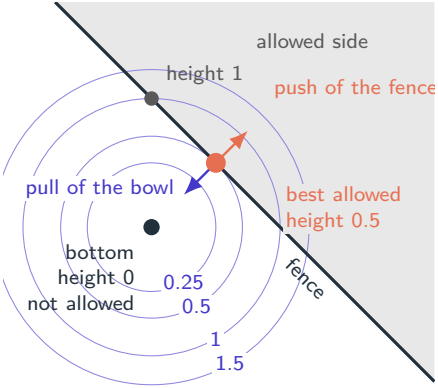
- You want the lowest point of a field, but a fence says you must stay inside a certain area.
- Two cases. The lowest point is inside the fence: the fence changes nothing, its "price" is zero. Or it is outside: you end up pressed against the fence, which pushes back with some force. That force is the **Lagrange multiplier**.
- "Either the fence does nothing or it pushes" is the KKT rule. In the SVM (Chapter 7) each training point is a fence; the few that push are the **support vectors**.



In one sentence

A constraint is a fence: it either does nothing or pushes back, and the push is the multiplier.

Picture: the fence and the bowl



- Blue circles join points of equal height (their number). The bottom, height 0, is at the centre; the fence says: stay on the grey side.
- Roll downhill but stay on the grey side. You stop pressed on the fence, at the coral dot: height 0.5, read on its circle. Every other allowed point is higher: the grey dot has height 1.
- There, the bowl's pull and the fence's push are exactly opposite: that balance is the rule of the next slides.

In one sentence

Bottom outside the fence: the best allowed point sits on the fence, where push balances pull.

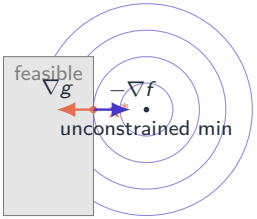
Turn a constraint into a price and add it to the objective.

Problem and Lagrangian

$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ s.t. } g_i(\mathbf{x}) \le 0 \ (i = 1..m), \ h_j(\mathbf{x}) = 0 \ (j = 1..p)$$

$$\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f(\mathbf{x}) + \textstyle\sum_i \alpha_i g_i(\mathbf{x}) + \sum_j \beta_j h_j(\mathbf{x}), \quad \alpha_i \ge 0.$$

- **Dual function**: $D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \min_{\mathbf{x}} \mathcal{L}$; always concave.
- **Weak duality**: $D(\boldsymbol{\alpha}, \boldsymbol{\beta}) \le f(\mathbf{x})$ for any feasible $\mathbf{x}$, $\boldsymbol{\alpha} \ge 0$.
- **Strong duality** ($\max D = \min f$) holds for convex problems (under a mild condition).



At $\mathbf{x}^*$: $\nabla f = -\alpha \nabla g$.

A constraint is either pushing (positive price) or irrelevant (zero price).

Karush–Kuhn–Tucker conditions

$\mathbf{x}^*$ is optimal (convex problem, strong duality) iff there exist $\boldsymbol{\alpha}^*, \boldsymbol{\beta}^*$ with

Stationarity:	$\nabla f(\mathbf{x}^*) + \sum_i \alpha_i^* \nabla g_i(\mathbf{x}^*) + \sum_j \beta_j^* \nabla h_j(\mathbf{x}^*) = \mathbf{0}$
Primal feasibility:	$g_i(\mathbf{x}^*) \leq 0, \;\; h_j(\mathbf{x}^*) = 0$
Dual feasibility:	$\alpha_i^* \geq 0$
Complementary slackness:	$\alpha_i^* \, g_i(\mathbf{x}^*) = 0 \quad \forall i$

Complementary slackness is the key

Either a constraint is *active* ($g_i = 0$) and may have $\alpha_i > 0$, or it is inactive ($g_i < 0$) and $\alpha_i = 0$. In the SVM (Chapter 7), the constraints are $y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1$; the samples with $\alpha_i > 0$ are exactly the *support vectors*. The solution $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$ depends only on them.

Lagrangian, solve for x in terms of α, decide if the constraint is active.

min $f(x_1, x_2) = x_1^2 + x_2^2$ subject to $g(x) = 1 - x_1 - x_2 \leq 0$ (that is, $x_1 + x_2 \geq 1$).

Step 1. Lagrangian:
$\mathcal{L} = x_1^2 + x_2^2 + \alpha(1 - x_1 - x_2)$, $\alpha \geq 0$.

Step 2. Stationarity: $2x_1 - \alpha = 0$ and $2x_2 - \alpha = 0$, so $x_1 = x_2 = \alpha/2$.

Step 3. Case $\alpha = 0$: $x = (0, 0)$, but then $g = 1 > 0$: not allowed.

Step 4. Case $\alpha > 0$: the constraint must be active, $x_1 + x_2 = 1$, so $\alpha = 1$ and $x^* = (\frac{1}{2}, \frac{1}{2})$, $f^* = \frac{1}{2}$.

Step 5. Dual check: plugging $x = \alpha/2$ into $\mathcal{L}$ gives $D(\alpha) = \alpha - \frac{\alpha^2}{2}$, maximal at $\alpha = 1$ with $D = \frac{1}{2} = f^*$: strong duality ✔

Pattern to remember

Solve stationarity for $\mathbf{x}$ in terms of $\boldsymbol{\alpha}$, substitute back to get the dual, maximize over $\boldsymbol{\alpha} \geq 0$. The SVM dual (Chapter 7) is derived in exactly this way.

Same multiplier trick for equalities; for simple sets, project back after each step.

PCA as a constrained problem

$\max_{\mathbf{u}} \mathbf{u}^\top \Sigma \mathbf{u}$ s.t. $\|\mathbf{u}\|^2 = 1$.
$\mathcal{L} = \mathbf{u}^\top \Sigma \mathbf{u} - \beta(\mathbf{u}^\top \mathbf{u} - 1)$, $\nabla_{\mathbf{u}} \mathcal{L} = 2\Sigma \mathbf{u} - 2\beta \mathbf{u} = \mathbf{0}$
$\Rightarrow \Sigma \mathbf{u} = \beta \mathbf{u}$: $\mathbf{u}$ is an eigenvector, objective $= \beta$, so take $\lambda_{\max}$.

Maximum entropy

$\max -\sum p_i \log p_i$ s.t. $\sum p_i = 1$ gives $p_i =$ const: the uniform distribution.

Projected gradient descent

For $\min_{\mathbf{x} \in C} f(\mathbf{x})$ with convex C:

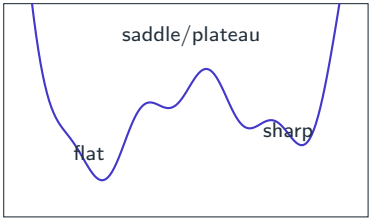
$$\mathbf{x} \leftarrow \Pi_C\big(\mathbf{x} - \eta \nabla f(\mathbf{x})\big)$$

$$\Pi_C(\mathbf{z}) = \arg\min_{\mathbf{x} \in C} \|\mathbf{x} - \mathbf{z}\|$$

- Box constraints: clip. Simplex: sort-and-shift. Ball: rescale.
- Proximal gradient = projected GD with a penalty instead of a set.

Not bowls, yet SGD works: saddles are escaped, most minima are good enough.

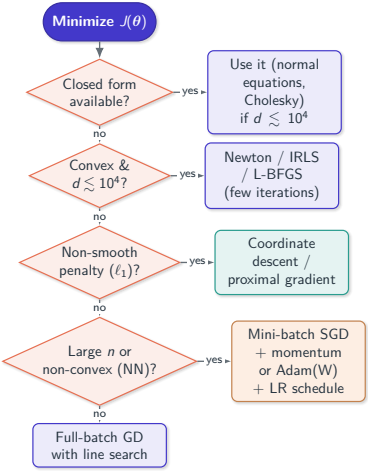
- **Saddle points** dominate in high dimension (a random critical point has some negative curvature with overwhelming probability); GD escapes them slowly, SGD noise helps.
- **Local minima** in over-parameterized nets tend to be nearly as good as the global one.
- **Plateaus**: vanishing gradients (saturated activations).
- **Sharp vs. flat minima**: flat ones generalize better; SGD noise biases towards flat regions.



Tools that help: good initialization, normalization layers, residual connections, learning-rate schedules, momentum (Chapter 9).

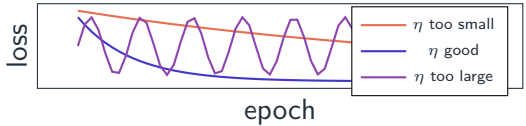
Choosing an optimizer: decision flowchart

Closed form if possible, Newton for small convex, Adam or SGD for the rest.



Scale inputs, check gradients, overfit a tiny set, sweep the learning rate.

1. Standardize inputs (conditioning).
2. Check the gradient numerically on a tiny example.
3. Overfit a tiny subset first (loss $\to$ 0?): catches bugs.
4. Sweep η on a log scale; pick the largest stable value.
5. Plot training *and* validation loss per epoch.
6. Loss explodes $\Rightarrow$ lower η, clip gradients.
7. Loss plateaus early $\Rightarrow$ raise η, add momentum, check for dead units / vanishing gradients.
8. Loss noisy $\Rightarrow$ larger batch or decay η.
9. Use a schedule (cosine / step) and early stopping.
10. Fix seeds; log everything.



Key formulas

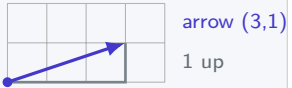
- Convex: $f(t\mathbf{x} + (1-t)\mathbf{y}) \leq tf(\mathbf{x}) + (1-t)f(\mathbf{y})$; $\nabla^2 f \succeq 0$; local $=$ global.
- GD $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J$; descent lemma needs $\eta < 2/L$; rate $O(1/T)$ convex, $(1-\mu/L)^T$ strongly convex; $\kappa = L/\mu$.
- SGD: unbiased noisy gradient, $O(bd)$ per step, decaying η_t; momentum $\mathbf{v} \leftarrow \beta\mathbf{v} + \mathbf{g}$; Adam normalizes by $\sqrt{\hat{\mathbf{v}}}$.
- Newton $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - H^{-1}\nabla J$: quadratic convergence, $O(d^3)$; IRLS for logistic regression.
- Proximal / soft-thresholding for ℓ_1: $\operatorname{sign}(z)\max(|z| - \tau, 0)$.
- Lagrangian $\mathcal{L} = f + \sum \alpha_i g_i + \sum \beta_j h_j$; KKT: stationarity, feasibility, $\alpha_i \geq 0$, $\alpha_i g_i = 0$.

Chapter Summary and Exercises

Section 9 — Chapter 2

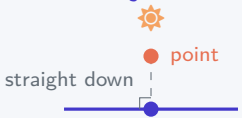
Picture: the chapter in six drawings

Vector



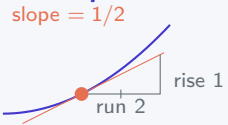
numbers become arrows

Projection



shadow = closest point

Slope



rise over run

Probability



3 red of 10 = 30%

count the marbles

Gaussian



bell around average

Gradient descent

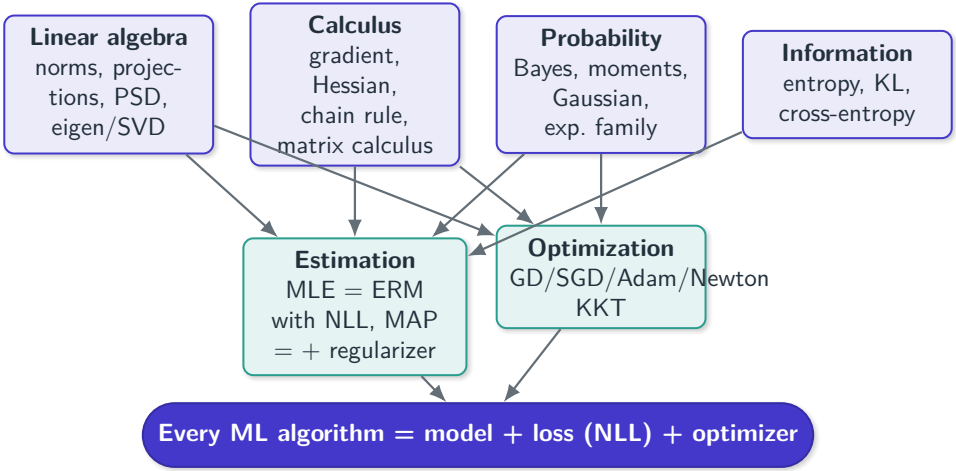


ball rolls down

In one sentence

The whole chapter is six drawings: arrow, shadow, slope, marbles, bell and bowl.

Six toolboxes feed estimation and optimization; every algorithm combines them.



Key equations of Chapter 2

Projection	$\text{proj}_{\mathbf{u}}\,\mathbf{x} = \frac{\mathbf{x}^\top \mathbf{u}}{\|\mathbf{u}\|^2}\mathbf{u}$; normal equations $X^\top X\mathbf{w} = X^\top \mathbf{y}$
Spectral / SVD	$A = Q\Lambda Q^\top$; $A = U\Sigma V^\top$; $\max_{\|\mathbf{x}\|=1} \mathbf{x}^\top A\mathbf{x} = \lambda_{\max}$; ridge shrinkage $\frac{\sigma_i^2}{\sigma_i^2+\lambda}$
Matrix calculus	$\nabla \mathbf{x}^\top A\mathbf{x} = (A + A^\top)\mathbf{x}$; $\nabla \|X\mathbf{w} - \mathbf{y}\|^2 = 2X^\top(X\mathbf{w} - \mathbf{y})$; $\nabla_W \mathbf{a}^\top W\mathbf{b} = \mathbf{a}\mathbf{b}^\top$
Chain rule	$\nabla_{\mathbf{x}} L = J_{\mathbf{g}}^\top \nabla_{\mathbf{z}} L$ (backprop)
Bayes	$p(\theta \mid \mathcal{D}) = p(\mathcal{D} \mid \theta)p(\theta)/p(\mathcal{D})$
Moments	$\text{Var}(\bar{X}) = \sigma^2/n$; $\text{Cov}(A\mathbf{x}) = A\Sigma A^\top$; Mahalanobis $(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})$
Gaussian conditional	$\boldsymbol{\mu}_a + \Sigma_{ab}\Sigma_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b)$, $\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$
Exponential family	$h(\mathbf{x})\exp(\boldsymbol{\eta}^\top T(\mathbf{x}) - A(\boldsymbol{\eta}))$, $\nabla A = \mathbb{E}[T]$
MLE / MAP	$\arg\max \sum \log p(\mathbf{x}_i \mid \boldsymbol{\theta})$; MAP adds $\log p(\boldsymbol{\theta})$; Gaussian prior $\Rightarrow$ ridge, $\lambda = \sigma^2/\tau^2$
Information	$H = -\sum p \log p$; $D_{KL}(p\|q) = \sum p \log \frac{p}{q} \geq 0$; $H(p,q) = H(p) + D_{KL}$; $I = H(Y) - H(Y \mid X)$
GD	$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla J$, $\eta < 2/L$, rate $(1 - \mu/L)^T$; Newton $-H^{-1}\nabla J$; Adam $\hat{\mathbf{m}}/(\sqrt{\hat{\mathbf{v}}} + \epsilon)$
KKT	stationarity, feasibility, $\alpha_i \geq 0$, $\alpha_i g_i = 0$

Exercises (1/3): linear algebra and calculus

Start with 1, 4 and 6: the skills used most in Chapter 3.

1. Compute the eigenvalues and unit eigenvectors of $\begin{pmatrix} 5 & 2 \\ 2 & 2 \end{pmatrix}$. Sketch the ellipse $\mathbf{x}^\top A\mathbf{x} = 1$ and its condition number.
2. Show that the hat matrix $H = X(X^\top X)^{-1}X^\top$ is a projection ($H^2 = H$, $H^\top = H$) and that $\operatorname{tr} H = d$.
3. Compute the thin SVD of $A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix}$ and its rank-1 approximation error.
4. Derive $\nabla_{\mathbf{w}}$ and the Hessian of $J(\mathbf{w}) = \frac{1}{2n}\|X\mathbf{w} - \mathbf{y}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$. Show $H \succ 0$ for $\lambda > 0$.
5. For $f(\mathbf{x}) = \log \sum_i e^{x_i}$ show $\nabla f = \operatorname{softmax}(\mathbf{x})$ and $H = \operatorname{diag}(\mathbf{s}) - \mathbf{s}\mathbf{s}^\top \succeq 0$ (hence f is convex).
6. Implement gradient checking: compare the analytic gradient of the logistic loss with central differences on random data.

Exercises (2/3): probability and estimation

Exercises 7 and 11 are the core; 9 and 10 are good practice.

7. A spam filter flags 99% of spam and 2% of ham; 20% of mail is spam. What fraction of flagged mail is spam? Draw the decision as a Bayes computation.
8. $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with $\Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$. Give the distribution of $x_1 \mid x_2 = 1$ and of $y = x_1 - x_2$.
9. Write the Poisson distribution in exponential-family form; identify η, T, A; verify $A'(\eta) = \mathbb{E}[x]$.
10. Derive the MLE of λ for i.i.d. Exponential(λ) data. Then derive the MAP with a Gamma(α, β) prior.
11. Show that for i.i.d. Gaussian data with known σ^2 and prior $\mu \sim \mathcal{N}(\mu_0, \tau^2)$, the posterior mean is a convex combination of $\bar{x}$ and μ_0. Interpret the weights.
12. Compute $D_{KL}(\text{Bern}(0.3) \| \text{Bern}(0.5))$ and $D_{KL}(\text{Bern}(0.5) \| \text{Bern}(0.3))$; comment on the asymmetry.

Exercises (3/3): optimization

Do 13 and 14 first; 18 prepares the SVM chapter.

13. For $J(\mathbf{w}) = \frac{1}{2}\mathbf{w}^\top A\mathbf{w}$ with eigenvalues $\{1, 100\}$, find the range of η for which GD converges and the number of iterations to reduce the error by 10^{-3} at the best η.
14. Implement GD, GD+momentum and Adam in NumPy on the Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$; plot the trajectories.
15. Implement logistic regression via (a) GD, (b) Newton/IRLS; compare iterations and wall-clock on a synthetic dataset with $d = 50$.
16. Apply ISTA (proximal gradient) to a Lasso problem with $d = 200$, $n = 100$, 10 true non-zero coefficients; plot the number of non-zeros vs. λ.
17. Solve $\min x_1^2 + 2x_2^2$ s.t. $x_1 + x_2 = 1$ with a Lagrangian; then with $x_1 + x_2 \geq 1$ via KKT.
18. Derive the dual of $\min \frac{1}{2}\|\mathbf{w}\|^2$ s.t. $y_i\mathbf{w}^\top\mathbf{x}_i \geq 1$ (hard-margin SVM) and identify the support vectors in a 4-point example.

Ex. 1. $\det(A-\lambda I)=\lambda^2-7\lambda+6=0\Rightarrow\lambda=6,1$. $\mathbf{v}_1=\frac{1}{\sqrt{5}}(2,1)$, $\mathbf{v}_2=\frac{1}{\sqrt{5}}(-1,2)$. Semi-axes $1/\sqrt{6}$, 1; $\kappa=6$.

Ex. 7. $P(\text{flag})=0.99\cdot0.2+0.02\cdot0.8=0.214$; $P(\text{spam}\mid\text{flag})=0.198/0.214=0.925$.

Ex. 8. $x_1\mid x_2=1\sim\mathcal{N}(\mu_1+\frac{1}{2}(1-\mu_2),\ 2-\frac{1}{2})=\mathcal{N}(\mu_1+\frac{1}{2}(1-\mu_2),\ 1.5)$; $y\sim\mathcal{N}(\mu_1-\mu_2,\ 2+2-2)=\mathcal{N}(\mu_1-\mu_2,2)$.

Ex. 10. $\ell=n\log\lambda-\lambda\sum x_i\Rightarrow\hat\lambda=n/\sum x_i=1/\bar{x}$. MAP: $(n+\alpha-1)/(\sum x_i+\beta)$.

Ex. 12. $D(0.3\|0.5)=0.3\log\frac{0.3}{0.5}+0.7\log\frac{0.7}{0.5}=0.082$ nats; $D(0.5\|0.3)=0.5\log\frac{0.5}{0.3}+0.5\log\frac{0.5}{0.7}=0.087$ nats.

Ex. 13. Converges iff $\eta<2/100=0.02$. Best $\eta=2/(\lambda_{\min}+\lambda_{\max})=2/101$, contraction $\frac{99}{101}$ per step; 10^{-3} needs $\log(10^{-3})/\log(99/101)\approx345$ iterations. Momentum would need $\approx\sqrt{\kappa}=10$ times fewer.

If the self-assessment list feels comfortable, you are ready.

Chapter 3 — Regression uses:

- projections and normal equations (least squares)
- SVD (ridge, ill-conditioning)
- MLE with Gaussian noise, MAP with Gaussian/Laplace priors
- GD convergence and conditioning
- proximal methods (Lasso)
- exponential family (GLMs)

Self-assessment

You are ready if you can, without notes:

- derive $\hat{\mathbf{w}} = (X^\top X)^{-1} X^\top \mathbf{y}$ from $\nabla J = 0$,
- explain why a Gaussian prior gives ℓ_2 regularization,
- state when GD converges and how fast,
- write the KKT conditions for a small problem.