DEEP LEARNING FROM SCRATCH

A Guide for Self-Learners

Episode II

The Art of Descent

AI evolves fast, but speed isn't the enemy: confusion is.
So let's go fast, in the right direction.

Why this mini-guide?

In Episode I we understood the neuron. In Episode II we make it move*: gradients, the chain rule, and the exact signal that tells a model how to learn.*

How this mini-guide?

No shortcuts, no magic formulas dropped from the sky.
We derive every link step by step, until the update rules emerge cleanly.

"The most difficult thing is the decision to act; the rest is merely tenacity."
– Amelia Earhart

Pierre Chambet

November 2025

Part of the *Deep Learning From Scratch* series — an independent reconstruction of AI.

1) Why Gradients Exist (and Why We Need Them)

Picture a night hike. Your parameters $(\mathbf{w}, b)$ are your position on a dark slope called *loss*. You can't see the valley, only feel the ground beneath your feet. The *gradient* is that ground's tilt under your boots: it points uphill. So you step the other way. Small step, check, repeat. That's learning.

That principle is the **gradient**.

What the gradient really answers

1. If I nudge a parameter, does the loss rise or fall? *(direction)*
2. By how much does it change, right here? *(magnitude)*
3. Which parameter matters most for this mistake? *(priority)*

In Episode I, the neuron turned inputs $\mathbf{x}$ into a score $z = \mathbf{w}^\top \mathbf{x} + b$, then a probability $a = \sigma(z)$, then a loss $\ell(a, y)$. Here's the catch: parameters don't touch the loss *directly*. They influence z, which shapes a, which sets ℓ. To know how to move the parameters, we must trace that influence back from the loss to the score to the weights. That tracing has a name: the **chain rule**.

In practice, the gradient — let's denote it $\nabla\mathcal{J}$ — is a tiny, clear instruction: change the parameter θ by a step of size η in the direction $-\nabla\mathcal{J}$:

$$\theta \leftarrow \theta - \eta \nabla\mathcal{J}.$$

If the neuron was overconfident $(a > y)$, the slope points to "*less*." If it was timid $(a < y)$, the slope points to "*more*." No guesswork, no mystique — just the terrain telling you how to walk.

Why gradients? Because speed without orientation is noise. The gradient gives orientation — a principled way to turn error into motion.

2) Notations (same as Episode I) — and what they mean in the real world

In Episode I, we already met the main characters of our story: the inputs, the weights, the bias, the score, the activation, and the loss.

Here, we simply bring them all back on stage — this time, with both their mathematical face and their real-world meaning.

$\mathbf{x} \in \mathbb{R}^d$	**Input features (vector).** *Real world: measurable attributes — pixel values, sensor data, or leaf measurements.*
$\mathbf{w} \in \mathbb{R}^d$	**Weights (vector).** *Real world: how much each feature matters in the model's current belief.*
$b \in \mathbb{R}$	**Bias (scalar).** *Real world: a constant that shifts all predictions — the model's natural leaning before seeing any input.*
$z = \mathbf{w}^\top \mathbf{x} + b$	**Linear score.** *Real world: raw evidence before any notion of probability.*
$a = \sigma(z) = \dfrac{1}{1 + e^{-z}}$	**Activation (sigmoid output).** *Real world: probability or confidence between* 0 *and* 1.
$y \in \{0, 1\}$	**True label.** *Real world: ground truth — toxic/non-toxic, cat/not-cat, digit* 7*/not* 7.
$\ell(a, y) = -[y \log a + (1 - y) \log(1 - a)]$	**Loss for one example.** *Real world: penalty for being wrong, calibrated in probability space.*
$\mathcal{J} = \dfrac{1}{m} \sum_{i=1}^{m} \ell(a_i, y_i)$	**Mean loss over a batch of m.** *Real world: the average "wrongness" across multiple samples.*

Important distinction. a is a *probability* (post-sigmoid), while z is a *raw score* (pre-sigmoid). Confusing the two is common; the loss is computed in terms of a because that's where meaning — and probability — live.

Vector refresher — one minute before the math. A vector is simply an *ordered list of numbers.* For example, an input with two features can be written as:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \qquad \text{and its weights as} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

The little arrow (or boldface) just means "this object has several components." You can think of $\mathbf{x}$ as a column of measured values (height, width, pixels, etc.), and $\mathbf{w}$ as a column of how important each of those values is.

The **dot product** combines them into a single number:

$$\mathbf{w}^\top \mathbf{x} = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d.$$

The symbol $^\top$ ("transpose") just turns a column into a row, so that when we multiply, each w_k meets its x_k — one by one — and all are summed. That's why we call it a **linear combination** of the features.

Finally, we add a single scalar b (the bias), giving:

$$z = \mathbf{w}^\top \mathbf{x} + b.$$

3) The Chain Rule — how learning connects cause and consequence

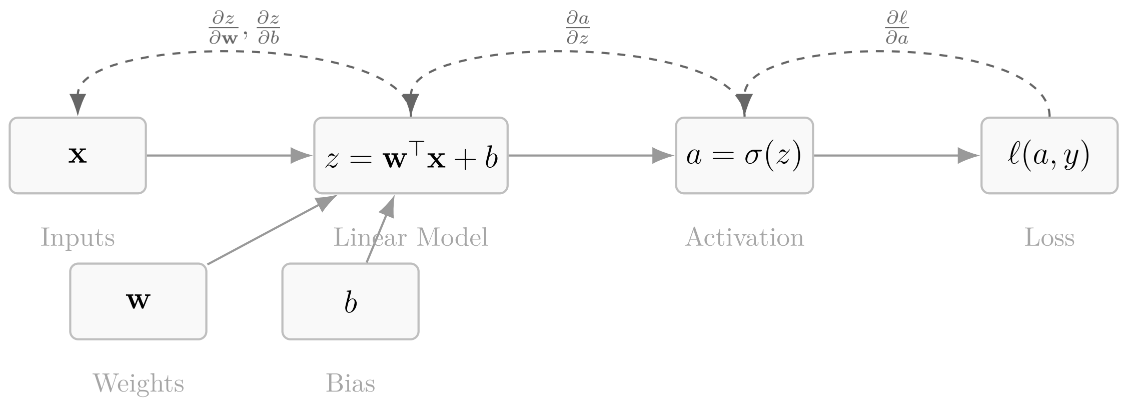
Intuition. Every parameter in a model affects the loss — but not directly. It does so through a chain of intermediate variables: weights shape the score, the score shapes the probability, and the probability shapes the final loss. The **chain rule** is the mathematical link that lets us follow this influence from start to finish.

3.1 The chain of dependencies

In our neuron, the loss for one example depends on three successive steps:

$$z = \mathbf{w}^\top \mathbf{x} + b, \qquad a = \sigma(z), \qquad \ell = \ell(a, y).$$

Each arrow represents a dependency. If we slightly change a weight w_k, z will change; this will modify a; and a will in turn affect the loss ℓ. The chain rule tells us how to quantify this entire cascade.



$\frac{\partial z}{\partial \mathbf{w}}, \frac{\partial z}{\partial b}$
$\frac{\partial a}{\partial z}$
$\frac{\partial \ell}{\partial a}$
$\mathbf{x}$
$z = \mathbf{w}^\top \mathbf{x} + b$
$a = \sigma(z)$
$\ell(a, y)$
Inputs
Linear Model
Activation
Loss
$\mathbf{w}$
b
Weights
Bias

Now we can see what the real question is.
To understand how the weights and bias affect the loss, we must trace the influence step by step:

- How does the loss change when the prediction a changes?
- How does the prediction a change when the score z changes?
- How does the score z change when the parameters $(\mathbf{w}, b)$ change?

Each question isolates one link in the chain.

The chain rule in words: To understand how the loss changes when a parameter changes, we multiply how each intermediate variable depends on the one before it. It's the mathematical equivalent of tracing cause and effect through the model.

3.2 Gradient refresher — from tiny changes to the chain rule

The picture. A gradient is about *how much a quantity changes when we nudge something else.* Think of it as a local "sensitivity meter." If we change a parameter by a tiny amount, what is the corresponding tiny change in the loss?

Notation. We use the curly-∂ symbol to emphasize that we are talking about *partial* derivatives — i.e., the rate of change with respect to one variable while keeping the others fixed. For example,

$$\frac{\partial \ell}{\partial a} \quad \text{means: "if I nudge } a \text{ a little, how much does } \ell \text{ change right here?"}$$

Differentials: the microscopic bookkeeping. At a single point, the loss reacts to a tiny change $\mathrm{d}a$ as

$$\underbrace{\mathrm{d}\ell}_{\text{change}} \approx \underbrace{\frac{\partial \ell}{\partial a}}_{\text{slope}} \times \underbrace{\mathrm{d}a}_{\text{step}}$$

This is the local linear approximation: for *small* moves, curvature is negligible, and the best predictor of the total change is slope $\times$ step.

Chaining small effects. In our neuron, a itself comes from z, so a tiny change $\mathrm{d}z$ produces

$$\mathrm{d}a \approx \frac{\partial a}{\partial z}\,\mathrm{d}z.$$

And z comes from the linear model; nudging a single weight w_k by $\mathrm{d}w_k$ creates

$$\mathrm{d}z \approx \frac{\partial z}{\partial w_k}\,\mathrm{d}w_k.$$

Putting the chain together. Substitute these tiny changes into one another:

$$\mathrm{d}\ell \;\approx\; \frac{\partial \ell}{\partial a}\underbrace{\mathrm{d}a}_{\approx\,(\partial a/\partial z)\,\mathrm{d}z} \;\approx\; \frac{\partial \ell}{\partial a}\frac{\partial a}{\partial z}\underbrace{\mathrm{d}z}_{\approx\,(\partial z/\partial w_k)\,\mathrm{d}w_k} \;\approx\; \Big(\frac{\partial \ell}{\partial a}\Big)\Big(\frac{\partial a}{\partial z}\Big)\Big(\frac{\partial z}{\partial w_k}\Big)\,\mathrm{d}w_k.$$

Read it as: *total change = product of three local sensitivities* $\times$ *the tiny nudge.* This is the chain rule in differential form.

Why this is coherent. Each factor answers a simple local question:

$\dfrac{\partial \ell}{\partial a}$ (loss reacts to prediction), $\dfrac{\partial a}{\partial z}$ (activation reacts to score), $\dfrac{\partial z}{\partial w_k}$ (score reacts to parameter).

Multiplying them composes the effects along the computation path — exactly how influence flows through the model.

And now we have the answer to our question: How the weights and bias affect the loss ?

CHAIN RULE

$$\boxed{\frac{\partial \ell}{\partial w_k} = \frac{\partial \ell}{\partial a}\cdot\frac{\partial a}{\partial z}\cdot\frac{\partial z}{\partial w_k}} \qquad \boxed{\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial a}\cdot\frac{\partial a}{\partial z}\cdot\frac{\partial z}{\partial b}}$$

Practical note. **1. Local meaning.** These formulas are inherently *local*: they describe what happens for infinitesimal changes — tiny nudges, not leaps. That's why we combine them with a learning rate η: a small, controlled step that turns mathematical sensitivity into gradual learning. Training is nothing more than this rhythm, repeated over and over: *measure the gradient, move a little, repeat.*

2. Each term's role. Every derivative answers a specific, local question:

- How does the loss react when the prediction a changes? $(\partial\ell/\partial a)$
- How does the prediction a react when the score z changes? $(\partial a/\partial z)$
- How does the score z react when the parameters shift? $(\partial z/\partial w_k,\ \partial z/\partial b)$

Together, these terms form the complete gradient — a precise thread linking cause (the parameters) to consequence (the loss).

In essence: the gradient is the invisible bridge between cause and effect. It carries information backward — from the loss to every parameter — telling each one how it contributed to the error, and how it should change next.
This backward flow of correction is what we call **backpropagation**.

4) Computing the three small gradients

Now that we know what a gradient means, let's compute the three links that make learning possible. Each is a small derivative — simple on its own — but together, they form the entire learning signal of a neuron.

4.1 First link — how the loss reacts to the prediction $(\partial\ell/\partial a)$

Setup. For a single example with label $y \in \{0,1\}$ and prediction $a \in (0,1)$ (post–sigmoid), the (binary) log-loss is

$$\ell(a,y) \;=\; -\Big[y\,\log a \;+\; (1-y)\,\log(1-a)\Big].$$

We differentiate *with respect to* a (treating y as constant).

Rules we will use.

R1. $\dfrac{d}{dx}\big(\log x\big) = \dfrac{1}{x}$ for $x > 0$.

R2. $\dfrac{d}{dx}\big(\log(1-x)\big) = \dfrac{-1}{1-x}$ for $x < 1$ (by chain rule with $u(x) = 1-x$, $u'(x) = -1$).

R3. Linearity: $\dfrac{d}{dx}(c_1 f(x) + c_2 g(x)) = c_1 f'(x) + c_2 g'(x)$.

R4. Constants differentiate to 0; here y and $1-y$ are constants w.r.t. a.

Step-by-step differentiation. Start from

$$\ell(a,y) = -\Big[y\,\log a + (1-y)\,\log(1-a)\Big].$$

Apply linearity and pull out the outer minus sign:

$$\frac{\partial\ell}{\partial a} = -\left[y \cdot \frac{\partial}{\partial a}(\log a) \;+\; (1-y)\cdot\frac{\partial}{\partial a}\big(\log(1-a)\big)\right].$$

Now use **R1** on $\log a$ and **R2** on $\log(1-a)$:

$$\frac{\partial\ell}{\partial a} = -\left[y\cdot\frac{1}{a} \;+\; (1-y)\cdot\left(\frac{-1}{1-a}\right)\right].$$

Distribute the minus sign carefully:

$$\frac{\partial\ell}{\partial a} = -\frac{y}{a} \;-\; (1-y)\cdot\left(\frac{-1}{1-a}\right) = -\frac{y}{a} \;+\; \frac{1-y}{1-a}.$$

Equivalently (factor the outer minus sign as in many texts):

$$\boxed{\frac{\partial\ell}{\partial a} = -\left(\frac{y}{a} - \frac{1-y}{1-a}\right) = \frac{1-y}{1-a} - \frac{y}{a}}$$

Optional compact form. Sometimes we combine over a common denominator:

$$\frac{\partial \ell}{\partial a} = \frac{(1-y)a - y(1-a)}{a(1-a)} = \frac{a - ay - y + ay}{a(1-a)} = \frac{a-y}{a(1-a)}.$$

So another equivalent expression is

$$\boxed{\frac{\partial \ell}{\partial a} = \frac{a-y}{a(1-a)}}$$

(Useful later when multiplying by $\partial a/\partial z = a(1-a)$.)

Domain notes (important for correctness). Since $a \in (0,1)$, both $\log a$ and $\log(1-a)$ are well-defined (no $\log 0$). That's why, in practice, we never let a be exactly 0 or 1 (we clip or add ϵ).

Sanity checks.

- If $y = 1$: $\dfrac{\partial \ell}{\partial a} = -\dfrac{1}{a} < 0$ — increasing a decreases the loss (as it should).
- If $y = 0$: $\dfrac{\partial \ell}{\partial a} = \dfrac{1}{1-a} > 0$ — decreasing a decreases the loss (as it should).

Takeaway. This derivative is the "feedback" from truth to prediction: it tells how the loss changes for a small change in the probability a. It is the first link of the chain.

4.2 Second link — how the activation reacts to the score ($\partial a/\partial z$)

Setup. We use the logistic (sigmoid) activation

$$a(z) \;=\; \sigma(z) \;=\; \frac{1}{1+e^{-z}}.$$

We differentiate a with respect to its input z.

Rules we will use.

R1. Chain rule: if $h(z) = f(g(z))$, then $h'(z) = f'(g(z)) \cdot g'(z)$.

R2. Reciprocal rule: $\dfrac{d}{dz}\Big(\dfrac{1}{u(z)}\Big) = -\dfrac{u'(z)}{u(z)^2}$.

R3. Exponential: $\dfrac{d}{dz}(e^{-z}) = -\,e^{-z}$.

R4. Quotient rule (optional route): for $a(z) = \frac{N(z)}{D(z)}$, $a'(z) = \dfrac{N'(z)D(z) - N(z)D'(z)}{D(z)^2}$.

R5. Algebraic identity (logistic): $\dfrac{1}{1+e^{-z}} = a \quad \Rightarrow \quad 1 - \dfrac{1}{1+e^{-z}} = 1 - a$.

Derivation (Route A: reciprocal + chain). Write $a(z) = \big(1 + e^{-z}\big)^{-1}$. Let $u(z) = 1 + e^{-z}$.

$$\frac{da}{dz} \;=\; -\frac{u'(z)}{u(z)^2} \quad \text{by } \mathbf{R2}.$$

Differentiate u:

$$u'(z) \;=\; \frac{d}{dz}\big(1 + e^{-z}\big) \;=\; -\,e^{-z} \quad \text{by } \mathbf{R3}.$$

Plug in:

$$\frac{da}{dz} \;=\; -\frac{-\,e^{-z}}{\big(1+e^{-z}\big)^2} \;=\; \frac{e^{-z}}{(1+e^{-z})^2}.$$

Now express everything in terms of a. Since $a = \dfrac{1}{1+e^{-z}}$, we have

$$\frac{1}{1+e^{-z}} = a \quad \Rightarrow \quad 1 - \frac{1}{1+e^{-z}} = 1 - a \quad \text{by } \mathbf{R5}.$$

Therefore,

$$\frac{da}{dz} \;=\; \underbrace{\frac{1}{1+e^{-z}}}_{a} \cdot \underbrace{\Big(1 - \frac{1}{1+e^{-z}}\Big)}_{(1-a)} \;=\; a(1-a).$$

$$\boxed{\frac{\partial a}{\partial z} \;=\; a(1-a)}$$

Derivation (Route B: quotient rule, for completeness). Let $N(z) = 1$ and $D(z) = 1 + e^{-z}$. Then by **R4**,

$$\frac{da}{dz} \;=\; \frac{N'(z)\,D(z) - N(z)\,D'(z)}{D(z)^2} \;=\; \frac{0 \cdot (1+e^{-z}) - 1 \cdot \big(-\,e^{-z}\big)}{(1+e^{-z})^2} \;=\; \frac{e^{-z}}{(1+e^{-z})^2}.$$

As above, rewrite in terms of a:

$$\frac{da}{dz} \;=\; a(1-a).$$

Interpretation and sanity checks.

- *Range.* Since $0 < a < 1$, we have $0 < a(1-a) \le \frac{1}{4}$, with the maximum at $a = \frac{1}{2}$. (The sigmoid is most sensitive near its midpoint.)
- *Saturation.* If a is near 0 or 1, then $a(1-a)$ is small — gradients shrink (saturation).
- *Sign.* The derivative is always nonnegative; increasing z never decreases a.

Why this form is practical. The compact identity $\dfrac{\partial a}{\partial z} = a(1-a)$ will cancel neatly with the denominator from $\partial\ell/\partial a = \frac{a-y}{a(1-a)}$ in the next step, yielding the classic and simple error term $(a-y)$ in the gradient.

4.3 Third link — how the score reacts to parameters ($\partial z/\partial w_k,\ \partial z/\partial b$) & final composition

Setup. For one example with input $\mathbf{x} = (x_1, \ldots, x_d)^\top$, the neuron's linear score is

$$z \ = \ \mathbf{w}^\top \mathbf{x} + b \ = \ \sum_{k=1}^{d} w_k x_k \ + \ b.$$

We differentiate z with respect to a single weight w_k, and with respect to the bias b.

Rules we will use.

R1. Linearity (sum rule): $\dfrac{d}{du}\left(\sum_j c_j\, f_j(u)\right) = \sum_j c_j\, f_j'(u)$.

R2. Constant factor: $\dfrac{d}{du}(c \cdot f(u)) = c \cdot f'(u)$ (constant c).

R3. Constant derivative: $\dfrac{d}{du}(c) = 0$.

R4. Monomial: $\dfrac{d}{du}(u) = 1$.

Derivative of z w.r.t. a weight w_k. Keep all x_j and all other w_j (for $j \neq k$) fixed. By **R1**–**R2**,

$$\frac{\partial z}{\partial w_k} \ = \ \frac{\partial}{\partial w_k}\left(\sum_{j=1}^{d} w_j x_j + b\right) \ = \ \sum_{j=1}^{d} x_j \frac{\partial w_j}{\partial w_k} \ + \ \frac{\partial b}{\partial w_k}.$$

Here $\frac{\partial w_j}{\partial w_k} = 0$ if $j \neq k$ and $= 1$ if $j = k$ (Kronecker delta), and b is a constant w.r.t. w_k (**R3**), hence

$$\boxed{\frac{\partial z}{\partial w_k} \ = \ x_k}$$

Derivative of z w.r.t. the bias b. All w_j and x_j are constants w.r.t. b:

$$\frac{\partial z}{\partial b} \ = \ \frac{\partial}{\partial b}\left(\sum_{j=1}^{d} w_j x_j + b\right) \ = \ 0 \ + \ \frac{\partial b}{\partial b} \ = \ 1 \quad \text{by } \mathbf{R4}.$$

$$\boxed{\frac{\partial z}{\partial b} \ = \ 1}$$

4.4 Compose the three links (single example)

Recall the three pieces already derived:

$$\frac{\partial \ell}{\partial a} = -\left(\frac{y}{a} - \frac{1-y}{1-a}\right) \ = \ \frac{a-y}{a(1-a)}, \qquad \frac{\partial a}{\partial z} = \ a(1-a), \qquad \frac{\partial z}{\partial w_k} = x_k,\ \frac{\partial z}{\partial b} = 1.$$

Gradient w.r.t. a weight w_k. By the chain rule,

$$\frac{\partial \ell}{\partial w_k} = \frac{\partial \ell}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_k}.$$

Substitute each piece:

$$\frac{\partial \ell}{\partial w_k} = \underbrace{\frac{a-y}{a(1-a)}}_{\partial \ell/\partial a} \cdot \underbrace{a(1-a)}_{\partial a/\partial z} \cdot \underbrace{x_k}_{\partial z/\partial w_k} = (a-y)\,x_k.$$

$$\boxed{\frac{\partial \ell}{\partial w_k} = (a-y)\,x_k}$$

Gradient w.r.t. the bias b. Similarly,

$$\frac{\partial \ell}{\partial b} = \frac{\partial \ell}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b} = \frac{a-y}{a(1-a)} \cdot a(1-a) \cdot 1 = a-y.$$

$$\boxed{\frac{\partial \ell}{\partial b} = a-y}$$

Vector shorthand (same result, optional). Let $\mathbf{w} \in \mathbb{R}^d$ and $\mathbf{x} \in \mathbb{R}^d$. Collecting all k:

$$\boxed{\frac{\partial \ell}{\partial \mathbf{w}} = (a-y)\,\mathbf{x}, \qquad \frac{\partial \ell}{\partial b} = a-y.}$$

Each component is exactly $(a-y)x_k$.

4.5 Batch version (mean over m examples)

Let $\mathcal{J} = \frac{1}{m}\sum_{i=1}^{m} \ell_i$ with

$$\boxed{\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)\,\mathbf{x}_i, \qquad \frac{\partial \mathcal{J}}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i).}$$

Update (gradient descent). With learning rate $\eta > 0$,

$$\mathbf{w} \leftarrow \mathbf{w} - \eta\,\frac{\partial \mathcal{J}}{\partial \mathbf{w}}, \qquad b \leftarrow b - \eta\,\frac{\partial \mathcal{J}}{\partial b}.$$

Sanity checks.

- If $a_i > y_i$ (overconfident toward class 1), then $(a_i - y_i) > 0$ pushes $\mathbf{w}$ and b *downward* (negative step), reducing future z and a.
- If $a_i < y_i$ (underconfident), $(a_i - y_i) < 0$ pushes parameters upward, increasing z and a.

The learning signal, distilled.

$$\frac{\partial \ell}{\partial w_k} = \Big(\frac{\partial \ell}{\partial a}\Big)\Big(\frac{\partial a}{\partial z}\Big)\Big(\frac{\partial z}{\partial w_k}\Big), \qquad \text{with} \quad \frac{\partial \ell}{\partial a} = \frac{a-y}{a(1-a)}, \quad \frac{\partial a}{\partial z} = a(1-a), \quad \frac{\partial z}{\partial w_k} = x_k.$$

$$\Rightarrow \quad \boxed{\frac{\partial \ell}{\partial w_k} = (a-y)\,x_k, \qquad \frac{\partial \ell}{\partial b} = a-y}$$

Three simple pieces. Multiplied, they tell the neuron exactly how to learn.

Afterword — From understanding to motion, from motion to code

Where we stand. Episode I made the neuron legible. Episode II made it *move*. We traced how a tiny nudge to a parameter travels through the model and returns as a clean update:

$$\frac{\partial \ell}{\partial w_k} = (a-y)\,x_k, \qquad \frac{\partial \ell}{\partial b} = (a-y),$$

and, over a batch,

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i)\,\mathbf{x}_i, \qquad \frac{\partial \mathcal{J}}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a_i - y_i).$$

With a learning rate η, gradient descent turns understanding into action.

What we'll build next. Episode III is where the math becomes a living program: a from-scratch neuron that trains on a real binary task (toxic vs non-toxic plants), with every piece implemented by hand: forward pass, loss, gradients, and update. No black boxes — just code that mirrors the equations you now own.