DEEP LEARNING FROM SCRATCH

A Guide for Self-Learners

Episode I

Theory of a Neuron

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

Why this little guide?

We talk about AI, ChatGPT, deep learning — yet even many who deploy these systems no longer remember what their core truly is: the neuron.

How this little guide?

This first guide goes back there — no shortcuts, no myths. Just the essence of how learning begins, rebuilt from scratch in ten pages.

"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.

What is a Neural Network?

A neural network is... a network of neurons. *That's not a joke.*

It's a system built on countless small decision-makers — called **neurons** — each making a micro-choice based on the information it receives. When all these small decisions are combined, they produce something astonishingly complex: recognition, prediction, understanding.

If you grasp how a single neuron works, you'll grasp how the whole network works, because they work exactly the same way. That's where we start.

To understand deep learning, you don't need to start big — you need to start small, but precisely. Then it works the very same way.

Most people jump straight into massive architectures, thousands of layers, billions of parameters. But behind every transformer, every diffusion model, every chat-based intelligence — there's still this: one equation, one neuron, one simple logic.

So let's strip everything down. Let's rebuild the neuron from scratch — with nothing but algebra, geometry, and curiosity.

1. Processing the Input Data — The Linear Function

Imagine you're trying to classify two types of plants:

- Toxic plants ($y = 1$, class 1)
- Non-toxic plants ($y = 0$, class 0)

We measure two features:

$$x_1 = \text{leaf length}, \qquad x_2 = \text{leaf width}.$$

Each plant becomes a data point $\mathbf{x} = (x_1, x_2)$.

Here's a random sample of our data — each point is a plant, colored by its true nature:

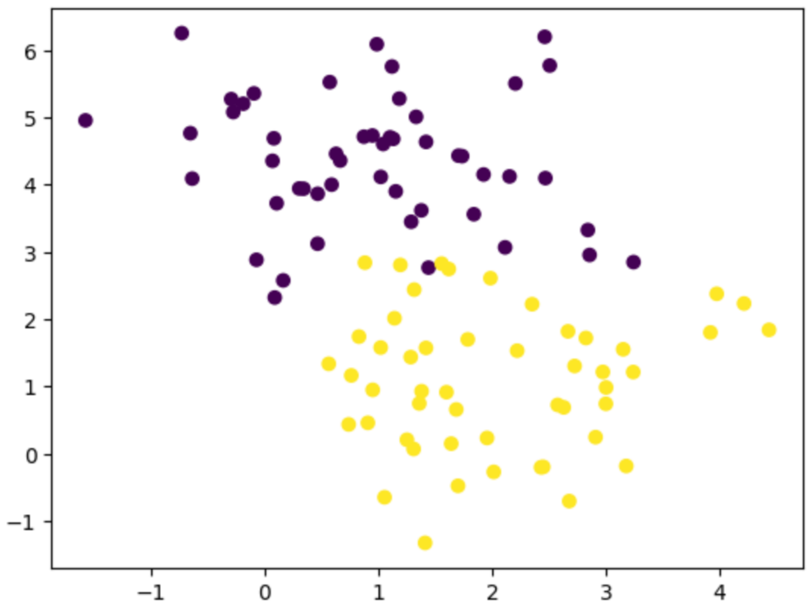

Figure 1: Random dataset — leaf width vs. length.

Purple = toxic (class 1), Yellow = non-toxic (class 0).

Our goal is to find the line — the **decision boundary** — that best separates the two groups. And to do that, we'll use the most fundamental tool in deep learning: the **linear function**.

From Input to Score

Each feature is multiplied by a weight, and we add a bias term to shift the line up or down:

$$z = w_1 x_1 + w_2 x_2 + b.$$

In vector form:

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

This simple formula defines a plane (or a line, if there are only two variables). It's the mathematical heart of the neuron.

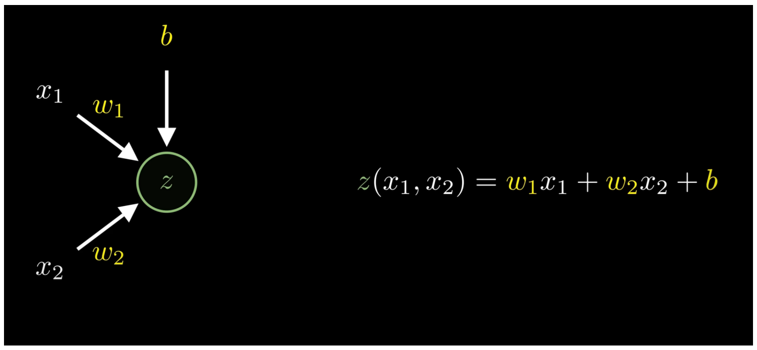

Figure 2: A single neuron — input features multiplied by weights, then summed with a bias.

So far, a neuron is nothing but a linear model — a weighted sum plus a bias.

You can picture w_1 and w_2 as the dials controlling the slope of your decision boundary, and b as the lever that shifts it up or down.

By adjusting these parameters, the neuron learns where to draw the line between toxic and non-toxic plants.

But Here's the Catch

A purely linear model is powerful — but limited. It can separate straight lines, but not curves. It can't "bend" to capture more subtle relationships.

That's why deep learning exists: to combine these simple linear units into something that can twist, turn, and adapt. But even before that, we can give a single neuron a spark of non-linearity — a way to *feel* the difference between "probably" and "definitely."

This spark is called the activation function.
It's what turns a line into a decision, and math into meaning.

And that's exactly what we'll explore next.

So far, our neuron can calculate. Next, we'll teach it to believe.

2. Adding a Sense of Probability: Activation Functions

Our neuron can now compute a score $z = \mathbf{w}^\top \mathbf{x} + b$. It's a clean, precise, linear output — but there's a problem.

It's also... lifeless. It's just numbers. It doesn't "decide" yet.

A linear neuron can draw a line, but it can't yet say "I'm sure."
It sees the world in raw geometry, not in belief.

A true neuron doesn't just compute — it *responds*. It transforms its input into a signal that means something. In other words, it turns a linear score into a sense of confidence.

This transformation is what we call the **activation function**.
It's the moment the neuron wakes up.

From Line to Life

The most classic activation function — and still one of the most elegant — is the **sigmoid**:

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

It takes any real number and bends it softly into a range between 0 and 1. Small z values become close to 0, large z values approach 1, and in between, the curve transitions smoothly — like a decision being made.

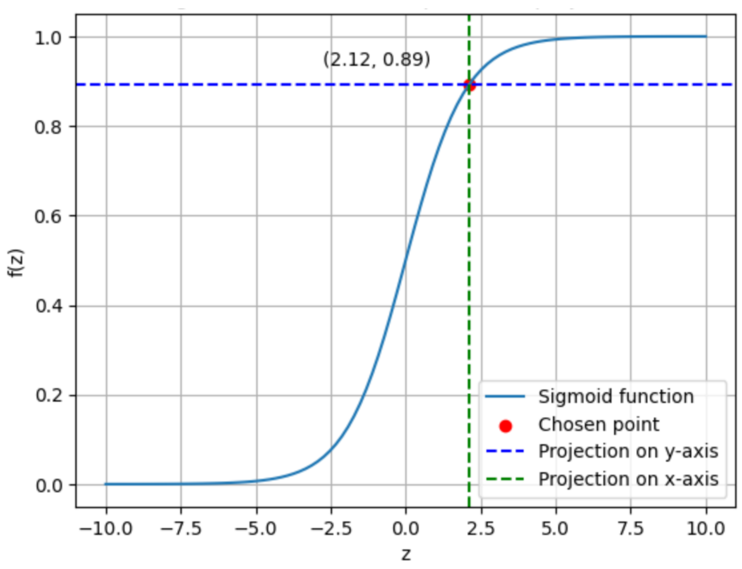

Figure 3: The sigmoid function: turning raw scores into confidence.

The sigmoid gives the neuron its first heartbeat —
a continuous scale between doubt and certainty.

From Score to Probability

Once we apply the sigmoid, something beautiful happens: the neuron's output can now be read as a **probability**.

$$a(z) = P(Y = 1 \mid \mathbf{x})$$

If $a(z)$ is close to 1, the neuron is confident the example belongs to class 1. If it's close to 0, it's confident it belongs to class 0. And when $a(z) = 0.5$, it hesitates — perfectly balanced between both.

The output is no longer a line — it's a feeling.
A soft confidence curve that breathes between 0 and 1.

This is what allows deep learning models to reason in terms of likelihoods, not absolutes. They no longer say "yes or no" — they say "probably," "almost," "barely," or "certainly." That's intelligence in its simplest form.

Binary Decisions and Smooth Doubt

The sigmoid's shape perfectly captures how humans decide under uncertainty. Think of it as a mental switch that takes time to flip — slow at first, fast in the middle, steady near the ends.

$$\text{If } a(z) > 0.5 \to \text{predict Class 1 (toxic)} \qquad \text{If } a(z) < 0.5 \to \text{predict Class 0 (non-toxic)}$$

The result is a probabilistic model: a system that not only classifies but also expresses *how sure* it is.

Other activation functions exist — tanh, ReLU, leaky ReLU — each one with its own rhythm, its own way of shaping information. But the principle remains universal: to make learning possible, you must let your model *bend*.

A Bernoulli Mindset

If the neuron outputs a probability $a(z)$ for class 1, then the probability for class 0 is $1 - a(z)$. The entire output now follows a **Bernoulli distribution**:

$$P(Y = y) = a(z)^y \cdot (1 - a(z))^{1-y}, \quad y \in \{0, 1\}$$

Each prediction is a coin flip weighted by confidence —
and the sigmoid decides how much weight each side deserves.

Now, our neuron can finally *believe*. It can look at a plant and say: "I'm 89% sure it's toxic." That's no longer just math — that's judgment.

But belief alone isn't enough. Just like us, a model needs to know when it's wrong — and how to get better next time.

3. Performance Measurement

Now comes a quiet but essential question: *How do we know if our neuron is doing a good job?*

Until now, our model has been like a child learning to throw darts in the dark. It throws, it hopes, but it doesn't yet know where the dart lands — or how far from the center it was.

Learning starts when we give feedback — when the model feels how wrong it was.

That's exactly what the **loss function** does: it transforms every decision into a number — a measure of how "off" the model was from reality.

Let's make this concrete.

From Likelihood to Logic

If our model says "89% chance the plant is toxic" and the plant *is* toxic, then it wasn't wrong — it was *89% confident right.* We can translate that confidence into a notion of **likelihood** — the plausibility of being correct.

The higher the model's confidence in the truth, the higher its likelihood.

Across the entire dataset, we multiply all these plausibilities together — one for each plant we try to classify:

$$L(\mathbf{w}, b) = \prod_{i=1}^{N} P(Y_i = y_i)$$

Each term rewards the model for being accurate on that example. It's like a global applause meter: every correct guess boosts the total, every wrong guess dims it a little.

But there's a problem — a very mathematical one.

The Fragility of Tiny Numbers

Multiplying dozens, hundreds, or millions of probabilities between 0 and 1 quickly gives numbers so small that even your computer can't represent them properly.

The applause turns into a whisper. The machine hears only silence.

Small probabilities multiplied together vanish into numerical noise.

So we apply one of math's simplest yet most powerful tricks: we take the logarithm.

Why Logarithms Save the Day

Logarithms turn products into sums, and whispers into clear signals:

$$\log L(\mathbf{w}, b) = \sum_{i=1}^{N} \log P(Y_i = y_i)$$

Now each prediction contributes its own voice to the choir. We call this the **log-likelihood**:

$$\mathcal{L}(\mathbf{w}, b) = \sum_{i=1}^{N} \left[y_i \log a(z_i) + (1 - y_i) \log(1 - a(z_i)) \right]$$

Maximizing this quantity means pushing the model toward predictions that feel more and more plausible.

Logarithms don't change what we're measuring — they just make the story computable.

From Likelihood to Loss

In deep learning, we don't usually talk about maximizing. We prefer to *minimize error* — it's more intuitive to aim for less pain than for more pleasure.

So we take the negative of the log-likelihood, average it over all examples, and call it the **log-loss** (or cross-entropy loss):

$$\mathcal{J}(\mathbf{w}, b) = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log a(z_i) + (1 - y_i)\log(1 - a(z_i))\Big]$$

Each term in this sum says: "If I was confident but wrong, punish me hard. If I was uncertain but right, forgive me softly."

That's why this function is so elegant — it encodes humility. It doesn't just reward accuracy; it rewards *calibrated confidence.*

The log-loss is like a mirror.
It doesn't scold the neuron — it reflects how sure and how right it was.

The Birth of Learning

At this stage, something subtle but fundamental has happened: our neuron has developed *a sense of self-awareness.*

It can look at its output and know if it did well or not. It can measure its own performance. And that, truly, is the beginning of learning.

But knowing how wrong you are is not yet the same as knowing how to fix it.

That's the next challenge. If the log-loss tells us *how bad* we are, we still need a way to know *which way to move* to get better.

4. Gradient Descent

So far, our neuron knows how to see — it can take inputs, process them, and tell us how confident it feels about its answer. But it doesn't yet know how to *improve.* It observes, it guesses, it even measures how wrong it is... and then it just stands there.

Learning means motion — it's the moment where math becomes alive.

Gradient descent is how a neuron learns to move — to walk down its own mistakes.

We call it a **descent** because the loss function can be imagined like a mountain landscape. Every point in that landscape represents one possible state of your neuron — its weights and bias. High ground means high error; valleys mean good predictions.

The goal? **Find the valley. Walk downhill. One small step at a time.**

That's what gradient descent does. It looks around, senses the direction of steepest decrease, and takes a careful step that way.

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

Here:

- $\mathcal{J}$ is the loss — the altitude of our model.
- $\frac{\partial \mathcal{J}}{\partial \mathbf{w}}$ and $\frac{\partial \mathcal{J}}{\partial b}$ tell us where the slope is steepest.
- α is the **learning rate**, the size of each step — too big and you trip, too small and you never arrive.

Why "descent"? Because we move downward, seeking less error.
Why "gradients"? Because they point the way.

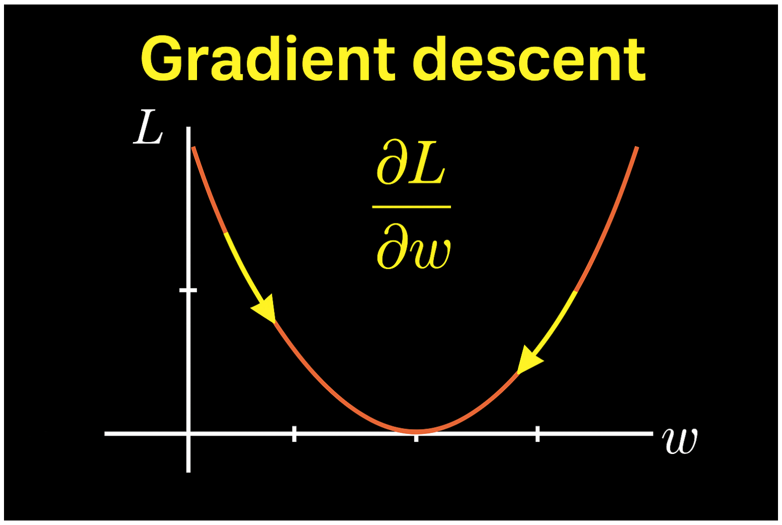

Figure 4: The loss landscape — learning is just walking downhill, one step at a time.

That's it. The whole movement of learning — packed in a single idea: *feel the slope, and go the right way.*

The neuron is no longer just an equation — it's a traveler.
It walks, stumbles, and finds its balance through feedback.

Closing — The Edge of Understanding

And that's where we pause.

You now have the entire structure of a learning system — the architecture, the logic, and even the reason it moves. But there's one mystery left: *how does it know which way to go?*

What are these gradients we keep talking about? How are they found? How can one number tell us the direction of learning?

That's what the next guide is about — the silent mechanism behind motion itself.

Episode II — The Secret of the Gradient

- Deriving $\partial \mathcal{J}/\partial \mathbf{w}$ and $\partial \mathcal{J}/\partial b$ from first principles.
- Understanding how each piece (linear $\rightarrow$ sigmoid $\rightarrow$ loss) links through the chain rule.
- Watching the neuron truly learn for the first time — line by line.

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

See you tomorrow, in Episode 2 with the true computation of the gradients, without shortcuts.