

King of Dumbs

Level: ♦ Intermediate

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

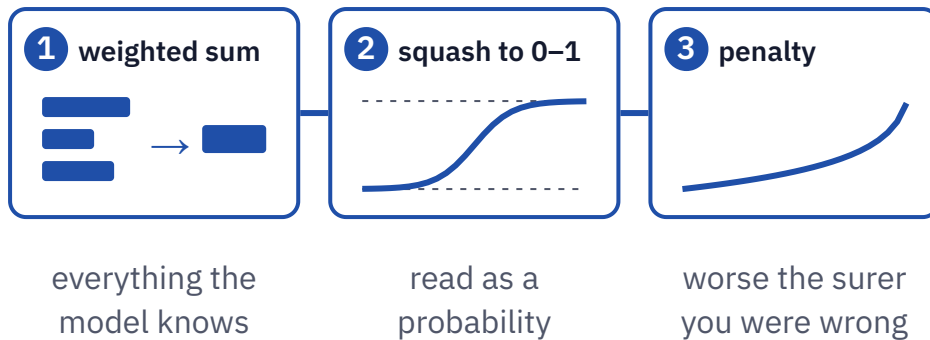
E_a subscript: the label underneath (♠) · Σ sum: add them all up (♠) · x_i index i: item number i (♠) · Π product: multiply them all (♠) · $\approx$ approximately equal (♠) · $\sim$ on the order of / distributed as (♠) · $\leq \geq$ less or equal / greater or equal (♠) · $|x|$ absolute value: drop the sign (♠) · 10^{-3} negative exponent: divide (♠) · $h^{(l)}$ 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



A logistic regression has three pieces, and they're worth knowing because each one explains a different way it fails.

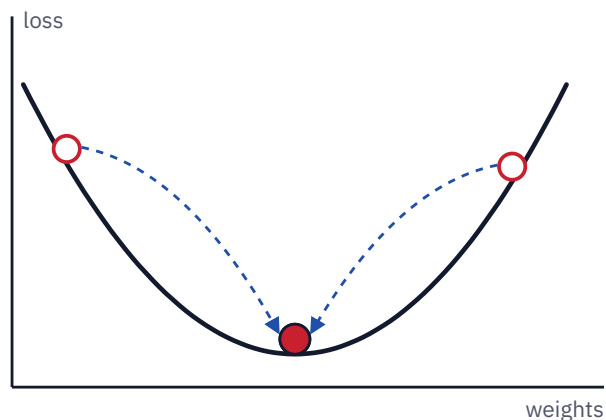
1. A weighted sum. Each column of your data is multiplied by a number, its weight, and everything is added up together with a constant. We'll call the result the *score*. Everything the model knows lives there: if the signal is in a combination of two columns or in a curve, the model can't see it unless you hand it over already built.

2. The sigmoid. It's an S-shaped curve that squashes any score between 0 and 1: a very negative one gives almost 0, a very positive one almost 1, and zero gives exactly 0.5. It isn't picked for looks: it comes from assuming the answer is a loaded coin whose bias depends on a weighted sum. That's why its output can be read as a probability.

3. The penalty. It compares the probability with the right answer and punishes being very sure and wrong. In the jargon it's called the *loss*; this one in particular is the *log-loss* or cross-entropy. To learn, the model checks which way the penalty goes down and moves the weights a little in that direction, over and over: that's *gradient descent*.

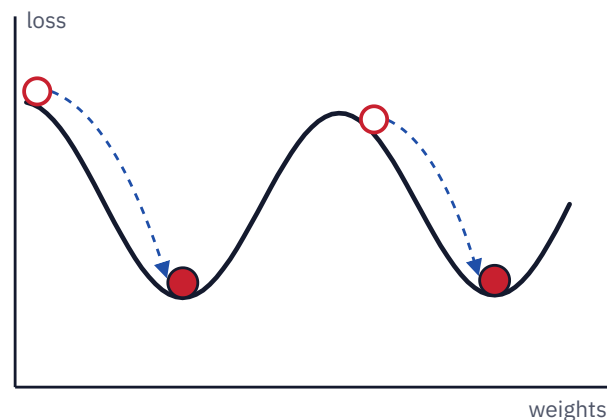
And here's the property that changes everything: for logistic regression, the penalty has **a single bottom**. Any way of going downhill ends up in the same place, so the random seed doesn't matter. A network with hidden layers loses that guarantee: it has many valleys, and where you end up depends on where you start.

Logistic: one bottom



same place, wherever you start

Deep network: several valleys



depends on where you start

Qualitative shapes, no data. What matters is how many bottoms there are.

I checked: scikit-learn's logistic regression and a neuron trained by hand give the same weights to the third decimal. A practical detail: many libraries add a brake by default that shrinks the weights, so if you want to read them, turn it off.

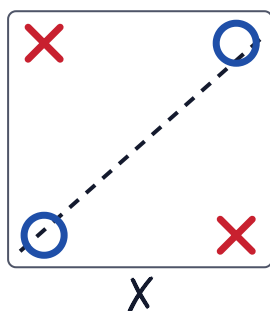
The King of Dumbs says:

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.

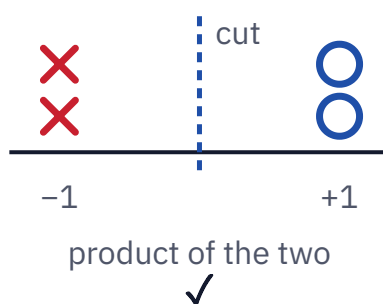


Who builds the features

no straight line
splits them



with the product
of the two, yes



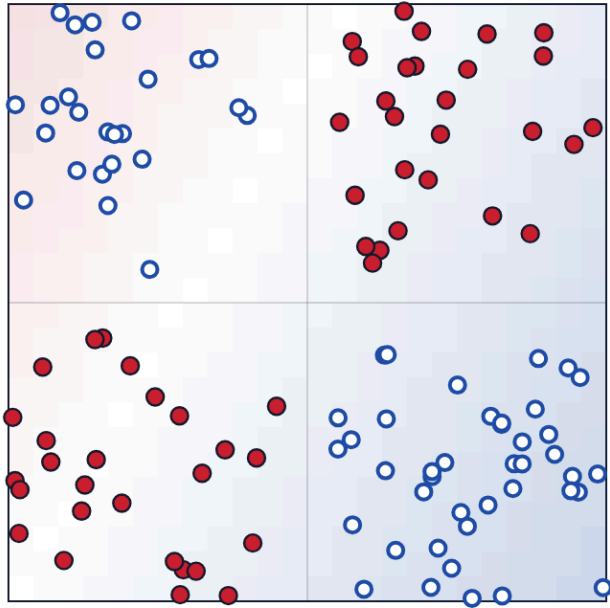
If the neuron at the end is the same, what a deep network adds is **whatever happens before that neuron**. A ten-layer network that answers yes or no is, in its last layer, a logistic regression. The difference is that its inputs aren't your columns: they're *features*, new combinations that the **hidden layers** learned from the data.

The classic example is XOR: four points at the corners of a square, with the class switching on each diagonal. A neuron can only draw a straight line, and no straight line splits them. But give it a new column, **the product of the two columns**, and the points split on their own. That new column is a feature: either you build it by hand, or a hidden layer builds it.

That gives the most useful rule of thumb I know. If your data is a table and the columns already mean something, the good features exist and the hidden layers have little to add. If it's pixels, audio or text, no weighted sum of pixels can see a cat: someone has to build the features. They can be built by hidden layers trained on your data, by hundreds of trees adding up their votes (*boosting*, on tables) or by a model already trained on half the world's data, of which you only adjust the end.

A practical detail: in deep learning libraries, the network returns the unsquashed score, and the sigmoid is applied inside the penalty, where the arithmetic doesn't break with huge numbers.

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:



● class 1 ● class 0

The points

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

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.



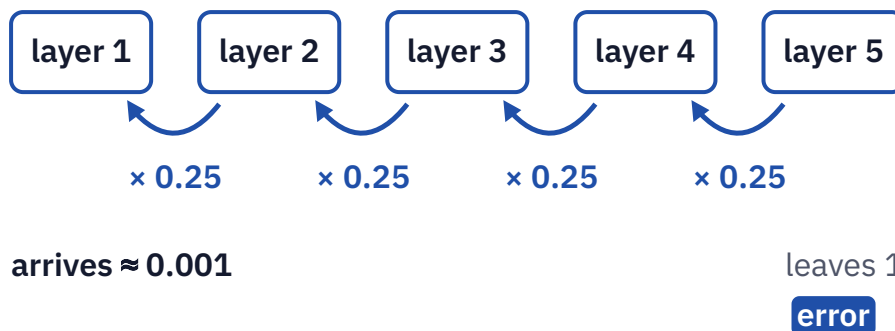
Interactive widget. Try it on the web: <https://kingofdumbs.com/en/lessons/logistic-regression-or-neural-network>

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.



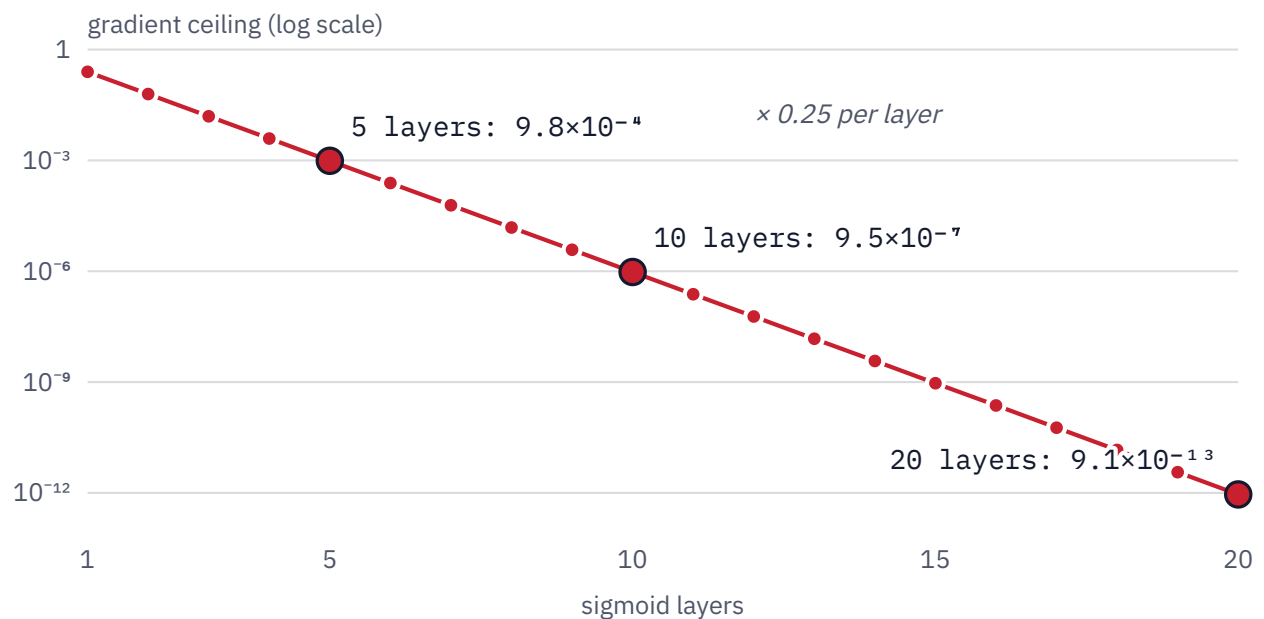
Why the sigmoid left the hidden layers

The error signal goes from the last layer to the first



If a network is “lots of logistic neurons stacked up”, the obvious move would be sigmoids in every layer. That’s how it was done for years, and it went badly.

The reason is the sigmoid’s **slope**. It’s 0.25 at most, right in the middle of the S, and almost zero at the ends, where the curve is already flat. To train a network, the signal saying how much and which way to move each weight (the *gradient*) is computed from the last layer to the first, multiplying by each layer’s slope: that’s *backpropagation*. With several sigmoids in a row, what reaches the first layer has a ceiling of 0.25 multiplied by itself once per layer, and that’s before counting how much the weights stretch or shrink it.



This is arithmetic, not a measurement: 0.25 multiplied by itself once per layer.

With 10 layers, the signal reaching the start is less than one millionth. The first layers barely learn. It's the *vanishing gradient* problem, which Hochreiter described in 1991. In practice it was just as bad: in 2010, Glorot and Bengio trained networks with 5 hidden layers and test error was **82.61% with sigmoids**, 27.15% with the hyperbolic tangent and 15.60% with the hyperbolic tangent plus a better initialization. In hidden layers, the sigmoid gave way to the **ReLU**, a function that lets positive values through as they are and sets negative ones to zero.

The sigmoid didn't disappear: it changed jobs. Outside the chain of hidden layers it isn't multiplied by itself dozens of times, and its 0-to-1 range is exactly what you need for a probability or a gate.

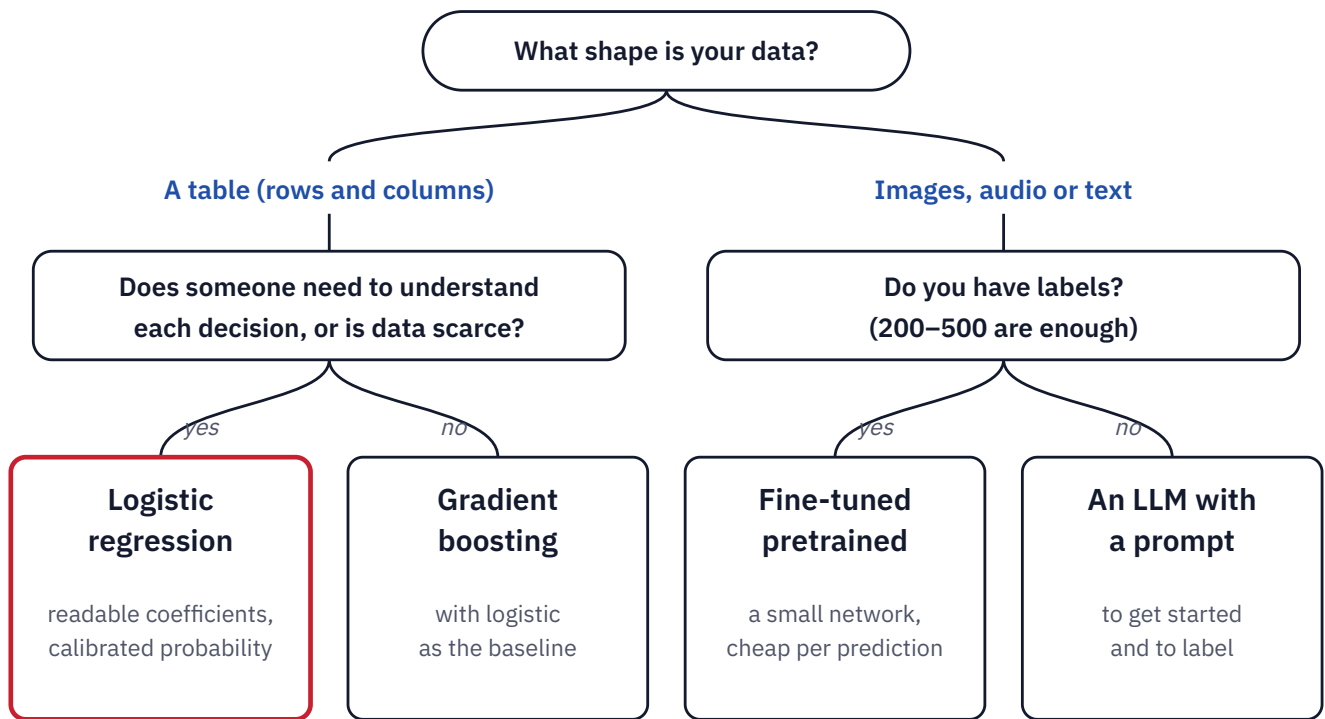
Where	What the sigmoid does
Output of a yes-or-no classifier	Turns the last score into a probability. It's the logistic regression at the end of the network
Classification with several labels at once	One independent sigmoid per class: a photo can be "beach" and "sunset"
Gates in LSTM and GRU networks	Decides what fraction of the memory gets through, from 0 to 1
SwiGLU, inside LLaMA and PaLM	One branch, through a sigmoid, decides how much of the other 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

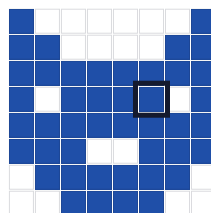
The decision is hardly ever about “how complex the problem is”. It’s about what shape your data has, how many examples with the right answer you have and who has to understand each decision. This tree works for all three levels:



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

age	amount	visits
34	120	3
51	15	9
27	300	1

each column
already means something

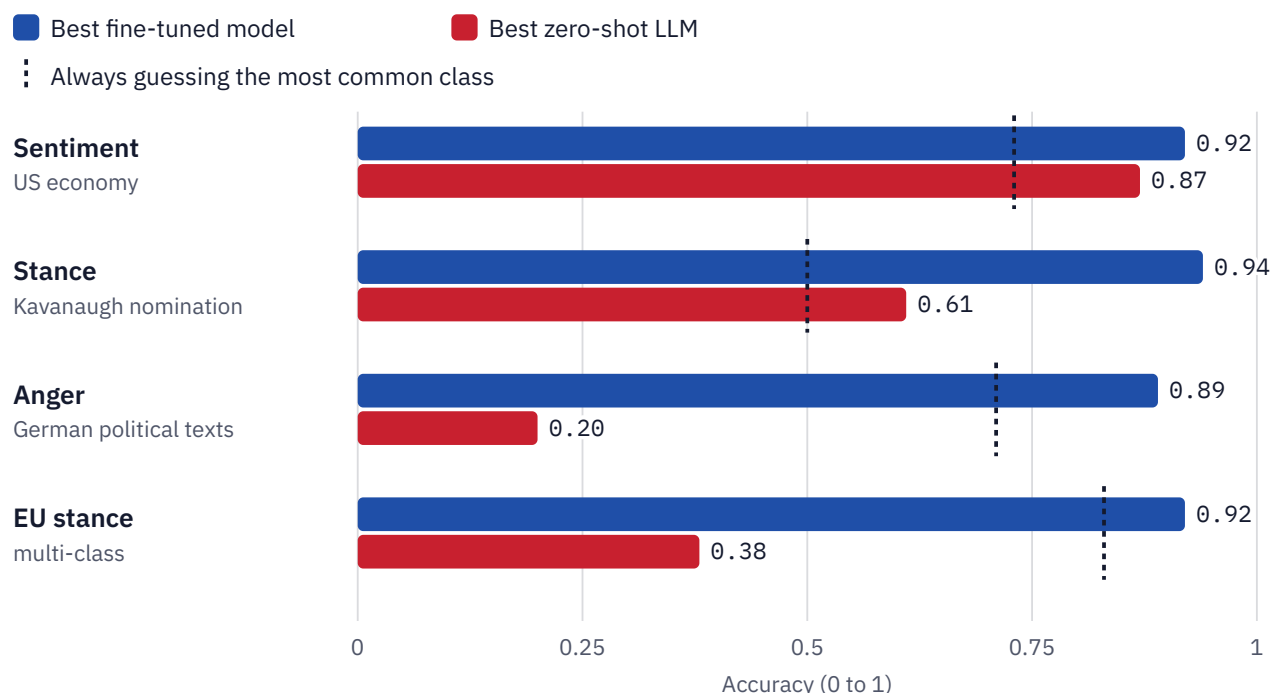


a single pixel
means nothing

Tables: the classic model is usually enough. Christodoulou and colleagues reviewed 71 clinical prediction studies with 282 comparisons between logistic regression and machine learning. They measured performance with the *AUC*, the probability that the model scores a positive case higher than a negative one. In the comparisons at low **risk of bias** (well-run studies), the difference was zero.

Machine learning only came out ahead in the studies at high risk of bias. I read the paper's abstract, not the full text. On large tables where nobody has to understand the decisions, trees adding up their votes usually beat neural networks, and logistic regression is still the baseline to start from.

Images, audio or text: the features have to be learned. But that doesn't always take a huge model. In 2024, Bucher and Martini compared small **fine-tuned** models (already trained and then adjusted with a few hundred of your own examples) with large language models given **no examples**, only the instructions:



Accuracy from Tables 1 to 4 of Bucher and Martini (2024). The dashed line is what you get by always guessing the most common class.

In two of the four tasks, the best language model with no examples scores **below always guessing the most common class**. The fine-tuned models perform well with between 200 and 500 labelled examples, according to the authors. A language model with no examples is good for getting going on day one and for labelling those first few hundred examples, but it usually isn't what you want in production.

One last warning: a number between 0 and 1 isn't automatically a probability you can trust. Logistic regression comes out well **calibrated**: if it says 80%, it's right about 80% of the time. Modern deep networks, not so much, although it can be fixed afterwards with a single adjustment.



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.