Linear Algebra for ML

The slice of linear algebra that ML engineers actually use — vectors, matmul intuition, decompositions, and why backprop is just the chain rule on tensors.

01 · VECTORS

A vector is a list — until it isn't

In code, a vector is a 1-D array. In math, it's an arrow in space, a point, OR a function — whichever interpretation makes the current problem easier.

$x_2 = \sqrt{(\Sigma x_i^2)}$	L2 norm — length of the arrow; dominates ML loss functions
$x_1 = \Sigma \lvert x_i \rvert$	L1 norm — induces sparsity; backbone of Lasso regression
$x \cdot y = \Sigma x_i y_i$	Dot product — measures alignment; 0 when perpendicular
$\cos \theta = (x \cdot y) / (\lVert x \rVert \lVert y \rVert)$	Cosine similarity — the metric behind text + image embeddings
$x - \text{proj}_y(x)$	Residual after projection; the geometry behind least squares

02 · MATRICES

Three ways to read a matrix

Every matrix can be viewed three ways. Switching between them is how you debug ML code.

- As a stack of ROWS — each row is one data point (the dataset view)
- As a stack of COLUMNS — each column is one feature (the linear-combination view)
- As a LINEAR MAP — Ax sends vector x to a new vector in a (possibly different) space
- Shape (m, n) means m rows × n columns. A @ B works iff A is (m, k) and B is (k, n)
- Inner dimension cancels: (m, k)(k, n) → (m, n). 90% of shape bugs are caught by reading this aloud

03 · MATMUL INTUITION

Four equivalent views of A @ B

(A @ B)[i,j] = row i of A · col j of B	Definitional view — slow but always correct
A @ B = sum of (col k of A) ⊗ (row k of B)	Sum of outer products — the SVD view
A @ B applies A to each column of B	B as a batch of vectors — the 'transform all at once' view
A @ B = B's rows as linear combos of A's rows	Each output row mixes A's rows by B's coefficients
Cost: O(m·n·k)	Why ML loves big batches: same FLOPS, way better cache utilisation

04 · SPACES & RANK

Where do the columns of A live?

- Column space of A = span of A's columns = where Ax can land
- Null space of A = vectors x with Ax = 0 — the degrees of freedom you've lost
- rank(A) = dim of column space ≤ min(m, n)
- Full-rank means the matrix is invertible-like; low-rank means there's redundant info you can compress
- Low-rank approximations are why PCA / LoRA / matrix factorisation all work — most data lives in a low-d subspace

05 · DECOMPOSITIONS

Four factorisations that earn their keep

A = QR	Q orthonormal, R upper-triangular — the stable way to solve least squares
A = LU	Triangular factors — how np.linalg.solve works under the hood
A = U Σ Vᵀ (SVD)	The Swiss Army knife — PCA, pseudo-inverse, low-rank approx
A = Q Λ Qᵀ (eigendecomp, symmetric A)	Diagonalises symmetric matrices — covariance matrices, kernels
A = Σᵢ σᵢ uᵢ vᵢᵀ	Outer-product form of SVD — keep top-k for rank-k approximation

06 · GRADIENTS

Backprop is the chain rule on shaped tensors

- ∇x (Ax) = Aᵀ — the gradient of a linear map is its transpose
- ∇x (xᵀ A x) = (A + Aᵀ)x — quadratic forms; becomes 2Ax if A is symmetric
- ∇W (Wx) w.r.t. matrix W is the outer product ∂L/∂y outer x — that's why a Linear layer's grad has the same shape as W
- Jacobian shape rule: if y = f(x) maps $\mathbb{R}^n \to \mathbb{R}^m$, ∂y/∂x is (m, n)
- Backprop never builds the full Jacobian — it uses vector-Jacobian products to keep memory in O(n)

Beyond the basics

COMMON PITFALLS

- ! Confusing row and column shape — (n,) and (n, 1) are different things and most APIs treat them differently
- ! Inverting a matrix to solve Ax = b — use np.linalg.solve(A, b); inv() is slower and numerically worse
- ! Comparing eigenvalues across different scales — normalise the matrix first (divide by max abs element)
- ! Assuming A @ B == B @ A — matrix multiplication is NOT commutative; this is the single most common bug in fresh ML code
- ! Forgetting that float32 SVD on a near-singular matrix is unreliable — use float64 or regularise (A + λI)

FURTHER READING

- → 3Blue1Brown — Essence of Linear Algebra (visual playlist) (https://www.3blue1brown.com/topics/linear-algebra)
- → Matrix Cookbook (PDF reference for identities) (https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf)
- → Gilbert Strang — Linear Algebra and Learning from Data (https://math.mit.edu/learningfromdata/)
- → Stephen Boyd — Applied Linear Algebra (free book + Stanford course) (https://web.stanford.edu/~boyd/vmls/)