# 1重点问题及推导

1.1神经元

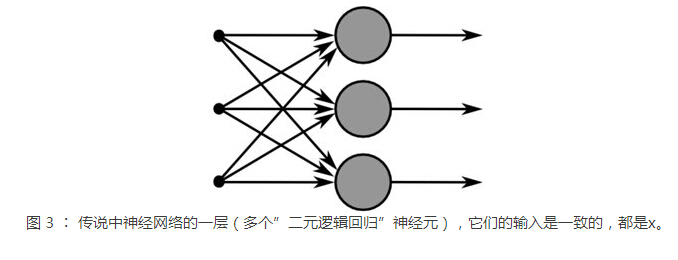
Sigmoid函数：

常见的激活函数.输入n维向量，经过处理后输出一个标量激活结果。

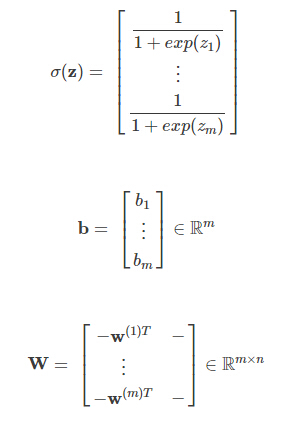
具体处理方法：

2捕获

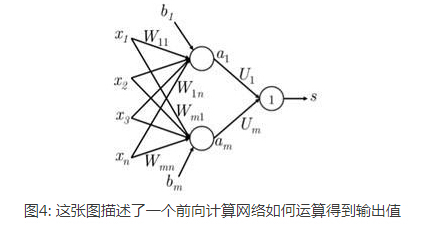
单层神经元



我们分别用{w(1),⋯,w(m)}，{b1⋯,bm}和{a1⋯,am} 来表示m个神经元的权重向量，偏移量以及激励输出，则有一下的结果：

且z=Wx+b

1.2前向计算



如果我们用4维词向量表示这些词，且用一个5词窗口作为输入（就像上面这个例子），那输入的变量就是x∈R20. 如果在隐藏层中使用8个sigmoid神经元，并且由其激励输出生成1个得分，我们就有W∈R20，b∈R8，U∈R8×1，s∈R

z=Wx+b   
a=σ(z)   
s=UTa

1.3最大化间隔目标函数

我们把一个正确标记的词窗 “Museums in Paris are amazing”(这里Paris是命名实体)的得分记做 s， 而错误标记的词窗“Not all Museums in Paris”(这里Paris不是命名实体)的得分记作sc。

目标函数就是要最大化 (s−sc) 或者最小化（sc−s）

只有在 sc>s=>（sc−s）>0 的时候才计算这个函数的值。因为当正确标记的词窗得分比错误标记的词窗得分高的时候，我们认为是满足要求的，并没有误差，我们只关心错误标记的词窗比正确标记的词窗得分高了多少，它代表了误差的程度。

优化目标变成： 

minimizeJ=max(sc−s,0)

我们希望那些被正确标记的词窗得分不仅要比错误标记的词窗得分高，还希望至少高出一个取值为正的间隔Δ。 换句话说， 我们希望在 （s−sc<Δ）的时候就开始计算误差值，而不是等到（s−sc<0）。

优化目标为 ︰ 

minimizeJ=max(Δ+sc−s,0)

1.4 反向传播训练法（未向量化的逐元素形式）

从链式法则的角度和从误差分配/分散的角度来理解反向传播。

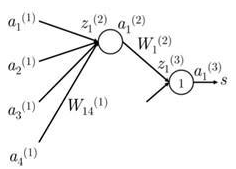
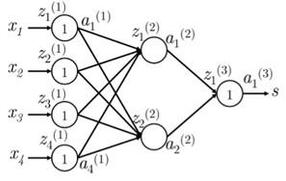
1. 利用导数的链式法则来计算损失函数（正向计算求得）在每个模型参数上的梯度。

误差δ(k)从k层传播到k−1层的过程就等价于求目标函数高1阶的导数。

反向传播误差δ(k)i其实就是最终的目标函数对于第k层上第i个激励输出值z(k)i的导数。其核心思想就是当我们要求目标函数关于Wk−1层的导数时， 因为第k层上只有z(k)的计算被涉及到， 所以可以把后者写成关于前者的函数，接着利用导数的链式法则，得到导数。

推导如下：

对于网络（左）和其局部图（右）

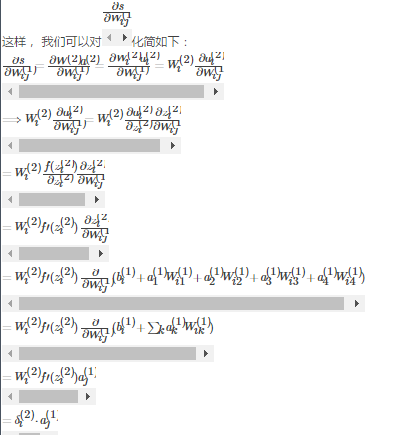


假设目标函数 J取正值，我们希望更新权重参数W14，我们注意到这里W14只在计算z(2)1 和 a(2)1时出现。

a(2)1在之后的正向计算中和 W(2)1相乘进而参与到分类得分的计算中。我们从最大化边界损失的形式看到：

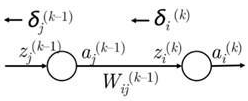


为了简化计算， 只考虑∂J/∂W(1)ij。这样，可以对∂s/∂W(1)ij进行化简。后续推导过程如图（图在网上找的）：



这个梯度最终可以简化为δ(2)i⋅a(1)j 形式。 这里δ(2)i 是反向逆推到第2层上第i个神经元的误差。 a(1)j则与Wij相乘后输入到第2层上第i个神经元的计算中。

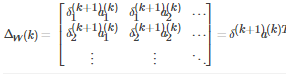
偏移量的更新 对于第k层上第i个神经元偏移量的梯度就是δ(k)i。

δ(k)到δ(k−1)反向传播的一般化步骤如图 

1.6 反向传播训练（向量化的形式）

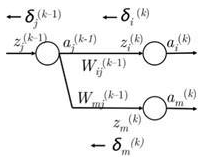
介绍了如何把模型中每个参数的梯度的计算向量化及矩阵化。

我们可以把误差信息对于整个矩阵W(k)的梯度表示成以下形式:



于是，梯度就写成(从下一层)反向传播过来的误差和（从这一层）参与到前向计算中的激励输出的外积。

向量化的计算 δ(k)：

如图

对于，

可以推广矩阵形式 

（这种乘法Hadamard积暑假学长报告中提到过）

2 神经网络：技巧和窍门

# 2引申出来的问题

# 3自己没有理解的问题

# 4阅读论文的重要内容

# 5每个人的工作总结

# 6附件：代码和论文