交叉熵损失函数

1.从方差代价函数说起

代价函数经常用方差代价函数（即采用均方误差MSE），比如对于一个神经元（单输入单输出，sigmoid函数）,定义其代价函数为：

$$C = \frac{(y-a)^2}{2}$$

其中y是我们期望的输出，a为神经元的实际输出【 a=σ(z), where z=wx+b 】。

在训练神经网络过程中，我们通过梯度下降算法来更新w和b，因此需要计算代价函数对w和b的导数：

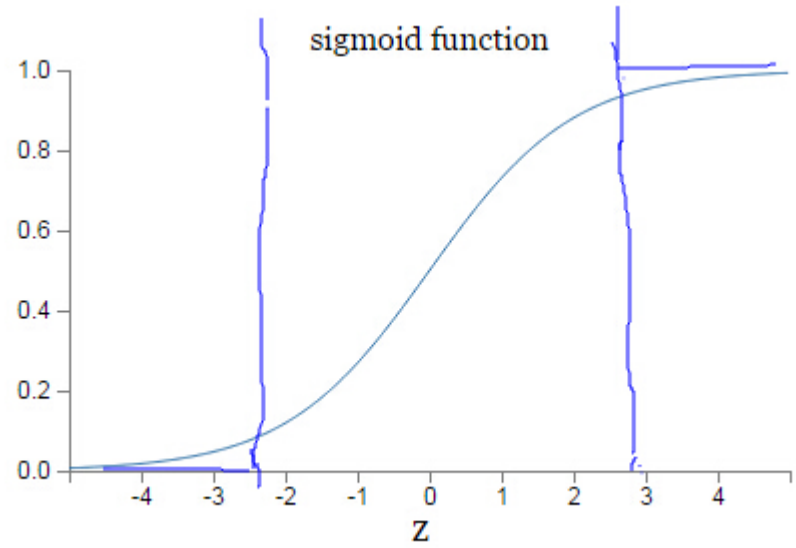
$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z),$$

然后更新w、b：

w <—— w - η* ∂C/∂w = w - η * a *σ'(z)

b <—— b - η* ∂C/∂b = b - η * a * σ'(z)

因为sigmoid函数的性质，导致σ'(z)在z取大部分值时会很小（如下图标出来的两端，几近于平坦），这样会使得w和b更新非常慢（因为η * a * σ'(z)这一项接近于0）。



2.交叉熵代价函数（cross-entropy cost function）

为了克服这个缺点，引入了交叉熵代价函数（下面的公式对应一个神经元，多输入单输出）：

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)]$$

其中y为期望的输出，a为神经元实际输出【a=σ(z), where z=∑Wj*Xj+b】

与方差代价函数一样，交叉熵代价函数同样有两个性质：

- 非负性。（所以我们的目标就是最小化代价函数）
- 当真实输出a与期望输出y接近的时候，代价函数接近于0.(比如y=0，a～0；y=1，a~1时，代价函数都接近0)。

另外，它可以克服方差代价函数更新权重过慢的问题。我们同样看看它的导数：

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y).$$

$$\frac{\partial C}{\partial b} = \frac{1}{n}\sum_x (\sigma(z)-y).$$

可以看到，导数中没有σ'(z)这一项，权重的更新是受σ(z)−y这一项影响，即受误差的影响。所以当误差大的时候，权重更新就快，当误差小的时候，权重的更新就慢。这是一个很好的性质。

3.总结

- 当我们用sigmoid函数作为神经元的激活函数时，最好使用交叉熵代价函数来替代方差代价函数，以避免训练过程太慢。
- 不过，你也许会问，为什么是交叉熵函数？导数中不带σ'(z)项的函数有无数种，怎么就想到用交叉熵函数？这自然是有来头的，更深入的讨论就不写了，少年请自行了解。
- 另外，交叉熵函数的形式是−[ylna+(1−y)ln(1−a)]而不是 −[alny+(1−a)ln(1−y)]，为什么？因为当期望输出的y=0时，lny没有意义；当期望y=1时，ln(1-y)没有意义。而因为a是sigmoid函数的实际输出，永远不会等于0或1，只会无限接近于0或者1，因此不存在这个问题。