神经网络数学推导

0.前置知识：线性关系和非线性关系

线性关系指输入特征（自变量）与输出（因变量）之间的关系可以通过线性组合（加权和）来描述，数学上表现为一次函数形式。任何线性关系都可以通过一次函数解释。

比如，y = 2x，这里 y 和 x 之间存在线性关系

输入特征与输出之间的关系无法通过简单线性组合描述，需借助更高阶项、交互项或非线性函数。

比如，在二维平面下，如果自变量 x 和因变量 y 在图上呈现为一个抛物线，我们无法使用一次函数来解释，因为一次函数在二维空间中只能是一条直线。所以这里 x 和 y 中间存在非线性关系。

但是我们用二次函数就可以解决了

所以，在这个例子中，二次函数可以体现 x 和 y 的非线性关系。

附加：线性可分性：

线性可分性指的是存在一个超平面（在二维中是直线，定义见《机器学习常用术语》）能够将不同类别的点完全分开。

1.线性变换

（注意：向量和矩阵从 0 开始数，不是 1）

我们现在有一个基础的前馈神经网络（最简单的）

我们有一个输入向量 x，通过输入层进入我们的神经网络

随后，x 开始通过神经网络中的隐藏层

每一层隐藏层都会对 x 通过矩阵运算进行变换，其公式符合：

$$y = wx + b$$

其中，y 是这一层的输出

w 是这一层的权重(weight)

b 是这一层偏置(bias)

这里的 w 是矩阵

b 是向量

举个例子：

假设我们的 x 是这样：$2 \quad 3$

假设我们的 w 是这样：$\begin{matrix} 1 & 0.5 & -1 \\ 0.5 & 2 & -0.5 \end{matrix}$

假设我们的 b 是这样：$0.1 \quad -0.2 \quad 0$

则：

y[0] = (x[0] * w[0][0]) + (x[1] * w[1][0]) + b[0]

= (2 * 1) + (3 * 0.5) + 0.1

= 2 + 1.5 + 0.1 = 3.6

y[1] = (x[0] * w[0][1]) + (x[1] * w[1][1]) + b[1]

= (2 * 0.5) + (3 * 2) + (-0.2)

= 1 + 6 - 0.2 = 6.8

y[2] = (x[0] * w[0][2]) + (x[1] * w[1][2]) + b[2]

= (2 * -1) + (3 * -0.5) + 0

= -2 - 1.5 + 0 = -3.5

所以这一层的输出 y 是：3.6　6.8　−3.5

然后 y 会通过一个激活函数 f 的处理得到 f(y)，f(y)又会被作为输入数据进入下一个神经元，又得到一个新的输出（激活函数见下文）

设我们的新输出为 z，则：

$$z = w'f(y) + b'$$

这里 w'是当前层的权重，b'是当前层的偏置

所以我们不难推出，对于第 i 层隐藏层：

$$y^{(i)} = W^{(i)}x^{(i-1)} + b^{(i)}$$

其中$W^{(i)}$是第 i 层的权重矩阵

$b^{(i)}$是第 i 层的权重向量

$x^{(i-1)}$是这一层的输入，是上一层的输出通过激活函数后的结果

$y^{(i)}$是本层线性变换后的结果

2.激活函数

从上面提到的线性变换不难得出，如果每一层都只有线性变换，无论经过多少层，这个神经网络对于输入数据的变换也只能是线性的。这样做有一个局限性，无法捕捉特征间的交互作用或复杂模式（如异或问题、螺旋数据分布）。

比如，著名的异或问题：

有两个数(x, y)

我们给不同的 x 和 y 分类--分为类别 0 或者类别 1

(0,0): 类别 0

(0,1): 类别 1

(1,0): 类别 1

(1,1): 类别 0

把上面这四个点画在平面直角坐标系里面

你觉得你能用一条直线划分这些点，然后说“直线上面的点都属于类别 0，直线下面的点都属于类别 1”，或者“直线上面的点都属于类别 1，直线下面的点都属于类别 0”吗？

显然不行

所以异或问题是线性不可分的

通过线性变换，我们不能解决异或问题

还有许多其他问题不能通过线性变换解决，所以我们需要引入非线性——激活函数（很多资料用字母 σ，读作 sigma，来表示）

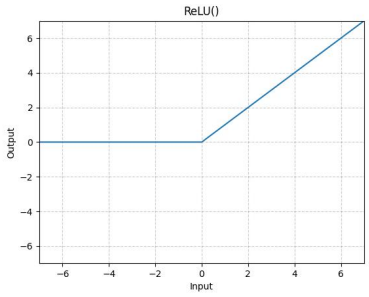
就是干这个的

初级阶段常见的激活函数类型：（注意，以下的 σ (x)和 x 都是向量）

(1)ReLU 函数：

这可以过滤掉输入部分小于等于 0（也就是经过线性变换后，看起来“不重要”的地方）

$$\sigma(x) = \max(0, x)$$



(2)Softmax 函数：

Softmax 不只是单个值的操作，而是对一整组数“统筹安排”。它先把每个值指数放大，再统一归一化成一个“概率分布”。

所以 Softmax 更像是一场选秀：谁的 xi 原始值越大，指数值越大，最后得到的“选中概率”越高。

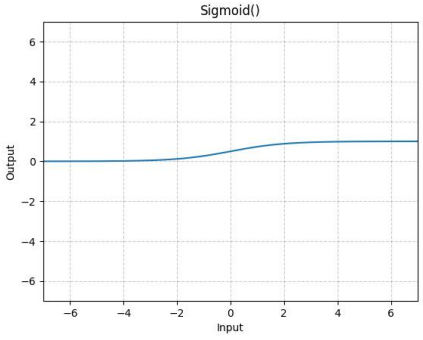
在做分类模型时，softmax 往往被放在最后一层隐藏层的位置，用来输出经过神经网络一系列变换后模型认为各个类别分别所对应的概率，就像是数字识别，最后的 softmax 输出数字 0 到 9 每个数字对应的概率

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

(3)Sigmoid 函数：

Sigmoid 就像一个“压缩器”，它把任意输入的值都压缩到(0, 1)之间

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



所以，神经网络每一层的输出可以表示为：

$$\sigma(W^{(i)}x^{(i-1)} + b^{(i)})$$

3.神经元的数学解释

假设某层的权重矩阵为 W∈R^{m*n}，即一个 m 行 n 列的矩阵——其中所有元素都属于实数，其中：

行数 m：表示当前层的神经元数量；

列数 n：表示前一层的输出维度（即前一层的神经元数量或输入向量的维度）。

例如，权重矩阵是 3 行 2 列（3×2），说明：

当前层有 k=3 个神经元；

前一层的输出是 m=2 维向量（或前一层有 2 个神经元）。

4. 优化方法——梯度下降（gradient descent）

模型会通过优化的方法，来调整 w 和 b，从而尝试让我们的损失函数 L（定义见《机器学习常用术语》）越来越小

偏导数：我们已经知道导数代表的是函数在某一点的“斜率”，也就是你站在函数图像上某点，前后左右哪边是“上坡”还是“下坡”。y 对 x 的导数是，表示 x 如何影响 y：

$$\frac{dy}{dx}$$

但是，现实世界不太可能只有 x 一个变量。在神经网络中，损失函数 L 可能同时依赖很多参数，每一层都会有权重和偏置，那如果有 n 层的话就会有 w1, w2, ... ,wn 和 b1, b2, ... , bn。而且 w 都还是矩阵，矩阵中包含的每一个标量都会对 L 产生影响。每个参数都需要调整。

这时候，我们就不能再用普通的导数了，而要用偏导数。

有一点要注意的是偏导数里面我们不再用字母 d，而是字母∂

偏导数就是：

“只动其中一个变量，看看函数值变化得有多快。”

比如说：

$$L = w_1^2 + 3w_2 + 5$$

这是一个损失函数，L 是损失函数，w_1 和 w_2 是两个参数。

这时用到偏导数：

$$\frac{\partial L}{\partial w_1} = 2w_1$$

$$\frac{\partial L}{\partial w_2} = 3$$

在第一个等式中，因为只关心 w1，所以把 w2 视为常数

在第二个等式中，因为只关心 w2，所以把 w1 视为常数

然后导出来就行了

你可以理解成——偏导数就是对每个“调节旋钮”的灵敏度测量。

在模型的参数优化中，

我们如果对所有参数（所有的 w 和所有的 b）都求一次偏导，就得到了偏导数的组合：

$$\nabla_w L = \frac{\partial L}{\partial \mathbf{W}} = \left[\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}, \ldots, \frac{\partial L}{\partial w_n}\right]$$

这个东西就叫做 梯度（Gradient），表示所有的 w 对 L 如何影响（注意这里的w_i是矩阵，矩阵可以求偏导），

同样地，

$$\nabla_b L = \frac{\partial L}{\partial \mathbf{b}} = \left[\frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial b_2}, \ldots, \frac{\partial L}{\partial b_n}\right]$$

表示所有的 b 对 L 如何影响

梯度告诉你在“所有变量维度上往哪边走，下降最快”。如果不理解这句话，请往下看

梯度下降的核心公式是：

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \eta \cdot \frac{\partial L}{\partial \mathbf{W}^{(t)}}$$

$$\mathbf{b}^{(t+1)} = \mathbf{b}^{(t)} - \eta \cdot \frac{\partial L}{\partial \mathbf{b}^{(t)}}$$

η是学习率（learning rate），这个是你自己在写代码的时候填写的，代表一步走多远；

梯度下降会进行很多次，模型会先用一个 w 和 b 试试看，看看 L 是多少，然后根据 L 在此刻对 w 和 b 的梯度，再对 w 和 b 进行调整，如此重复很多次。所以以上的公式表示，在第 t+1 次优化时，w 和 b 相比第 t 次时进行了哪些调整

想象一个开口向上的函数，最低点的地方导数为 0

那么这里也用了相似的思想，对于 w 和 b，我们找到一个最低点的时候，$\frac{\partial L}{\partial \mathbf{b}^{(t)}}$和$\frac{\partial L}{\partial \mathbf{W}^{(t)}}$等于 0，L 在这个位置找到了最低点，那么 w 和 b 都走不动了，优化也就完成了

但是想想会有什么问题？

损失函数不一定只有一个梯度为 0 的地方