**反向传播算法——MLP二值分类**

**1 采用Sigmoid传导函数的用于二值分类的3层的神经网络**

图2是一个比图1(上篇文章)更加复杂的3层神经网络[[1]](#footnote-1)，复杂性在于隐藏层的节点和输出层的节点都采用sigmoid函数作为激活函数。该神经网络用于二值分类，分类的损失函数采用二值交叉熵损失函数(Binary Cross Entropy Error Function)。

Input Layer Hidden Layer Output Layer

X1

X2

J

图2. 采用sigmoid激活函数的3层神经网络结构

图2的神经网络，其正向传导过程可以用如下公式进行表达。

(1) (2\*1) = (2\*2)(2\*1) +(2\*1)

(2) (2\*1) = (2\*1)，注意是对矩阵的每个元素进行转换

(3) (1\*1) = (1\*2)(2\*1) +(1\*1)

(4) (1\*1)=(1\*1) ，注意是对矩阵的每个元素进行转换

即即 🡪 🡪🡪🡪，最后又构造损失函数。

我们对第一个传导式中的矩阵简写形式进行展开，结合图2以便更加深入理解该传导过程，具体如下。

**2 反向传播算法的基本流程**

基于反向传播算法的神经网络的训练过程，如图3所示。首先随机地初始化网络权重，然后用这些权重和输入x，进行前向传导，获得预测值。计算预测值和实际值之间的误差，计算误差针对各层网络连接权重的导数。根据这个导数，利用更新公式对参数进行更新。更新公式为，其中J为损失函数。

Calculate cost

Predict using w, b, x

Random initialize

W, b

Update w, b using dw & db

Find dJ / dw

& dJ / db

图3. 反向传播的一般流程

**3层的神经网络的前向传导和反向传播**

**3.1 导数计算**

针对图2所示的神经网络，我们计算如下的导数。注意。

,

,

由上述式子可以看出，对神经网络进行训练，求导的任务很繁重。这还是一个仅仅包含少量网络层数和少量神经元的网络，试想一下如果神经网络有100个隐藏层，每个隐藏层有1 000个神经元，计算的复杂度就太高了。

**3.2 优化的导数计算**

我们必须提高求导数的效率，具体办法是在计算的时候，利用已经计算过的…等已有的结果。

针对图2所示的网络，我们尝试进行公式推导，然后把结论推广到任意多层的神经网络(MLP，即多层感知机)。

二值分类器的交叉熵损失函数的形式为。注意即预测值。

我们开始计算(1/n在这里先省略)。

在这个公式中，第一个部分用到了对数函数的导数公式，第二个部分用到了sigmoid函数的导数公式，第三个部分用到了传导式。

我们现在对该式子进行约减

=  **=**，

因为===。

类似地，我们得到，具体如下。

= =

|  |
| --- |
| 注意，这里的公式，并不保证形式的完全正确；请参考后文实例部分给出的正确形式；正确形式的推导，请参考本文同目录下的“misc矩阵求导——MLP神经网络.pptx”文件 |

继续计算和。注意

= =

=

注意即X。此外，我们准备如下的推导式。

=**= (这个公式将在下文通用算法用到)**

现在计算。

= =

= =

**4 N层NLP的前向传导和反向传播**

把上述结论推广到N层的神经网络(这里的N层是除了输入层的)，前向传导和反向传播的具体算法如下。

|  |
| --- |
| .初始化  .设置，L为总的网络层数  .执行如下迭代过程(直到最大迭代次数)  .前向传导  For l=1 to L-1  ，注意为X    保存在内存里，备用  最后  ,即  .计算代价函数J=  .反向传播（如下式子，需要乘上）          =(请参考上文的推导)  for l=L-1 to 1        =  .更新W和b  For l=1 to L |

**5 神经网络实例**

在上述基本原理的基础上，我们结合MNIST数据集，构造一个实际的神经网络，看看前向传导和反向传播过程是如何展开的。

MNIST数据集有60000个样本，10个类别，在这里仅仅选取数字5和数字8的样本，共11272个样本。由于只有两个类别，是一个2值分类问题。每个样本是28×28的图片(矩阵)，这些图片经过转化，变成784个分量的一维向量形式。

我们设计的神经网络为输入层有784个神经元，隐藏层有196个神经元，输出层为1个神经元。

Layer1的前向传导过程具体如下：

|  |
| --- |
| X=(11272, 784) 11272个样本，每个样本784维  =(196, 784)  =(196,1)  =(784, 11272)  =  =(196,784)\*(784, 11272) +(196,1)，注意(196, 11272)的每一列都加(196,1)的  =(196, 11272)  =(196, 11272) |

Layer2的前向传导过程具体如下：

|  |
| --- |
| =(1, 196)  =(1,1)    =(1,196)\*(196, 11272) +(1,1)，注意(1,11727)的每一列都加(1,1)的  =(1,11727)  =(1, 11272) |

Layer2的反向传播过程具体如下：

|  |
| --- |
| =(1,11727)  =(1, 11272) ，注意，矩阵的各个位置相除  =(1, 11272)\* (1, 11272) = (1, 11272)，注意，矩阵的各个位置点乘  =(1, 11272)( 11727, 196) = (1, 196)，注意，是矩阵乘法  [1] =(1, 11272)\*(11272,1)= (1,1)，相当于每行的各列累加  ==(196,1)(1,11272) = (196, 11272) |

Layer1的反向传播过程具体如下：

|  |
| --- |
| = (196, 11272)\* (196, 11272) = (196, 11272)，注意，矩阵的各个位置点乘  = (196, 11272)( 11272, 784) = (196\*784)，注意，是矩阵乘法  [1] = (196, 11272)\*(11272,1) =(196,1)，相当于每行的各列累加 |

具体的代码实现，读者可以参考脚注所注的参考文献。需要注意的是，损失函数是有一个的元素的，在上述前向传播和反向传播的公式中没有体现，在计算各个网络隐藏层的和的时候，考虑除以n即样本数量即可。

|  |
| --- |
| 注意上述矩阵的导数(该转置的转置)是正确的，正确形式的推导，请参考本文同目录下的“misc矩阵求导——MLP神经网络.pptx”文件 |

**6 逻辑斯蒂回归与神经网络的联系**

对于逻辑斯蒂回归来讲，我们可以把它看作只有一个隐藏层的神经网络。它的前向传导和反向传播过程如图4所示。

X1

W1

dw1

X2 dw2

W2 dz d J (cost function)

图4. 逻辑斯蒂回归与梯度下降求解

前向传导公式为：+b, =A=。

计算损失函数对W和b的导数，具体如下。注意这里使用的损失函数是联合概率分布。

, 。

得到导数以后，采用梯度下降算法对参数进行调整，不断迭代该过程。

, 。

**7 线性回归与神经网络的联系**

对于线性回归来讲，我们可以把它看作只有一个隐藏层的神经网络，而且它的传导函数是线性的。它的前向传导和反向传播过程如图5所示。

X1

W1

dw1

X2  dw2

W2  dz d J (cost function)

图5. 线性回归与梯度下降求解

前向传导公式为：, =Z。

计算损失函数对W和b的导数，具体如下。注意线性回归一般使用的损失函数J是均方差。

, 。

得到导数以后，采用梯度下降算法对参数进行调整，不断迭代该过程。

, 。

1. https://www.adeveloperdiary.com/data-science/machine-learning/understand-and-implement-the-backpropagation-algorithm-from-scratch-in-python/. [↑](#footnote-ref-1)