

King of Dumbs

Level: ♣ Newbie

Logistic regression or neural network? Same neuron

A logistic regression is a neural network with a single neuron. What deep learning adds, when it pays off and when the classic model is enough, at three levels.

English Published October 4, 2026



Scan to open the interactive lesson

<https://kingofdumbs.com/en/lessons/logistic-regression-or-neural-network>

What you'll learn

- Logistic regression
- Neural networks
- Sigmoid function
- Vanishing gradient
- Choosing between classic machine learning and deep learning

Symbols you'll meet

E_a subscript: the label underneath (♠) · Σ sum: add them all up (♠) · x_i index i: item number i (♠) · Π product: multiply them all (♠) · $\approx$ approximately equal (♠) · $\sim$ on the order of / distributed as (♠) · $\leq \geq$ less or equal / greater or equal (♠) · $|x|$ absolute value: drop the sign (♠) · 10^{-3} negative exponent: divide (♠) · $h^{(1)}$ superscript in brackets: not a power (♠) · e the number e (≈ 2.718) (♠) · $\log$ logarithm: how many times do I multiply? (♠♠) · σ sigma: depends on the lesson (♠) · λ lambda: a stretching number (♠) · δ small delta: a small error (♠) · $\|x\|$ norm: the length of the arrow (♠) · x^T transpose: lay the column down (♠) · $\in$ belongs to (♠) · $\mathbb{R} \mathbb{Z}$ sets of numbers (♠) · dE/dt derivative: the rate of change (♠) · ∂ partial derivative: change one thing only (♠) · ∇ gradient: where it climbs fastest (♠)

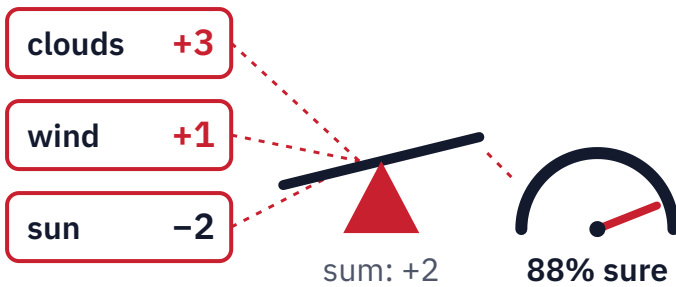
The first time I saw a neural network drawn out, all little circles and arrows, I assumed it was something different from good old statistics. Then I had to code a logistic regression by hand and ended up asking this lesson's dumb question: **is a logistic regression a neural network?**

The short answer is yes. A logistic regression is a neural network with **a single neuron**. They aren't lookalikes or cousins: they're the same model with two names. If it's the same neuron, the interesting question changes: what do big networks add, and when are they worth paying for?

Pick your card: ♣ if you're starting from zero, ♦ if you know what a function or an average is, ♠ if you want the notation. You can switch at any time.

One neuron, two names

Rain tomorrow?



Sure and wrong? Scolding.

Imagine you want to guess whether it'll rain tomorrow. You look at clues: there are clouds, it's windy, the sun comes out in the afternoon. They don't all count the same: clouds weigh a lot, wind a little and sun takes away. You add up what each clue weighs, and that sum moves a needle that goes from "definitely not" to "definitely yes".

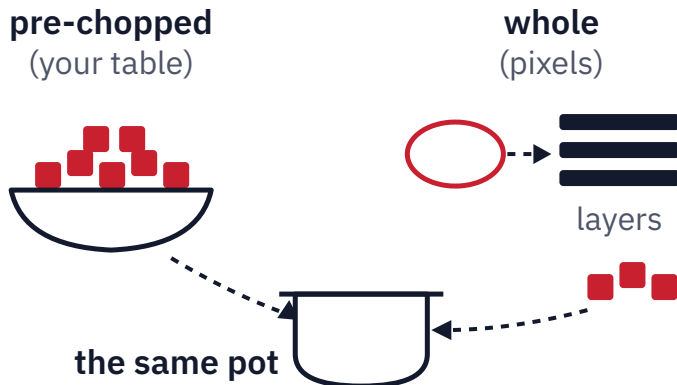
That scale is a **logistic regression**, a good old statistics method for answering yes-or-no questions. And it's also a **neuron**, the building block of neural networks. Same scale, two names.

The most important part is still missing: learning. At first, the scale doesn't know how much each clue weighs. You show it lots of past days and, every time it's wrong, it gets told off, harder the surer it was. To avoid telling-offs, the scale adjusts its weights bit by bit. And this scale has a reassuring property: wherever you start, you always end up with the same weights.

The King of Dumbs says:

I spent a week thinking I had to choose between statistics and deep learning. Turns out I'd been doing one-neuron deep learning for years without knowing it.

Who builds the features

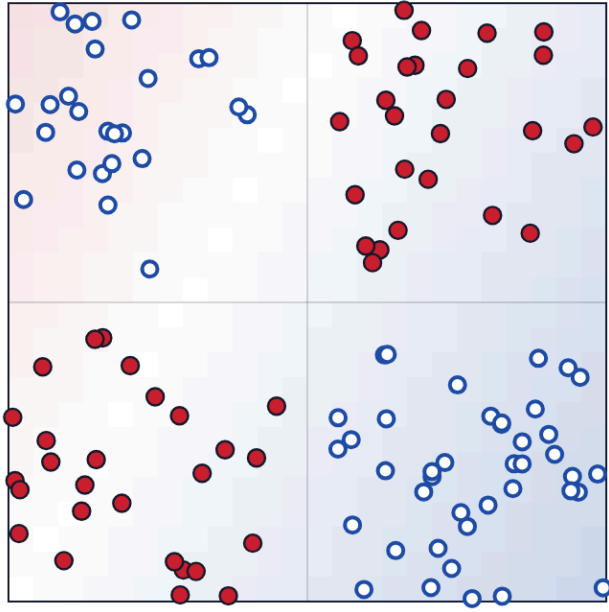


If the scale at the end is the same, what's so special about a big neural network? Whatever happens **before** the scale.

Think of a pot. If you cook with pre-chopped ingredients, you throw them in and you're done. That's what data in a table is like: age, amount, number of visits. Each column is already a meaningful clue, and the scale weighs them directly.

A photo is a whole potato. A single dot of colour doesn't tell you whether there's a cat. Someone has to chop the photo into useful clues, like "there are ears here" or "this looks like a whisker". The **layers** of a neural network do exactly that: they learn to chop by themselves. People in the know call those manufactured clues *features*. In the end, everything lands in the same pot: the same scale as before.

Try it. XOR and the ring are the two classic examples of points a straight line can't split. Pick some data and a model, and train it:



The points

Line
XOR
Ring

The model

1 neuron (logistic regression) 3 parameters

1 neuron + hand-made feature: $x_1 \cdot x_2$ 4 parameters

Hidden layer of 8 + 1 neuron 33 parameters

Train
Reset

Weights: $w = (-0.44; 0.24) \cdot b = -0.18$

Accuracy: 43% (always guessing the most common class: 55%). One neuron can only draw a straight line, and no straight line splits an XOR. It does about as well as a coin toss.

● class 1 ○ class 0



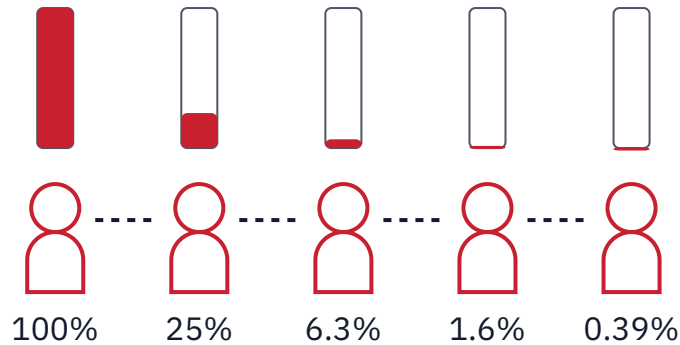
Interactive widget. Try it on the web: <https://kingofdumbs.com/en/lessons/logistic-regression-or-neural-network>

What I like most about this experiment is the middle row. The same neuron that fails on XOR gets it right as soon as you hand it the product of the two coordinates, built by hand. The hidden layer gets to the same place without anyone telling it what to look for, at the cost of 33 numbers to tune instead of 4 and of no longer being able to read what it learned.



Why the sigmoid left the hidden layers

Each one repeats it at a quarter of the volume



If a network is “lots of scales in a row”, the obvious move would be to use the same needle, the S-shaped curve, in all of them. That’s how it was done for years, and it went badly.

To learn, a network gets a message at the end, “you were off by this much”, and that message has to travel backwards, from the last layer to the first, so each one can adjust its weights. It’s like the telephone game: each layer with that curve repeats the message at **a quarter of the volume** at most. After five people, the message arrives at less than half of one percent. With ten, not even a whisper gets through, and the first layers barely learn.

The fix was to swap the needle in the middle layers for one that doesn’t turn the volume down when the message is positive. The S-shaped curve didn’t retire: it went to work as a **doorman**. It’s still at the end of the network, to say “88% yes”, and inside large language models, deciding how much information gets through.

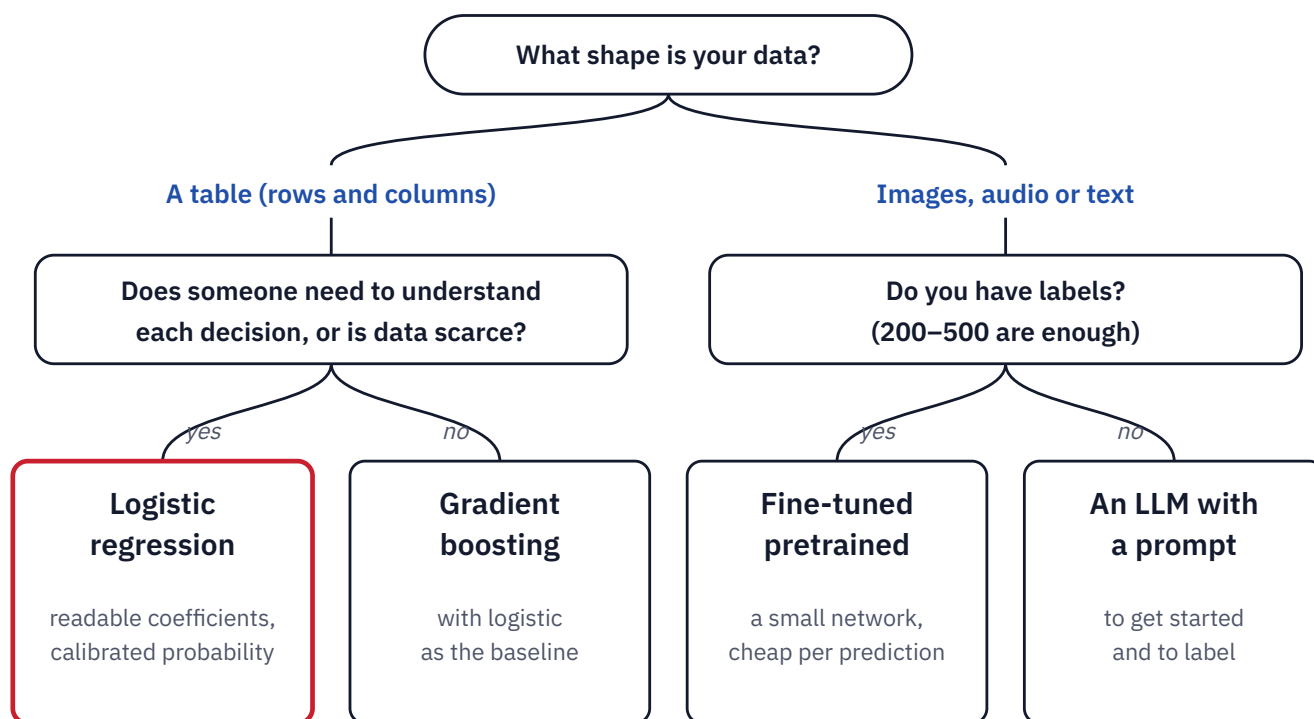
The King of Dumbs says:

The sigmoid is like me: kicked out of the hidden layers, it ended up as the doorman of every language model.

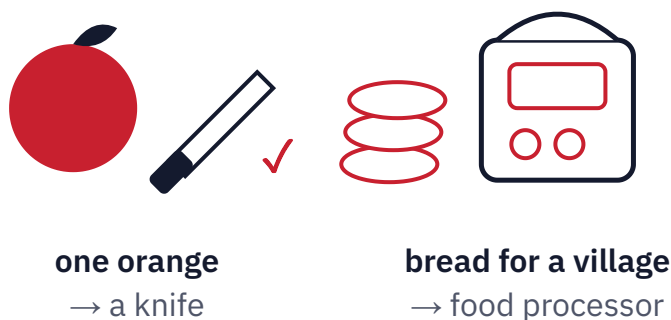


When to use what

The decision is hardly ever about “how complex the problem is”. It’s about what shape your data has, how many examples with the right answer you have and who has to understand each decision. This tree works for all three levels:



For predicting a label. Stepping down in complexity is cheap; stepping up is expensive: start on the left.



You don't buy a food processor to peel an orange. The machine is expensive, takes a while to set up and, if something goes wrong, you don't know why. It pays off when you have to make bread for a whole village.

Models are the same. If your data is a table with columns that already mean something, the good old scale is usually enough, and you can explain every decision too: "I said no because of this column". If your data is photos, sound or text, someone has to chop the ingredients, and that's where a network pays off. One piece of advice that always works: try the scale first. It takes minutes and tells you how much the machine would have to win by to be worth it.



Final challenge

1. A logistic regression and a neuron with a sigmoid, trained with the same penalty: what are they?

- a. Cousins: they look alike but give different results
- b. The same model with two names
- c. The neuron always does better, because it's deep learning

2. Why did the sigmoid stop being used in hidden layers?

- a. Because it's slow to compute
- b. Because the error signal shrinks at every layer and the first ones barely learn
- c. Because numbers between 0 and 1 are useless

3. You have a 5,000-row table (age, amount, visits) and someone has to understand each decision. Where do you start?

- a. A language model with a prompt
- b. A ten-layer deep network
- c. A logistic regression



In three sentences

1. A logistic regression is a neural network with a single neuron: the same weighted sum, the same S-shaped curve (the sigmoid) and the same penalty.
2. What deep learning adds is the layers before it, which build the features; the sigmoid left those layers because it drowns out the error signal, and today it works at the output and in the gates.
3. If your data is a table with columns that already mean something, start with logistic regression, which takes minutes and tells you how much the complex option has to win by;

if it's pixels, audio or text, someone has to build the features.



Sources

- Logistic regression as a generalized linear model and the sigmoid as its canonical link: [Stanford CS229 notes](#). Convexity of the log-loss: Boyd and Vandenberghe, [Convex Optimization](#), §7.1.1.
- LogisticRegression defaults (L2, C=1.0): [scikit-learn documentation](#).
- Vanishing gradient and the sigmoid's maximum derivative: Hochreiter, 1991 thesis ([PDF](#), in German) and 1998 paper ([PDF](#)).
- Test error with sigmoid, tanh and tanh with normalized initialization: Glorot and Bengio, *AISTATS* 2010, [Table 1](#).
- SwiGLU: Shazeer, [arXiv:2002.05202](#); its use in [LLaMA](#) and [PaLM](#). LSTM gates: [Hochreiter and Schmidhuber, 1997](#).
- Machine learning versus logistic regression in clinical prediction: Christodoulou et al., *J Clin Epidemiol* 110, 12 (2019), [doi:10.1016/j.jclinepi.2019.02.004](#). Abstract read.
- Trees versus neural networks on tabular data: Grinsztajn et al., [arXiv:2207.08815](#).
- Fine-tuned models versus zero-shot LLMs: Bucher and Martini, [arXiv:2406.08660](#), Tables 1 to 4 and §5.5.
- Calibration of modern networks: Guo et al., *ICML* 2017, [arXiv:1706.04599](#), Table 1.
- BCEWithLogitsLoss: [PyTorch documentation](#).

How this lesson was made: from my machine learning notes, with help from Claude to organise them and write the three levels, translated from Spanish by Claude. The numbers were checked against each paper's original tables on 4 October 2026. The neuron lab trains in your browser with gradient descent, just like the code in the ♠ level. The drawings are my own.

Answers

1. **b)** The same model with two names. That's it. Same three pieces, same weights at the end.
2. **b)** Because the error signal shrinks at every layer and the first ones barely learn. Exactly: 0.25 per layer at most. After ten layers, not even a whisper gets through.
3. **c)** A logistic regression. Good. The columns already mean something and the decision has to be explainable: logistic regression is the starting point.

A King of Dumbs lesson, kingofdumbs.com. Offline version; the original, with widgets you can play with, is at: <https://kingofdumbs.com/en/lessons/logistic-regression-or-neural-network>

Updated on October 4, 2026.