

King of Dumbs

笨蛋之王

级别: ♠ 技术

逻辑回归还是神经网络？ 其实是同一个神经元

逻辑回归就是只有一个神经元的神经网络。深度学习多了什么、什么时候值得用、什么时候经典模型就够了，分三个层次讲清楚。

中文 发布于 2026年10月4日



扫码打开可互动的课程

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

你将学到

- 逻辑回归
- 神经网络
- sigmoid 函数
- 梯度消失
- 在经典机器学习和深度学习之间做选择

本课会出现的符号

E_a 下标: 右下角的标签 (♠) · Σ 求和: 全部加起来 (♠) · x_i 下标 i : 第 i 个元素 (♠) · Π 连乘: 全部乘起来 (♠) · $\approx$ 约等于 (♠) · $\sim$ 数量级约为 / 服从...分布 (♠) · $\leq \geq$ 小于等于 / 大于等于 (♠) · $|x|$ 绝对值: 去掉正负号 (♠) · 10^{-3} 负指数: 表示除 (♠) · $h^{(l)}$ 带括号的上标: 不是乘方 (♠) · e 数 e (≈ 2.718) (♠) · $\log$ 对数: 要乘几次? (♠) · σ sigma: 含义因课而异 (♠) · λ lambda: 一个缩放用的数 (♠) · δ 小写 delta: 小误差 (♠) · $\|x\|$ 范数: 箭头的长度 (♠) · x^T 转置: 把列放倒 (♠) · $\in$ 属于 (♠) · $\mathbb{R} \mathbb{Z}$ 数集 (♠) · dE/dt 导数: 变化的快慢 (♠) · ∂ 偏导数: 只改变一个量 (♠) · ∇ 梯度: 上升最快的方向 (♠)

第一次看到画出来的神经网络，一堆小圆圈和箭头，我以为它和传统统计学是两回事。后来我得亲手写一个逻辑回归，于是冒出了这节课的笨问题：**逻辑回归是不是神经网络？**

简短的回答是：是的。逻辑回归就是只有一个**神经元**的神经网络。它们不是长得像，也不是亲戚，而是同一个模型，只是有两个名字。既然是同一个神经元，真正有意思的问题就变了：大型网络到底多了什么？什么时候值得为它付出代价？

选一张牌吧：♣ 适合从零开始，♦ 适合知道函数或平均数是什么的人，♠ 适合想看数学记法的人。随时可以换。

一个神经元，两个名字

♦ → ♠ 从进阶到技术

♦ 里的说法	♠ 里的写法
得分	$z = w^T x + b$
权重；常数	$w \in \mathbb{R}^d$; b (偏置)
把得分压到 0 和 1 之间的 sigmoid	$\sigma(z) = 1/(1 + e^{-z})$
概率	$p = \sigma(z) = P(y = 1 \mid x)$
做了手脚的硬币	$y \sim \text{Bernoulli}(p)$; logit 连接函数 (GLM)
惩罚 (损失、对数损失)	$\text{BCE} = -[y \log p + (1 - y) \log(1 - p)]$
惩罚往哪个方向下降	$-\nabla L = -X^T(p - y)/n$
只有一个底	凸函数: $\nabla^2 L = X^T S X / n \geq 0$
把权重往小里压的刹车	L2 正则化; sklearn 中 $C = 1/\lambda$



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

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

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

⇒ 凸函数：只有一个最小值

逻辑回归是一个广义线性模型： $y \sim \text{Bernoulli}(p)$ ，使用典型的 logit 连接函数，所以 $\text{logit}(p) = w^T x + b$ ， $p = \sigma(w^T x + b)$ 。sigmoid 并不是设计上的选择，而是伯努利分布典型连接函数的反函数。一个用 σ 作激活函数、用二元交叉熵 (BCE) 训练的神经元，正是这个模型，似然函数也完全相同。

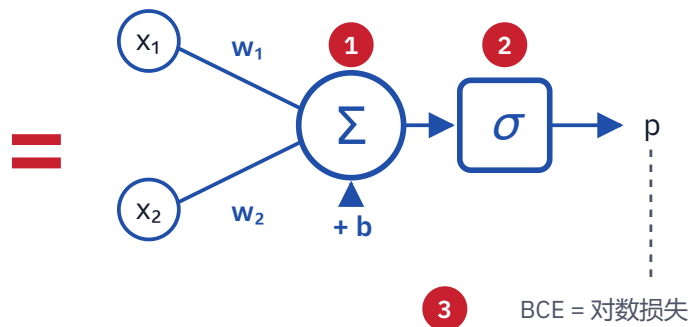
逻辑回归

统计学的写法

- 1 线性得分
 $z = w_1 x_1 + w_2 x_2 + b$
- 2 概率
 $p = \sigma(z)$
- 3 损失 (对数损失)
 $-[y \cdot \log p + (1-y) \cdot \log(1-p)]$

一个神经元

深度学习的画法



同样的三个部分，两种记法。变的是画法，不是模型。

取平均损失 $L(w) = (1/n) \sum_i \text{BCE}(y_i, \sigma(w^T x_i))$ ，梯度为 $\nabla L = X^T(p - y)/n$ ，Hessian 矩阵为 $\nabla^2 L = X^T S X/n$ ，其中 $S = \text{diag}(p_i(1 - p_i))$ 。由于 S 的对角线为正， $X^T S X \geq 0$ ：损失是凸的；若 X 满秩，则是严格凸的。所以梯度下降、牛顿法或 IRLS 都会到达同一个最优解。数据线性可分时，最优解不存在 ($\|w\| \rightarrow \infty$)，需要正则化。

我用不加正则化的 scikit-learn ($C=\text{np.inf}$) 和一个用梯度下降训练的神经元做了对比：

```
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):
    # 一个神经元：线性 + sigmoid
    p = 1 / (1 + np.exp(-(X @ w + b)))
    # 对数损失的梯度
    w -= 0.5 * X.T @ (p - y) / len(y)
    b -= 0.5 * (p - y).mean()

# 权重相同
print(clf.coef_[0].round(3), w.round(3))
```

用 scikit-learn 1.9.1，两边的权重一直到小数点后第三位都相同。这不是这组数据碰巧如此：它们是同一个凸优化问题，走两条路解出来而已。

scikit-learn 的逻辑回归不是教科书上的那个

`LogisticRegression()` 默认使用 $C=1.0$ 的 L2 正则化 ($C = 1/\lambda$)。系数会被压小, 优势比会被拉向 1。如果你要解释系数, 用 `C=np.inf` (或者 `statsmodels`)。如果你要做预测, 用交叉验证调 `C`。

笨蛋之王说:

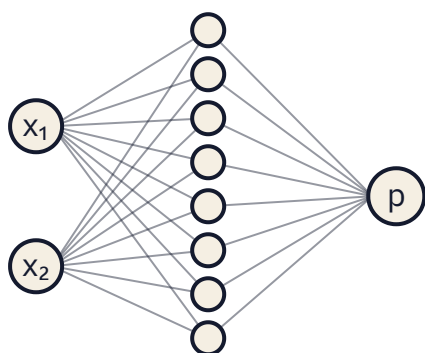
我曾经纠结了一个星期, 以为必须在统计和深度学习之间二选一。结果发现, 我早就在做只有一个神经元的深度学习了。



谁来制造特征

♦ → ♠ 从进阶到技术

♦ 里的说法	♠ 里的写法
隐藏层	$h = \tanh(W^{(1)}x + b^{(1)})$
它们制造的特征	$h \in \mathbb{R}^8$: 学到的表示
最后那个神经元	$p = \sigma(W^{(2)T}h + b^{(2)})$
两列的乘积	$x_1 x_2$
几百棵把票数加起来的树	$p = \sigma(\sum_k f_k(x))$, 梯度提升
已经用全世界数据训练好的模型	预训练嵌入 + 微调
没有压缩的得分	logit z ; <code>BCEWithLogitsLoss</code>



学到的特征

$$h = \tanh(W^{(1)}x + b^{(1)})$$

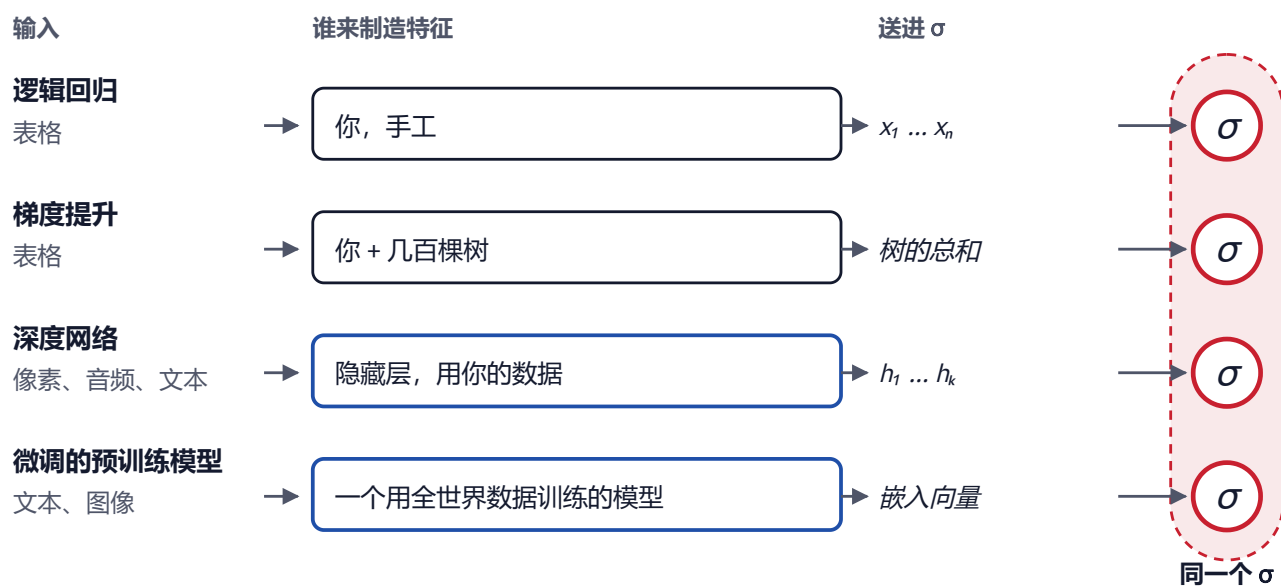
$$p = \sigma(W^{(2)T}h + b^{(2)})$$

$$W^{(1)}: 8 \times 2, W^{(2)}: 8$$

$$|\theta| = 2 \cdot 8 + 8 + 8 + 1$$

$$|\theta| = 33 \text{ vs } 4$$

一个 L 层的二分类网络可以写成 $p = \sigma(w^T \varphi(x) + b)$ ，其中 $\varphi = h^{(L-1)} \circ \dots \circ h^{(1)}$ 与 w 一起学习。给定 φ ，最后一层就是以特征 h 为输入的逻辑回归。如今几乎所有二分类器都是如此：在二分类梯度提升中 (XGBoost binary:logistic、LightGBM binary)， $p = \sigma(\sum_k f_k(x))$ ， f_k 是树；在微调的预训练模型中， φ 就是嵌入。



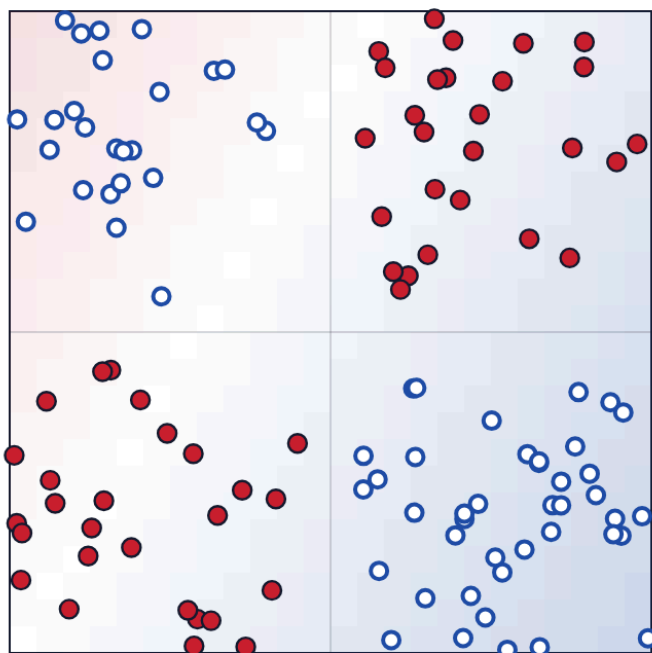
在二分类的梯度提升里，输出同样是把 sigmoid 作用在所有树的总和上。

在异或问题中， $x \in \{-1, 1\}^2$ ，当 $x_1 x_2 > 0$ 时 $y = 1$ ：在 (x_1, x_2) 空间里没有超平面能把它分开，但在 $(x_1, x_2, x_1 x_2)$ 空间里，超平面 $x_1 x_2 = 0$ 就够了。实验室里的隐藏层（8 个 tanh 神经元）不需要任何人告诉它找哪个乘积，也能得到等价的边界，代价是 $| \theta | = 33$ 而不是 4，失去凸性，而且 $W^{(1)}$ 再也读不懂。

别把 sigmoid 用两次

在 PyTorch 里，网络输出不带 sigmoid 的 logit z ，损失用 BCEWithLogitsLoss，它借助 log-sum-exp 技巧稳定地计算 $\log \sigma(z)$ 。在最后加 nn.Sigmoid() 再配 BCELoss 也能跑，直到某个很大的 $|z|$ 产生 $\log(0)$ 。sigmoid 只在预测时使用。

自己试试。异或和圆环是两个经典例子：没有任何一条直线能把这些点分开。选一组数据和一个模型，然后训练：



● 第1类 ○ 第0类

数据点

直线

异或

圆环

模型

1 个神经元 (逻辑回归)

3 个参数

1 个神经元 + 手工特征:

4 个参数

$x_1 \cdot x_2$

8 个神经元的隐藏层 + 1

个神经元

33 个参数

训练

重置

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

准确率: 43% (永远猜最常见的类别: 55%)。一个神经元只会画直线, 而没有任何直线能分开异或。准确率和抛硬币差不多。



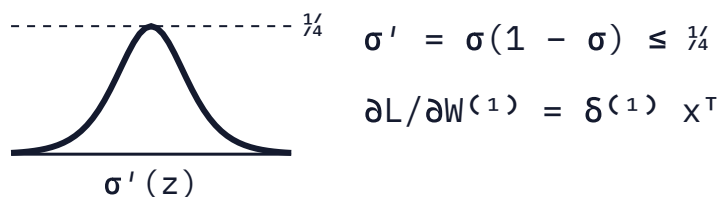
互动小工具。 在网站上试一试: <https://kingofdumbs.com/zh/lessons/logistic-regression-or-neural-network>

这个实验里我最喜欢的是中间那一行。在异或上失败的同一个神经元, 只要手工给它两个坐标的乘积, 马上就全对了。隐藏层不需要任何人告诉它该找什么, 也能到达同样的地方, 代价是要调 33 个数而不是 4 个, 而且你再也读不懂它学到了什么。

sigmoid 为什么退出了隐藏层

♦ → ♠ 从进阶到技术

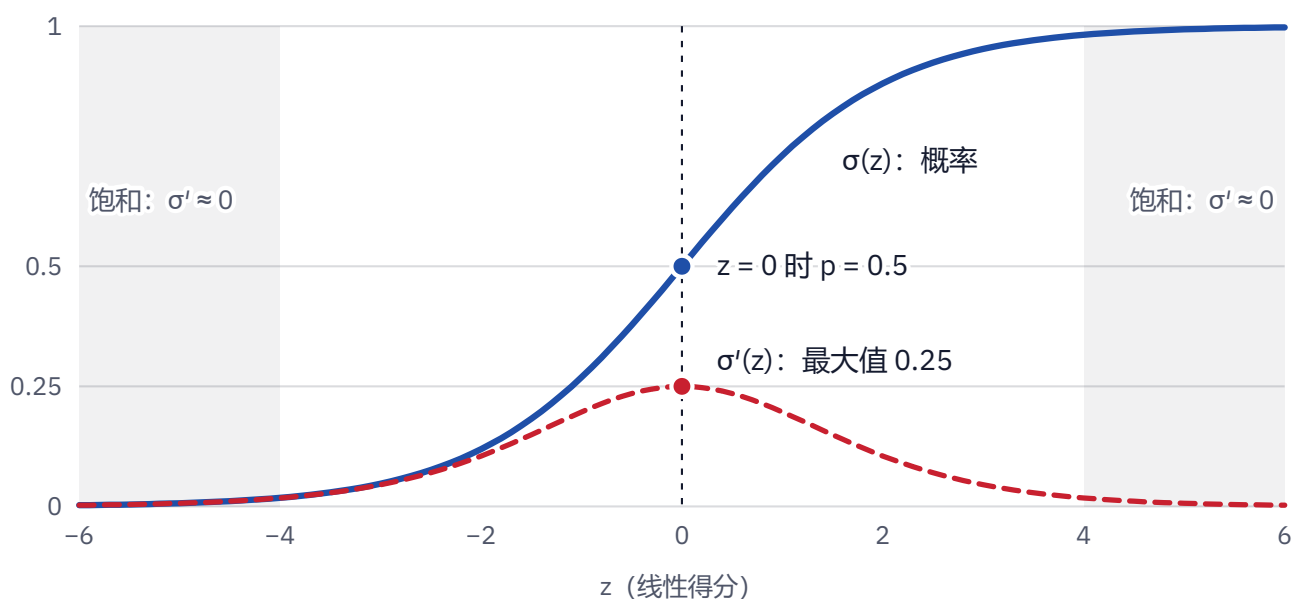
♦ 里的说法	♠ 里的写法
sigmoid 的斜率	$\sigma'(z) = \sigma(z)(1 - \sigma(z))$
最大只有 0.25	$\sigma' \leq 1/4$, 在 $z = 0$ 处取等号
告诉每个权重该往哪里挪、挪多少的信号（梯度）	$\delta^{(1)} = \partial L / \partial z^{(1)}; \partial L / \partial W^{(1)} = \delta^{(1)} h^{(1-1)T}$
每经过一层就乘上那一层的斜率（反向传播）	$\delta^{(1)} = \text{diag}(\sigma'(z^{(1)})) W^{(1+1)T} \delta^{(1+1)}$
0.25 自乘层数那么多次	$(1/4)^n$
权重把它放大或缩小了多少	$\ W^{(1)}\ $ （谱范数）
更好的初始化	Glorot/Xavier: $\text{Var}(W) = 2 / (n_{in} + n_{out})$
ReLU	$\text{ReLU}(z) = \max(0, z)$
门	$g = \sigma(Wx + b) \odot v$



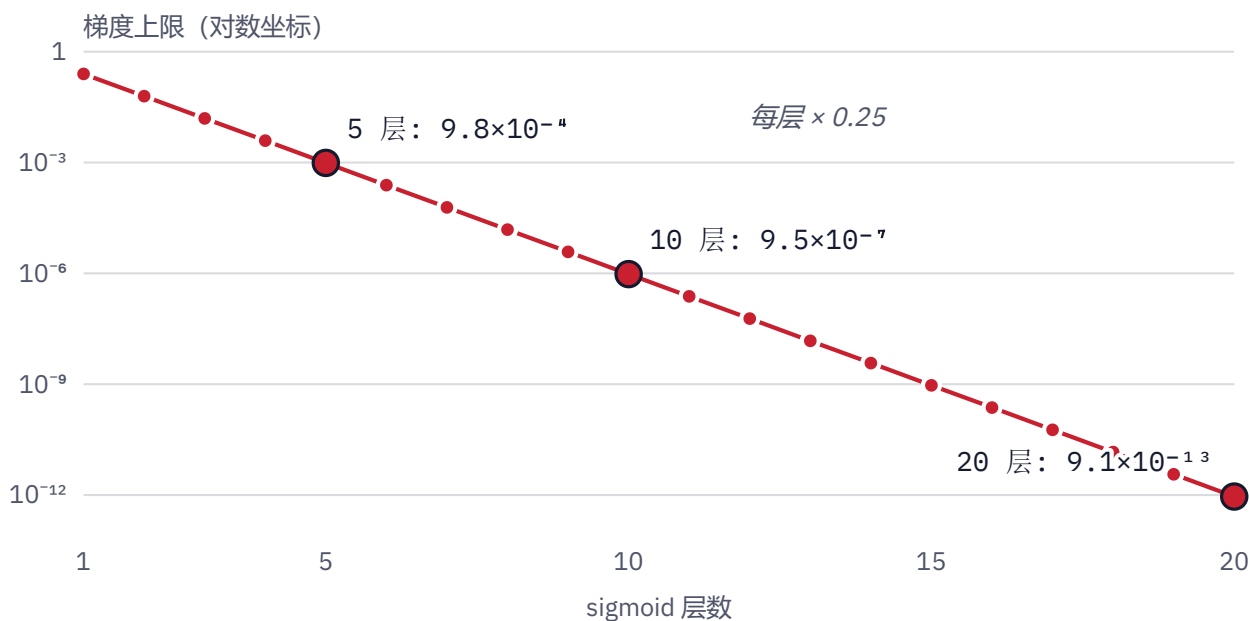
$$\delta^{(1)} = \text{diag}(\sigma'(z^{(1)})) W^{(1+1)T} \delta^{(1+1)}$$

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

设 $h^{(i)} = \sigma(z^{(i)})$, $z^{(i)} = W^{(i)}h^{(i-1)} + b^{(i)}$, 反向传播给出 $\delta^{(i)} = \text{diag}(\sigma'(z^{(i)})) W^{(i+1)T} \delta^{(i+1)}$, 第一层的梯度为 $\partial L / \partial W^{(1)} = \delta^{(1)} x^T$ 。由于 $\sigma' = \sigma(1 - \sigma) \leq 1/4$, 有 n 个 sigmoid 层时, $\|\delta^{(1)}\| \leq (1/4)^n \prod \|W^{(i)}\| \cdot \|\partial L / \partial h^{(n)}\|$ 。除非权重的范数能抵消每层 4 倍的衰减, 梯度会随深度指数衰减 (Hochreiter, 1991 年和 1998 年)。如果 $|z|$ 落在尾部, $\sigma' \approx 0$, 上界还要低得多。



曲线是算出来的，不是手画的。虚线是 σ' ：它的最大值 $\frac{1}{4}$ 就是上界里的那个上限。



n 从 1 到 20 时的上限 $(\frac{1}{4})^n$ ，对数坐标：一条每层斜率为 $\log_{10} \frac{1}{4} \approx -0.6$ 的直线。

Glorot 和 Bengio (AISTATS 2010, 表 1) 在 Shapenet-3×2 上用 5 个隐藏层做了测量：测试误差用 sigmoid 是 82.61%，用 tanh 是 27.15%，用 tanh 加归一化初始化是 15.60%。这种初始化 $\text{Var}(W) = 2/(n_{in} + n_{out})$ 如今叫作 Xavier 或 Glorot 初始化。此外， σ 不是以零为中心的：0.5 的平均激活值会让更新产生偏差，并使高层饱和。ReLU，即 $\max(0, z)$ ，在 $z > 0$ 时导数为 1，打破了这串小于 $\frac{1}{4}$ 的因子连乘。

离开深层链条， σ 仍有两个角色：一是作为输出的连接函数 $p = \sigma(z)$ ，用于二分类和多标签分类（每个类别一个 σ ，而不是 softmax）；二是作为乘法门。在 LSTM 中，输入门是 $i_t = \sigma(W_i[h_{t-1}, x_t] + b_i)$ 。在 SwiGLU (Shazeer, 2020) 中， $\text{FFN}(x) = (\text{Swish}(xW) \odot xV)W_2$ ，其中 $\text{Swish}(u) = u \cdot \sigma(\beta u)$ ：LLaMA 和 PaLM 在每个 Transformer 块里都用它。

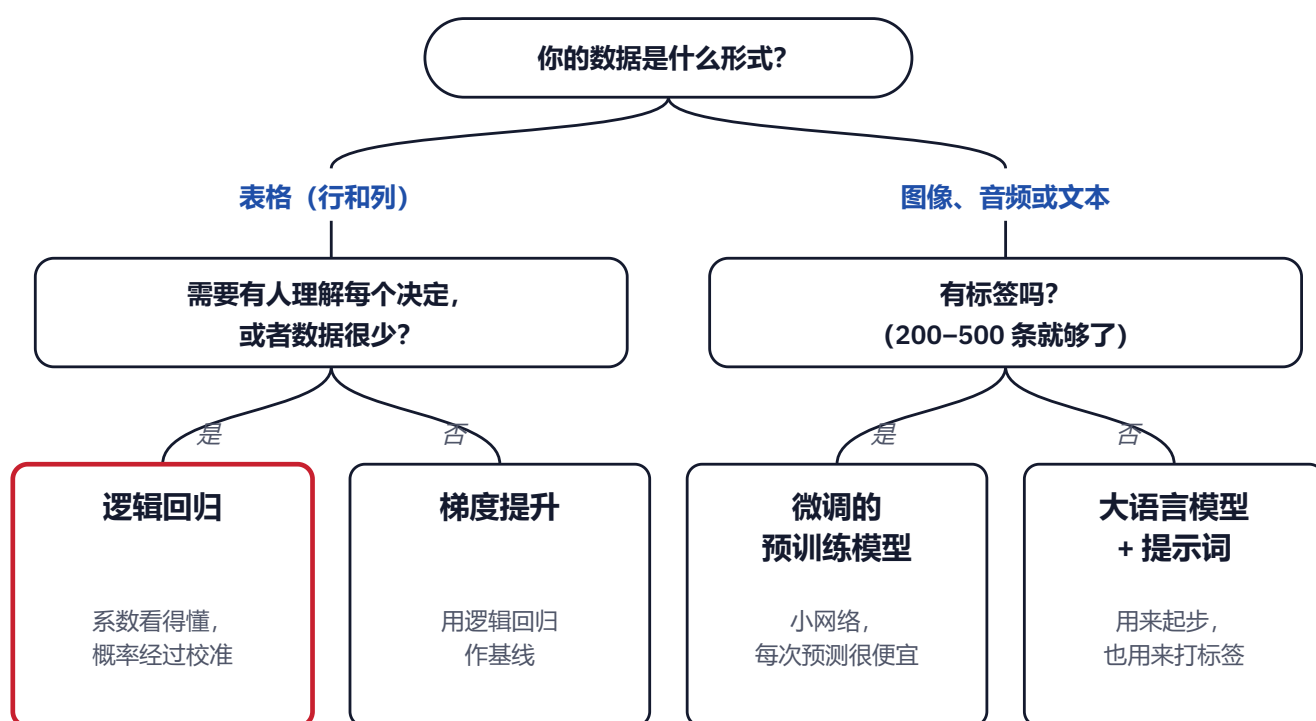
笨蛋之王说：

sigmoid 跟我一样：被赶出了隐藏层，最后在每个大语言模型里当了门卫。



什么时候用什么

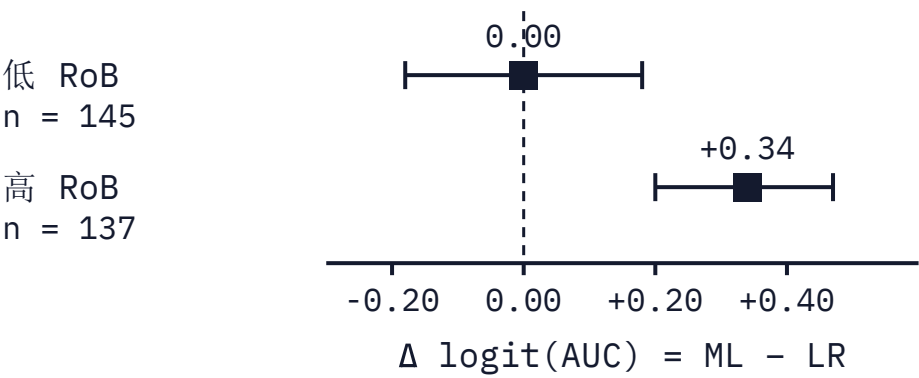
做决定时几乎从来不看“问题有多复杂”，而是看你的数据是什么形式、你有多少带正确答案的样本、谁需要理解每一个决定。这棵决策树三个层次都适用：



用于预测一个标签。往简单走很便宜，往复杂走很贵：从左边开始。

◆→♣从进阶到技术

◆里的说法	♣里的写法
用 AUC 衡量表现	$AUC = P(s(x^+) > s(x^-))$
两者没有差别	$\Delta \text{logit}(AUC) = 0.00, 95\% \text{ CI } [-0.18, 0.18]$
偏倚风险	RoB (risk of bias)
永远猜最常见的类别	多数类基线
微调; 不给示例	fine-tuning; zero-shot
把票数加起来的树	GBDT (XGBoost、LightGBM)
校准良好	$P(y = 1 \mid \hat{p} = q) \approx q$; ECE
一个调整	温度缩放: $p = \sigma(z/T)$



Christodoulou 等 (*J Clin Epidemiol*, 2019) : 71 项研究, 282 组对比。低 RoB (145 组) 时, $\Delta \text{logit}(AUC) = 0.00 [-0.18, 0.18]$; 高 RoB (137 组) 时, 为 $+0.34 [0.20, 0.47]$ 。机器学习表面上的优势集中在验证做得最差的地方。我只读了摘要。在一个包含 45 个表格数据集的基准上, Grinsztajn 等 (2022) 发现, 对于中等规模的数据 (约一万行), 树模型仍是最先进的方法: 神经网络难以应对无信息的特征和不规则的目标函数。

在文本上, Bucher 和 Martini (2024) 在四个任务上比较了最好的微调模型和最好的零样本生成式大语言模型 (GPT-3.5、GPT-4 和 Claude 3 Opus)。其中两个任务里, 零样本的结果低于多数类基线 (0.73 和 0.50)。据作者说 (§5.5), 微调用 200 到 500 个标签就能表现很好。

落在 0 到 1 之间不等于概率

逻辑回归最大化的是伯努利似然，所以只要模型设定正确，它天生就校准良好。现代深度网络不是这样：Guo 等（ICML 2017，表 1）测得一个 ResNet-110 在 CIFAR-100 上的 ECE 是 16.53%，用温度缩放后降到了 1.26%。温度缩放只有一个参数 T ，在验证集上拟合： $p = \sigma(z/T)$ ，多分类时是 $\text{softmax}(z/T)$ 。如果输出要用于有成本的决策，记得重新校准。



最后的挑战

1. 一个逻辑回归和一个带 sigmoid 的神经元，用同样的惩罚训练，它们是什么关系？

- a. 亲戚：长得像，但结果不同
- b. 同一个模型，两个名字
- c. 神经元总是更准，因为它是深度学习

2. 为什么隐藏层不再用 sigmoid 了？

- a. 因为它算起来很慢
- b. 因为误差信号每过一层就缩小，前面几层几乎学不到东西
- c. 因为 0 到 1 之间的数没什么用

3. 你有一张 5000 行的表格（年龄、金额、访问次数），而且有人需要理解每一个决定。你从哪里开始？

- a. 给大语言模型写一个提示词
- b. 一个十层的深度网络
- c. 一个逻辑回归



三句话总结

1. 逻辑回归就是只有一个神经元的神经网络：同样的加权求和，同样的 S 形曲线（sigmoid），同样的惩罚。
2. 深度学习多出来的是前面那些层，它们负责制造特征；sigmoid 退出这些层，是因为它会把误差信号压没，如今它在输出端和门里工作。
3. 如果你的数据是表格，而且每一列本来就有含义，就从逻辑回归开始，几分钟就能跑完，还能告诉你复杂方案要赢多少才值得；如果是像素、音频或文本，就得有人来制造特征。



资料来源

- 逻辑回归作为广义线性模型、sigmoid 作为其典型连接函数：[斯坦福 CS229 讲义](#)。对数损失的凸性：Boyd 和 Vandenberghe, [Convex Optimization](#), §7.1.1。
- LogisticRegression 的默认设置 (L2, C=1.0)：[scikit-learn 文档](#)。
- 梯度消失和 sigmoid 导数的最大值：Hochreiter, 1991 年的博士论文 ([PDF](#), 德语) 和 1998 年的论文 ([PDF](#))。
- sigmoid、tanh 以及带归一化初始化的 tanh 的测试误差：Glorot 和 Bengio, *AISTATS* 2010, [表 1](#)。
- SwiGLU: Shazeer, [arXiv:2002.05202](#); 在 [LLaMA](#) 和 [PaLM](#) 中的使用。LSTM 的门：[Hochreiter 和 Schmidhuber, 1997](#)。
- 临床预测中机器学习与逻辑回归的对比：Christodoulou 等, *J Clin Epidemiol* 110, 12 (2019), [doi:10.1016/j.jclinepi.2019.02.004](#)。只读了摘要。
- 表格数据上树模型与神经网络的对比：Grinsztajn 等, [arXiv:2207.08815](#)。
- 微调模型与零样本大语言模型的对比：Bucher 和 Martini, [arXiv:2406.08660](#), 表 1 到表 4 及 §5.5。
- 现代网络的校准：Guo 等, *ICML* 2017, [arXiv:1706.04599](#), 表 1。
- BCEWithLogitsLoss: [PyTorch 文档](#)。

这节课是怎么写成的：基于我的机器学习笔记，由 Claude 协助整理并写出三个层次，再由 Claude 从西班牙语翻译，正式发布前会经过人工审校。所有数字都在 2026 年 10 月 4 日对照各篇论文的原始表格核对过。神经元实验室在你的浏览器里用梯度下降训练，和 ♠ 层次里的代码一样。插图均为原创。

答案

1. **b)** 同一个模型，两个名字. 答对了。同样的三个部分，最后得到同样的权重。
2. **b)** 因为误差信号每过一层就缩小，前面几层几乎学不到东西. 没错：每层最多 0.25。十层之后，连一点声音都传不过来。
3. **c)** 一个逻辑回归. 很好。每一列本来就有含义，决定还要能解释：逻辑回归就是起点。

King of Dumbs 的课程，kingofdumbs.com。这是离线版；可以动手玩的原版在：

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

更新于 2026年10月4日.