交叉熵代价函数

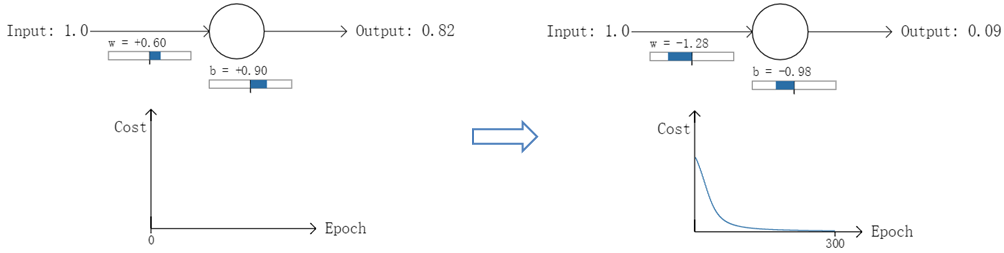
本文是《Neural networks and deep learning》概览中第三章的一部分，讲machine learning算法中用得很多的交叉熵代价函数

1. 从方差代价函数说起

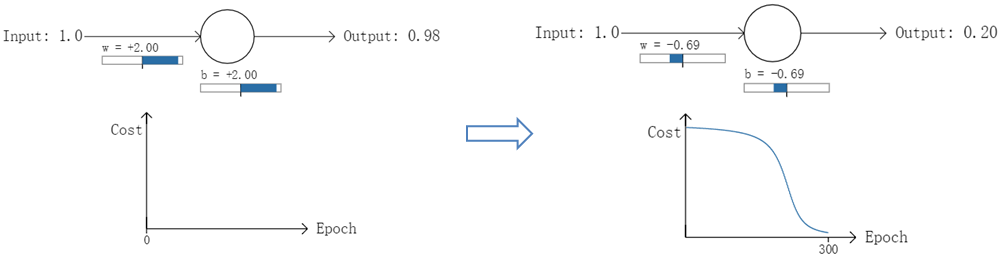
交叉熵代价函数（Cross-entropy cost function）是用来衡量人工神经网络（ANN）的预测值与实际值的一种方式。与二次代价函数相比，它能更有效地促进ANN的训练。在介绍交叉熵代价函数之前，本文先简要介绍二次代价函数，以及其存在的不足。

ANN的设计目的之一是为了使机器可以像人一样学习知识。人在学习分析新事物时，当发现自己犯的错误越大时，改正的力度就越大。比如投篮：当运动员发现自己的投篮方向离正确方向越远，那么他调整的投篮角度就应该越大，篮球就更容易投进篮筐。同理，我们希望：ANN在训练时，如果预测值与实际值的误差越大，那么在反向传播训练的过程中，各种参数调整的幅度就要更大，从而使训练更快收敛。然而，如果使用二次代价函数训练ANN，看到的实际效果是，如果误差越大，参数调整的幅度可能更小，训练更缓慢。

以一个神经元的二类分类训练为例，进行两次实验（ANN常用的激活函数为sigmoid函数，该实验也采用该函数）：输入一个相同的样本数据x=1.0（该样本对应的实际分类y=0）；两次实验各自随机初始化参数，从而在各自的第一次前向传播后得到不同的输出值，形成不同的代价（误差）：



实验一：第一次输出值为0.82



实验二：第一次输出值为0.98

在实验1中，随机初始化参数，使得第一次输出值为0.82（该样本对应的实际值为0）；经过300次迭代训练后，输出值由0.82降到0.09，逼近实际值。而在实验2中，第一次输出值为0.98，同样经过300迭代训练，输出值只降到了0.20。

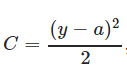
从两次实验的代价曲线中可以看出：实验1的代价随着训练次数增加而快速降低，但实验2的代价在一开始下降得非常缓慢；直观上看，初始的误差越大，收敛得越缓慢。

其实，误差大导致训练缓慢的原因在于使用了二次代价函数。二次代价函数的公式如下：

http://img.blog.csdn.net/20160402180717102

其中，C表示代价，x表示样本，y表示实际值，a表示输出值，n表示样本的总数。

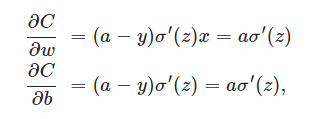
代价函数经常用方差代价函数（即采用均方误差MSE），比如对于一个神经元（单输入单输出，sigmoid函数）,定义其代价函数为：



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 】。

目前训练ANN最有效的算法是反向传播算法。简而言之，训练ANN就是通过反向传播代价，以减少代价为导向，调整参数。参数主要有：神经元之间的连接权重w，以及每个神经元本身的偏置b。调参的方式是采用梯度下降算法（Gradient descent），沿着梯度方向调整参数大小。

在训练神经网络过程中，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：

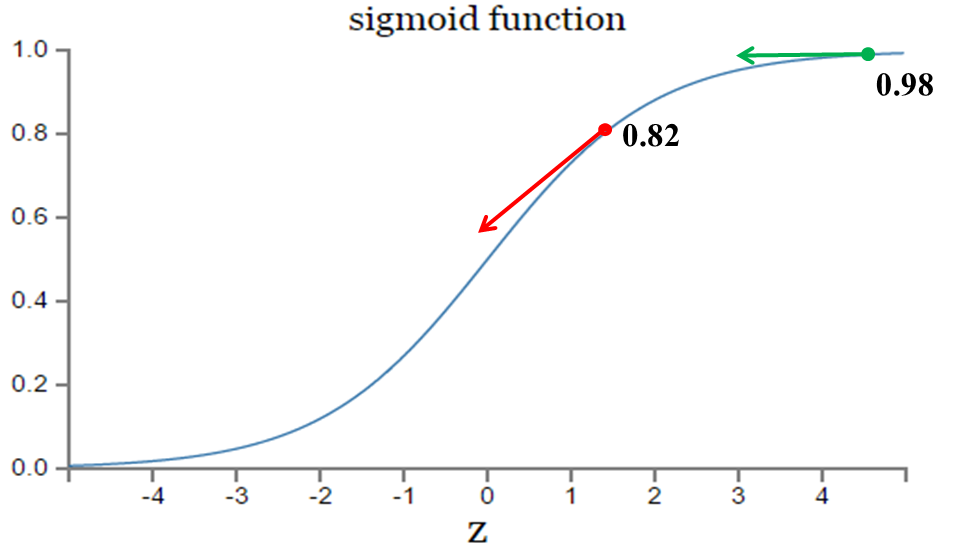


其中，z表示神经元的输入，表示激活函数。从以上公式可以看出，w和b的梯度跟激活函数的梯度成正比，激活函数的梯度越大，w和b的大小调整得越快，训练收敛得就越快。然后更新w、b：

w <—— w - η\* ∂C/∂w = w - η \* a \*σ′(z)

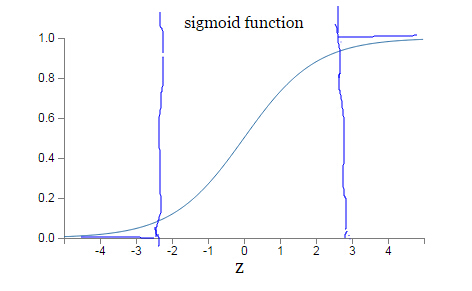
b <—— b - η\* ∂C/∂b = b - η \* a \* σ′(z)

经网络常用的激活函数为sigmoid函数，该函数的曲线如下所示：



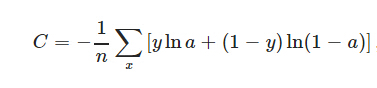
如上图所示，实验2的初始输出值（0.98）对应的梯度明显小于实验1的输出值（0.82），因此实验2的参数梯度下降得比实验1慢。这就是初始的代价（误差）越大，导致训练越慢的原因。与我们的期望不符，即：不能像人一样，错误越大，改正的幅度越大，从而学习得越快。

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）。



二、交叉熵代价函数（cross-entropy cost function）

由于激活函数使用sigmoid这样的函数有很多优点，在不替换激活函数的前提下，为了克服这个缺点，替换掉二次代价函数，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）：

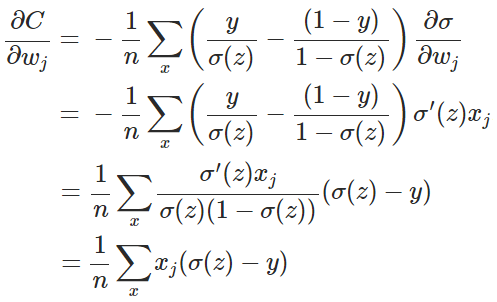


其中，x表示样本，n表示样本的总数，y为期望的输出，a为神经元实际输出【a=σ(z), where z=∑Wj\*Xj+b】。

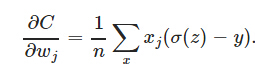
与方差代价函数一样，交叉熵代价函数同样有两个性质：

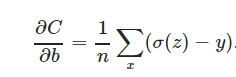
* 非负性。（所以我们的目标就是最小化代价函数）。
* 当真实输出a与期望输出y接近的时候，代价函数接近于0.(比如y=0，a～0；y=1，a~1时，代价函数都接近0)。

另外，它可以克服方差代价函数更新权重过慢的问题。那么，重新计算参数w的梯度：



得到导数，





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。即（具体证明见附录），

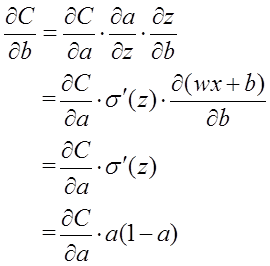
http://img.blog.csdn.net/20160402172758429

因此，w的梯度公式中原来的被消掉了；另外，该梯度公式中的表示输出值与实际值之间的误差。所以，当误差越大，梯度就越大，参数w调整得越快，训练速度也就越快。同理可得，b的梯度为：  
http://img.blog.csdn.net/20160402173528448

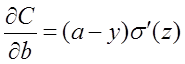
实际情况证明，交叉熵代价函数带来的训练效果往往比二次代价函数要好。

三、交叉熵代价函数是如何产生的

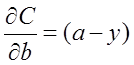
以偏置b的梯度计算为例，推导出交叉熵代价函数：



在第1小节中，由二次代价函数推导出来的b的梯度公式为：



为了消掉该公式中的，我们想找到一个代价函数使得：



即：



对两侧求积分，可得：

http://img.blog.csdn.net/20160402180140959

而这就是前面介绍的交叉熵代价函数。

四、各种形式的熵函数，KL距离

1、自信息量

I(x)=-log(p(x))

其他依次类推。

2、离散变量x的熵

H(x)=E(I(x))=-$\sum\limits\_{x}{p(x)lnp(x)}$

3、连续变量x的微分熵H(x)=E(I(x))=-$\int{p(x)lnp(x)dx} $

4、条件熵H(y|x)=-$\int\int{p(x,y)lnp(y|x)dydx}$

5、两个变量X和 Y 的联合熵定义为：

H(X,Y)=-$\int\int{p(x,y)lnp(x,y)dxdy}$

H(x,y)=H(y|x)+H(x)

若x，y独立，H(x,y)=H(x)+H(y),此时对x的了解不能增进对y的了解。

6、交叉熵Cross Entropy

H(p;q)=-$\int{p(x)lnq(x)dx}$

很少见，通常使用KL距离

Kullback-Leibler divergence:KL（p||q）=-$\int{p(x)lnq(x)dx}-(-\int{p(x)lnp(x)dx})$=H(p)+H(p;q)=-$\int{p(x)ln{\frac{q(x)}{p(x)}}dx}$

p=q时，KL(p||q)=0,H(p;q)=H(p)

交叉熵与kl距离相差一个H(p)

当p未知而q已知时，通过改变KL中的p、q的位置，可以减少未知量，便于计算相似度。

交叉熵是一种万能的Monte-Carlo技术，常用于稀有事件的仿真建模、多峰函数的最优化问题。交叉熵技术已用于解决经典的旅行商问题、背包问题、最短路问题、最大割问题等。这里给一个文章链接：A Tutorial on the Cross-Entropy Method

交叉熵算法的推导过程中又牵扯出来一个问题：如何求一个数学期望？常用的方法有这么几种：

概率方法，比如Crude Monte-Carlo

测度变换法change of measure

偏微分方程的变量代换法

Green函数法

Fourier变换法

在实际中变量X服从的概率分布h往往是不知道的，我们会用g来近似地代替h----这本质上是一种函数估计。有一种度量g和h相近程度的方法叫 Kullback-Leibler距离，又叫交叉熵,通常选取g和h具有相同的概率分布类型（比如已知h是指数分布，那么就选g也是指数分布）----参数估计，只是pdf参数不一样（实际上h中的参数根本就是未知的）。

基于期望交叉熵的特征项选择

CE(w)=$\sum\limits\_{i}p(c\_{i}|w)log\frac{p(c\_{i}|w}{p(c\_{i}}$

p(ci|w)表示在出现词条w时文档属于类别ci的概率。

交叉熵反应了文本类别的概率分布与在出现了某个词条的情况下文本类别的概率分布之间的距离。词条的交叉熵越大，对文本类别分布影响也就越大。所以选CE最大的K个词条作为最终的特征项。

互信息Mutual Informantion

yj对xi的互信息定义为后验概率与先验概率比值的对数。

I(x,y)=log$\frac{p(x|y)}{p(x)}=I(x)-I(x|y)$

互信息越大，表明y对于确定x的取值的贡献度越大。

系统的平均互信息

I(X,Y)=H(X)-H(X|Y)=H(Y)-H(Y|X)

可见平均互信息就是信息增益！

I(X,Y)=KL(p(x,y)||p(x)p(y))=-$\int\int{p(x,y)ln(\frac{p(x)p(y)}{p(x,y)})dxdy}$

互信息在特征选择中的应用

词条w与类别ci的互信息为

MI(w,c)=log$\frac{p(w|c)}{p(w)}$

p(w)表示出现w的文档点总文档数目的比例，p(w|ci)表示在类别ci中出现w的文档点总文档数目的比例。

对整个系统来说，词条w的互信息为

$MI\_{avg}(w,c)=\sum\limits\_{i}p(c)log\frac{p(w|c)}{p(w)}$

最后选互信息最大的前K个词条作为特征项。

五、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。
* 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。

**附录：**

sigmoid函数为：



可证：

