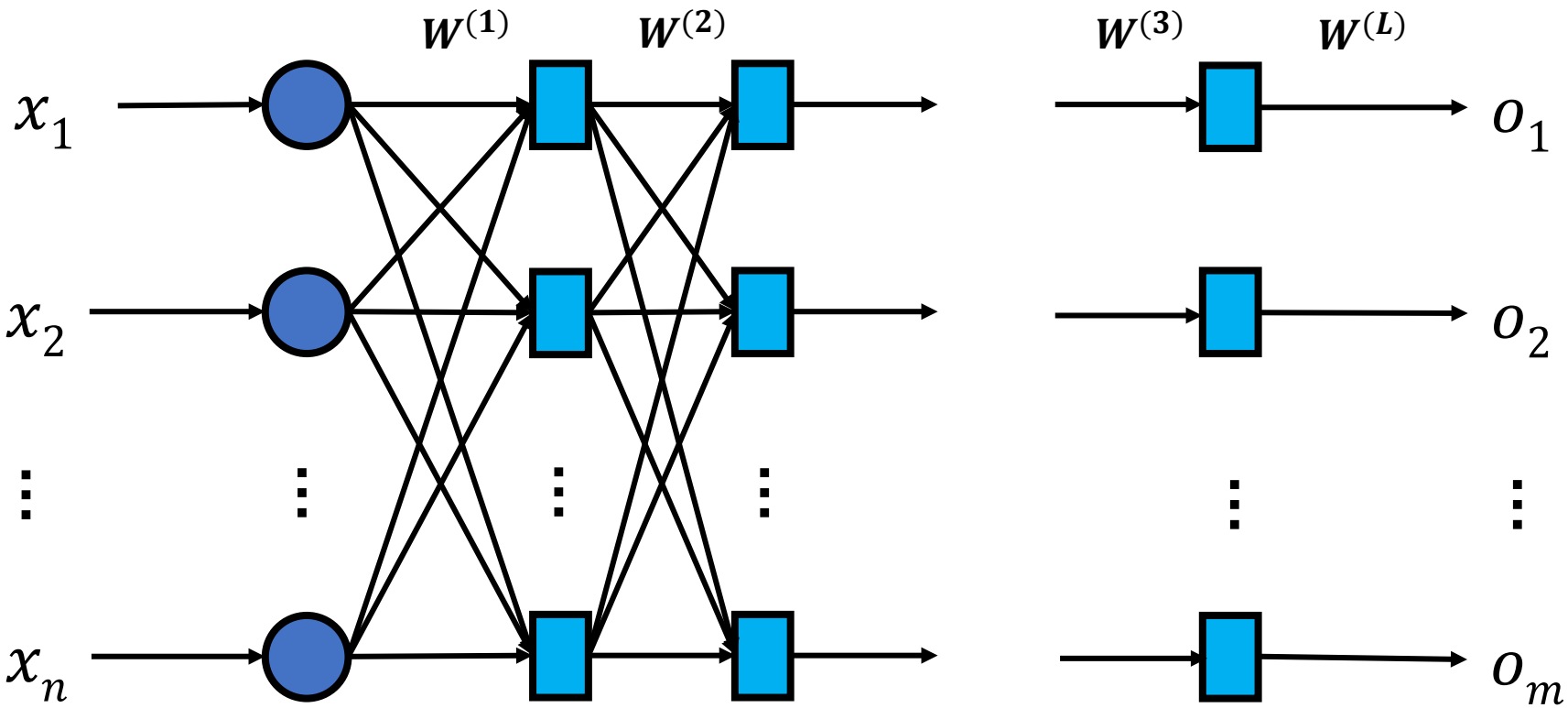
第五讲 多层感知器

一、概 述

多层感知器的拓扑结构



网络的拓扑结构

1. 多层感知器拓扑结构特点:
    1. 层级结构。
    2. 超参数：输入向量、输出向量的维数、网络隐藏层的层数和各个隐藏层神经元的个数，人为设计。
    3. 多层感知器结构一般都选用二级网络。

网络的拓扑结构

多层感知器泛化性能：

多层感知器是单层感知器的加深结构，更深层的网络所表达的数学形式更复杂，能够拟合更复杂的分布。

相比于宽度，增加深度更能使网络具有复杂的表达，但也会带来学习上的一些问题（不收敛或难收敛等）

实验：增加隐藏层的层数和隐藏层神经元个数不一定总能够提高网络精度和表达能力。

多层感知器的基本架构

神经元的输入:

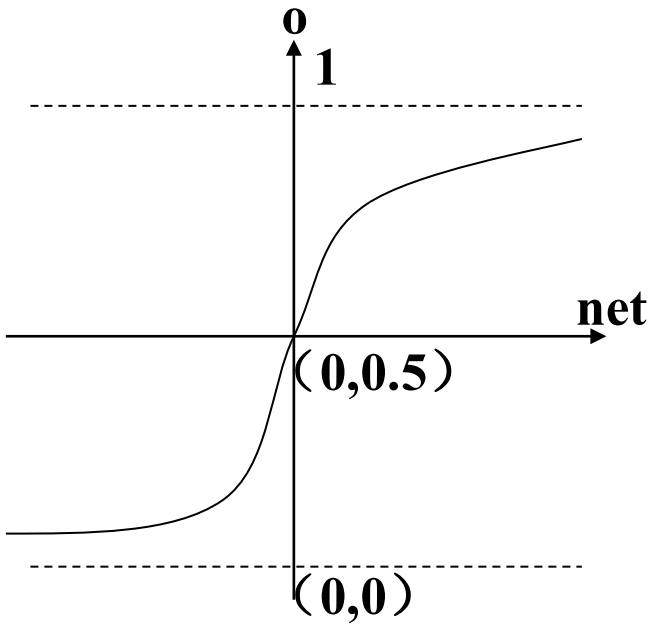
$$net_i = x_1 w_{1i} + x_2 w_{2i} + \cdots + x_n w_{ni}$$

神经元的输出:

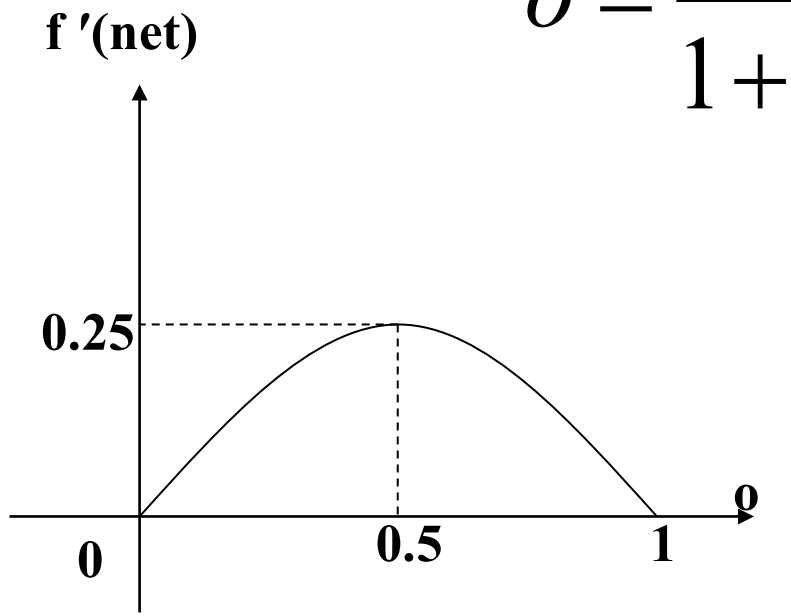
$$o = f(net) = \frac{1}{1 + e^{-net}}$$

$$f'(net) = -\frac{1}{(1 + e^{-net})^2}(-e^{-net}) = o - o^2 = o(1 - o)$$

输出函数分析



$$o = \frac{1}{1+e^{-net}}$$



- 应该将net的值尽量控制在收敛比较快的范围内
- 可以用其它的函数作为激活函数，只要该函数连续可导

- 我们手工设置了权重和感知器结构，但对于容量稍大的网络，手工设计所有参数成本较大。

- 是否存在一种方法可以自动地训练感知器学习参数?

二、BP网络的训练—基本算法

多层感知器的训练——BP算法

- BP算法的出现

  非循环多级网络的训练算法

  UCSD PDP 小 组 的 Rumelhart、Hinton 和 Williams1986年独立地给出了BP算法清楚而简单的描述

  1982年，Paker就完成了相似的工作

  1974年，Werbos已提出了该方法

- 弱点：训练速度非常慢、局部极小点的逃离问题、算法不一定收敛
- 优点：广泛的适应性和有效性

训练过程概述

样本：(输入向量，理想输出向量)

权初始化：

“小随机数”与饱和状态；“不同”保证网络可以学。

- 向前传播阶段：

（1）从样本集中取一个样本$(\boldsymbol{X}_p, \boldsymbol{Y}_p)$，将$\boldsymbol{X}_p$输入网络；

（2）计算相应的实际输出$\boldsymbol{O}_p$：

$$\boldsymbol{O}_p = FL(\ldots(F_2(F_1(\boldsymbol{X}_p\boldsymbol{W}^{(1)})\boldsymbol{W}^{(2)})\ldots)\boldsymbol{W}^{(L)})$$

训练过程概述

- 向后传播阶段——误差传播阶段:

(1) 计算实际输出$\boldsymbol{O}_p$与相应的理想输出$\boldsymbol{Y}_p$的差;

(2) 按极小化误差的方式调整权矩阵。

(3) 网络关于第p个样本的误差测度:

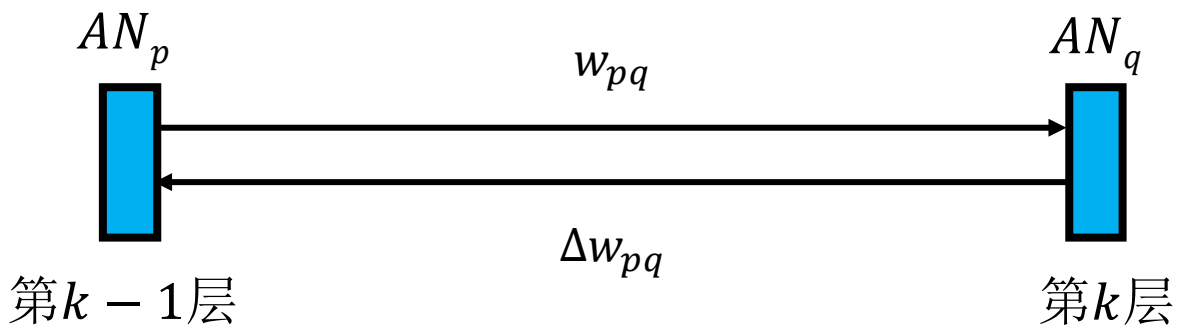
$$E_p = \frac{1}{2}\sum_{j=1}^{m}\left(y_{pj} - o_{pj}\right)^2$$

(4) 网络关于整个样本集的误差测度:

$$E = \sum_{p} E_p$$

误差传播分析

1. 输出层权的调整



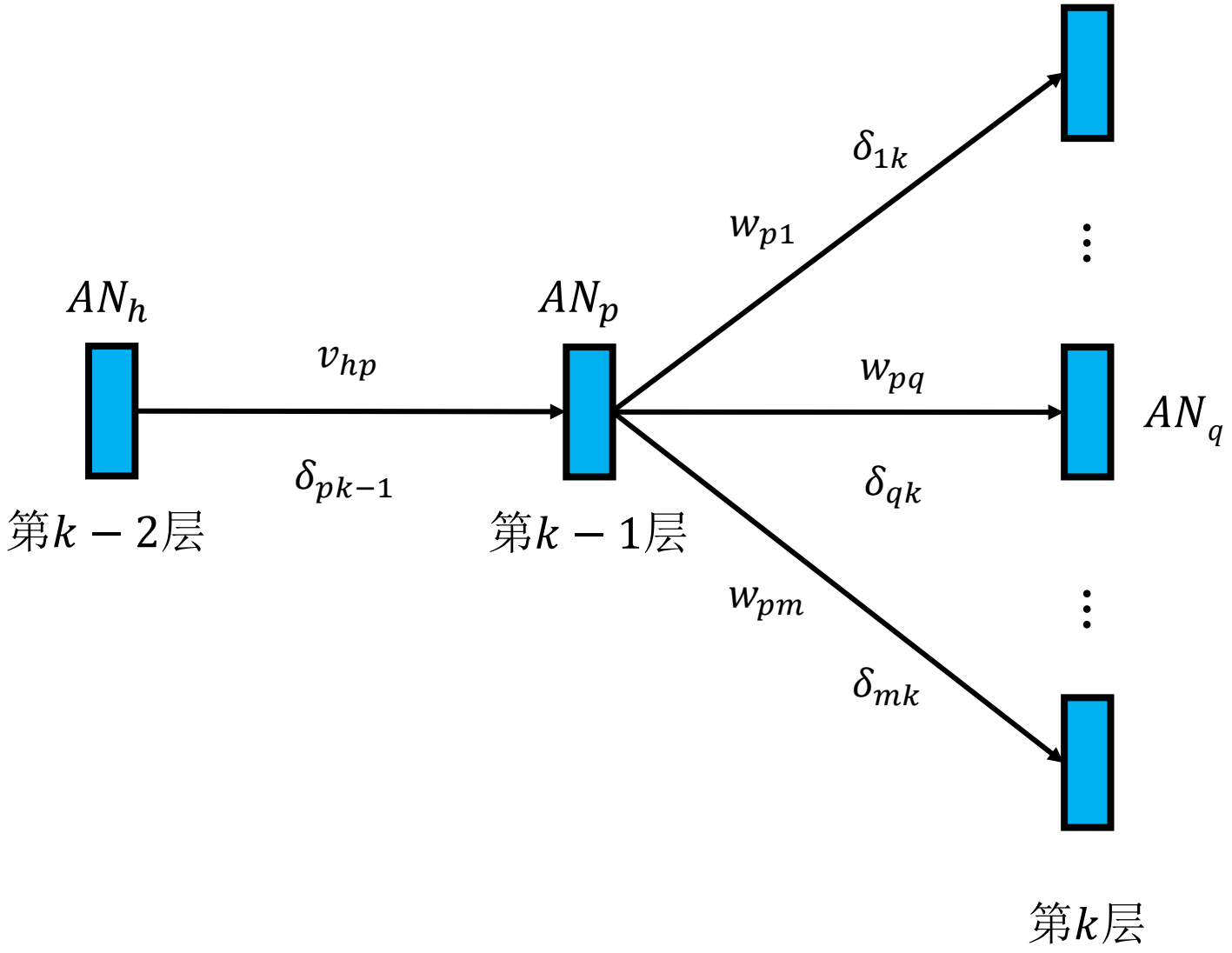
$$w_{pq} = w_{pq} + \Delta w_{pq}$$

$$\Delta w_{pq} = \alpha \delta_q o_p$$

$$= \alpha f_k'(net_q)(y_q - o_q)o_p$$

$$= o_q(1 - o_q)(y_q - o_q)o_p$$

隐藏层权的调整


AN_h
AN_p
AN_q
v_{hp}
δ_{pk-1}
w_{p1}
w_{pq}
w_{pm}
δ_{1k}
δ_{qk}
δ_{mk}
第$k-2$层
第$k-1$层
第k层

隐藏层权的调整

δ_{pk-1}的值和δ_{1k}，δ_{2k}，…，δ_{mk}有关

不妨认为δ_{pk-1}

通过权w_{p1}对δ_{1k}做出贡献，

通过权w_{p2}对δ_{2k}做出贡献，

……

通过权w_{pm}对δ_{mk}做出贡献。

$$\delta_{pk-1} = f'_{k-1}(net_p)(w_{p1}\delta_{1k} + w_{p2}\delta_{2k} + \cdots + w_{pm}\delta_{mk})$$

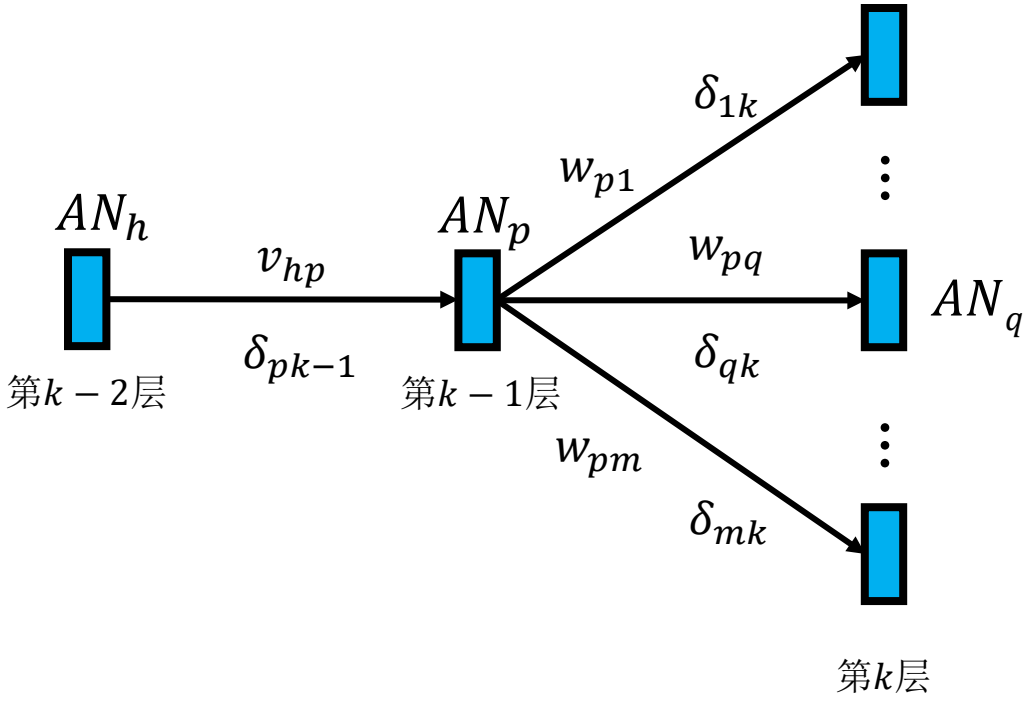
隐藏层权的调整

$$v_{hp} = v_{hp} + \Delta v_{hp}$$

$$\Delta v_{hp} = \alpha \delta_{pk-1} o_{hk-2}$$

$$= \alpha f'_{k-1}(net_p)(w_{p1}\delta_{1k} + w_{p2}\delta_{2k} + \cdots + w_{pm}\delta_{mk}) o_{hk-2}$$

$$= \alpha o_{pk-1}(1 - o_{pk-1})\,(w_{p1}\delta_{1k} + w_{p2}\delta_{2k} + \cdots + w_{pm}\delta_{mk}) o_{hk-2}$$



基本BP算法

- 样本集：$\boldsymbol{S} = \{(\boldsymbol{X}_1, \boldsymbol{Y}_1), (\boldsymbol{X}_2, \boldsymbol{Y}_2), \ldots, (\boldsymbol{X}_s, \boldsymbol{Y}_s)\}$
- 基本思想：
  - 逐一地根据样本集中的样本$(\boldsymbol{X}_p, \boldsymbol{Y}_p)$计算出实际输出$\boldsymbol{O}_p$和误差测度$E_p$，对$W^{(1)}, W^{(2)}, \ldots, W^{(L)}$各做一次调整，重复这个循环，直到$\sum_p E_p < \varepsilon$
  - 用输出层的误差调整输出层权矩阵，并用此误差估计输出层的直接前导层的误差，再用输出层前导层误差估计更前一层的误差。如此获得所有其它各层的误差估计，并用这些估计实现对权矩阵的修改，形成将输出端表现出的误差沿着与输入信号相反的方向逐级向输入端传递的过程

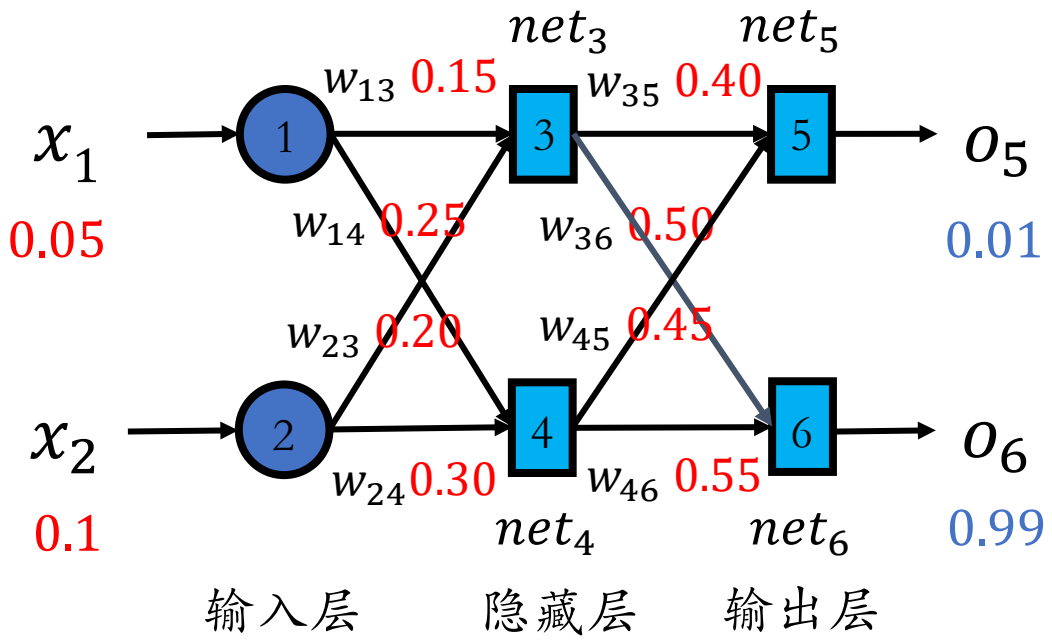
基本BP算法

Algorithm 1 基本BP算法

for k = 1 to L **do**
    初始化W^k
end for
初始化精度控制参数ϵ
$E = \epsilon + 1$
while $E > \epsilon$ **do**
    $E = 0$
    while 对S中每一个样本(X_p, Y_p) **do**
        计算出X_p对应的实际输出O_p;
        计算出E_p;
        $E = E + E_p$;
        根据相应式子调整W^L;
        $k = L - 1$;
        while $k \neq 0$ **do**
            根据相应式子调整W^k;
            $k = k - 1$.
        end while
    end while
    $E = E/2.0$
end while

例子：BP训练的实例

一、输入输出与网络初始化



训练样本：$\boldsymbol{x} = [0.05, 0.1], \boldsymbol{y} = [0.01, 0.99]$

随机初始化权矩阵：

$$\boldsymbol{W}^{(1)}: w_{13} = 0.15, w_{14} = 0.25, w_{23} = 0.20, w_{24} = 0.30$$

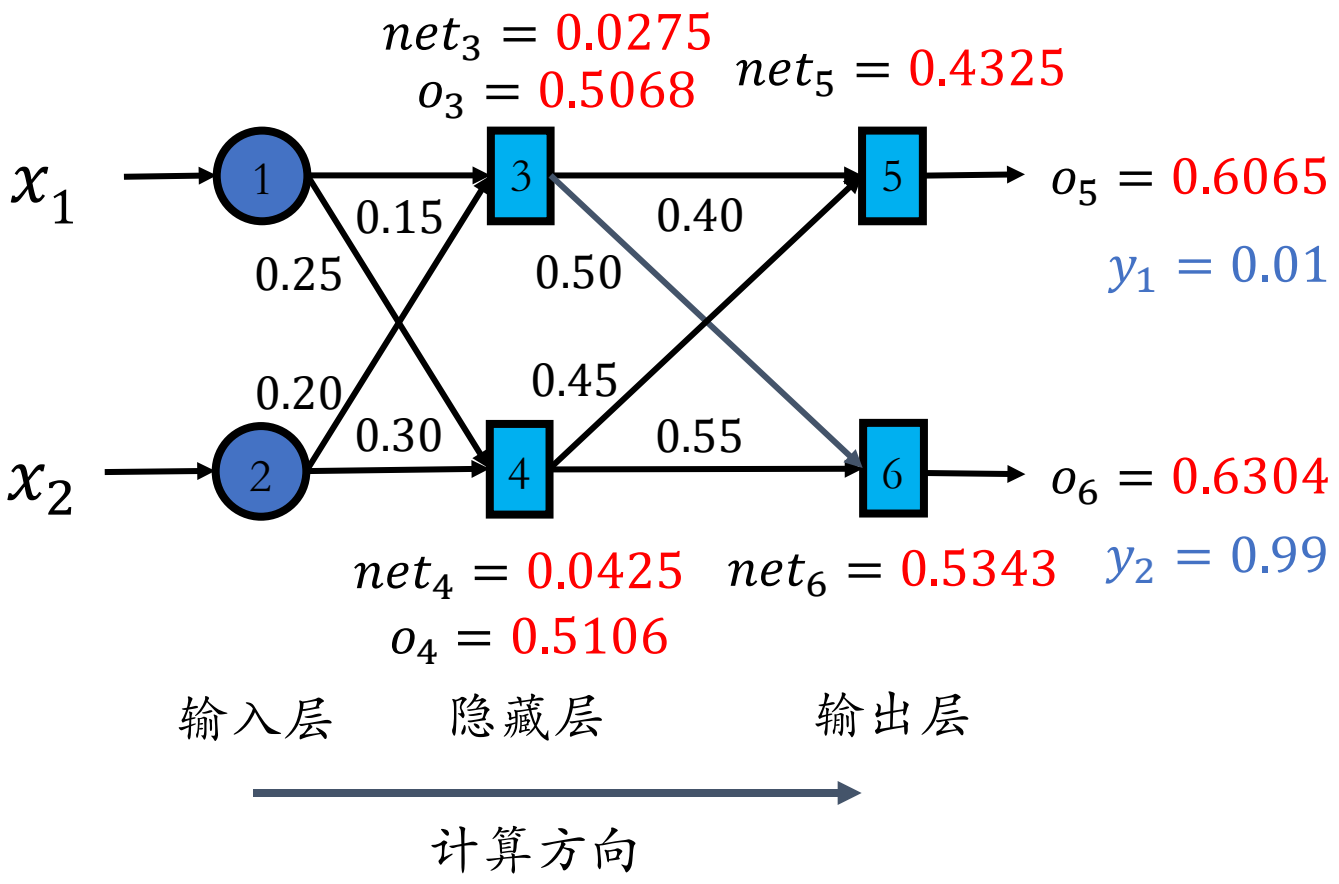
$$\boldsymbol{W}^{(2)}: w_{35} = 0.40, w_{36} = 0.50, w_{45} = 0.45, w_{46} = 0.55$$

神经元的激活函数：$f(net) = \frac{1}{1+e^{-net}}$

学习率：$\alpha = 0.5$

例子：BP训练的实例

二、前向过程



前向计算：

$$net_3 = x_1 \cdot w_{13} + x_2 \cdot w_{23}$$
$$= 0.05 \times 0.15 + 0.2 \times 0.1$$
$$= 0.0275$$

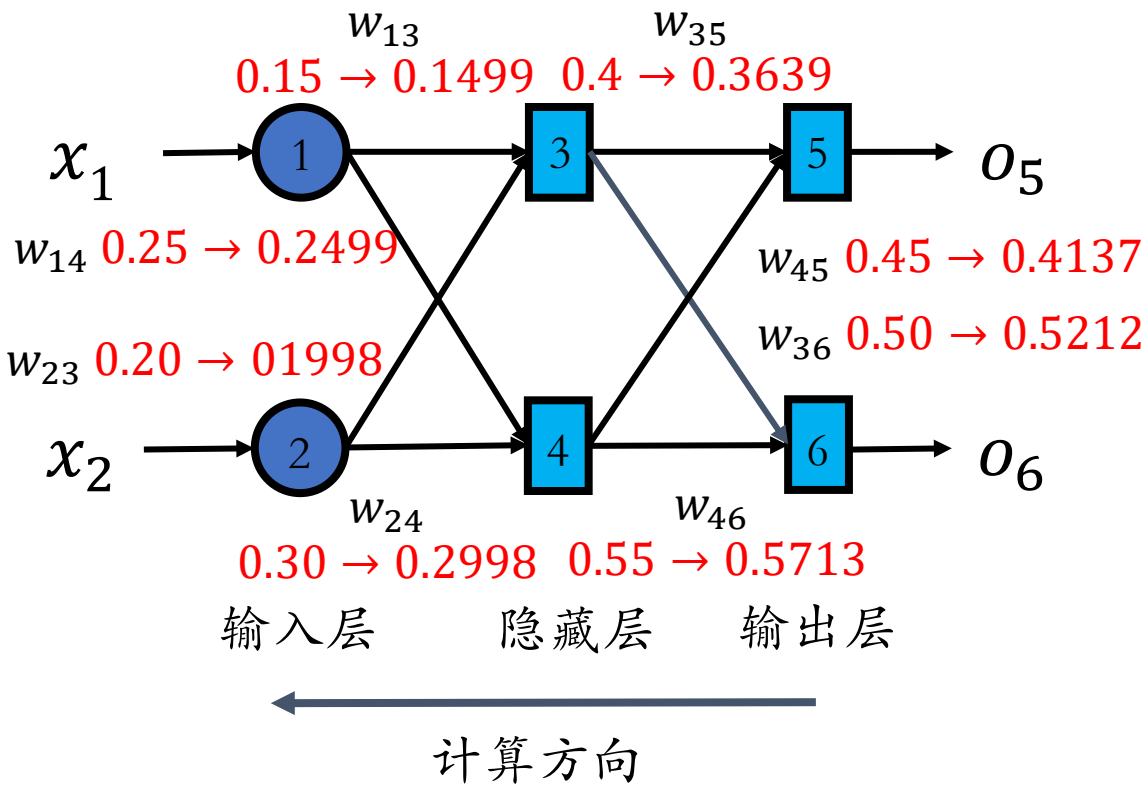
$$o_3 = \frac{1}{1+e^{-net}} = \frac{1}{1+e^{-0.0275}}$$
$$= 0.5068$$

$$net_5 = o_3 \cdot w_{35} + o_4 \cdot w_{45}$$
$$= 0.5068 \times 0.40 + 0.5106 \times 0.45$$
$$= 0.4325$$

……

例子：BP训练的实例

三、误差反向传播过程



输出层权值调整：

$$\frac{\partial o_j}{\partial net_j}=\frac{\partial(\frac{1}{1+e^{-net_j}})}{\partial net_j}=o_j(1-o_j)$$

$$\delta_5=(y_1-o_5)o_5(1-o_5)$$
$$=(0.01-0.6065)\times 0.6065\times(1-0.6065)=-0.1424$$

$$w_{35}^{+}=w_{35}+\alpha\delta_5 o_3$$
$$=0.4-0.5\times 0.1424\times 0.5068=0.3639$$

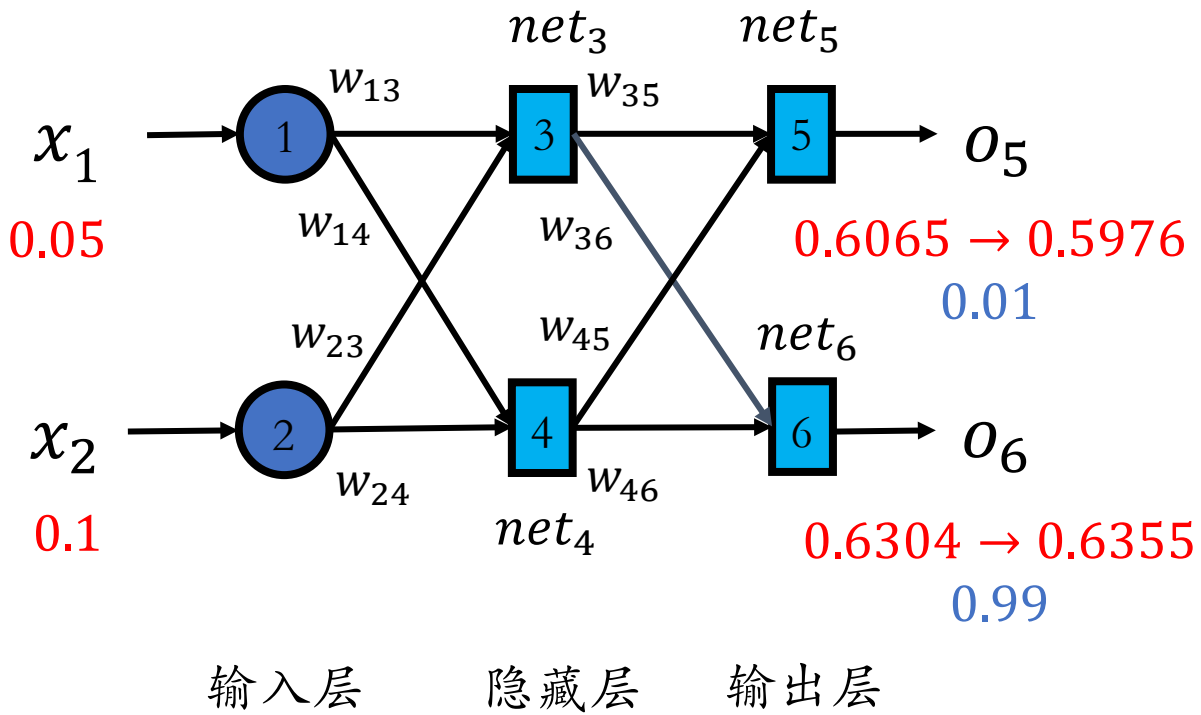
$$\delta_6=(y_2-o_6)o_6(1-o_6)$$
$$=(0.99-0.6304)\times 0.6304\times(1-0.6304)=0.0838$$

输入层权值调整：

$$w_{13}^{+}=w_{13}+\alpha o_3(1-o_3)\left(\delta_5 w_{35}+\delta_6 w_{36}\right)x_1$$
$$=0.15+0.5\times 0.5068\times(1-0.5068)\times$$
$$(-0.1424\times 0.4+0.0838\times 0.5)\times 0.05$$
$$=0.1499$$

例子：BP训练的实例

四、训练结果



第二次前向结果相比第一次前向结果更靠近目标输出，误差更小

$$E: 0.2355 \rightarrow 0.2286$$

当经过10000轮训练之后，误差将减少到$5.583e-0.5$，输出为

$$(0.0174, 0.9825)$$

三、BP算法的改进

思考：BP算法存在什么潜在的问题?

（hint：BP针对单个样例进行更新）

基本BP算法的缺陷

- 不同样例的更新的效果可能“抵消”
- BP网络接受样本的顺序对训练结果有较大影响。它更“偏爱”较后出现的样本
- 给样本安排一个适当的顺序，是非常困难的。
- 用$(\boldsymbol{X}_1, \boldsymbol{Y}_1), (\boldsymbol{X}_2, \boldsymbol{Y}_2), \dots, (\boldsymbol{X}_s, \boldsymbol{Y}_s)$的“总效果”修改$W^{(1)}, W^{(2)}, \dots, W^{(L)}$
  - $\Delta w_{ij}^{(k)} = \sum_p \Delta w_{ij}^{(k)}$

消除样本顺序影响的BP算法

Algorithm 2 消除样本顺序影响的BP算法

1: **for** k = 1 to L **do**
2: 初始化W^k
3: **end for**
4: 初始化精度控制参数ϵ
5: $\mathrm{E} = \epsilon + 1$
6: **while** $\mathrm{E} > \epsilon$ **do**
7: $\mathrm{E} = 0$
8: 对所有的i,j,k: $\triangle w_{ij}^k = 0$

对每个样本(X_p, Y_p)进行的操作

9: **while** 对S中每一个样本$(\mathrm{X}_p, \mathrm{Y}_p)$ **do**
10: 计算出X_p对应的实际输出O_p;
11: 计算出E_p;
12: $\mathrm{E} = \mathrm{E} + \mathrm{E}_p$;
13: 对所有的i,j根据相应式子计算$\triangle_p w_{ij}^{\mathrm{L}}$;
14: 对所有的i,j：$\triangle w_{ij}^{\mathrm{L}} = \triangle w_{ij}^{\mathrm{L}} + \triangle_p w_{ij}^{\mathrm{L}}$
15: $\mathrm{k} = \mathrm{L} - 1$;
16: **while** $\mathrm{k} \neq 0$ **do**
17: 对所有i,j根据相应式子计算$\triangle_p w_{ij}^{\mathrm{k}}$;
18: 对所有i,j：$\triangle w_{ij}^{\mathrm{k}} = \triangle w_{ij}^{\mathrm{k}} + \triangle_p w_{ij}^{\mathrm{k}}$
19: $\mathrm{k} = \mathrm{k} - 1$.
20: **end while**
21: 对所有i,j,k：$w_{ij}^{k} = w_{ij}^{k} + \triangle w_{ij}^{k}$;
22: **end while**
23: $\mathrm{E} = \mathrm{E}/2.0$
24: **end while**

分析

- 较好地解决了因样本的顺序引起的精度问题和训练的抖动问题
- 收敛速度：比较慢
- 偏移量：给每一个神经元增加一个偏移量来加快收敛速度
- 冲量：联接权的本次修改要考虑上次修改的影响，以减少抖动问题

分析——冲量设置

- Rumelhart等人1986年
  - $\Delta w_{ij} = \alpha \delta_j o_i + \beta \Delta w_{ij}'$
  - $\Delta w_{ij}'$为上一次的修改量，β为冲量系数，一般可取到0.9
- Sejnowski与Rosenberg，1987年
  - $\Delta w_{ij} = \alpha((1-\beta)\delta_j o_i + \beta \Delta w_{ij}')$
  - $\Delta w_{ij}'$也是上一次的修改量，β在0和1之间取值

设置冲量的BP算法

- 主要数据结构

$W[H,m]$——输出层的权矩阵；

$V[n,H]$——输入（隐藏）层的权矩阵；

$\Delta_o[m]$——输出层各联接权的修改量组成的向量；

$\Delta_h[H]$——隐藏层各联接权的修改量组成的向量；

O_1——隐藏层的输出向量；

O_2——输出层的输出向量；

(X,Y)——一个样本。

设置冲量的BP算法

Algorithm 3 设置冲量的BP算法

用不同的小伪随机数初始化W, V;
初始化精度控制参数ϵ;学习率α;
循环控制参数$\mathrm{E} = \epsilon + 1$;循环最大次数M;循环次数控制参数$\mathrm{N} = 0$;
while $\mathrm{E} > \epsilon$ & $\mathrm{N} < \mathrm{M}$ **do**
    $\mathrm{N} = \mathrm{N} + 1$, $\mathrm{E} = 0$
    while 对每一个样本(X,Y) **do**

对每一个样本(X,Y)，执行的操作

计算$O_1 = F_1(\mathrm{XV})$; $O_2 = F_2(O_1\mathrm{W})$;
计算输出层的权修改量
for $i = 1$ to m **do**
  $\triangle_o[i] = O_2[i] * (1 - O_2[i]) * (\mathrm{Y}[i] - O_2[i])$;
end for
计算输出误差：
for $i = 1$ to m **do**
  $\mathrm{E} = \mathrm{E} + (\mathrm{Y}[i] - O_2[i])^2$;
end for
计算隐藏层的权修改量：
for $i = 1$ to H **do**
  $\mathrm{Z} = 0$;
  for $j = 1$ to m **do**
    $\mathrm{Z} = \mathrm{Z} + \mathrm{W}[i,j] * \triangle_o[j]$;
  end for
  $\triangle_h[i] = \mathrm{Z} * O_1[i](1 - O_1[i])$;
end for
修改输出层权矩阵：
for $k = 1$ to H & $i = 1$ to m **do**
  $\mathrm{W}[k,i] = \mathrm{W}[k,i] + \alpha * O_1[k] * \triangle_o[i]$;
end for
修改隐藏层权矩阵：
for $k = 1$ to n & $i = 1$ to H **do**
  $\mathrm{V}[k,i] = \mathrm{V}[k,i] + \alpha * \mathrm{X}[k] * \triangle_h[i]$;
end for
end while
end while

建议

- 隐藏层的神经元的个数H作为一个输入参数
- 同时将ε、循环最大次数M等，作为算法的输入参数
- 在调试阶段，最外层循环内，加一层控制，以探测网络是否陷入了局部极小点
- 其他
  - 训练数据是否要归一化、饱和分析、多层神经网络导致的梯度不稳定（梯度消失、梯度爆炸）

四、算法的理论基础

- 基本假设
  - 网络含有L层
  - 联接矩阵：$W^{(1)}, W^{(2)}, \ldots, W^{(L)}$
  - 第k层的神经元：H_k个
  - 自变量数：$n * H_1 + H_1 * H_2 + H_2 * H_3 + \cdots + HL * m$
  - 样本集：$\boldsymbol{S} = \{(\boldsymbol{X}_1, \boldsymbol{Y}_1), (\boldsymbol{X}_2, \boldsymbol{Y}_2), \ldots, (\boldsymbol{X}_s, \boldsymbol{Y}_s)\}$
  - 误差测度:

$$E = \sum_{p=1}^{s} E_p$$

误差测度

$$E=\sum_{p=1}^{s}E_p$$

用E代表E_P，用$(\boldsymbol{X},\ \boldsymbol{Y})$代表$(\boldsymbol{X}_p,\boldsymbol{Y}_p)$

$$\boldsymbol{X}=(x_1,x_2,\dots,x_n)$$

$$\boldsymbol{Y}=(y_1,y_2,\dots,y_m)$$

该样本对应的实际输出为

$$\boldsymbol{O}=(o_1,o_2,\dots,o_m)$$

误差测度

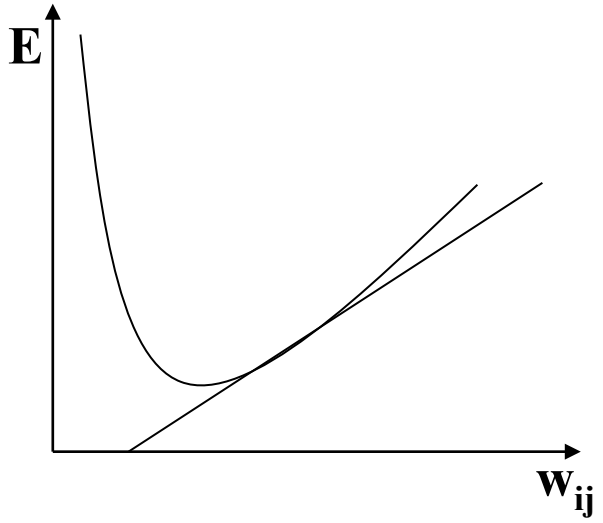
$$E = \sum_{p=1}^{s} E_p$$

用理想输出与实际输出的方差作为相应的误差测度

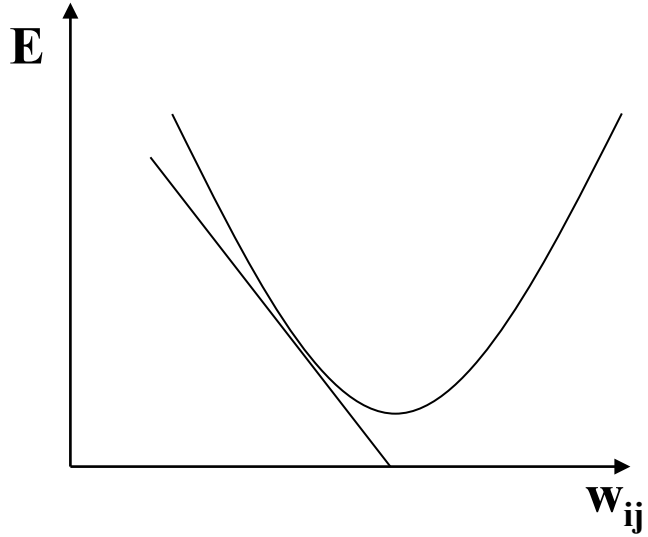
$$E = \frac{1}{2}\sum_{k-1}^{m}(y_k - o_k)^2$$

BP算法采用梯度最速下降法，沿负梯度方向求E的极小点 调整连接矩阵

$$取 \quad \Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}}$$

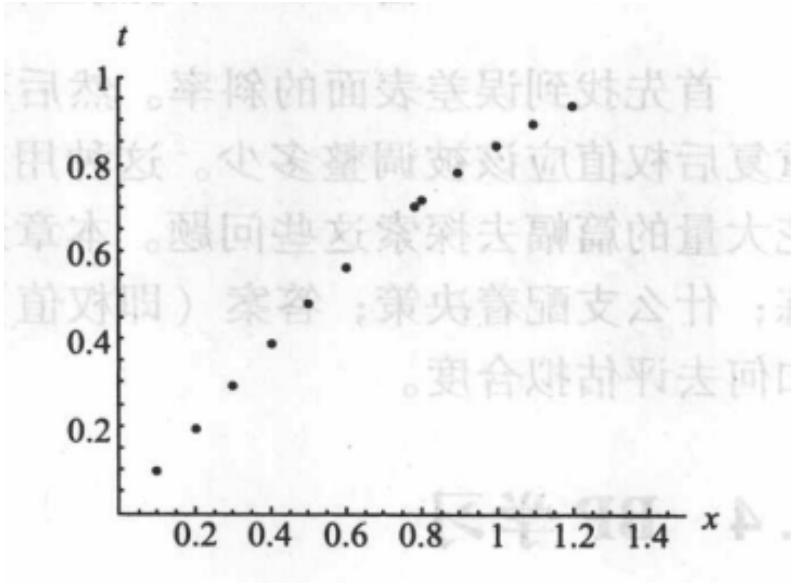


$\frac{\partial E}{\partial w_{ij}} > 0$, 此时$\Delta w_{ij} < 0$

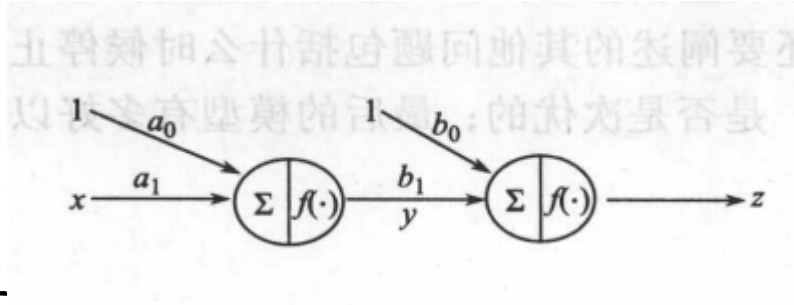


$\frac{\partial E}{\partial w_{ij}} < 0$, 此时$\Delta w_{ij} > 0$

实例：拟合正弦波的第一个1/4周期波形



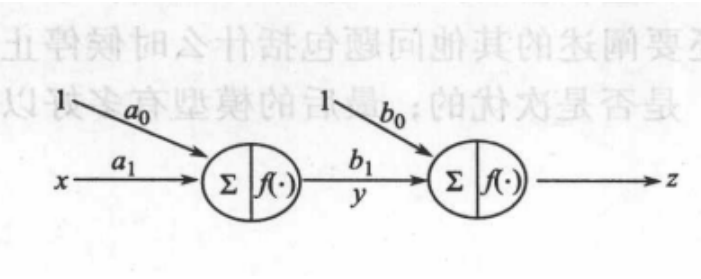
- 设计网络结构如下：



- 随机初始化参数如下：

a_0	a_1	b_0	b_1
0.3	0.2	-0.1	0.4

- 选择两个输出-输出对:



X	t
0.7853	0.707
1.571	1.00

- 对输入-输出对（0.7853,0.707），
- 网络输入的权值和:

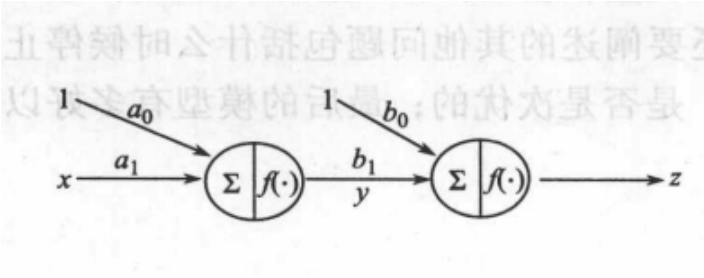
$$u = a_0 + a_1 x = 0.3 + 0.2(0.7853) = 0.457$$

$$y = \frac{1}{1+e^{-u}} = \frac{1}{1+e^{-0.457}} = 0.612$$

$$v = b_0 + b_1 y = -0.1 + 0.4(0.612) = 0.143$$

$$z = \frac{1}{1+e^{-v}} = \frac{1}{1+e^{-0.143}} = 0.536$$

- 网络的输出z和目标t不相等，利用平方和误差：
- $E = \frac{1}{2}(z-t)^2 = \frac{1}{2}(0.536 - 0.707)^2 = 0.0146$

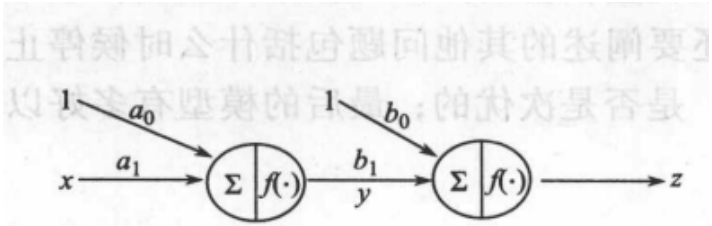


- 由网络结构，输出的计算涉及到之前的神经元权重参数：

- $E = \frac{1}{2}(z-t)^2 = \frac{1}{2}\left\{\frac{1}{1+e^{-(b_0+b_1)\frac{1}{1+e^{-(a_0+a_1x)}}}}\right\}^2$

- 为了考虑误差对各个参数的灵敏度，需要计算偏导数

关于输出神经元权值的误差梯度



- 考虑E对权重b的偏导数（根据链式法则）：

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial b}$$

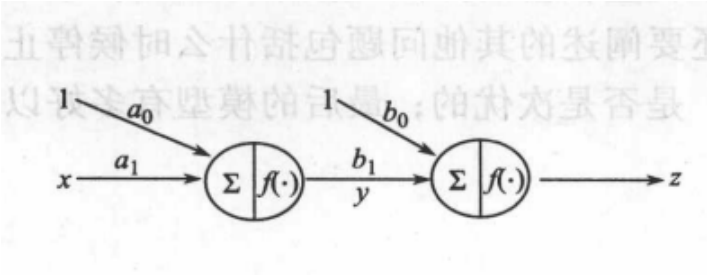
- 三个部分分别是

$\frac{\partial E}{\partial z}$是误差对网络输出的偏导

$\frac{\partial z}{\partial v}$是z相对于输入到输出神经元的加权和v的偏导

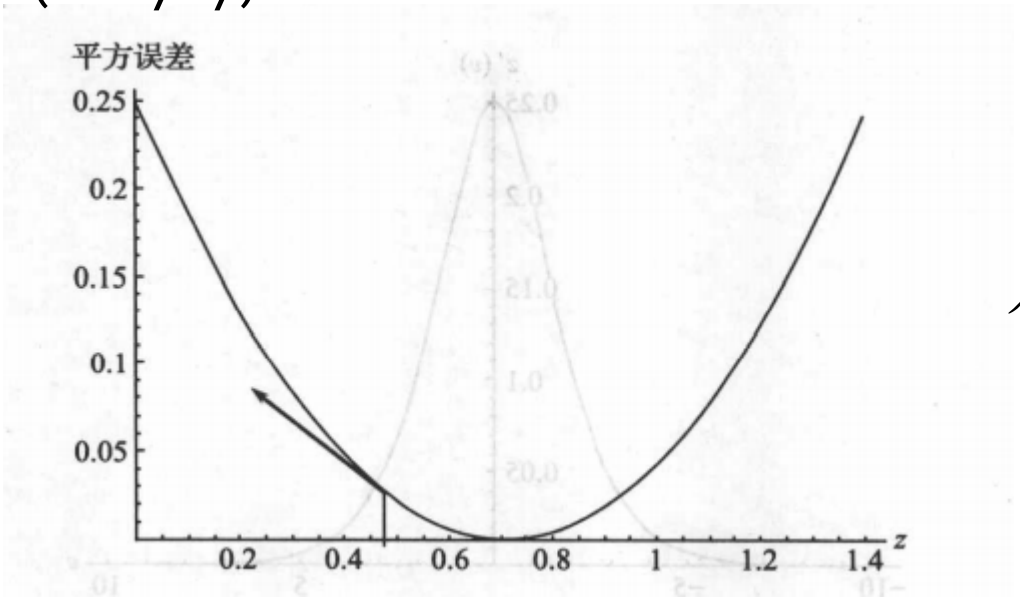
$\frac{\partial v}{\partial b}$是v相对于隐含-输出权值b的偏导

关于输出神经元权值的误差梯度



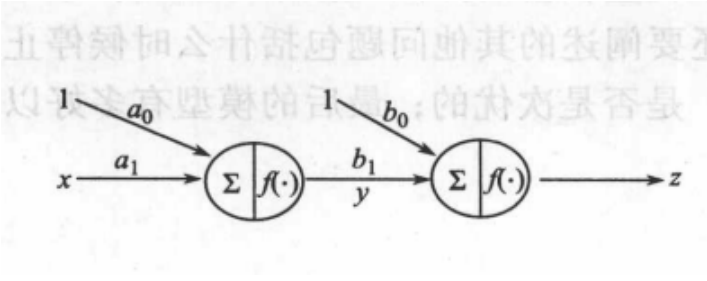
- 第一部分 $\frac{\partial E}{\partial z}$ ：
- $\frac{\partial E}{\partial z} = z - t = 0.536 - 0.707 = -0.171$
- 误差对于网络输出的灵敏度

E=(z-0.707)^2



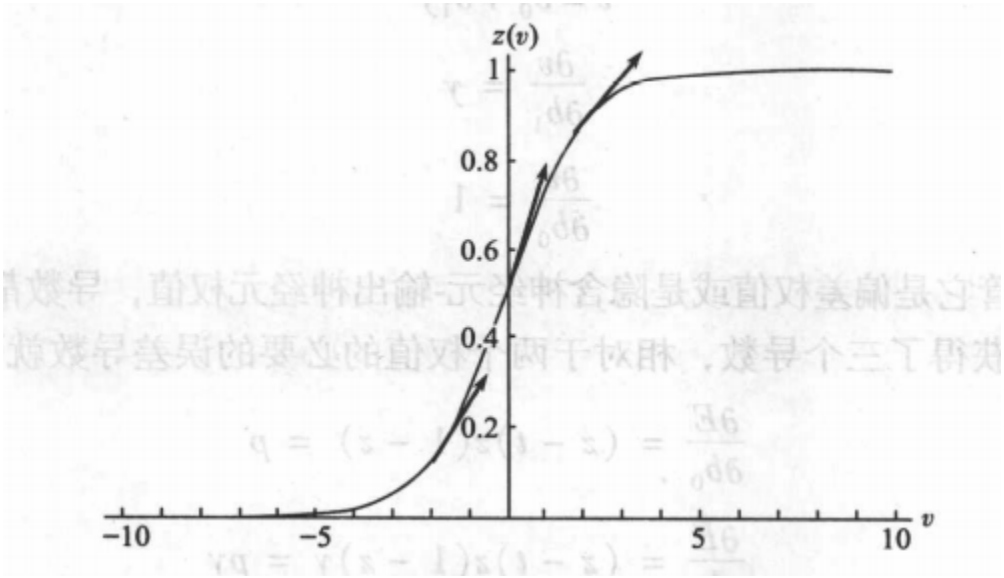
在z=0.536处的误差斜率为-0.171

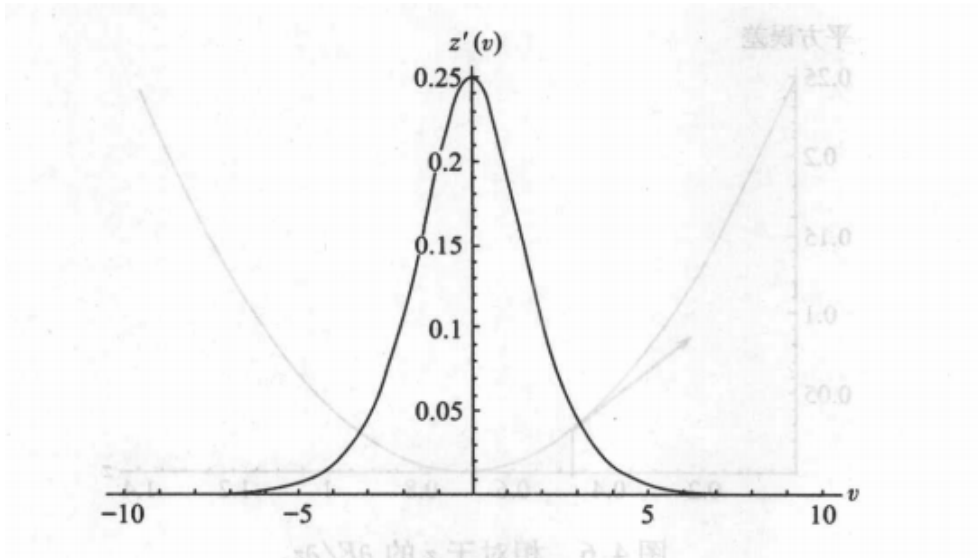
关于输出神经元权值的误差梯度



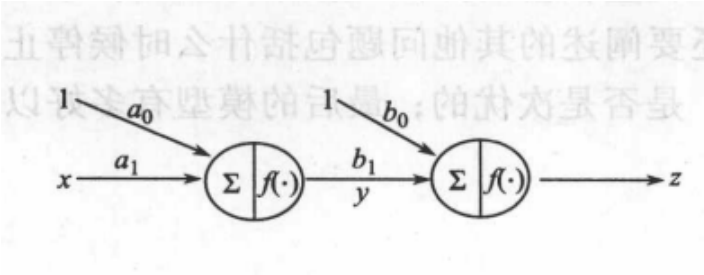
- 第二部分$\frac{\partial z}{\partial v}$:
- $z = \frac{1}{1+e^{-v}}$
- $\frac{\partial z}{\partial v} = \frac{(1-z)/z}{1/z^2} = z(1-z)$
- 是输出神经元激活函数的斜率

在v=o时，斜率最大





关于输出神经元权值的误差梯度



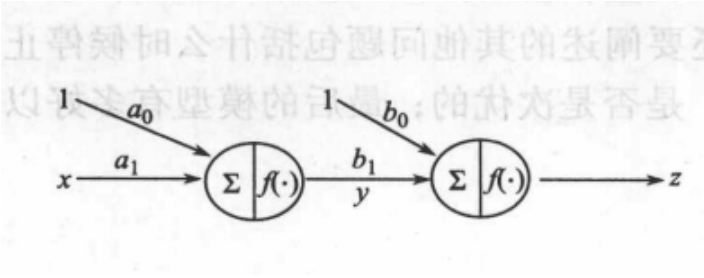
- 第三部分$\frac{\partial v}{\partial b}$：
- 是输入加权和v对输出神经元权值变化的灵敏度
- v= $b_0 + b_1 y$
- $\frac{\partial v}{\partial b_1} = y$
- $\frac{\partial v}{\partial b_0} = 1$

$\frac{\partial E}{\partial b_0}$=(z-t)z(1-z)=p

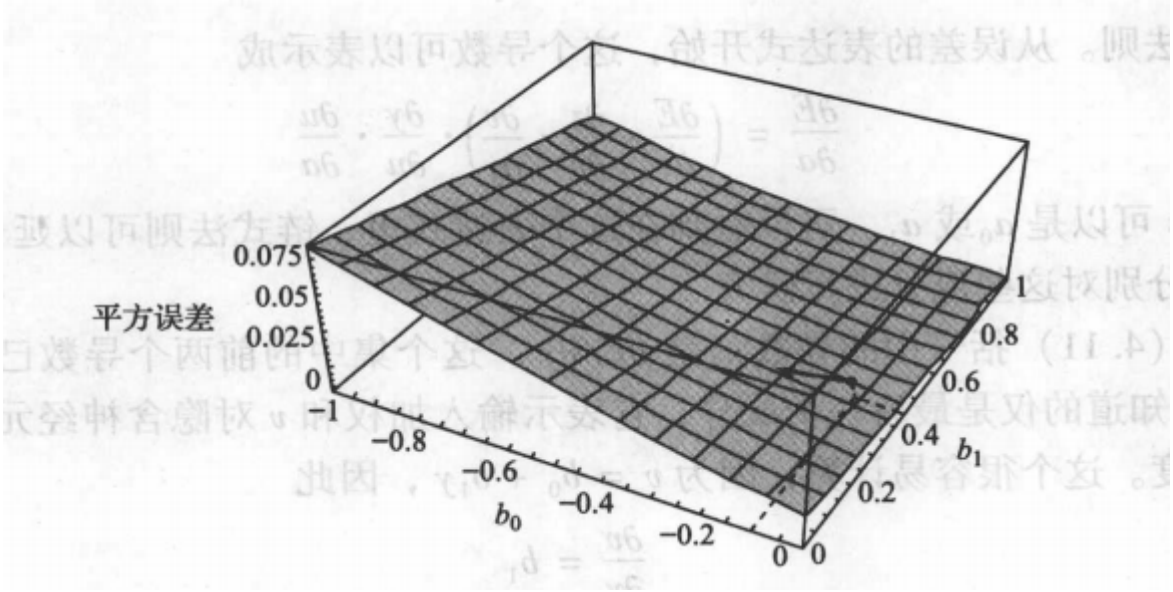
$\frac{\partial E}{\partial b_1}$=(z-t)z(1-z)y=py

- 对于实例(z=0.536,t=0.707,y=0.612)代入
- $\frac{\partial E}{\partial b_0} = (z-t)z(1-z) = p = (0.536 - 0.707)0.536(1 - 0.536) = -0.042$
- $\frac{\partial E}{\partial b_0} = py = (-0.042)(0.6120) = -0.026$

关于输出神经元权值的误差梯度

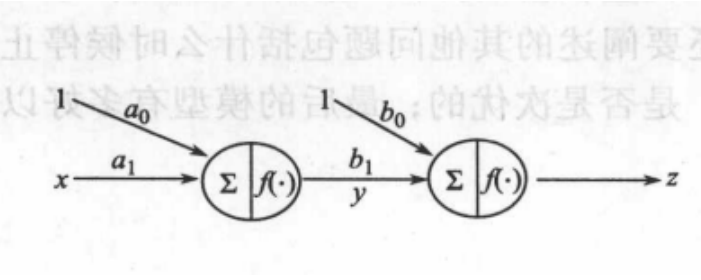


- 关于b_0和b_1的平方误差图解



- 当前权值$b_0 = -0.1\ b_1 = 0.4$
- 可以看出 相对于最优的平方误差，参数b还有差距
- 由此根据参数调整规则，修正参数b

关于隐含神经元权值的误差梯度

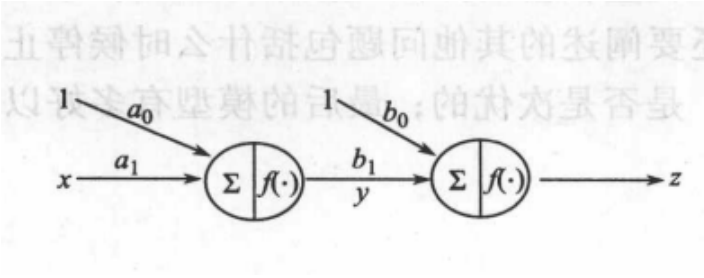


- 这里需要的导数是$\frac{\partial \mathrm{E}}{\partial \mathrm{a_0}}$和$\frac{\partial \mathrm{E}}{\partial \mathrm{a_1}}$，使用链式法则：
- $\frac{\partial E}{\partial a} = \left(\frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial y}\right)\frac{\partial y}{\partial u}\frac{\partial u}{\partial a}$
- 上式括号中的导数集是$\frac{\partial \mathrm{E}}{\partial \mathrm{y}}$。这个集中的前两个导数已经知道是p，需要知道的仅是最后一个成分，表示输入加权和v对隐含神经元权值变化的灵敏度。
- 代入得到：

$$\frac{\partial v}{\partial y} = b_1$$

$$\frac{\partial E}{\partial y} = \frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial y} = \mathrm{p}\, b_1$$

关于隐含神经元权值的误差梯度



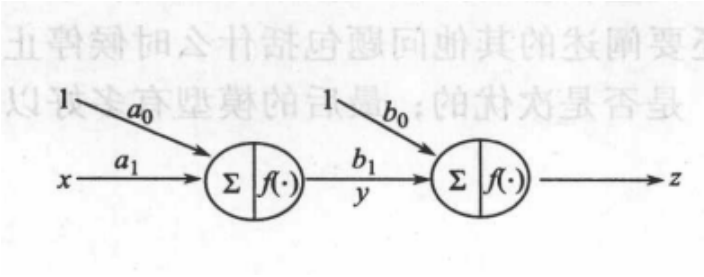
$$\frac{\partial E}{\partial a} = \left(\frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial y}\right)\frac{\partial y}{\partial u}\frac{\partial u}{\partial a}$$

- 上式的第二个成分是$\frac{\partial y}{\partial u}$，是隐含神经元对数函数关于输入到隐含神经元的加权和的导数

$$\frac{\partial y}{\partial u} = y(1-y)$$

- 其中y是隐含神经元的输出。

关于隐含神经元权值的误差梯度



$$\frac{\partial E}{\partial a} = \left(\frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial y}\right)\frac{\partial y}{\partial u}\frac{\partial u}{\partial a}$$

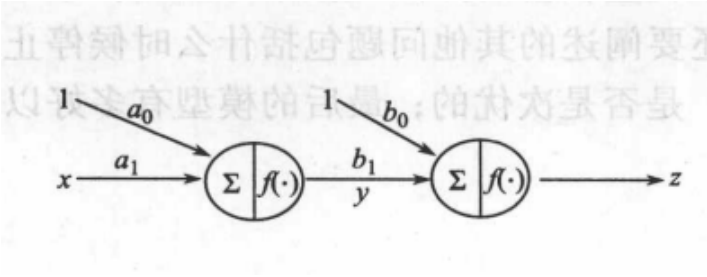
- 上式的最后成分是$\frac{\partial u}{\partial a}$，它依赖于所考虑的权值类型。

$$u = a_0 + a_1 x$$

$$\frac{\partial u}{\partial a_1} = x$$

$$\frac{\partial u}{\partial a_0} = 1$$

关于隐含神经元权值的误差梯度



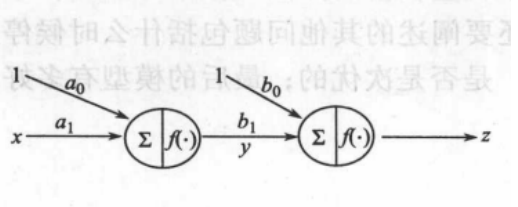
$$\frac{\partial E}{\partial a} = \left(\frac{\partial E}{\partial z}\frac{\partial z}{\partial v}\frac{\partial v}{\partial y}\right)\frac{\partial y}{\partial u}\frac{\partial u}{\partial a}$$

- 三个成分组合后，得到偏差及输出-隐含神经元权值的误差导数

$$\frac{\partial E}{\partial a_1} = p\, b_1 y(1-y)x = qx$$

$$\frac{\partial E}{\partial a_0} = p\, b_1 y(1-y) = q$$

关于隐含神经元权值的误差梯度



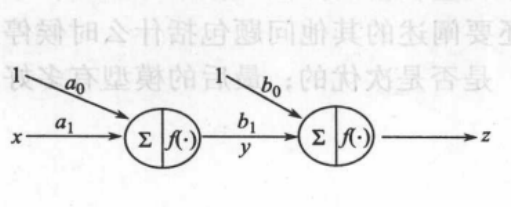
- 在之前的实例问题中，$p = -0.042, b_1 = 0.4, y = 0.612, x = 0.7853$

代入计算得到：

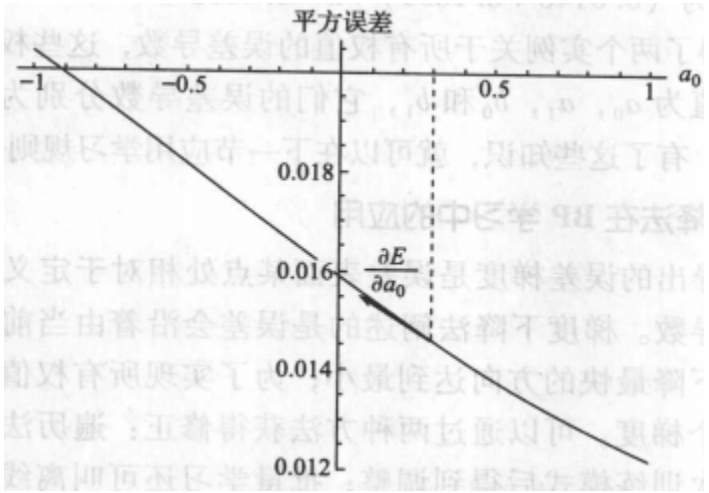
$$\frac{\partial E}{\partial a_0} = p\, b_1 y(1-y) = q = -0.042 * 0.4 * 0.612(1-0.612) = -0.004$$

$$\frac{\partial E}{\partial a_1} = p\, b_1 y(1-y)x = qx = -0.004 * 0.7853 = -0.00314$$

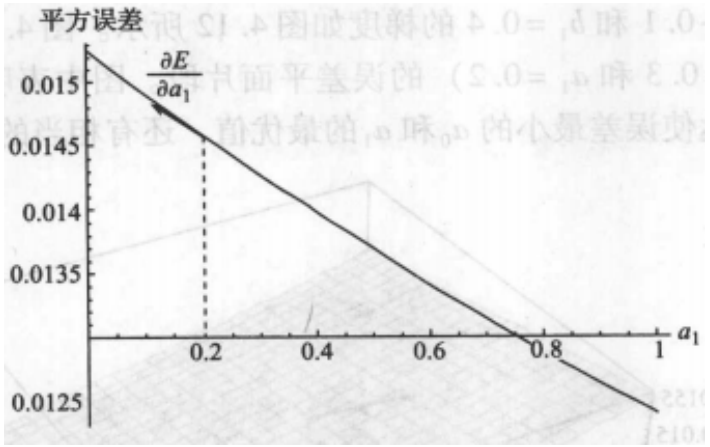
关于隐含神经元权值的误差梯度



- 对于当前权值（$a_0 = 0.3, a_1 = 0.2$）的误差平面片段



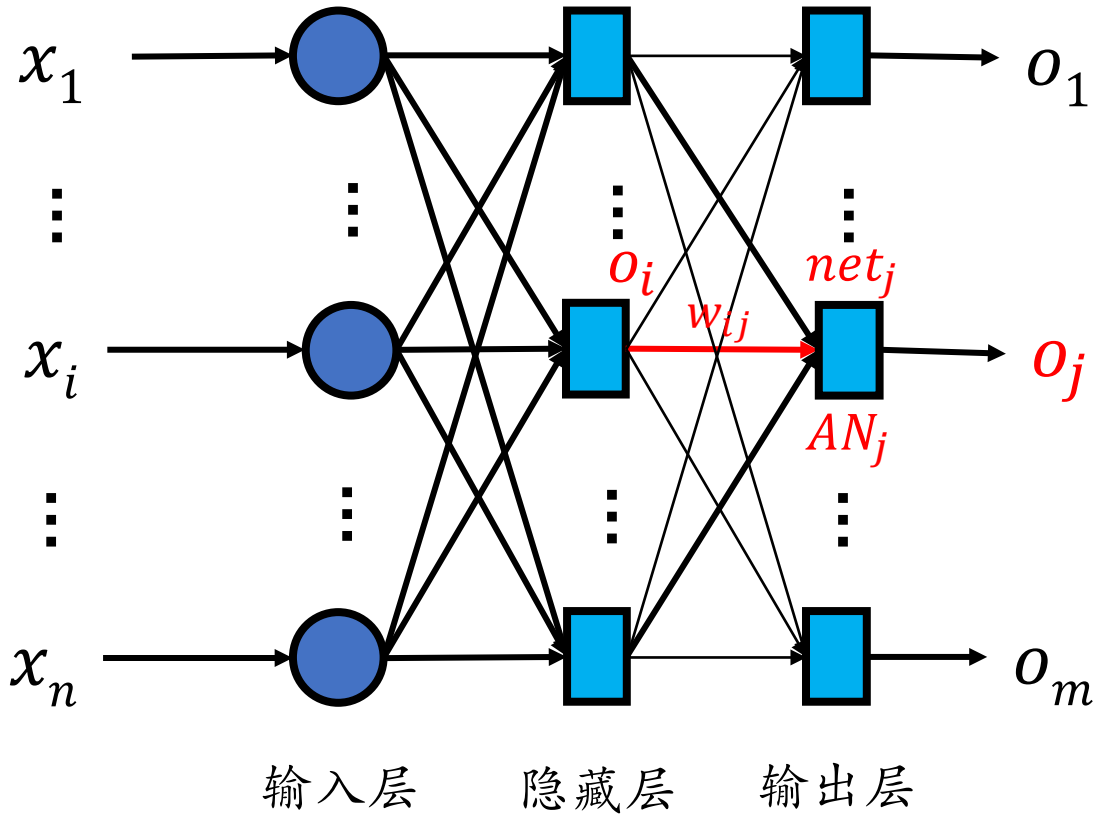
$a_1 = 0.2$时关于权值a_0的平方误差



$a_0 = 0.3$时关于权值a_1的平方误差

图中表明梯度是负的，要穿越误差表面到达使误差最小的a_0和a_1的最优值，还有一段距离

BP算法采用梯度最速下降法，沿负梯度方向求E的极小点 调整连接矩阵



$$-\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

而其中 $$net_j = \sum_i w_{ij} o_i$$

因此，

$$\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial (\sum_i w_{ij} o_i)}{\partial w_{ij}} = o_i$$

BP算法采用梯度最速下降法，沿负梯度方向求E的极小点 调整连接矩阵

$$-\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ij}}$$

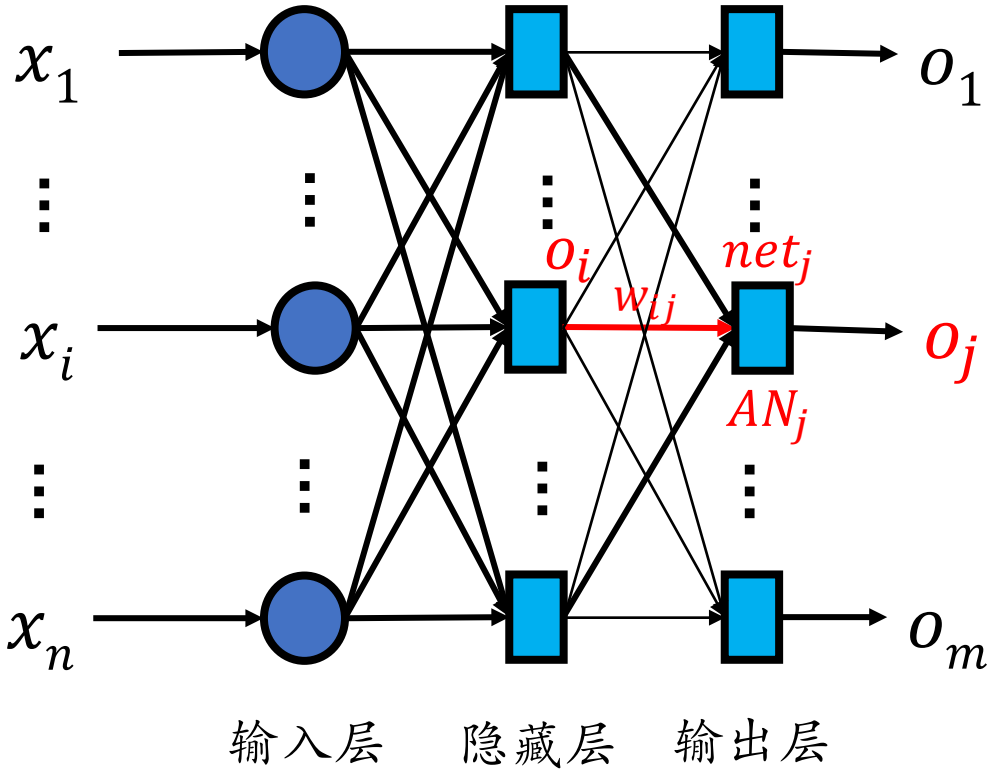
令 $\delta_j = -\frac{\partial E}{\partial net_j}$

$$= -\frac{\partial E}{\partial net_j} \cdot \frac{\partial \left(\sum_k w_{kj} o_k \right)}{\partial w_{ij}}$$

则 $\Delta w_{ij} = \alpha \delta_j o_i$，$\alpha$为学习率

$$= -\frac{\partial E}{\partial net_j} \cdot o_i$$

AN_j为输出层神经元



从而

$$\delta_j = -\frac{\partial E}{\partial net_j}$$

$$= -\frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$= -\frac{\partial E}{\partial o_j} \cdot f'(net_j)$$

AN_j为输出层神经元

$$-\frac{\partial E}{\partial o_j}=-\frac{\partial\left(\frac{1}{2}\sum_{k=1}^{m}\left(y_k-o_k\right)^2\right)}{\partial o_j}$$

$$=-(\frac{1}{2}\frac{\partial\left(y_j-o_j\right)^2}{\partial o_j})$$

$$=-(-\frac{2}{2}(y_j-o_j))$$

$$=(y_j-o_j)$$

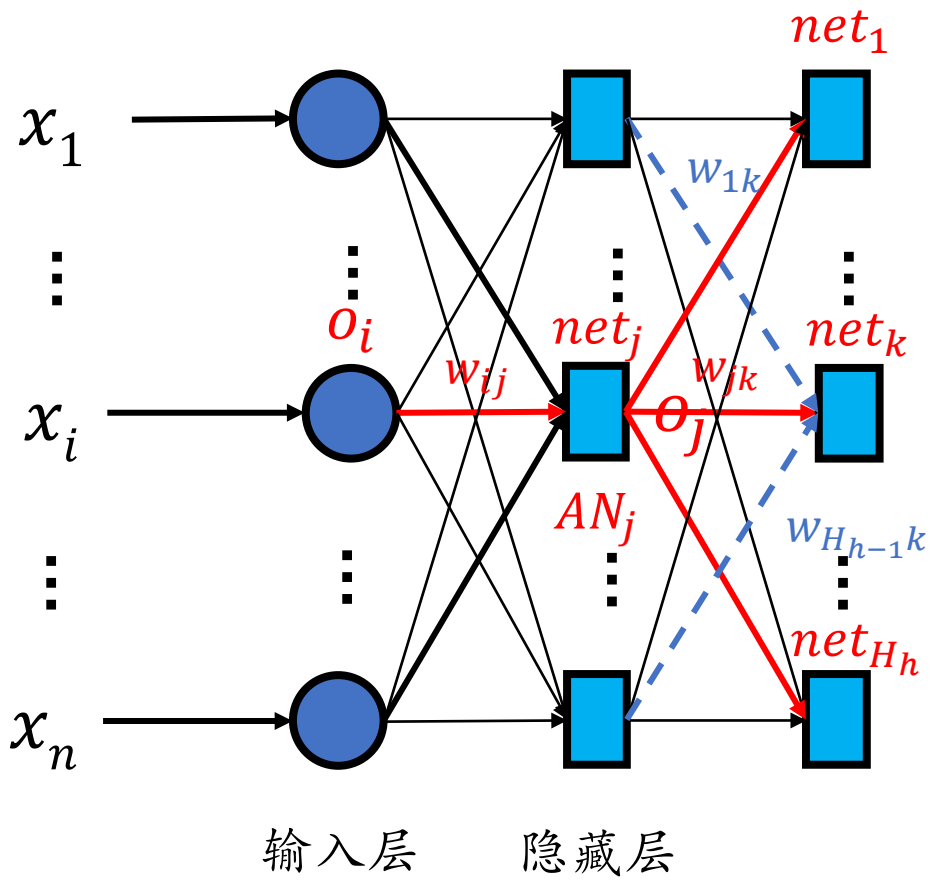
因此 $\delta_j=(y_j-o_j)f'(net_j)$

故，当AN_j为输出层的神经元时，它对应的联接权w_{ij}应该按照下列公式进行调整:

$$w_{ij}=w_{ij}+\alpha\delta_j o_i$$

$$=w_{ij}+\alpha f'(net_j)(y_j-o_j)o_i$$

AN_j为隐藏层神经元



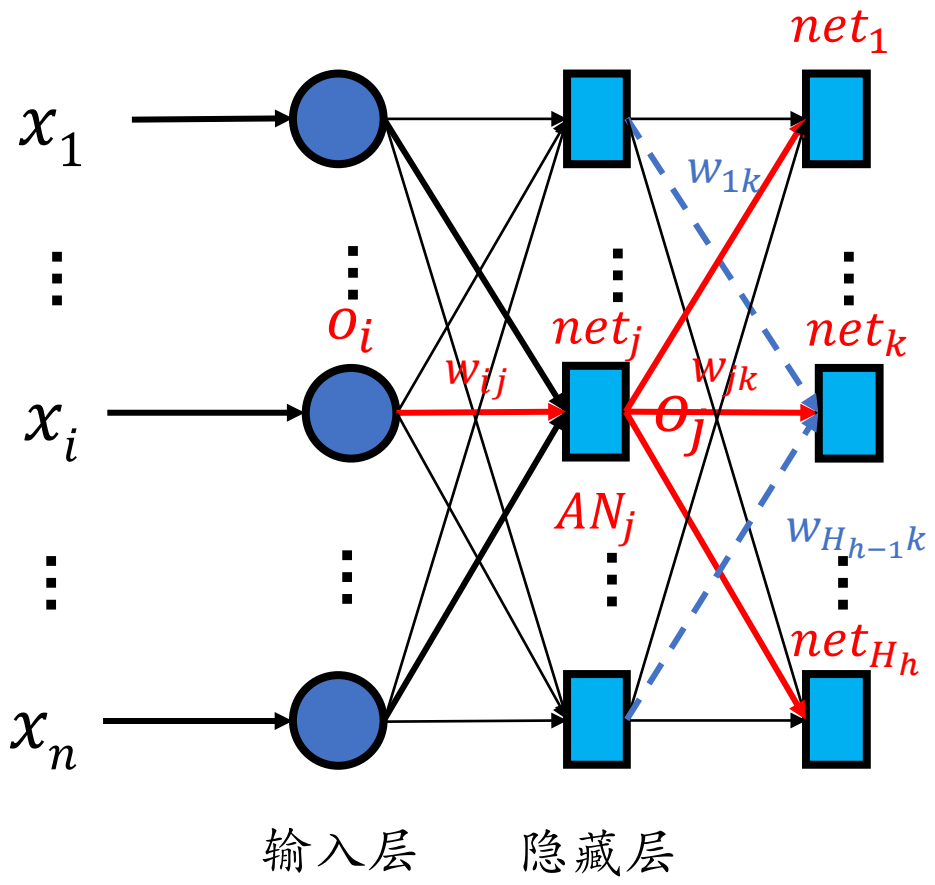
$$\frac{\partial E}{\partial w_{ij}} = -\frac{\partial E}{\partial o_j}\cdot\frac{\partial o_j}{\partial net_j}\cdot\boxed{\frac{\partial net_j}{\partial w_{ij}}}$$

o_i

$$\frac{\partial E}{\partial o_j} = \sum_{k=1}^{H_h}\left(\frac{\partial E}{\partial net_k}\cdot\frac{\partial net_k}{\partial o_j}\right)$$

$$net_k = \sum_{t=1}^{H_{h-1}} w_{tk}o_t \qquad \frac{\partial net_k}{\partial o_j} = \frac{\partial\left(\sum_{t=1}^{H_{h-1}} w_{tk}o_t\right)}{\partial o_j} = w_{jk}$$

AN_j为隐藏层神经元



$$\frac{\partial E}{\partial o_j}=\sum_{k=1}^{H_h}\left(\frac{\partial E}{\partial net_k}\cdot\frac{\partial net_k}{\partial o_j}\right)$$

$$=\sum_{k=1}^{H_h}\left(\frac{\partial E}{\partial net_k}\cdot w_{jk}\right)$$

令 $\delta_k=-\frac{\partial E}{\partial net_k}$

$$\frac{\partial E}{\partial o_j}=-\sum_{k=1}^{H_h}\delta_k w_{jk}$$

AN_j为隐藏层神经元

令 $\delta_j = -\frac{\partial E}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$

$$= -(-\sum_{k=1}^{H_h} \delta_k w_{jk}) \cdot f'(net)$$

$$= \sum_{k=1}^{H_h} \delta_k w_{jk} \cdot f'(net)$$

故，当AN_j为输出层的神经元时，它对应的联接权w_{ij}应该按照下列公式进行调整:

$$w_{ij} = w_{ij} + \alpha \delta_j o_i$$

$$= w_{ij} + \alpha \left(\sum_{k=1}^{H_h} \delta_k w_{jk} \right) \cdot f'(net_j) o_i$$

局部极小点问题

- 局部极小点问题
  - 逃离/避开局部极小点：修改W、V的初值——并不是总有效。
  - 逃离——统计方法；[Wasserman，1986]将Cauchy训练与BP算法结合起来，可以在保证训练速度不被降低的情况下，找到全局极小点。

局部极小点问题

- 局部极小：邻域点误差函数值均不小于该点的函数值。
- 全局最小：参数空间内所有点的误差函数值均不小于该点的误差函数值。
- 参数空间内梯度为0的点对应局部极小点，可能存在多个局部极小点，但只会有一个全局最小点。也就说全局最小点一定是局部极小点，反之则不成立。
- 在梯度下降法中，当梯度等于0的时候，参数就不会发生改变，这个点对应的是局部极小点，因此利用梯度下降法只能找到一个局部极小点，如果损失函数是凸函数，那么只有一个局部极小点，局部极小点就是全局最小，如果损失函数不是凸函数，那么我们找到的就是局部极小点而不是全局最优，由于初始值不同，利用梯度下降法，可以找到不同的局部极小点，我们可以比较这些局部最优，从而找出全局最优。

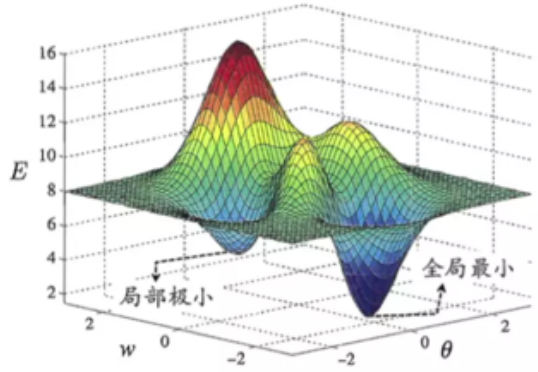


图 5.10 全局最小与局部极小

跳出局部极小的策略：

- 不同的初始参数
- 模拟退火
- 随机扰动
- 遗传算法

几个问题的讨论

- 收敛速度问题
- 网络瘫痪问题
  - 在训练中，权可能变得很大，这会使神经元的网络输入变得很大，从而又使得其激活函数的导函数在此点上的取值很小。根据相应式子，此时的训练步长会变得非常小，进而将导致训练速度降得非常低，最终导致网络停止收敛
- 稳定性问题
  - 用修改量的综合实施权的修改
  - 连续变化的环境，它将变成无效的

几个问题的讨论

- 步长问题
  - BP网络的收敛是基于无穷小的权修改量
  - 步长太小，收敛就非常慢
  - 步长太大，可能会导致网络的瘫痪和不稳定
  - 自适应步长，使得权修改量能随着网络的训练而不断变化。[1988年，Wasserman]
- 学习率
  - 自适应地对每一个权值进行独立调整

自适应学习率方法

- AdaGrad方法：在模型训练初期的，希望参数变化更多更快，而在模型训练后期，希望参数变化的更慢且值更小。

$$\Delta w_t = -\frac{\eta}{\sqrt{\sum_{k=1}^{t} g_k^2 + \varepsilon}} g_t$$

- Adadelta方法：解决AdaGrad后期梯度很小的弊端

$$\Delta w_t = -\frac{\eta}{\sqrt{\lambda * \sum_{k=1}^{t-1} g_K^2 + (1-\lambda) g_t^2 + \varepsilon}} * g_t \qquad E|g^2|_t = \lambda * E|g^2|_{t-1} + (1-\lambda) * g_t^2$$

- 不依赖全局的学习率，和近似牛顿法迭代

$$\Delta w_t = -\frac{\sqrt{\sum_{\varphi=1}^{t-1} \Delta w_\varphi}}{\sqrt{E|g^2|_t + \varepsilon}}$$

五、应用

BP神经网络的应用

- 模式识别
- 数据压缩
- 特征检测
  - 通过非线性变换将输入数据变换到一种称之为隐藏空间或特征空间的新空间
- 泛化
  - 学习过程可以看成一个曲线拟合的过程，泛化可以作为关于输入数据非线性插值的结果
- 函数逼近
  - 提供连续映射的近似实现：通用逼近定理表明，单个隐藏层足够，但并不代表其在学习时间、实现的难易程度或者泛化意义上是最优的。

- 单一非线性神经元
  - 性能和统计学中的非线性回归类似
  - 单输入、多输入
- 单输入—单输出的多层网络
  - 能任意逼近复杂的单输入单输出函数
  - 隐含神经元要接受单一输入
  - 输出神经元从多个隐含神经元接受输入
  - 神经元通过局部处理数据而全局互相影响产生输出
- 多输入——多输出的多层网络
  - 分类：产生合适的分类边界
  - 每个隐藏神经元会将样本分成不同的聚类
  - 输出神经元综合隐藏神经元的分类线形成最终非线性分类边界

六、作业

- 设计双月数据，使其线性不可分，用单层感知器和BP神经网络分别进行分类，比较其结果。尝试分析BP网络隐藏神经元在非线性分类中所起的作用。
- 从网上下载optdigits数据库，这是一个10类手写数字库。建立一个BP网络用于这个10类手写数字的分类，使用一层隐藏层。用training集对该网络进行训练，用test集测试。
  - 比较不同的隐藏层神经元个数对结果有什么影响。
  - 思考如何进一步提高识别率。
- 训练一个1-5-1的神经网络来逼近函数sin(x)：(1)由sin函数产生100个输入-输出对，训练该神经网络使其能根据x预测sin(x)，报告其精度（误差）；(2)报告输入-隐藏层权值及隐藏-输出层权值；（3）报告作为x的函数的隐藏神经元输出y，报告输出神经元的输出z，找到每一个隐藏神经元输出函数的分界点，讨论参与产生输出的隐藏神经元。

第五讲结束