

King of Dumbs

Level: ♠ Technical

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?**

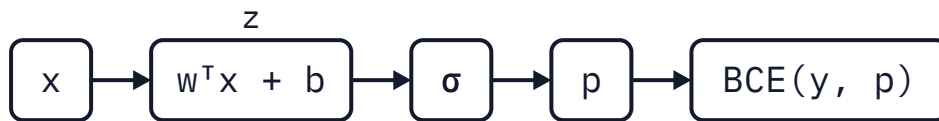
The short answer is yes. A logistic regression is a neural network with **a single neuron**. They aren't lookalikes or cousins: they're the same model with two names. If it's the same neuron, the interesting question changes: what do big networks add, and when are they worth paying for?

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

♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
the score	$z = w^T x + b$
the weights; the constant	$w \in \mathbb{R}^d$; b (bias)
the sigmoid, which squashes between 0 and 1	$\sigma(z) = 1/(1 + e^{-z})$
the probability	$p = \sigma(z) = P(y = 1 \mid x)$
a loaded coin	$y \sim \text{Bernoulli}(p)$; logit link (GLM)
the penalty (loss, log-loss)	$\text{BCE} = -[y \log p + (1 - y) \log(1 - p)]$
which way the penalty goes down	$-\nabla L = -X^T(p - y)/n$
a single bottom	convex: $\nabla^2 L = X^T S X / n \geq 0$
a brake that shrinks the weights	L2 regularization; in sklearn, $C = 1/\lambda$



$$\nabla L = X^T(p - y) / n$$

$$\nabla^2 L = X^T S X / n \geq 0$$

$$S = \text{diag}(p \odot (1 - p))$$

⇒ **convex: a single minimum**

Logistic regression is a generalized linear model: $y \sim \text{Bernoulli}(p)$ with the canonical logit link, so $\text{logit}(p) = w^T x + b$ and $p = \sigma(w^T x + b)$. The sigmoid isn't a design choice: it's the inverse of the Bernoulli's canonical link. A neuron with σ activation trained with binary cross-entropy (BCE) is exactly that model, with the same likelihood.

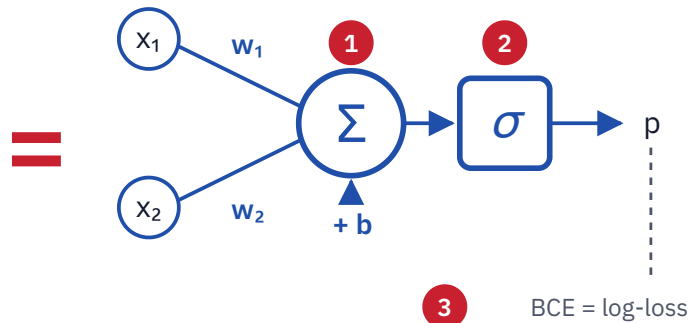
Logistic regression

the way statistics writes it

- 1 Linear score
 $z = w_1 x_1 + w_2 x_2 + b$
- 2 Probability
 $p = \sigma(z)$
- 3 Loss (log-loss)
 $-[y \cdot \log p + (1-y) \cdot \log(1-p)]$

One neuron

the way deep learning draws it



The same three pieces in two notations. The drawing changes, the model doesn't.

With the mean loss $L(w) = (1/n) \sum_i \text{BCE}(y_i, \sigma(w^T x_i))$, the gradient is $\nabla L = X^T(p - y)/n$ and the Hessian is $\nabla^2 L = X^T S X/n$ with $S = \text{diag}(p_i(1 - p_i))$. Since S has a positive diagonal, $X^T S X \geq 0$: the loss is convex, and strictly convex if X has full rank. That's why gradient descent, Newton or IRLS reach the same optimum. With separable data the optimum doesn't exist ($\|w\| \rightarrow \infty$) and you need to regularize.

I checked it with scikit-learn without regularization ($C=\text{np.inf}$) against a neuron trained with gradient descent:

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression

X, y = make_classification(n_samples=2000, n_features=5,
                          flip_y=0.1, random_state=0)
clf = LogisticRegression(C=np.inf).fit(X, y)

w, b = np.zeros(5), 0.0
for _ in range(20000):
    # one neuron: linear + sigmoid
    p = 1 / (1 + np.exp(-(X @ w + b)))
    # gradient of the log-loss
    w -= 0.5 * X.T @ (p - y) / len(y)
    b -= 0.5 * (p - y).mean()

# the same weights
print(clf.coef_[0].round(3), w.round(3))
```

With scikit-learn 1.9.1, the weights match to the third decimal. It isn't a fluke of this dataset: it's the same convex problem solved along two paths.

scikit-learn's logistic regression isn't the textbook one

`LogisticRegression()` applies L2 regularization with $C=1.0$ by default ($C = 1/\lambda$). The coefficients come out shrunk and the odds ratios pulled towards 1. If you want to interpret coefficients, use `C=np.inf` (or `statsmodels`). If you want to predict, tune C with cross-validation.

The King of Dumbs says:

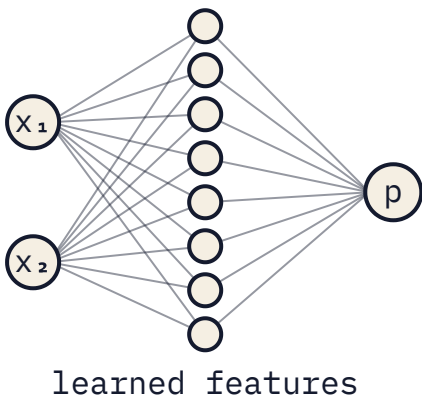
I spent a week thinking I had to choose between statistics and deep learning. Turns out I'd been doing one-neuron deep learning for years without knowing it.



Who builds the features

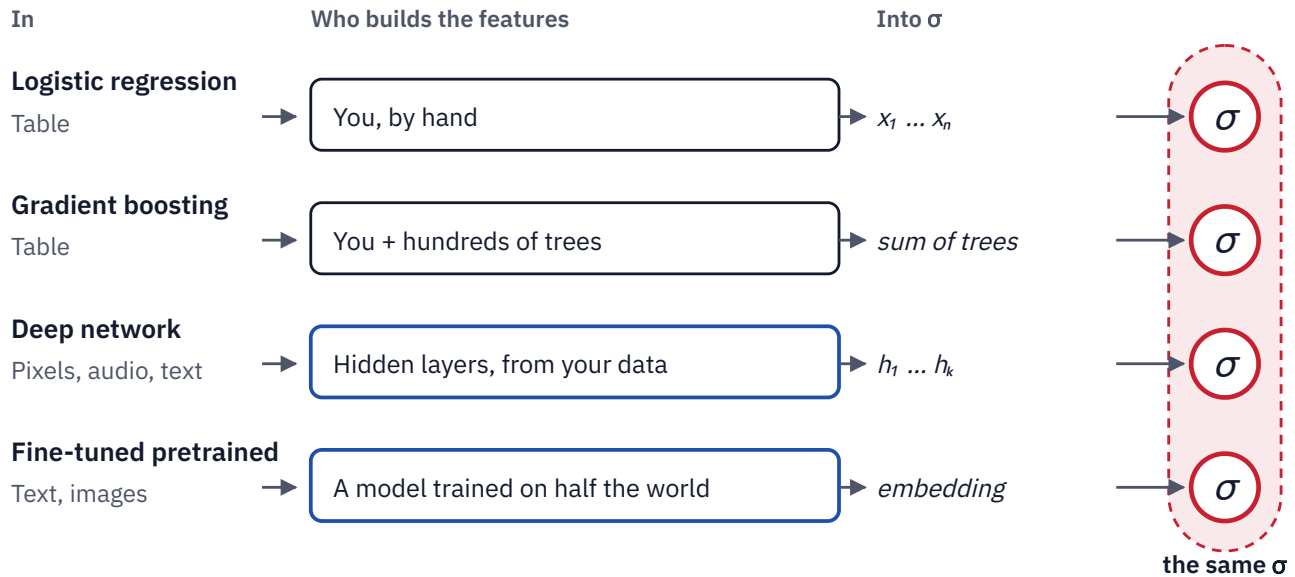
♦ → ♠ From intermediate to technical

In ♦ we said	In ♠ we write
the hidden layers	$h = \tanh(W^{(1)}x + b^{(1)})$
the features they build	$h \in \mathbb{R}^8$: learned representation
the neuron at the end	$p = \sigma(w^{(2)\top}h + b^{(2)})$
the product of the two columns	x_1x_2
hundreds of trees adding up their votes	$p = \sigma(\sum_k f_k(x))$, gradient boosting
a model already trained on half the world's data	pretrained embedding + fine-tuning
the unsquashed score	logit z ; <code>BCEWithLogitsLoss</code>



$$h = \tanh(W^{(1)}x + b^{(1)})$$
$$p = \sigma(w^{(2)\top}h + b^{(2)})$$
$$W^{(1)}: 8 \times 2, w^{(2)}: 8$$
$$|\theta| = 2 \cdot 8 + 8 + 8 + 1$$
$$|\theta| = 33 \text{ vs } 4$$

A binary network with L layers is $p = \sigma(w^T \phi(x) + b)$, with $\phi = h^{(L-1)} \circ \dots \circ h^{(1)}$ learned jointly with w . Conditional on ϕ , the last layer is a logistic regression on the features h . The same holds for almost every binary classifier in use today: in binary boosting (XGBoost binary:logistic, LightGBM binary), $p = \sigma(\sum_k f_k(x))$ with trees f_k ; in a fine-tuned pretrained model, ϕ is the embedding.



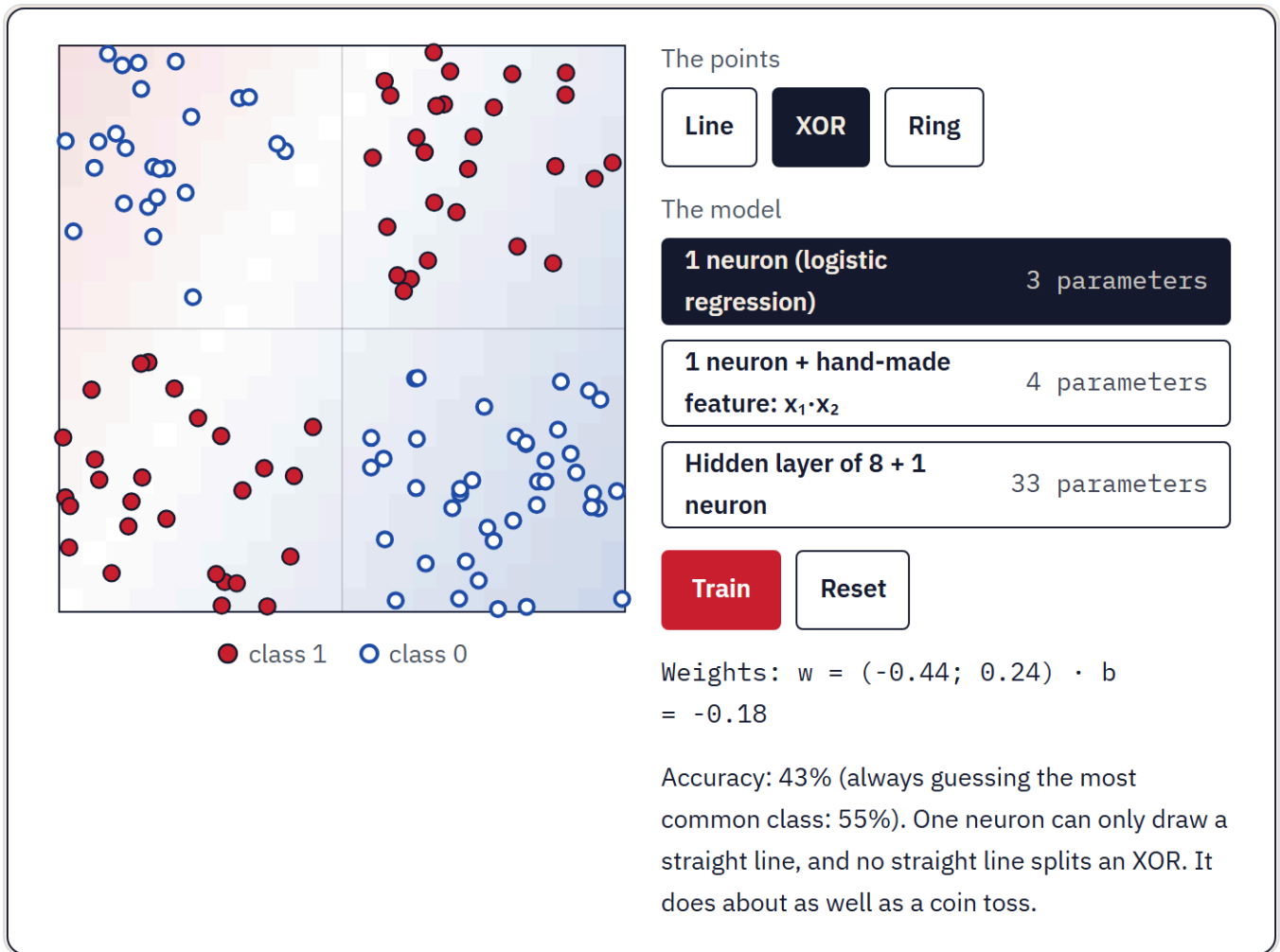
In binary boosting, the output is also a sigmoid applied to the sum of the trees.

In XOR, with $x \in \{-1, 1\}^2$, the class is $y = 1$ if $x_1 x_2 > 0$: no hyperplane in (x_1, x_2) separates it, but in $(x_1, x_2, x_1 x_2)$ the hyperplane $x_1 x_2 = 0$ is enough. The lab's hidden layer (8 tanh neurons) reaches an equivalent boundary without anyone telling it which product to look for, at the cost of $|\theta| = 33$ instead of 4, of losing convexity and of $W^{(1)}$ no longer being readable.

Don't apply the sigmoid twice

In PyTorch, the network returns the logit z without a sigmoid and the loss is `BCEWithLogitsLoss`, which computes $\log \sigma(z)$ stably with the log-sum-exp trick. Putting `nn.Sigmoid()` at the end plus `BCELoss` works until a large $|z|$ gives $\log(0)$. Apply the sigmoid only when predicting.

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:



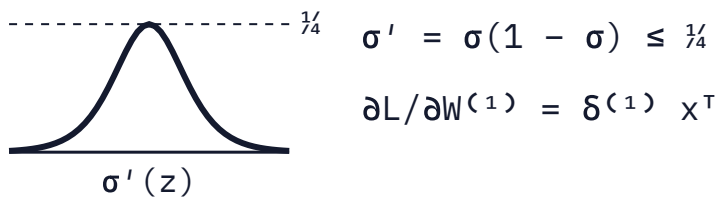
Interactive widget. Try it on the web: <https://kingofdumbs.com/en/lessons/logistic-regression-or-neural-network>

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

♦ → ♠ From intermediate to technical

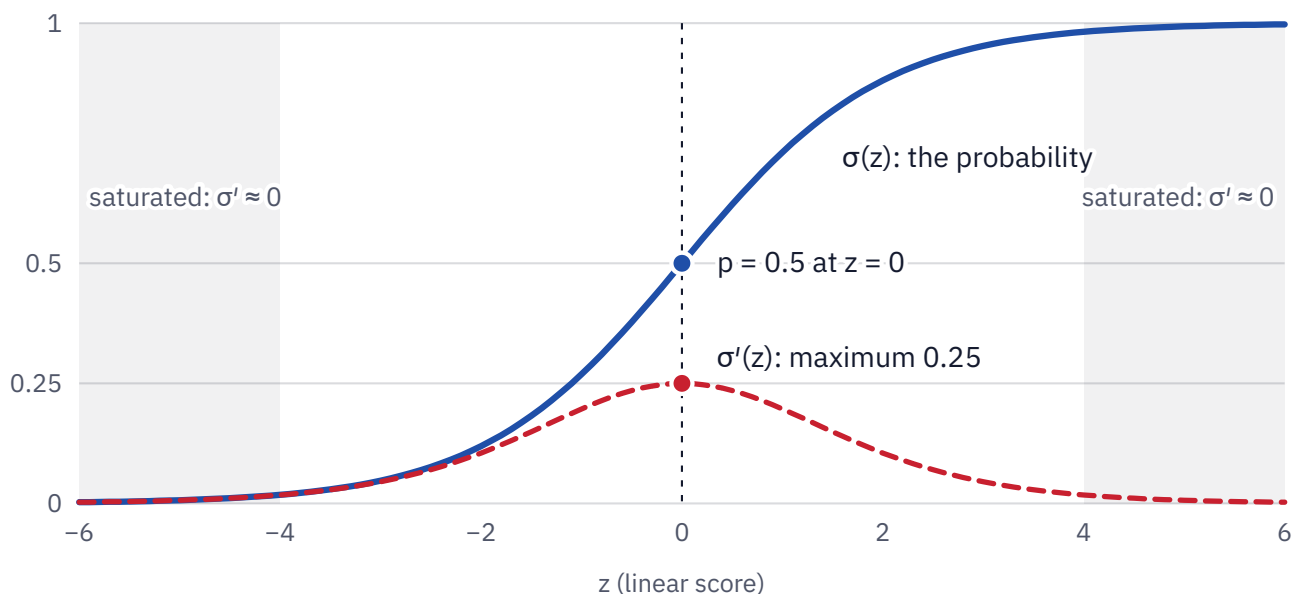
In ♦ we said	In ♠ we write
the sigmoid's slope	$\sigma'(z) = \sigma(z)(1 - \sigma(z))$
0.25 at most	$\sigma' \leq \frac{1}{4}$, with equality at $z = 0$
the signal saying how much and which way to move each weight (gradient)	$\delta^{(1)} = \partial L / \partial z^{(1)}; \partial L / \partial W^{(1)} = \delta^{(1)} h^{(1-1)\top}$
multiplying by each layer's slope (backpropagation)	$\delta^{(1)} = \text{diag}(\sigma'(z^{(1)})) W^{(1+1)\top} \delta^{(1+1)}$
0.25 multiplied by itself once per layer	$(\frac{1}{4})^n$
how much the weights stretch or shrink it	$\ W^{(1)}\ $ (spectral norm)
a better initialization	Glorot/Xavier: $\text{Var}(W) = 2 / (n_{\text{in}} + n_{\text{out}})$
the ReLU	$\text{ReLU}(z) = \max(0, z)$
a gate	$g = \sigma(Wx + b) \odot v$



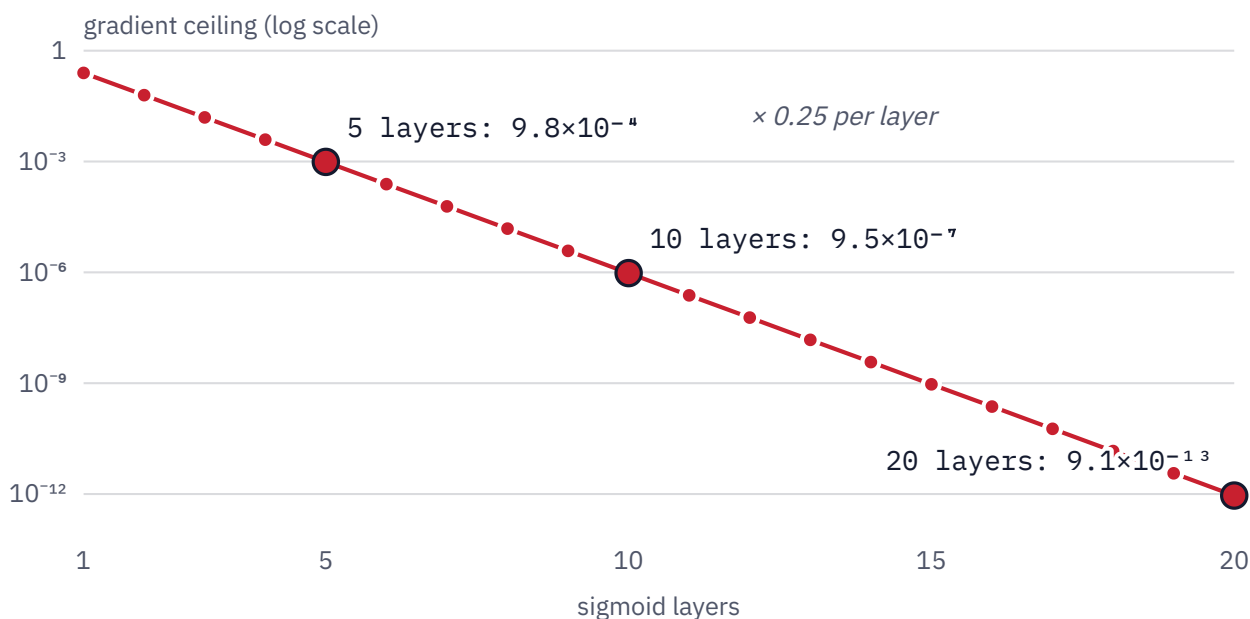
$$\delta^{(1)} = \text{diag}(\sigma'(z^{(1)})) W^{(1+1)\top} \delta^{(1+1)}$$

$$\|\delta^{(1)}\| \leq (\frac{1}{4})^n \prod \|W^{(1)}\| \cdot \|\partial L / \partial h^{(n)}\|$$

With $h^{(l)} = \sigma(z^{(l)})$ and $z^{(l)} = W^{(l)}h^{(l-1)} + b^{(l)}$, backpropagation gives $\delta^{(l)} = \text{diag}(\sigma'(z^{(l)})) W^{(l+1)\top} \delta^{(l+1)}$, and the first layer's gradient is $\partial L / \partial W^{(1)} = \delta^{(1)} x^T$. Since $\sigma' = \sigma(1 - \sigma) \leq \frac{1}{4}$, with n sigmoid layers $\|\delta^{(1)}\| \leq (\frac{1}{4})^n \prod \|W^{(l)}\| \cdot \|\partial L / \partial h^{(n)}\|$. Unless the weight norms make up for the factor of 4 per layer, the gradient decays exponentially with depth (Hochreiter, 1991 and 1998). If the $|z|$ sit in the tails, $\sigma' \approx 0$ and the bound is much lower.



Computed, not hand-drawn. The dashed curve is σ' : its maximum, $\frac{1}{4}$, is the ceiling in the bound.



The ceiling $(\frac{1}{4})^n$ for n from 1 to 20, on a log scale: a straight line with slope $\log_{10} \frac{1}{4} \approx -0.6$ per layer.

Glorot and Bengio (AISTATS 2010, Table 1) measured it with 5 hidden layers on Shapenet-3×2: test error of 82.61% with sigmoid, 27.15% with tanh and 15.60% with tanh and normalized initialization, $\text{Var}(W) = 2/(n_{\text{in}} + n_{\text{out}})$, now called Xavier or Glorot. Also, σ isn't zero-centred: mean activations of 0.5 bias the updates and saturate the top layers. The ReLU, $\max(0, z)$, has derivative 1 for $z > 0$ and breaks the product of factors below $\frac{1}{4}$.

Outside the deep chain, σ keeps two roles: as the output link, $p = \sigma(z)$, in binary and multi-label settings (one σ per class instead of softmax), and as a multiplicative gate. In an LSTM, the input gate is $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$. In SwiGLU (Shazeer, 2020), $\text{FFN}(x) = (\text{Swish}(xW) \odot x)W_2$ with $\text{Swish}(u) = u \cdot \sigma(\beta u)$: LLaMA and PaLM use it in every transformer block.

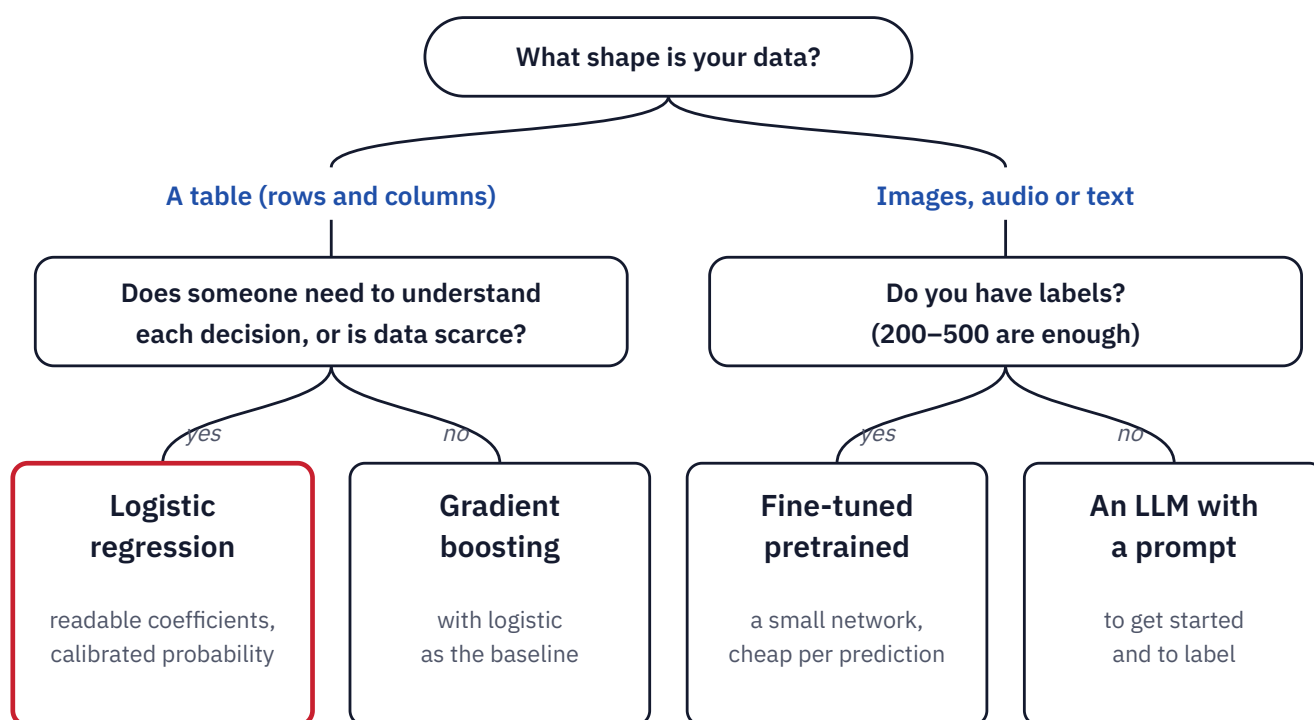
The King of Dumbs says:

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

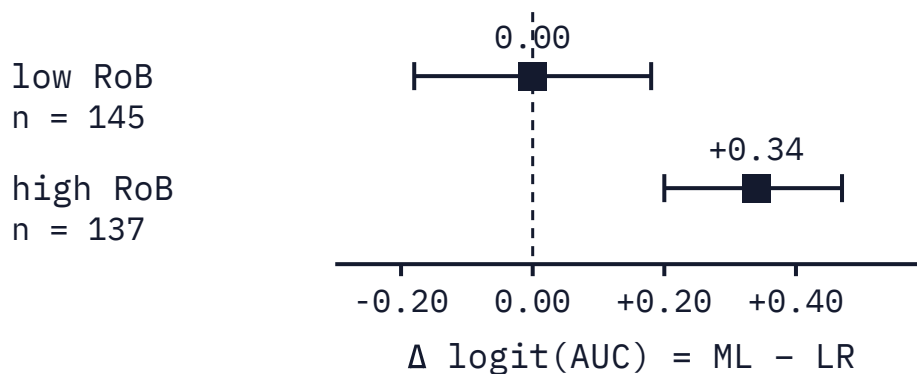
The decision is hardly ever about “how complex the problem is”. It’s about what shape your data has, how many examples with the right answer you have and who has to understand each decision. This tree works for all three levels:



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

◆→♠ From intermediate to technical

In ◆ we said	In ♠ we write
performance measured with the AUC	$\text{AUC} = P(s(x^+) > s(x^-))$
the difference was zero	$\Delta \text{logit}(\text{AUC}) = 0.00, 95\% \text{ CI } [-0.18, 0.18]$
risk of bias	RoB (risk of bias)
always guessing the most common class	majority-class baseline
fine-tuned; no examples	fine-tuning; zero-shot
trees adding up their votes	GBDT (XGBoost, LightGBM)
well calibrated	$P(y = 1 \mid \hat{p} = q) \approx q$; ECE
a single adjustment	temperature scaling: $p = \sigma(z/T)$



Christodoulou et al. (*J Clin Epidemiol*, 2019): 71 studies and 282 comparisons. At low RoB (145), $\Delta \text{logit}(\text{AUC}) = 0.00 [-0.18, 0.18]$; at high RoB (137), $+0.34 [0.20, 0.47]$. ML's apparent advantage is concentrated where validation is weakest. I only read the abstract. On a benchmark of 45 tabular datasets, Grinsztajn et al. (2022) find that tree-based models remain state of the art on medium-sized data (around 10,000 rows): networks struggle with uninformative features and irregular target functions.

On text, Bucher and Martini (2024) compare the best fine-tuned model with the best zero-shot generative LLM (GPT-3.5, GPT-4 and Claude 3 Opus) on four tasks. In two of them, zero-shot falls below the majority-class baseline (0.73 and 0.50). According to the authors (§5.5), fine-tuning performs well with between 200 and 500 labels.

Being between 0 and 1 doesn't make it a probability

Logistic regression maximizes a Bernoulli likelihood, so it comes out well calibrated if the model is well specified. Modern deep networks don't: Guo et al. (*ICML* 2017, Table 1) measured an ECE of 16.53% for a ResNet-110 on CIFAR-100, which dropped to 1.26% with *temperature scaling*, a

single parameter T fitted on validation data: $p = \sigma(z/T)$, or $\text{softmax}(z/T)$ with several classes. If the output feeds a decision that costs money, recalibrate.



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.