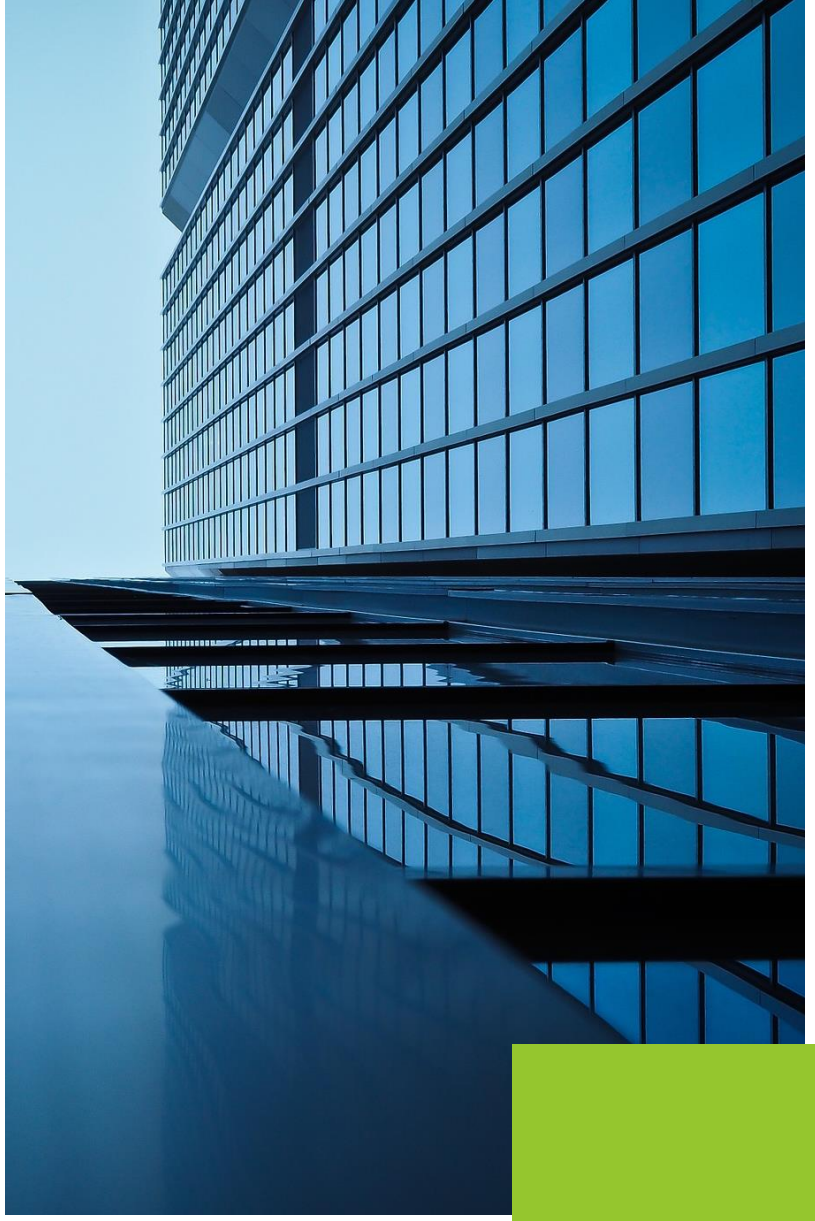
人工智能 – 神经网络基础

通汇诚泰信息科技（上海）有限公司

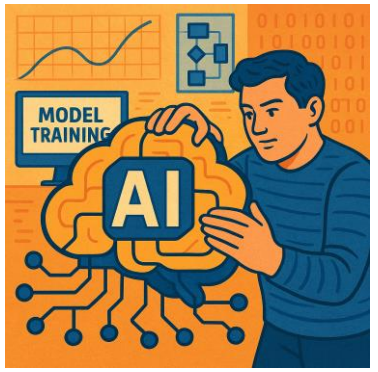
THChengtay Information Technology (Shanghai) Co., Ltd

先行通
XIANXINGTONG

目录
CONTENTS

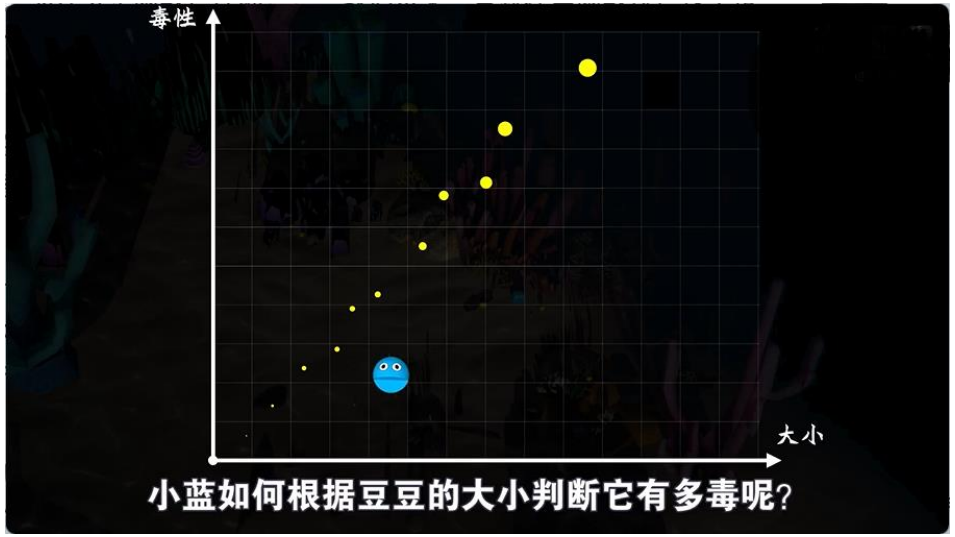


01 神经网络基础



小蓝的生存智慧

在很深很深很深的海底住着一个简简单单的生物小蓝，他的世界很简单，去寻找周围能够吃到了一种叫做豆豆的植物，并靠此生存。但是这些豆豆在漫长的演化历史中，通过和小蓝的反复博弈，为了保护自己逐渐进化出了毒性，毒性的强弱和豆豆的大小有关系，而小蓝也进化出了一个能够检测豆豆大小的器官，那么问题来了，小蓝如何根据豆豆的大小判断它到底有多毒呢 ？



1 它还缺一个思考的器官，也就是我们常说的脑子

认知以前当然就是一无所知，这种情况下认知事物唯一的方法就是依靠直觉。那我们又该如何去描述直觉呢？很明显是函数。为什么是函数？仔细想想，我们本来一直就在用函数认识这个世界，比如在经济学中我们把消费投资、政府购买和进出口作为自变量，可以形成一个关于GDP的函数。

2 对世界认知的过程

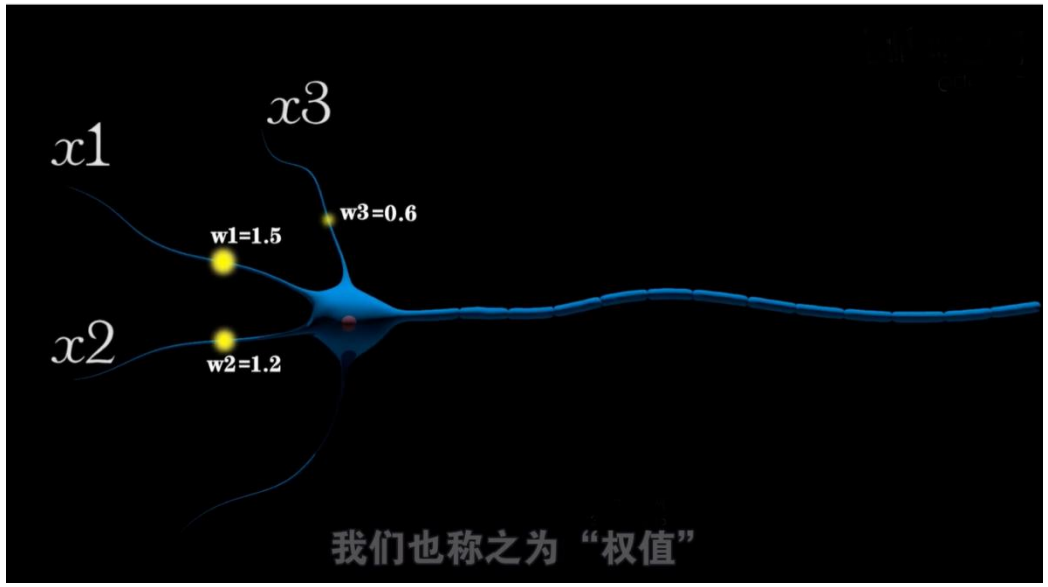
如此这般把智能体对世界认知的过程看作是在脑中不断形成各种函数，似乎也就好有道理，既然是直觉，那么就不需要太多的理由，豆豆的毒性和它的大小有关系，所以这里的直觉自然是一个一元一次函数，豆豆的大小x是自变量，毒性y是因变量，而w是一个确定的参数。就是我们常说的直线的斜率，一个简单的一元一次函数就可以描述一个直觉，建立一种思考的模型。当我们去类比生物神经元的时候，你就会发现用一个一元一次函数去描述认知，可不是乱用的，完全是有备而来。

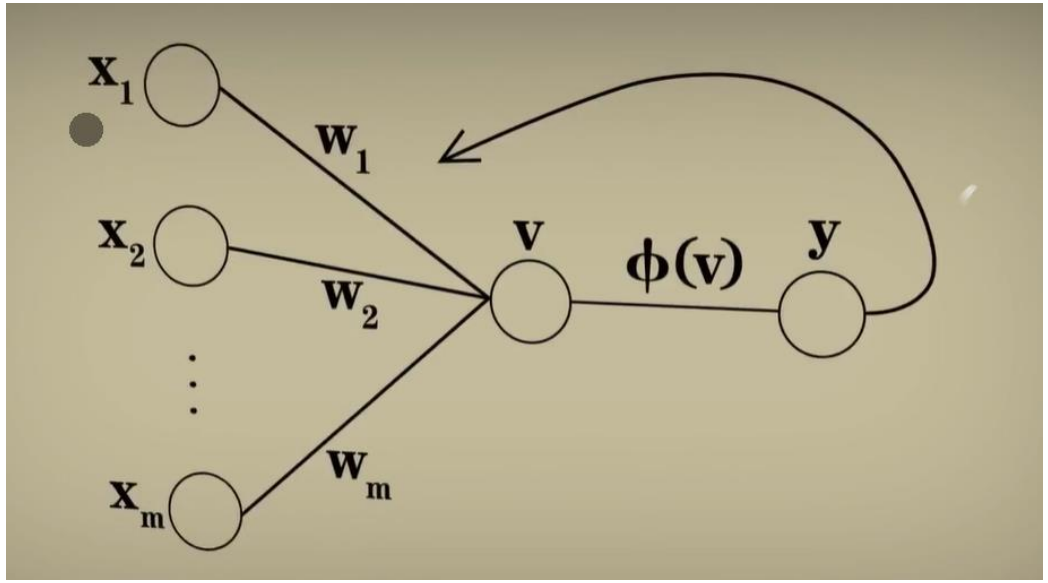
$$y = w*x$$

3 McCulloch-Pitts模型

1943年由神经学家McCulloch和数学家Pitts在他们合作的论文中提出了一种神经元模型，McCulloch-Pitts模型。

McCulloch-Pitts模型





早在1943年由神经学家McCulloch和数学家Pitts在他们合作的论文中提出了一种神经元模型，McCulloch-Pitts模型。这个模型是对生物神经元一种相当简化的模仿，通过树突进入神经元，再通过轴突输出结果，它们分别对应着函数中的自变量和因变量输出。

1 树突 输入信号 (Inputs - 对应图中的 x1, x2, x3)

它接收来自外界的多种不同信息。比如，对于“小蓝”来说：
x1 可能代表“豆豆”的大小。
x2 可能代表“豆豆”的颜色鲜艳程度。
x3 可能代表“豆豆”散发的气味强度。

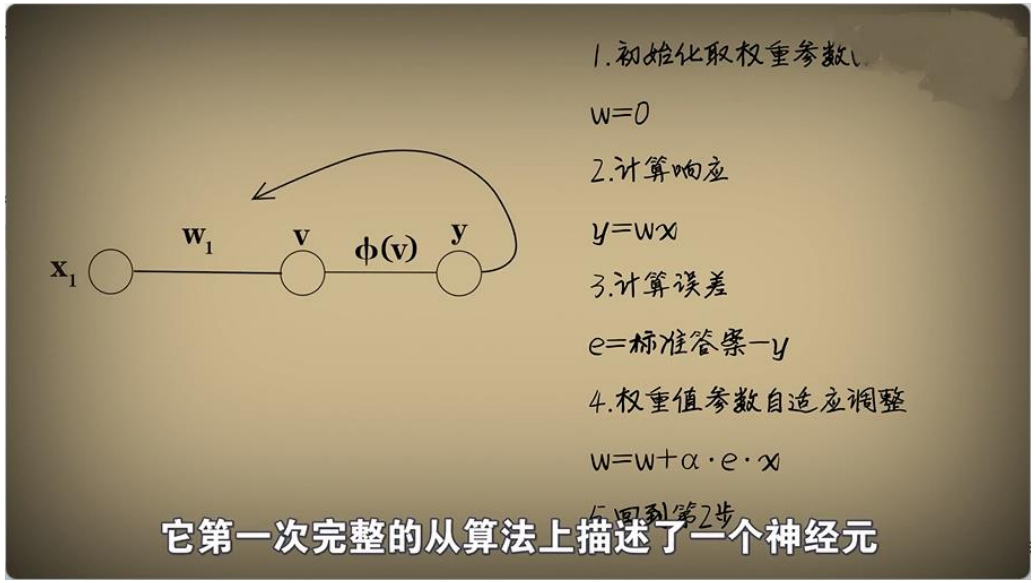
2 权重(Weights - 对应 w1=1.5, w2=1.2, w3=0.6)

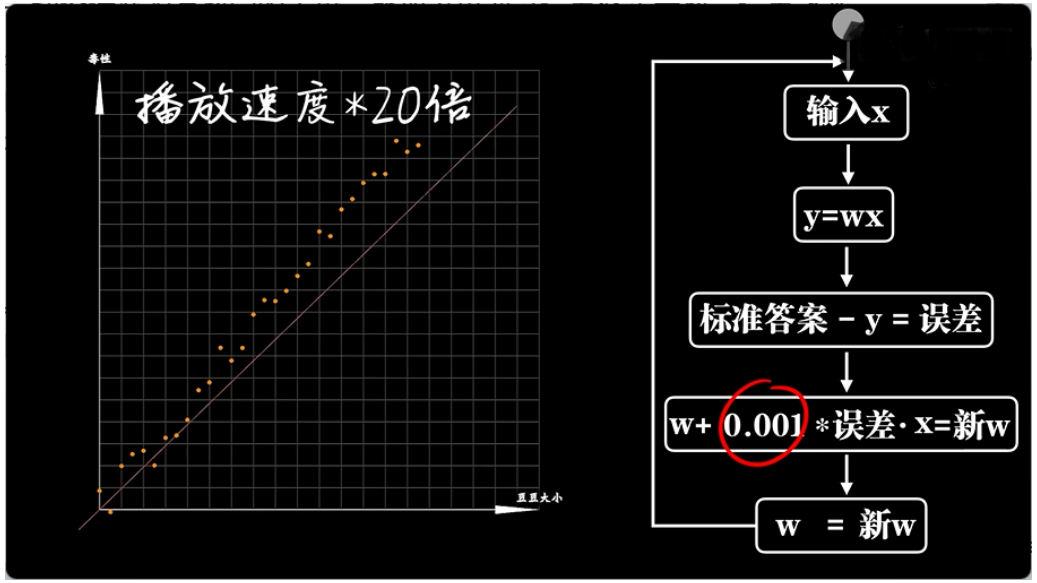
“小蓝”的大脑发现，并非所有信息都同等重要。
例如，经验告诉它，“豆豆”的大小 (x1) 对判断毒性的影响最大，所以它给 x1 一个较大的权重 (比如 w1=1.5)。
颜色 (x2) 可能也有一定影响，权重为 w2=1.2。
气味 (x3) 的影响可能相对较小，权重为 w3=0.6。
权重，代表了不同输入信息的重要性程度。

3 轴突 输出结果

这个“思考单元”会将所有接收到的信息，根据各自的“重要性”（权重）进行加权汇总。计算方式：(x1 * w1) + (x2 * w2) + (x3 * w3)
当然完整的 McCulloch-Pitts 模型中还有一个偏置项b而在一次函数后还会加上一个激活函数用来激活神经元的输出，我们会在后面讨论这些东西，简单起见暂且不表

感知器-计算过程(罗森布拉特)





我们知道 M-P 模型能处理信息，但它的“权重”(w) 和“阈值”需要我们预先设定好。“小蓝”如果每次都要手动调整这些参数才能适应新的“豆豆”或环境，那也太不智能了！
“小蓝”真正需要的是一个能够从经验中自动学习和调整这些参数的“思考单元”。

初始化权重参

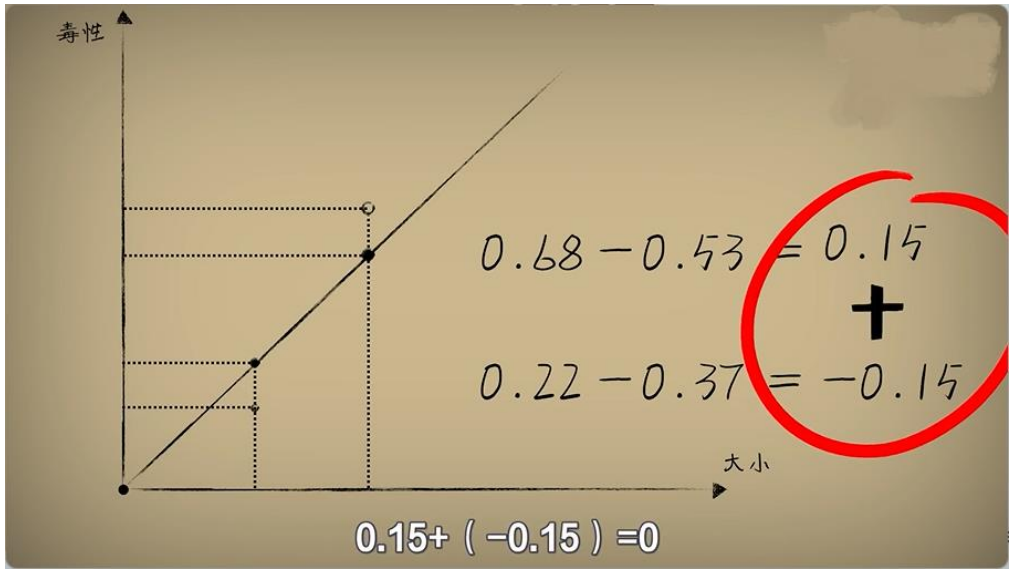
初始化权重参数 (w=0 或随机小值)：
一开始，“小蓝”对这颗新豆豆的特性一无所知，所以它的“判断标准”（权重w）可能是个初始值 (比如图中的 w=0)。
计算响应 (y=w*x)：“小蓝”感知到豆豆的某个特征 x (比如大小)。
它用当前的权重 w 和特征 x 计算出一个初步的判断 y (比如，根据当前 w 和 x，它预测这颗豆豆“无毒”)。

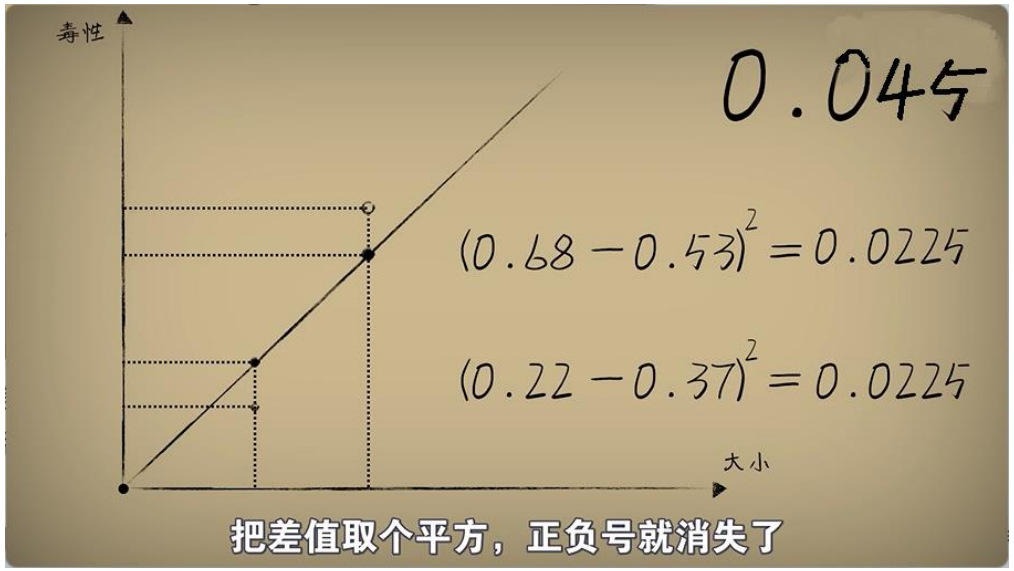
计算误差 (e = 标准答案 - y)

需要告诉“小蓝”它的判断对不对。假设这颗豆豆实际上是“有毒”的。
如果“小蓝”预测“无毒”(y)，而实际“有毒”，那么就产生了误差 e。这个误差告诉“小蓝”它错得多离谱。

权重自适应更新

权重参数自适应这是最关键的一步！“小蓝”会根据这个误差 e 来调整它的权重 w。
α (alpha) 是学习率：控制每次调整的幅度有多大。
e (误差)：如果误差大，调整就多；误差小，调整就少。如果预测正确 (e=0)，就不调整。
x (输入特征)：也影响调整的方向和大小。
核心思想：如果因为某个特征 x 导致了错误，就调整与该特征相关的权重 w，使得下次遇到类似情况时，预测更接近标准答案。
例如：如果“小蓝”把一个大的有毒豆豆错误地判断为无毒，那么它就会增加“大小”这个特征对应的权重，让它下次更倾向于把大的豆豆判断为有毒。





我们看到“小蓝”的感知器能通过计算“**预测**”和“**实际答案**”之间的**差别 (误差 e)** 来调整自己的“**判断标准**”（权重 w）。
但这种简单的“差别”是衡量错误的最佳方式吗？“小蓝”在成长中发现了一些问题...

1 正负抵消的“迷惑行为”

如果“小蓝”一次把微毒豆豆的毒性高估了1个单位 (e = +1)，另一次把中等毒豆豆的毒性低估了1个单位 (e = -1)。
如果简单地把这些“差别”加起来 (+1)，看起来好像“小蓝”没犯错，但实际上它两次都判断失误了！

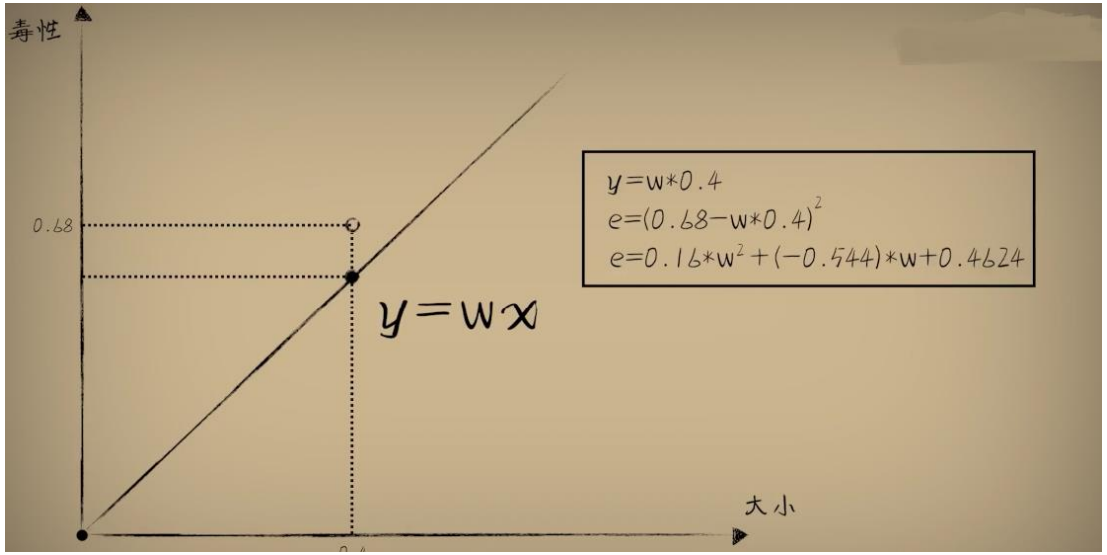
2 对错误的“惩罚”力度不够分明：

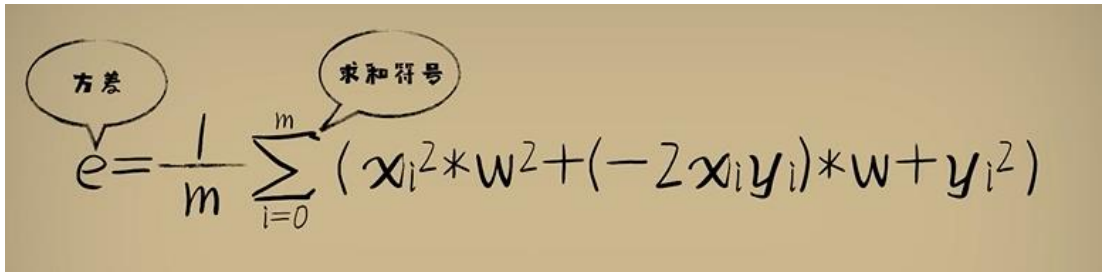
把微毒豆豆错判为无毒，和把剧毒豆豆错判为微毒，对“小蓝”的生存威胁是完全不同的！
简单的“差别”可能无法充分体现这种“错误严重性”的差异。

3 平方误差 —— 错得越离谱，代价越“惨重”！

1. 错误无处遁形 (始终为正)： 无论是高估还是低估，平方后结果都是正数。这样，错误就不会因为正负抵消而被“隐藏”。$(+1)^2$
2. “严惩”重大失误 (放大效应)：如果差别是1，平方误差是 1 ($1^2=1$)。
如果差别是3，平方误差是 9 ($3^2=9$)！
如果差别是10，平方误差是 100 ($10^2=100$)！
平方误差对越大的错误给予越大的“惩罚”值。 这非常符合“小蓝”的生存需求：一次致命的判断失误远比十次微小的判断失误更可怕。
它促使“小蓝”优先去修正那些会导致“灾难性”后果的判断。

知错(认知) – 方差代价函数





我们知道了，“小蓝”通过计算平方误差，能更科学地衡量自己对“豆豆”毒性判断的失误程度。失误越大，平方误差这个“代价”就越大。

“小蓝”的目标是：调整自己的“认知标准”（也就是权重 w），让这个“代价”尽可能小。

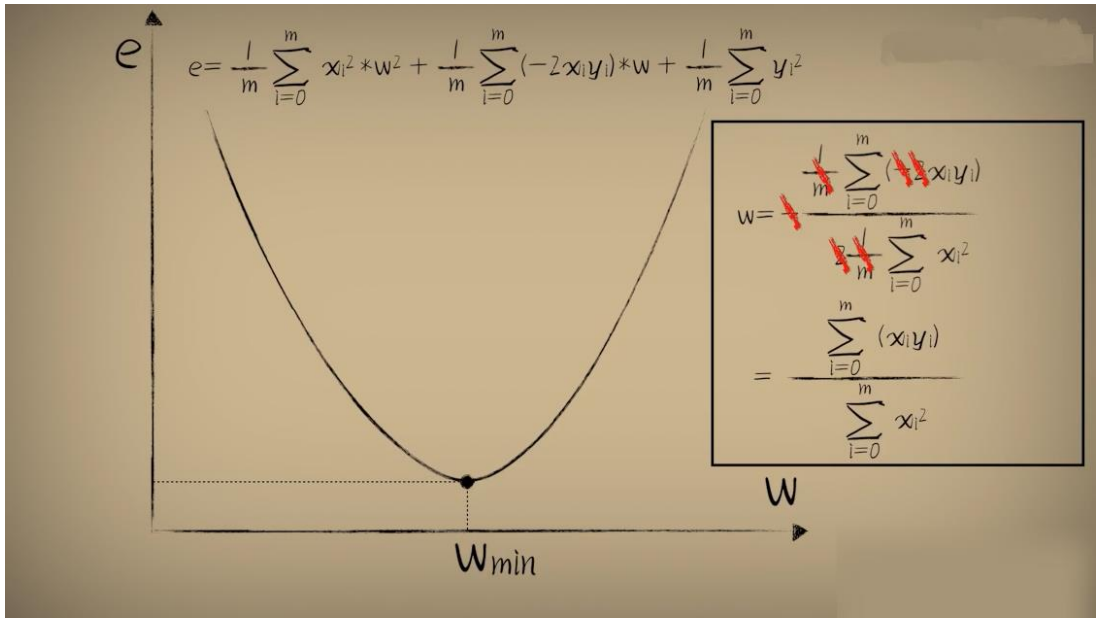
1 聚焦一个“豆豆”

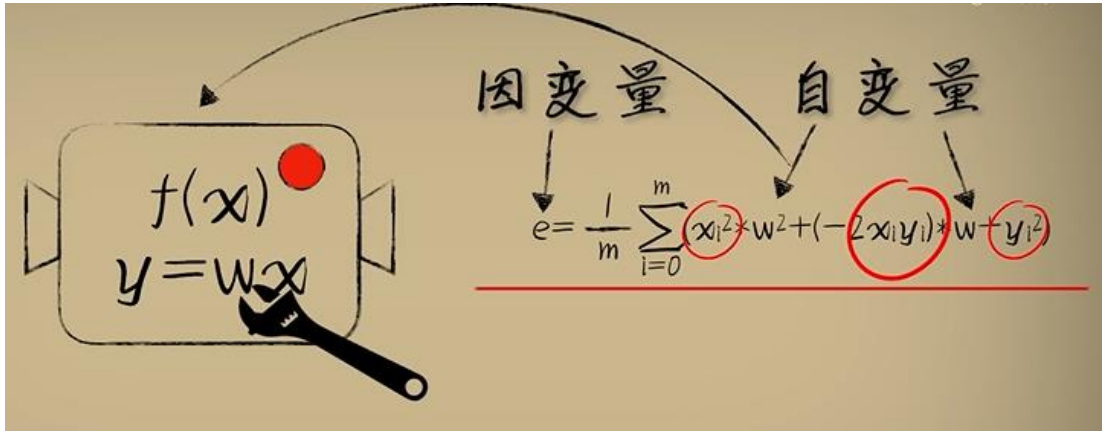
假设“小蓝”遇到了一个特定的“豆豆”。
这个“豆豆”的大小 x 是 0.4 (X轴)。
通过某种方式“小蓝”知道这颗豆豆的真实毒性 y 是 0.68 (Y轴空心圆点)。
“小蓝”当前的“判断模型”很简单：预测毒性 y_预测 = w * x (对应图中的斜线 y=wx)。
这里的 w 是“小蓝”大脑里对“大小”这个特征赋予的“重要性”或“判断系数”。

2 量化“小蓝”的失误：构建损失函数

对于这颗大小为 x=0.4 的豆豆，“小蓝”的预测毒性就是 y_预测 = w * 0.4。
那么，它这次判断的平方误差 (我们称之为损失函数 e，针对这一个豆豆的损失) 是多少呢?
e = (y_真实 - y_预测)² 代入具体数值：e = (0.68 - w * 0.4)²
关键洞察：**损失 e 是权重 w 的函数！(对于这颗特定的豆豆，它的大小 x=0.4 和真实毒性 y_真实=0.68 都是已知的、固定的)。**
唯一能改变损失 e 大小的，就是“小蓝”的“认知标准” - 权重 w!
展开这个函数：
e(w) = (0.68 - 0.4w)²
e(w) = 0.68² - 2 * 0.68 * 0.4w + (0.4w)²
e(w) = 0.4624 - 0.544w + 0.16w²
整理一下顺序，得到：e(w) = 0.16w² - 0.544w + 0.4624
其本质就是 以w为自变量，以e为因变量的一元二次函数

在“误差山谷”中寻找最佳“认知标准 (w)





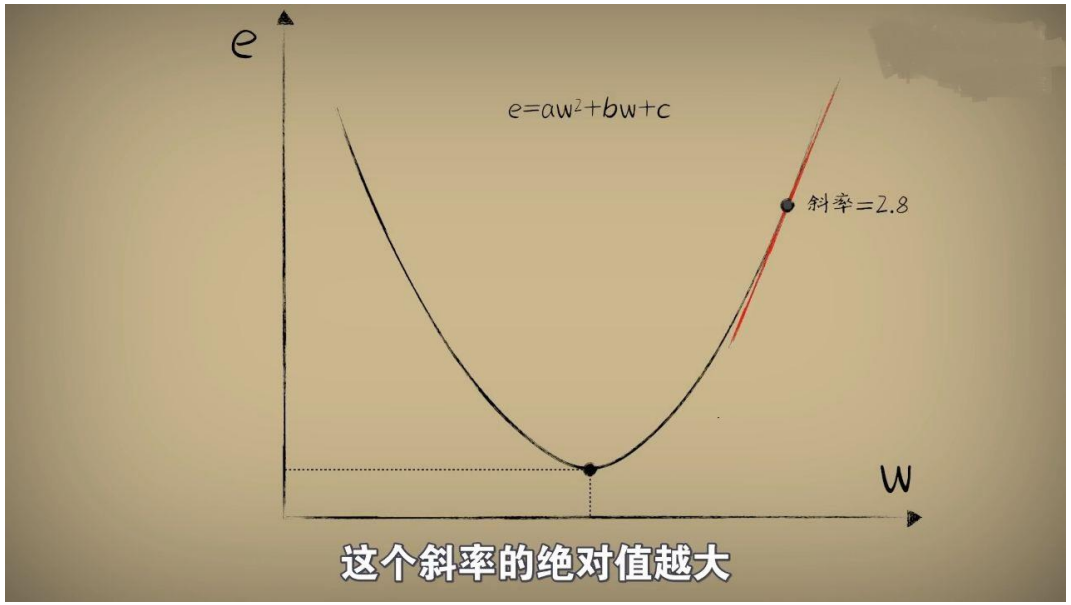
3 损失函数的“庐山真面目”：一个开口向上的“山谷”！

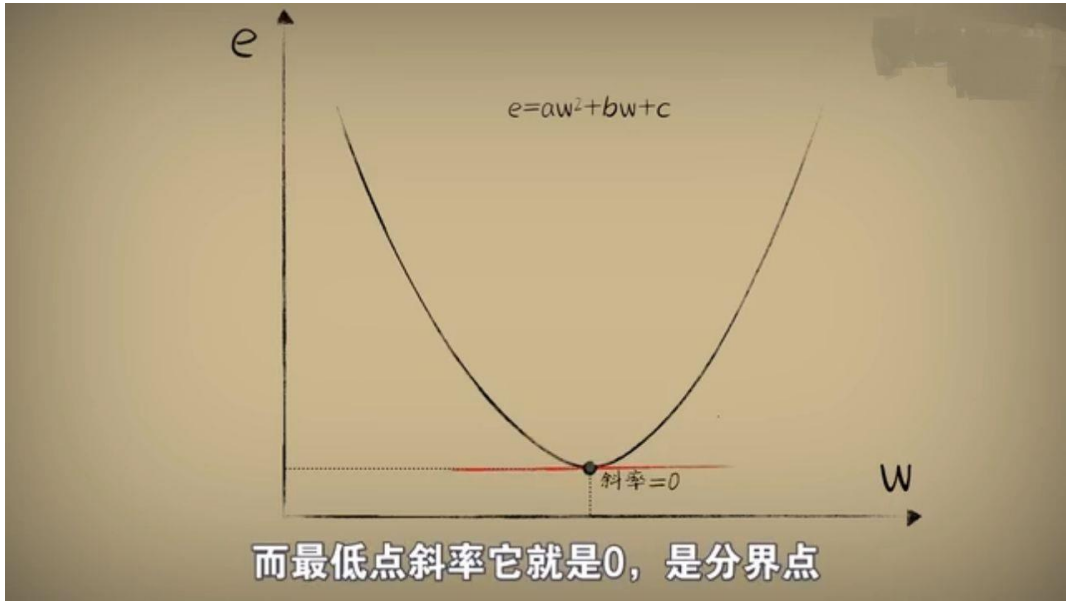
这个关于 w 的函数 e(w) = 0.16w² - 0.544w + 0.4624 是什么形状呢?
这是一个标准的一元二次函数 (形如 **aw² + bw + c**)!
因为 w² 项的系数 0.16 是一个正数，所以这个函数的图像是一个开口向上的抛物线——就像一个“山谷”的形状。
这对“小蓝”意味着什么?
这个“山谷”的最低点，对应着一个特定的权重 w 值。
在这个最低点，损失函数 e(w) 的值最小，也就是说，“小蓝”对这颗特定豆豆的判断误差最小，预测最准确!

4 小蓝 的目标：找到“误差山谷”的“谷底”

现在“小蓝”的**任务**变得清晰了：**它需要找到那个能让损失 e(w) 最小的权重 w，也就是找到这个“误差山谷”的“谷底”**。
（铺垫） 对于这一个豆豆，我们或许能用数学方法直接解出谷底。但如果“小蓝”遇到了成千上万颗不同的豆豆，每颗豆豆都有自己的“误差山谷”，整体的“误差地貌”会变得非常复杂。
“小蓝”需要一种更通用的方法，能够一步步地向着“谷底”前进。这就是我们接下来要讲的“秘密武器”——**梯度下降**！。

“小蓝”的“寻底”神技：微积分揭示“误差山谷”的秘密





我们知道了“小蓝”的整体判断失误（方差代价函数 e(w)）与它的“判断标准”（权重 w）构成了一个开口向上的“误差山谷”。
“小蓝”的目标明确：到达山谷的最低点，因为那里的 w 能让它的判断最准确。但问题是，怎么才能精确地找到这个最低点呢?

1 数学家的“探照灯”：微积分与导数

数学家们给了“小蓝”一个强大的工具，就像一盏能照亮山谷地形的“探照灯”- 微积分，尤其是其中的求导 (Differentiation)。
导数的魔力： 一个函数在某一点的导数，代表了函数曲线在该点的瞬时变化率，也就是切线的斜率 (slope)。

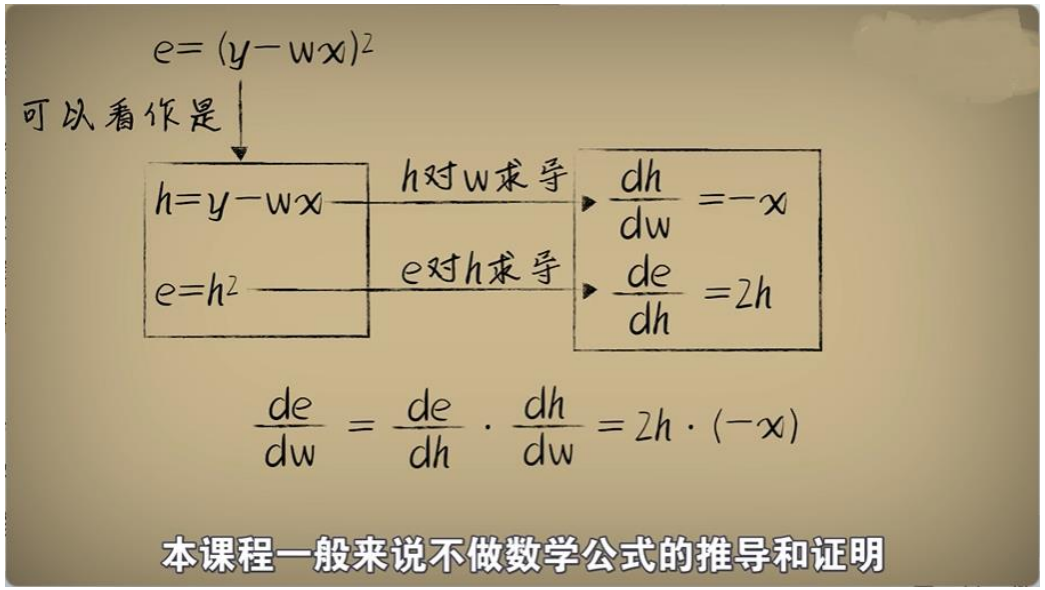
2 “山谷最低点”的数学特征：斜率为零!

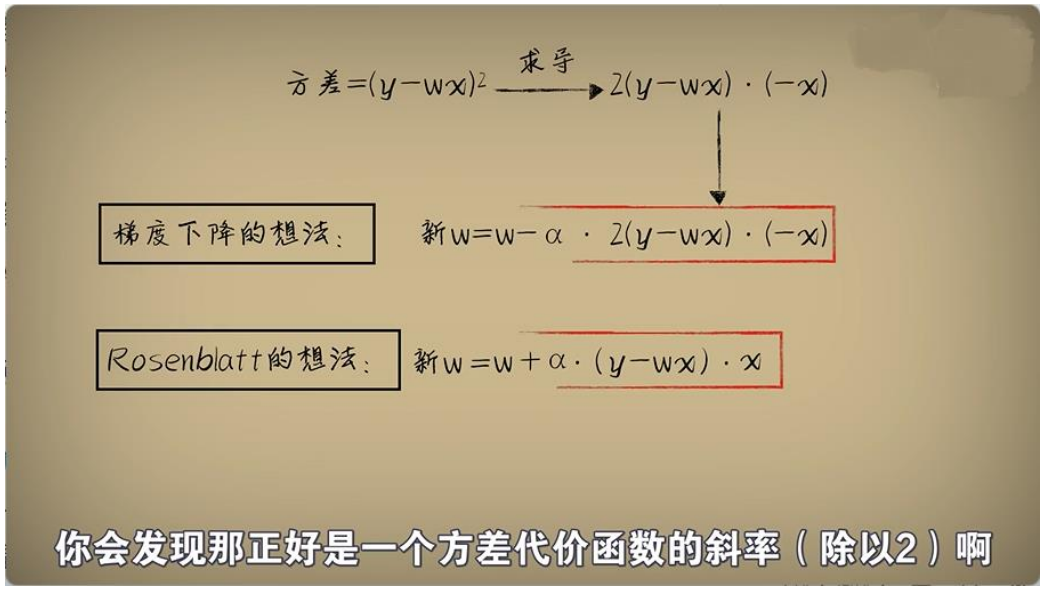
对于一个光滑的“误差山谷”（代价函数曲线）：
在山谷的最低点，曲线是平坦的，那里的切线是水平的，所以它的斜率必定为零!（同理，在“山峰”的最高点，斜率也为零。）
这意味着，如果我们能找到代价函数 e(w) 的导数 dJ/dw，并令这个导数等于零，解出来的 w 值，就有可能是我们梦寐以求的“谷底”对应的最佳权重!

3 对平方误差损失函数求导

让我们以之前单个“豆豆”的平方误差（方差）为例，看看如何计算斜率。
损失函数 $e(w) = (y_{真实} - w*x)^2$
所以，损失函数在任意权重 w 点的斜率（导数）是：$-2x(y - wx)$
这个导数值告诉我们，在当前 w 下，“误差山谷”有多陡，以及是向上还是向下倾斜。

链式法则求导：分解复杂函数





数学家们提供了一个巧妙的方法
链式法则 (Chain Rule)，专门用来处理这种“函数套函数”（复合函数）的情况。

1 “庖丁解牛”：分解复杂函数

我们可以把 **e = (y - wx)²** 这个函数“拆解”成两个更简单的部分：
第一部分： 内部函数，令 **h = y - wx**
这里的 h 可以理解为“小蓝”的预测值 wx 与真实值 y 之间的“直接差距”。
第二部分： 外部函数，那么误差 e 就变成了 **e = h²**
这里的 e 就是这个“直接差距 h”的平方。
现在，e 不再直接是 w 的函数，而是先通过 h 间接与 w 关联：w 影响 h，h 再影响 e。

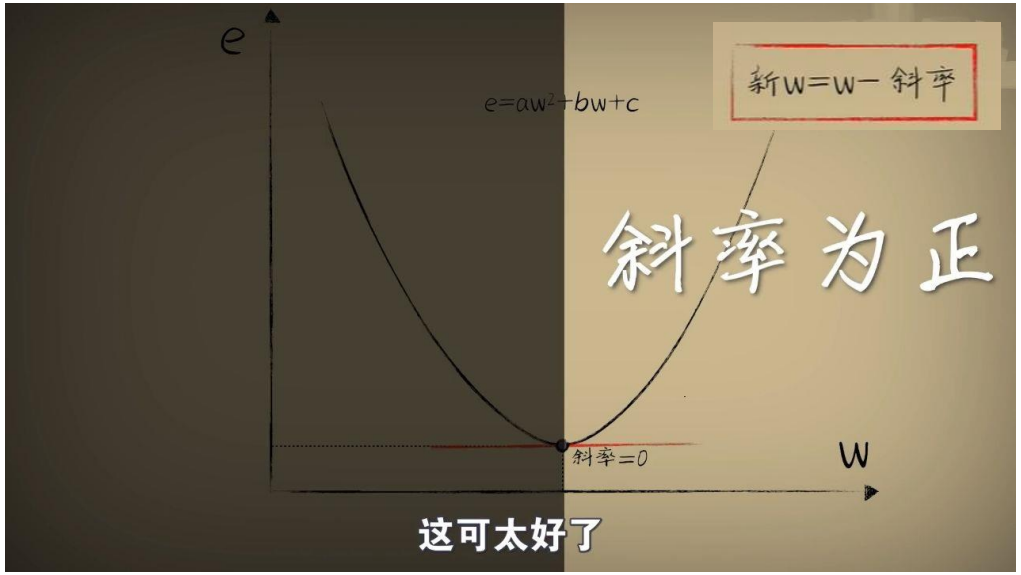
2 分而治之：分别求导

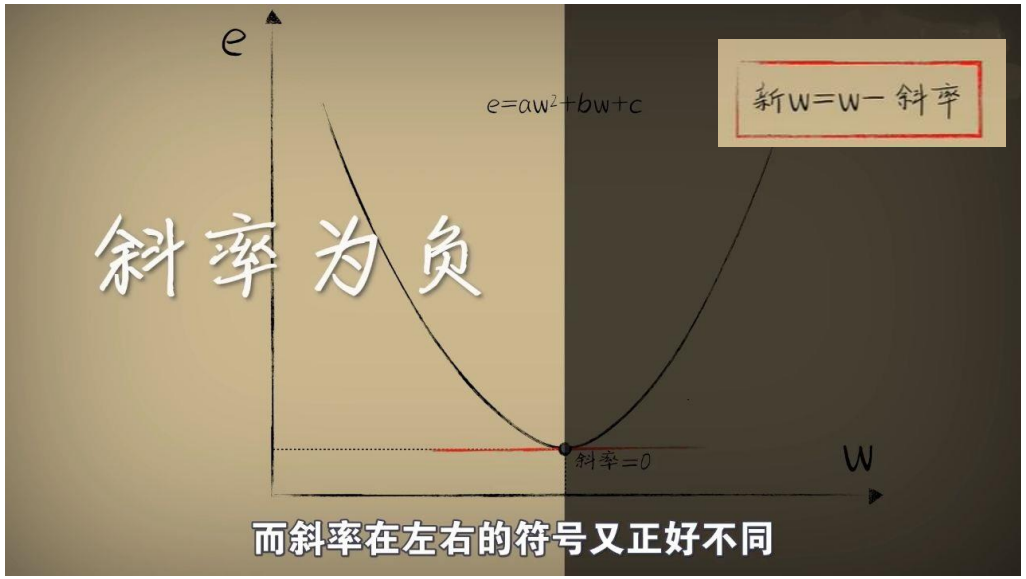
h 对 w 求导 (dh/dw)：
我们看 h = y - wx。当 w 变化时，h 如何变化?
因为 y 和 x 在这里是已知的常数（对于某个特定的豆豆），所以 **dh/dw = -x。**
这意味着，如果权重 w 增加1个单位，那么“直接差距” h 就会减少 x 个单位（或者说，向 -x 方向变化）。
e 对 h 求导 (de/dh)：
我们看 e = h²。当 h 变化时，e 如何变化?
de/dh = 2h。
这意味着，如果“直接差距” h 增加1个单位，平方误差 e 会增加 2h 个单位（变化率是 2h）。

3 “链”起来！— 链式法则的魔力

链式法则告诉我们：要求 e 对 w 的总导数，我们只需要把上面两个“分段”的导数乘起来就行了！
de/dw = (de/dh) * (dh/dw)
代入我们刚才算出的结果：
de/dw = (2h) * (-x)
再把 h = y - wx 代回去：
de/dw = 2(y - wx)(-x)

梯度下降(指引“下山”之路)





虽然我们知道，“小蓝”的方差代价函数 e(w) 形成了一个“误差山谷”，谷底对应着最佳权重 w。我们也知道了，代价函数在任意点 w 的导数 de/dw (即斜率) 告诉我们山谷在该点的陡峭程度和方向。现在的问题是：“小蓝”如何利用这个“斜率”信息，一步一步地、智能地走到谷底呢?

1 梯度下降的核心思想：“跟着坡度反着走，总能下山坡”

想象“小蓝”站在“误差山谷”的某个山坡上，它想尽快下到谷底。
最直观的方法是什么？— 找到当前位置最陡峭的下坡方向，然后朝着那个方向迈出一步。
梯度 (Gradient)： 在数学上，函数在某一点的梯度（对于一元函数就是导数 de/dw）指向的是函数值增长最快的方向。
梯度下降 (Gradient Descent)： 因此，如果我们想让函数值（误差）下降最快，就应该沿着梯度的反方向前进!

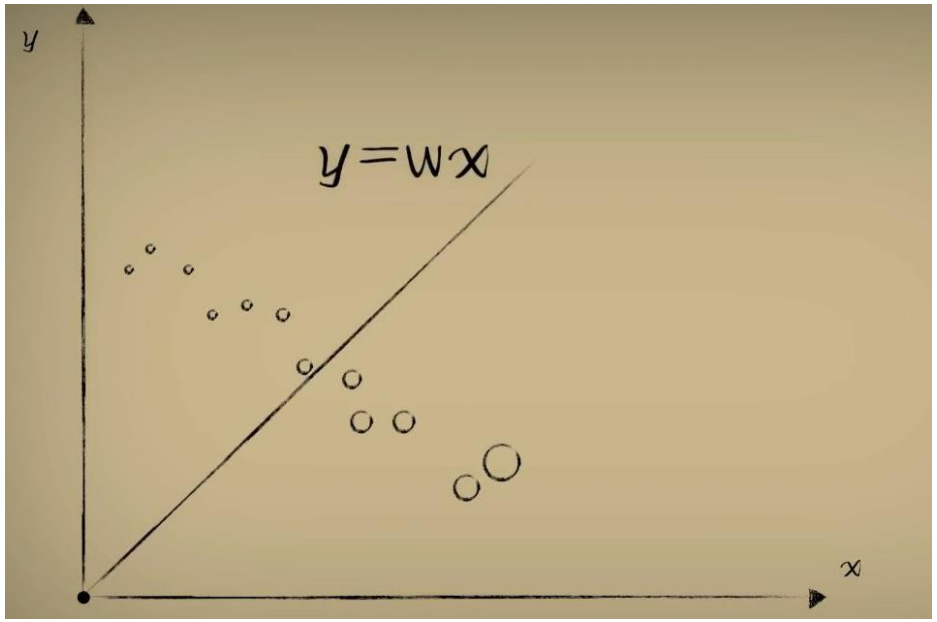
2 “小蓝”的导航指令：梯度下降更新规则

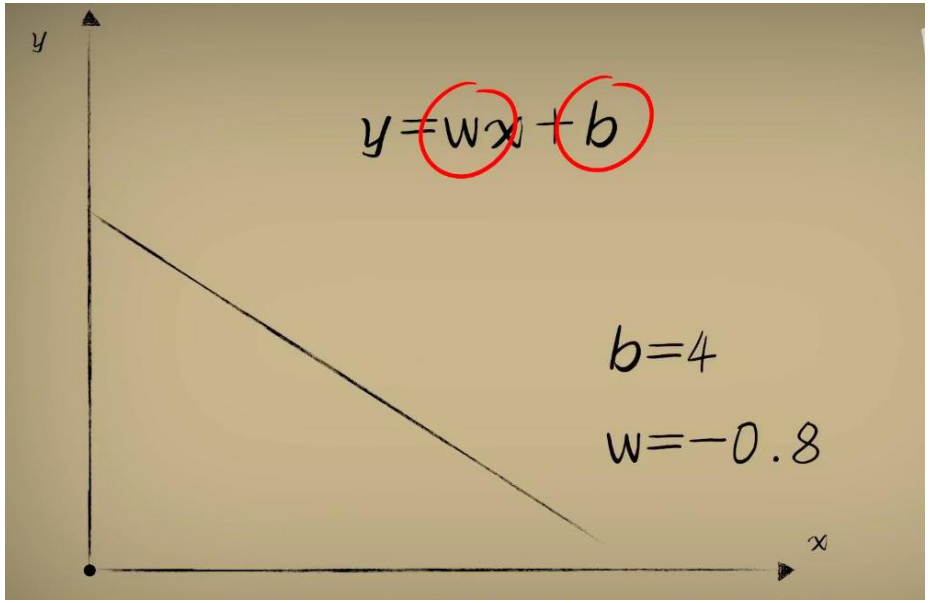
新w = 旧w - α * (de/dw)。新w: 更新后的权重。旧w: 当前的权重。
α (alpha): 学习率 (Learning Rate)，一个小正数，控制“小蓝”每一步迈多大。
(de/dw): 代价函数 e(w) 对权重 w 的导数（梯度/斜率）。
这个公式告诉“小蓝”：**根据当前位置的坡度（导数），决定下一步往哪个方向调整 w，以及调整多少。**
情景：小蓝当前所处的权重 w 位于“误差山谷”最低点的右侧。
观察：此时，代价函数曲线的斜率 de/dw 为正
导航指令生效：
新w = 旧w - α * (一个正数)
因为 α 也是正数，所以 α * (正数) 还是正数。
新w = 旧w - (某个正值)，这意味着 新w 会比 旧w 小。
结果： 权重 w 向左移动，朝着谷底前进!

3 “小蓝”的持续学习：迭代的力量

梯度下降是一个迭代 (Iterative) 的过程。
“小蓝”会不断地重复“计算当前坡度 -> 更新权重 w”这个过程。
随着一步步的迭代，w 会越来越接近谷底的最佳值，代价函数 e(w) 的值也会越来越小。
当坡度接近于零，或者 w 的变化非常微小时，我们就可以认为“小蓝”已经找到了足够好的“认知标准”，学习过程可以停止。

“小蓝”的进化瓶颈：当直线必须穿过原点...引入“偏置b”打破僵局！





我们已经为“小蓝”装备了强大的梯度下降算法，它能有效地帮助“小蓝”在“误差山谷”中找到最佳的“判断系数 w”。
但是，“小蓝”的预测模型 y_预测 = w * x (预测毒性 = w * 豆豆大小) 本身是不是有点太简单了？如果仔细想想，会发现这个模型存在一些“硬伤”。

1 “必须过原点”的尴尬：当现实不按“套路”出牌

y = w*x 这个函数无论 w 怎么变，它画出来的直线永远都要经过坐标原点 (0,0)。

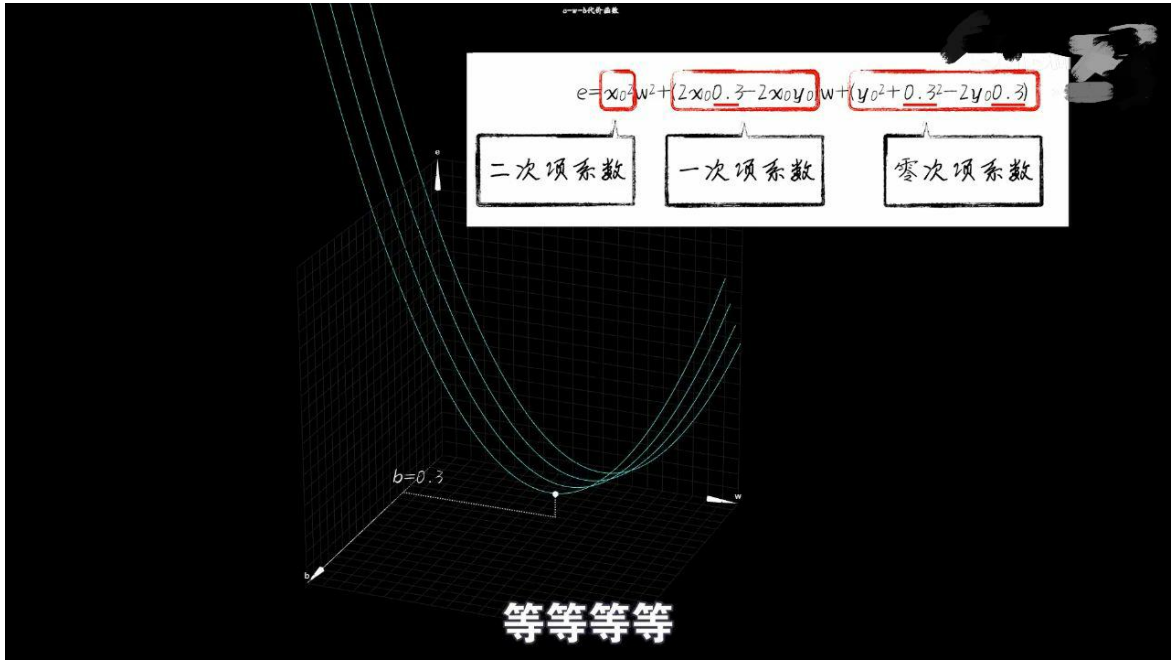
场景一：豆豆世界的“最低消费”
想象在另一片海域，“小蓝”发现，那里的“豆豆”就算体积非常非常小 (接近0)，也还是有那么一点点基础毒性。或者，特别小的豆豆根本就不存在，有记录的豆豆都是从一定大小起步，并且它们对应的毒性并不从0开始。
此时，如果“小蓝”还用 y=w*x 去拟合，它无论怎么调整 w，都无法画出一条既能描述整体趋势，又能准确反映“小豆豆也有基础毒性”的直线。它的预测线总是被“强按”在原点上。

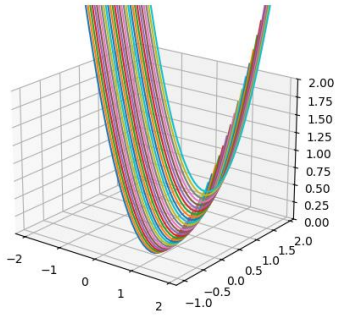
场景二：反常的“越大越安全”
更极端的情况，如果那里的“豆豆”是越大，毒性反而越低！y=wx 确实可以通过负的 w 来表示这种负相关。
但如果这种“越大越安全”的规律，其对应的直线在Y轴上的截距（比如，非常大的豆豆毒性趋近于某个正值，而不是0）不为零，那么 y=w*x 依然会很“憋屈”。

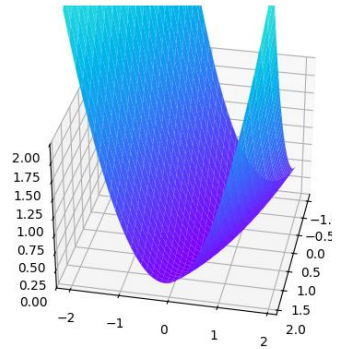
2 “小蓝”的装备升级：引入“偏置项b”— 解锁自由平移！

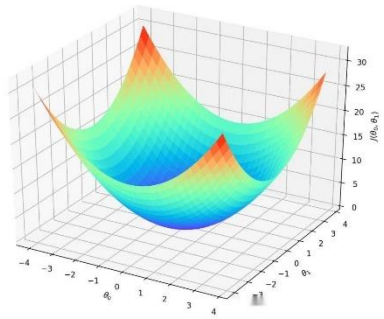
为了让“小蓝”的预测模型更强大、更灵活，科学家们为它引入了一个新的参数——偏置项 b。
“小蓝”的新预测函数变成了：y_预测 = **wx + b**
这个 b 有什么用呢？
它允许预测直线在 Y轴上自由上下平移！不再被强制绑定在原点。
现在，“小蓝”不仅能学习“豆豆大小”对“毒性”的影响程度 (w)，还能学习到一个“基础毒性值”或者说“整体偏移量” (b)。

从“误差山谷”到三维“代价大碗”









3 坐标系升维：引入“偏置b”后的新局面

现在，“小蓝”的预测模型进化了，加入了“偏置项b”：y_预测 = w * x + b。
这意味着，“小蓝”的判断结果现在同时受到两个参数的影响：权重 w (决定直线的斜率) 和 偏置 b (决定直线在Y轴上的截距)。
因此，衡量“小蓝”总体判断失误的代价函数也必须同时考虑 w 和 b。我们表示为 e(w, b)。
代价函数的**坐标系由二维(y、w)升级到了三维(y、w、b)**

e(w, b) = (y_真实 - (w * x + b))²

4 从“线”到“面”：代价函数 e(w,b) 的三维形态！

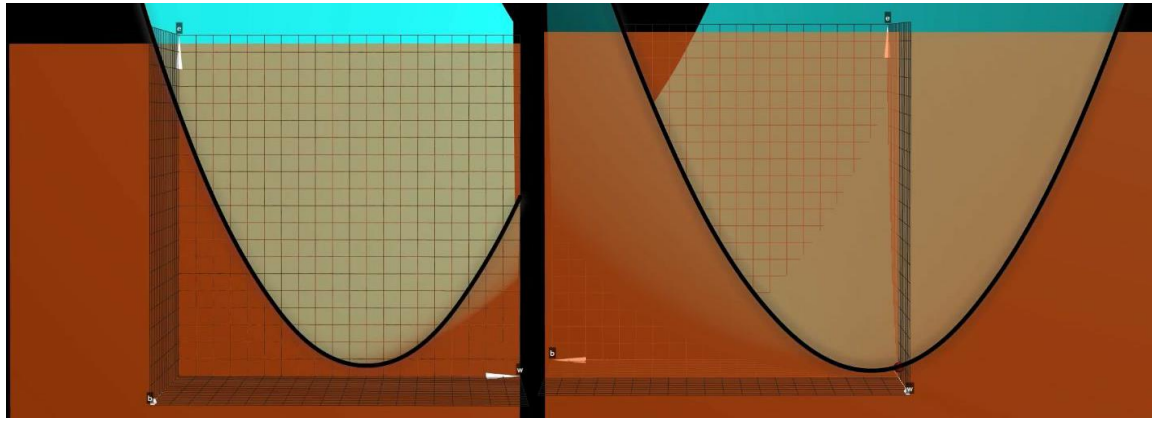
如何理解这个三维曲面?
想象一个水平的平面：一个轴代表所有可能的 w 值，另一个轴代表所有可能的 b 值。
对于这个平面上的每一个点 (每一对 w, b 组合)，“小蓝”都会计算出一个对应的总代价（总误差）e(w,b)。
我们将这个 e(w,b) 值作为高度，画在这个 (w,b) 点的正上方。
把所有这些高度点连接起来，就形成了如图所示的一个连续的、光滑的三维曲面。

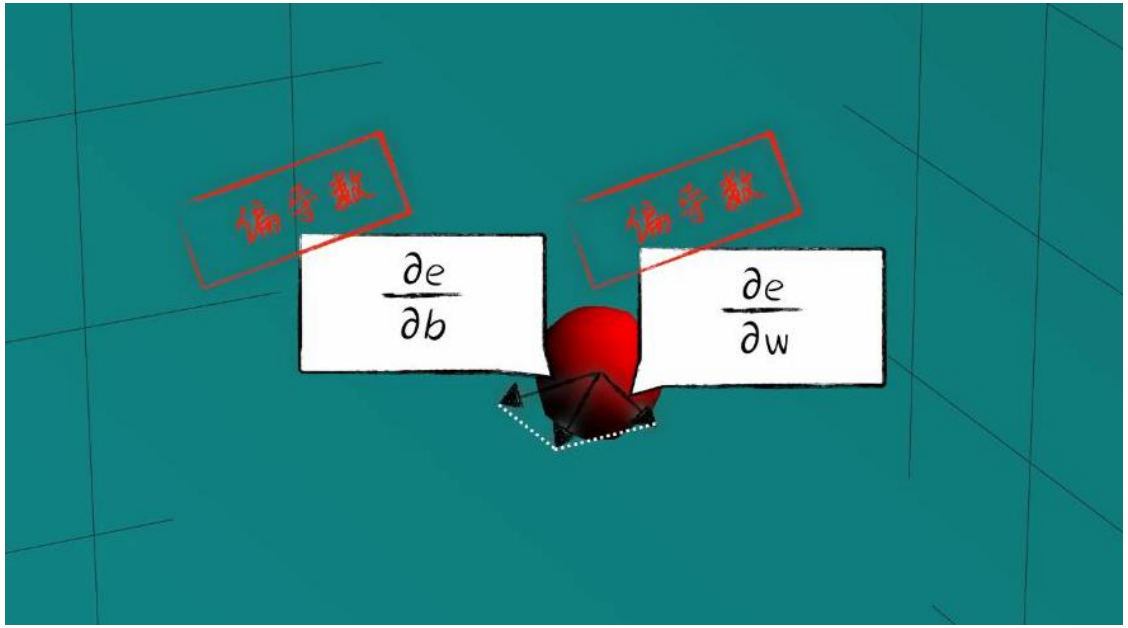
这个曲面通常是一个开口向上的形状，像一个巨大的“碗”或者一个被拉伸的“U型槽”。
图中可以看出“扁的几乎看不出来是个碗状”，这可能因为在某个方向上变化比较平缓。
单个样本的曲面实际上是一个U型的特殊的碗。
这点很重要：单个样本产生的误差曲面可能更像一个沿着某个方向无限延伸的“U型槽”；
但当我们将所有豆豆样本的误差叠加（平均）起来后，总的代价函数 e(w,b) 通常会形成一个更接近“碗状”的、在各个方向上都有底部约束的形态。

5 为什么会形成一个开口向上的“大碗”？

1. 平方误差的“功劳”： 因为我们的代价函数是基于平方误差 (y_真实 - y_预测)²。平方操作确保了误差值永远是非负的（大于等于零）。所以，这个代价曲面永远不会掉到“海平面”以下。
2. 二次函数的本质： 当我们把 y_预测 = w*x + b 代入平方误差并展开后，代价函数 e(w,b) 会包含 w²、b²、w*b、w、b 以及常数项。这种关于 w 和 b 的二次型（quadratic form）的数学特性，决定了其图像在三维空间中呈现为一个抛物面 (paraboloid)，也就是我们所说的“碗状”。
3. 开口向上： 由于 w² 和 b² 项的系数通常是正的（源于平方项），这保证了“碗口”是朝上的，意味着它有一个唯一的最低点。

在三维“大碗”中寻找唯一的“碗底”





“小蓝”的目标依然是找到这个“碗”的最低点，因为那里对应着最佳的 (w, b) 组合。但现在它不能只沿着 w 的方向找了，它有两个“旋钮” (w 和 b) 需要同时调整。

1 “下山”策略升级：从“单行道”到“平面导航”— 偏导数

想象分别在w、b方向切一刀，各自形成一个切面

当只有一个参数 w 时，我们用导数 de/dw (斜率) 来指引w切面方向。
当只有一个参数 b 时，我们用导数 de/db (斜率) 来指引b切面方向。

现在有两个参数 w 和 b，我们需要分别知道代价函数 e 对它们各自的敏感程度：
∂e/∂w (e对w的偏导数)： 当 b 保持不变时，w 的微小变化会引起 e 多大的变化？这告诉我们“碗”在 w 方向上的坡度。

∂e/∂b (e对b的偏导数)：
当 w 保持不变时，b 的微小变化会引起 J 多大的变化？这告诉我们“碗”在 b 方向上的坡度。

这两个偏导数共同构成了代价函数在 (w,b) 点的梯度 (Gradient)，它是一个向量 (∂e/∂w, ∂e/∂b)，指向“碗壁”上坡最陡的方向。你可以理解为一个指向“碗底”的合向量。

2 梯度下降“双引擎”启动：同时优化 w 和 b

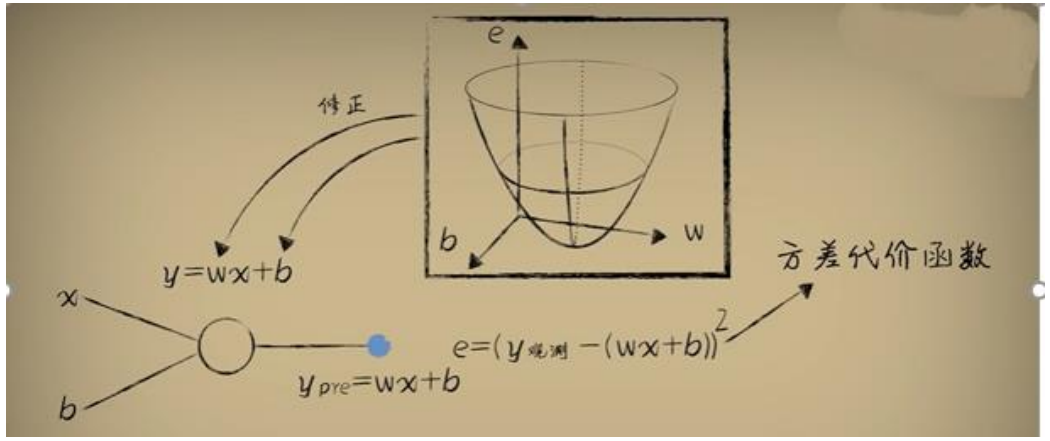
在计算时候，小蓝 的梯度下降算法需要同时更新 w 和 b：

新w = 旧w - α * (∂e/∂w)
新b = 旧b - α * (∂e/∂b)

这里的 α 仍然是学习率。这意味着，“小蓝”会根据“碗”在 w 方向的坡度和在 b 方向的坡度，分别调整 w 和 b，努力使总误差 e(w,b) 下降。

在“碗” 中的表现为，按照合两个斜率形成的 合向量 方向移动。

前向感知世界，反向修正认知(“小蓝”的学习心法)



梯度下降算法告诉我们，需要计算代价函数对 w 和 b 的偏导数（即梯度 (∂J/∂w, ∂J/∂b)），
然后沿着梯度的反方向更新 w 和 b。
那么，小蓝 具体是如何得到这些信息并完成学习的呢？答案就在于不断循环的“前向传播”与“反向传播”。

1 第一步：前向传播 —“小蓝”的感知与预测

定义： 前向传播是指数据从输入层开始，经过神经网络（在小蓝的简单模型中，就是一个神经元）的计算，最终得到输出（预测值）的过程。
小蓝的体验：
1. 接收信息：小蓝遇到一颗“豆豆”，感知到它的特征 x (比如大小)。同时，它内部有一个当前的偏置参数 b。
2. 处理信息（模型计算）： 它将这些信息输入到自己的“大脑模型” y_预测 = w*x + b 中，利用当前的权重 w 和偏置 b 进行计算。
3. 做出预测： 得到一个对这颗“豆豆”毒性的预测值 y_预测。
4. 衡量失误（计算误差）： 小蓝将自己的预测 y_预测 与这颗豆豆的真实毒性 y_真实 (图中 y观测) 进行比较，计算出两者之间的差距，通常是平方误差 e = (y_真实 - y_预测)²。这个误差 e 就是“小蓝”这次预测“错”了多少的量化。
总结： 前向传播是“小蓝”利用现有知识（w 和 b）去理解世界、做出判断，并得到反馈（误差）的过程。

2 第二步：反向传播 — “小蓝”的反思与归因

定义：指当小蓝的预测与真实答案产生偏差（误差）后，这个‘误差信息’会从最终结果开始，像信号一样反向传递，帮助小蓝计算出它的每一个‘决策关键点’（即参数 w 和 b）各自对当前误差的‘影响程度’（即它们的梯度），并指明如何调整这些关键点才能在下一次做得更好。

小蓝 的“灵魂拷问”：
刚才的误差 e 这么大（或这么小），到底是我的“判断系数 w”出了问题，还是我的“基础认知 b”有偏差？或者两者都有责任？各自要负多大责任呢？
核心任务：计算梯度 (∂e/∂w, ∂e/∂b)

反向传播的核心目的就是精确计算出代价函数 e对每一个参数 (w 和 b) 的偏导数。
这些偏导数告诉我们，如果轻微调整 w（或 b），总误差 J 会如何变化。这正是梯度下降所需要的“方向指引”。
（回顾H11的链式法则） 对于更复杂的、有多层神经元的网络，链式法则是实现反向传播计算梯度的关键数学工具。它能有效地将输出层的误差逐层分解，传递到网络中的每一个参数。

总结： 反向传播是“小蓝”根据预测的失误，深刻反思自己的“知识体系”（参数 w 和 b），找出导致错误的关键因素及其影响程度的过程。

3 学习的闭环：
感知 -> 预测 -> 误差 -> 反思 -> 修正 -> 再感知...

小蓝 的学习是一个持续不断的循环：

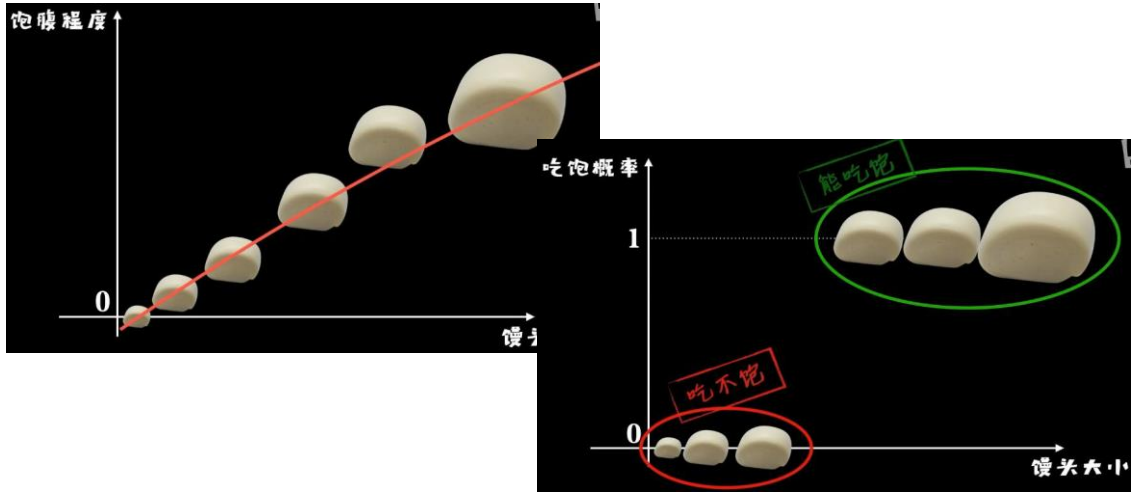
前向传播： 用当前的 w, b 预测，得到误差。
反向传播： 根据误差计算出梯度 (∂e/∂w, ∂e/∂b)。

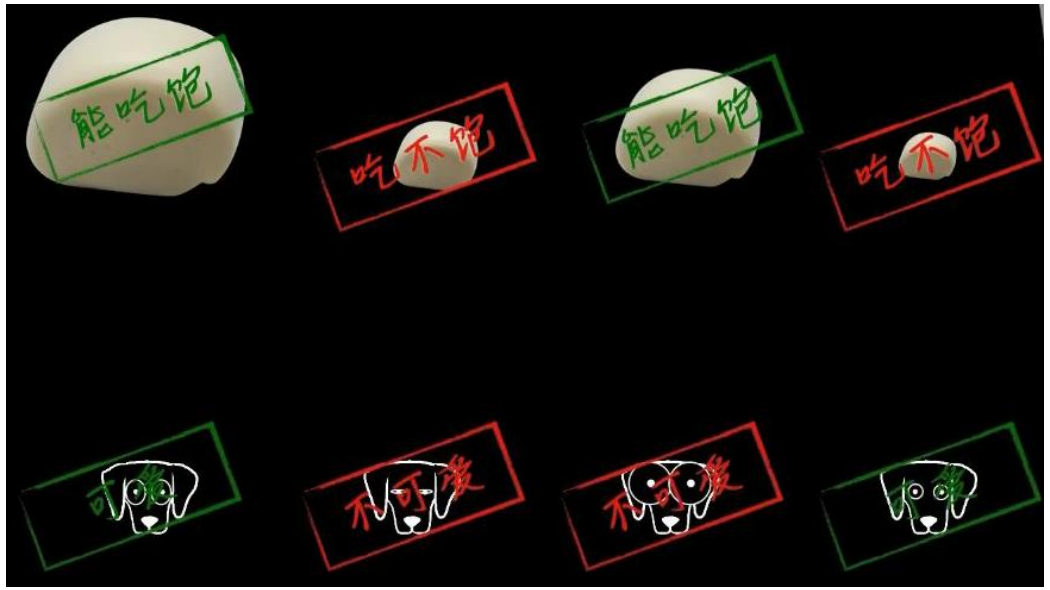
参数更新 (梯度下降)： 使用梯度和学习率 α 更新 w 和 b：
w = w - α * (∂e/∂w)
b = b - α * (∂e/∂b)

小蓝 带着**更新后的 w 和 b**，再次面对新的（或同一批）“豆豆”，开始新一轮的前向传播...**如此往复。**

每一次循环，都期望让 w 和 b 更接近“代价大碗”的碗底，使“小蓝”的预测越来越准确。

人类的“盖章式”思维：我们常常做的是分类





人类的很多判断和决策，本质上是离散的、分类的 (Categorical)。
我们喜欢给事物“盖章”，将它们归入不同的类别。
我们思考问题的方式往往往都是离散的分类，而不是精确的拟合

1 “小蓝”的“精准打击”能力回顾

到目前为止，小蓝 通过学习权重 w 和偏置 b，已经能够对“豆豆”的毒性进行预测了。

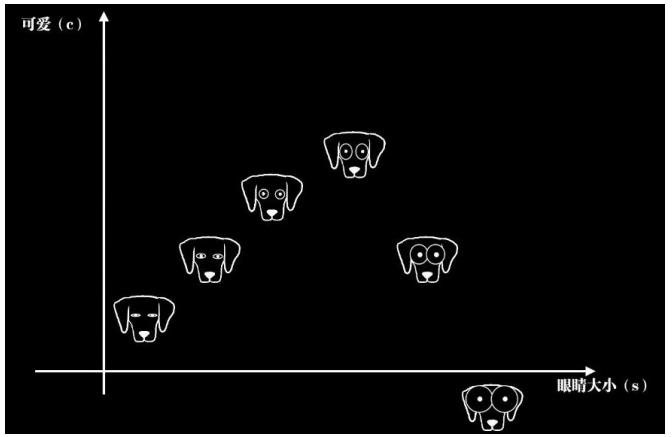
它的输出 y_预测 = w*x + b 是一个**连续**的数值，这在某些情况下（比如预测房价、温度）非常有用，我们称之为**回归 (Regression)** 问题。

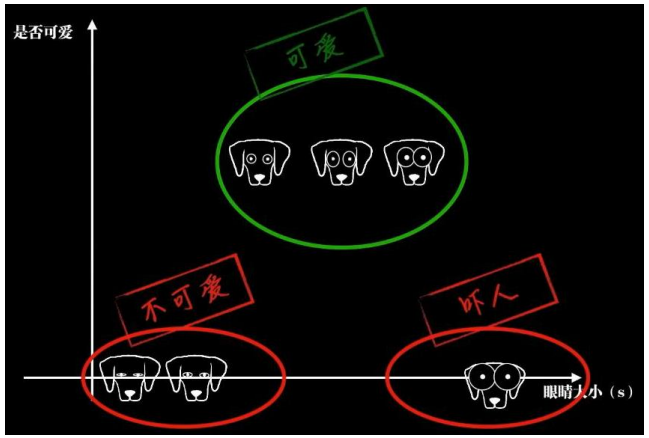
2 人类的“盖章式”思维

看到一个馒头，我们通常不会想“这个馒头的可食用指数是0.87”，而是直接判断“能吃饱”或者“吃不饱”。

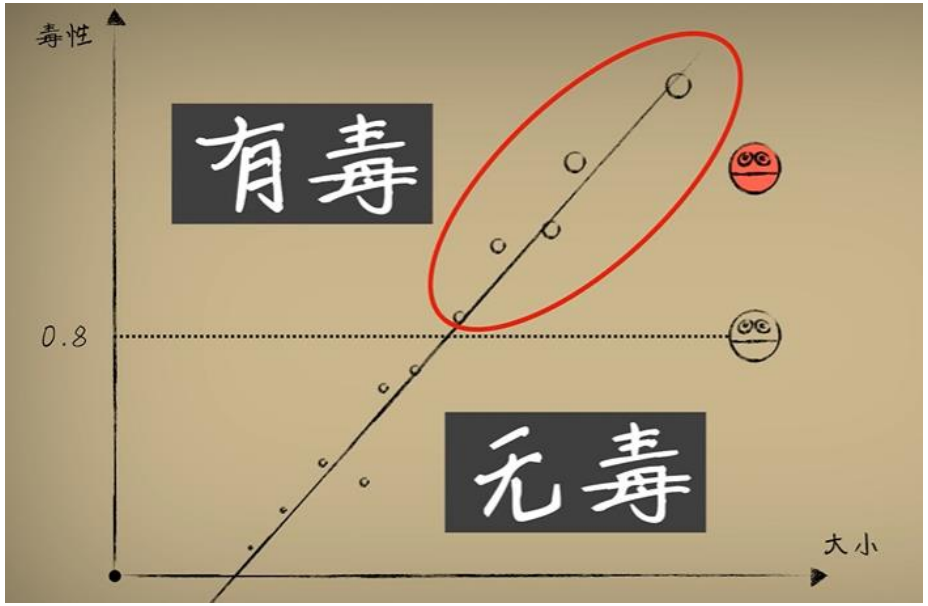
看到一只小狗，我们可能判断它“可爱”或者“不可爱”（如图片下半部分），而不是给它的可爱度打一个精确到小数点后三位的分数。

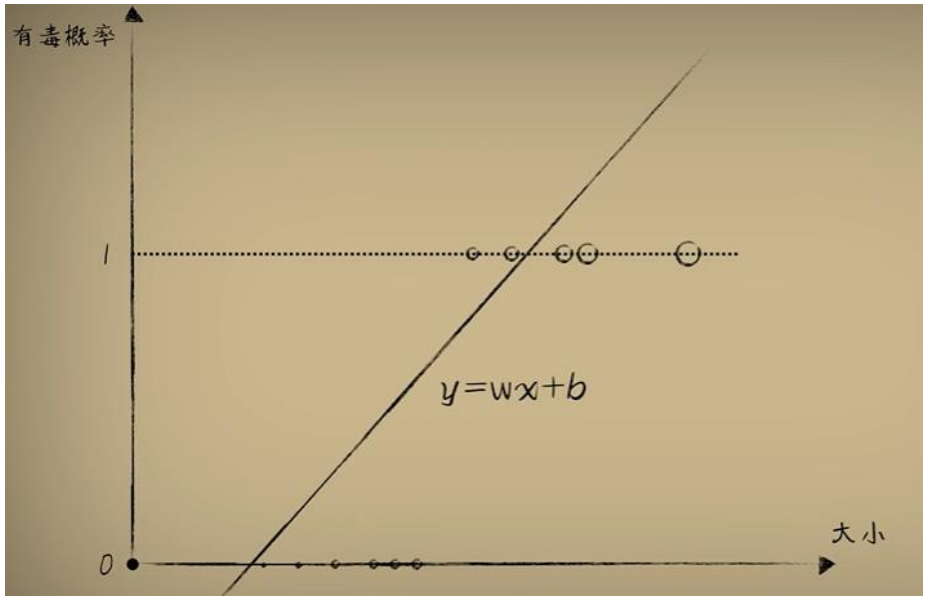
人类的很多判断和决策，本质上是离散的、分类的。 我们喜欢给事物“盖章”，将它们归入不同的类别。





“小蓝”的新挑战：豆豆“有毒”或“无毒”？线性模型的窘境





很多时候我们更关心的是分类判断，比如一颗豆豆到底是“有毒”还是“无毒”。
我们不再需要“小蓝”给出一个精确的毒性值（比如0.73），而是希望它能给出一个明确的类别判断。
为了方便模型学习，我们可以用数字来代表类别：比如，0 代表“无毒”，1 代表“有毒”。

1 旧模型新用：y=w*x + b 能胜任分类任务吗？

横轴 (大小 x)： 代表豆豆的大小。
纵轴 (有毒概率/类别)： 标记了0和1两个关键位置。

数据点： 我们可以看到，实际的豆豆数据点（那些小圆圈）主要集中在两个水平线上：
一些小豆豆（x值较小）对应的是“无毒”，它们聚集在纵坐标为 0 的附近。
一些大豆豆（x值较大）对应的是“有毒”，它们聚集在纵坐标为 1 的附近。
直线 y=w*x + b： 这是“小蓝”当前学会的线性模型，它试图穿过这些数据点。
我们之前的神经元预测函数模型是否还有效呢？ 这正是我们要思考的问题。

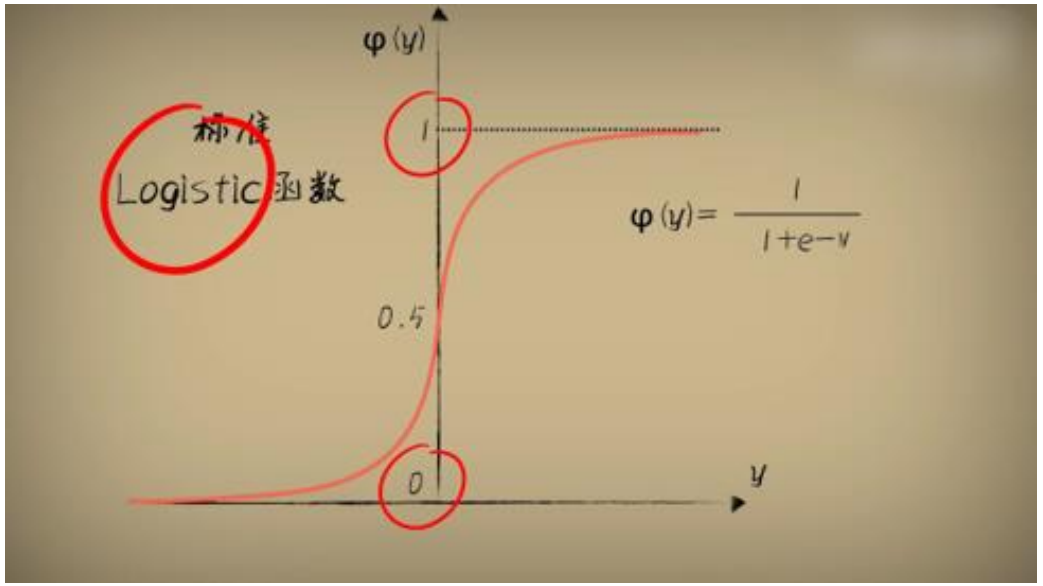
2 线性模型在分类任务上的“水土不服”

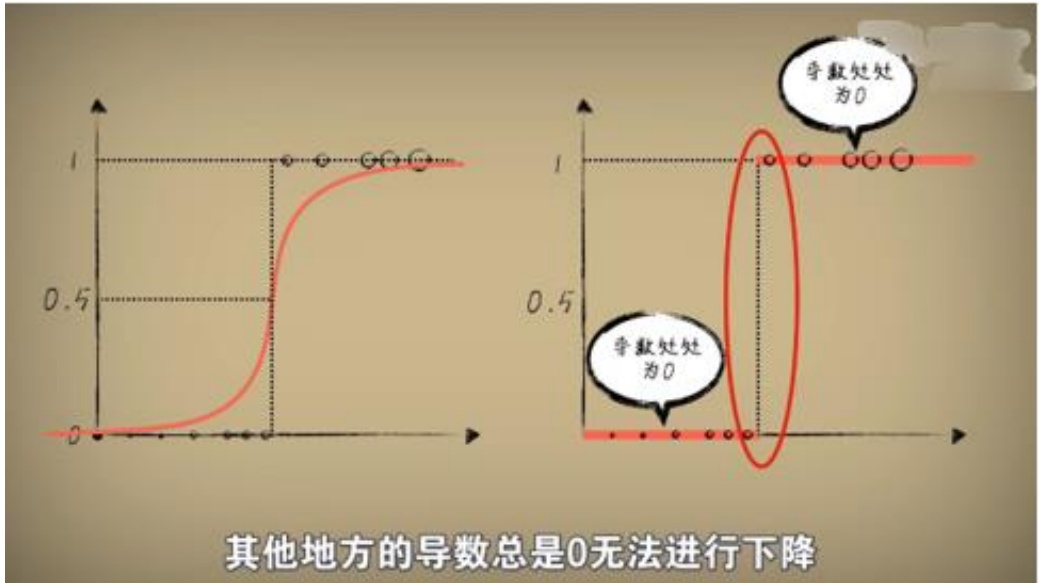
1. 输出范围的尴尬：
y=w*x + b 的输出是一个连续值，它的范围可以是任意实数（比如从负无穷到正无穷）。
2. 概率解释的缺失：
我们更希望模型的输出能代表一个“概率”。比如，输出0.8意味着“小蓝”有80%的把握认为这颗豆豆有毒。
线性模型 y=wx+b 的直接输出很难被赋予这样的概率意义。
3. 过于“硬”的判断边界：
虽然图中 y=wx+b 的直线似乎能大致分开两类豆豆，但它对处于“中间地带”的豆豆的判断可能过于敏感或不平滑。
我们理想中的分类器，应该能更平滑地从“确定无毒”过渡到“不确定”，再到“确定有毒”。

3 “小蓝”的迫切需求：一个能将输出“掰弯”并“压缩”的魔法

面对这种0/1的分类数据，直接用一条直线去拟合，显得力不从心。
“小蓝”迫切需要一种新的数学工具，能够：
接收 wx+b 计算出来的原始值。
将这个原始值进行一种“非线性”的转换（不再是直来直去的）。
把转换后的输出“压缩”到一个有意义的范围，比如 [0, 1] 区间，这样就可以解释为概率了。
这个魔法工具，就是我们之前提到的激活函数！它能帮助“小蓝”的输出从一条生硬的直线，变成一条更适合分类任务的、弯曲的S型曲线（例如Sigmoid函数）。

“小蓝”的新技能：让分类更“稳”的逻辑函数





我们需要一个“魔法工具”，将 wx+b 的输出转换为一个平滑的、概率化的值

1 引入 Logistic 函数：让“小蓝”的输出变得更有意义

Logistic 函数的公式：
σ(z) = 1 / (1 + e^(-z))
其中 z = wx + b，是线性模型的原始输出。
Logistic 函数的特性：
平滑过渡： 它的输出介于 0 和 1 之间，是一条 S 型的曲线。
概率解释： 输出值可以直接看作“属于某个类别的概率”（比如，0.8 表示有80%的概率是“有毒”）。
非线性变换： Logistic 函数将线性模型的输出“掰弯”成更适合分类的形状。

2 为什么 Logistic 函数比“分段函数”更强大?

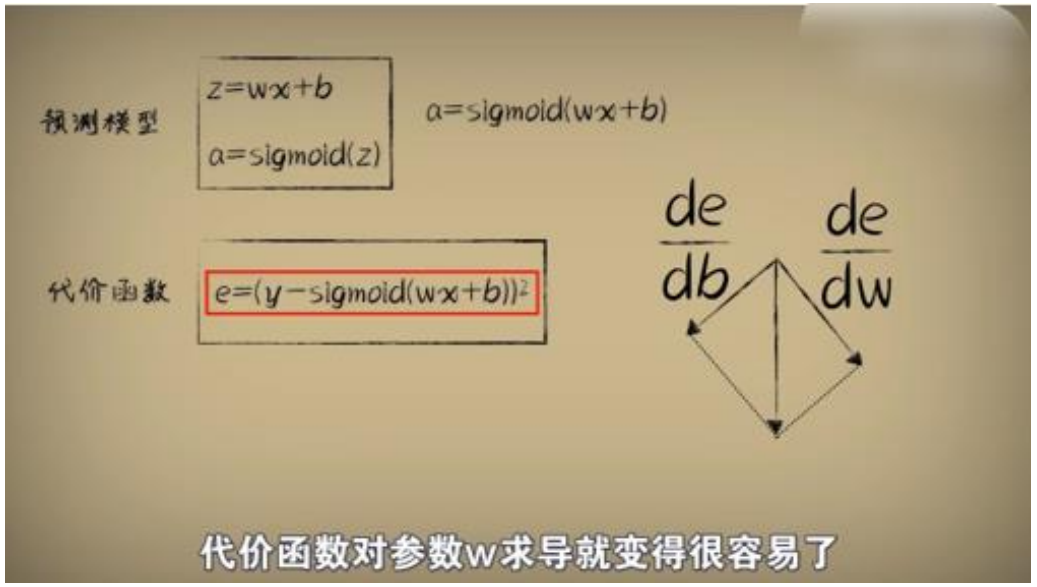
概率解释：
Logistic 函数的输出值不仅仅是 0 或 1，它还能表达模型对分类的“信心”。比如，输出值 0.9 表示“高度确定有毒”，而 0.6 表示“稍微倾向于有毒”。
平滑决策边界：
在横轴上，线性函数的输出是直线，但 Logistic 函数的 S 型曲线使得模型在决策边界附近的表现更加平滑和稳定。
可导性：
Logistic 函数是连续可导的（除了极端情况下导数趋近于0），这对于梯度下降等优化算法至关重要。

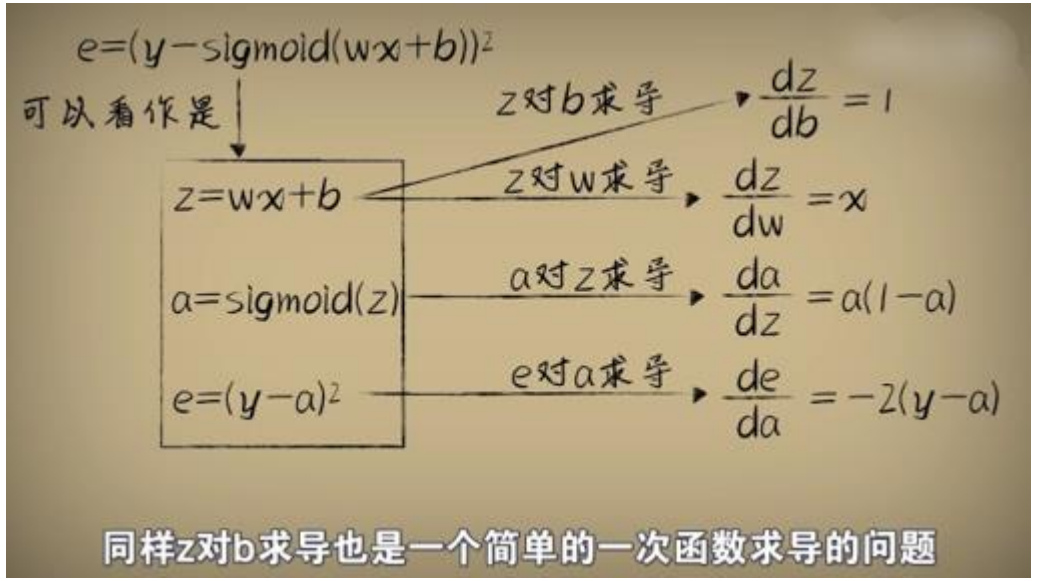
3 Logistic 函数如何帮助“小蓝”完成分类任务

Logistic 函数的引入，让“小蓝”从一个只能**输出连续值的“分析师”**，变成了一个能**输出概率值的“分类器”**。

它为“小蓝”提供了平滑、稳定的决策边界，并为后续的梯度下降优化提供了合适的导数。

“小蓝”的新视角：激活函数如何改变代价函数的世界





我们引入了 Sigmoid 激活函数，将线性模型 z=wx+b 的输出转换为一个在 [0,1] 范围内的概率值：a = sigmoid(z)。
激活函数的加入，不仅改变了模型的输出形式，还改变了代价函数的定义和计算方式。

1 代价函数的新定义：从线性误差到非线性误差

原来的代价函数是基于线性模型的误差计算的：**e = (y - (wx + b))²**。
加入激活函数后，模型的输出变成了 **a = sigmoid(wx + b)**，
代价函数随之变为：e = (y - sigmoid(wx + b))²

代价函数的目的是衡量模型预测值 a = sigmoid(wx + b) 与真实值 y 的差距。
我们需要计算代价函数对模型参数 w 和 b 的梯度 (∂e/∂w 和 ∂e/∂b)，以便在梯度下降算法中更新参数。

2 梯度计算的核心：链式求导规则

为了计算 ∂e/∂w 和 ∂e/∂b，我们需要逐层应用链式求导规则：

∂e/∂w = ∂e/∂a * ∂a/∂z * ∂z/∂w
∂e/∂b = ∂e/∂a * ∂a/∂z * ∂z/∂b

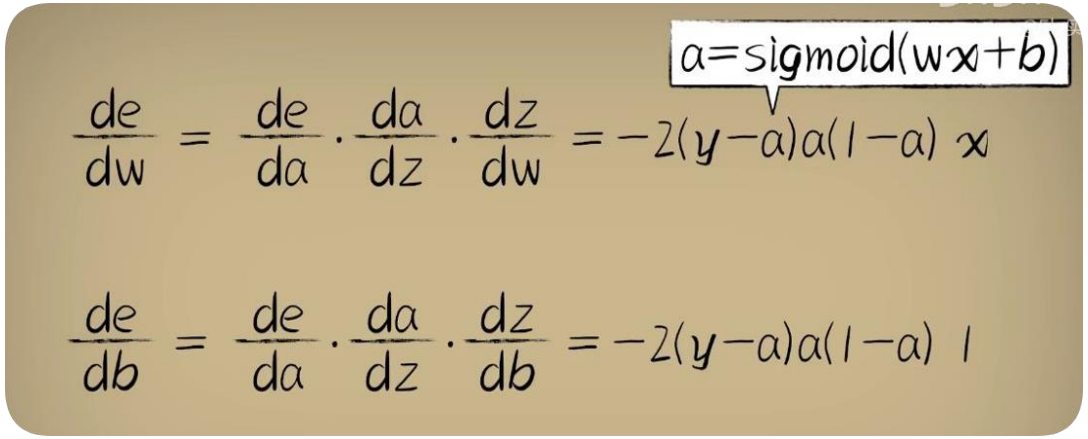
逐步解析：

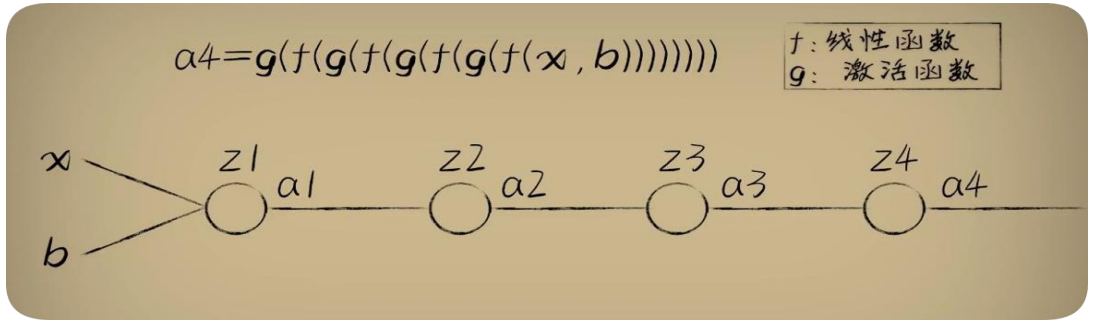
误差对激活值的导数 (∂e/∂a)：e = (y - a)²
对 a 求导，得到：∂e/∂a = -2(y - a)

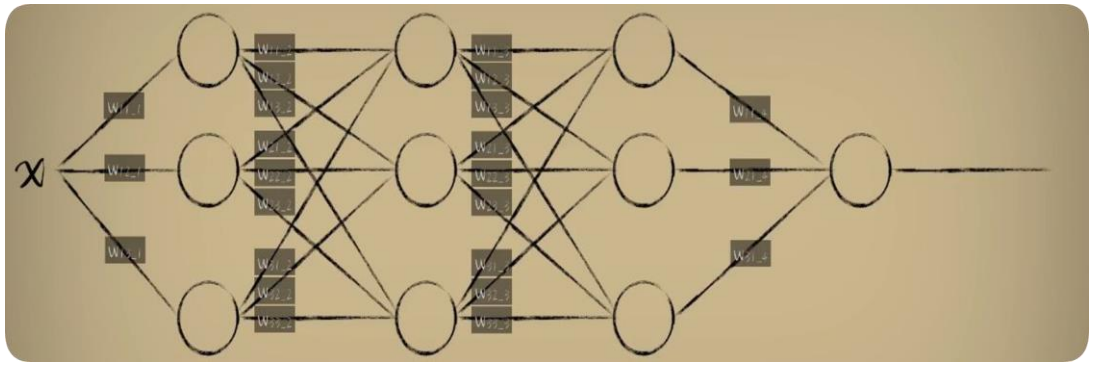
激活值对线性输出的导数 (∂a/∂z)：a = sigmoid(z)
对 z 求导，利用 Sigmoid 导数公式：∂a/∂z = a(1 - a)

线性输出对参数的导数 (∂z/∂w 和 ∂z/∂b)：z = wx + b
对 w 求导：∂z/∂w = x
对 b 求导：∂z/∂b = 1

“小蓝”的新视角：激活函数如何改变代价函数的世界







3 梯度的最终表达式

将以上各部分联合起来，我们得到：

∂e/∂w = -2(y - a) * a(1 - a) * x
∂e/∂b = -2(y - a) * a(1 - a) * 1

这些公式清晰地描述了代价函数关于每个参数的梯度，并成为更新参数的依据。

4 激活函数如何提升“小蓝”的学习能力

激活函数的加入，不仅改变了输出的形式，还优化了梯度计算的稳定性与效率。

梯度的分步计算逻辑清晰，易于实现，也为复杂神经网络提升了处理非线性的复杂问题能力。

5 更深层的复杂模型：嵌套的复合函数

对于深度神经网络，每一层的输出会成为下一层的输入，
形成一个嵌套的复合函数：a4 = g(f(g(f(g(f(x, b))))))

公式解读：
f 表示线性变换，例如 z = wx + b。
g 表示激活函数，例如 g(z) = sigmoid(z)。

多层神经网络，其本质就是一个嵌套的非常非常深的复合函数，当然不仅可以纵向的增加神经元的层数，也可以横向的增加每一层神经元的个数，而利用复合函数求导的链式法则，把梯度下降和反向传播完美结合后，不论这个网络有多深有多宽，我们都可以用同样的套路计算出代价函数在每个神经元上全部参数的导数，然后利用梯度下降算法去调整这些参数

感知器 到 反向传播算法

1 激活函数 - 赋予了灵魂

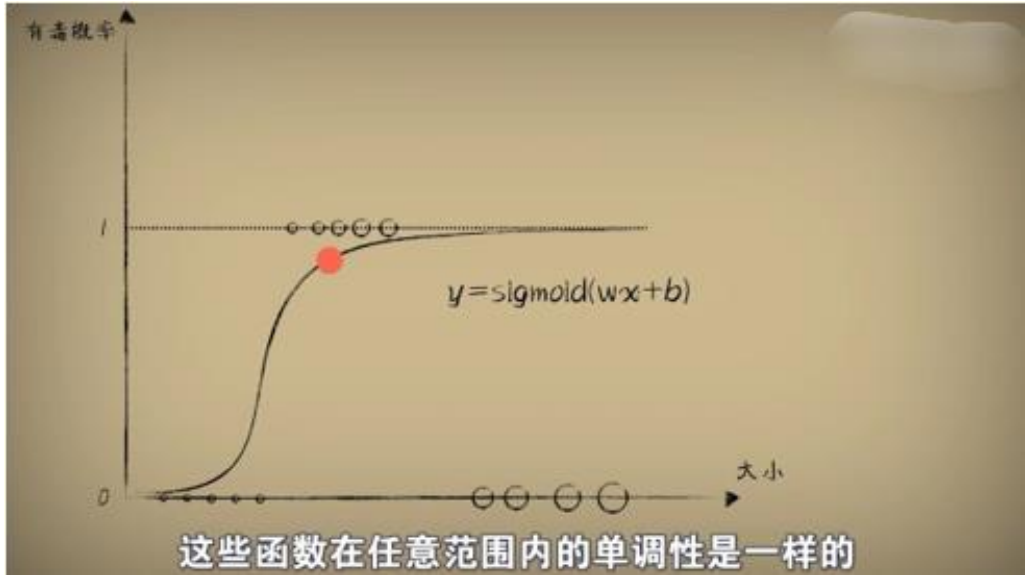
对于一个多层的神经网络，如果每个神经元都是一个线性函数，那么即使我们有很多很多的神经元构建出一个复杂的神经网络，从数学上来说，他们在一起仍然是一个线性系统，因为线性函数不论怎么叠加，结果都是一个线性函数。

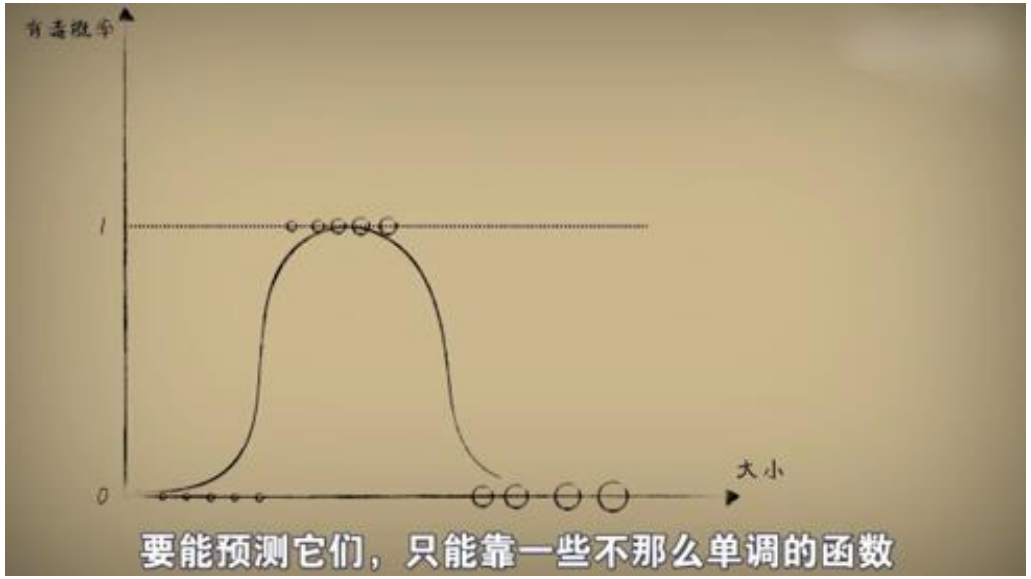
而激活函数是非线性的，它让我们的神经网络摆脱线性的约束，开始具备处理越来越复杂问题的能力。我们将在接下来详细的说明这个问题，加入激活函数神经元扩展到多个节点的网络，这时候也就可以说是更一般情况下真正的反向传播算法：把代价函数在神经网络中反向传播到各个节点，计算参数对代价函数的影响，然后调整参数。正是深度学习的开山鼻祖乔弗里辛顿和戴维诺姆特等人在1986年开始在神经网络的研究中引入的反向传播算法成为了现代神经网络的基石。

目前深度学习又统治着整个人工智能领域，而在此之前，人们对于多层感知器深度神经网络一直持有悲观的态度，正如明斯基和帕波特在反向传播算法发明之前的1961年是这么评价的。

本来平平无奇的利用本科一年级就学过的复合函数的链式求导法则的反向传播算法，却是在罗森布拉特感知器发明的27年后才被提出。正如深度学习领域的另一位巨头人物benjio说的那样，很多看似显而易见的想法，只有在事后才变得显而易见。

世界变了！“小蓝”的旧尺子量不准新豆豆





大自然往往是变幻莫测，喜怒无常。在一次地球环境巨变之后，小蓝所在的海底生物们也经历了巨大的进化，豆豆变得不再是简单的越大或者越小越可能有毒，而是在某个大小范围内有毒，而某些范围内无毒。

1 新挑战：豆豆世界出现了进化

但是，随着豆豆的进化！一批全新的豆豆出现了，它们的毒性分布非常奇特：
“看图中的数据点：小个头的豆豆是安全的（纵坐标接近0）。”
“中等个头的豆豆却是剧毒的（纵坐标接近1）。”
“但神奇的是，那些非常非常大的豆豆，反而又变得安全了（纵坐标再次接近0）！”
“这形成了一个‘低-高-低’的模式，就像一座小山丘。”

2 小蓝的S型曲线失灵了！

小蓝拿出它最擅长的 **a = sigmoid(w*x + b) 模型**，试图去理解这些新豆豆。
但它很快就发现，这个模型只能画出一条S形的曲线。这条S曲线要么一直往上走，要么一直往下走（取决于w的正负）。它根本无法画出一个‘先上去再下来’的形状！

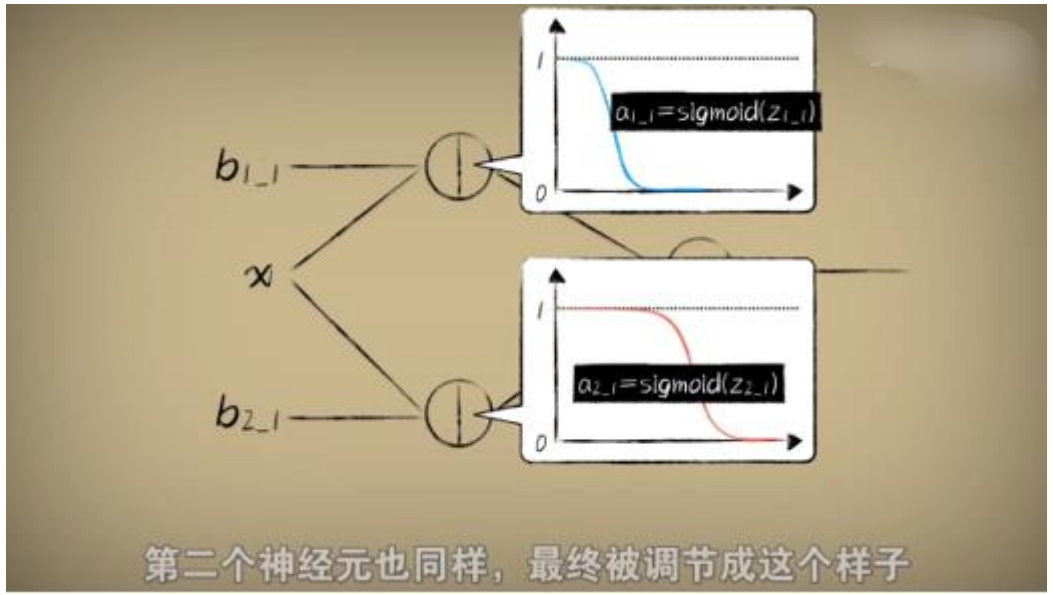
要想预测它们，只能靠一些不那么单调的函数。小蓝的S曲线是单调的，它遇到了前所未有的麻烦。
无论小蓝怎么调整它的 w 和 b，它的S曲线都无法准确地拟合这些‘山丘状’分布的豆豆数据。

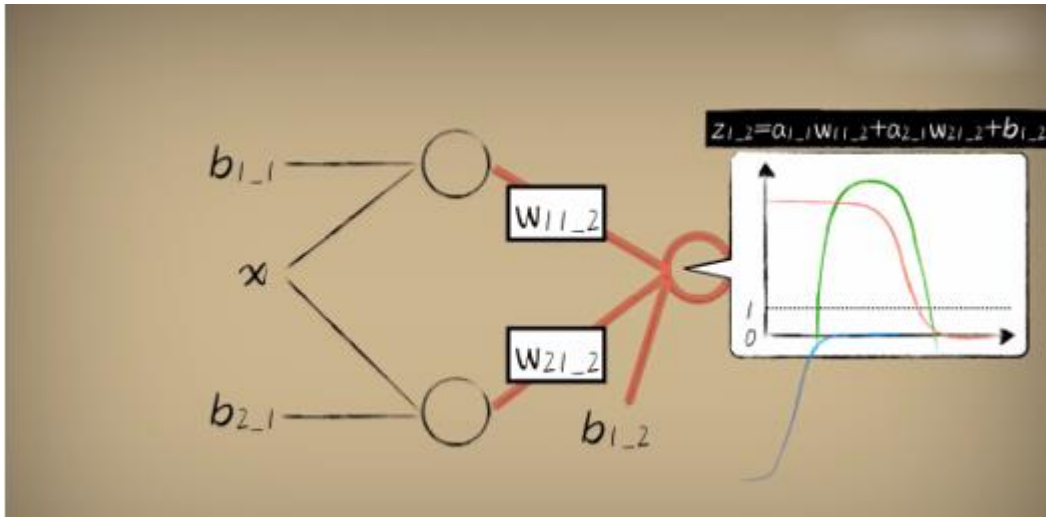
3 灵光一闪：“一个好汉三个帮”，组合的力量

小蓝意识到，要画出更复杂的图案，不能只靠一个画家。
如果能有一个‘绘画团队’，**每个成员负责一部分，然后巧妙地组合起来**，是不是就能解决问题了呢？

这个思路，正是从单个神经元迈向‘**神经网络**’的关键一步！

小蓝的进化之路：从单枪匹马到神经元“梦之队”的雏形





一个神经元，就像一个只能做简单判断的‘小算盘’，它的计算能力和表达模式都太单一了

是时候让神经元形成一个网络了。我们唯一做的事情就是添加了两个神经元，并把输入分别送入到这两个神经元进行计算，再把计算的结果送入到第三个神经元计算，最后输出

1 小蓝的顿悟：如果一个“小算盘”不够，那就来一队！

小蓝苦思冥想：**如果我能让多个‘小算盘’协同工作，每个‘小算盘’关注数据的一个方面，然后把它们的结果汇总起来，是不是就能拼凑出更复杂的规律了呢?**

这，就是从单个神经元向‘神经网络’迈进的灵感火花!

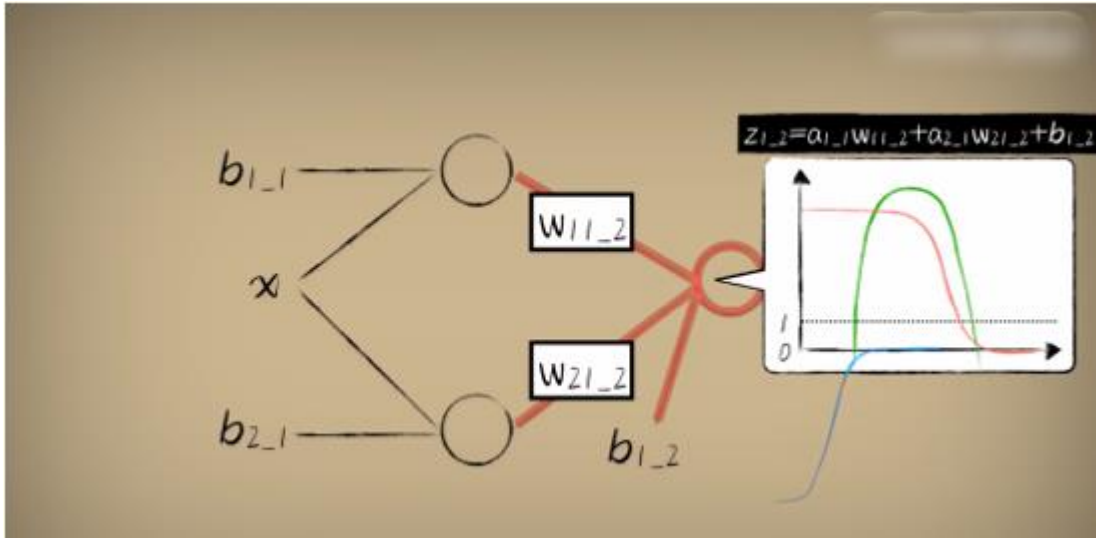
2 神经元“梦之队”的初次登场

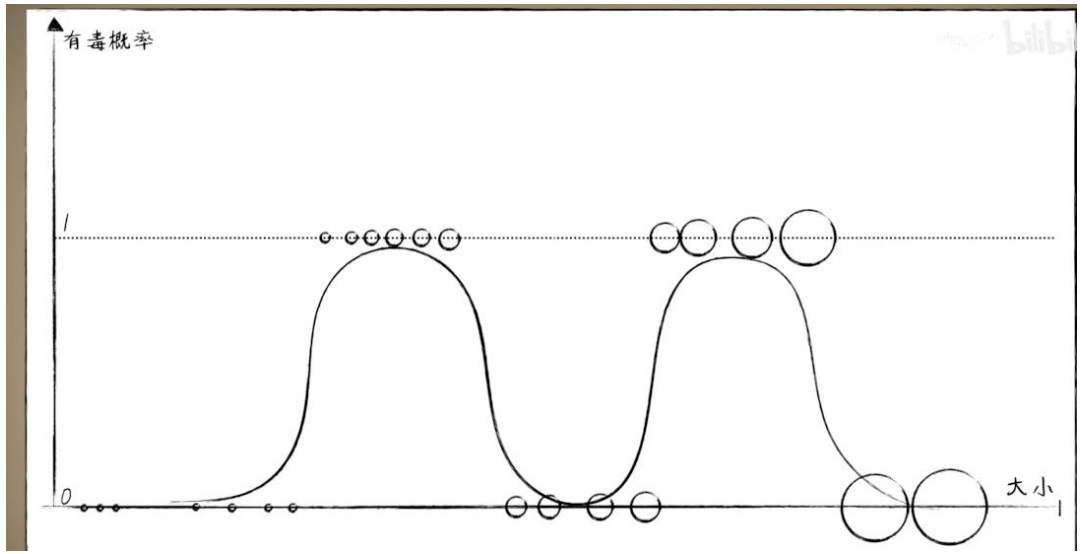
看！小蓝设计出了它的第一个‘神经元小队’。这不再是一个神经元单打独斗了。

第一层：特征提取员 (Feature Extractors)
“输入信号 x（比如豆豆的某种特性）同时传递给了两个位于第一层的神经元。它们各自有自己的‘小算盘’（权重，图中未标出但暗含在线条中）和‘起跑线’（偏置 b1_1 和 b2_1）。”
“这两个神经元，我们可以称它们为 **‘特征提取员A’和‘特征提取员B’**。它们可以学习从输入 x 中提取不同的简单特征。比如，‘A’可能对 x 在某个区间的变化敏感，‘B’则对另一个区间的变化敏感。”

第二层：决策指挥官 (Decision Maker/Combiner)
“‘特征提取员A’和‘特征提取员B’完成初步计算后（假设它们的输出分别是 a1_1 和 a2_1，都是经过Sigmoid等激活函数的），它们的结果会带着各自的‘重要性’（权重 W11_2 和 W21_2）被送到第二层的‘决策指挥官’那里。”
“这个‘指挥官’也有自己的‘微调旋钮’（偏置 b1_2）。**它负责把前一层传来的多个‘特征信号’进行加权组合，形成一个更高级的判断 z1_2 = a1_1*W11_2 + a2_1*W21_2 + b1_2**，最后再通过激活函数输出最终结果。”

小蓝的进化之路：从单枪匹马到神经元“梦之队”的雏形





3 “梦之队”如何创造奇迹

这个‘神经元小队’是如何画出更复杂曲线的呢?

想象一下，特征提取员A 的输出（经过激活后）可能是图中的蓝色曲线，它捕捉了数据的一种简单趋势。

特征提取员B 的输出可能是红色的曲线，它捕捉了数据的另一种简单趋势。

当‘决策指挥官’把这两个简单的、可能是单调的曲线（蓝和红）通过不同的权重（W11_2, W21_2）进行线性组合，再加上偏置 b1_2，然后整体再经过一次激活函数（比如Sigmoid），
它就有可能创造出一个全新的、非单调的、更复杂的曲线—就像图中那条绿色的‘山丘’曲线!

这就是组合的力量！通过叠加和变换多个简单函数，我们得到了一个表达能力远超单个神经元的新模型!

简单总结，我们添加一个神经元后，相当于增加了一个抽象的维度，把输入放进这些不同的维度中，每个维度通过不断的调整权重并进行激活，从而产生对输入的不同理解。最后再把这些抽象维度中的输出进行合并，降维得到最终输出。这个输入数据由于在多个抽象维度中被产生了不同的解读，从而让输出得到了更多的可能

随着的训练数据越充足，最后训练得到的模型也就能够越好的去预测新的问题，因为越充足的训练数据就在越大程度上蕴藏了问题的规律和特征，新的问题数据也就越难以逃脱这些规律的约束。

所以我们总是说机器学习神经网络的根基是海量的，一个训练之后拟合适当的模型，进而在遇到新的问题数据的时候，也能大概率产生正确的预测。我们把这个现象称之为模型的泛化，模型的泛化能力，也就是神经网络追求的核心问题。

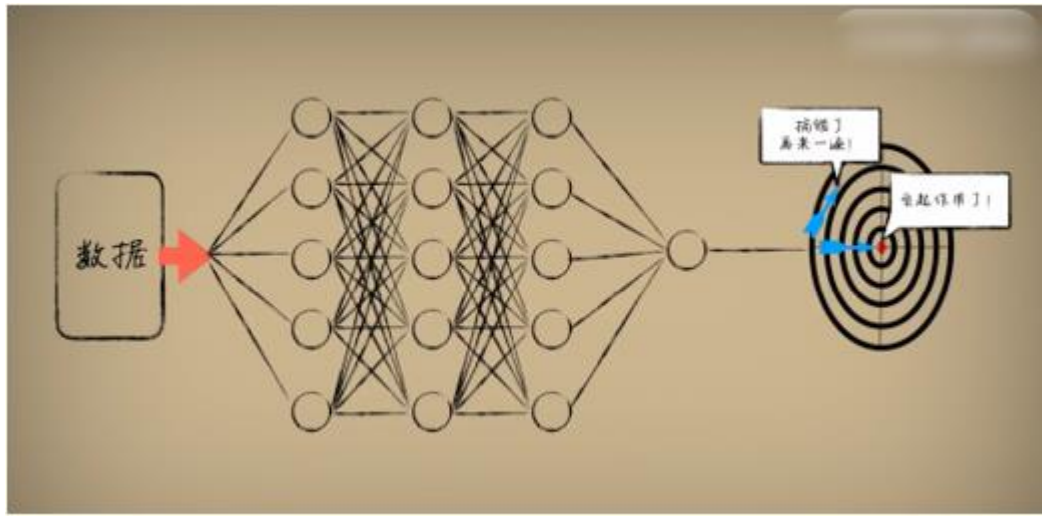
4 模型强大了，但“尺子”依然是灵魂!

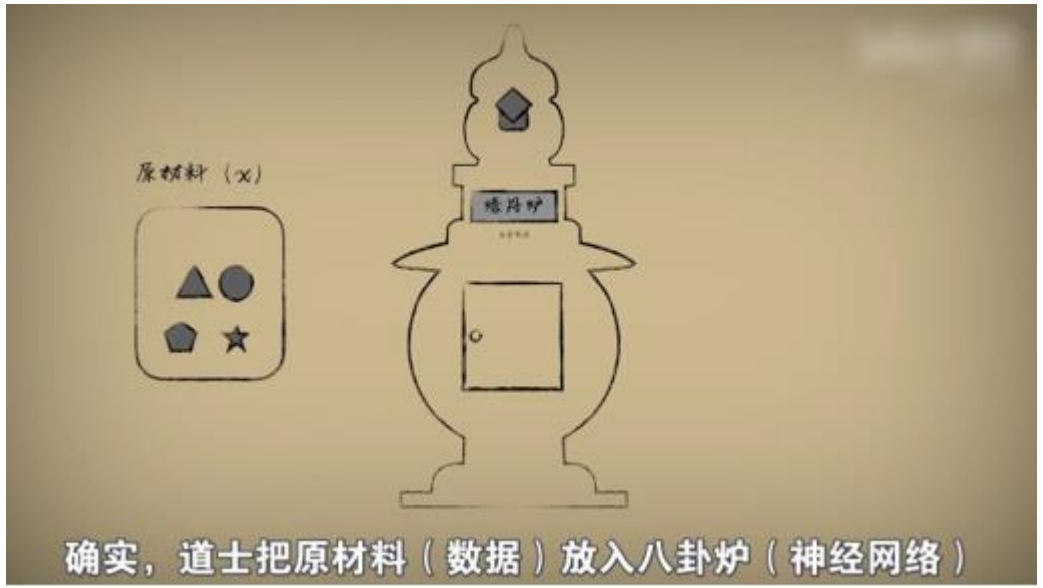
现在，小蓝拥有了一个更强大的画图工具（多神经元模型），能够画出更复杂的形状了。那么，我们之前讨论的‘尺子’— 代价函数，是不是就不那么重要了呢?
“恰恰相反！模型越强大、越灵活，一把好的‘尺子’就越关键！”

指引方向：这个由多个神经元组成的“梦之队”，内部有更多的参数（所有的w和b）需要调整。我们需要一个精确的代价函数来告诉整个团队：“你们现在的整体表现如何？距离目标还有多远？应该朝哪个方向集体调整？”
避免“乱画”：更强大的模型也意味着它有能力画出更多奇奇怪怪的、不符合数据规律的曲线。一个好的代价函数（比如针对分类问题精心设计的交叉熵）能够更有效地惩罚那些“离谱”的预测，引导模型学到数据中真正有意义的模式，而不是仅仅拟合噪声。

这正是我们之前深入探讨代价函数的原因 - 为‘小蓝’在不同情境下选择最合适的‘尺子’，让它的学习之路更高效、更精准！”

深度神经网络





我们经常听到深度学习，其中深度二字其实并没有什么特别的奥义，只是指一个神经网络中纵向的隐藏层比较多，换句话说很深，我们一般把隐藏层超过三层的网络也就称之为深度神经网络

1 深度学习中广受诟病的地方

当数据第一次通过网络，‘小蓝’的‘军团’会根据其内部当前的参数（所有神经元的权重和偏置）给出一个预测结果——就像射出了第一箭。

很可能，第一次尝试并不能完美命中。箭射偏了！这时，‘小蓝’需要知道：我射偏了多少？偏向了哪个方向？这，正是我们之前反复强调的‘智能尺子’— 代价函数 — 大显身手的时候！代价函数会精确地衡量‘预测结果’（箭的落点）与‘真实目标’（靶心）之间的‘差距’或‘成本’。

一旦‘智能尺子’（代价函数）量出了‘错误’的大小和方向，这个信息就会被用来调整‘小蓝’的‘军团’。这个‘错误信号’会像指令一样，通过一个叫做‘反向传播’的精妙过程，在整个神经网络中逐层传递回去。它会告诉网络中的每一个神经元：‘嘿，因为你的这一点点贡献，我们最终射偏了这么多，你需要这样调整一下你的权重和偏置，下次才能射得更准！’

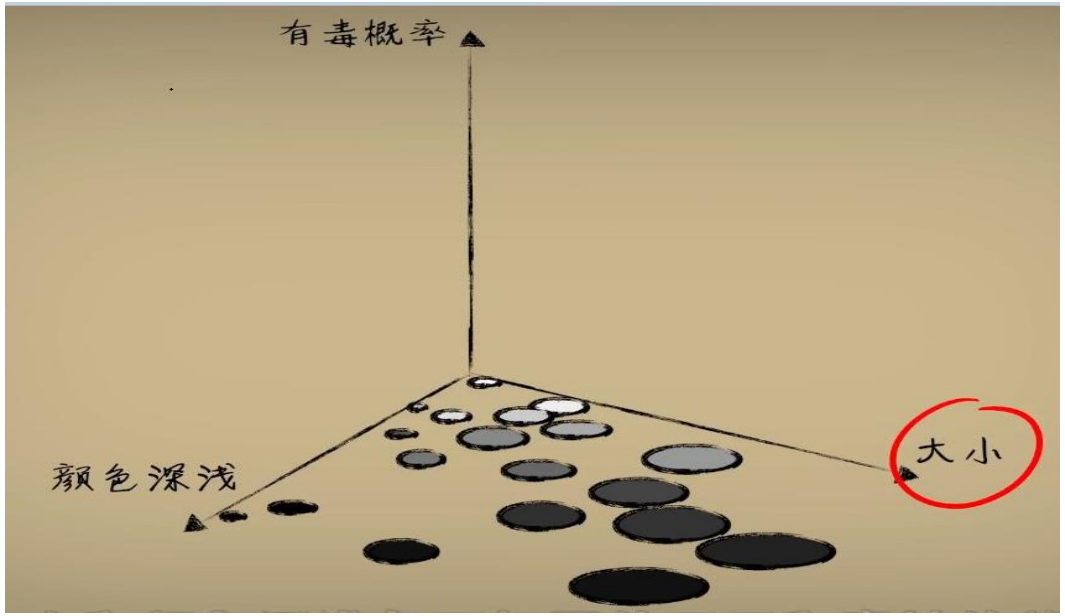
“这个‘射击 -> 测量误差 -> 调整参数 -> 再次射击’的过程会不断重复。每一次迭代，‘小蓝’的‘军团’都会根据代价函数提供的反馈，对内部成千上万的参数进行微调，努力减小下一次的‘射击误差’。”

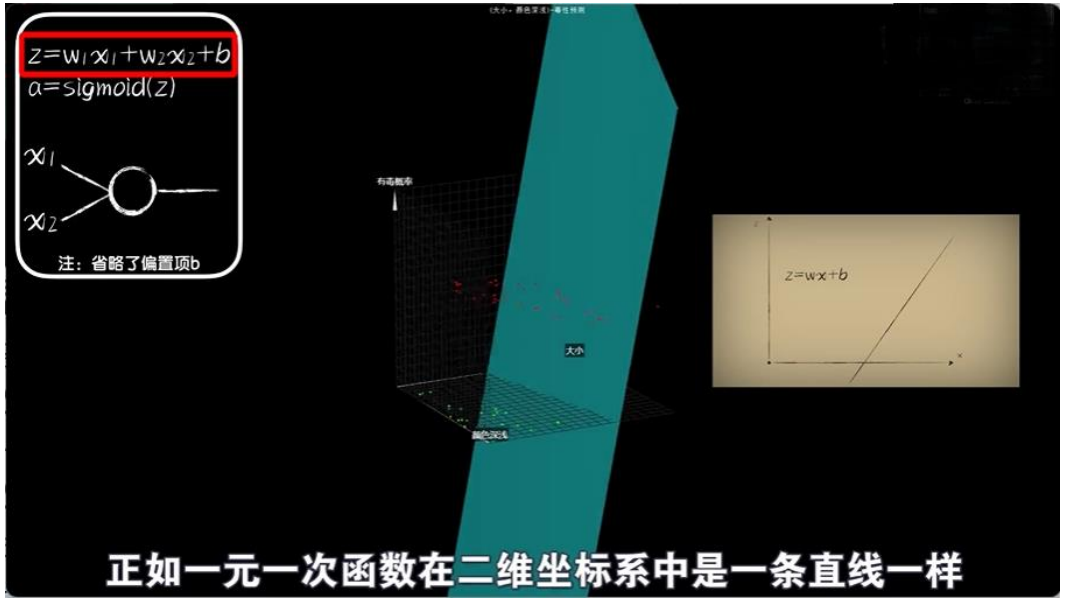
隐藏层的神经元在理解什么提取什么都太过微妙，虽然我们对大致的结果有所把握，但却很难用精确的数学去进行描述，我们能做的也只有设计一个网络送入数据，然后充分的训练，如果得到的预测效果好，我们就会说它起作用，如果不好，那也只能说搞错了，调校参数在再来一遍。

2 训练的过程像炼丹

就像道士把原材料放入八卦炉开火炼丹，最后得到的仙丹可能让人长生不老，也可能让人一命呜呼。炼丹过程中八卦炉里发生的微妙事情，道士也是不得而知的，虽然是他设计了炼丹的一切。

高维空间：机器如何面对越来越复杂的问题





从直线到平面：“小蓝”眼中的豆豆世界升级了！

1 二维 -> 三维

如果我们想要去**判断一个人是否善于打篮球**，很明显仅仅考虑他的**身高特征**是不合理的。虽然这个特征很关键，但还需要从其他的角度去分析，比如**体重身体灵活性以及是否经常见到凌晨4:00的太阳等等因素。**

同样在之前讲解中，为了让大家更加清晰的了解神经网络的工作原理，一直仅仅在用豆豆大小这个特征去预测它的毒性，但现实世界往往并非如此简单。比如豆豆的**毒性不仅和它的大小有关系，也和它的颜色深浅有关**，当毒性只和大小有关系的时候，我们可以用一个二维的坐标系来描述这个关系。

还记得吗？当‘小蓝’只关心豆豆的‘大小’（一个特征 x）时，它的神经元计算 **z = w*x + b**。
如果我们要画出 z 和 x 的关系，**那就会是一条直线。之后，Sigmoid函数再把这条直线‘掰弯’成S形，得到0到1的概率。**

现在，小蓝 变得更聪明了，它发现豆豆的‘颜色深浅’也是判断是否有毒的重要线索！
于是，‘小蓝’的神经元开始接收**两个输入信号**：
x1：豆豆的大小
x2：豆豆的颜色深浅
神经元内部的计算也升级了，变成了 **z = w1*x1 + w2*x2 + b**。这里，w1 是大小的‘重要性’（权重），w2 是颜色深浅的‘重要性’，b 依然是那个偏置项。

当只有一个输入 x 时，z 的值画出来是一条线（在x-z平面上）。
“现在我们有了两个输入 x1 (大小) 和 x2 (颜色深浅)，还有一个输出 z (有毒概率)。这需要一个三维的空间来展示它们的关系！”
在这个三维空间里，方程 z = w1*x1 + w2*x2 + b 不再是一条直线了，它变成了一个平面！

“这个平面根据豆豆的大小 (x1) 和颜色深浅 (x2)，给出一个综合的‘有毒概率’值 z。图中那些红色和绿色的点，就代表了不同大小和颜色的豆豆样本。”

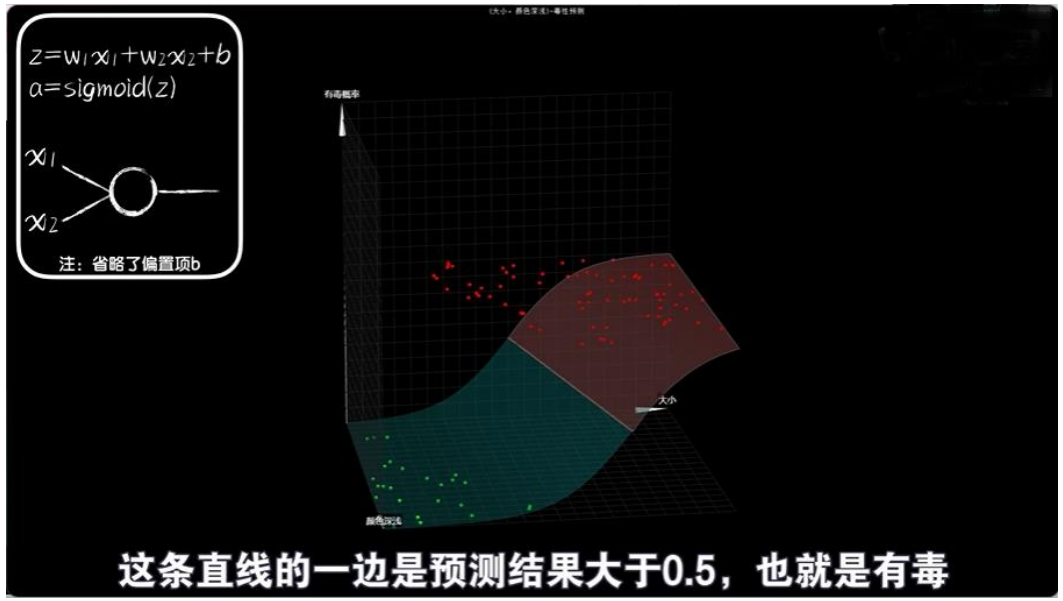
2 小蓝 的线性“分割板”

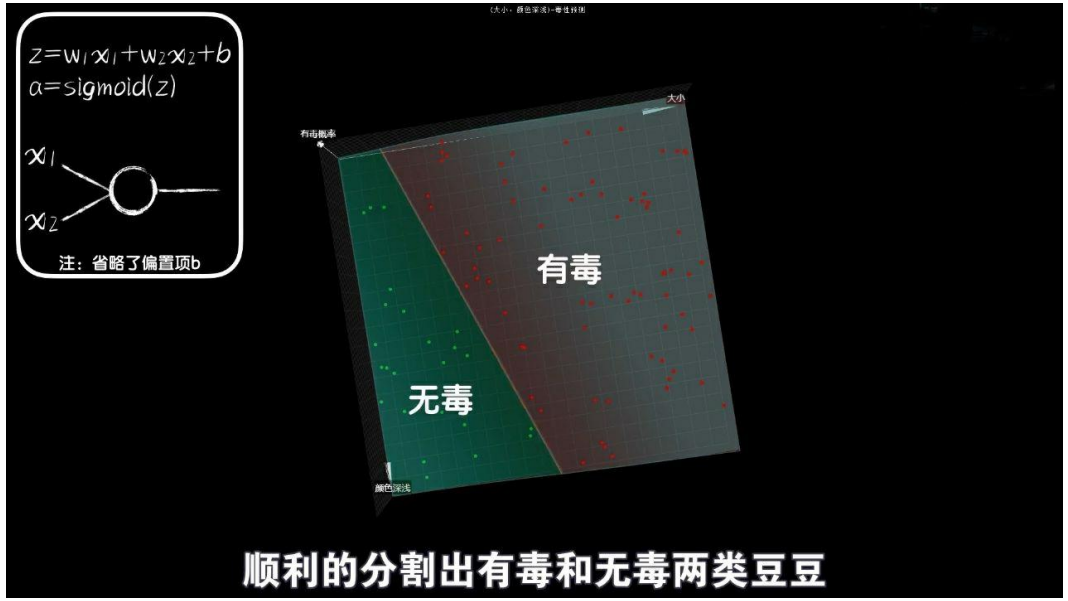
从只看‘大小’（一维输入，直线模型），到同时看‘大小’和‘颜色’（二维输入，平面模型），‘小蓝’对豆豆世界的理解，在维度上进了一步！

虽然这里我们只增加了颜色这一个维度，使得模型从直线变成了平面。但想象一下，如果豆豆还有形状、纹理、产地等更多特征呢？每增加一个特征，我们描述豆豆的数据空间就会增加一个维度，而神经元（或神经网络）内部的线性计算部分，也会在更高维的空间中形成所谓的‘超平面’(hyperplane)来进行分割。

理解数据如何在多维度上分布，以及我们的模型如何在这些高维空间中学习和划分，是通往更强大人工智能的关键一步。”

高维空间：机器如何面对越来越复杂的问题





从平面到曲面，再到决策“等高线”：“小蓝”的概率地图

1 激活函数的魔力：把平面“掰弯”成概率曲面！

现在，小蓝的老朋友——**Sigmoid激活函数**登场了！它的作用就是把线性计算得到的 **z 值（也就是那个平面的‘高度’）转换成一个0到1之间的概率值 a。**

当Sigmoid函数作用于之前那个平面上的每一个点后，原来那个直愣愣的平面就被‘掰弯’成了一个光滑的S形曲面。这个曲面的‘高度’现在就代表了‘小蓝’预测这颗豆豆有毒的概率 a。

“这个曲面，我们可以称之为‘概率等高面’或者‘概率地形图’。地势高的地方（比如图中红色区域对应的曲面部分），代表有毒的概率大；地势低的地方（绿色区域对应的曲面部分），代表有毒的概率小。”

“在二分类问题中（比如判断豆豆有毒/无毒），我们通常会选择一个阈值来做决策。最常用的阈值就是概率 a = 0.5。”
“如果 a > 0.5，我们就判断豆豆有毒。”
“如果 a < 0.5，我们就判断豆豆无毒。”
“如果 a = 0.5，那就表示模型认为有毒和无毒的概率正好相等，处于‘摇摆不定’的边界上。”

“那么，在这个弯曲的概率曲面上，所有概率 a 等于0.5的点连接起来，会形成一条线。这条线就是这个概率曲面在‘海拔’0.5高度上的一条‘等高线’”

2 从三维曲面到二维“决策地图”

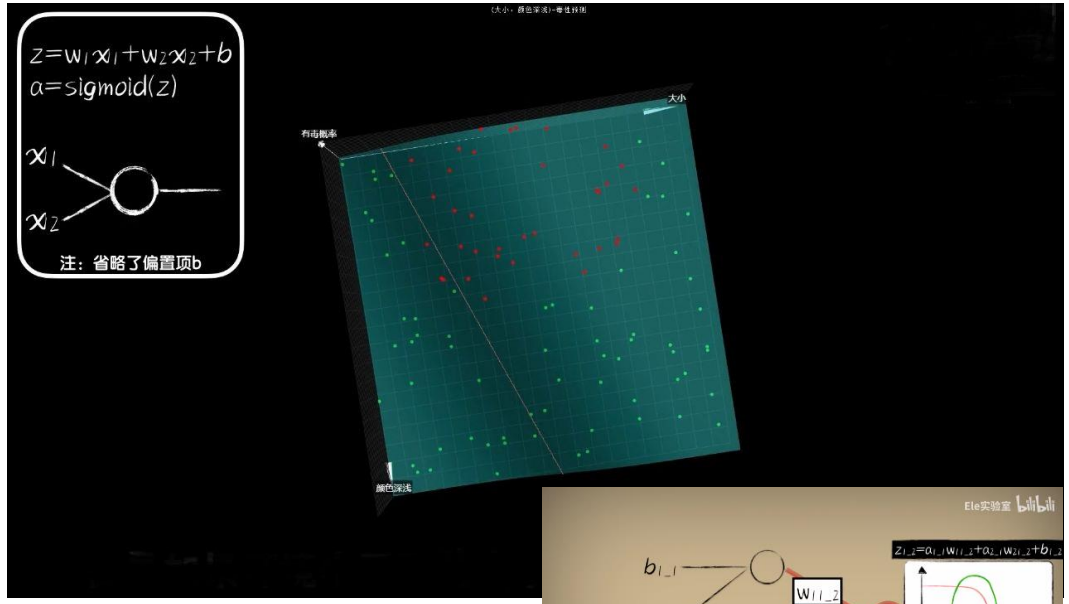
虽然这个三维的概率曲面很直观，但有时候我们更关心**在原始的二维输入空间**（由‘大小’x1 和‘颜色’x2 构成的‘地面’）上，**那条‘有毒’和‘无毒’的分界线**在哪里。”

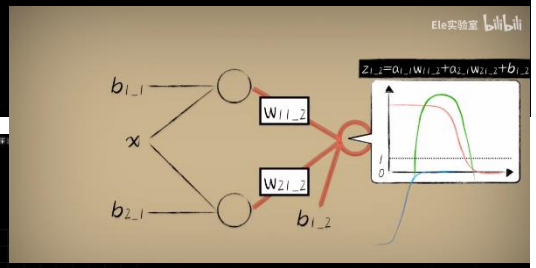
怎么做呢？**我们可以把刚才在概率曲面上找到的那条 a=0.5 的‘等高线’，垂直地投影到 x1-x2 这个‘地面’上。这样，我们就在二维的输入空间中得到了一条线。”**

“这条投影下来的线，就是图下方文字所指的‘这条直线’（虽然它在二维平面上可能不是严格的直线，而是一条曲线，但当Sigmoid的输入z=0时对应a=0.5，而z=0即w1x1+w2x2+b=0，这在x1-x2平面上确实是一条直线）。这条线把二维的输入空间分成了两部分：

“线的一边，对应的是概率曲面上 a > 0.5 的区域，‘小蓝’会预测‘有毒’。”
“线的另一边，对应的是概率曲面上 a < 0.5 的区域，‘小蓝’会预测‘无毒’。”
“这条线，就是‘小蓝’在这个二维豆豆世界（大小 vs 颜色）中画出的决策边界！它就像一张地图上的分界线，告诉我们哪些组合的豆豆特征会被判定为有毒。”

神经元“合力”扭转乾坤：让决策边界“弯”起来！





多个神经元通过组合多个线性边界形成非线性决策边界

3 小蓝 的智慧升级：组建神经元“小分队”！

当‘小蓝’只有一个神经元（接收 x1, x2 输入）时，它在二维的豆豆世界（大小 vs 颜色）中画出的决策边界（概率 a=0.5 的等高线）是一条直线。
这对于某些简单的数据分布可能够用，但如果好豆豆和坏豆豆的分布更复杂，一条直线可能就无法完美分开了。

为了应对更复杂的挑战，‘小蓝’不再让一个神经元单打独斗，而是组建了一个‘**神经元小分队**’— 也就是一个包含‘**隐藏层**’的神经网络！
“输入 x1 和 x2 首先被送入第一层的两个‘隐藏神经元’（图中中间的两个圆圈）。”
“这两个隐藏神经元，**我们叫它们‘队员A’和‘队员B’。它们各自独立地进行计算**：”
z1_1 = w11_1*x1 + w21_1*x2 + b1_1 => a1_1 = sigmoid(z1_1) (队员A的输出)
z2_1 = w12_1*x1 + w22_1*x2 + b2_1 => a2_1 = sigmoid(z2_1) (队员B的输出)
可以想象，‘队员A’和‘队员B’各自在二维输入空间中，都试图画出自己的（经过Sigmoid掰弯后的）‘分界线’或‘概率坡度。”

“然后，‘队员A’的输出 a1_1 和‘队员B’的输出 a2_1，会作为新的信息，**被送入第二层的‘输出神经元’**。
“**这个‘输出神经元’（我们叫它‘队长’）也会进行自己的计算**：”
z2_2 = w11_2*a1_1 + w21_2*a2_1 + b1_2 => a1_2 = sigmoid(z2_2) (最终的输出概率)
“关键就在这里！‘队长’通过对‘队员A’和‘队员B’的输出进行加权组合，再经过一次Sigmoid激活，就能够创造出比单个队员更复杂的决策模式！”

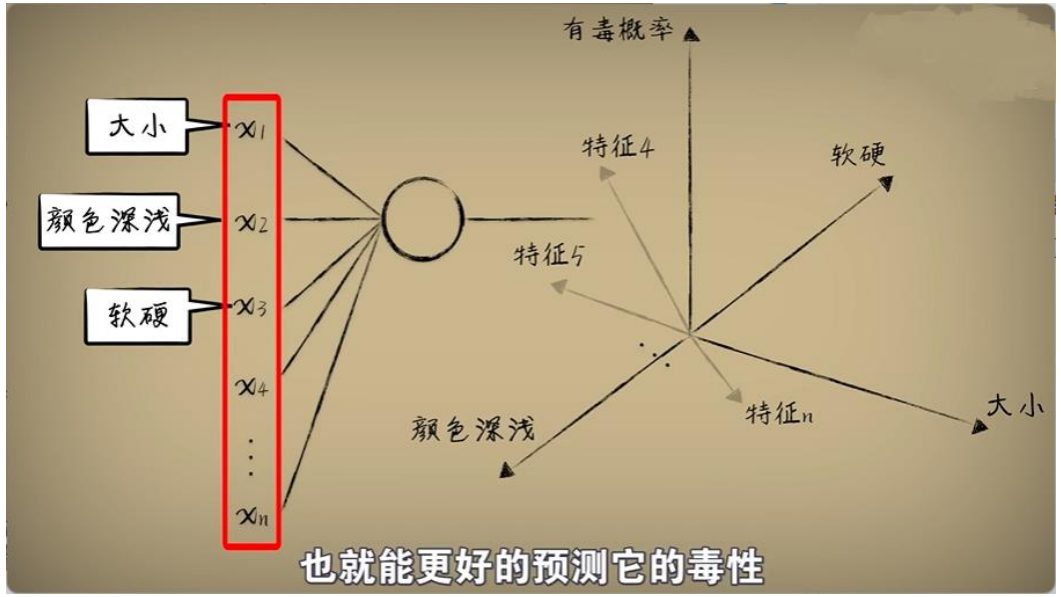
看！现在这个三维空间中的概率曲面，不再是那个相对简单的S形弯曲平面了，它变得更加复杂，可以被‘扭曲’和‘塑造’。”通过增加神经元层次和数量，神经网络获得了‘扭曲’其决策面的能力。”

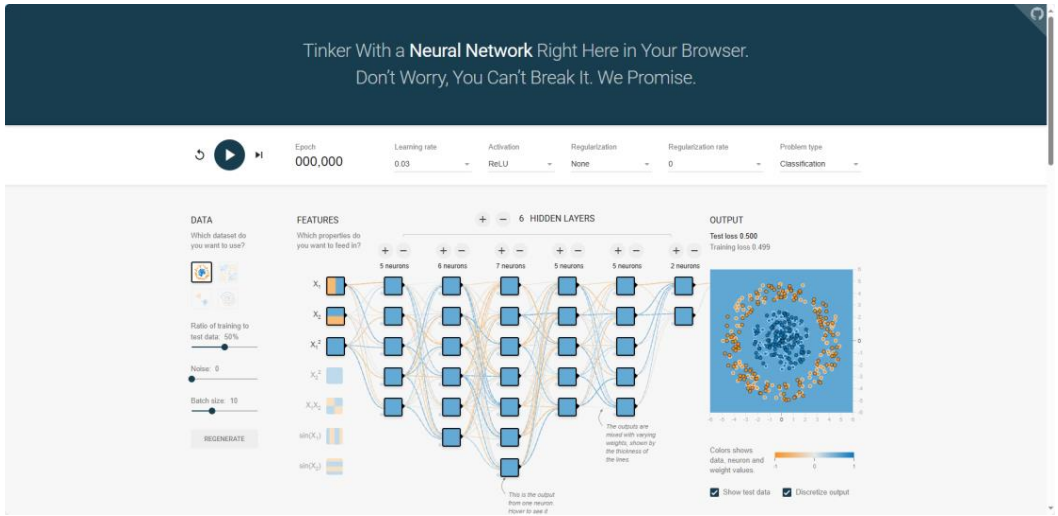
4 弯曲的“等高线”：更智能的决策边界

当我们在右边这个更复杂的、被扭曲的概率曲面上寻找概率 a1_2 = 0.5 的那条‘等高线’，并把它投影到二维的 x1-x2 输入平面上时，我们会惊奇地发现：
“**这条决策边界不再是一条直线了！它变成了一条曲线！**”（如图中概率曲面上那条隐约可见的橙色分界线）

“这意味着，通过神经元的“团队合作”（一个隐藏层加上一个输出层），**‘小蓝’现在有能力画出弯曲的、非线性的分界线，从而能够更好地适应和区分那些用直线无法完美分开的复杂数据分布。”**
比如，它可能学会画一个圆圈，把某一类豆豆圈在里面，或者画出更复杂的波浪形边界。

神经元“合力”扭转乾坤：让决策边界“弯”起来！





5 更丰富的数据特征

到这里可以想象当问题**数据特征越来越多**，也就是说我们**采集豆豆的更多特征**，比如除了**大小颜色深浅**以外，再加一个**外壳硬度**作为输入数据的时候，以此类推。

这样我们就只能再**添加一个维度去描述预测模型的图像。**

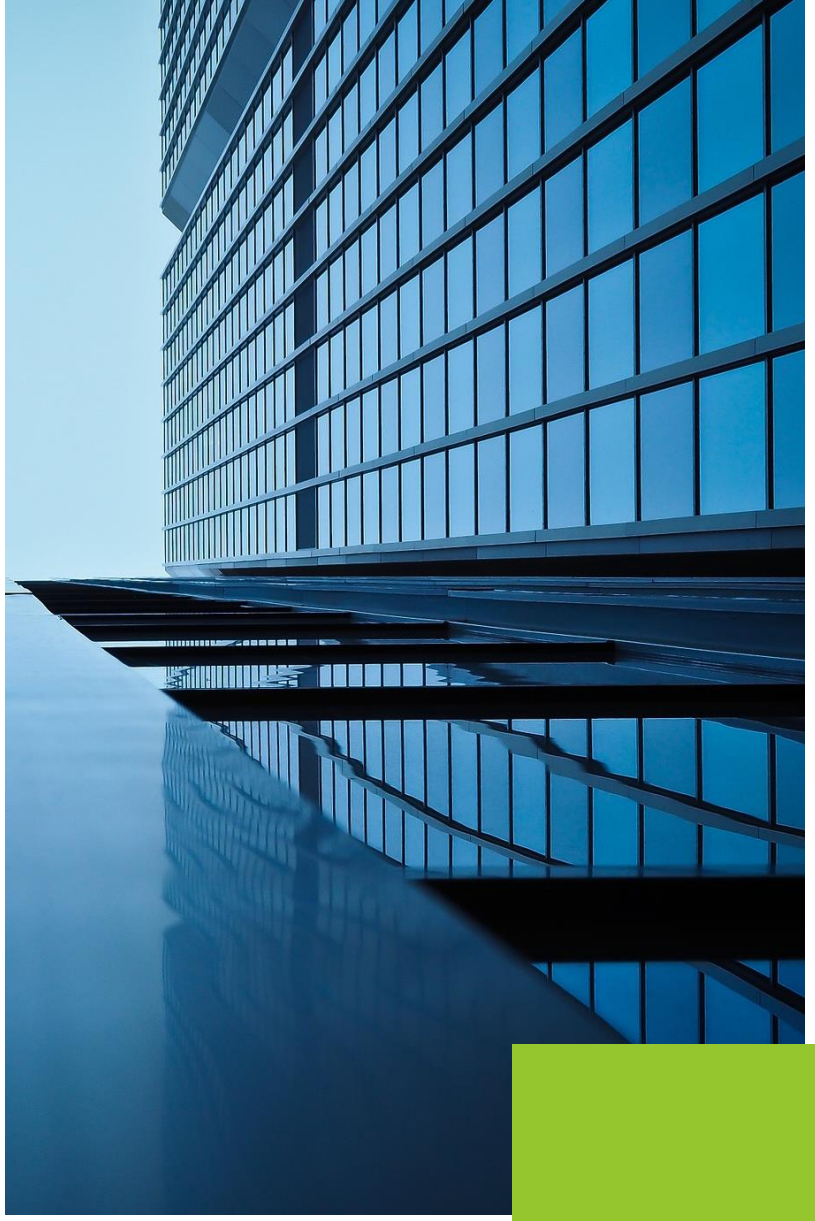
换句话说我们需要在四维空间作图，作为三维世界的我们也就心有余而力不足了，只可意会而不可言传，但是**数学作为一种抽象的工具**，却可以很容易的做到这一点，在数学看来，**这不过就是把输入多加了一个维度而已，而输入数据有多少元素，也就是所谓的特征维度也叫数据维度。**

6 高维空间的力量：组合创造无限可能

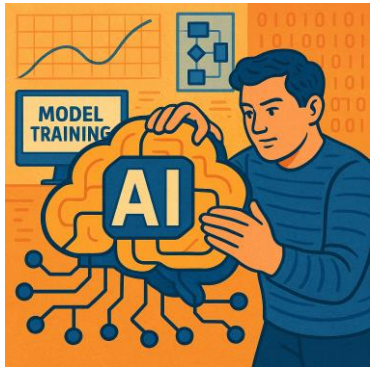
这个从**‘直线’到‘曲线’的飞跃**，正是神经网络强大能力的核心体现之一。**每一个隐藏神经元可以看作是在学习数据的一个简单特征或一个简单的线性分割面。而输出神经元则学习如何将这些简单的特征/分割面进行巧妙的非线性组合**，从而形成对整体复杂模式的识别。

我们只展示了增加一层、两个隐藏神经元的情况。如果增加**更多的隐藏神经元，或者增加更多的隐藏层**，神经网络就能学习到**更复杂、更精细的决策边界**，去**拟合更高维度、更难以想象的数据分布。**

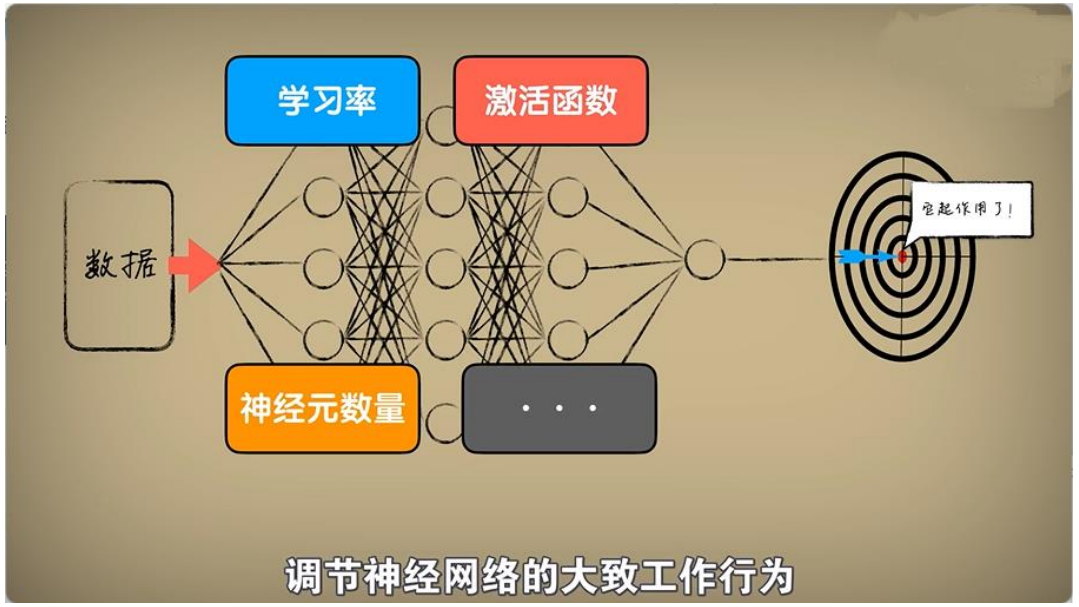
这正是‘小蓝’在探索‘高维空间’时，不断提升自己能力的关键所在—**通过构建更深、更宽的网络，它就能在那些难以用简单几何形状描述的高维数据迷宫中，找出通往正确答案的路径。**

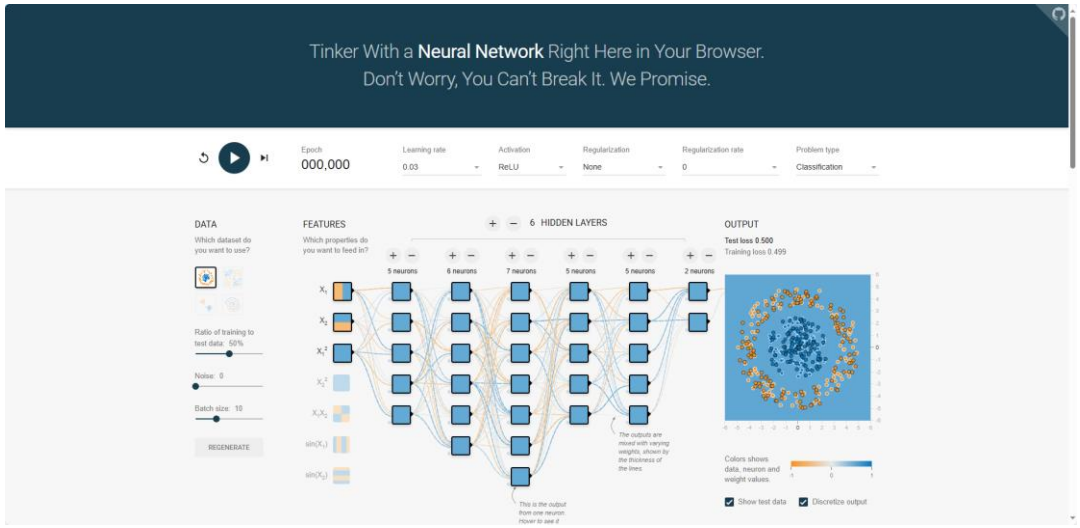


02 Tensor Flow游乐场



深度学习：神奇的DeepLearning





1 深度学习

深度学习就是**不断的增加一个神经网络的隐藏层神经元**，让输入的数据被这些神经元**不断的抽象和理解**，最后得到一个**具有泛化能力的预测网络模型**，而我们一般把**隐藏层超过三层的网络也就称之为深度神经网络。**

如此说来深度学习，这个听起来相当高端大气上档次，尤其是近些年来被媒体包装的十分具有神秘感的名字，瞬间你就觉得没有那么的道貌岸然。

至于网络中每个节点到底在理解什么，**很难用精确的数学手段去分析**，我们唯一能做的事情就是**收集数据送入数据进行训练，然后期待结果。**

当然也不是说我们对一个深度神经网络完全不可把控，起码我们能在比如学习率、激活函数、神经元的层数和数量等等方面，调节神经网络的大致工作行为，俗称调参。

2 Tensor Flow游乐场

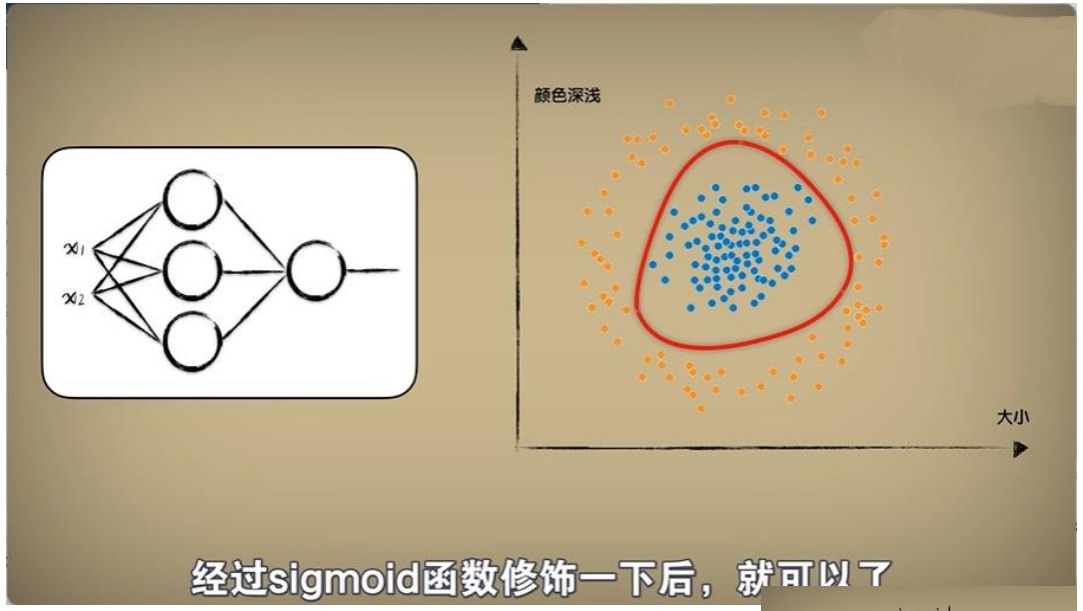
我们以著名的Tensor Flow游乐场为例，很多初学者一开始接触神经网络的时候，大概率都会被一些神秘的力量引导到这个网站来，这样点两下就可以搭建神经网络的体验真不错，

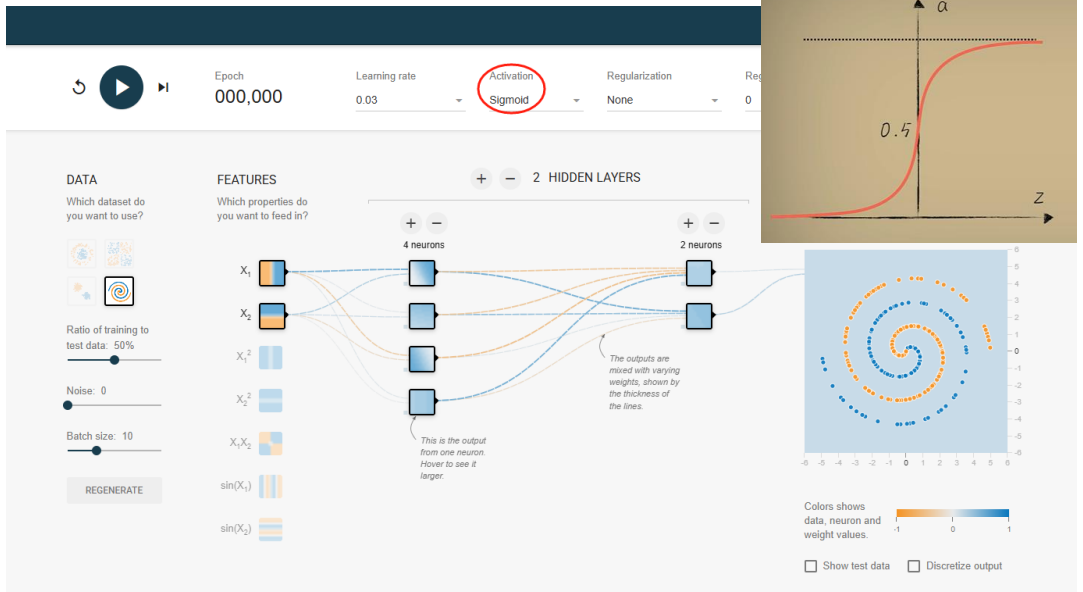
但除了真不错以外，这些**参数怎么设置**，神经元怎么搭建合适就很懵懂了。

而经过前面的学习，我们在对神经网络的工作原理有了相对系统的了解之后，这些东西都将变得明朗起来。

现在我们就来把玩把玩这个经典的入门神器get到一点**调参的感觉。**

地址：https://playground.tensorflow.org/





3 圈圈数据集

深我们试试第一个数据集，这是一个圈圈，中间的蓝色痘痘是有毒的，4周的黄色痘痘是无毒的，这个圈圈数据集实际上，**需要几个神经元**才能**分割出他们的类型。**

不知道大家的思考结果如何要组合出能够**分割出这种痘痘的预测曲面，也就是让它的0.5等高线在俯视图上呈现出一个闭合的圈圈**，至少需要**一层三个神经元的隐藏层**，为什么是三个两个不行吗？

我们简单的来定性分析一下，还是看图说话。

一个神经元的等高线是一条直线，两个是两条直线，但遗憾的是平面中的两条直线是无法完成闭合的，但是三个可以三个神经元是三条直线可以形成一个闭合的形状，把这三个神经元的计算结果再通过一个输出层的神经元进行汇合，通过sigmoid的激活函数修饰一下之后就可以了，如此也就可以解决这种圈圈分布数据的分类问题

4 蚊香数据集

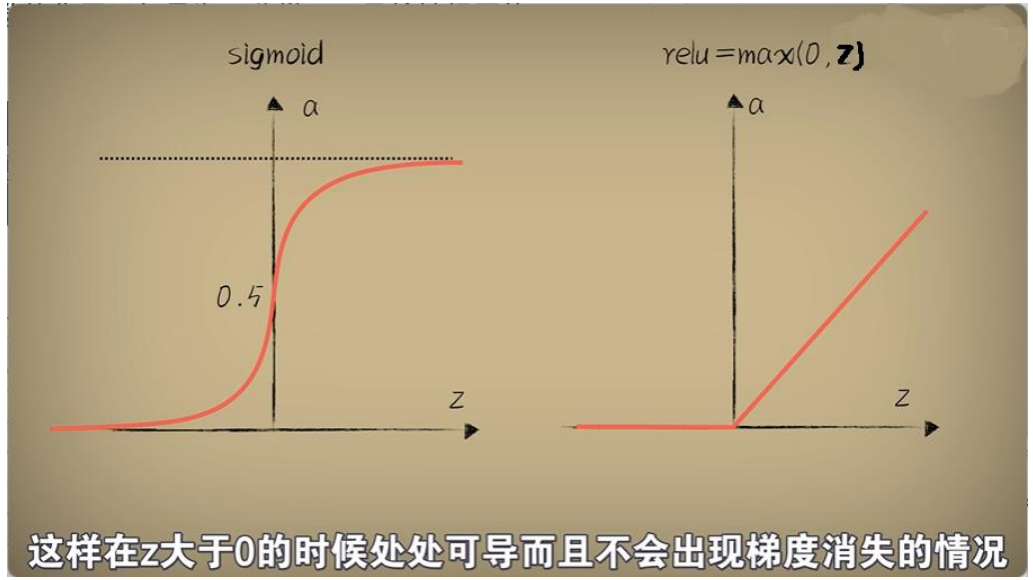
我们再来看最后一种数据集，这是一个蚊香一样的螺旋型数据集，极很明显只有**一层包含三个神经元的隐藏层的神经网络，在这种复杂的数据集上将表现得十分疲软**，那我们就用更深神经元数量以更多的网络来尝试一下。

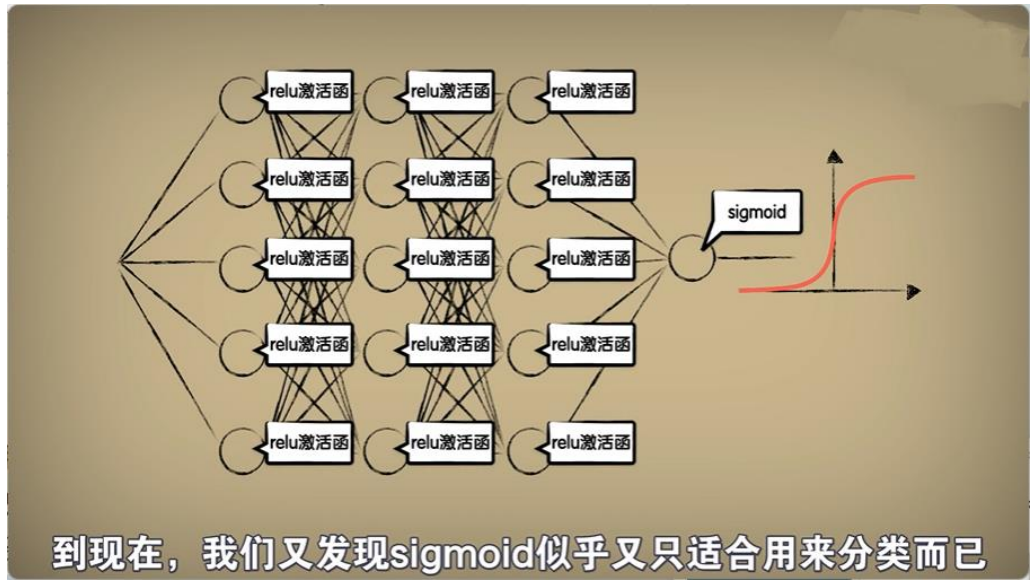
按照我们说的一般的深度网络只超过3层，那我们就把隐藏层的数量增加到3层，每个隐藏层用4个神经元运行一下，唉好像没有反应，预测结果随着训练动也不动，这是因为我们的**老伙计sigmoid的激活函数的问题。**

我们之前说之所以选择sigmoid的函数作为激活函数，那是因为相比于**阶跃函数，sigmoid的函数处处可导，而且导数处处不为0**，这样在反向传播的时候，我们知道使用梯度下降算法计算函数的导数，然后利用这个导数去修正参数，**但是我们并没有说sigmoid的其实有一个很严重的问题**，在sigmoid的中心位置附近，这没有啥问题，但如果一旦进入了sigmoid的函数，远离中心点位置，比如这里虽然仍旧可导导数仍旧不为0，但是导数却极其的小，这样梯度下降就很难进行。

距离中心越远的位置这个导数越小离也就是所谓的梯度消失的问题，越深的神经网络就越容易出现这种梯度消失的问题，所以这个网络变得很难训练，所以目前人们普遍不在使用sigmoid的激活函数。

而经常使用一种叫做**Relu的激活函数。**





5 Relu

这是一个分段函数，在线性结果z大于0的时候，输出值为z；

在z小于0的时候固定为0，这样在z大于0的时候处处可导，而且不会出现梯度消失的情况。
在z等0的时候，从数学上来看，这个折点确实不可导，但一般很难恰好遇到这个点，真要是遇到了那就特殊处理一下，比如认为这一点的导数是0或者1或者一个很小值，比如0.01，**当然如果陷入了z小于0的部分，很有可能导致这个神经元死亡，也就是所谓的Dead Relu Problem。**

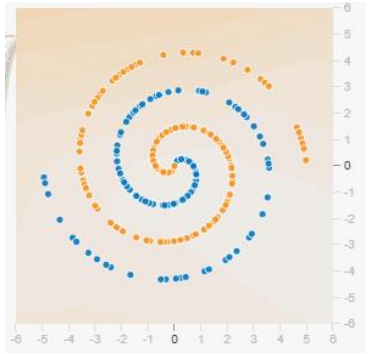
因为根据反向传播中的链式求导法则，如果激活函数的导数是0，那么就锁定了这个神经元上的参数梯度为0，权重无法更新，所以人们又提出了一种改进版的value函数，**leaky ReLU在z小于0的时候输出不再是0**，而是一个缓缓倾斜的直线，如此就可以避免死亡value问题。**但有趣的是当神经网络比较复杂隐藏层比较深的时候，人们通过实践得到了经验，往往是直接使用ReLU激活函数往往也有很好的效果。**当然关于梯度消失的问题，激活函数只是一个方面，还有很多因素会导致这个现象，目前也有很多相关的研究，**但按照经验大多数时候使用ReLU函数都会有不错的效果，所以现在最为流行的激活函数还ReLU。**

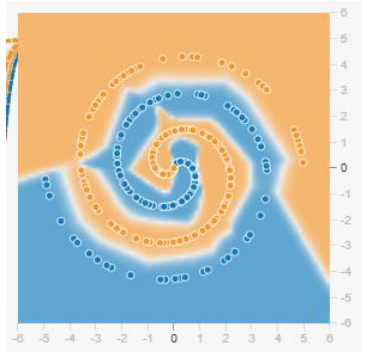
实际上对于隐藏层的神经元，如果没有其他想法或者特殊的需求，**relu应该作为激活函数的第一选择。**当然在最后的输出层，**因为sigmoid的上下线刚好在0~1之间很适合做分类**，所以输出层的激活函数还是选择sigmoid。到此我们从为了分类而引入sigmoid的激活函数，然后发现激活函数的作用不仅是为了分类，而且给神经网络注入了灵魂，让它成为了能够预测复杂问题的非线性系统。

而到现在我们又发现sigmoid似乎只似乎适合用来做分类而已，当然我们已经有了其他更好的激活函数，给神经网络注入了更有趣的灵魂，兜兜转转之间sigmoid的函数也算完成了他的使命。

终于我们在这个螺旋形分布的数据集上取得了非常不错的分类结果，不过我们来看一下这个网络，**每个神经元上的参数都被大大小小的调节成为了不同的数值，每个神经元也产生了对输入不同的中间抽象和理解，到此我们似乎已经没有什么精确的数学手段去分析每个神经元到底是在抽象什么，在理解什么了，**

所以欢迎开始【炼丹】。





扎根场景生态 科技驱动未来

先行通数字科技有限公司

Xianxingtong Digital Technology Co., Ltd