神经网络BP反向传播算法推导

本文详细给出了用于神经网络训练的反向传播（Back Propagation）算法的证明推导过程。过程虽然有点繁琐，却并未涉及到复杂的数学变换，所需要的仅是基本的导数和矩阵运算知识，以及耐心。

本文是我自己整理的BP算法推导过程。它原来的推导过程，在矩阵运算方面，毛病不少。特此，我自己找了一些数学工具，独立完成BP算法的推导。 2018年1月27日 22:04:44--YDH

# 一、DNN神经网络定义 && 符号定义

为了使数学推导过程更加清晰简洁，我们首先定义一些符号表达：

L ———神经网络总层数

———第层的神经元个数

———连接第层的第j个神经元和第l-1层的第k个神经元的权重系数

———第l层的第j个神经元的偏置项（bias，偏置值）

———第l层神经元的权重系数**矩阵**（由构成的矩阵，这个矩阵第j**行**代表第l层第j个神经元，第k**列**代表第l-1层第k个神经元，下标jk代表神经元k指向j的权重，这也是为什么k🡪j，矩阵中的下标是jk，而不是kj。），大小为()

———第l层神经元的偏置向量（由构成的向量），大小()

———第l层的第j个神经元的加权输入

———第l层神经元的加权输入向量()，其每个元素为

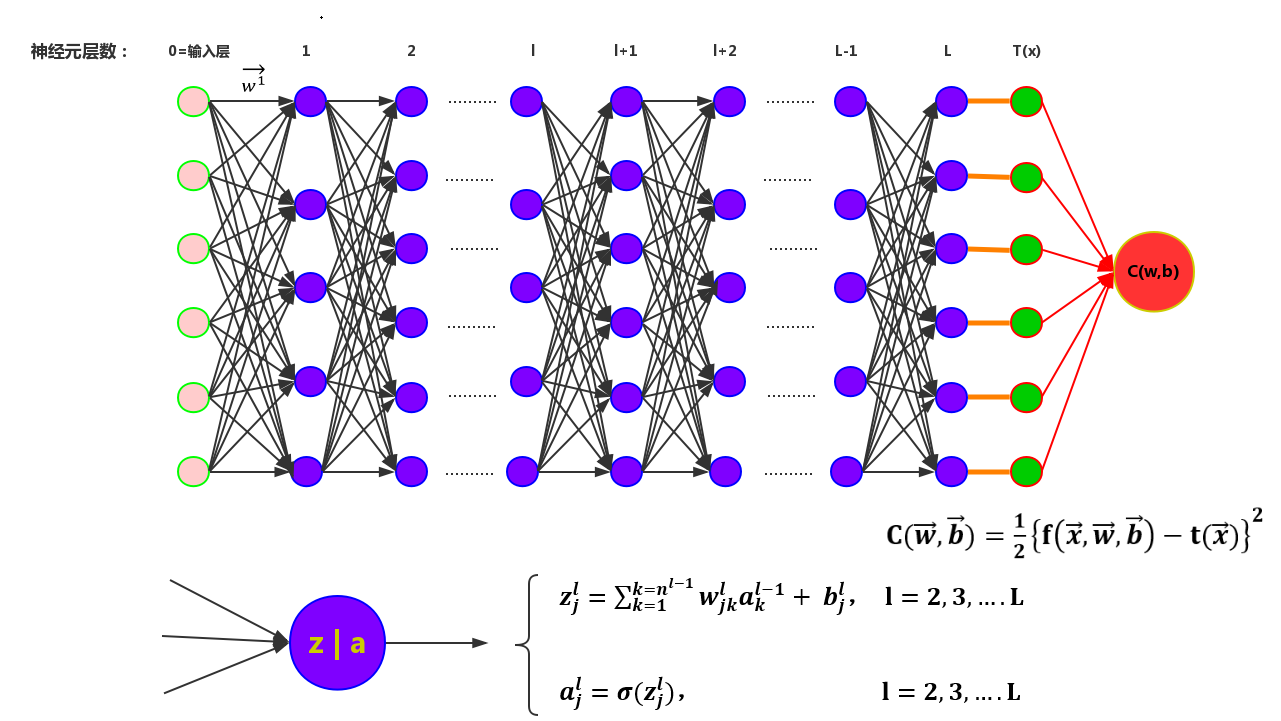
———第l层的第j个神经元的激活函数输出

———第l层神经元的激活函数输出向量()，其每个元素为

——— 输入x后期望得到的目标结果，它就是导师数据。

【约定：下面，凡是加粗后的如等，都代表向量或矩阵**，**否则都是标量**。】**

神经网络模型：



根据神经网络的定义我们有：

， （式 1.1）

， （式 1.2）

其中，为神经网络采用的激活函数。写成**向量/矩阵**形式有：

**，** （这里有问题吗？**，都是(**,1**)向量，是()矩阵， 是()向量，结果还真是没问题的！！！！**） （另外，这不就是吗？就是关于x,y向量的**非齐次线性空间**变换。）

**，**  （式 1.3）

对于第**1**(数字0)层，则直接有。（x就是输入）

最后令C表示整个神经网络的代价函数，其应是关于最后一层的输出的函数，即：

**cost**  （式 1.4）

**C**的定义规则，下一章介绍。

# 二、有监督学习 & 梯度下降算法

## 2.1 有监督学习

机器学习的常用方法，主要分为有监督学习(supervised learning)和无监督学习(unsupervised learning)。

无监督学习，就是“瞎JB乱学”，一切听天由命。我不知道怎么输出才对，我也不知道每一个神经节点应该怎样调整，只是用一个个随机参数不断尝试。

有监督学习，是通过已有的训练样本（即已知输入以及其对应的输出）去训练得到一个最优神经网络模型。也就是，我知道如果我输入“苹果”，你应该的输出的是“苹果汁”，如果网络输出正确，那还好；如果错误，那我应该怎么调整这个网络呢？

下面介绍梯度下降算法来解决这个问题。

## 2.2 梯度下降算法：

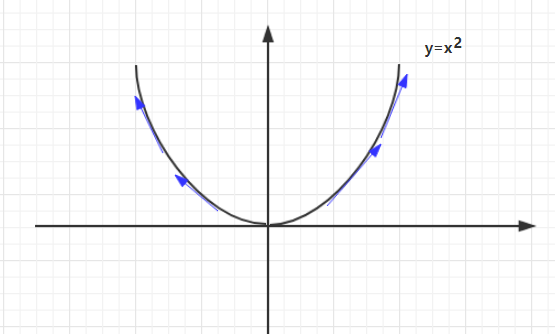
在DNN中，我们知道，网络一旦确定，那么网络的结构，数量，我们是无法调整的；唯一能调整的就是神经节之间的权重**w**和神经节的偏置**b**。实际上，训练DNN，就是训练**w**和**b**。

我们网络的最终输出可以表示为函数式：; (其中，是输入向量，是权重矩阵，是神经元偏置)。

而在有监督学习中，“导师”会要求每一个输入都有一个目标(target)输出：

于是问题就变成了如何让 与 变得一致的问题，换句话说就是要求 ，是不是就是我们想要的结果？

但是这个问题，仔细想想还是有点麻烦啊。不知道哪一个聪明人，他想到了我们的梯度，在梯度的几何意义上，我们有这样一个性质：**当变量顺着梯度的方向改变，函数增长最快，当变量逆着梯度的方向改变，函数减小最快**(*《高数：下》7.2 梯度*)。那么我们不就可以把问题转变成求一个关于的极小值的问题了吗？你是否还记得我们学过一个经典一元二次函数，还记得它的图形吗？



(图2.2.1 *图中蓝色箭头，就是切点对应的梯度方向*)

看上图，是不是相当满足上文提到的定理？我们来验证一下：

，

函数极小值的更新规则：

（式2.2.1）

上式所体现的就是：将向着与梯度相反的方向改变，以使函数接近她的极小值点。叫做学习率，一般取值(0,0.1)，它是为了防止跳得太快，y()错过极值点。你将上式2.2.1，代入图2.2.1中验证，是不是就想通了？

所以模仿，我们定义了误差函数，如下式：

**（式2.2.2）**

当然，你这样定义也是没问题的：

**（式2.2.3）**

上式中，定个系数，原因很好猜，一切为了简单，往下看你就懂了。C=cost,取花费，成本之意，成本当然是越小越好，所以要找极小值，也有的地方也用E=error来表示，本质上都是一个用意。

根据（式2.2.2），我们的问题就变成了：

随机初始化后，再将沿着梯度的反方向调整，得到新的，不断重复这个过程，以使能够不断逼近它的极小值点。虽然最终理想是C=0；不过，即使做不到，我们只要取到极小极点处的，代入后，也将是我们所能训练出来的最好网络了，不是吗？

而根据C的梯度公式：

**（式2.2.4）**

上式中,是方向的方向向量，而当为单位方向向量时，函数的变化最快（见《高数》第七节，二、梯度，p107），而我们要让它往极小值方向变化，所以**的调整方案如下：**

**的调整方案：**

**（式2.2.5）**

**的调整方案：**

**（式2.2.6）**

上两式中，是学习率，一般取值，原因是防止在极值点 两边不断跳动而不收敛，所以把每一次的改变量，缩小一点。

当然，最终目标的判定，理论上当然应该看C值的大小。理想状态下，当C=0时，目标就找到了，的调整就应当全面结束。而实际上C=0经常达不到，所以一般有两种停止训练方案：

1. C小于/等于某一值时，训练结束；
2. 当训练达到一个最大训练步数时，训练结束；

和都是经验值，一般取，。

你看，我们的目标本来是调整，通过梯度下降算法(式2.2.5) ，(式2.2.6)实际目标就变成了求，，这两个值的具体推导，这们将在第三章讲解。这就是BP算法中用到的梯度下降算法的内容。

（**\*补**：**关于（式2.2.2）和（式2.2.3）其实本质上是没有差别的**。

我们以最外层BP网络的w,b的推导为例（注意：最外层BP网络的a(x,w,b)=f(x,w,b)）。

以（式2.2）为基础：

而以（式2.3）为基础时：

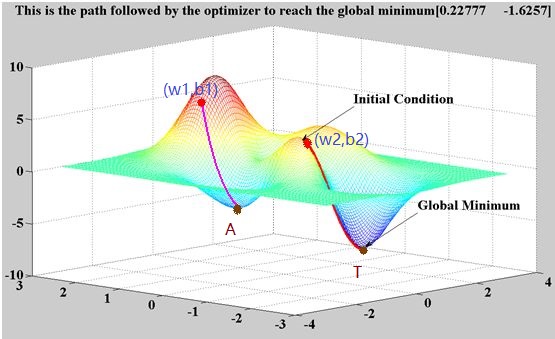
你看，这两个式子，结果是不是一样滴？ 注意红色部分，其实就是复合求导的细节上的不同。上式中，是指上一层与对应w相关联的激励输出,当然这个顺序肯定就不能换了，否则你找到的就是极大值了，训练结果南辕北辙。

）

## 2.3 \*梯度下降算法的问题

**局部极小值**：

由于有界集上的连续可微函数是一定能通过梯度下降法找到极值点的（比如图2.1的函数）。对于凸函数，从任意一点出发，沿着梯度下降的方向走，最终一定下降到全局最小值附近，而非凸函数，从任意一点出发，沿着梯度下降方向最大可能则是走向那个离出发点最近的局部最小值。但是多层神经网络恰恰不是凸函数，所以理论上存在多个局部最小值，而且随着层数的增加，局部最小值就越多，这就是CNN的致命缺陷之一。



以上图为例，如果起点在(w1,b1)则极有可能最终到达A点，而不是目标T点。

一般在两个方向上解决这个问题：

首先，就是优化起始点：

1、均匀分布初始权重（Uniform distribution）   
2、高斯分布初始权重（Gaussian distribution）   
3、PositiveUnitball   
4、Xavier   
5、MSRA   
6、Bilinear

然后，就是在梯度下降中自适应的调节学习率，使得梯度有可能跳出某个局部最小值。比如加入动量的梯度下降，在梯度下降后期陷入局部最小值的时候，有可能借助之前相同的动量越出局部最小值。

**梯度消失**：

...

## 2.4 交叉熵

上面所说的梯度下降算法中定义的误差函数C，也叫损失函数，它叫做**均方差函数**。在解决回归问题（如房价啊，销量啊，输出是一个或一组连续实数）时，常用**均方差函数**。

但是，在求分类问题时，就不能用**均方差函数**了，因为输出是离散值，无法对训练结果进行更细致的评估，所以要用交叉熵函数：

具体分析，请看CNN算法原理。

总结一个损失函数的使用，如下：

关于均方差函数与交叉熵函数的优劣比较，请看专文《交叉熵代价函数：“我为什么比均方差函数优秀”》。

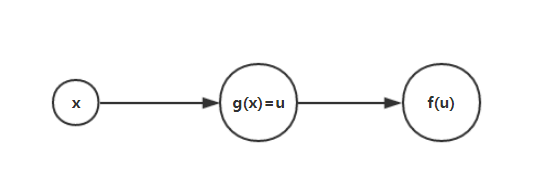
# 三、反向传播算法推导

## 3.1 推导总体思路：链式法则

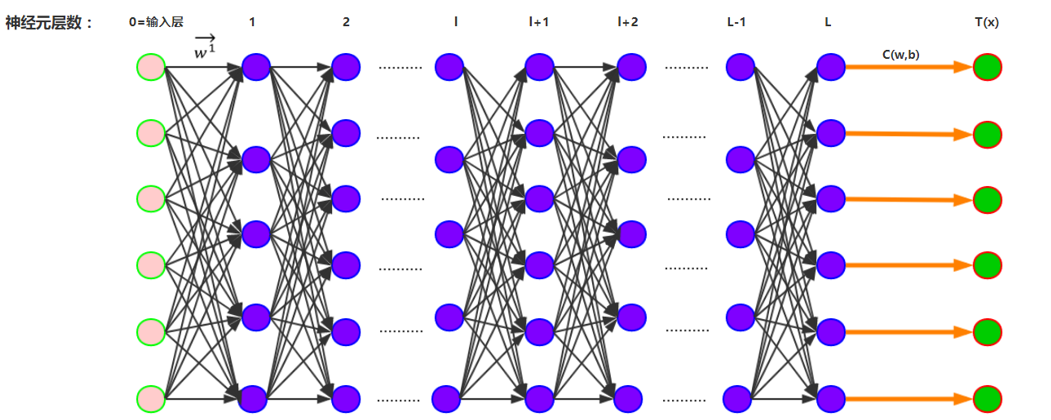
**链式法则**：

在《高数：上，2.3 复合函数的求导法则》中有这样一个性质在点x处可导，则：

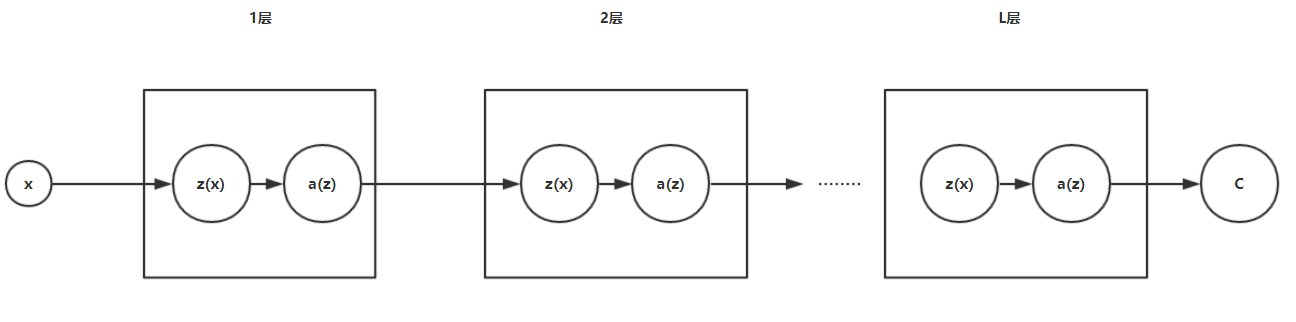
上式，你可以这样看：



你看上图，像不像一个只有三层，每层只有一个神经元的DNN？ 再看看DNN的网络图：



实际上的链式法则是不是：



上图所有的x,z,a都是向量式，将x改为w,b，它们的传递方式不会有任何改变；上图的链式法则依然成立。

接下来我们就可以依此进行推导了。

## 3.2 的推导

### 3.2.1 最后一层的推导

**（式 3.2.1.1）**

而根据梯度下降算法，我们知道：

但是有个问题维度是()，而维度是()，无法一一对应，所以需要降维，根据实附录2.5的算法，可得：

**为什么一定要这样降维？说实话，我也不清楚原理，我是根据实际网络参数的计算结果，倒推出来的，现在没什么时间，暂时就不去深究了。**

### 3.2.2 倒数第二层的推导

上式紫色部分与第L层计算结果重合**。**

与3.2.1同理：

### 3.2.3 倒数第三层的推导

上式紫色部分与第L-2层计算结果重合**。**

同理：

### 3.2.4 提公因式

从L-1层，L-2层的最终公式中，我们已经能看出L层以后的各层规律了。

令：

则将代入得：

再往下代入，得：

所以：

（式3.2.4）

由上式可知**：**的物理意义，其实就是输出误差，对l层的输入的偏导。

### 3.2.4第l层的推导

(l=1,2,3,...,L, )

## 3.3 的推导

### 3.3.1第l层的推导

根据链式法则，可以直接得到:

同理：

## 3.4 通用w,b偏导求法总结

本章最重要的四个公式总结如下：

# 附录、算法中用到的数学工具

## 1．矩阵偏导规则(雅可比法则)

其实是我自己总结出来的矩阵偏导规则，但是后来发现这个东西的学名叫雅可比法则。

规则如下：

令n维向量通过某种关系能够得到一个m维向量，则y关于x的偏导为：

总结：结果矩阵的行数 = 的维度，矩阵的列数 = 的维度。

即使将，由向量变成矩阵，上面的结论依然成立。

## 2．对角矩阵

对角矩阵(diagonal matrix)是一个**主对角线**之外的元素**皆为0**的矩阵，常写为。

(式0.2.1)

**运算规律：**对角矩阵的运算包括和、差运算、数乘运算、同阶对角阵的乘积运算，且结果仍为对角阵。

扩展： 其实式0.2.1中，如果是相同维度的**向量**（每一个元素为），上面的运算规律**依然适用**。

## 2.5 对角矩阵降维

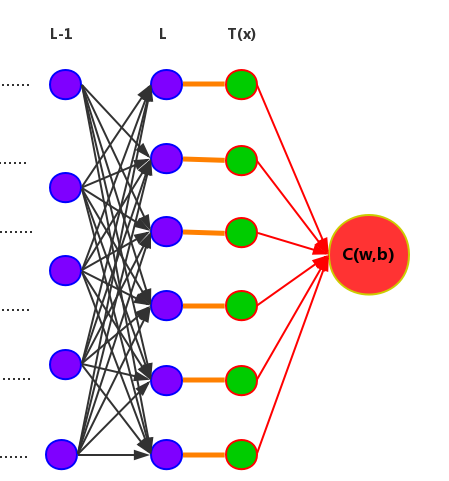
这个算法，我不知道标准说法是什么，原理又是什么，但是根据实际在梯度下降算法中的推导，可知对角矩阵降维法为：

只保留主对角线上的元素（也可以是向量），组成新的维度。

## 3. 算法

通过式2.2.2，我们知道

由上式直接可知，**f**，**t**维度相同，而对于网络的最后一层(L层)，其实，而且，**f**，，**t**三者维度相同。



这个很好理解，你从上图也能看出来，如果与**f**维度不同或不相等，则证明L与T之间**至少**还间隔了一个L+1层，那么L就不应该是最后一层了，L+1层才是。所以最后一层与**f**，**t**维度肯定相同。

所以：

（式0.3.1）

上式的中间过程，用了复合求导。

所以：

**=**  （式0.3.2）

上式中，表示对角矩阵，即只有主对角线上有值，其余值都为0的矩阵，我们的结果就是这样一个()矩阵。

## 4. 算法

**=**  （式0.4.1）

## 5. 算法

**=**

**=**  （式0.5.1）

上式中是**我自己创造的表达式**，代表个的排列。

## 6. 算法