反向传播算法（过程及公式推导）

一、反向传播的由来

在我们开始DL的研究之前，需要把ANN—人工神经元网络以及bp算法做一个简单解释。
关于ANN的结构，我不再多说，网上有大量的学习资料，主要就是搞清一些名词：
输入层/输入神经元，输出层/输出神经元，隐层/隐层神经元，权值，偏置，激活函数

接下来我们需要知道ANN是怎么训练的，假设ANN网络已经搭建好了，在所有应用问题中（不管是网络结

构，训练手段如何变化）我们的目标是不会变的，那就是网络的权值和偏置最终都变成一个最好的值，这个值可以让我们由输入可以得到理想的输出，于是问题就变成了y=f(x，w，b)（x是输入，w是权值，b为偏置，所有这些量都可以有多个，比如多个x1，x2，x3......最后f()就好比我们的网络它一定可以用一个函数来表示，我们不需要知道f(x)具体是怎样的函数，从小我们就认为只要是函数就一定要是可表示的，像f(x)=sin(x)一样，但是请摈弃这样的错误观念，我们只需要知道一系列的w和b决定了一个函数f（x），这个函数让我们由输入可以计算出合理的y）

最后的目标就变成了尝试不同的w，b值，使得最后的y=f（x）无限接近我们希望得到的值t

但是这个问题依然很复杂，我们把它简化一下，让（y-t）^2的值尽可能的小。于是原先的问题化为了C（w，b）=（f（x，w，b）-t）^2取到一个尽可能小的值。这个问题不是一个困难的问题，不论函数如何复杂，如果C降低到了一个无法再降低的值，那么就取到了最小值（假设我们不考虑局部最小的情况）

如何下降？数学告诉我们对于一个多变量的函数f(a,b,c,d,......)而言，我们可以求得一个向量，它称作该函数的梯度，要注意的是，梯度是一个方向向量，它表示这个函数在该点变化率最大的方向（这个定理不详细解释了，可以在高等数学教材上找到）于是C（w，b）的变化量ΔC就可以表示成

$$\frac{\partial C}{\partial w_1}\Delta w_1 + \frac{\partial C}{\partial w_2}\Delta w_2 + \ldots\ldots + \frac{\partial C}{\partial b_1}\Delta b_1 + \frac{\partial C}{\partial b_2}\Delta b_2 + \ldots\ldots$$

其中

$\Delta w_1 \;\; \Delta w_2 \ldots\ldots \Delta b_1 \;\; \Delta b_2 + \ldots\ldots$是该点上的微小变化，我们可以随意指定这些微小变化，只需要保证ΔC<0就可以了，但是为了更快的下降，我们为何不选在梯度方向上做变化呢？

29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	1	2
3	4	5	6	7	8	9

搜索

找找看

随笔分类

ARM编程 NEON(7)

ARM与嵌入式Linux(18)

C++学习(62)

GPU (1)

Jetson(1)

Linux学习(15)

事实上，梯度下降的思想就是这样考虑的，我们使得$\Delta w = -\eta \frac{\partial C}{\partial w}$从而保证C一直递减，而对于w来说只要每次更新$w' = w - \eta \frac{\partial C}{\partial w}$即可。

ok，到这里，似乎所有的问题都解决了，让我们重新整理一下思绪，我们将问题转化了很多步：
网络权值偏置更新问题 ==> f（x，w，b）的结果逼近t ==> C（w，b）=（f（x，w，b）-t）^2取极小值问题 ==> C（w，b）按梯度下降问题 ==>取到极小值，网络达到最优

千万别忘了一点！！推导基于一个前提：我们已经提前知道了当前点的梯度。然而事实并非如此！！
这个问题困扰了NN研究者多年，1969年M.Minsky和S.Papert所著的《感知机》一书出版，它对单层神经网络进行了深入分析，并且从数学上证明了这种网络功能有限，甚至不能解决象"异或"这样的简单逻辑运算问题。同时，他们还发现有许多模式是不能用单层网络训练的，而对于多层网络则没有行之有效的低复杂度算法，最后他们甚至认为神经元网络无法处理非线性问题。然而于1974年，Paul Werbos首次给出了如何训练一般网络的学习算法—back propagation。这个算法可以高效的计算每一次迭代过程中的梯度，让以上我们的推导得以实现！！
不巧的是，在当时整个人工神经网络社群中无人知晓Paul所提出的学习算法。直到80年代中期，BP算法才重新被David Rumelhart、Geoffrey Hinton及Ronald Williams、David Parker和Yann LeCun独立发现，并获得了广泛的注意，引起了人工神经网络领域研究的第二次热潮。

二、原理的引入

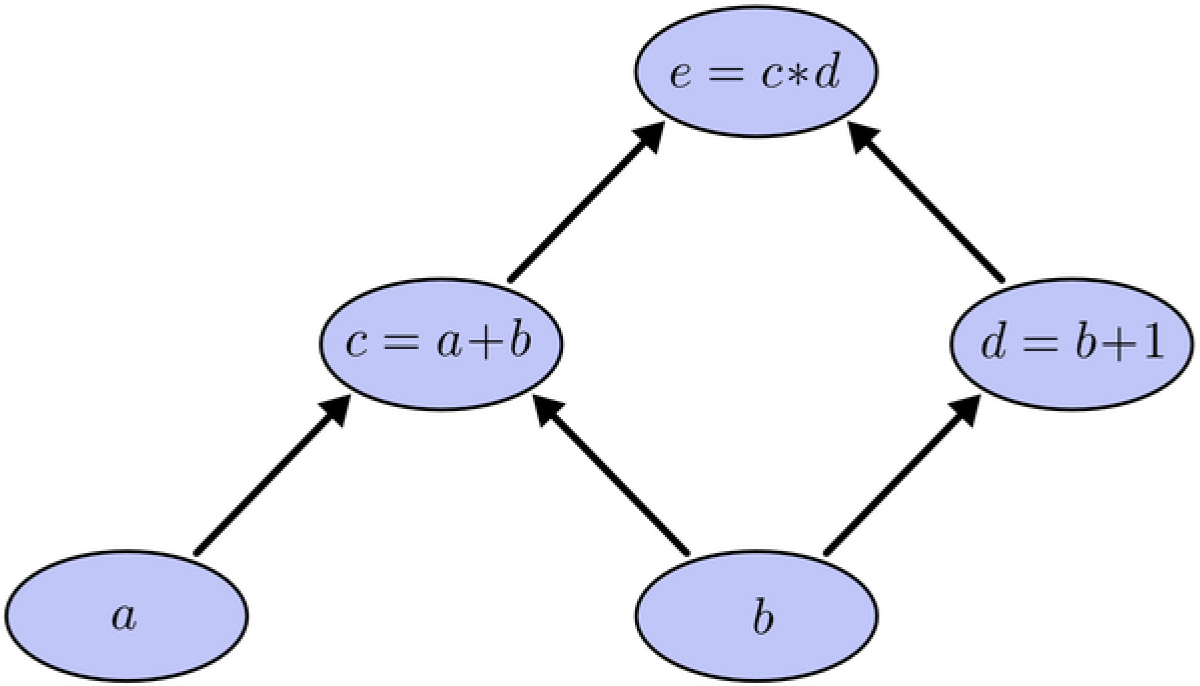
上面已经提到，所谓反向传播，就是计算梯度的方法。对于反向传播，先不急着介绍它的原理，很多文章直接引入公式，反而使得我们很难去理解。这里先引入知乎上某位大神的回答。

MCU-SKEA(3)
OpenCV(7)
perl语言(2)
ROS(14)
SQL(2)
TinyOS(5)
Ubuntu学习(8)
机器学习(41)
基础数学(15)
更多

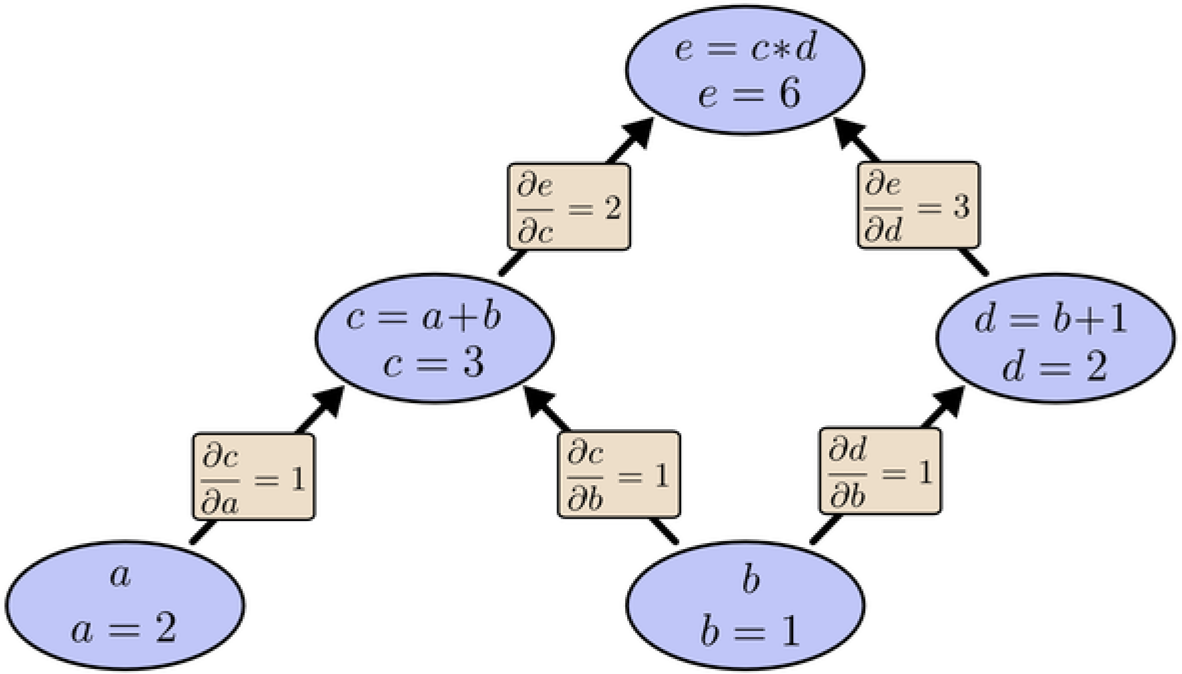
随笔档案

2019年6月(1)

来源：知乎https://www.zhihu.com/question/27239198?rf=24827633



假设输入a=2，b=1，在这种情况下，我们很容易求出相邻节点之间的偏导关系



利用链式法则：

$\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$ 以及 $\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$

$\frac{\partial e}{\partial a}$ 的值等于从a到e的路径上的偏导值的乘积，而 $\frac{\partial e}{\partial b}$ 的值等于从b到e的路径1(b-c-e)上的偏导值的乘积加上路径2(b-d-e)上的偏导值的乘积。也就是说，对于上层节点p和下层节点q，要求得 $\frac{\partial p}{\partial q}$，需要找到从q节点到p节点的所有路径，并且对每条路径，求得该路径上的所有偏导数之乘积，然后将所有路径的 “乘积” 累加起来才能得到 $\frac{\partial p}{\partial q}$ 的值。

2017年11月(9)
2017年10月(5)
更多
ARM与嵌入式Linux
Qt资源
NVIDIA Jetson TK1学习与开发
U-BOOT-2016.07移植
VIM插件: EASYMOTION[快速跳转]
其他
Introduction to A*
BM3D算法

这种情况下偏导很容易求得，因为我们已经知道网络的函数关系式，e=（a+b）*（b+1），这是一个没有权值干预，已知输入与输出之间关系的网络。实际当中我们只是知道e与输出之间的关系，就是上面说的C=（y-t）^2，而且会有成千上万的权值和偏置干预求导的过程。那么换个思路，能不能求输出对结果的偏导呢?

再利用上图的关系。节点c对e偏导2并将结果堆放起来，节点d对e偏导3并将结果堆放起来，至此第二层完毕，求出各节点总堆放量并继续向下一层发送。节点c向a发送2*1并对堆放起来，节点c向b发送2*1并堆放起来，节点d向b发送3*1并堆放起来，至此第三层完毕，节点a堆放起来的量为2，节点b堆放起来的量为2*1+3*1=5, 即顶点e对b的偏导数为5。简要的概括，就是从最上层的节点e开始，以层为单位进行处理。对于e的下一层的所有子节点，将1乘以e到某个节点路径上的偏导值，并将结果“堆放”在该子节点中。等e所在的层按照这样传播完毕后，第二层的每一个节点都“堆放"些值，然后我们针对每个节点，把它里面所有“堆放”的值求和，就得到了顶点e对该节点的偏导。然后将这些第二层的节点各自作为起始顶点，初始值设为顶点e对它们的偏导值，以"层"为单位重复上述传播过程，即可求出顶点e对每一层节点的偏导数。

三、一个很好的例子

现在，我们再把权值考虑进去，以下是一个很好的例子，有助于我们去理解反向传播
来源：Charlotte77的博客http://www.cnblogs.com/charlotte77/p/5629865.html

假设，你有这样一个网络层：

徐立Homepage

潘金山Homepage

C++设计模式

Python快速教程

支持向量机通俗导论（理解SVM的三层境界）

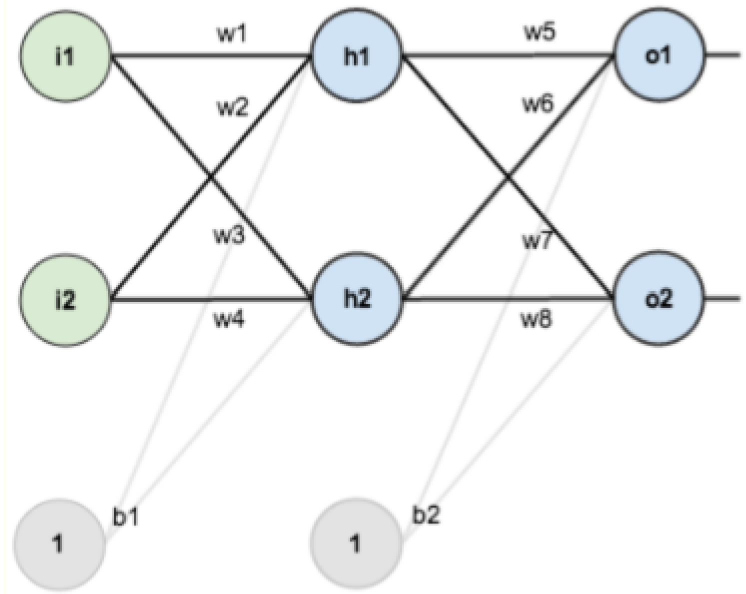
拉格朗日乘子法、KKT条件、拉格朗日对偶性

Winograd

阅读排行榜

1. 反向传播算法（过程及公式推导）(105529)

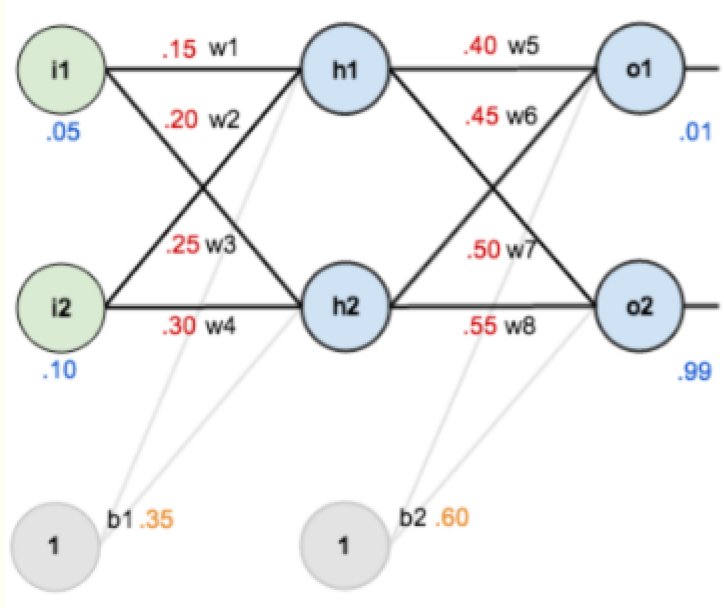
2. Bresenham直线算法与画圆算法(36886)



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：

评论排行榜



其中，输入数据 i1=0.05，i2=0.10;

输出数据 o1=0.01,o2=0.99;

初始权重 w1=0.15,w2=0.20,w3=0.25,w4=0.30;

w5=0.40,w6=0.45,w7=0.50,w8=0.88

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。

Step 1 前向传播

1.输入层---->隐含层:

计算神经元h1的输入加权和:

推荐排行榜

1. 反向传播算法（过程及公式推导）(20)
2. [转]矩阵求导实例(3)
3. [转]CMake cache(2)
4. [转]MPEG-4视频编解码知识点(2)
5. [转] A*寻路算法C++简单实现(2)

最新评论

1. Re:[转]GCC系列: __attribute__((visibility("")))

赞，简洁明了

--郝姬友

2. Re:反向传播算法（过程及公式推导）

非常棒，支持博主

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

神经元h1的输出o1:(此处用到激活函数为sigmoid函数)：

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$

同理，可计算出神经元h2的输出o2：

$$out_{h2} = 0.596884378$$

2.隐含层---->输出层：

计算输出层神经元o1和o2的值：

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

$$out_{o2} = 0.772928465$$

--咸鱼铮

3. Re:反向传播算法（过程及公式推导）

相当nice

--PanLiu

4. Re:反向传播算法（过程及公式推导）

写的非常清楚， 没有废话，很适合

--yuanzhoulvpi

5. Re:反向传播算法（过程及公式推导）

对于学过微积分的人来说，都是基础知识吧，顶多一个偏导的链式法则，但是这些符号太多看的人真是头晕。

--我的锅

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。

Step 2 反向传播

1.计算总误差

总误差：(square error)

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

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：

$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274811083 + 0.023560026 = 0.298371109$$

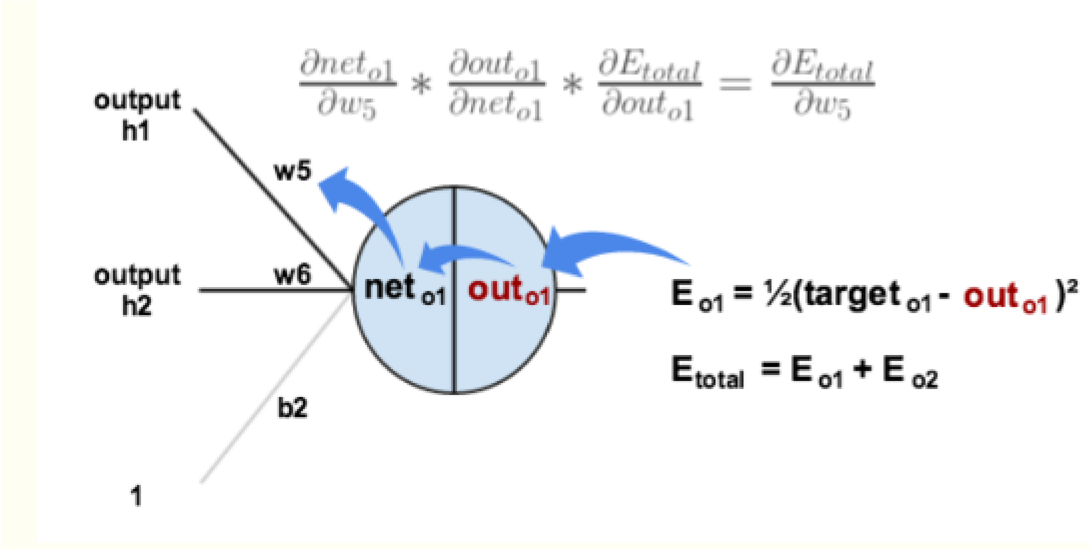
2.隐含层---->输出层的权值更新：

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：

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

（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 $\frac{\partial E_{total}}{\partial out_{o1}}$ ：

$$E_{total} = \frac{1}{2}(target_{o1} - out_{o1})^2 + \frac{1}{2}(target_{o2} - out_{o2})^2$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}(target_{o1} - out_{o1})^{2-1} * -1 + 0$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

计算 $\frac{\partial out_{o1}}{\partial net_{o1}}$ ：

$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$

$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1 - out_{o1}) = 0.75136507(1 - 0.75136507) = 0.186815602$

（这一步实际上就是对sigmoid函数求导，比较简单，可以自己推导一下）

计算 $\frac{\partial net_{o1}}{\partial w_5}$：

$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$

$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$

最后三者相乘：

$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$

$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$

这样我们就计算出整体误差E(total)对w5的偏导值。

回过头来再看看上面的公式，我们发现：

$\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1}) * out_{h1}$

为了表达方便，用δ_{o1}来表示输出层的误差：

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}}$$

$$\delta_{o1} = -(target_{o1} - out_{o1}) * out_{o1}(1 - out_{o1})$$

因此，整体误差E(total)对w5的偏导公式可以写成：

$$\frac{\partial E_{total}}{\partial w_5} = \delta_{o1} out_{h1}$$

如果输出层误差计为负的话，也可以写成：

$$\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} out_{h1}$$

最后我们来更新w5的值：

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

（其中，η是学习速率，这里我们取0.5）

同理，可更新w6,w7,w8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

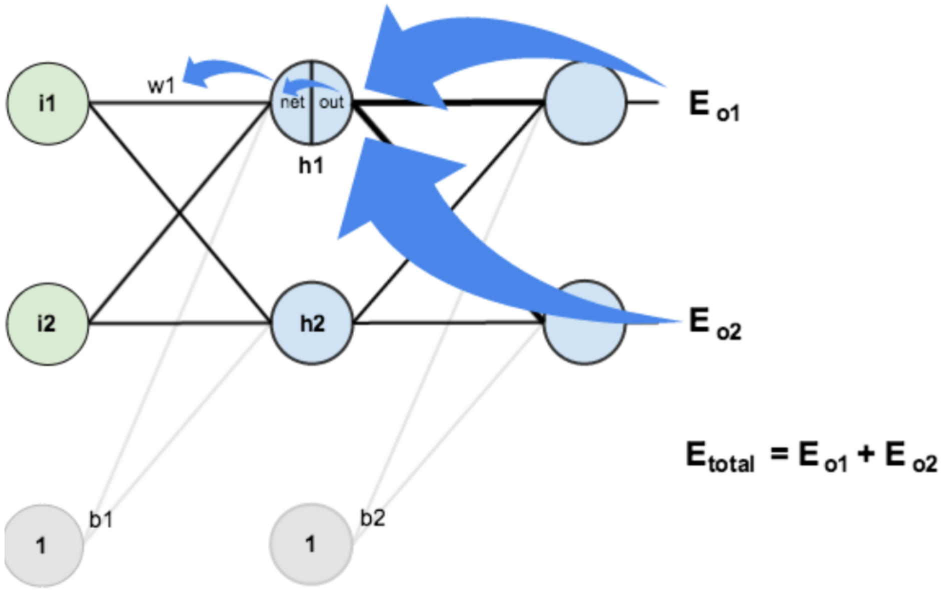
$$w_8^+ = 0.561370121$$

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。

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



计算 $\frac{\partial E_{total}}{\partial out_{h1}}$ ：

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$

先计算 $\frac{\partial E_{o1}}{\partial out_{h1}}$ ：

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}}$$

$$\frac{\partial E_{o1}}{\partial net_{o1}} = \frac{\partial E_{o1}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} = 0.74136507 * 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

$$\frac{\partial E_{o1}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

同理，计算出:

$$\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$$

两者相加得到总值:

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

再计算 $\frac{\partial out_{h1}}{\partial net_{h1}}$ ：

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$$

再计算 $\frac{\partial net_{h1}}{\partial w_1}$ ：

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

最后，三者相乘：

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

为了简化公式，用sigma(h1)表示隐含层单元h1的误差：

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho}\right) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

最后，更新w1的权值：

$$w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$$

同理，额可更新w2,w3,w4的权值：

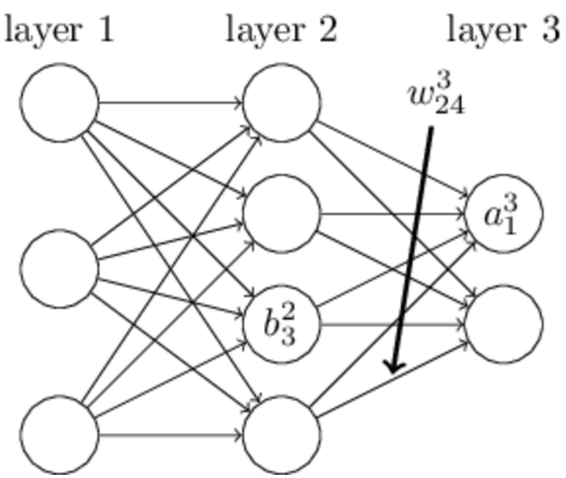
$$w_2^+ = 0.19956143$$

$$w_3^+ = 0.24975114$$

$$w_4^+ = 0.29950229$$

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

四、最一般的情况



上图是一个三层人工神经网络，layer1至layer3分别是输入层、隐藏层和输出层。如图，先定义一些变量：

w_{jk}^{l}表示第l-1层的第k个神经元连接到第l层的第j个神经元的权重；

b_{j}^{l}表示第l层的第j个神经元的偏置；

z_{j}^{l}表示第l层的第j个神经元的输入，即：

$$z_{j}^{l} = \sum_{k} w_{jk}^{l} a_{k}^{l-1} + b_{j}^{l}$$

a_{j}^{l}表示第l层的第j个神经元的输出，即：

$$a_{j}^{l} = \sigma\left(\sum_{k} w_{jk}^{l} a_{k}^{l-1} + b_{j}^{l}\right)$$

其中σ表示激活函数。

L表示神经网络的最大层数，也可以理解为输出层。

将第l层第j个神经元中产生的错误（即实际值与预测值之间的误差）定义为：

$$\delta_{j}^{l} \equiv \frac{\partial C}{\partial z_{j}^{l}}$$

代价函数，依然用C来表示

Summary: the equations of backpropagation

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \quad \text{(BP1)}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \quad \text{(BP2)}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad \text{(BP3)}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad \text{(BP4)}$$

以上4个方程中，第一个方程其实不难理解，就是求输出对估价函数C的偏导。

唯一比较困难的，就是第二个方程，它给出了根据下一层的错误量δl+1计算δl的等式。为证明该等式，我们先依据δkl+1=∂C/∂zkl+1重新表达下等式δlj =∂C/∂zlj。这里可以应用链式法则：

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}$$

在最后一行，我们互换了下表达式右侧的两项，并取代了 δkl+1的定义。为了对最后一行的第一项求值，注意：

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

作微分，我们得到

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1}\sigma'(z_j^l)$$

代回 (42) 我们得到

$$\delta_j^l = \sum_k w_{kj}^{l+1}\delta_k^{l+1}\sigma'(z_j^l)$$

这就是以分量形式呈现的 (BP2)。后两式在完成了BP2证明之后就不太难了，留给读者来证明。

四、证明

反向传播算法（Backpropagation）是目前用来训练人工神经网络（Artificial Neural Network，ANN）的最常用且最有效的算法。其主要思想是：

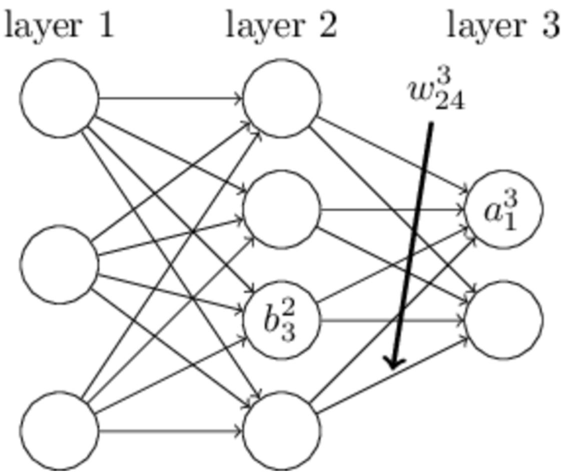
（1）将训练集数据输入到ANN的输入层，经过隐藏层，最后达到输出层并输出结果，这是ANN的前向传播过程；

（2）由于ANN的输出结果与实际结果有误差，则计算估计值与实际值之间的误差，并将该误差从输出层向隐藏层反向传播，直至传播到输入层；

（3）在反向传播的过程中，根据误差调整各种参数的值；不断迭代上述过程，直至收敛。

反向传播算法的思想比较容易理解，但具体的公式则要一步步推导，因此本文着重介绍公式的推导过程。

1. 变量定义



上图是一个三层人工神经网络，layer1至layer3分别是输入层、隐藏层和输出层。如图，先定义一些变量：

w_{jk}^l 表示第 $(l-1)$ 层的第 k 个神经元连接到第 l 层的第 j 个神经元的权重；

b_j^l 表示第 l 层的第 j 个神经元的偏置；

z_j^l 表示第 l 层的第 j 个神经元的输入，即：

$$z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$$

a_j^l 表示第 l 层的第 j 个神经元的输出，即：

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right)$$

其中 σ 表示激活函数。

2. 代价函数

代价函数被用来计算ANN输出值与实际值之间的误差。常用的代价函数是二次代价函数（Quadratic cost function）：

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

其中，x 表示输入的样本，y 表示实际的分类，a^L 表示预测的输出，L 表示神经网络的最大层数。

3. 公式及其推导

本节将介绍反向传播算法用到的4个公式，并进行推导。**如果不想了解公式推导过程，请直接看第4节的算法步骤。**

首先，将第 l 层第 j 个神经元中产生的错误（即实际值与预测值之间的误差）定义为：

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$$

本文将以一个输入样本为例进行说明，此时代价函数表示为：

$$C = \frac{1}{2}\|y - a^L\|^2 = \frac{1}{2}\sum_j (y_j - a_j^L)^2$$

公式1（计算最后一层神经网络产生的错误）：

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

其中，$\odot$表示Hadamard乘积，用于矩阵或向量之间点对点的乘法运算。公式1的推导过程如下：

$$\because \ \delta_j^L = \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L}$$

$$\therefore \ \delta^L = \frac{\partial C}{\partial a^L} \odot \frac{\partial a^L}{\partial z^L} = \nabla_a C \odot \sigma'(z^L)$$

公式2（由后往前，计算每一层神经网络产生的错误）：

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

推导过程：

$$\because \delta_j^l = \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l}$$

$$= \sum_k \delta_k^{l+1} \cdot \frac{\partial (w_{kj}^{l+1} a_j^l + b_k^{l+1})}{\partial a_j^l} \cdot \sigma'(z_j^l)$$

$$= \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma'(z_j^l)$$

$$\therefore \delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$

公式3（计算权重的梯度）：

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

推导过程：

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \cdot \frac{\partial (w_{jk}^l a_k^{l-1} + b_j^l)}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

公式4（计算偏置的梯度）：

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

推导过程：

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \cdot \frac{\partial (w_{jk}^l a_k^{l-1} + b_j^l)}{\partial b_j^l} = \delta_j^l$$

4. 反向传播算法伪代码

- **输入训练集**

- **对于训练集中的每个样本x，设置输入层（Input layer）对应的激活值a^1：**
  - **前向传播：**

$$z^l = w^l a^{l-1} + b^l, \ a^l = \sigma(z^l)$$

  - **计算输出层产生的错误：**

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

  - **反向传播错误：**

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$$