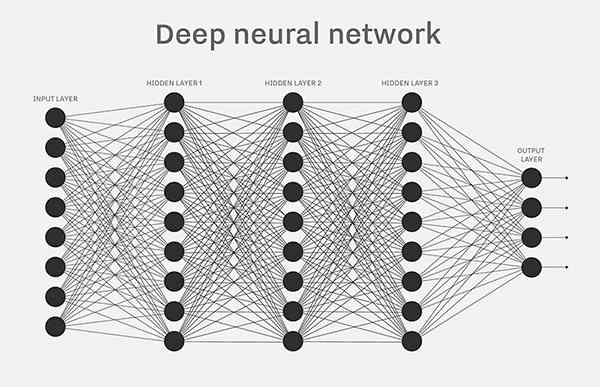
# 深度学习入门



## 1. 大脑与神经网络

### 1.1 大脑的神经结构

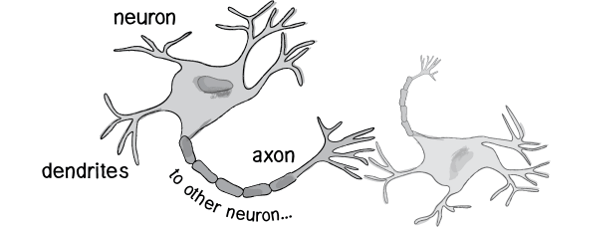
历史上，科学家一直希望模拟人的大脑，造出可以思考的机器。人为什么能够思考？科学家发现，原因在于人体的神经网络。

(1)外部刺激通过神经末梢，转化为电信号，转导到神经细胞（又叫神经元）。

(2)无数神经元构成神经中枢。

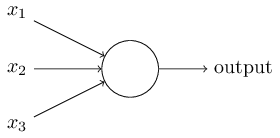
(3)神经中枢综合各种信号，做出判断。

(4)人体根据神经中枢的指令，对外部刺激做出反应。



### 1.2 感知器

既然思考的基础是神经元，如果能够"人造神经元"（artificial neuron），就能组成人工神经网络，模拟思考。上个世纪六十年代，提出了最早的"人造神经元"模型，叫做["感知器"](https://zh.wikipedia.org/wiki/%E6%84%9F%E7%9F%A5%E5%99%A8)（perceptron），直到今天还在用。



上图的圆圈就代表一个感知器。它接受多个输入（x1，x2，x3...），产生一个输出（output），好比神经末梢感受各种外部环境的变化，最后产生电信号。

下面来看一个例子。城里正在举办一年一度的游戏动漫展览，小明拿不定主意，周末要不要去参观。

他决定考虑三个因素。

天气：周末是否晴天？

同伴：能否找到人一起去？

价格：门票是否可承受？

这就构成一个感知器。上面三个因素就是外部输入，最后的决定就是感知器的输出。如果三个因素都是 Yes（使用1表示），输出就是1（去参观）；如果都是 No（使用0表示），输出就是0（不去参观）。

### 1.3 权重和阀值

现实中，各种因素很少具有同等重要性：某些因素是决定性因素，另一些因素是次要因素。因此，可以给这些因素指定权重（weight），代表它们不同的重要性。

例如上个例子中，可以假设为：

天气：权重为8

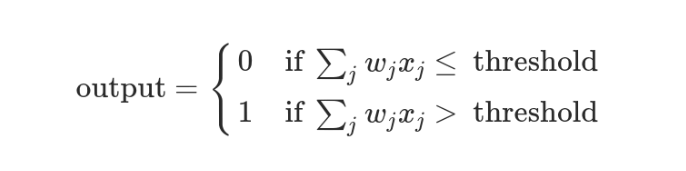
同伴：权重为4

价格：权重为4

天气是决定性因素，同伴和价格都是次要因素。如果三个因素都为1，它们乘以权重的总和就是 8 + 4 + 4 = 16。如果天气和价格因素为1，同伴因素为0，总和就变为 8 + 0 + 4 = 12。

这时，还需要指定一个阈值（threshold）。如果总和大于阈值，感知器输出1，否则输出0。假定阈值为8，那么 12 > 8，小明决定去参观。阈值的高低代表了意愿的强烈，阈值越低就表示越想去，越高就越不想去。

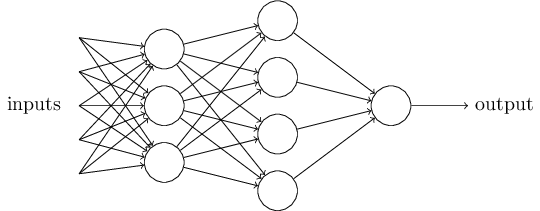
上面的决策过程，使用数学表达如下。



上面公式中，x表示各种外部因素，w表示对应的权重。

### 1.4 决策模型

单个的感知器构成了一个简单的决策模型，已经可以拿来用了。真实世界中，实际的决策模型则要复杂得多，是由多个感知器组成的多层网络。



上图中，底层感知器接收外部输入，做出判断以后，再发出信号，作为上层感知器的输入，直至得到最后的结果。

### 1.5 矢量化

为了方便后面的讨论，需要对上面的模型进行一些数学处理。

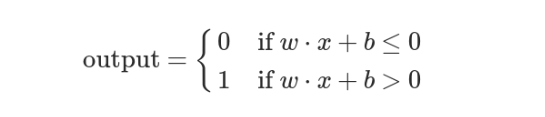
外部因素 x1、x2、x3 写成矢量 **<**x1**,** x2**,** x3**>**，简写为 x

权重 w1、w2、w3 也写成矢量 **(**w1**,** w2**,** w3**)**，简写为 w

定义运算 w⋅x **=** ∑ wx，即 w 和 x 的点运算，等于因素与权重的乘积之和

定义 b 等于负的阈值 b **=** **-**threshold

感知器模型就变成了下面这样。



### 1.6 神经网络的运作过程

一个神经网络的搭建，需要满足三个条件。

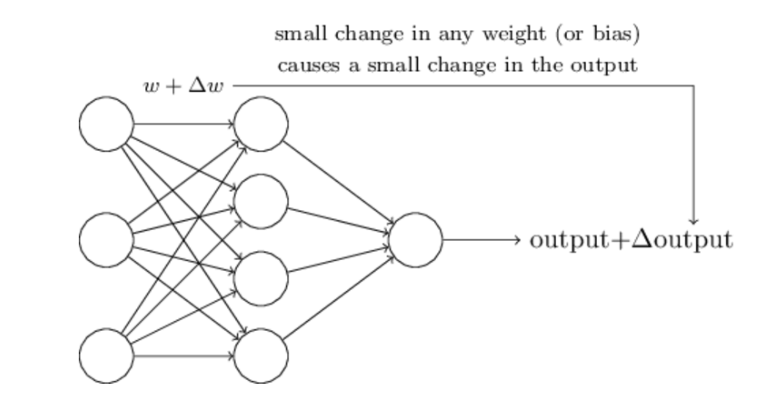
输入和输出

权重（w）和阈值（b）

多层感知器的结构

其中，最困难的部分就是确定权重（w）和阈值（b）。目前为止，这两个值都是主观给出的，但现实中很难估计它们的值，必需有一种方法，可以找出答案。

这种方法就是试错法。其他参数都不变，w（或b）的微小变动，记作Δw（或Δb），然后观察输出有什么变化。不断重复这个过程，直至得到对应最精确输出的那组w和b，就是我们要的值。这个过程称为模型的训练。



因此，神经网络的运作过程如下：

1.确定输入和输出

2.找到一种或多种算法，可以从输入得到输出

3.找到一组已知答案的数据集，用来训练模型，估算w和b

4.一旦新的数据产生，输入模型，就可以得到结果，同时对w和b进行校正

### 1.7 输出的连续性(激活函数)

上面的模型有一个问题没有解决，按照假设，输出只有两种结果：0和1。但是，模型要求w或b的微小变化，会引发输出的变化。如果只输出0和1，未免也太不敏感了，无法保证训练的正确性，因此必须将"输出"改造成一个连续性函数。

这就需要进行一点简单的数学改造。

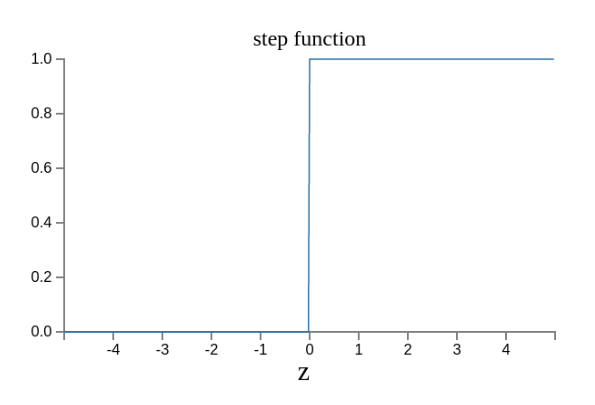
首先，将感知器的计算结果wx + b记为z。

然后，计算下面的式子，将结果记为σ(z)。

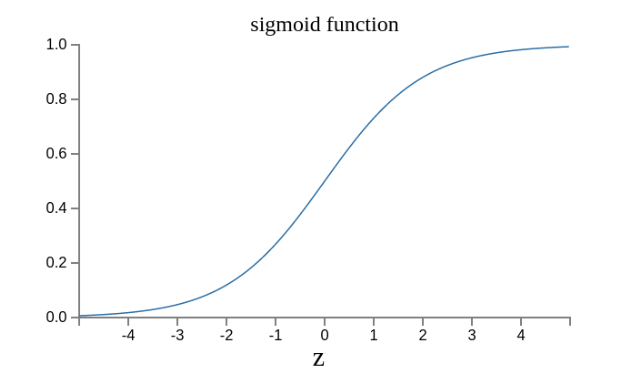
σ**(**z**)** **=** 1 **/** **(**1 **+** e**^(-**z**))**

这是因为如果z趋向正无穷z → +∞（表示感知器强烈匹配），那么σ(z) → 1；如果z趋向负无穷z → -∞（表示感知器强烈不匹配），那么σ(z) → 0。也就是说，只要使用σ(z)当作输出结果，那么输出就会变成一个连续性函数。

原来的输出曲线是下面这样。

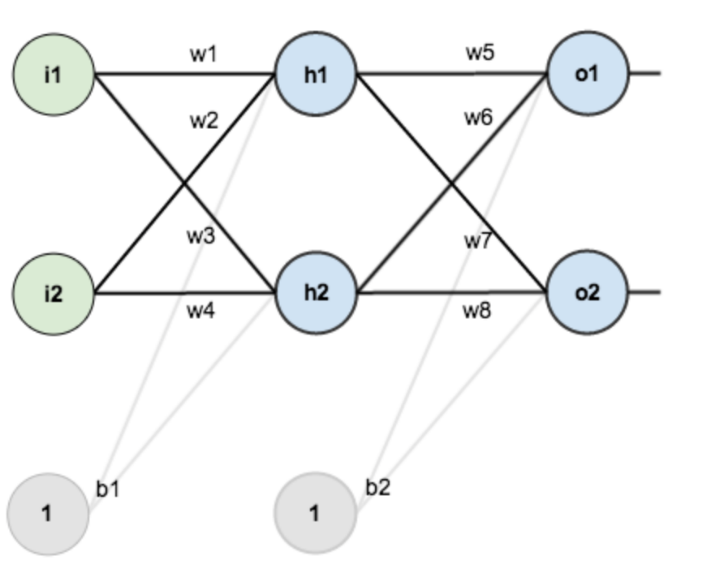


现在变成了这样。



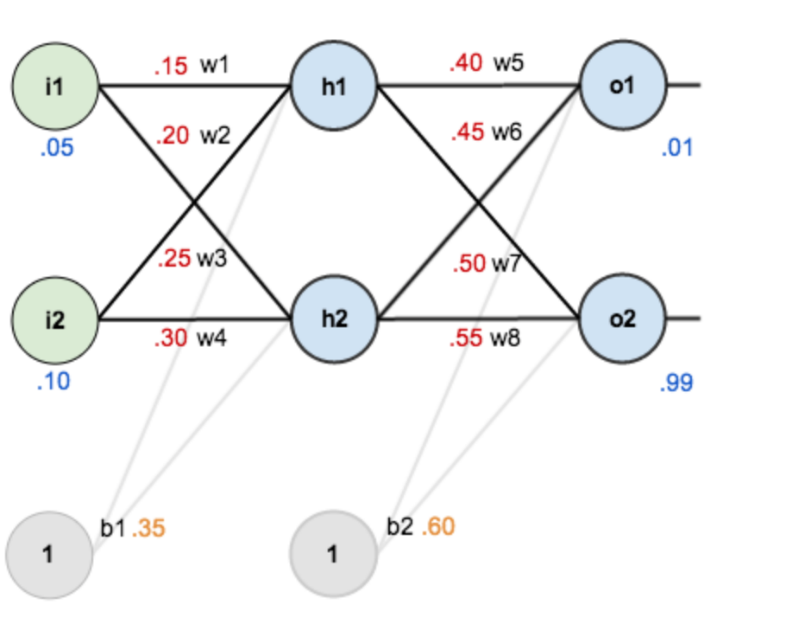
## 2.前向传播和反向传播

假设有一个网络层如下：



第一层是输入层，包含两个神经元i1，i2，和截距项b1；第二层是隐含层，包含两个神经元h1,h2和截距项b2，第三层是输出o1,o2，每条线上标的wi是层与层之间连接的权重，激活函数我们默认为sigmoid函数。

现在对他们赋上初值，如下图：

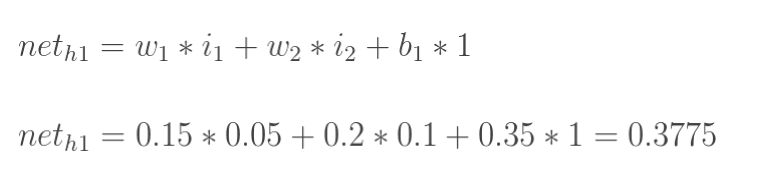


目标：给出输入数据i1,i2(0.05和0.10)，使输出尽可能与原始输出o1,o2(0.01和0.99)接近。

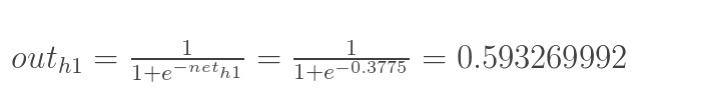
### 2.1 前向传播

**(1)输入层---->隐含层**

计算神经元h1的输入加权和：



神经元h1的输出o1:(此处用到激活函数为sigmoid函数)：

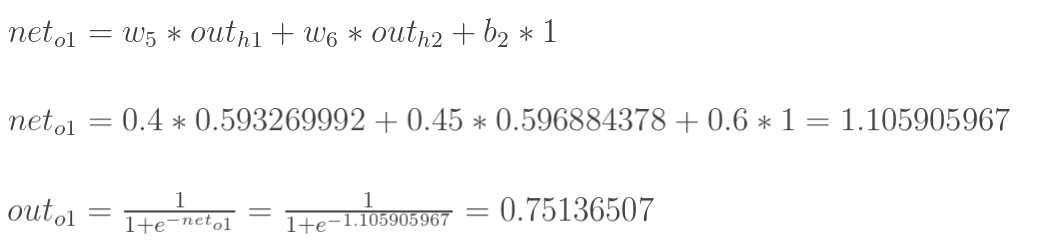


同理，可计算出神经元h2的输出o2：

http://images2015.cnblogs.com/blog/853467/201606/853467-20160630150244265-1128303244.png

**(2) 隐含层---->输出层**

计算输出层神经元o1和o2的值：



这样前向传播的过程就结束了，我们得到输出值为[0.75136079 , 0.772928465]，与实际值[0.01 , 0.99]相差还很远，现在我们对误差进行反向传播，更新权值，重新计算输出。

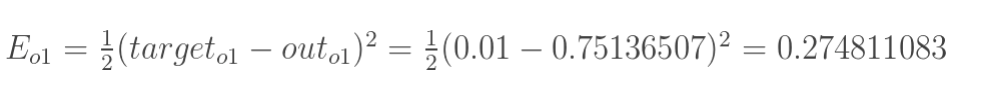
### 2.2 反向传播

**(1)计算总误差**

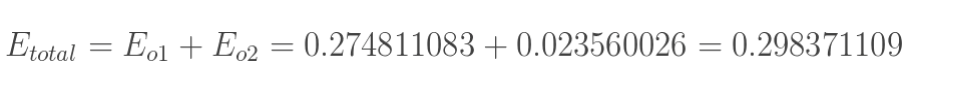
总误差：(square error)



但是有两个输出，所以分别计算o1和o2的误差，总误差为两者之和：

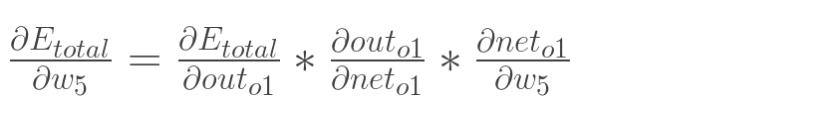




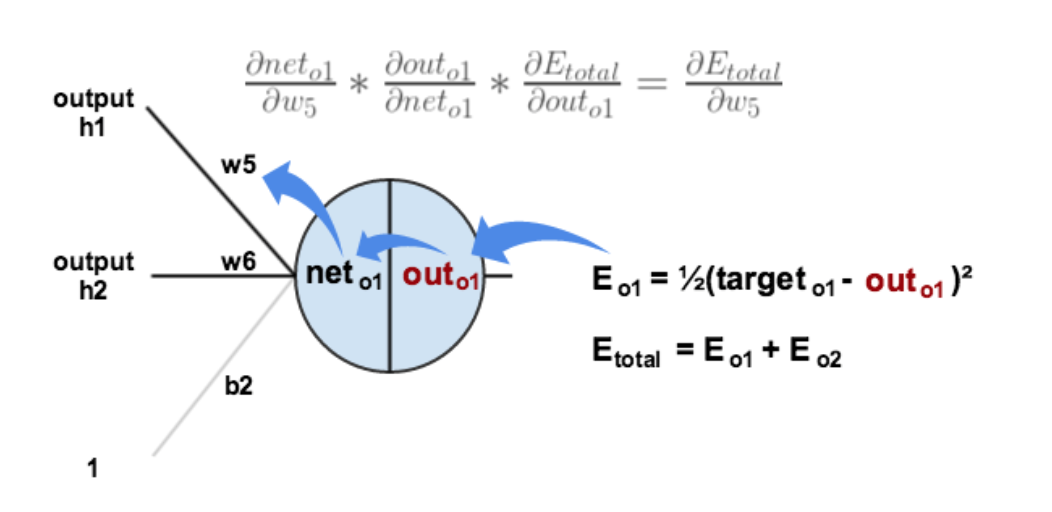


**(2)隐含层--->输出层的权值更新**

以权重参数w5为例，如果我们想知道w5对整体误差产生了多少影响，可以用整体误差对w5求偏导求出：（链式法则）

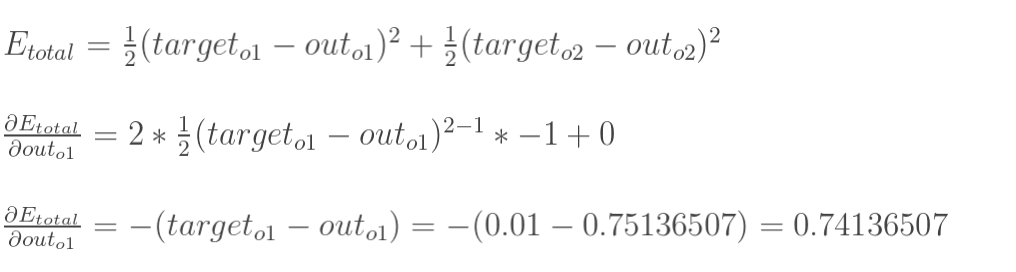


下面的图可以更直观的看清楚误差是怎样反向传播的：

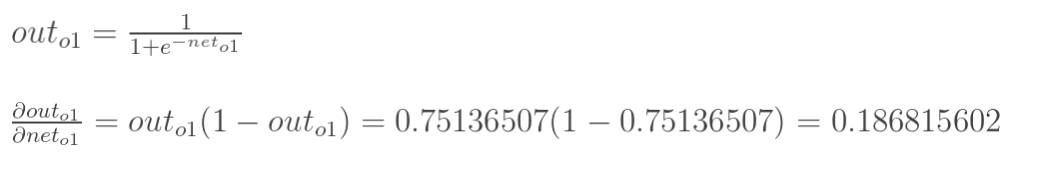


现在我们来分别计算每个式子的值：

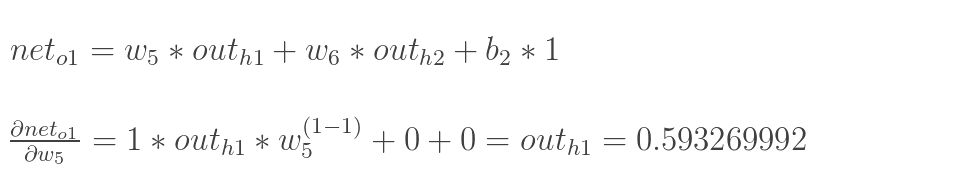
计算：



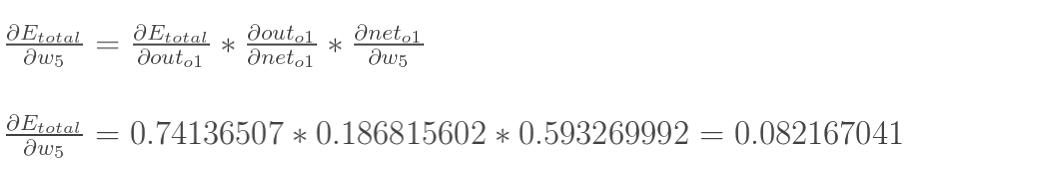
计算：



计算：



最后三者相乘：



这样我们就计算出整体误差E(total)对w5的偏导值。

最后我们来更新w5的值：



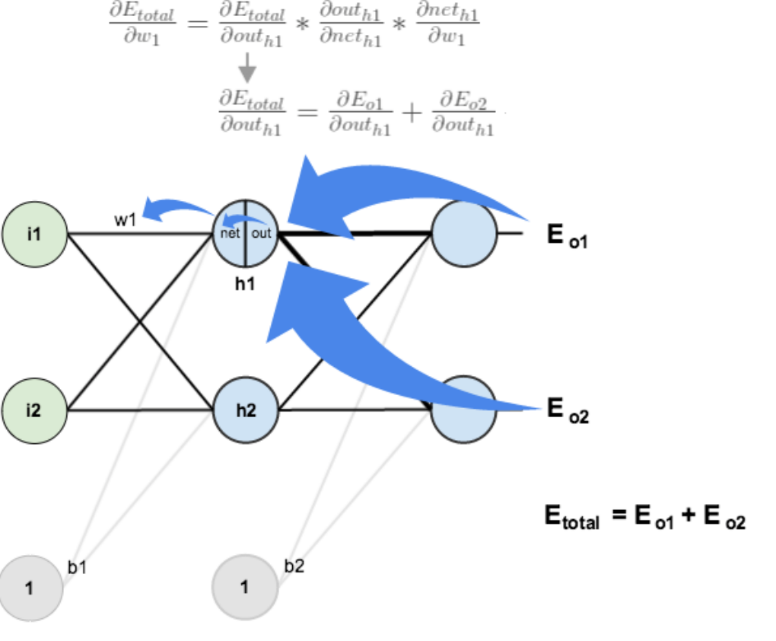
（其中，http://images2015.cnblogs.com/blog/853467/201606/853467-20160630153700093-743859667.png是学习速率，这里我们取0.5）

同理，可更新w6,w7,w8:



**(3)隐含层--->隐含层的权值更新**

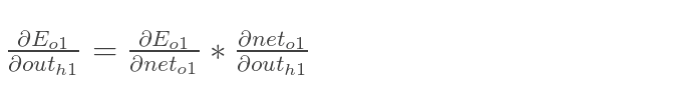
方法其实与上面说的差不多，但是有个地方需要变一下，在上文计算总误差对w5的偏导时，是从out(o1)---->net(o1)---->w5,但是在隐含层之间的权值更新时，是out(h1)---->net(h1)---->w1,而out(h1)会接受E(o1)和E(o2)两个地方传来的误差，所以这个地方两个都要计算。

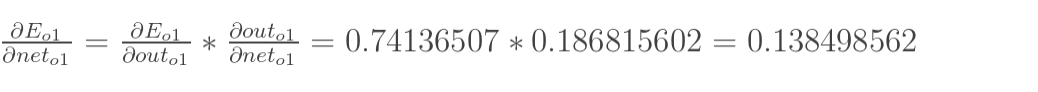


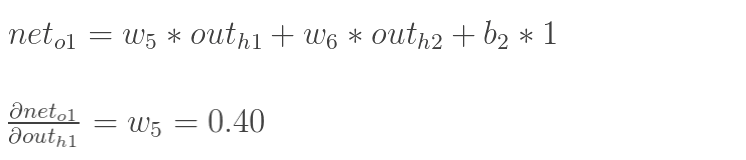
计算：



先计算：





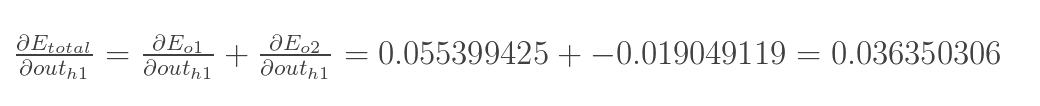




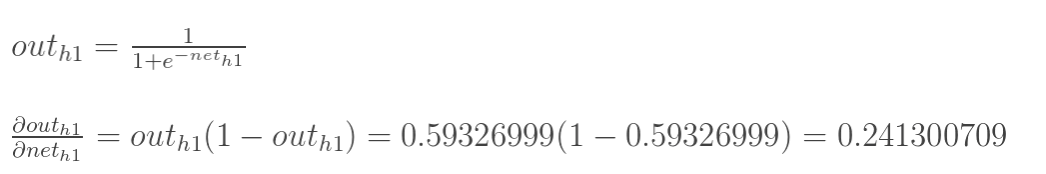
同理，计算出：



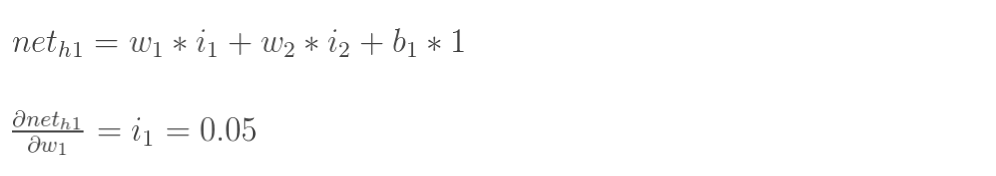
两者相加得到总值：



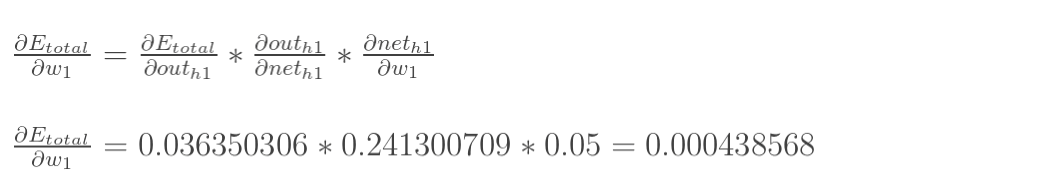
再计算：



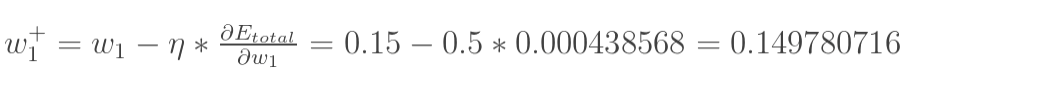
再计算：



最后，三者相乘：



最后，更新w1的权值：



同理，额可更新w2,w3,w4的权值：



这样误差反向传播法就完成了，最后我们再把更新的权值重新计算，不停地迭代，在这个例子中第一次迭代之后，总误差E(total)由0.298371109下降至0.291027924。迭代10000次后，总误差为0.000035085，输出为[0.015912196,0.984065734](原输入为[0.01,0.99]),证明效果还是不错的。

## 3.卷积神经网络



我们需要四个基本的元素来定义一个基本的卷积网络：

1. 卷积层

2. 激励层（可选）

2. 池化层（可选）

3. 输出层

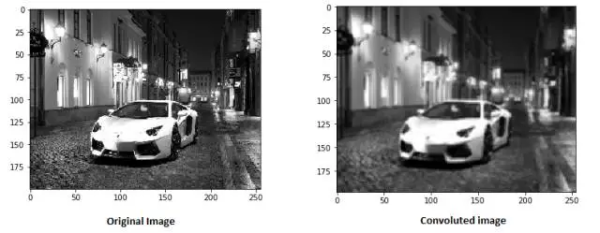
### 3.1 卷积层

假设我们有一个 6\*6 的图像。我们定义一个权值矩阵，用来从图像中提取一定的特征。下面为卷积层动图：



现在 6\*6 的图像转换成了 4\*4 的图像。想象一下权值矩阵就像用来刷墙的刷子。首先在水平方向上用这个刷子进行刷墙，然后再向下移，对下一行进行水平粉刷。当权值矩阵沿着图像移动的时候，像素值再一次被使用。实际上，这样可以使参数在卷积神经网络中被共享。

下面我们以一个真实图像为例。



**步长和边界的概念：**

像我们在上面看到的一样，过滤器或者说权值矩阵，在整个图像范围内一次移动一个像素。我们可以把它定义成一个超参数（hyperparameter），从而来表示我们想让权值矩阵在图像内如何移动。如果权值矩阵一次移动一个像素，我们称其步长为 1。下面我们看一下步长为 2 时的情况。



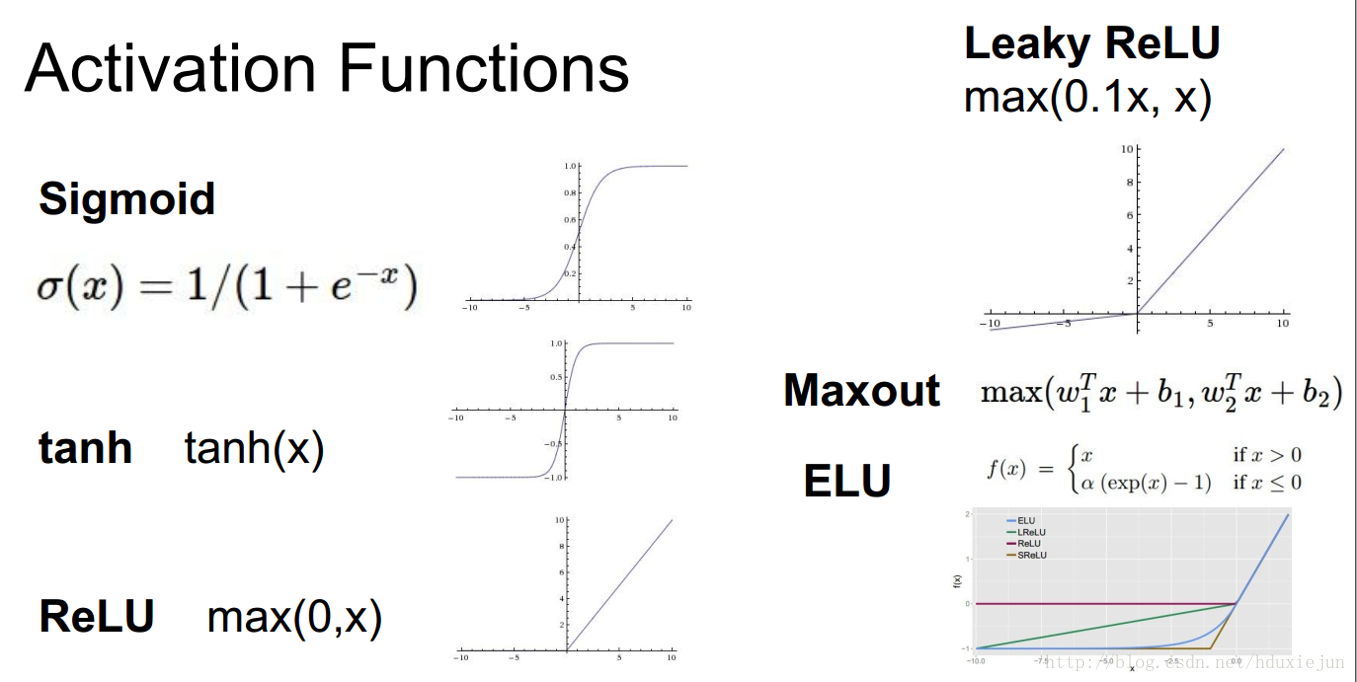
你可以看见当我们增加步长值的时候，图像的规格持续变小。在输入图像四周填充 0 边界可以解决这个问题。我们也可以在高步长值的情况下在图像四周填加不只一层的 0 边界。



### 3.2 激励层

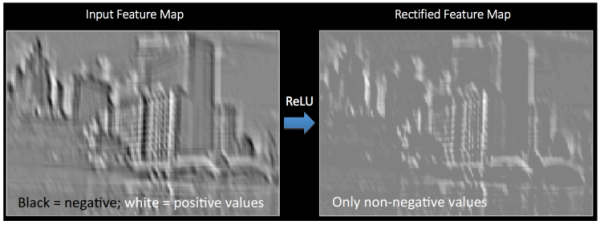
激励层使用一个激活函数将输入压缩到一个范围中，典型的[0,1][-1,1]。

神经网络中如果不加入激活函数，其一定程度可以看成线性表达，最后的表达能力不好，如果加入一些非线性的激活函数，整个网络中就引入了非线性部分，增加了网络的表达能力。目前比较流行的激活函数主要分为以下7种：



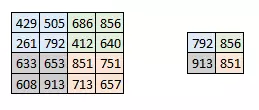
比较常用的激活函数是Relu，人们有许多喜欢relu的理由，但是最重要的一点就是它非常的易于实现，如果数值是负数则输出0，否则输出本身。这种函数运算简单，所以训练网络也非常快。函数将矩阵x内所有负值都设为零，其余的值不变。ReLU函数的计算是在卷积之后进行的。

下图为Relu前后对比图。



### 3.3 池化层

有时图像太大，我们需要减少训练参数的数量，它被要求在随后的卷积层之间周期性地引进池化层。池化的唯一目的是减少图像的空间大小。池化在每一个纵深维度上独自完成，因此图像的纵深保持不变。池化层的最常见形式是最大池化。



在这里，我们把步幅定为 2，池化尺寸也为 2。最大化执行也应用在每个卷机输出的深度尺寸中。正如你所看到的，最大池化操作后，4\*4 卷积的输出变成了 2\*2。

让我们看看最大池化在真实图片中的效果如何。

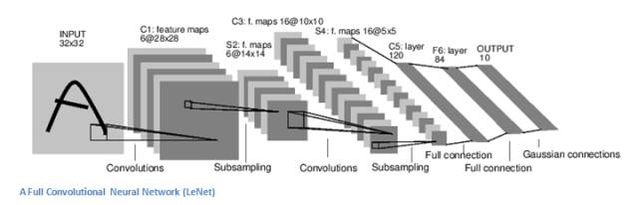


正如你看到的，我们卷积了图像，并最大池化了它。最大池化图像仍然保留了汽车在街上的信息。如果你仔细观察的话，你会发现图像的尺寸已经减半。这可以很大程度上减少参数。

### 3.4 输出层

在多层卷积和填充后，我们需要以类的形式输出。卷积和池化层只会提取特征，并减少原始图像带来的参数。然而，为了生成最终的输出，我们需要应用全连接层来生成一个等于我们需要的类的数量的输出。仅仅依靠卷积层是难以达到这个要求的。卷积层可以生成 3D 激活图，而我们只需要图像是否属于一个特定的类这样的内容。输出层具有类似分类交叉熵的损失函数，用于计算预测误差。一旦前向传播完成，反向传播就会开始更新权重与偏差，以减少误差和损失。

### 3.5 一个完整的卷积神经网络



从左至右依次为卷积→子采样→卷积→子采样→全连接→全连接→高斯连接。

### 3.6 DropOut层

Dropout的目的是用来减少overfitting(过拟合)。

Dropout的思想是训练整体DNN，并平均整个集合的结果，而不是训练单个DNN。DNNs是以概率P舍弃部分神经元，其它神经元以概率q=1-p被保留，舍去的神经元的输出都被设置为零。