Week 11 — Deep Neural Networks & LLMs

Friday, July 10, 2026

Prof. Joseph Austerweil

Housekeeping

- **Today’s paper presentation** kicks us off — then the lecture.
- **Due TONIGHT, 8 PM:** the **Reinforcement Learning** assignment **and** the **Monte Carlo** assignment.
- **Final-project presentations:** next week (Week 12, Jul 17) — bring a laptop.
- Two new widgets today — I’ll drive them live, and they’re linked from the slides to replay.

Paper presentation

Agenda

Where we left off — from fixed kernels to *learned features*	0:05
Represent I — connectionism, distributed reps & **vectors, concretely**	0:17
Learn I — perceptron, gradient descent, the delta rule	0:43
Learn II — **matrices**, XOR, depth, and backpropagation	1:01
Represent II — architecture as prior: CNN → RNN → **attention**	1:23
Understand — scaling, emergence, **in-context learning**	1:36
Where this leaves us — one question, four answers	1:50

Times are lecture-clock, after the paper presentation.

Where we left off

Quick poll — do nonparametric models have parameters?

Last week: **Bayesian nonparametric** models (CRP, Dirichlet process, GPs). Do they have parameters?

A. Yes — infinitely many

B. Yes — just a few

C. No — it’s called *non*parametric for a reason

Reveal

A — infinitely many.

A nonparametric model doesn’t *fix* how many parameters it has — it lets the number **grow with the data**. Hold that thought: today’s networks are also enormous, and last week we saw the infinite-width limit *is* a Gaussian process.

Quick poll — which is NOT parametric?

Which of these is **not** a parametric model?

A. a 3-cluster Gaussian mixture

B. a single Gaussian

C. a kernel / exemplar method

Reveal

C — kernel / exemplar methods.

A kernel method's "parameters" *are* the stored examples. Week 10's punchline: an **infinitely wide net** trained by gradient descent behaves like a **fixed-kernel** method (the **NTK**). So the natural next question —

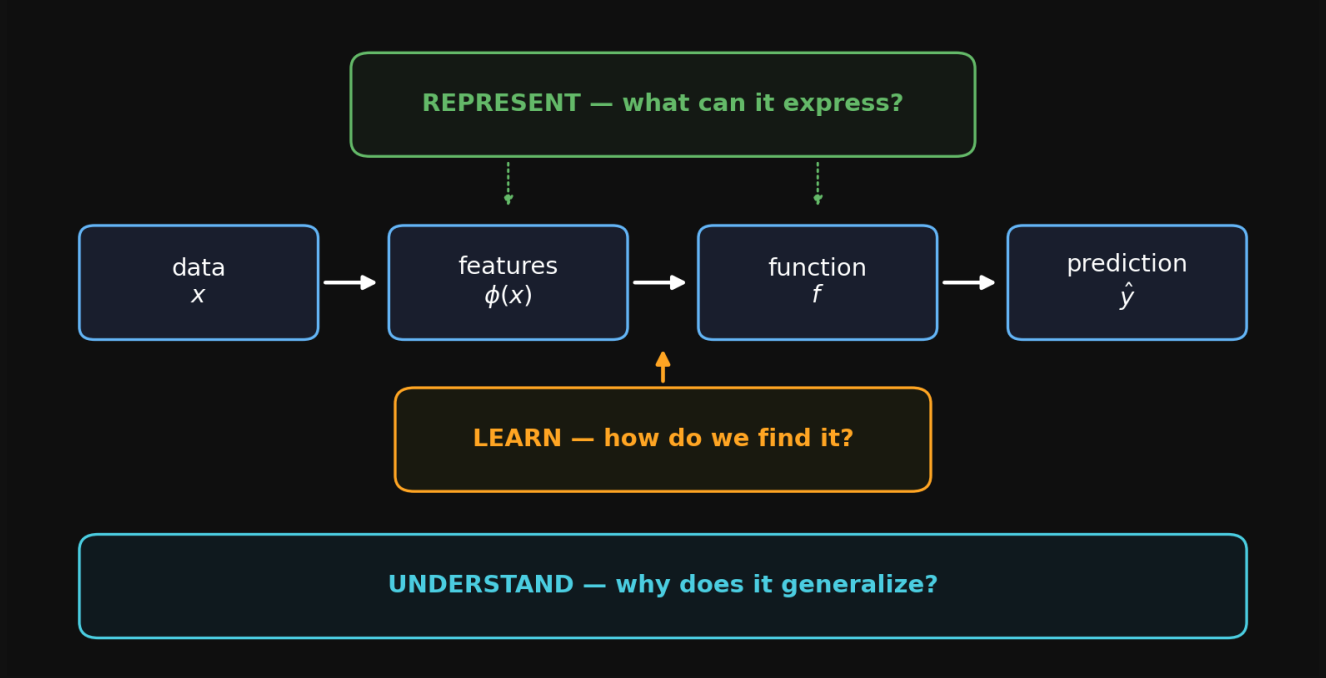
The question the whole course has been asking

Where does the hypothesis space come from? Week 10 gave *three* answers — and all three fix the features (the kernel) in advance.

- **Penalize complexity** → ridge · **grow with the data** → Bayesian nonparametrics · **prior over functions** → Gaussian processes

Today: the fourth answer — *learn the features themselves.*

One machine, three questions



Every model today is this pipeline. We'll light up one question per block: **Represent** (what can it express?), **Learn** (how do we find it?), **Understand** (why does it generalize — and is it secretly Bayesian?).

Our standing critic: Lake et al. 2017

This week's reading — *Building machines that learn and think like people* — is the **"what's missing from deep learning"** case. We'll keep a running **scorecard**: does each idea today address the critique, or dodge it?

- Their asks: **compositionality**, **causality**, **learning-to-learn**, **intuitive physics/psychology**.

The charge isn't new: Lake et al. modernize **Fodor & Pylyshyn (1988)**, who argued neural nets lack the **systematicity** of symbolic thought — anyone who can think "John loves Mary" can also think "Mary loves John." A **2025** re-test at the end asks whether today's models close the gap.

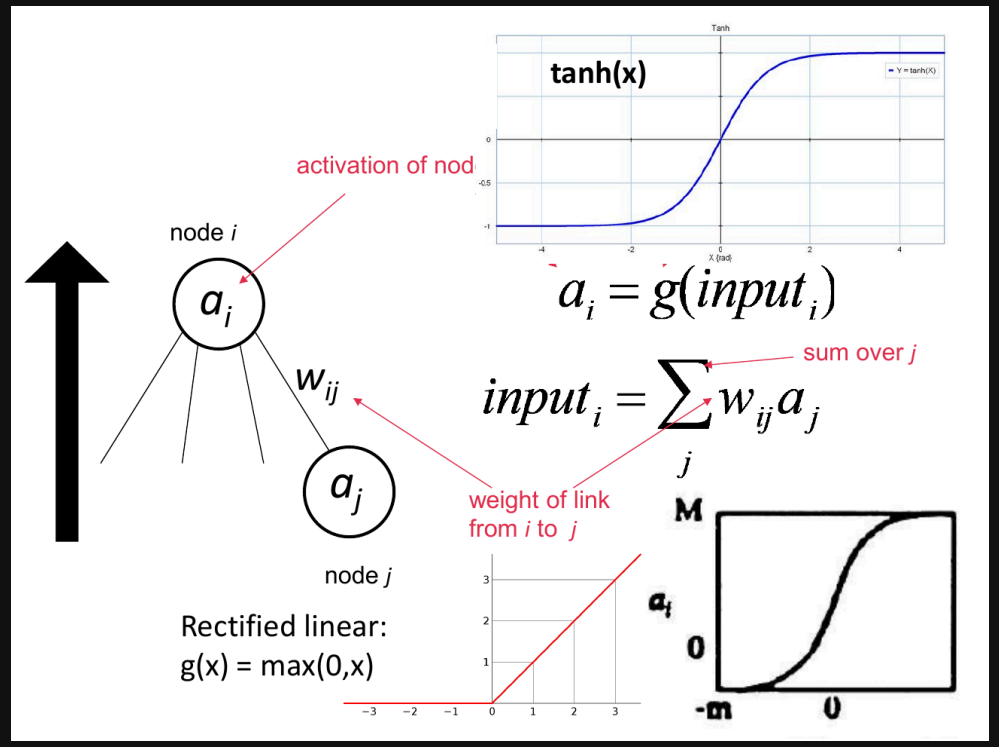
Represent I — connectionism

Emergence: order from simple units

- A **termite mound** — meters tall, ventilated, temperature-regulated — is built by thousands of termites with **no architect** and no blueprint.
- Each unit follows tiny local rules; the structure is **emergent**.

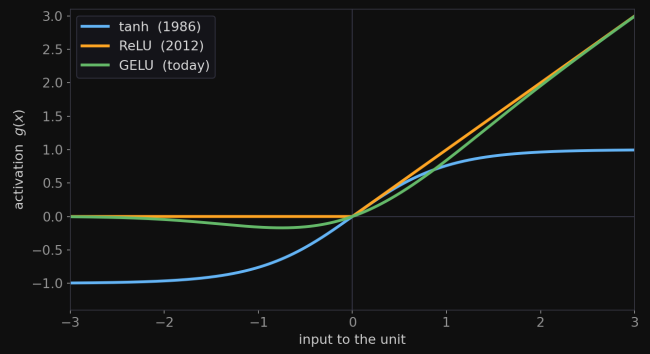
Connectionism (Parallel Distributed Processing): model cognition as computation over **many interconnected simple units** — each unit's activity depends only on the units feeding it. Loosely neuron-inspired.

The artificial neuron



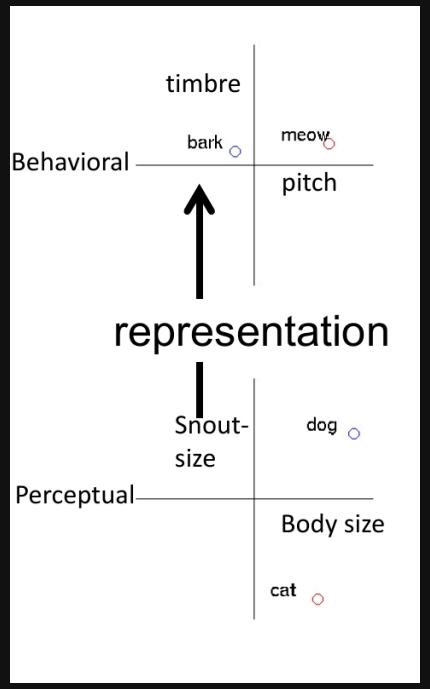
One unit: $a_i = g\left(\sum_j w_{ij}\, a_j\right)$ — a **weighted sum** of its inputs, squashed by an **activation function** g. The weights w_{ij} are what the network learns.

Activation functions: a short history



- **tanh / sigmoid** (1980s): smooth, nice derivatives — good for the early learning rules.
- **ReLU** $= \max(0, x)$ (2012–): cheap, doesn't saturate — the deep-learning workhorse.
- **GELU / SiLU** (today): smooth ReLU-like; the standard in transformers.

Distributed representations



A concept isn’t one node — it’s a **pattern of activity across many units**. “Cat” and “dog” are nearby points in a feature space (snout size, body size → pitch, timbre).

Distributed vs localist — why it matters

Distributed (a pattern over units):

- **noise tolerance for free** — flip one unit, the pattern survives
- **interpolation** — points between “cat” and “dog” are meaningful

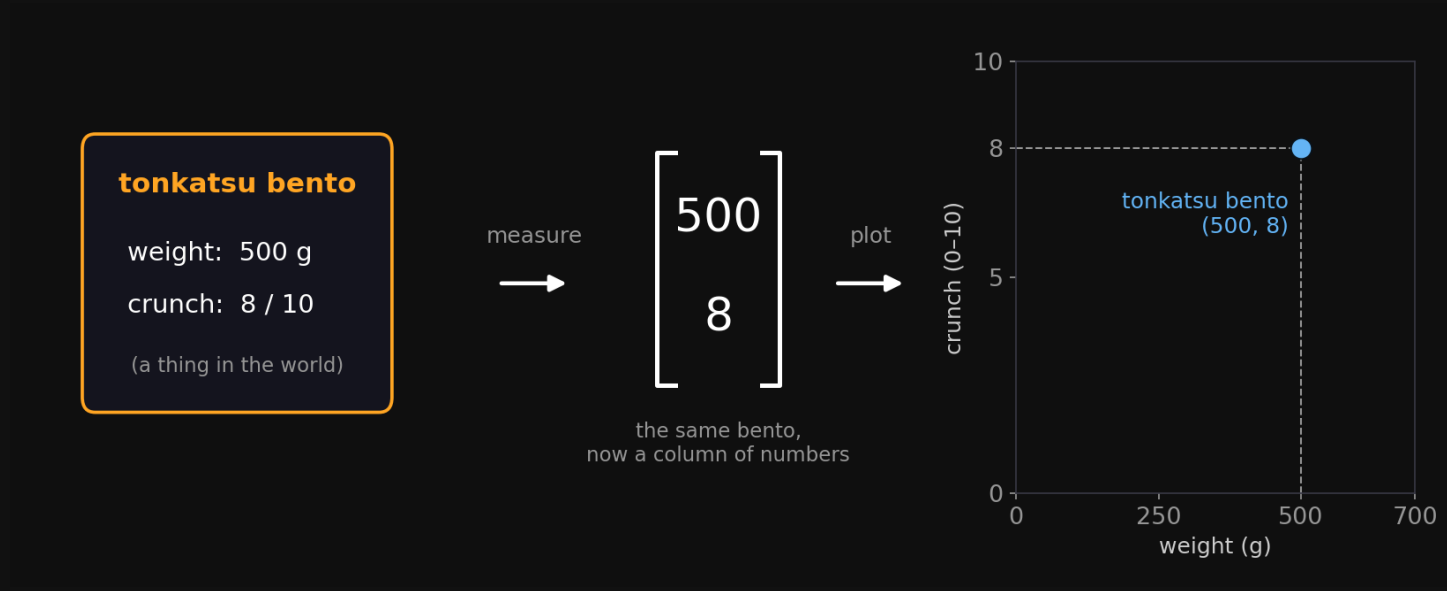
Localist (one node per concept):

- transparent, but **brittle**
- no natural notion of “between”

The distributed view is what makes **word embeddings** possible.

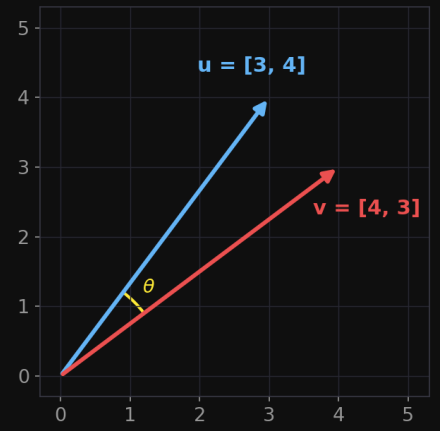
📋 Lake scorecard: distributed reps buy **similarity** and **interpolation** — but *not* **compositionality** by themselves.

Data become vectors



Machines only eat numbers — so **measure**: this bento weighs 500 g, crunch 8/10. The list of measurements $[500, 8]$ is a **vector**; each measurement is a **component**; the number of components is the **dimension**. A 2-D vector *is* a point on a plane — plotting data and “representing” data are the same act.

Similarity is direction: the dot product



$u \cdot v = 3\times4 + 4\times3$

$= 12 + 12 = 24$

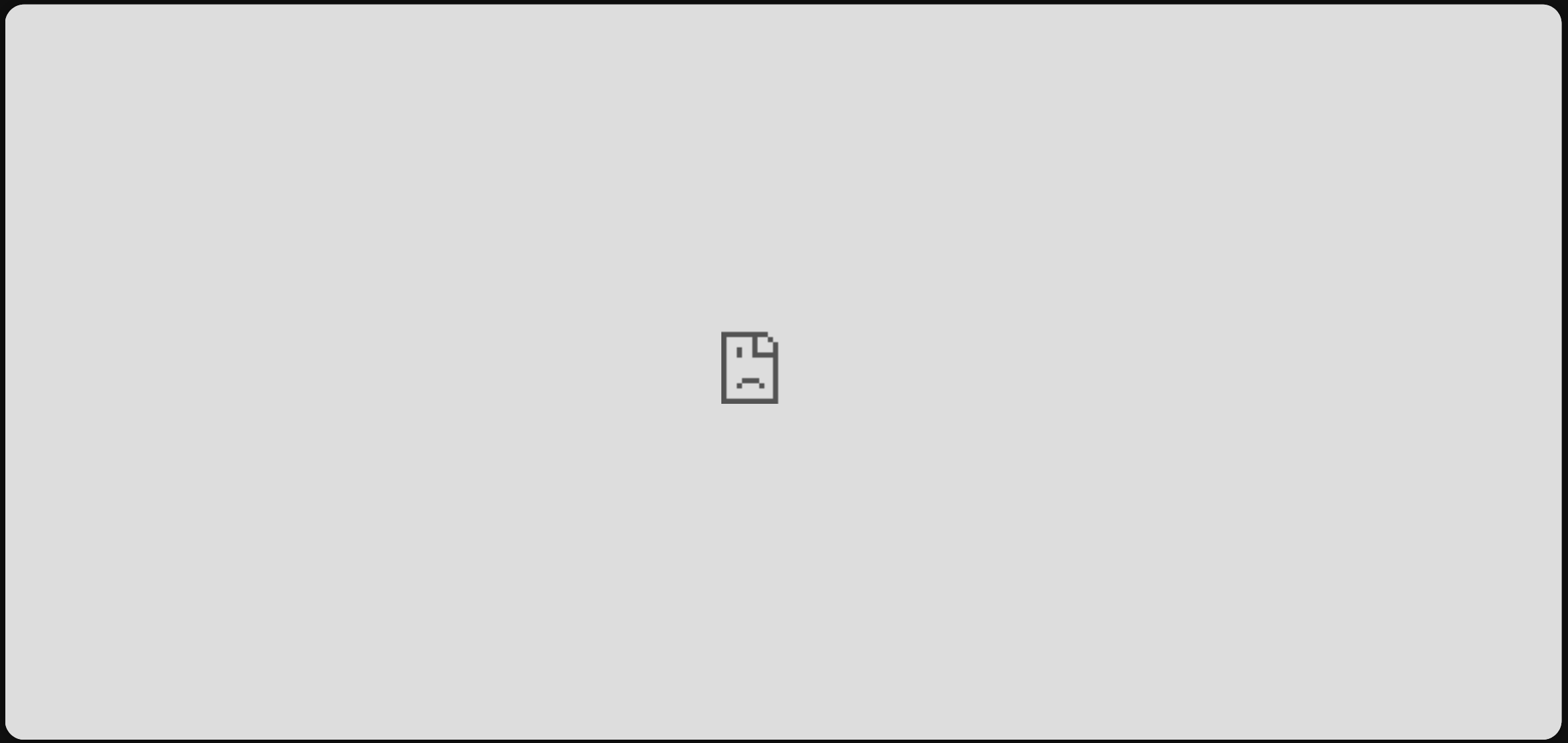
$|u| = \sqrt{3^2 + 4^2} = 5, \quad |v| = \sqrt{4^2 + 3^2} = 5$

$\cos\theta = 24/(5\times5) = 0.96$

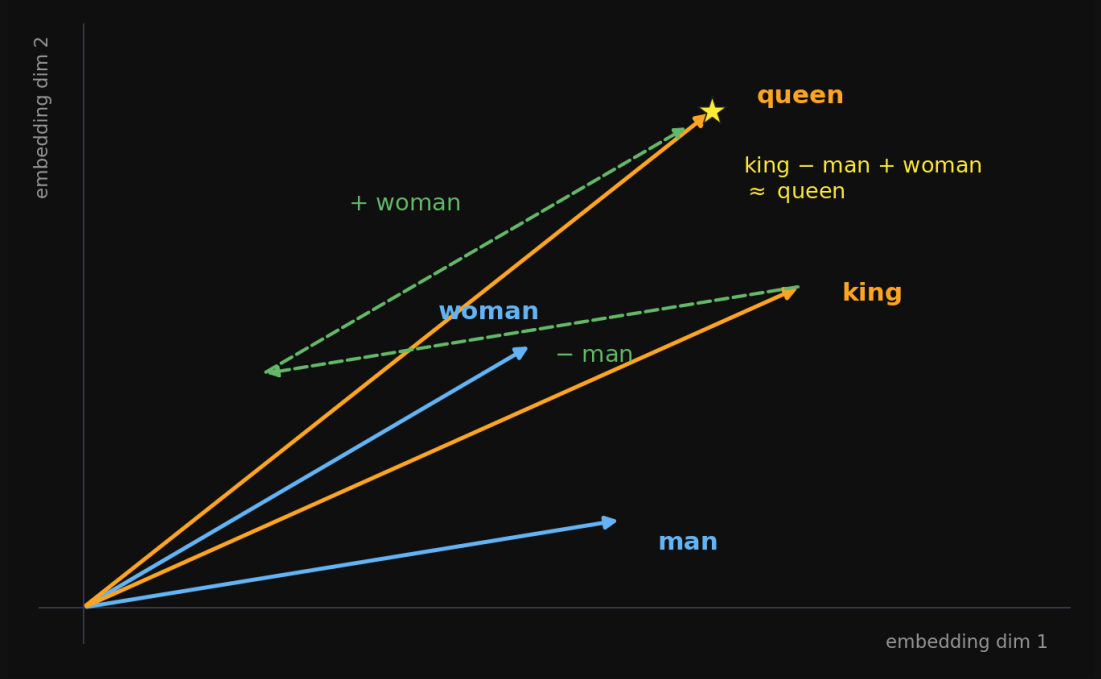
→ nearly the same direction

Multiply matching components, add them up: $\mathbf{u} \cdot \mathbf{v} = 3\times4 + 4\times3 = 24$ — the **dot product**. Divide by the arrows' lengths (5×5, by Pythagoras) and you get 0.96: **cosine similarity** — *pointing the same way* = similar. And look back: the neuron's $\sum_j w_{ij} a_j$ **is a dot product** — every unit scores *how similar its input is to its weights*.

Drive it — bentos as vectors



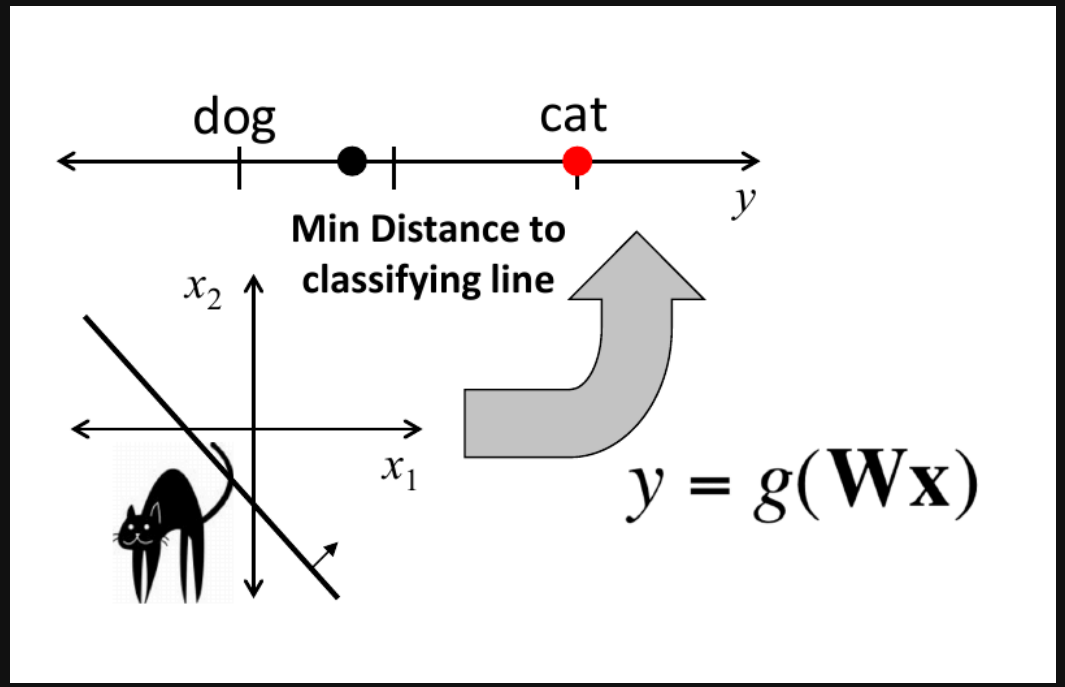
Words as vectors: king − man + woman ≈ queen



Learn a vector per word from co-occurrence (word2vec, Mikolov et al. 2013) and **analogies become arithmetic**. The features weren't designed — they were **learned**. This is the representational substrate every LLM is built on.

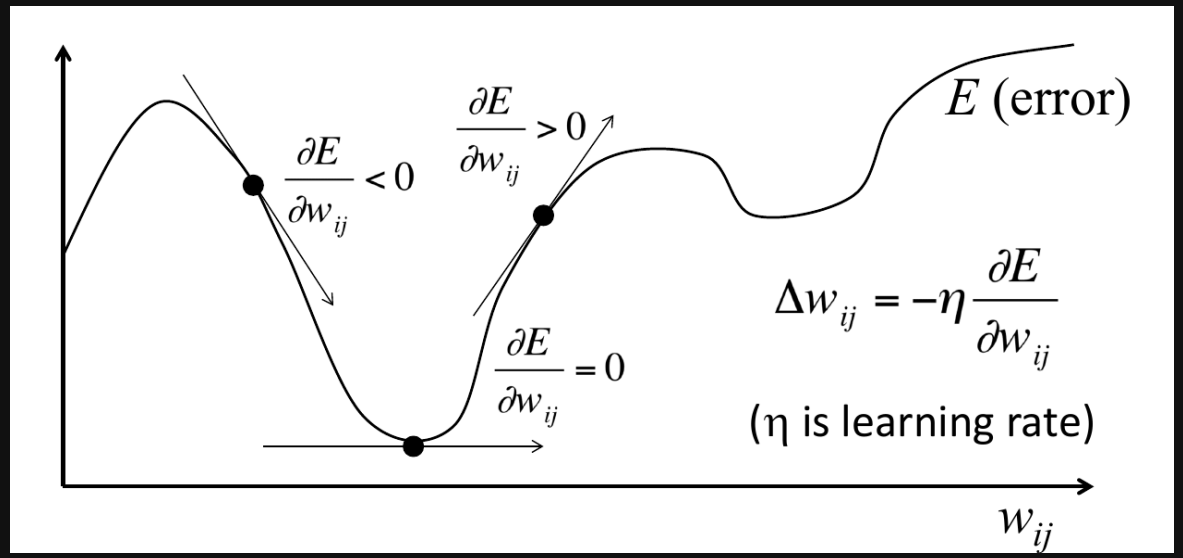
Learn I — the perceptron

The perceptron: a linear classifier



Rosenblatt (1958). Output $y = \text{sign}(\mathbf{w} \cdot \mathbf{x})$ — score the input's **similarity to the weight vector** (the dot product you just met), and sign reports which side of the **decision line** you're on. Learning = **moving the line**.

Error-driven learning = gradient descent



Define an **error** E (how wrong the outputs are). Walk **downhill** on E: nudge each weight in the direction that lowers it, stop where the slope is zero. η = **learning rate** (step size).

You’ve seen this update before

Q-learning (Week 8):

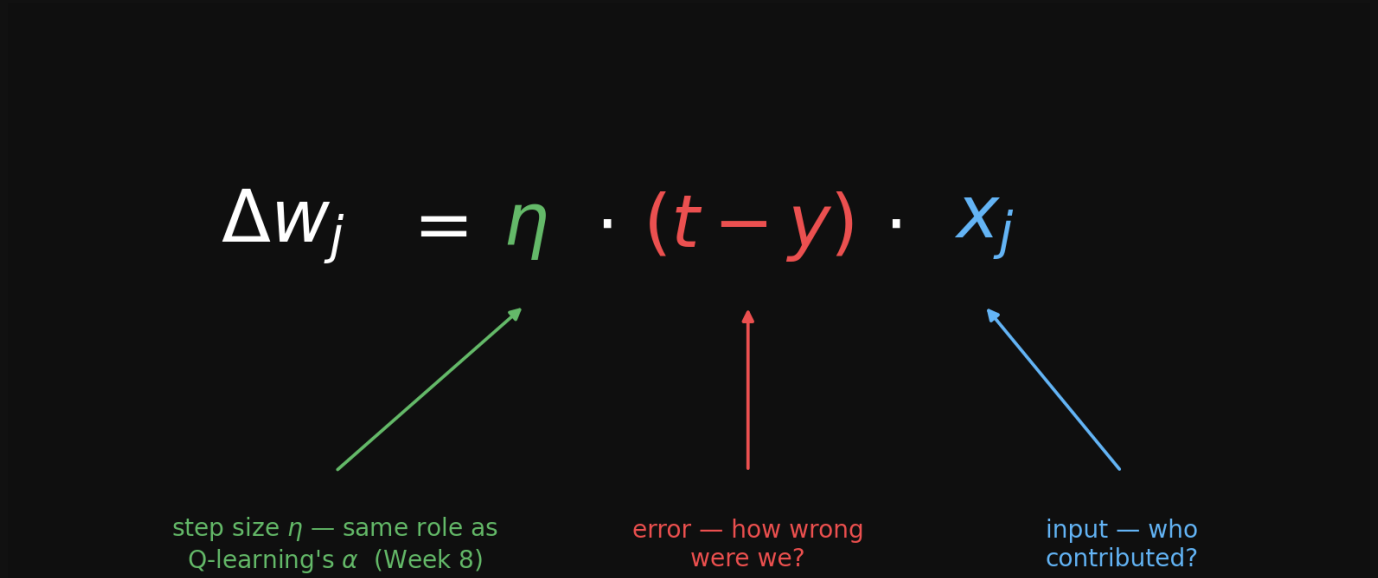
$$Q \leftarrow Q + \alpha \underbrace{\left(\text{target} - Q\right)}_{\text{error}}$$

The delta rule (today):

$$w_j \leftarrow w_j + \eta \underbrace{(t - y)}_{\text{error}} x_j$$

Same shape: **new = old + (step size) × (error) × (who’s responsible).**

Anatomy of the delta rule



For a general activation g, the update also carries a g' term — “correct for how fast the unit is changing.” That g' factor is exactly what will **propagate backward** through a deep net.

Learn II — depth & backpropagation

Quick poll — the limits of a straight line

A single perceptron draws **one straight line**. It can learn AND and OR. Which can it **not** learn?

A. NAND (not-and)

B. XOR (exclusive-or)

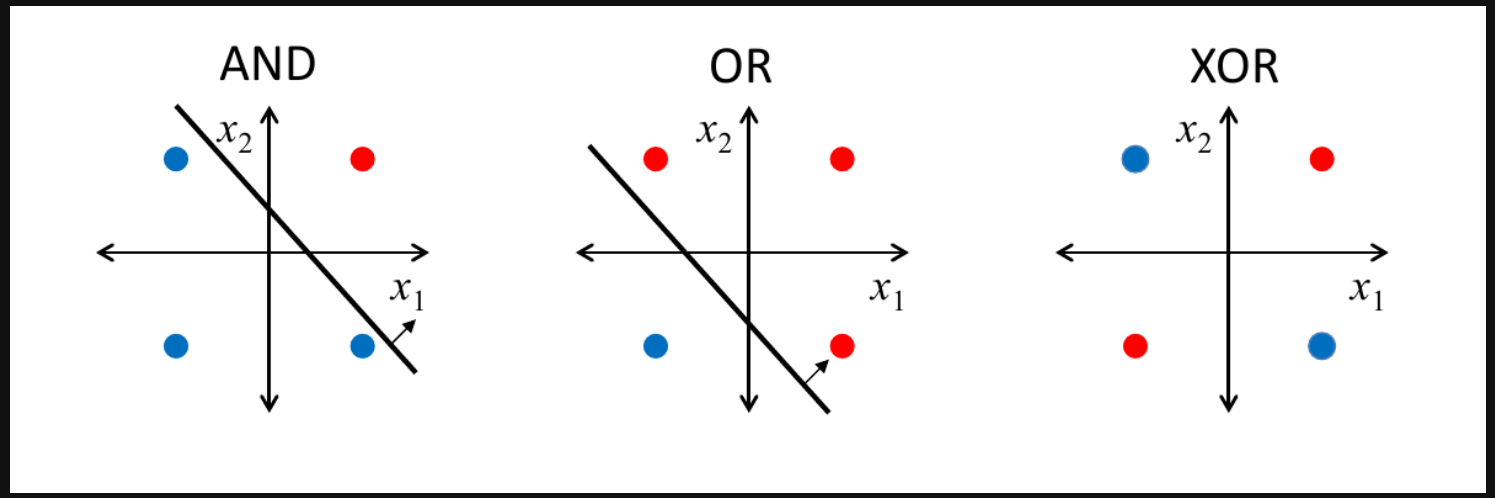
C. it can learn all three

Reveal

B — XOR.

No straight line separates XOR’s two classes. Minsky & Papert (1969) proved it — and argued important abilities (like judging whether a shape is *connected*) need exactly this. Neural-network research went quiet for over a decade.

Why XOR is hard



AND and OR: one line separates the classes. XOR: **no line works** — the two "true" corners are diagonal from each other.

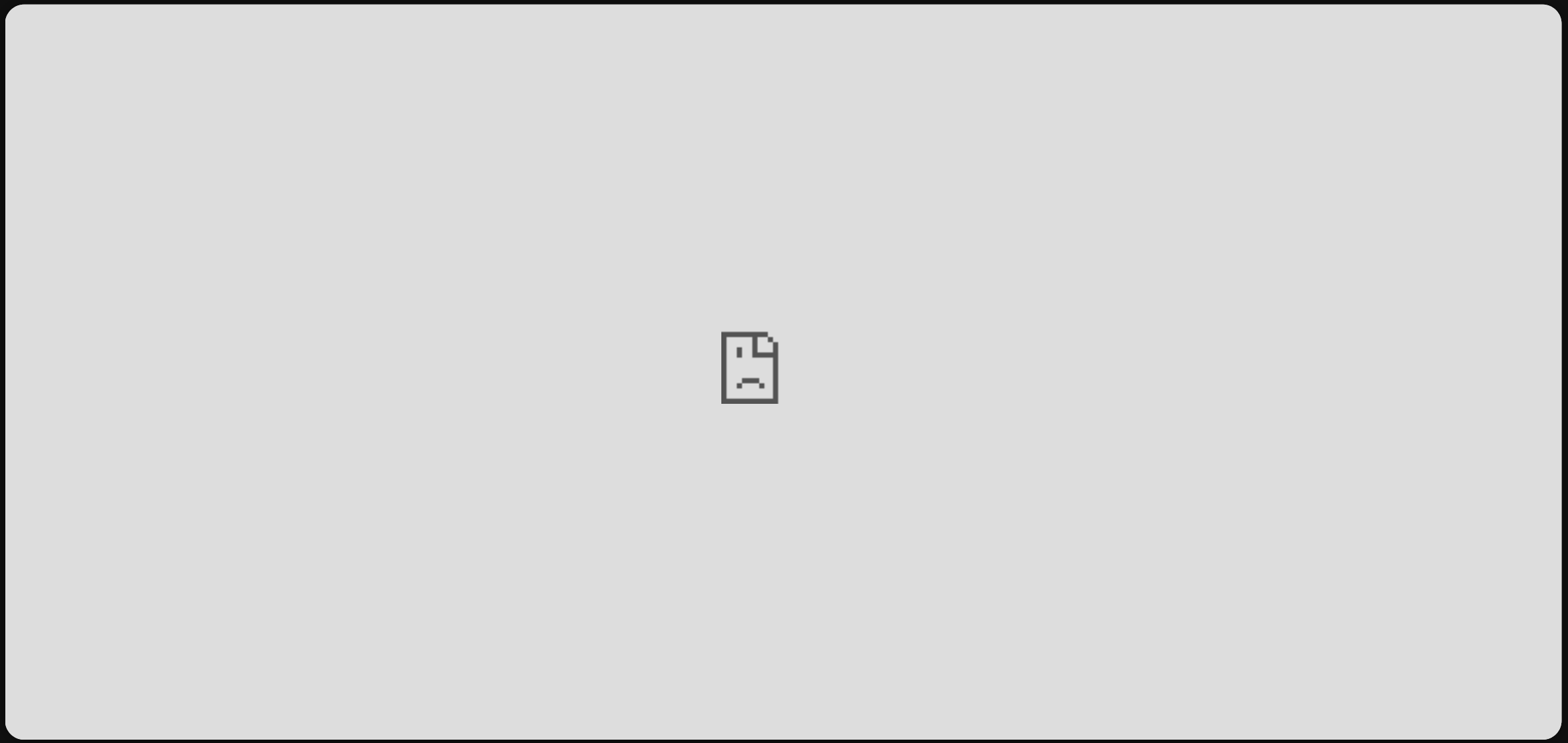
One more tool: the matrix

Adding a **layer** means transforming *every* input vector at once. The machine that does that is a **matrix** — and multiplying by one is just **a dot product per row**:

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2\times 3 + 0\times 4 \\ 0\times 3 + 1\times 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

The first **column** is where $[1, 0]$ lands; the second is where $[0, 1]$ lands — the matrix moves the whole plane.

Drive it — a matrix moves every point



Does stacking layers help? (1/3)

Try two **linear** layers, one after the other:

$$\mathbf{h} = W_1\mathbf{x}, \qquad y = W_2\mathbf{h}.$$

Surely two layers can do more than one?

Does stacking layers help? (2/3)

Multiply it out — two machines run in a row are **one machine** (matrix × matrix is just another matrix):

$$y = W_2(W_1\mathbf{x}) = \underbrace{(W_2W_1)}_{\text{one matrix } W'}\ \mathbf{x} = W'\mathbf{x}.$$

Does stacking layers help? (3/3)

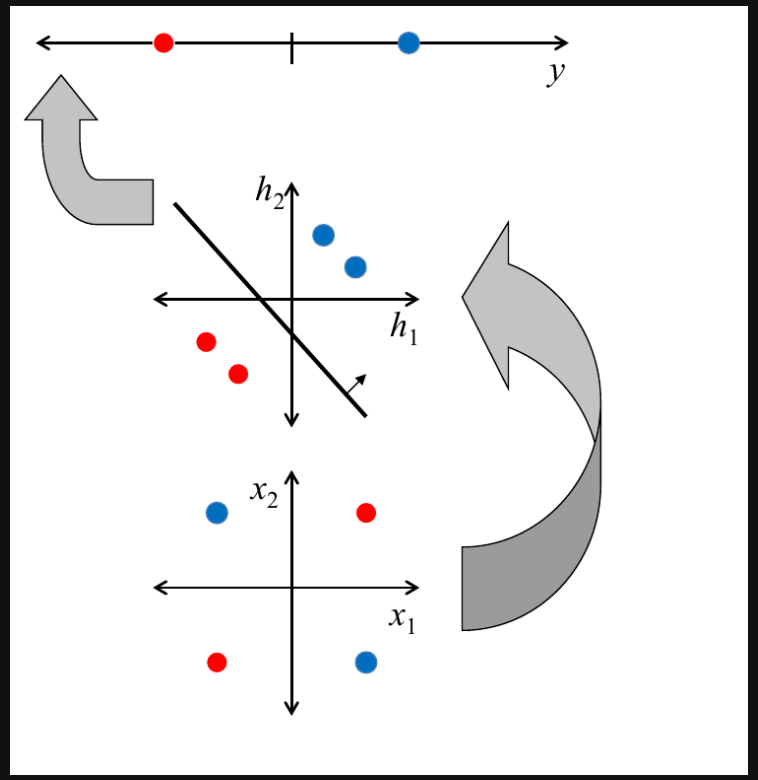
Depth alone buys nothing — it's still one straight line.

The fix is the **nonlinearity**: put an activation g between the layers,

$$\mathbf{h} = g(W_1\mathbf{x}),$$

and the composition can no longer collapse — **that** is why depth needs g.

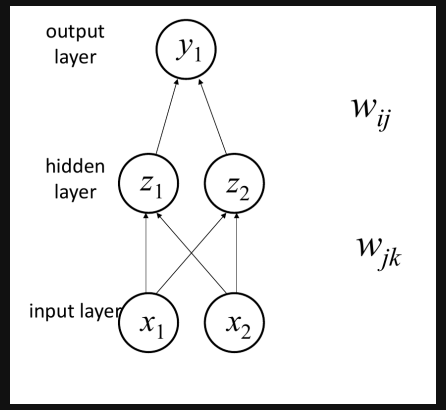
Depth + nonlinearity solves XOR



A nonlinear hidden layer **warps the space** so the two XOR classes become linearly separable — then the output layer draws its line in the warped space.

Drive it — can your net learn XOR?

How do we train the hidden weights? Backprop



Backpropagation (Rumelhart, Hinton & Williams, 1986): run the input forward, measure the output error, then **apportion the blame backward** — each hidden unit's share is how much it fed into the error (that g' term, chained by the chain rule). In very deep stacks those g' factors multiply and can shrink toward zero — the **vanishing-gradient** problem — which is why modern nets add residual connections and normalization (Block 5).

Backprop is just automatic differentiation

You've used JAX all term. Meet its autodiff tool, `jax.grad`: hand it a function, get back the function's gradient — the chain rule, automated.

```
import jax, jax.numpy as jnp

def loss(w, x, t):                      # a tiny network's error
    y = jnp.tanh(x @ w)                 # forward pass
    return jnp.mean((y - t) ** 2)       # squared error

grad_fn = jax.grad(loss)               # <- reverse-mode autodiff = backprop
g = grad_fn(w, x, t)                   # the gradient, one call
w = w - 0.1 * g                         # one gradient-descent step
```

Reverse-mode autodiff — what `jax.grad` computes — *is* backpropagation. Backprop is not a bespoke algorithm; it's the chain rule automated, in one call. No hand-derived update rules.

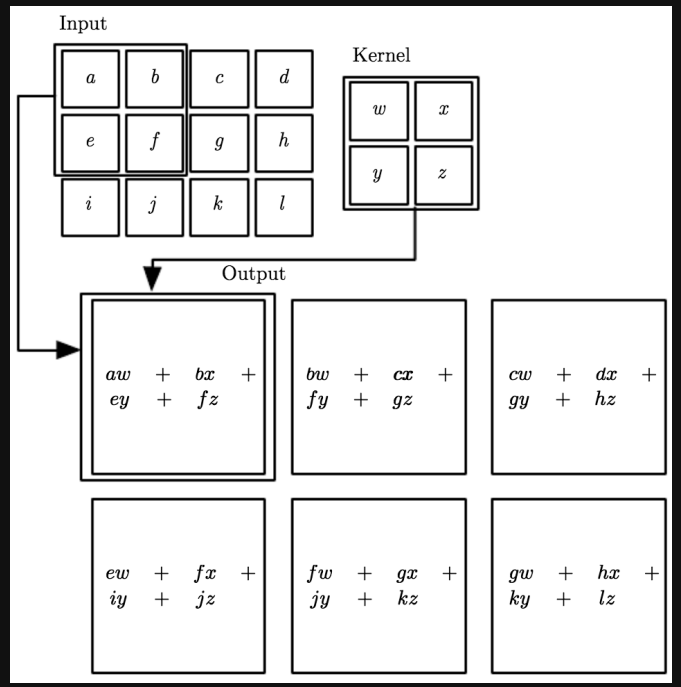
Represent II — architecture as prior

Architecture is an inductive bias — a prior

A fully-connected net treats every input the same. Real architectures **bake in an assumption** about the data — which is exactly a **prior** (the Bayesian word from Week 4 onward).

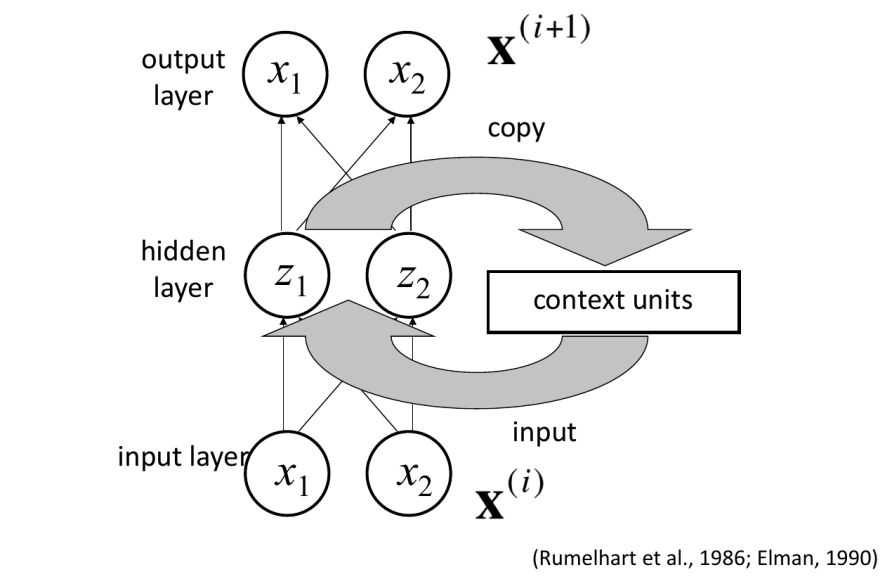
- **CNN:** nearby pixels matter together, and the same pattern can appear anywhere.
- **RNN:** the data is a **sequence**; process it one step at a time.

CNNs: bake in translation invariance



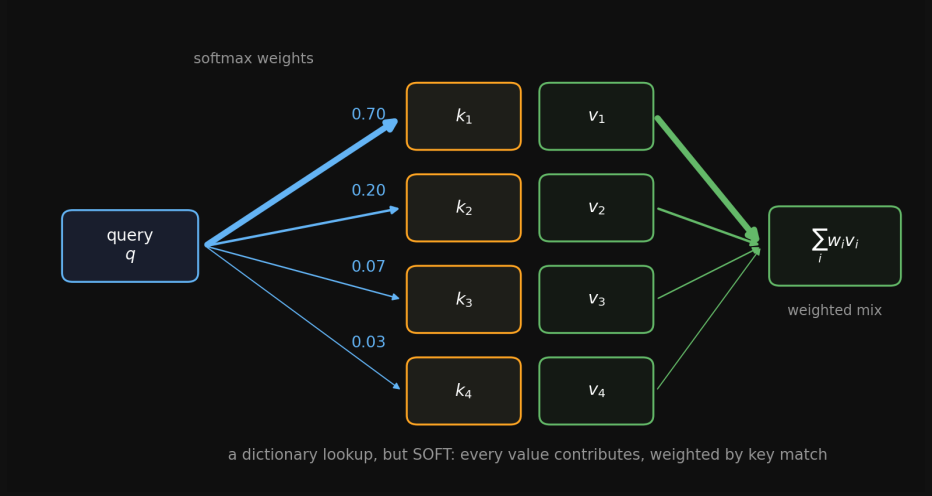
Slide one small **kernel** across the image and reuse its weights everywhere (**weight sharing**). A cat is a cat wherever it appears — **translation invariance**, built in rather than learned from scratch.

RNNs: bake in sequence — but a bottleneck



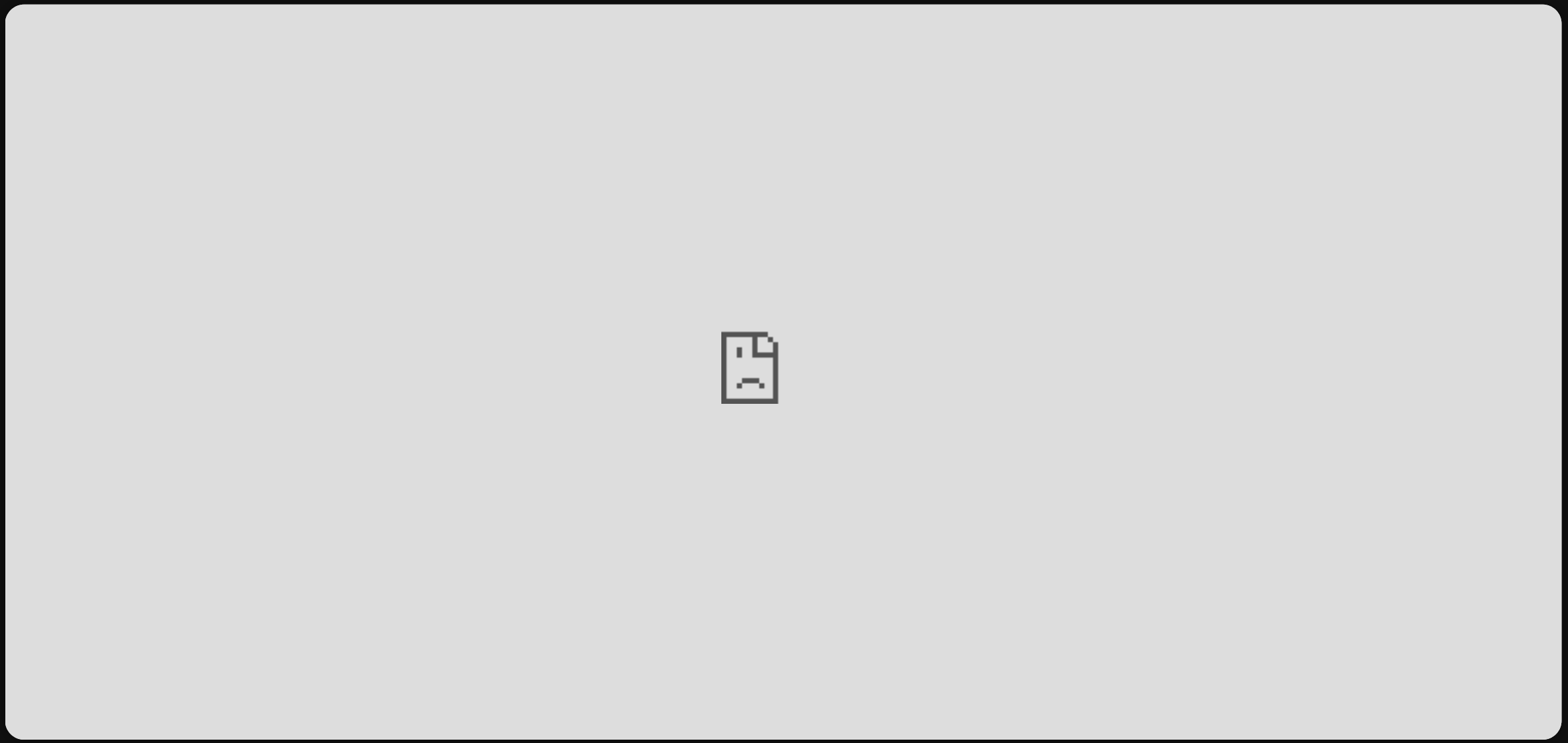
Feed the hidden state back in at each step (Elman, 1990) — the net carries a memory of what came before. But everything must squeeze through **one hidden state**, step by step: a famous **bottleneck**. RNNs learned *local* structure well and *global* structure poorly.

Attention: a soft dictionary lookup

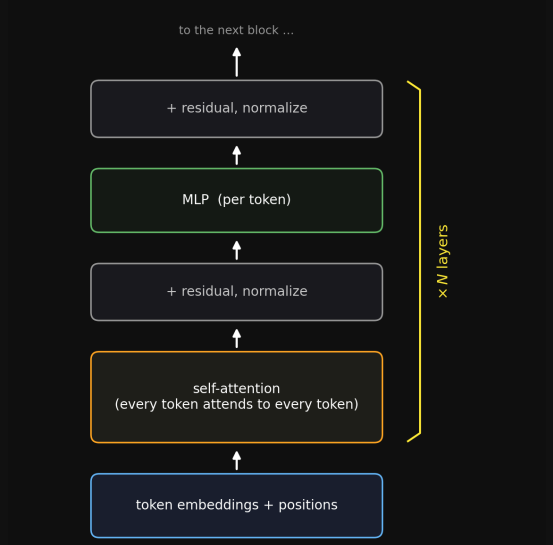


Instead of one bottleneck, let **every token look at every other token**. Each token issues a **query**; it's matched against every token's **key** — the match score is a **dot product**, the same similarity you met this morning. The **softmax** of those scores weights a blend of the **values** — the same softmax from Week 9, sharpness β and all.

Drive it — attention as lookup



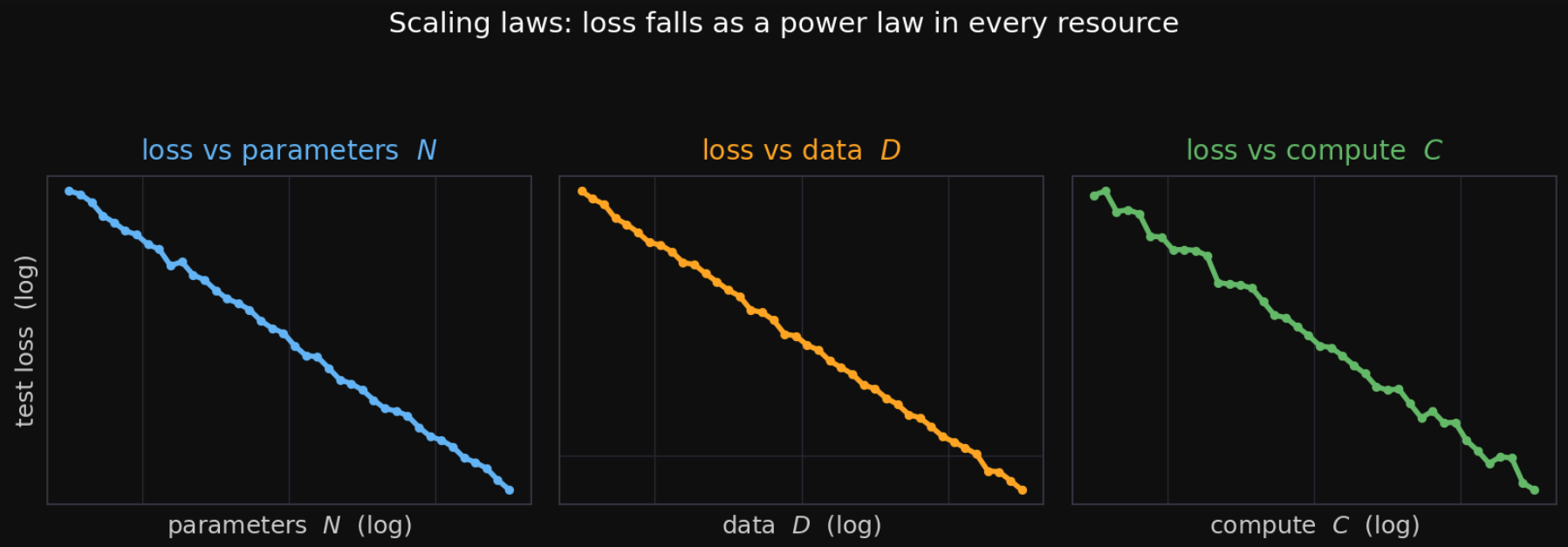
The transformer: attention + MLP, stacked



One block = **self-attention** (mix information across tokens) + a **per-token MLP** (transform each), wrapped in **residual connections** and **normalization** (the vanishing-gradient fixes). Stack N of them. No recurrence, no bottleneck — this is the architecture behind every modern LLM.

Understand — scale, emergence, in-context learning

Scaling laws: bigger is (predictably) better



Test loss falls as a **straight line on a log-log plot** against parameters, data, and compute (Kaplan et al. 2020; Hoffmann et al. 2022). Performance is, to a surprising degree, **predictable from scale alone** — no new idea required.

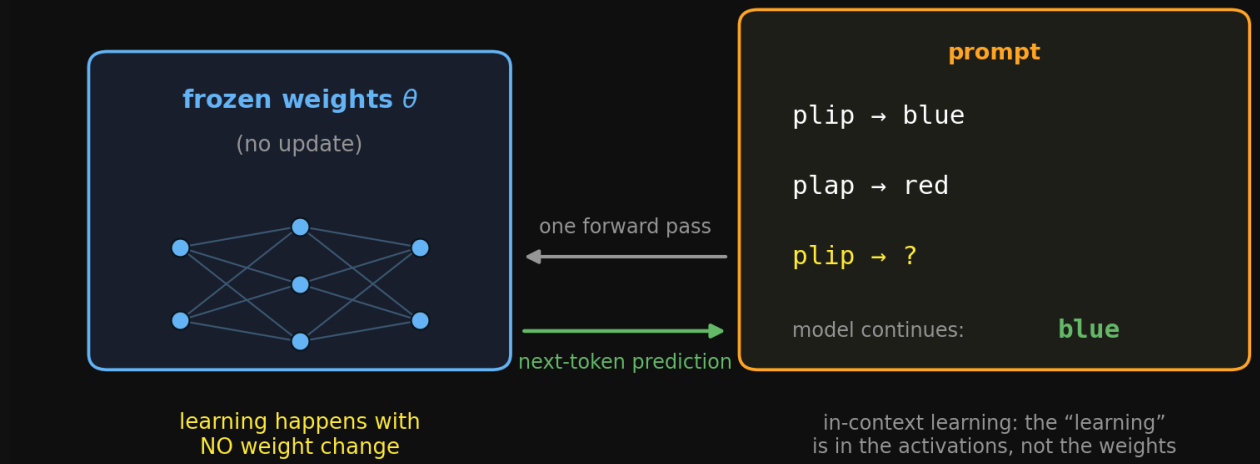
Emergence — and a caveat

Some abilities appear to switch on **suddenly** past a scale threshold — arithmetic, multi-step reasoning — abilities the smaller model seemed not to have at all (“**emergent abilities**”).

Caveat (Schaeffer et al. 2023): some “emergence” is a **mirage** of the metric — switch to a smoother score and the jump becomes a gentle curve. Real or measured? Still debated.

📋 Lake scorecard: in-context learning is real **learning-to-learn** — adapting from a few examples — one of Lake et al.’s four asks. Whether it’s *compositional* is exactly the debate on the next slide.

In-context learning: learning without weight updates



Give a **frozen** model a few examples in its prompt — *plip → blue*, *plap → red*, *plip → ?* — and it continues the pattern. Nothing was retrained; the "learning" happened entirely **in the context window**.

Quick poll — where did the learning go?

A frozen LLM picks up a made-up rule from 3 prompt examples. Where does what-it-just-learned **live**?

A. in updated weights

B. in the context window — weights unchanged

C. in a retrieval database

Reveal

B — in the context window; the weights never change.

Conditioning on examples without updating parameters looks a lot like **inference**, not training. That resemblance is the basis of a striking — and **contested** — claim.

Frozen weights, adaptive answer — how?

- **Fixed weights are a fixed *function*, not a fixed answer.** A sorting routine's code never changes, yet it sorts any list. Weights are the code; the prompt is the input.
- **Pretraining wrote a *procedure*, not facts.** To predict the next token across an ocean of mixed text, the net must read the prefix, infer *what kind of text this is*, and predict to match — the loss-minimizing move.
- **That "kind" is a latent concept** — a topic, a task, the plip/plap rule. *Read examples → infer the concept → continue* is inference, run in the forward pass.
- **So the weights hold an *amortized* inference engine** — trained once to emit an inference's answer in one pass. It never stored "plip → blue," only *how to read examples and continue*.

The Bayesian lens — and its critics

The claim — ICL ≈ implicit Bayes. Pretraining fits a **prior** over latent patterns; the prompt's examples are **observations**; the continuation is the **posterior predictive**. Pretraining is *empirical Bayes* — estimating the prior itself from data. (Xie et al. 2022; Ye et al. 2024)

The pushback — the math doesn't check out. If prompt examples are exchangeable (order shouldn't matter), the running prediction must be a **martingale**: no systematic drift as examples arrive. Real LLM predictions **violate** that. (Falck et al. 2024)

The lens is *suggestive*, not settled — and it's the payoff of everything from **Week 4's hierarchical Bayes** onward.

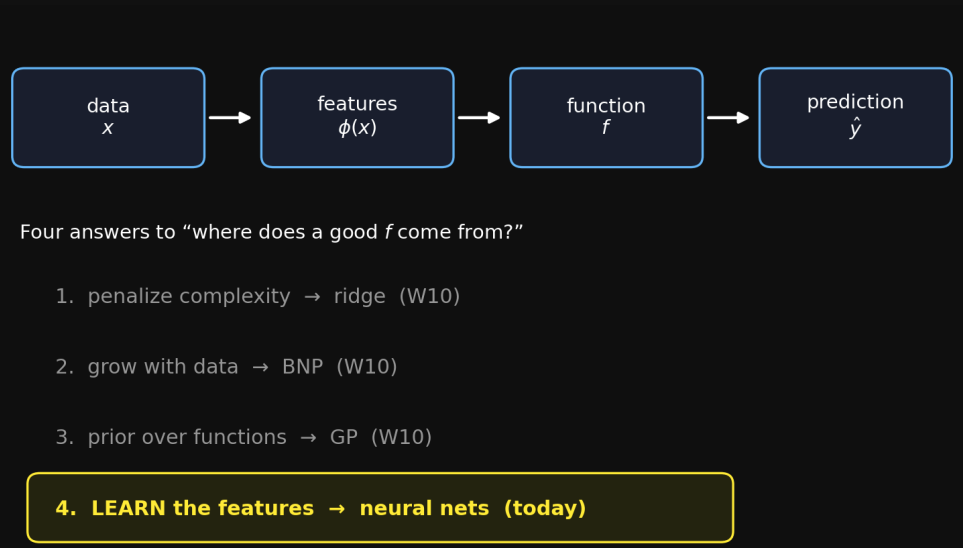
How these get aligned → next week

A pretrained model predicts text; making it **helpful and safe** is a second step — **RLHF** (Week 9): fit a reward model to human preferences (Bradley–Terry), then RL the policy. Same inverse-RL machinery, now aimed at values.

- **Next week (Week 12):** ethics, fairness, adversarial examples, alignment.

Where this leaves us

One question, four answers



“Where does the hypothesis space come from?” — penalize complexity (**ridge**), grow with data (**BNP**), a prior over functions (**GP**), or **learn the features** (neural nets). The first three fix the kernel; the fourth learns it — and at infinite width, it circles back to a **GP** (Week 10).

Lake’s scorecard, settled

Deep nets deliver — genuinely

- Learned representations
- Similarity
- Scale

Still missing (Lake et al. 2017)

- Compositional generalization
- Causal, model-based understanding

Meta-learned systematicity comes only **with effort** (Lake & Baroni, 2023); a 2025 **multimodal** re-test on Lake’s own domains stays **short of people** (Schulze Buschoff et al., 2025).

Honest verdict: the 1988 → 2017 → 2025 critique still bites *and* learned features are powerful. Both true.

Next week

WEEK 12 · JULY 17

Ethics, fairness & alignment

- **Fairness** & bias in learned models
- **Adversarial examples** — where nets break
- **Alignment** — steering models toward human values
- **Final-project presentations** — bring a laptop

Meanwhile — play the two widgets: XOR net & attention lookup, linked from the slides.