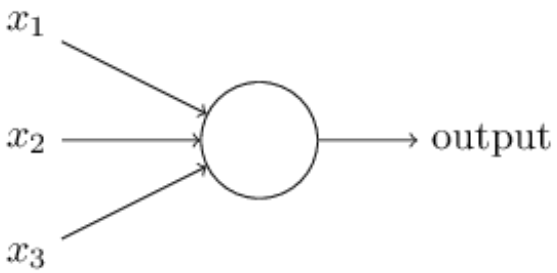
(5条消息)深度学习（一）：DNN前向传播算法和反向传播算法_anshuai_aw1的博客-CSDN博客_前向传播和反向传播

一、深度神经网络（DNN）模型

深度神经网络（Deep Neural Networks， 以下简称DNN）是深度学习的基础，而要理解DNN，首先我们要理解DNN模型，下面我们就对DNN的模型与前向传播算法做一个总结。

1.1 从感知机到神经网络

感知机的模型大家都比较熟悉，它是一个有若干输入和一个输出的模型，如下图:



输出和输入之间学习到一个线性关系，得到中间输出结果：
z=i=1∑m wi xi +b

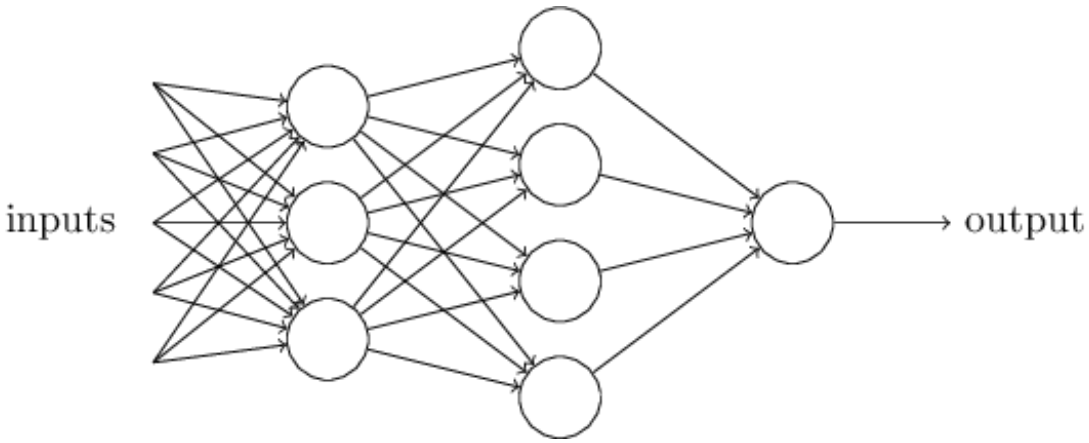
接着是一个神经元激活函数:
sign(z)={−11 z<0z≥0

从而得到我们想要的输出结果1或者-1。

这个模型只能用于二元分类，且无法学习比较复杂的非线性模型，因此在工业界无法使用。

而神经网络则在感知机的模型上做了扩展，总结下主要有三点：

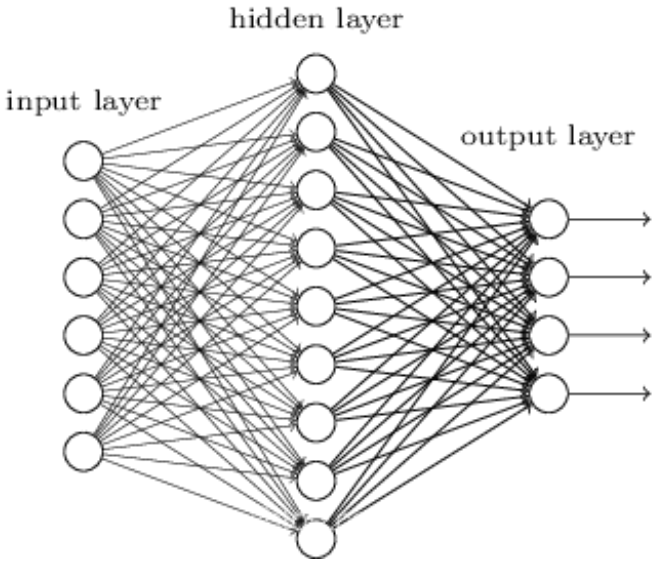
1）加入了隐藏层，隐藏层可以有多层，增强模型的表达能力，如下图实例，当然增加了这么多隐藏层模型的复杂度也增加了好多。



2）输出层的神经元也可以不止一个输出，可以有多个输出，这样模型可以灵活的应用于分类回归，以及其他的机器学习领域比如降维和聚类等。多个神经元输出的输出层对应的一个实例如下图，输出层现在有4个神经元了。

3） 对激活函数做扩展，感知机的激活函数是sign(z)，虽然简单但是处理能力有限，因此神经网络中一般使用的其他的激活函数，比如我们在逻辑回归里面使用过的Sigmoid函数，即：
$f(z)=1+e-z1$

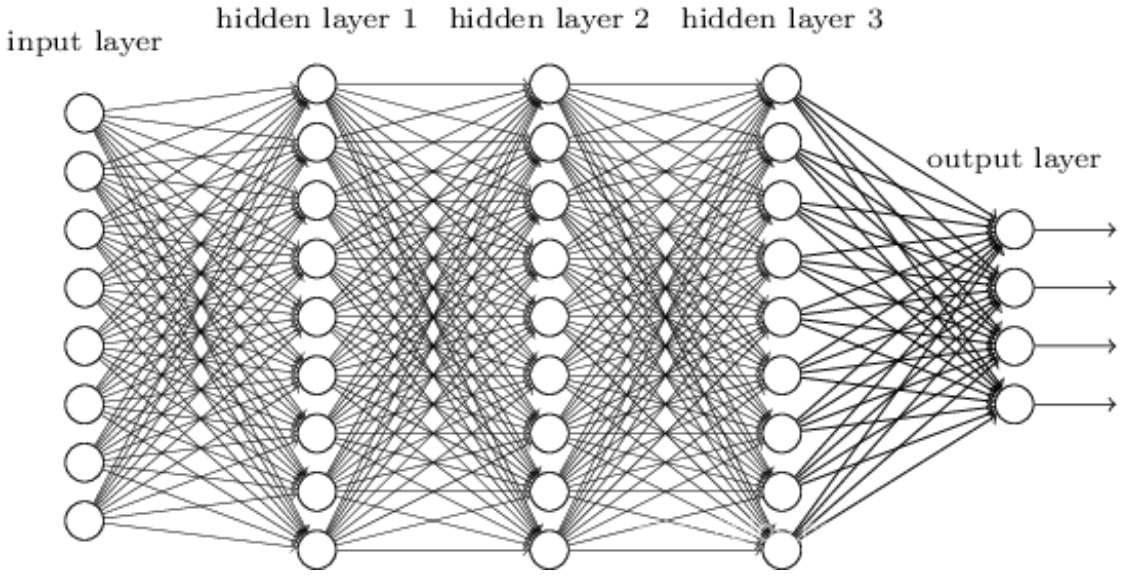
还有后来出现的tanx, softmax,和ReLU等。通过使用不同的激活函数，神经网络的表达能力进一步增强。对于各种常用的激活函数，我随后会整理一份资料。



1.2 DNN的基本结构

上一节我们了解了神经网络基于感知机的扩展，而DNN可以理解为有很多隐藏层的神经网络。这个很多其实也没有什么度量标准, 多层神经网络和深度神经网络DNN其实也是指的一个东西，当然，DNN有时也叫做多层感知机（Multi-Layer perceptron,MLP）, 名字实在是多。后面我们讲到的神经网络都默认为DNN。

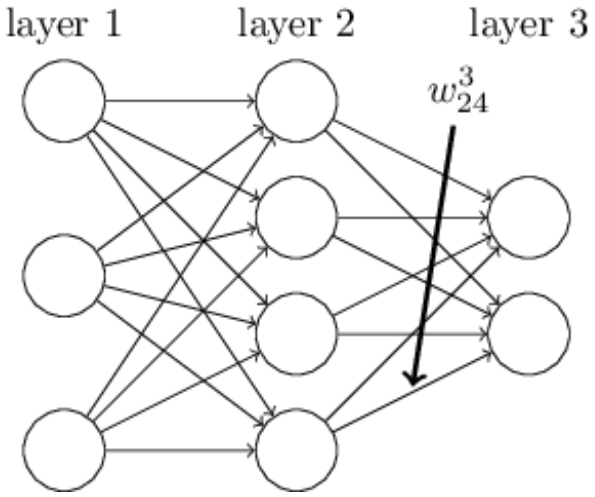
从DNN按不同层的位置划分，DNN内部的神经网络层可以分为三类，**输入层，隐藏层和输出层**，如下图示例，一般来说第一层是输入层，最后一层是输出层，而中间的层数都是隐藏层。



层与层之间是全连接的，也就是说，第i层的任意一个神经元一定与第i+1层的任意一个神经元相连。虽然DNN看起来很复杂，但是从小的局部模型来说，还是和感知机一样，即一个线性关系$z=\sum wi\ xi\ +b$ 加上一个激活函数$\sigma(z)$。

由于DNN层数多，则我们的线性关系系数w和偏倚b的数量也就是很多了。具体的参数在DNN是如何定义的呢？

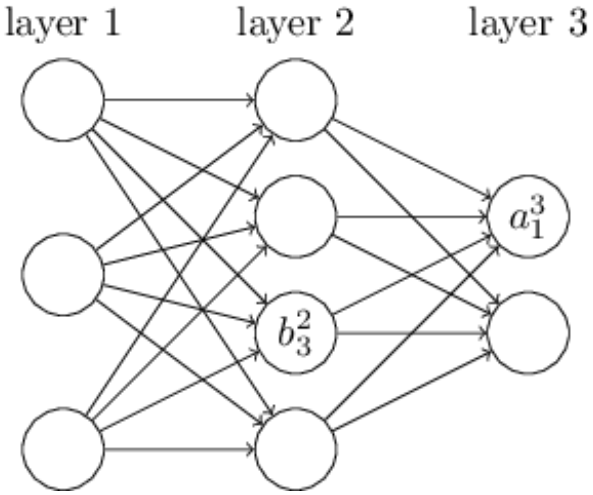
首先我们来看看线性关系系数w的定义。以下图一个三层的DNN为例，第二层的第4个神经元到第三层的第2个神经元的线性系数定义为w243 。上标3代表线性系数w所在的层数，而下标对应的是输出的第三层索引2和输入的第二层索引4。你也许会问，为什么不是w423，而是w243 呢？**这主要是为了便于模型用于矩阵表示运**算，如果是w423 每次进行矩阵运算是wTx+b，需要进行转置。将输出的索引放在前面的话，则线性运算不用转置，**即直接为wx+b。总结下，第l−1层的第k个神经元到第l层的第j个神经元的线性系数定义为wjkl** 。注意，输入层是没有w参数的。



w^l_{jk} is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

再来看看偏倚b的定义。还是以这个三层的DNN为例，第二层的第3个神经元对应的偏倚定义为b32 。其中，上标2代表所在的层数，下标3代表偏倚所在的神经元的索引。输入层是没有偏倚参数b的。

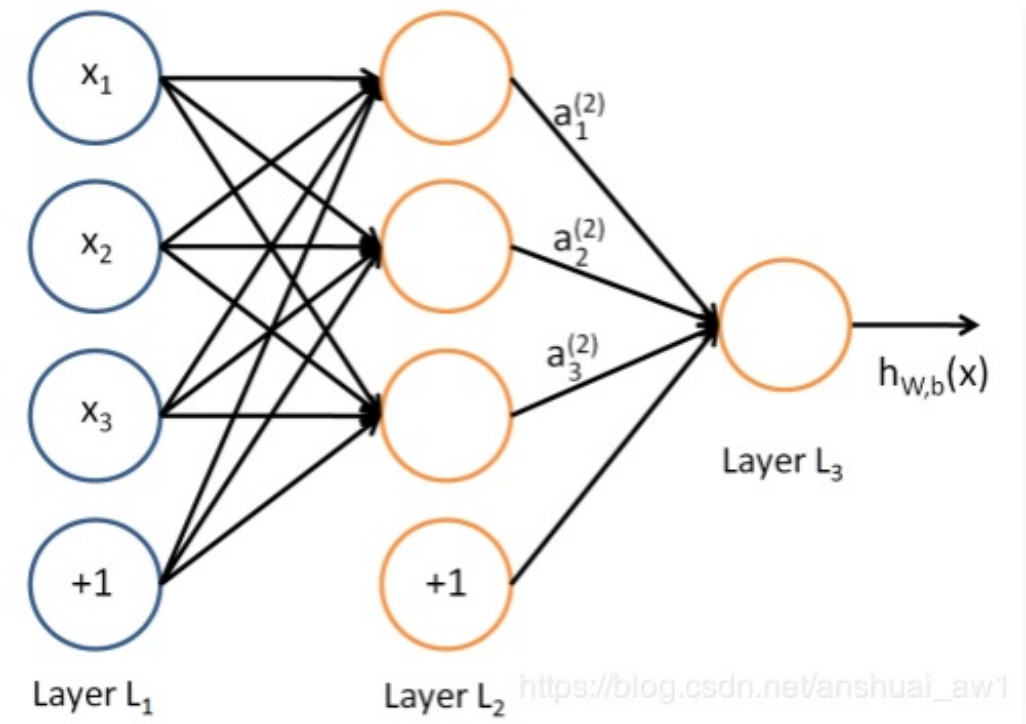
同样的道理，对于神经元的激活值而言，第3层的第1个神经元的激活值应该表示为a13 。



二、DNN前向传播算法

2.1 DNN前向传播算法数学原理

在上一节，我们已经介绍了DNN各层线性关系系数w和偏倚b的定义。假设我们选择的激活函数是σ(z)，隐藏层和输出层的输出值为a，则对于下图的三层DNN，利用和感知机一样的思路，**我们可以利用上一层的输出计算下一层的输出**，也就是所谓的DNN前向传播算法。



对于第二层的的输出a12 ,a22 ,a32 ，我们有：
a12 =σ(z12)=σ(w112 x1 +w122 x2 +w132 x3 +b12)

a22 =σ(z22)=σ(w212 x1 +w222 x2 +w232 x3 +b22)

a32 =σ(z32)=σ(w312 x1 +w322 x2 +w332 x3 +b32)

对于第三层的的输出a13 ，我们有：
a13 =σ(z13)=σ(w113 a12 +w123 a22 +w133 a32 +b13)

将上面的例子一般化，假设第l−1层共有m个神经元，则对于第l层的第j个神经元的输出ajl ，我们有：
ajl =σ(zjl)=σ(k=1∑m wjkl akl−1 +bjl)

其中，如果l=2，则对于的ak1 即为输入层的xk 。

从上面可以看出，使用代数法一个个的表示输出比较复杂，而如果使用矩阵法则比较的简洁。假设第l−1层共有m个神经元，而第l层共有n个神经元，则第l层的线性系数w组成了一个n×m的矩阵Wl, 第l层的偏倚b组成了一个n×1的向量bl , 第l−1层的输出a组成了一个m×1的

向量al−1，第l层的未激活前线性输出z组成了一个n×1的向量zl, 第l层的的输出a组成了一个n×1的向量al。则用矩阵法表示，第l层的输出为：
al=σ(zl)=σ(Wlal−1+bl)

这个表示方法简洁漂亮，后面我们的讨论都会基于上面的这个矩阵法表示来。**所以，应该时刻记住我们符号的含义，否则在后面推导反向传播公式时会比较懵。**

2.2 DNN前向传播算法

有了上一节的数学推导，DNN的前向传播算法也就不难了。所谓的DNN的前向传播算法也就是利用我们的若干个权重系数矩阵W和偏倚向量b来和输入值向量x进行一系列线性运算和激活运算，从输入层开始，一层层的向后计算，一直到运算到输出层，得到输出结果为止。

输入: 总层数L，所有隐藏层和输出层对应的矩阵W，偏倚向量b，输入值向量x
输出：输出层的输出aL

- 1） 初始化a1=x
- 2） for l=2 to L, 计算：

al=σ(zl)=σ(Wlal−1+bl)

最后的结果即为输出aL。

2.3 DNN前向传播算法小结

单独看DNN前向传播算法，似乎没有什么大用处，而且这一大堆的矩阵W,偏倚向量b对应的参数怎么获得呢？怎么得到最优的矩阵W,偏倚向量b呢？这个我们在下一章讲DNN的反向传播算法时再讲。而理解反向传播算法的前提就是理解DNN的模型与前向传播算法。这也是我们先讲前向传播算法的原因。

三、DNN反向传播算法

3.1 DNN反向传播算法要解决的问题

在了解DNN的反向传播算法（Back Propagation，BP）前，我们先要知道DNN反向传播算法要解决的问题，也就是说，**什么时候我们需要这个反向传播算法？**

回到我们监督学习的一般问题，假设我们有m个训练样本：{(x1 ,y1),(x2 ,y2),...,(xm ,ym)}，其中x为输入向量，特征维度为nin ，而y为输出向量，特征维度为nout 。我们需要利用这m个样本训练出一个模型，当有一个新的测试样本(xtest ,?)来到时, 我们可以预测ytest 向量的输出。

如果我们采用DNN的模型，即我们使输入层有nin 个神经元，而输出层有nout 个神经元。再加上一些含有若干神经元的隐藏层。此时我们需要找到合适的所有隐藏层和输出层对应的线性系数矩阵W和偏倚向量b，让所有的训练样本输入计算出的输出尽可能的等于或很接近样本输出。怎么找到合适的参数呢？

如果大家对传统的机器学习的算法优化过程熟悉的话，这里就很容易联想到我们可以用一个合适的损失函数来度量训练样本的输出损失，接着对这个损失函数进行优化求最小化的极值，对应的一系列线性系数矩阵W和偏倚向量b即为我们的最终结果。在DNN中，损失函数优化极值求解的过程最常见的一般是通过梯度下降法来一步步迭代完成的，当然也可以是其他的迭代方法比如牛顿法与拟牛顿法。

对DNN的损失函数用梯度下降法进行迭代优化求极小值的过程即为我们的反向传播算法。

3.2 DNN反向传播算法的基本思路

在进行DNN反向传播算法前，我们需要选择一个损失函数，来度量训练样本计算出的输出和真实的训练样本输出之间的损失。你也许会问：训练样本计算出的输出是怎么得来的？这个输出是随机选择一系列W,b，用前向传播算法计算出来的。即通过一系列的计算：al=σ(zl)=σ(Wlal−1+bl)。计算到输出层第L层对应的aL即为前向传播算法计算出来的输出。

回到损失函数，DNN可选择的损失函数有不少，为了专注算法，这里我们使用**最常见的均方差来度量损失**。当然，针对不同的任务，可以选择不同的损失函数。即对于每个样本，我们期望最小化下式：
J(W,b,x,y)=21 ||aL−y||22

其中，aL和y为特征维度为nout 的向量，而||S||2 为S的L2范数。

损失函数有了，现在我们开始用梯度下降法迭代求解每一层的W,b。

注：以下是BP算法推导的过程，是本文最核心，也是神经网络最基本的公式推导。

思路第一步：
首先是输出层第L层。注意到输出层的W,b满足下式：
aL=σ(zL)=σ(WLaL−1+bL)(1)

这样对于输出层的参数，我们的损失函数变为：
J(W,b,x,y)=21 ||aL−y||22 =21 ||σ(WLaL−1+bL)−y||22 (2)

这样求解W,b的梯度就简单了：
∂WL∂J(W,b,x,y) =∂zL∂J(W,b,x,y) ∂WL∂zL =∂aL∂J(W,b,x,y) ∂zL∂aL ∂WL∂zL =(aL−y)⊙σ′(zL)(aL−1)T(3)

∂bL∂J(W,b,x,y) =∂zL∂J(W,b,x,y) ∂bL∂zL =∂aL∂J(W,b,x,y) ∂zL∂aL ∂bL∂zL =(aL−y)⊙σ′(zL)(4)

注意上式中有一个符号⊙,它代表Hadamard积，对于两个维度相同的向量A（a1 ,a2 ,...an ）T和B（b1 ,b2 ,...bn ）T，则A⊙B=(a1 b1 ,a2 b2 ,...an bn)T。

对于公式（3）和（4），我在这里多解释一下为什么是这样：

对于公式（3）：前两项之所以是Hadamard积的形式，是因为∂aL∂J(W,b,x,y) ∂zL∂aL 都是针对**同一层**的神经元。如果我们考虑对于L层的第j个神经元，即∂ajL ∂J(W,b,x,y) σ′(zjL)，那么整合这一层的神经元，自然是(aL−y)⊙σ′(zL)这样Hadamard积的形式。那么(aL−1)T为什么在（3）中的最后呢？这涉及到矩阵求导的知识，不是我们本文的重点。在这里，用到的知识是：如果
Y=WX+B

那么
∂W∂C =∂Y∂C XT

这样，即可推出公式（3）。公式（4）与公式（3）类似。

思路第二步：
我们注意到在求解输出层的W,b的时候，有公共的部分∂zL∂J(W,b,x,y) ，因此我们可以把公共的部分即对zL先算出来，记为：
δL=∂zL∂J(W,b,x,y) =(aL−y)⊙σ′(zL)(5)

根据公式（3）（4），我们可以把输出层的梯度算出来，那么如何计算上一层L−1层的梯度，上上层L−2层的梯度呢？这里我们需要一步步的递推，注意到对于第l层的未激活输出zl，它的梯度可以表示为:
δl=∂zl∂J(W,b,x,y) =∂zL∂J(W,b,x,y) ∂zL−1∂zL ∂zL−2∂zL−1 ...∂zl∂zl+1 (6)

如果我们可以依次计算出第l层的δl,则该层的Wl,bl很容易计算？为什么呢？注意到根据前向传播算法，我们有：
zl=Wlal−1+bl(7)

所以根据上式我们可以很方便的计算出第l层的Wl ,bl 的梯度如下：
∂Wl∂J(W,b,x,y) =∂zl∂J(W,b,x,y) ∂Wl∂zl =δl(al−1)T(8)

∂bl∂J(W,b,x,y) =∂zl∂J(W,b,x,y) ∂bl∂zl =δl(9)

思路第三步：
那么现在问题的关键就是要求出δl了。这里我们用数学归纳法，第L层的δL上面我们已经求出， 假设第l+1层的δl+1已经求出来了 ，那么我们如何求出第l层的δl呢？我们注意到：
δl=∂zl∂J(W,b,x,y) =∂zl+1∂J(W,b,x,y) ∂zl∂zl+1 =δl+1∂zl∂zl+1 (10)

可见，用归纳法递推δl+1和δl的关键在于求解∂zl∂zl+1 。

而zl+1和zl的关系其实很容易找出：
zl+1=Wl+1al+bl+1=Wl+1σ(zl)+bl+1(11)

这样很容易求出：
∂zl∂zl+1 =(Wl+1)T⊙nl+1 ⏟ ⏟ ⏟ (σ′(zl),..,σ′(zl)) (12)

公式（12）的意思就是(Wl+1)T的每一列都Hadamard积σ′(zl)。

将公式（12）代入公式（10）我们得到：
δl=δl+1∂zl∂zl+1 =((Wl+1)Tδl+1)⊙σ′(zl)(13)

公式（13）的推导过程为：
δl+1∂zl∂zl+1 =∂zl∂(δl+1)Tzl+1 =∂zl∂(δl+1)T(Wl+1σ(zl)+bl+1) =∂zl∂(δl+1)TWl+1σ(zl) =((δl+1)TWl+1)T⊙σ′(zl)==((Wl+1)Tδl+1)⊙σ′(zl)

现在我们得到了δl的递推关系式，只要求出了某一层的δl，求解Wl,bl的对应梯度就很简单。

总结：

其实，对于更新每一层的Wl,bl的对应梯度，我们仔细观察整个过程，发现只需要四个公式就可以完整地进行更新。这就是**著名的反向传播的四个公式**，即公式(5)、(13)、(8)、(9)。我们稍加改动，使其可以应用到多种损失函数，即：

δL=∂aL∂J ⊙σ′(zL)(BP1)δl=(Wl+1)Tδl+1⊙σ′(zl)(BP2)∂Wl∂J =δl(al−1)T(BP3)∂bl∂J =δl(BP4)

3. 3 DNN反向传播算法过程

现在我们总结下DNN反向传播算法的过程。由于梯度下降法有批量（Batch），小批量(mini-Batch)，随机三个变种，为了简化描述，这里我们以**最基本的批量梯度下降法为例**来描述反向传播算法。实际上在业界使用最多的是mini-Batch的梯度下降法。不过区别仅仅在于迭代时训练样本的选择而已。

输入: 总层数L，以及各隐藏层与输出层的神经元个数，激活函数，损失函数，迭代步长α，最大迭代次数MAX与停止迭代阈值ϵ，输入的m个训练样本{(x1 ,y1),(x2 ,y2),...,(xm ,ym)}

输出：各隐藏层与输出层的线性关系系数矩阵W和偏倚向量b

1）初始化各隐藏层与输出层的线性关系系数矩阵W和偏倚向量b的值为一个**随机值**。

2）for iter to 1 to MAX：

2-1) for i=1 to m：

a) 将DNN输入a1设置为xi

b) for l=2 to L，进行前向传播算法计算ai,l=σ(zi,l)=σ(Wlai,l−1+bl)

c) 通过损失函数计算输出层的δi,L（**BP1**）

d) for l=L−1 to 2, 进行反向传播算法计算δi,l=(Wl+1)Tδi,l+1⊙σ′(zi,l)（**BP2**）

2-2) for l=2 to L，更新第l层的Wl,bl:

$$W^l = W^l - \alpha \sum_{i=1}^{m} \delta^{i,l} (a^{i,l-1})^T \quad (BP3)$$

$$b^l = b^l - \alpha \sum_{i=1}^{m} \delta^{i,l} \quad (BP4)$$

2-3) 如果所有W，b的变化值都小于停止迭代阈值ϵ，则跳出迭代循环到步骤3。

3） 输出各隐藏层与输出层的线性关系系数矩阵W和偏倚向量b。

注：上述的算法其实看起来有些复杂了，而实际上根据机器学习里普通的批梯度下降算法，我们在求出m个样本的δi,L后，求平均得到δL，然后反向继续更新δl。

3.4 DNN反向传播算法小结

有了DNN反向传播算法，我们就可以很方便的用DNN的模型去解决各种监督学习的分类回归问题。当然DNN的参数众多，矩阵运算量也很大，直接使用会有各种各样的问题。有哪些问题以及如何尝试解决这些问题并优化DNN模型与算法，我们会在其它博客里讲。