



信阳师范学院  
数学与统计学院  
SCHOOL OF MATHEMATICS AND STATISTICS

# 第14章 神经网络与深度学习



讲授人：牛言涛



日期：2020年5月17日

# 目录

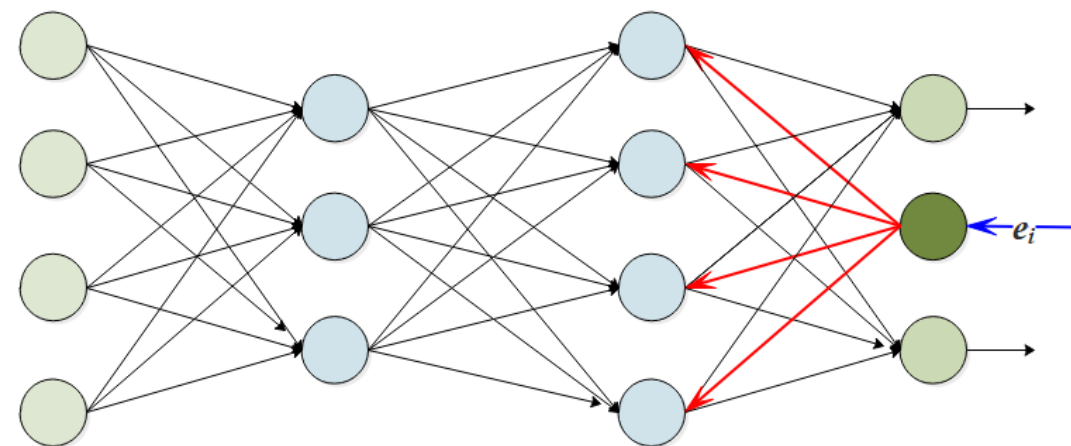
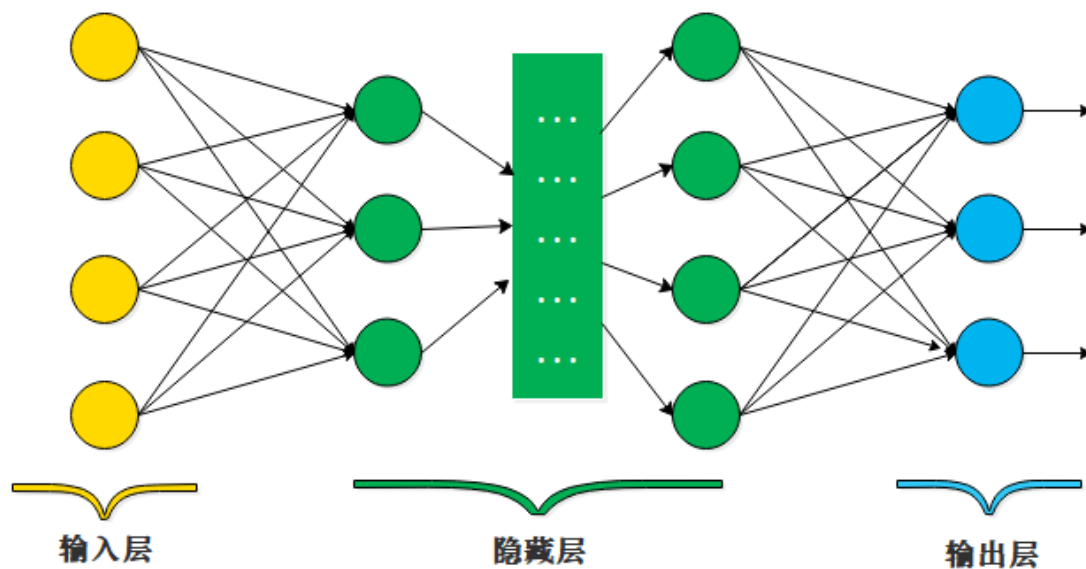
## CONTENTS

- A 机器学习简介
- B 单层神经网络
- C 多层神经网络
- D 神经网络及其分类
- E 深度学习
- F 卷积神经网络



# 1. 多层神经网络

- 将误差反向传播的多层神经网络模型

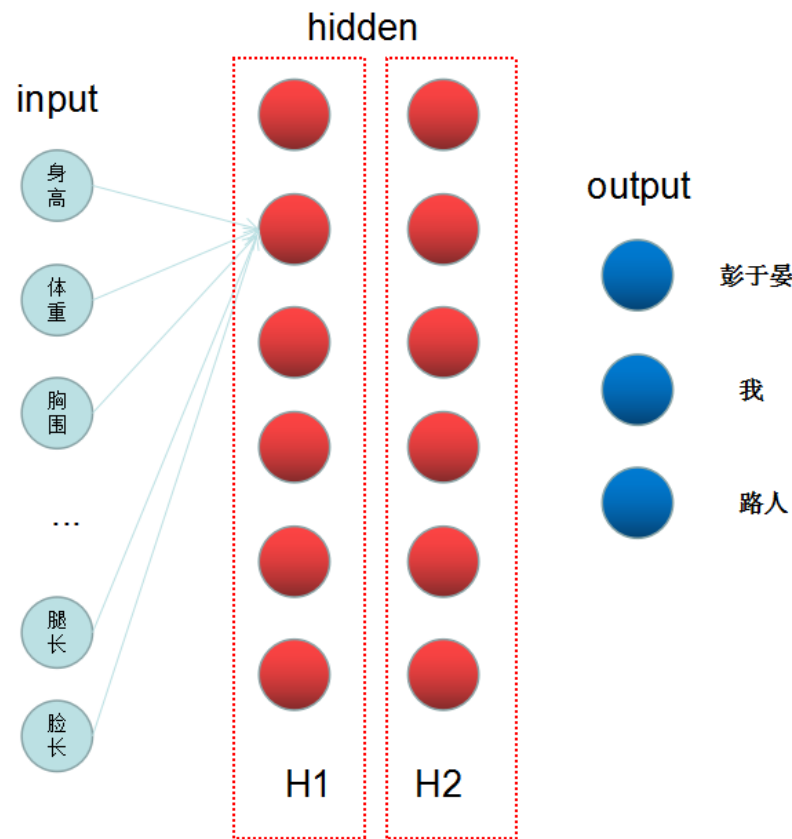


神经网络的输入数据通过输入层、隐藏层和输出层正向移动。相反，在反向传播算法中，输出误差从输出层开始反向移动，直至到达输入层右侧的那个隐藏层。

# 1. 多层神经网络



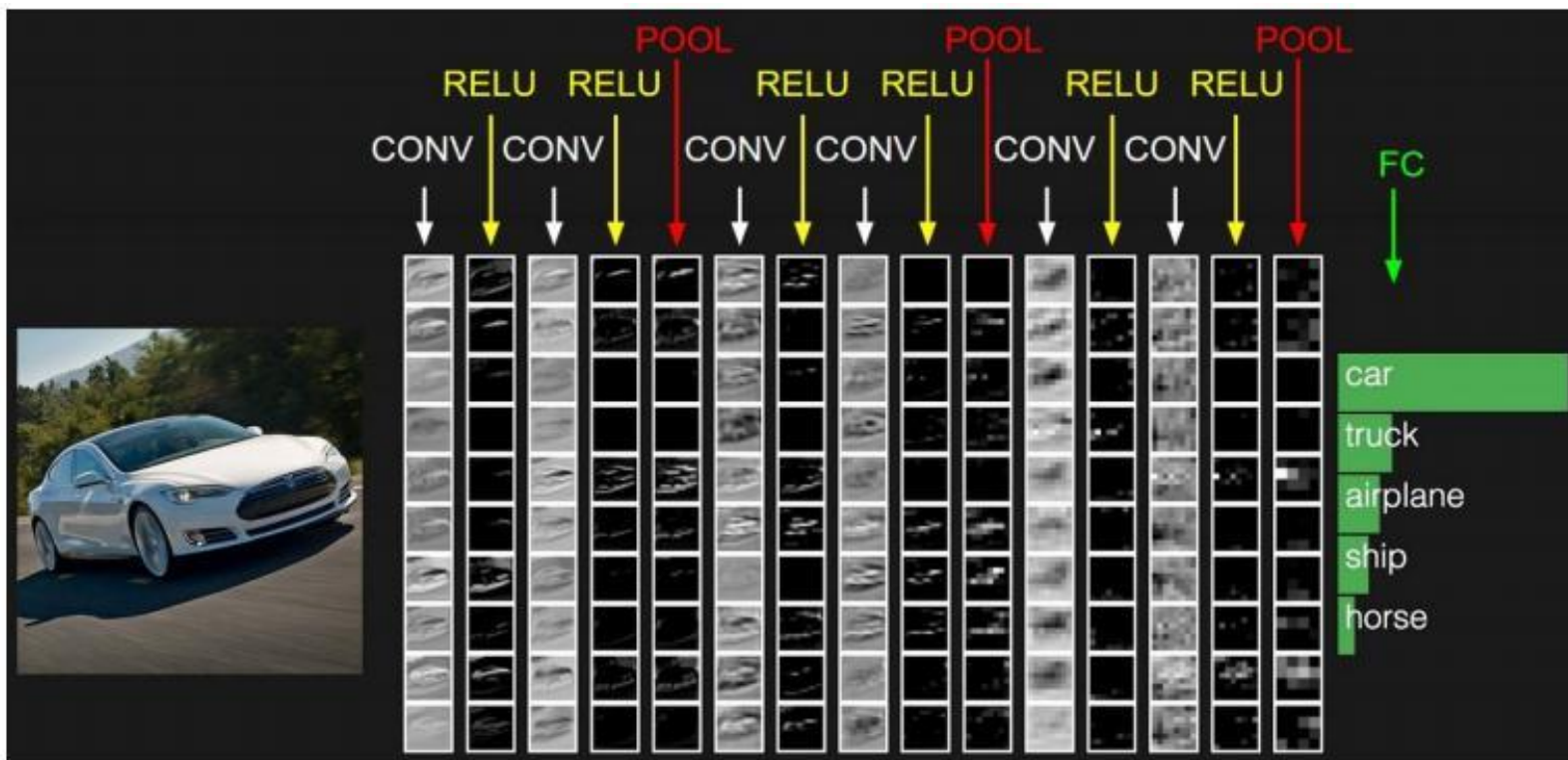
- 隐藏层的意义就是把输入数据的特征，抽象到另一个维度空间，来展现其更抽象化的特征，这些特征能更好的进行线性划分。
- 多个隐藏层其实是对输入特征多层次的抽象，最终的目的就是为了更好的线性划分不同类型的数据（隐藏层的作用）。
- 输入特征是：身高、体重、胸围、腿长、脸长等等一些外貌特征，输出是三个类：帅气如彭于晏，帅气如我，路人。那么隐藏层H1中，身高体重腿长这些特征，在H1中表达的特征就是身材匀称程度，胸围，腰围，臀围这些可能表达的特征是身材如何，脸长和其他的一些长表达的特征就是五官协调程度。
- 那么把这些特征，再输入到H2中，H2中的神经元可能就是在划分帅气程度，身材好坏了，然后根据这些结果，分出了三个类。



# 1. 多层神经网络

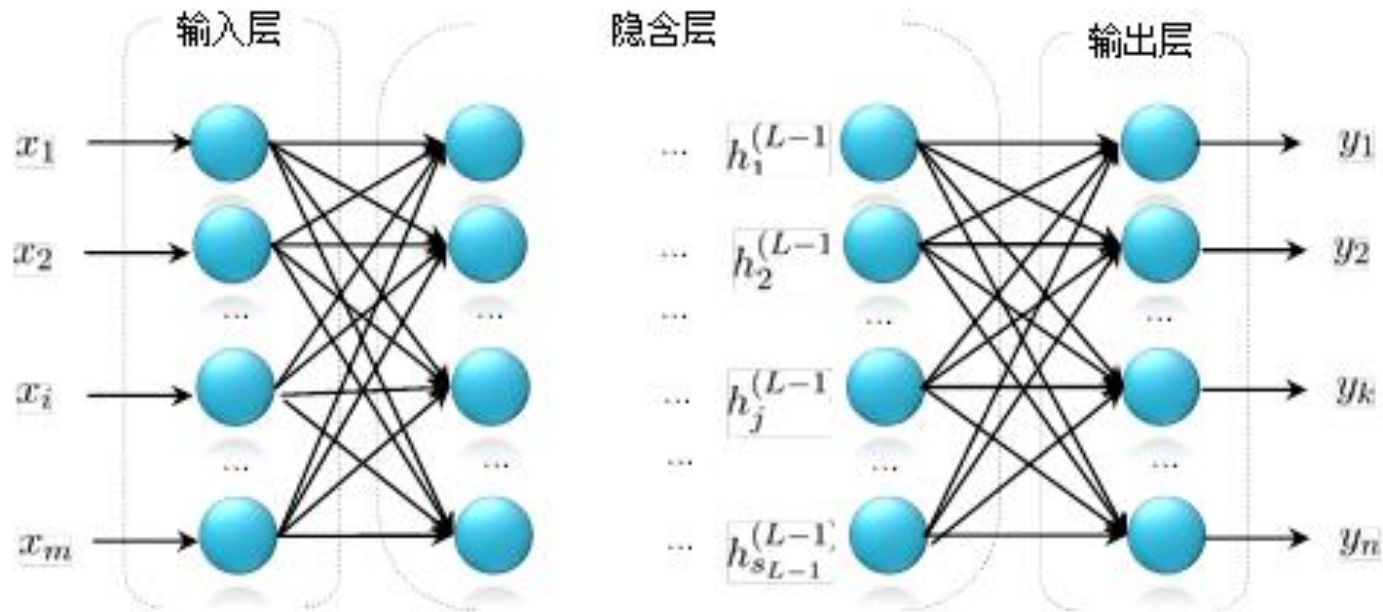


## 卷积神经网络



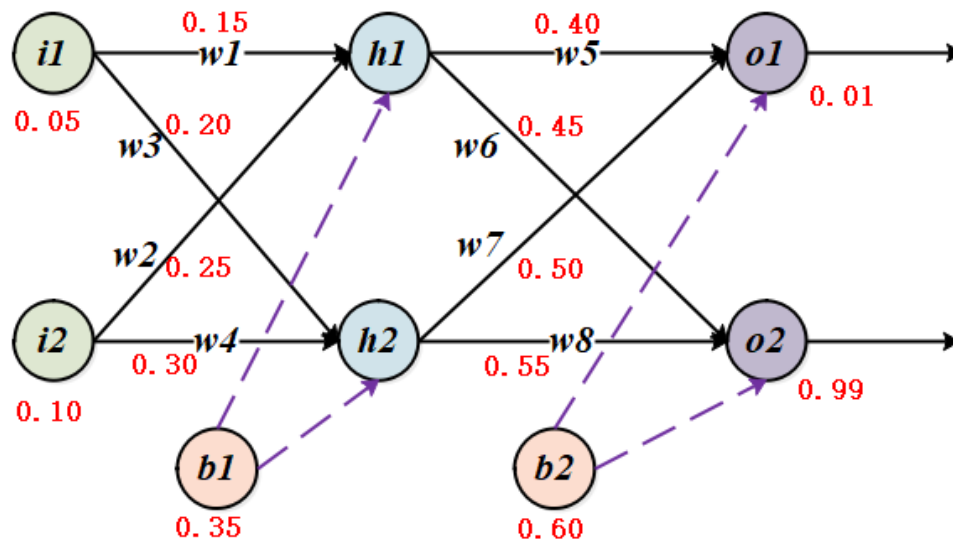
- 是不是隐藏层越多就越好呢，可以特征划分的更清楚啊？理论上是这样的，但实际这样会带来两个问题
  - 层数越多参数会爆炸式增多；
  - 到了一定层数，再往深了加隐藏层，分类效果的增强会越来越不明显。上面那个例子，两层足以划分人是有多帅，再加几层是要得到什么特征呢？这些特征对划分没有什么提升了。

# 1. 多层神经网络



- BP算法是一种最有效的多层神经网络学习方法，其主要特点是信号前向传递，而误差后向传播，通过不断调节网络权重值，使得网络的最终输出与期望输出尽可能接近，以达到训练的目的。
- BP (back propagation) 神经网络是1986年由Rumelhart和McClelland为首的科学家提出的概念，是一种按照误差逆向传播算法训练的多层前馈神经网络，是应用最广泛的神经网络。

## 2. 反向传播算法



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 $w_i$ 是层与层之间连接的权重，激活函数我们默认为sigmoid函数。

另外，三层网络给出了输出值、权值、偏差值和输出值。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：

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

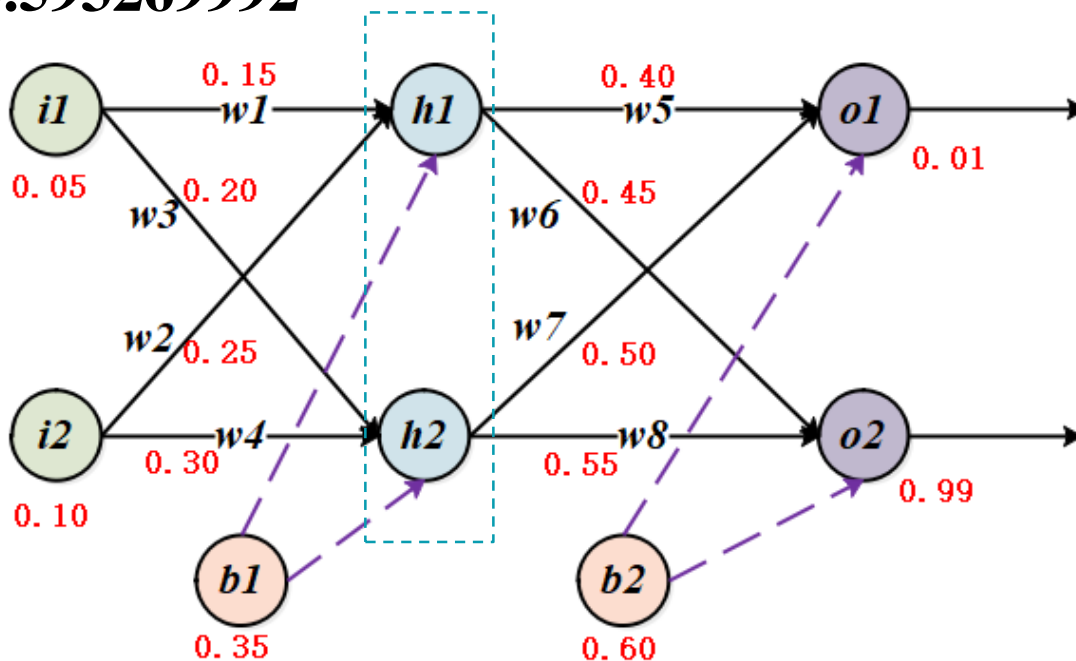
$$net_{h1} = 0.15 \times 0.05 + 0.2 \times 0.1 + 0.35 \times 1 = 0.3775$$

神经元h1的输出o1(此处用到sigmoid激活函数)：

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

同理，可计算出神经元h2的输出o2：

$$out_{h2} = 0.596884378$$





# Step 1 前向传播

2.隐含层---->输出层: 计算输出层神经元o1和o2的值:

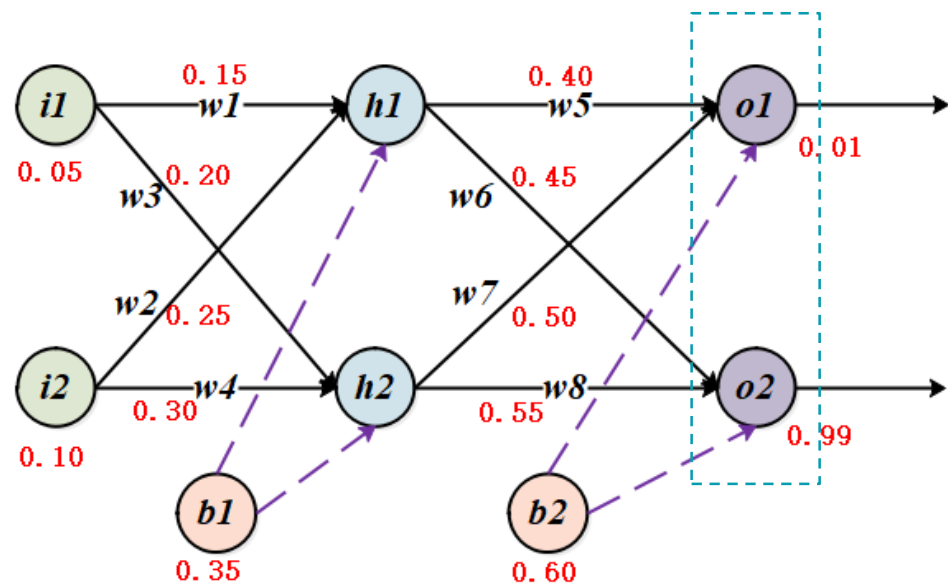
$$net_{o1} = w_5 \cdot out_{h1} + w_6 \cdot out_{h2} + b_2 \cdot 1$$

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

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

同理，可计算出神经元o2的输出值:

$$out_{o2} = 0.772928465$$



这样前向传播的过程就结束了，得到输出值为[0.75136507, 0.772928465]，与实际值[0.01, 0.99]相差还很远，现在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

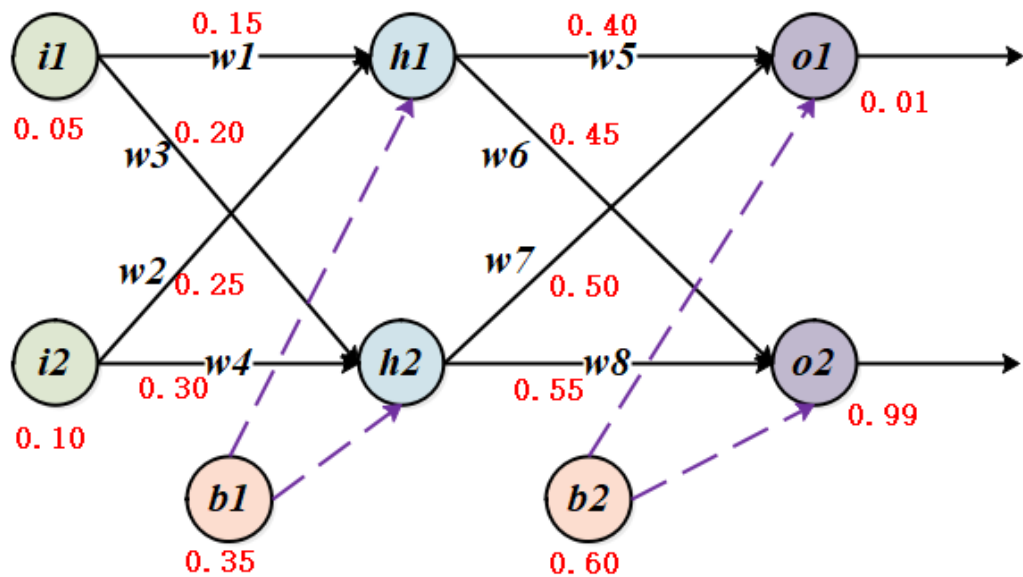
### 1. 计算总误差 (square error):

有两个输出，分别计算o1和o2的误差，总误差为两者之和： $E_{total} = \sum \frac{1}{2} (target - output)^2$

$$E_{o1} = \frac{1}{2} (0.01 - 0.75136507)^2 = 0.274911083$$

$$E_{o2} = 0.023560026$$

$$E_{total} = E_{o1} + E_{o2} = 0.274911083 + 0.023560026 = 0.298371109$$



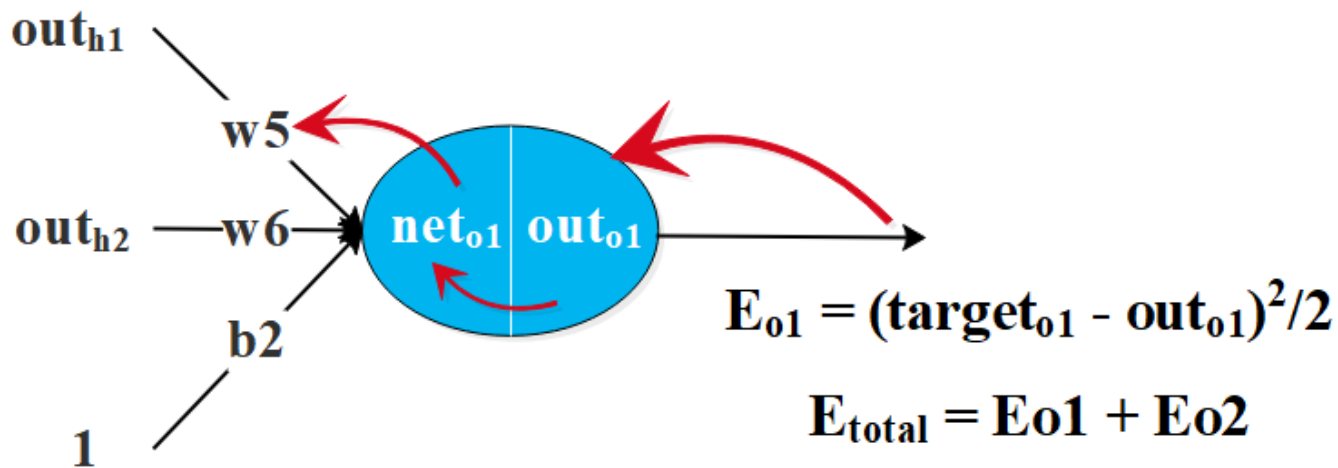
## Step 2 反向传播

### 2. 隐含层---->输出层的权值更新:

以权重参数 $w_5$ 为例, 如果想知道 $w_5$ 对整体误差产生了多少影响, 可以用整体误差对 $w_5$ 求偏导导出: (链式法则)

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

$$\begin{cases} E_{total} = \sum (target - out)^2 / 2 \\ out_{o1} = \frac{1}{1 + e^{-net_{o1}}} \\ net_{o1} = w_5 \cdot out_{h1} + w_6 \cdot out_{h1} + b_2 \cdot 1 \end{cases}$$



# 隐含层-->输出层的权值更新

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

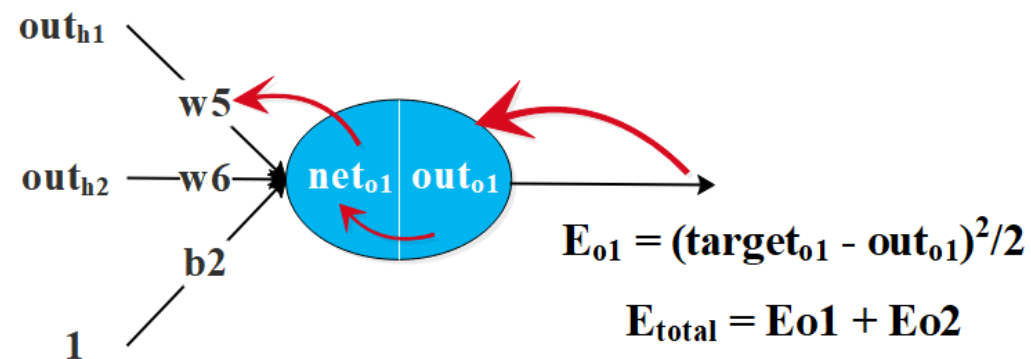
$$\frac{\partial E_{total}}{\partial out_{o1}} = -(target_{o1} - out_{o1}) = -(0.01 - 0.75136507) = 0.74136507$$

$$(2) out_{o1} = \frac{1}{1 + e^{-net_{o1}}}$$

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

$$(3) net_{o1} = w_5 \cdot out_{h1} + w_6 \cdot out_{h2} + b_2 \cdot 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = out_{h1} = 0.593269992$$



这样就计算出整体误差E(total)对w5的偏导值。

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{o1}} \cdot \frac{\partial net_{o1}}{\partial w_5} = 0.74136507 \times 0.186815602 \times 0.593269992 = 0.082167041$$

## 隐含层-->输出层的权值更新

总结公式, 发现:  $\frac{\partial E_{total}}{\partial w_5} = -(target_{o1} - out_{o1}) \cdot out_{o1} \cdot (1 - out_{o1}) \cdot out_{h1}$

为了表达方便, 用 $\delta_1$ 表示误差

$$\delta_{o1} = \frac{\partial E_{total}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{o1}} = \frac{\partial E_{total}}{\partial net_{o1}} = \boxed{-(target_{o1} - out_{o1}) \cdot out_{o1} \cdot (1 - out_{o1})}$$

因此, 整体误差 $E(total)$ 对 $w_5$ 的偏导公式可以写成:

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

如果输出层误差计为负的话, 也可以写成:  $\frac{\partial E_{total}}{\partial w_5} = -\delta_{o1} \cdot out_{h1}$

# 隐含层-->输出层的权值更新

最后，更新w5的值：

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

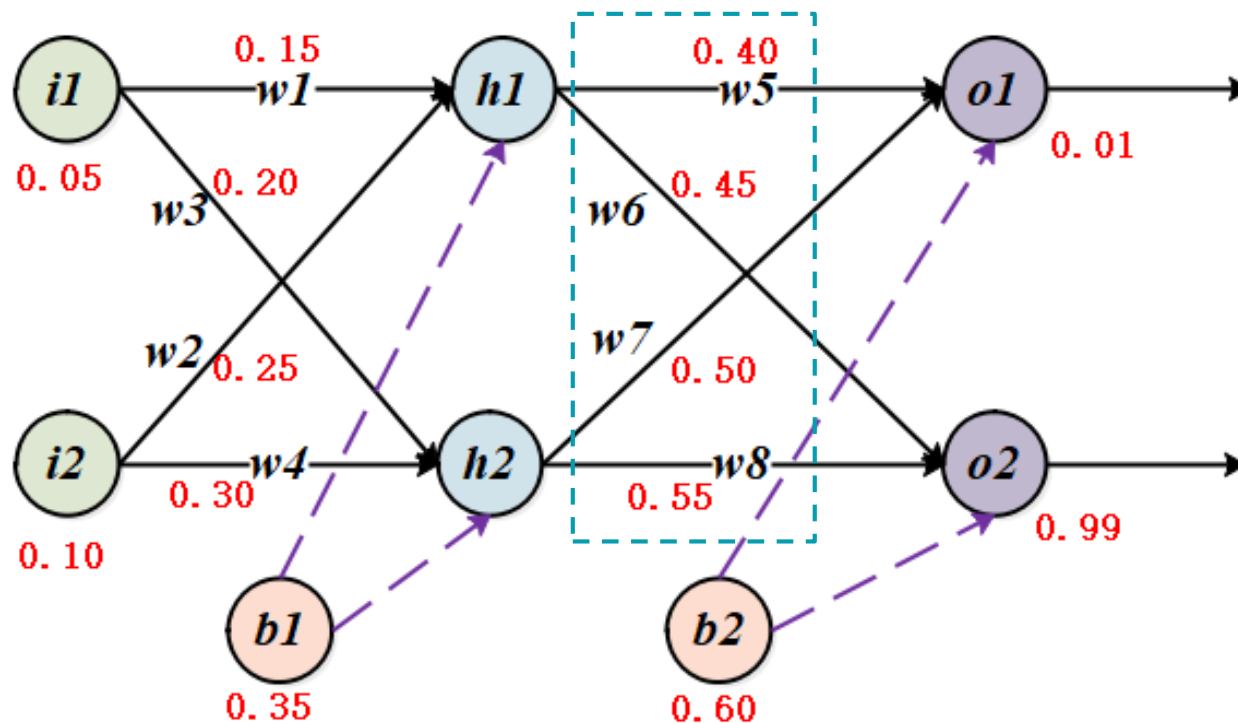
其中 $\eta$ 是学习速率，这里取0.5

同理，可更新w6,w7,w8:

$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

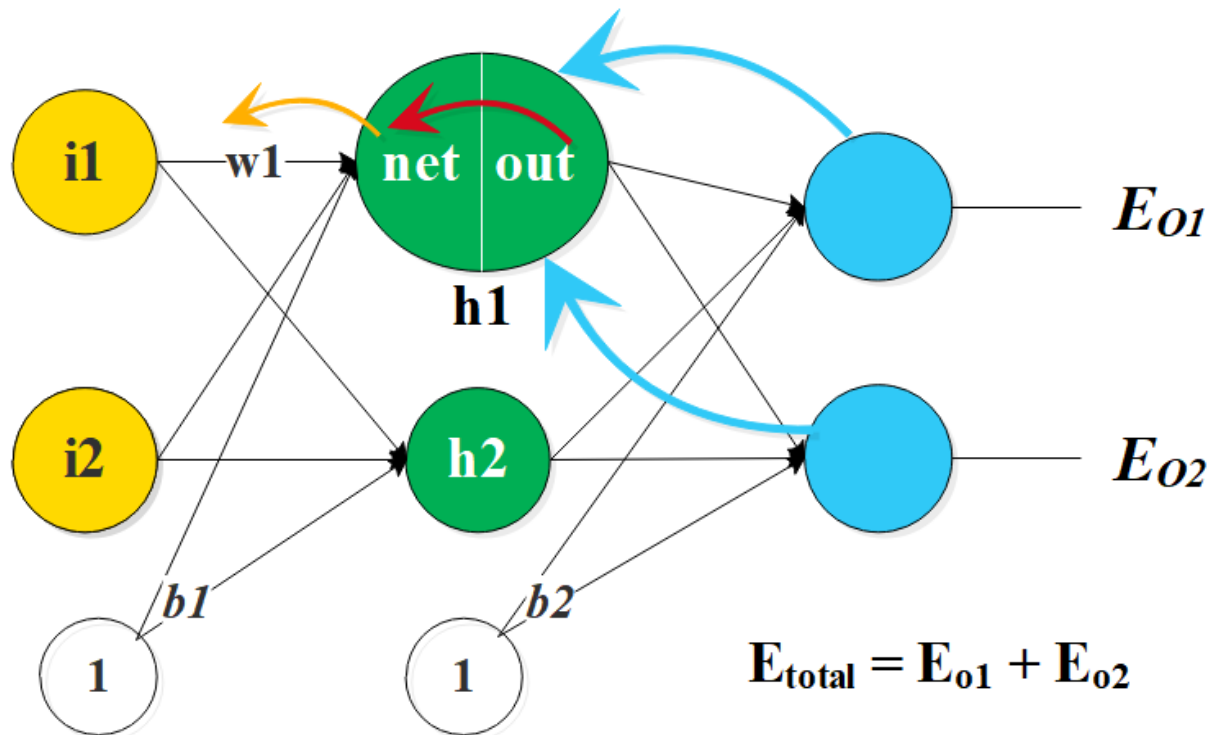


## Step 2 反向传播

### 3. 输入层---->隐含层的权值更新:

方法类似，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。

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} \cdot \frac{\partial out_{h1}}{\partial net_{h1}} \cdot \frac{\partial net_{h1}}{\partial w_1}$$
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$





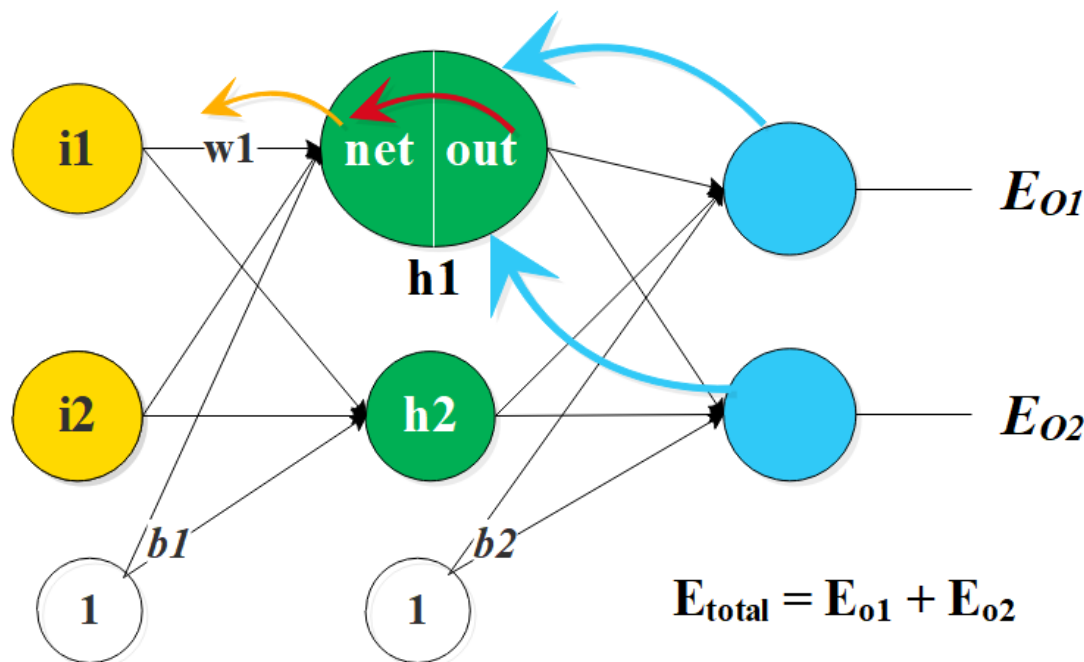
# 输入层-->隐含层的权值更新

$$\frac{\partial E_{o1}}{\partial net_{h1}} = \frac{\partial E_{o1}}{\partial out_{o1}} \cdot \frac{\partial out_{o1}}{\partial net_{h1}} = 0.74136507 \times 0.186815602 = 0.138498562$$

$$net_{o1} = w_5 \cdot out_{h1} + w_6 \cdot out_{h2} + b_2 \rightarrow \frac{\partial net_{o1}}{\partial out_{h1}} = w_5 = 0.40$$

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

同理，计算出  $\frac{\partial E_{o2}}{\partial out_{h1}} = -0.019049119$



# 输入层-->隐含层的权值更新

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

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

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

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

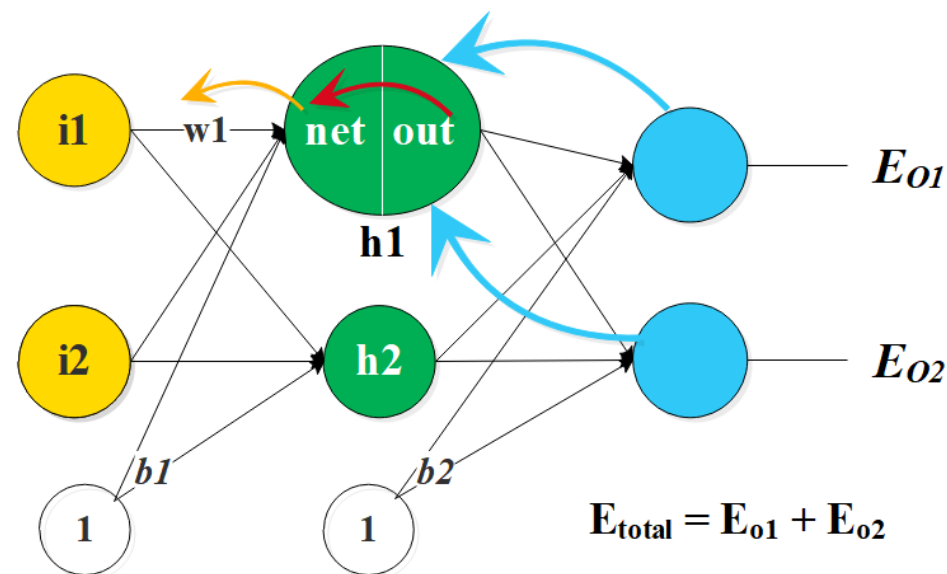
$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

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

$$= 0.036350306 \times 0.241300709 \times 0.05 = 0.000438568$$

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

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



# 输入层-->隐含层的权值更新

为了简化公式，用 $\sigma(h1)$ 表示隐含层单元h1的误差：

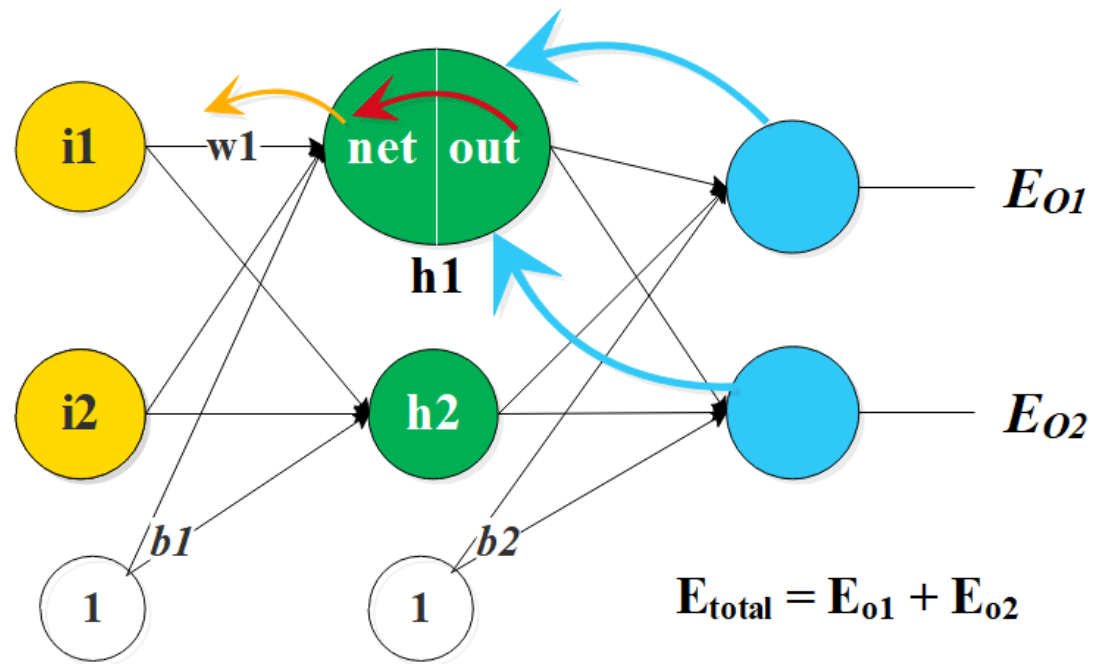
$$\frac{\partial E_{total}}{\partial w_1} = \left( \sum_o \frac{\partial E_{total}}{\partial out_o} \cdot \frac{\partial out_o}{\partial net_o} \cdot \frac{\partial net_o}{\partial out_{h1}} \right) \cdot \frac{\partial out_{h1}}{\partial net_{h1}} \cdot \frac{\partial net_{h1}}{\partial w_1}$$

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

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

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

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



## 输入层-->隐含层的权值更新

最后，更新w1的权值：

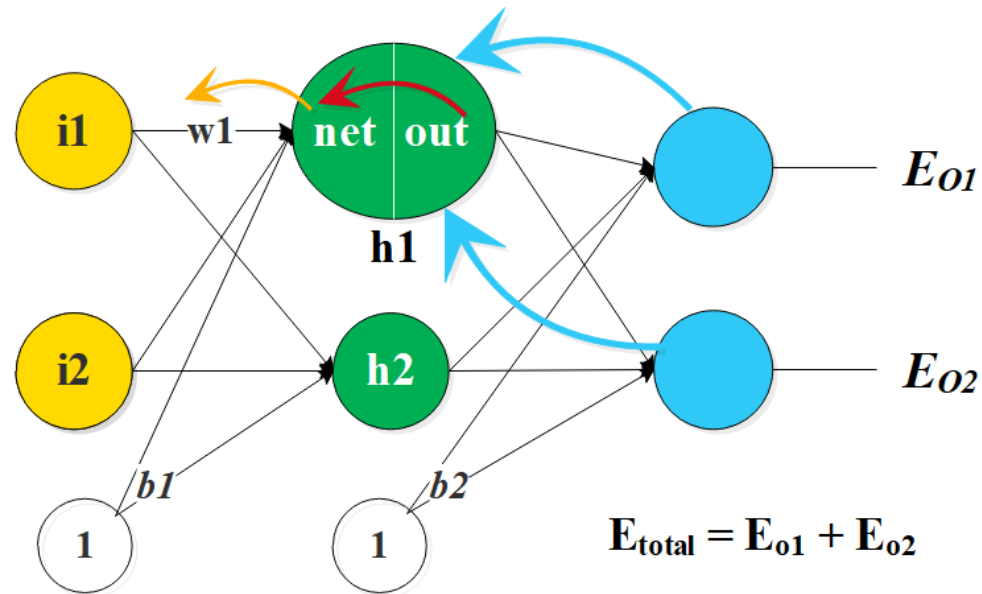
$$w_1^+ = w_1 - \eta \cdot \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 \cdot 0.000438568 = 0.149780716$$

同理，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。

# 本例MATLAB编码实现

```
BackpropNN_ex1.m  +
1 function [Wh, Wo, Y] = BackpropNN_ex1(X, D, alpha, epochs, W1, W2, B)
2 % BackpropNN_ex1针对此例，需进行扩展修改
3
4 %% BP神经网络训练
5 Etotal = zeros(1, epochs);
6 for i = 1:epochs
7     %% 向前传播
8     inhid = W1*X+B(1); %从输入层到隐藏层计算权重和, 4*4
9     hiddout = Sigmoid(inhid); %激活函数，隐藏层输出 4*4
10    out = W2*hiddout + B(2); %从隐藏层到输出层1*4
11    py = Sigmoid(out); %激活函数，输出层输出4*1
12
13    Etotal(i) = sum((D-py).^2)/2; %计算总误差1*1
14
15    %% 反向传播，更新权重
16    DWh = -(D-py). *py. *(1-py); %4*1
17    %反向传播，更新隐藏层到输入层w1的权重
18    W1 = W1 - alpha*sum(DWh'. *W2). *hiddout. *(1-hiddout)*X;
19    delta = DWh. *hiddout;
20    W2 = W2-alpha*delta; %反向传播，更新输入层到隐藏层权重
21 end
22 Wh = W1; Wo = W2;
23
24 %% 最终结果的输出
25 his = W1*X+B(1); %从输入层到隐藏层计算权重和
26 hiddout = Sigmoid(his); %激活函数，隐藏层输出
27 out = W2*hiddout + B(2); %从隐藏层到输出层
28 Y = Sigmoid(out); %激活函数，输出层输出
```

```
30 %% 可视化
31 plot(Etotal(1:500:end), 'b.-', 'LineWidth', 1);
32 title(strcat('BP神经网络SGD误差曲线 Loss = ', ...
33             num2str(Etotal(end))))
34 xlabel('Epochs/500'); ylabel('Loss');grid on
35
36 %% 激活函数
37 function y = Sigmoid(x)
38     y = 1./(1+exp(-x));
39 end
40 end
```

X = [0.05;0.1]; %输入

D = [0.01;0.99]; %实际输出

%输入层到隐藏层权重

W1 = [0.15 0.20; 0.25 0.30];

%隐藏层到输出层权重

W2 = [0.40 0.45; 0.50 0.55];

B = [0.35;0.60]; %偏差值

# 本例MATLAB编码实现

```
>> [Wh,Wo,Y] = BackpropNN_ex1(X,D,0.5,1,W1,W2,B) %训练1次
```

Wh =

0.1497 0.1997

0.2497 0.2997

Wo =

0.3589 0.4089

0.5114 0.5614

Y =

0.7421 0.7753

```
>> [Wh,Wo,Y] = BackpropNN_ex1(X,D,0.5,30000,W1,W2,B)
```

Wh =

0.1221 0.1721

0.2221 0.2721

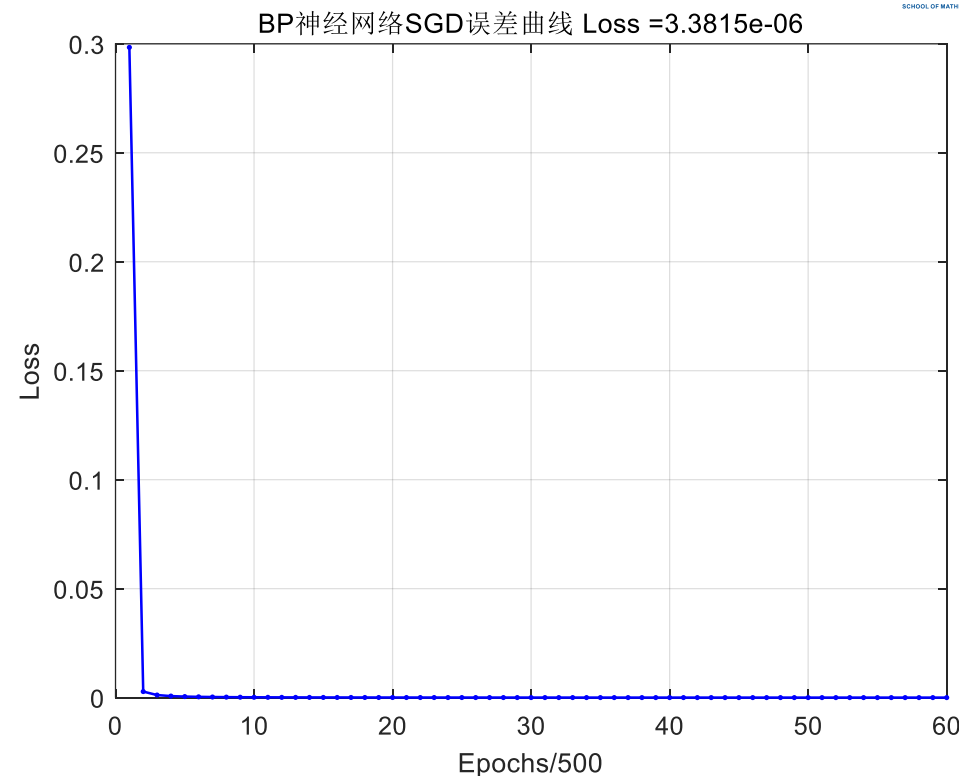
Wo =

-4.2530 -4.2030

3.1947 3.2447

Y =

0.0119 0.9882



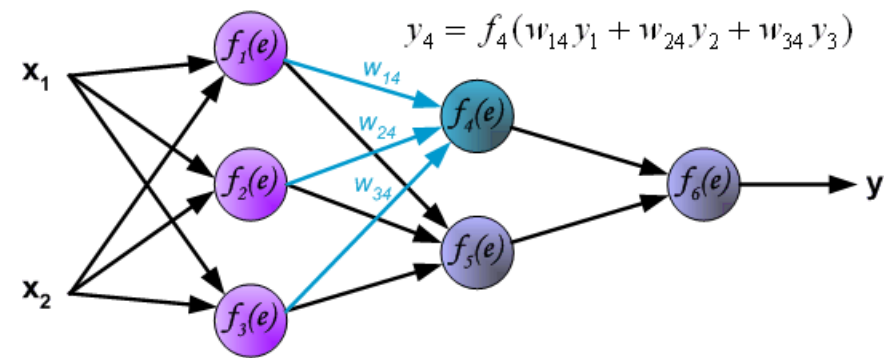
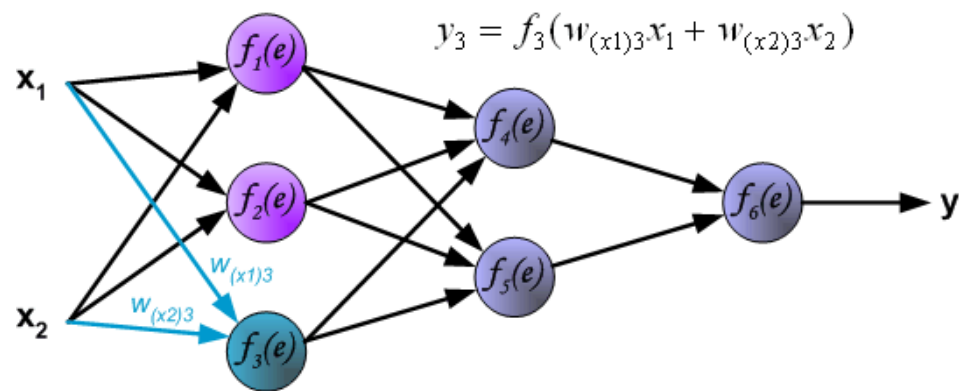
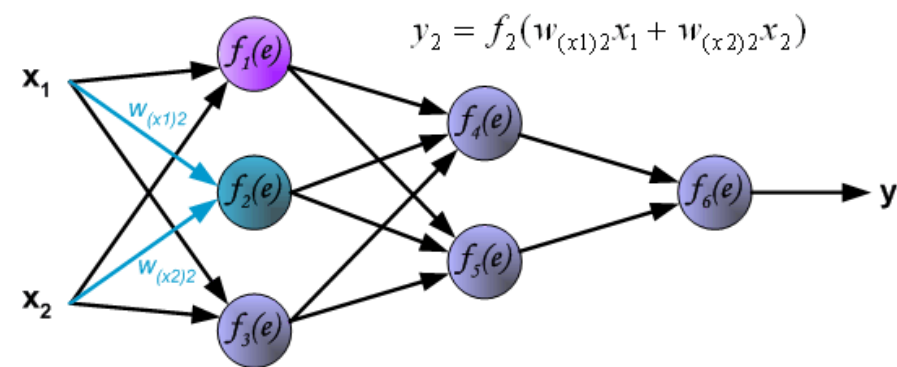
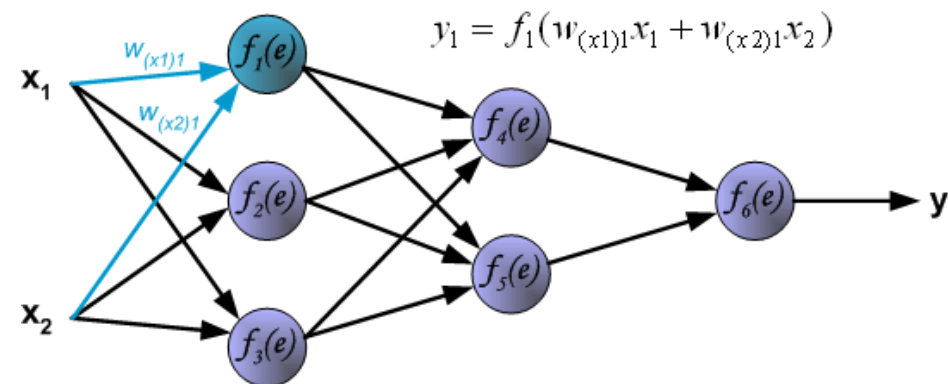
误差收敛，且经过30000次训练，最终  
节点的输出为：

0.0119(正确输出为0.01)

0.9882(正确输出为0.99)

# BP神经网络原理

第一步：前向传播，计算输入层到隐藏层、隐藏层到输出层权重和

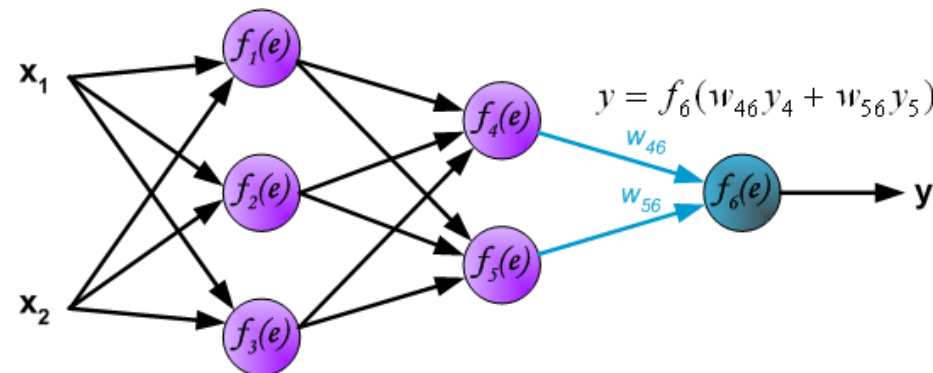
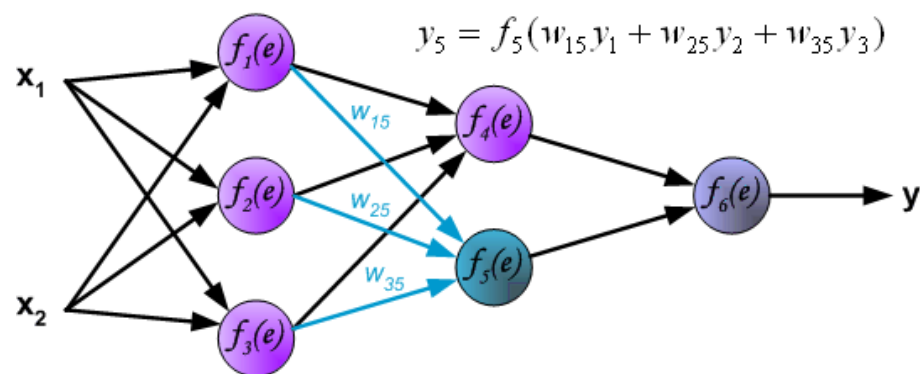


激活函数采用Sigmoid函数

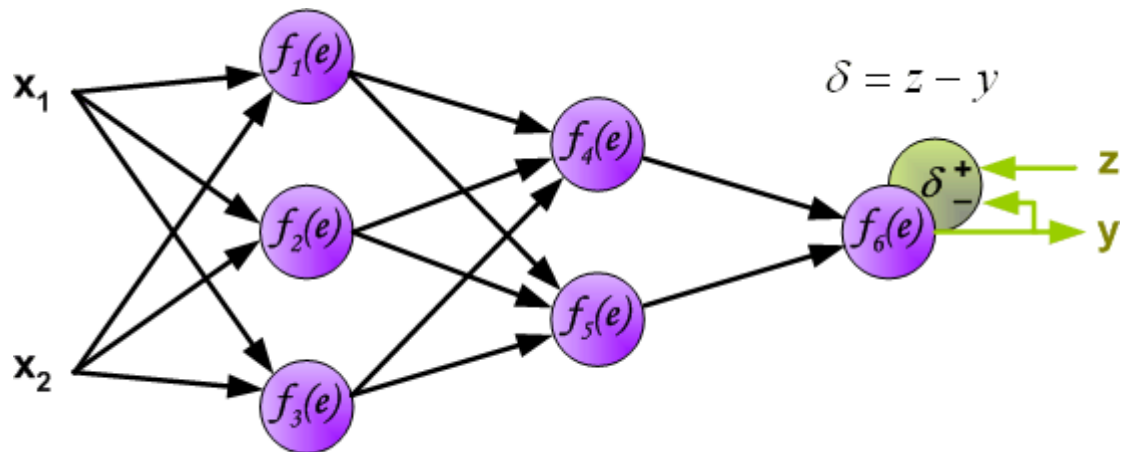


# BP神经网络原理

第一步：前向传播，计算输入层到隐藏层、隐藏层到输出层权重和



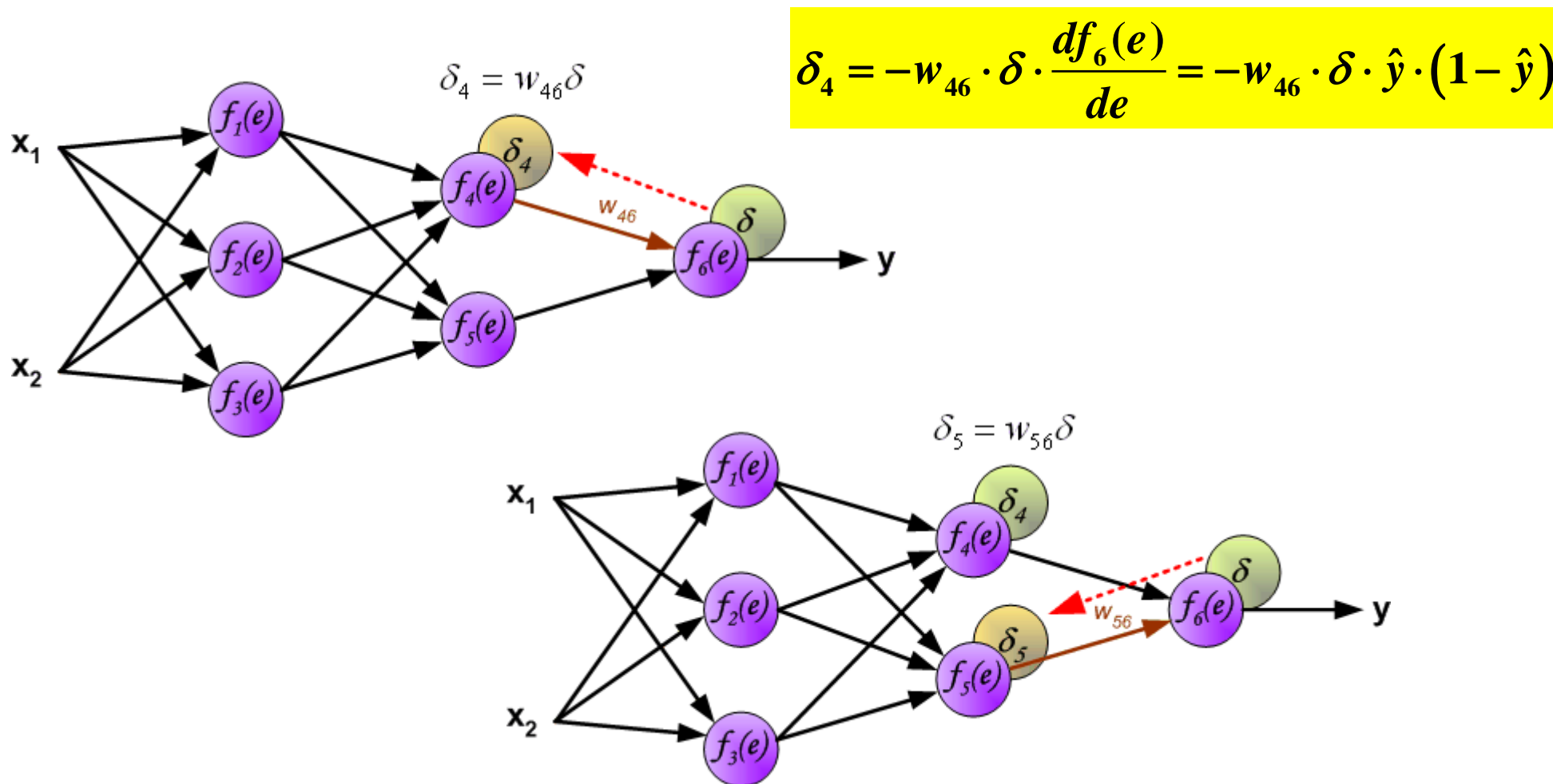
第二步：根据输出值与正确值计算误差 $\delta$



$$\delta = z - y = y - \hat{y}$$

# BP神经网络原理

第三步：反向传播，计算输入层到隐藏层（右）的增量规则

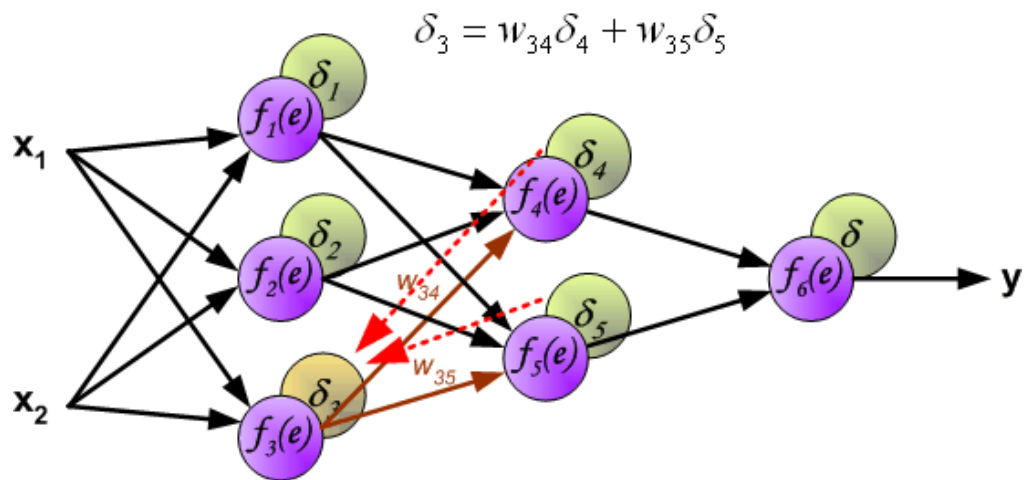
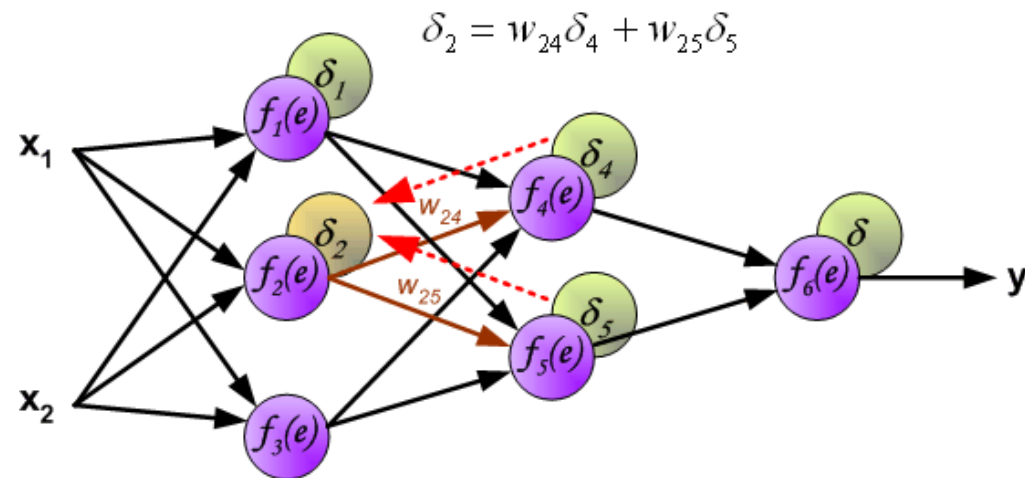
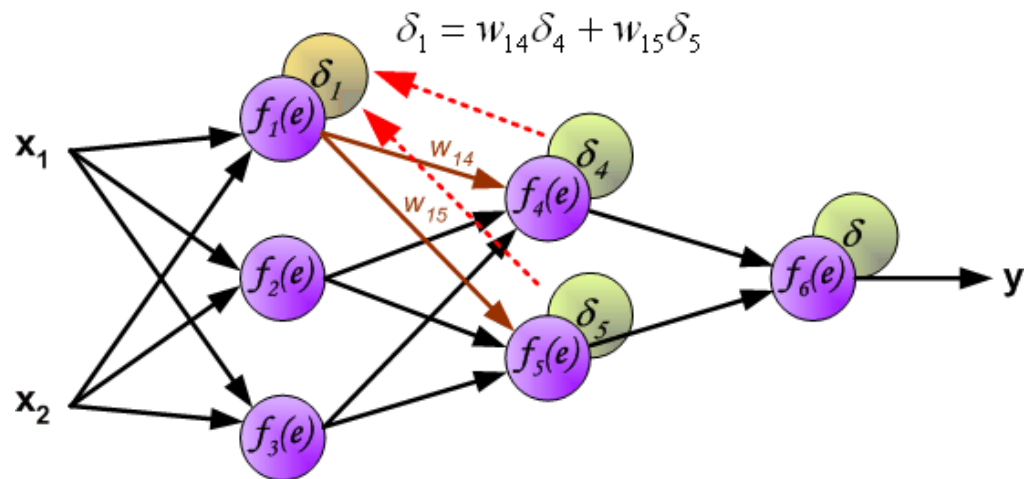


$$\delta_4 = -w_{46} \cdot \delta \cdot \frac{df_6(e)}{de} = -w_{46} \cdot \delta \cdot \hat{y} \cdot (1 - \hat{y})$$

$$\delta = y - \hat{y}$$

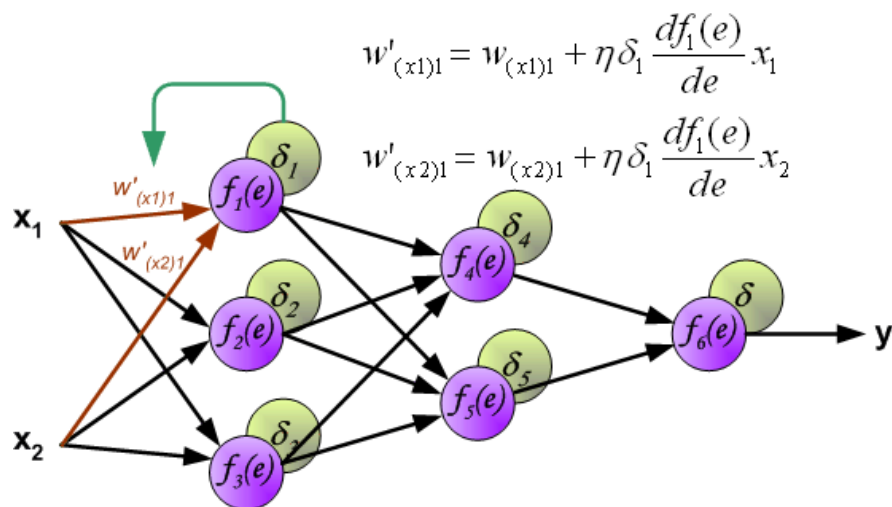
# BP神经网络原理

第四步：反向传播，计算隐藏层（右）到隐藏层（左）的增量规则



$$\begin{bmatrix} \delta_1 \\ \delta_2 \\ \delta_3 \end{bmatrix} = \begin{bmatrix} w_{14} & w_{15} \\ w_{24} & w_{25} \\ w_{34} & w_{35} \end{bmatrix} \begin{bmatrix} \delta_4 \\ \delta_5 \end{bmatrix}$$

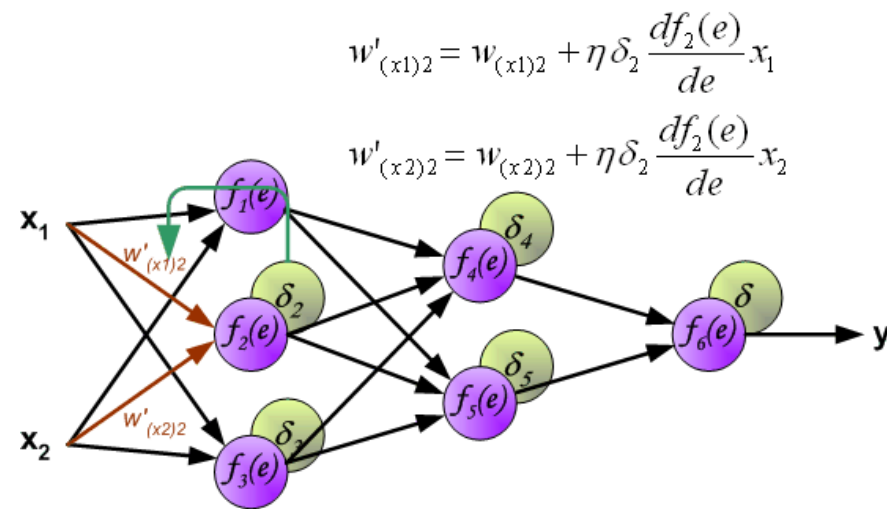
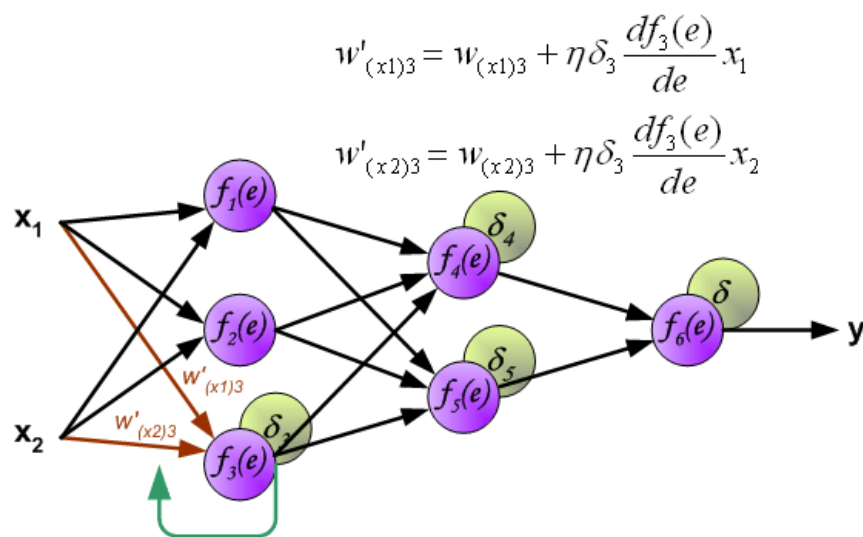
第五步：反向传播，计算隐藏层（左）到输入层的权重更新



$$\hat{w} = w - \eta \cdot \nabla$$

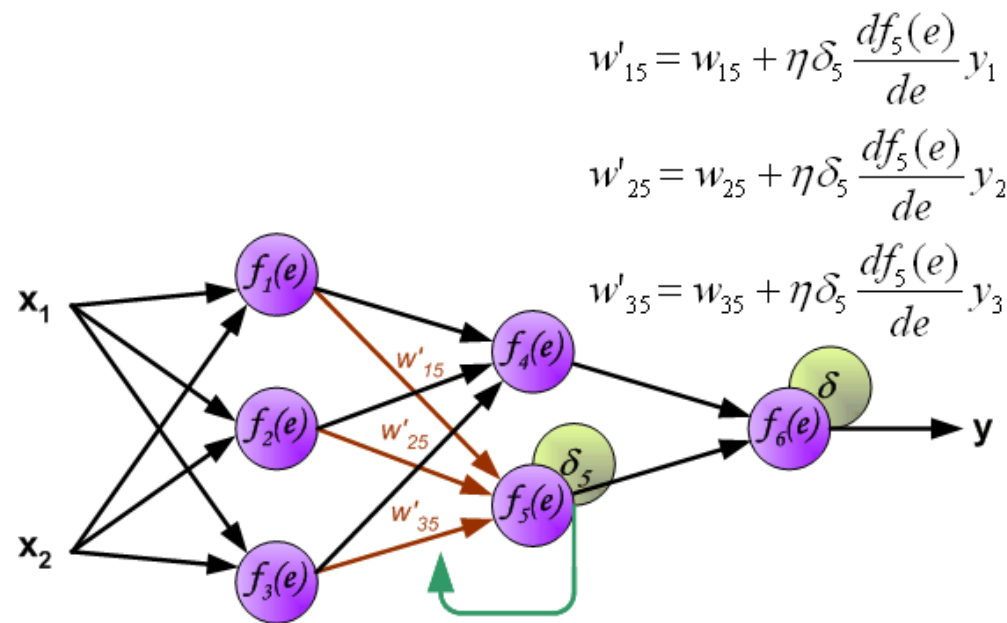
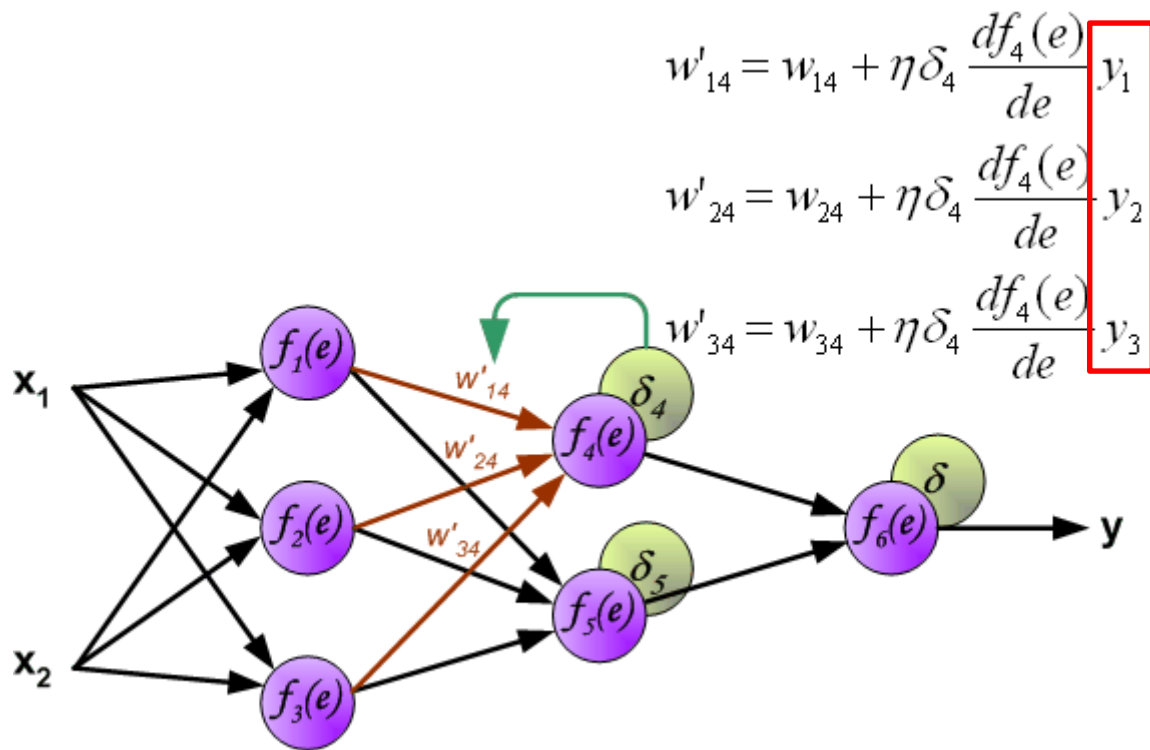
$$= w + \eta \cdot \delta_i \cdot f_i \cdot (1 - f_i) \cdot x$$

$$= w + \eta \cdot \delta_i \cdot \frac{\partial f_i}{\partial e} \cdot x, i = 1, 2, 3$$



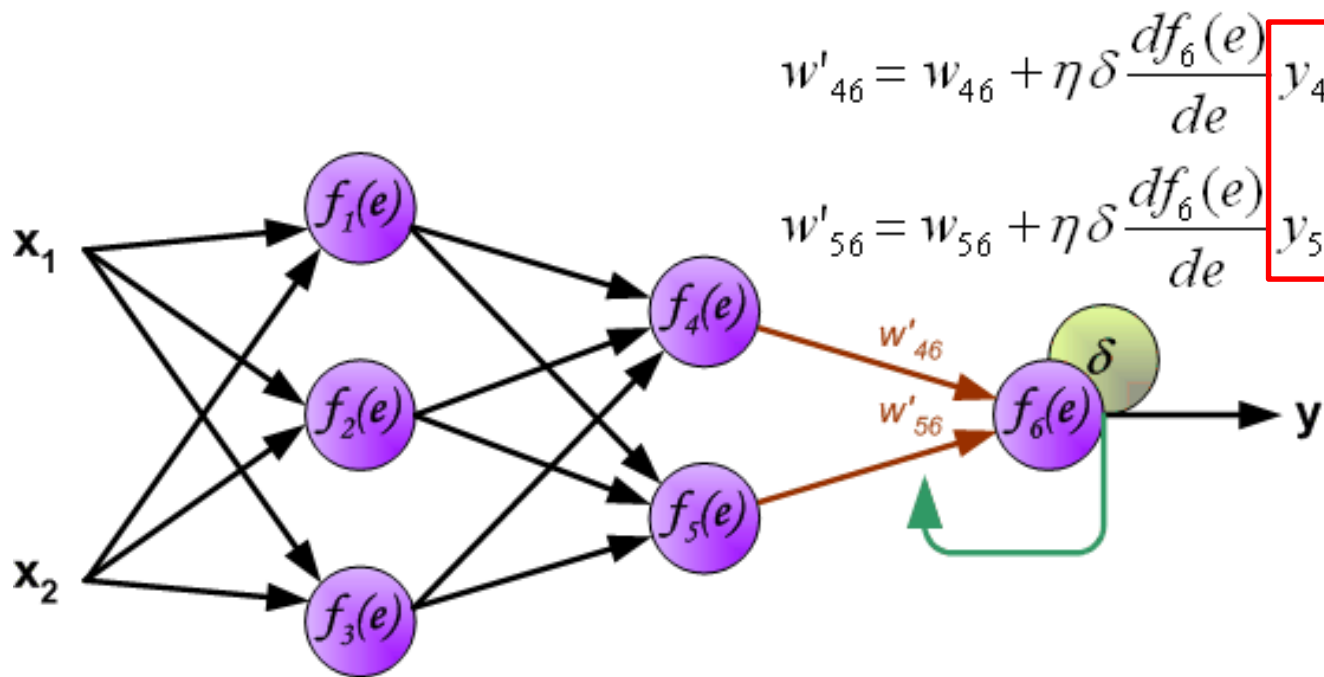
# BP神经网络原理

第六步：反向传播，计算隐藏层（左）到隐藏层（右）的权重更新



# BP神经网络原理

第七步：反向传播，计算隐藏层（右）到输出（右）的权重更新



## 4. 反向传播算法训练网络步骤

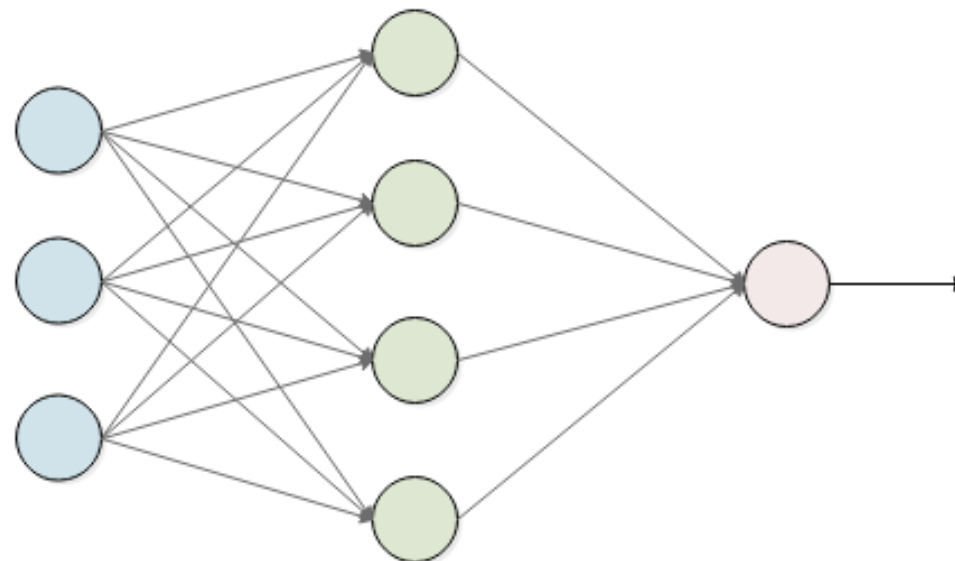
- ① 用适当的值初始化权重;
- ② 计算正确输出与模型输出的误差向量 $E$ , 然后计算输出节点的增量 $\delta$ , 即 $e = d - y$ ,  $\delta = \varphi'(V)E$ ;
- ③ 计算反向传播输出节点的增量 $\delta$ , 并计算下一层 (左侧) 节点的增量值, 即 $E^{(k)} = W^T \delta$ ,  $\delta^{(k)} = \varphi'(V^{(k)})E^{(k)}$ ;
- ④ 重复第③步, 直至计算到输入层右侧的那一隐藏层为止;
- ⑤ 根据公式调整权重值, 即 $\Delta w_{ij} = \alpha \delta_i x_j$ ,  $w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$
- ⑥ 对所有的训练数据节点重复第②~⑤步;
- ⑦ 重复第②~⑥步, 直到神经网络得到了合适的训练。



## 5. 示例

- 单隐藏层神经网络，采用随机梯度下降（SGD）算法。

输入节点			输出节点
0	0	1	0
0	1	1	1
1	0	1	1
1	1	1	0



# MATLAB算法实现



```
1 function [W1,W2,Y] = BackpropNN_demo(X,D,nh,alpha,epochs)
2 % BackpropNN_demo函数简单实现单隐藏层BP神经网络的训练
3 % 输入X和D分别是训练数据和正确输出, epochs为训练次数, err_goal为误差损失目标
4 % alpha是学习率, nh是隐藏层节点个数
5 % W1、W2和Etotal分别是训练最终的权重和总误差
6
7 %% 权重初始化
8 [n,m] = size(X);
9 W1 = 2*rand(nh,m) - 1;
10 W2 = 2*rand(1,nh) - 1;
11 B = rand(2,1);
12
13 %% 网络训练
14 Error = zeros(1,epochs);
15 for i = 1:epochs
16     outo = zeros(1,n);
17     for k = 1:n
18         x = X(k,:); %每次取一个训练数据
19         neth = W1*x' + B(1); %从输入层到隐藏层计算权重和
20         outh = Sigmoid(neth); %激活函数, 隐藏层输出
21         neto = W2*outh + B(2); %从隐藏层到输出层
22         outo(k) = Sigmoid(neto); %激活函数, 输出层输出
23         erro = D(k) - outo(k);
24         delta_o = outo(k)*(1-outo(k))*erro; %反向传播输出节点增量
25         errh = W2'*delta_o; % 反向传播隐藏节点的误差
26         delta_h = outh.*(1-outh).*errh; %计算左侧节点(隐藏层)的增量值
27         dWh = alpha*delta_h*x; %调整隐藏层权重
28         W1 = W1 + dWh; %更新权值(隐藏层到输入层)
29         dWo = alpha*delta_o*outh; %调整权重
30         W2 = W2 + dWo; %更新权值(输出层到隐藏层)
31     end
32 end
```

```
Error(i) = mean((D'-outo).^2); %计算总误差
```

```
end
```

```
%% 可视化
```

```
plot(Error(1:50:epochs),'r.-','LineWidth',1)
```

```
grid on
```

```
xlabel('epochs/50'); ylabel('Loss')
```

```
title(strcat('BP神经网络SGD误差损失曲线 Loss = ',num2str(Error(end))))
```

```
%% 最终节点预测输出
```

```
his = W1*X' + B(1); %从输入层到隐藏层计算权重和
```

```
hiddout = Sigmoid(his); %激活函数, 隐藏层输出
```

```
out = W2*hiddout + B(2); %从隐藏层到输出层
```

```
Y = Sigmoid(out); %激活函数, 输出层输出
```

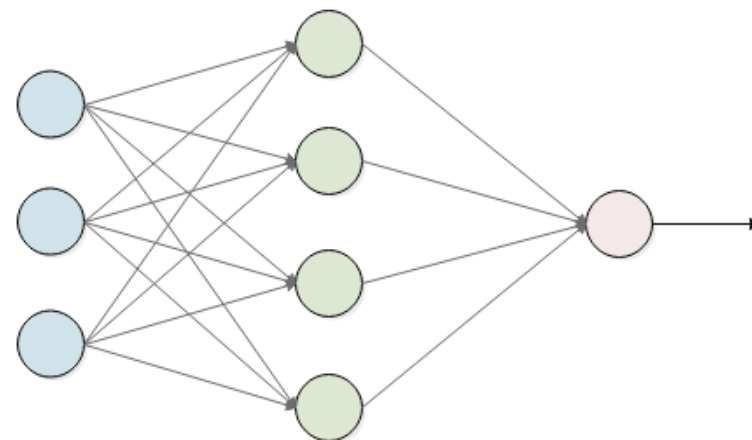
```
%% 激活函数
```

```
function y = Sigmoid(x)
```

```
y = 1./(1+exp(-x));
```

```
end
```

```
end
```



# BP神经网络非线性划分——解决异或问题

```
>> X = [0 0 1;0 1 1;1 0 1; 1 1 1]; %输入  
>> D = [0 1 1 0]'; %实际输出  
>> [W1,W2,Y] = BackpropNN_demo(X,D,4,0.9,10000)
```

W1 =

```
-8.3786   4.9697  -2.1783  
-0.9092   1.0248   2.2492  
 3.6597  -8.2148  -1.1598  
-4.8739  -4.6464   0.9832
```

W2 =

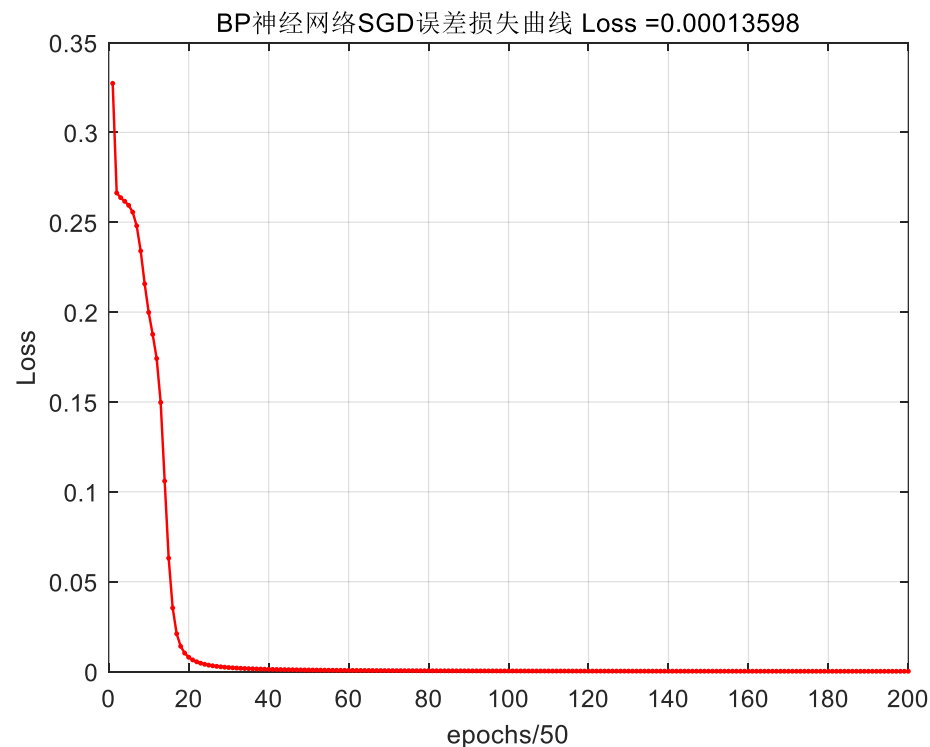
```
 9.3524  -4.4783   8.8060  -6.0683
```

Y =

```
 0.0067   0.9883   0.9883   0.0170
```

从结果可知，添加隐藏层解决了单层神经网络只能线性可分问题，故提升了模型的非线性能力。

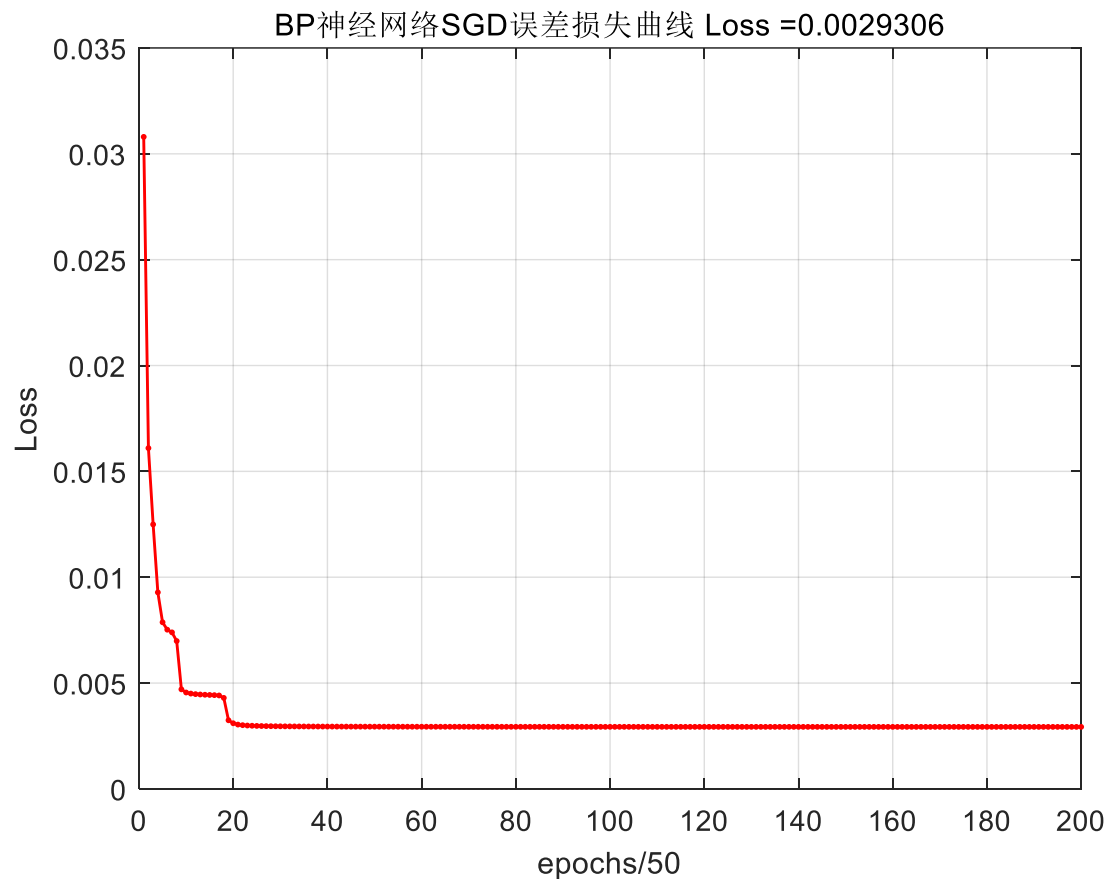
$$\begin{bmatrix} 0.0067 \\ 0.9883 \\ 0.9883 \\ 0.0170 \end{bmatrix} \approx D = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$



# BP神经网络分类：肺癌数据二分类



```
BPNN_classtest.m x +
1 bc = xlsread('breastcancerdata.xlsx');
2 bc(:,1) = [];
3 %% 数据的处理和分类编码
4 X = zscore(bc(:,1:end-1));
5 D = zeros(size(X,1),1);
6 for i = 1:size(X,1)
7     if bc(i,end) == 4
8         D(i) = 1;
9     end
10 end
11 %% 训练网络
12 [W1,W2,Y] = BackpropNN_demo(X,D,9,0.9,10000);
13 %% 实现分类和正确率的计算
14 Yc = zeros(1,100);
15 acc = 0;
16 for i = 1:length(D)
17     if Y(i) < 0.5
18         Yc(i) = 0;
19     elseif Y(i) > 0.5
20         Yc(i) = 1;
21     end
22     if Yc(i) == D(i)
23         acc = acc + 1;
24     end
25 end
26 accrate = acc/length(D)*100
```



分类正确率accrate = 99.7072，比单层神经网络略高。

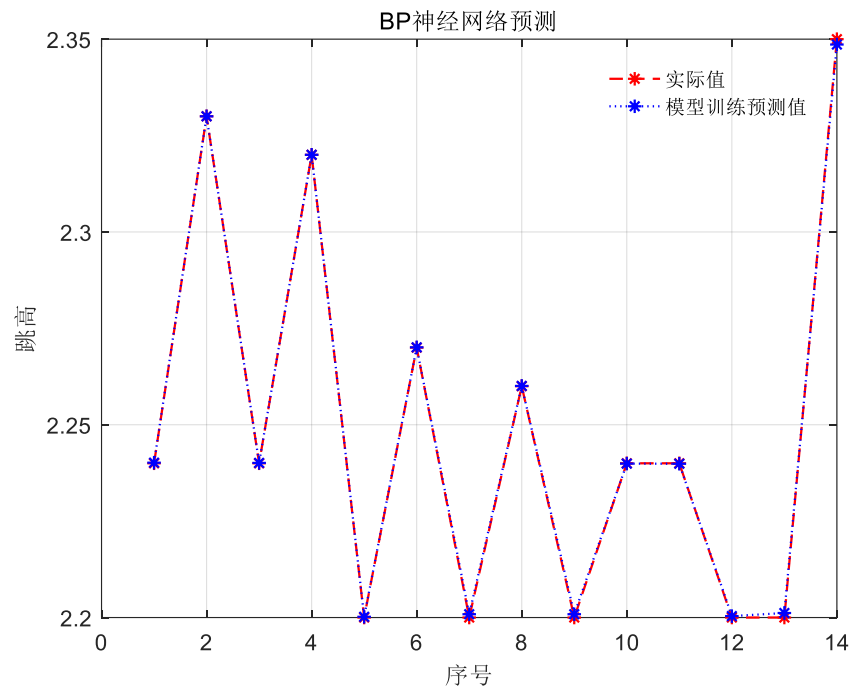
# BP神经网络预测：调高成绩

例1: 国内男子跳高运动员各项素质指标，预测序号15的跳高成绩。

序号	跳高成绩	30行进跑	立定三级跳远	助跑摸高	助跑4—6步跳高	负重深蹲杠铃	杠铃半蹲系数	100米	抓举
1	2.24	3.2	9.6	3.45	2.15	140	2.8	11.0	50
2	2.33	3.2	10.3	3.75	2.2	120	3.4	10.9	70
3	2.24	3.0	9.0	3.5	2.2	140	3.5	11.4	50
4	2.32	3.2	10.3	3.65	2.2	150	2.8	10.8	80
5	2.2	3.2	10.1	3.5	2	80	1.5	11.3	50
6	2.27	3.4	10.0	3.4	2.15	130	3.2	11.5	60
7	2.2	3.2	9.6	3.55	2.1	130	3.5	11.8	65
8	2.26	3.0	9.0	3.5	2.1	100	1.8	11.3	40
9	2.2	3.2	9.6	3.55	2.1	130	3.5	11.8	65
10	2.24	3.2	9.2	3.5	2.1	140	2.5	11.0	50
11	2.24	3.2	9.5	3.4	2.15	115	2.8	11.9	50
12	2.2	3.9	9.0	3.1	2.0	80	2.2	13.0	50
13	2.2	3.1	9.5	3.6	2.1	90	2.7	11.1	70
14	2.35	3.2	9.7	3.45	2.15	130	4.6	10.85	70
15		3.0	9.3	3.3	2.05	100	2.8	11.2	50

# BP神经网络预测：调高成绩

```
hdata = xlsread('highjump.xlsx');  
Train = hdata(1:end-1,3:end)'; %训练数据  
T_train = hdata(1:end-1,2)'; %目标数据  
Test = hdata(end,3:end)'; %待预测数据  
[input,PS] = mapminmax(Train,0,1); %训练数据归一化  
[Output,Pso] = mapminmax(T_train,0,1); %训练数据目标归一化  
%测试数据与训练数据使用相同的归一方法  
testInput = mapminmax('apply',Test,PS);  
[W1,W2,Y] = BackpropNN_demo(input',Output',10,0.9,10000);  
preOut = mapminmax('reverse',Y,Pso); %反归一化  
plot(T_train,'r--*','LineWidth',1)  
hold on; grid on  
plot(preOut,'b:*','LineWidth',1)
```



$y1 = 1./(1+\exp(-W1*\text{testInput}));$

$y = 1./(1+\exp(-W2*y1));$

$\text{preOut} = \text{mapminmax}(\text{'reverse'},y,\text{Pso})$  %反归一化

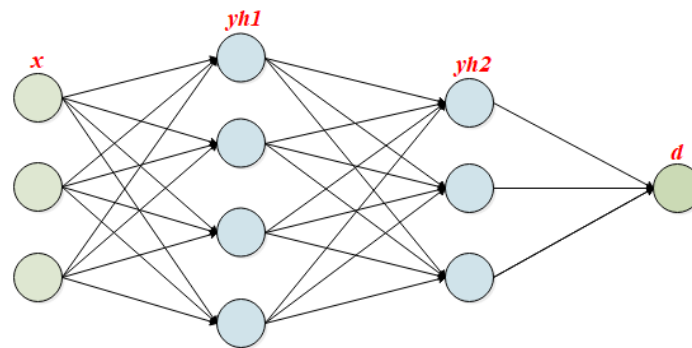
**preOut = 2.2223 %预测值**

# 两层隐藏层MATLAB实现



```
BPNN_h2Train.m  +
1 function [W1, W2, W3, Y] = BPNN_h2Train(X, D, hidden, epochs, alpha)
2 % BPNN_h2Train实现两个隐藏层单个输出节点的BP神经网络
3 % 输入X和D分别是训练数据和正确输出, epochs为训练次数, alpha是学习率
4 % hidden是隐藏层向量, 如[4, 5]表示两层隐藏层, 每层节点数为4和5
5 % W1、W2和Etotal分别是训练最终的权重和总误差
6
7 %% 权重初始化
8 [m, n] = size(X);
9 %随机初始化从输入层到隐藏层权重
10 W1 = 2*rand(hidden(1), n) - 1;
11 %随机初始化从隐藏层到输出层权重
12 W2 = 2*rand(hidden(2), hidden(1)) - 1;
13 %随机初始化从隐藏层到输出层权重
14 W3 = 2*rand(1, hidden(2)) - 1;
15
16 %% 训练网络, 权重更新
17 Error = zeros(1, epochs);
18 for epoch = 1:epochs % train
19     [W1, W2, W3, Etotal] = BackpropXOR2(W1, W2, W3, X, D, alpha);
20     Error(epoch) = Etotal;
21 end
22
23 %% 模型最终输出
24 Y = zeros(1, m);
25 for j = 1:m
26     x = X(j, :);
27     yh1 = Sigmoid(W1*x);
28     yh2 = Sigmoid(W2*yh1);
29     Y(j) = Sigmoid(W3*yh2); %模型输出
30 end
```

```
32 %% 可视化
33 plot(Error(1:50:end), 'r.-', 'LineWidth', 1)
34 grid on
35 xlabel('epochs/50'); ylabel('Loss')
36 title(strcat('BP神经网络SGD误差损失曲线 Loss = ', num2str(Error(end))))
37
38 %% 实现一遍网络训练和权重更新
39 function [W1, W2, W3, Etotal] = BackpropXOR2(W1, W2, W3, X, D, alpha)
40 % 以神经网络的权重和训练数据作为输入, 返回调整后的权重, 2层隐藏层
41 % 其中W1、W2和W3为相应层的权重矩阵; X和D是训练数据的输入和标准输入
42 N = length(D);
43 yo = zeros(1, N);
44 for k = 1:N
45     %% 向前传播
46     x = X(k, :)'; %取每一行作为输入, 以列向量形式
47     yh1 = Sigmoid(W1*x); % 从输入层到隐藏层1权值和, 一列
48     yh2 = Sigmoid(W2*yh1); % 从隐藏层1到隐藏层2权值和, 一列
49     yo(k) = Sigmoid(W3*yh2); % 从隐藏层2到输出层, 模型输出, 单个值
50
51 %% 误差
52 err = D(k) - yo(k);
```

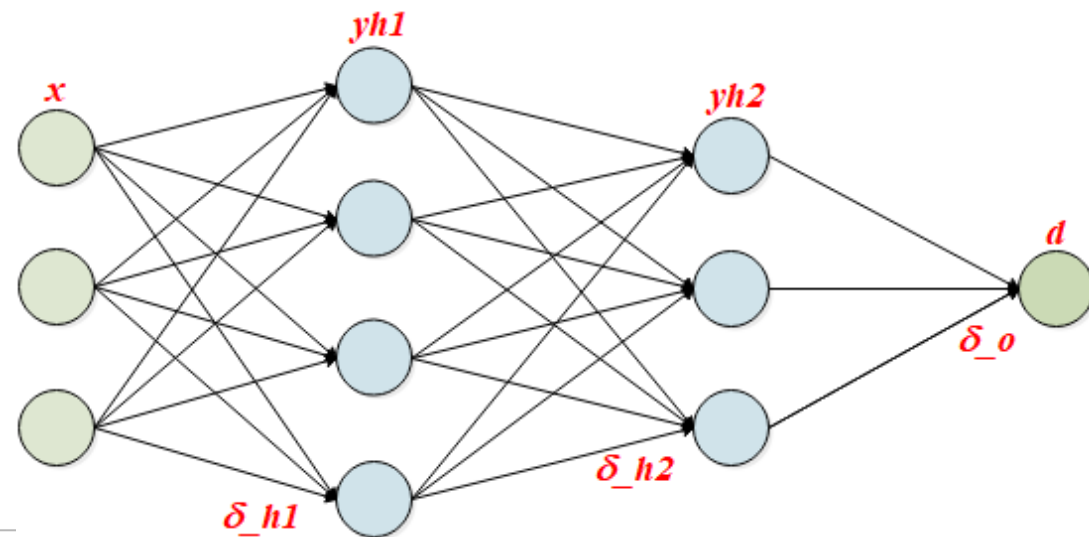




# 两层隐藏层MATLAB实现



```
54 %% 反向传播,更新权重
55 delta_o = Diff_Sigmoid(yo(k))*err; %反向传播输出节点增量
56 eh2 = W3'*delta_o; % 反向传播隐藏节点的误差
57 %计算输出层左侧节点(隐藏层2)的增量值
58 delta_h2 = Diff_Sigmoid(yh2).*eh2;
59 eh1 = W2'*delta_h2; % 反向传播隐藏节点的误差
60 %计算输出层输出层右侧节点(隐藏层1)的增量值
61 delta_h1 = Diff_Sigmoid(yh1).*eh1;
62 dWh1 = alpha*delta_h1*x'; %调整隐藏层权重
63 W1 = W1 + dWh1; %更新权值(输入层到隐藏层)
64 dWh2 = alpha*delta_h2*yh1'; %调整隐藏层权重
65 W2 = W2 + dWh2; %更新权值(隐藏层1到隐藏层2)
66 dWo = alpha*delta_o*yh2'; %调整权重
67 W3 = W3 + dWo; %更新权值(隐藏层2到输出层)
68 end
69 Etotal = 0.5*mean((abs(D'-yo).^2)); %计算总误差
70 end
71
72 %% 激活函数
73 function y = Sigmoid(x)
74     y = 1 ./ (1 + exp(-x));
75 end
76 %% 激活函数导数
77 function y = Diff_Sigmoid(x)
78     y = x.*(1-x);
79 end
80 end
```



# 两层隐藏层MATLAB实现

```
>> X = [0 0 1;0 1 1;1 0 1; 1 1 1]; %输入  
>> D = [0 1 1 0]'; %实际输出  
>> [W1,W2,W3,Y] = BPNN_h2Train(X,D,[4,3],10000,0.9)
```

W1 =

-2.4436	6.1101	0.2458
-5.7124	2.9250	-1.0908
3.5653	4.5488	-0.8842
-3.1076	-3.9600	1.1964

W2 =

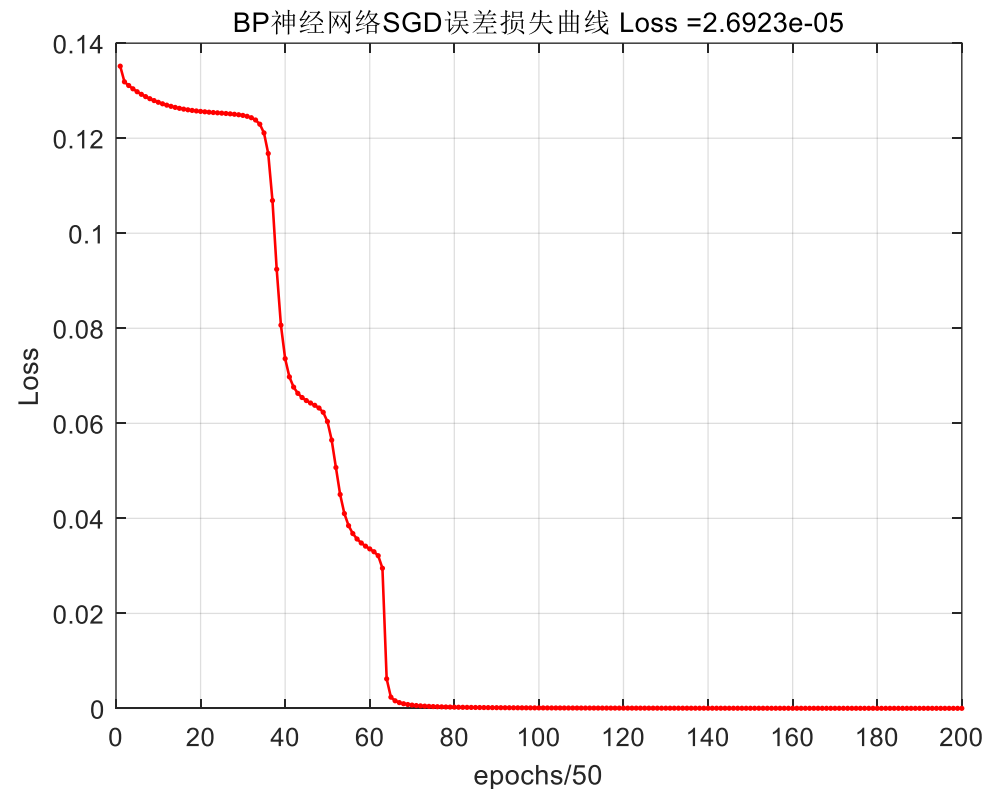
2.7332	-6.7199	-0.3275	-0.5033
-5.8436	2.9740	3.9014	-0.9219
-1.1118	-1.6654	-3.8678	4.9204

W3 =

-6.4340	8.5323	-5.1663
---------	--------	---------

Y =

0.0037	0.9937	0.9921	0.0100
--------	--------	--------	--------



误差曲线下降收敛，且最终训练的结果比单隐藏层要好，但同时也增加了计算量。

## 6. 动量法(Momentum)

- 动量 $m$ 是一个添加到delta规则中用于调整权重的项。
- 使用动量项推动权重在一定程度上向某个特定方向调整，而不是产生立即性改变。
- 它的行为类似于物理学中的动量，能够阻碍物体本身对外力的反应。

$$\begin{cases} \Delta w = \alpha \delta x \\ w = w + \Delta w \end{cases} \Rightarrow \begin{cases} \Delta w = \alpha \delta x \\ m = \Delta w + \beta m^- \\ w = w + m \\ m^- = m \end{cases}$$

$m^-$  是前一个（已计算出的）动量， $\beta$ 是小于1的正的常量。

## 6. 动量法(Momentum)

- 动量随时间的变化

$$\left\{ \begin{array}{l} m(0) = 0 \\ m(1) = \Delta w(1) + \beta m(0) = \Delta w(1) \\ m(2) = \Delta w(2) + \beta m(1) = \Delta w(2) + \beta \Delta w(1) \\ m(3) = \Delta w(3) + \beta m(2) = \Delta w(3) + \beta \Delta w(2) + \beta^2 \Delta w(1) \\ \vdots \quad \quad \quad \vdots \end{array} \right.$$

在过程的每一步，都向动量加上上一步的权重更新，如 $\Delta w(1)$ 、 $\Delta w(2)$ 等。由于 $\beta$ 小于1，所以越早的权重更新对动量的影响越小。虽然影响力随着时间的推移而减弱，但是更早的权重更新仍然存在于动量之中。因此，权重不仅受某个特定权重更新值的影响。故而，学习的稳定性得到提高。此外，动量随着权重更新而逐渐增大。因而权重更新量也随之越来越大。因此，学习的效率也提高了。

## 6. 动量法(Momentum)



```
BackpropMmt.m  +
1 function [W1,W2,Y] = BackpropMmt(X,D,nh,alpha,epochs)
2 % BackpropNN_demo函数简单实现单隐藏层BP神经网络的训练
3 % 输入X和D分别是训练数据和正确输出, epochs为训练次数
4 % alpha是学习率, nh是隐藏层节点个数
5 % W1、W2和Etotal分别是训练最终的权重和总误差
6
7 %% 权重初始化
8 [n,m] = size(X);
9 W1 = 2*rand(nh,m) - 1;
10 W2 = 2*rand(1,nh) - 1;
11 B = rand(2,1);
12 mmt1 = zeros(size(W1));
13 mmt2 = zeros(size(W2));
14 beta = 0.8;
15
16 %% 网络训练
17 Error = zeros(1,epochs);
18 for i = 1:epochs
19     outo = zeros(1,n);
20     for k = 1:n
21         x = X(k,:); %每次取一个训练数据
22         neth = W1*x' + B(1); %从输入层到隐藏层计算权重和
23         outh = Sigmoid(neth); %激活函数, 隐藏层输出
24         neto = W2*outh + B(2); %从隐藏层到输出层
25         outo(k) = Sigmoid(neto); %激活函数, 输出层输出
26         erro = D(k) - outo(k);
27         delta_o = outo(k)*(1-outo(k))*erro; %反向传播输出节点增量
28         errh = W2'*delta_o; % 反向传播隐藏节点的误差
29         delta_h = outh.*(1-outh).*errh; %计算左侧节点(隐藏层)的增量值
30         dWh = alpha*delta_h*x; %调整隐藏层权重
```

```
31 mmt1 = dWh + beta*mmt1; %动量方法
32 W1 = W1 + mmt1; %更新权重(隐藏层到输入层)
33 dWo = alpha*delta_o*outh'; %调整权重
34 mmt2 = dWo + beta*mmt2; %动量方法
35 W2 = W2 + mmt2; %更新权重(输出层到隐藏层)
36 end
37 Error(i) = mean((D'-outo).^2); %计算总误差
38 end
39
40 %% 可视化
41 plot(Error(1:50:epochs),'r.-','LineWidth',1)
42 grid on
43 xlabel('epochs/50'); ylabel('Loss')
44 title(strcat('BP神经网络SGD误差损失曲线 Loss = ',num2str(Error(end))))
45
46 %% 最终节点预测输出
47 his = W1*X' + B(1); %从输入层到隐藏层计算权重和
48 hiddout = Sigmoid(his); %激活函数, 隐藏层输出
49 out = W2*hiddout + B(2); %从隐藏层到输出层
50 Y = Sigmoid(out); %激活函数, 输出层输出
51
52 %% 激活函数
53 function y = Sigmoid(x)
54     y = 1./(1+exp(-x));
55 end
56 end
```

$$\begin{cases} \Delta w = \alpha \delta x \\ m = \Delta w + \beta m^- \\ w = w + m \\ m^- = m \end{cases}$$

## 6. 动量法(Momentum)

```
X = [0 0 1;0 1 1;1 0 1; 1 1 1]; %输入
```

```
D = [0 1 1 0]'; %实际输出
```

```
[W1,W2,Y] = BackpropMmt(X,D,4,0.9,10000)
```

```
W1 =
```

```
4.4233 -9.8045 -2.9389
```

```
0.6853 -2.2071 1.2809
```

```
4.0900 -9.6781 -2.7572
```

```
-10.9999 5.4334 -3.6586
```

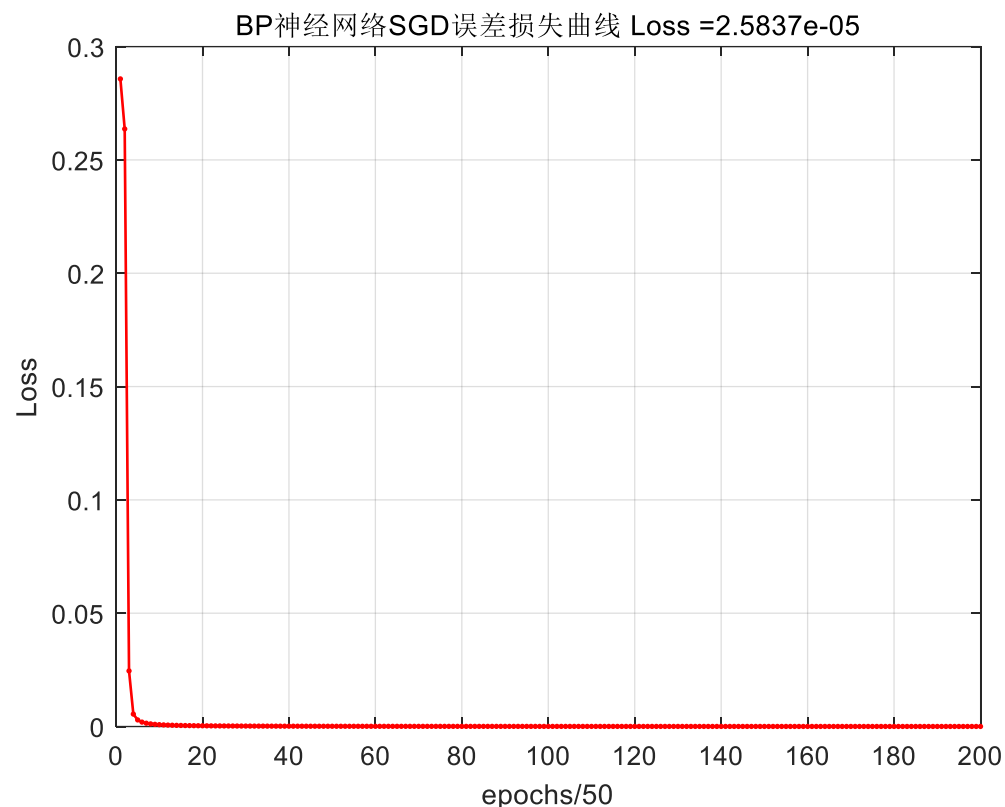
```
W2 =
```

```
7.5427 -9.0019 7.0653 10.1459
```

```
Y =
```

```
0.0057 0.9957 0.9948 0.0039
```

```
(对比普通法) 0.0067 0.9883 0.9883 0.0170
```



同样的训练次数和学习率，动量法比普通权重更新方法得到的结果更精确，学习速度更快。动量法训练2000次的结果与普通法一致。

## 7. 代价函数与学习规则



- 神经网络误差的度量就是代价函数。神经网络的误差越大，代价函数的值就越高。对于神经网络的监督学习，主要有两种代价函数：

$$J = \sum_{i=1}^M \frac{1}{2} (d_i - y_i)^2$$

$y_i$  是输出节点输出

$d_i$  是训练数据的标准输出

$$J = \sum_{i=1}^M \left\{ -d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i) \right\}$$

$M$  是输出节点的数量

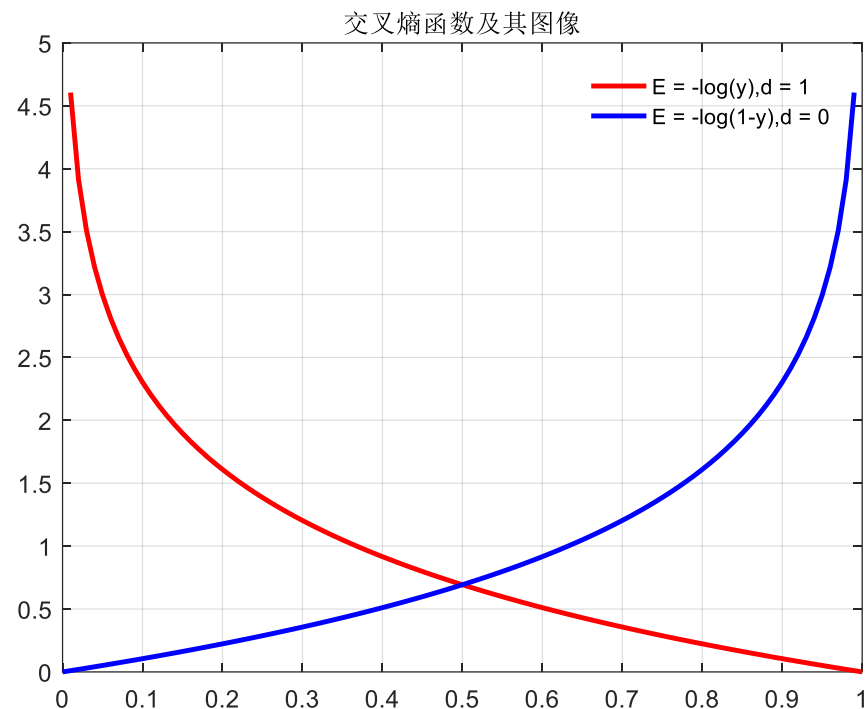
- 交叉熵函数对误差更敏感，且与误差是成正比的，通常认为交叉熵函数导出的学习规则能够得到更好的性能。
- 交叉熵函数与二次函数最主要的不同是，交叉熵函数随着误差的增大而呈几何上升趋势。

# 交叉熵函数分析

$$J = \sum_{i=1}^M \left\{ -d_i \ln(y_i) - (1 - d_i) \ln(1 - y_i) \right\}$$

$$E = \begin{cases} -\ln(y), & d = 1 \\ -\ln(1 - y), & d = 0 \end{cases}$$

交叉熵函数对误差更敏感，且与误差是成正比的，交叉熵函数随着误差的增大而呈几何上升趋势。通常认为交叉熵函数导出的学习规则能够得到更好的性能。





# 训练神经网络的步骤

① 用适当的值初始化神经网络的权重；

② 从训练数据中获得“输入”，将“输入”传递到神经网络模型中，从模型获得输出，并依据“正确输出”计算误差  $E = D - Y$ ,  $\delta = E$ ;

(如果输出节点激活函数为Sigmoid, 那么增量delta就等于输出误差)  $\frac{\partial J}{\partial y} = \frac{1-d}{1-y} - \frac{d}{y} = \frac{y-d}{(1-y)y}$

③ 将输出节点的增量反向传播，计算下一个隐藏层（左侧）节点的增量，即  $E^{(k)} = W^T \delta$ ,  
 $\delta^{(k)} = \varphi'(V^{(k)})E^{(k)}$ ;

④ 重复第③步，直至计算到输入层右侧的那一个隐藏层为止；

⑤ 调整权重，即  $\Delta w_{ij} = \alpha \delta_i x_j$ ,  $w_{ij} \leftarrow w_{ij} + \Delta w_{ij} = w_{ij} + \alpha \delta_i x_j$ ;

⑥ 将所有训练数据重复执行第②—⑤步；

⑦ 重复第②—⑥步，直至神经网络得到合适的训练为止。

# 交叉熵函数BP神经网络实现



```
BackpropNN_CE.m
1 function [W1,W2,Y] = BackpropNN_CE(X,D,nh,alpha,epochs)
2 % BackpropNN_demo函数简单实现单隐藏层BP神经网络的训练,采用交叉熵误差函数
3 % 输入X和D分别是训练数据和正确输出, epochs为训练次数
4 % alpha是学习率, nh是隐藏层节点个数
5 % W1、W2和Etotal分别是训练最终的权重和总误差
6
7 %% 权重初始化
8 [n,m] = size(X);
9 W1 = 2*rand(nh,m) - 1;
10 W2 = 2*rand(1,nh) - 1;
11 B = rand(2,1);
12
13 %% 网络训练
14 Error = zeros(1,epochs);
15 for i = 1:epochs
16     outo = zeros(1,n);
17     for k = 1:n
18         x = X(k,:); %每次取一个训练数据
19         neth = W1*x' + B(1); %从输入层到隐藏层计算权重和
20         outh = Sigmoid(neth); %激活函数, 隐藏层输出
21         neto = W2*outh + B(2); %从隐藏层到输出层
22         outo(k) = Sigmoid(neto); %激活函数, 输出层输出
23         erro = D(k) - outo(k);
24         delta_o = erro; %交叉熵函数
25         errh = W2'*delta_o; % 反向传播隐藏节点的误差
26         delta_h = outh.*(1-outh).*errh; %计算左侧节点(隐藏层)的增量值
27         dWh = alpha*delta_h*x; %调整隐藏层权重
28         W1 = W1 + dWh; %更新权值(隐藏层到输入层)
29         dWo = alpha*delta_o*outh'; %调整权重
30         W2 = W2 + dWo; %更新权值(输出层到隐藏层)
31     end
end
```

```

Error(i) = mean((D'-outo).^2); %计算总误差
end

%% 可视化
plot(Error(1:50:epochs),'r.-','LineWidth',1)
grid on
xlabel('epochs/50'); ylabel('Loss')
title(strcat('BP神经网络交叉熵函数误差损失曲线 Loss = ',num2str(Error(end))))

%% 最终节点预测输出
his = W1*X' + B(1); %从输入层到隐藏层计算权重和
hiddout = Sigmoid(his); %激活函数, 隐藏层输出
out = W2*hiddout + B(2); %从隐藏层到输出层
Y = Sigmoid(out); %激活函数, 输出层输出

%% 激活函数
function y = Sigmoid(x)
    y = 1./(1+exp(-x));
end
end
```

# 交叉熵函数训练示例

```
>> X = [0 0 1;0 1 1;1 0 1; 1 1 1]; %输入  
>> D = [0 1 1 0]'; %实际输出  
>> [W1,W2,Y] = BackpropNN_CE(X,D,4,0.9,10000)
```

W1 =

-8.2483	4.9064	-2.3774
3.3993	3.3520	-5.0972
-7.9739	-8.3302	2.8772
4.9472	-8.4142	-2.3951

W2 =

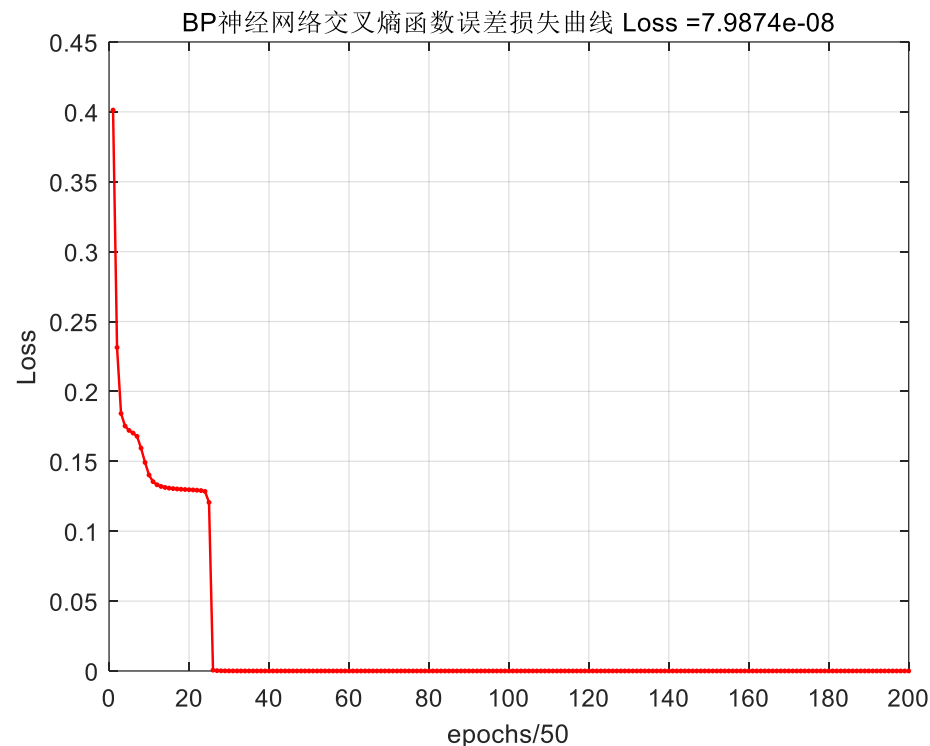
11.3387	-10.7067	-13.5619	11.4734
---------	----------	----------	---------

Y =

0.0001	0.9999	0.9999	0.0002
--------	--------	--------	--------

(动量法: 0.0044 0.9937 0.9930 0.0119)

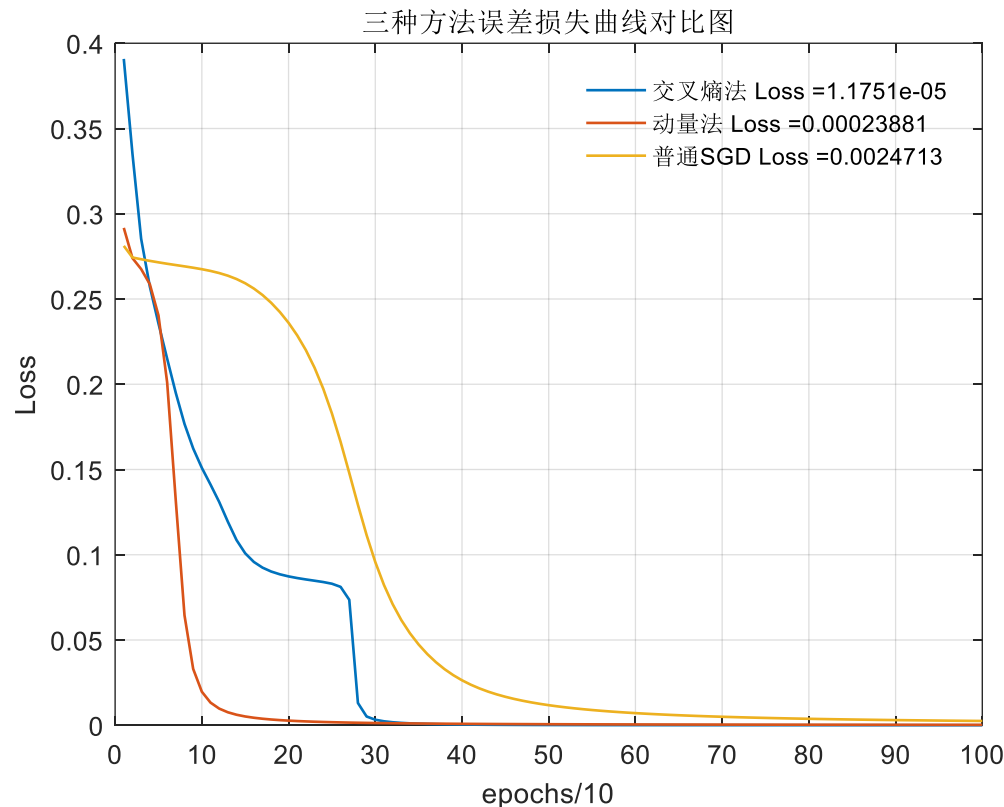
(SGD: 0.0079 0.9896 0.9903 0.0147)



从训练结果可以看出，交叉熵函数训练，其学习速度更快。

# 交叉熵函数训练示例

```
X = [0 0 1;0 1 1;1 0 1; 1 1 1]; %输入
D = [0 1 1 0]'; %实际输出
[~,~,~,Errorsgd] = BackpropNN_demo(X,D,4,0.9,1000);
[~,~,~,Errormmt] = BackpropMmt(X,D,4,0.9,1000);
[~,~,~,Errorce] = BackpropNN_CE(X,D,4,0.9,1000);
plot(Errorce(1:10:end),'LineWidth',1)
hold on;grid on
plot(Errormmt(1:10:end),'LineWidth',1)
plot(Errorsgd(1:10:end),'LineWidth',1)
le1 = strcat('交叉熵法 Loss = ',num2str(Errorce(end)));
le2 = strcat('动量法 Loss = ',num2str(Errormmt(end)));
le3 = strcat('普通SGD Loss = ',num2str(Errorsgd(end)));
legend(le1,le2,le3); legend('boxoff')
title('三种方法误差损失曲线对比图')
xlabel('epochs/10'); ylabel('Loss')
```



从图中可以看出，交叉熵函数在最初的训练过程中不如动量法，但且后期下降速度非常快，最终误差最小。

## 8. 神经网络工具箱

### (1) 归一化函数

- $[Y, PS] = \text{mapminmax}(X)$
- $[Y, PS] = \text{mapminmax}(X, FP)$  : 指定FP, 默认(-1,1)
- $Y = \text{mapminmax}('apply', X, PS)$  : 主要用于测试数据
- $X = \text{mapminmax}('reverse', Y, PS)$  : 预测后数据还原

### (2) 神经网络常见训练函数'trainFun'

### (3) 神经网络学习参数'BLF'

### (4) 神经网络创建、训练和模拟预测: fitnet、train、sim

# (1) 归一化处理函数

```
>> x1 = [1 2 4];  
>> x2 = [5 2 3];  
>> [y,ps] = mapminmax(x1)
```

```
y =  
-1.0000    -0.3333     1.0000
```

```
ps =  
包含以下字段的 struct:
```

```
    name: 'mapminmax'  
   xrows: 1  
    xmax: 4  
    xmin: 1  
   xrange: 3  
   yrows: 1  
    ymax: 1  
    ymin: -1  
   yrange: 2  
    gain: 0.6667  
  xoffset: 1  
no_change: 0
```

- $[Y,PS] = \text{mapminmax}(X)$  : 其中y是对进行某种规范化后得到的数据, 这种规范化的映射记录在结构体ps中, 默认YMIN和YMAX分别是-1和1。
- $y = (y_{\max} - y_{\min}) * (x - x_{\min}) / (x_{\max} - x_{\min}) + y_{\min}$ ; [ 算法的假设是每一行的元素都不相同, 如果有一行的元素都相同, 比如 $x_t = [1 \ 1 \ 1]$ , 此时 $x_{\max} = x_{\min} = 1$ , 把此时的变换变为 $y = y_{\min}$ , 否则除以0, 没有意义! ]
- 对 $x_1 = [1 \ 2 \ 4]$ 采用这个映射 f:  $2 * (x - x_{\min}) / (x_{\max} - x_{\min}) + (-1)$ , 就可以得到  $y = [-1.0000 \ -0.3333 \ 1.0000]$

对于 $x_1$ 而言,  $x_{\min} = 1$ 、 $x_{\max} = 4$ ; 则

- $y(1) = 2 * (1 - 1) / (4 - 1) + (-1) = -1$ ;
- $y(2) = 2 * (2 - 1) / (4 - 1) + (-1) = -1/3 = -0.3333$ ;
- $y(3) = 2 * (4 - 1) / (4 - 1) + (-1) = 1$ ;

# (1) 归一化处理函数

对于上面algorithm中 $y = (y_{\max} - y_{\min}) * (x - x_{\min}) / (x_{\max} - x_{\min}) + y_{\min}$ 的映射函数，其中 $y_{\min}$ 、 $y_{\max}$ 是参数，可以自己设定，默认为-1,1。

```

>> [y,ps] = mapminmax(x1);
>> ps.ymin = 0;
>> [y1,ps1] = mapminmax(x1,ps)
y1 =
    0    0.3333    1.0000
% 对x2数据采用相同的归一化方法
% 这里注意的是归一化结构体的选择ps1
>> y2 = mapminmax('apply',x2,ps1)
y2 =
    1.3333    0.3333    0.6667
  
```

- 则此时的映射函数为： $f: 1 * (x - x_{\min}) / (x_{\max} - x_{\min}) + (0)$
- 如果对 $x_1 = [1 \ 2 \ 4]$ 采用了某种规范化的方式，现在要对 $x_2 = [5 \ 2 \ 3]$ 采用同样的规范化方式[同样的映射]，方法： $\text{mapminmax('apply',x2,ps)}$
- $X = \text{mapminmax('reverse', Y, PS)}$ 的作用就是进行反归一化，将归一化的数据反归一化再得到原来的数据。

```

>> xt = mapminmax('reverse',y1,ps1) %数据的还原
xt =
    1     2     4
>> xt2 = mapminmax('reverse',y2,ps1) %数据的还原
xt2 =
     5     2     3
  
```

## (2) 神经网络常见训练函数



序号	训练方法	训练函数	适用网络规模
1	梯度下降法	traingd	
2	有动量的梯度下降法	traingdm	
3	自适应lr梯度下降法	traingda	
4	自适应lr动量梯度下降法	traingdx	
5	弹性梯度下降法	trainrp	大型网络的首选算法
6	Fletcher-Reeves共轭梯度法	traincgf	共轭梯度算法
7	Ploak-Ribiere共轭梯度法	traincgp	共轭梯度算法
8	Powell-Beale共轭梯度法	traincgb	共轭梯度算法
9	量化共轭梯度法	trainscg	大型网络的首选算法
10	拟牛顿算法	trainbfg	大型网络的首选算法
11	一步正割算法	trainoss	大型网络的首选算法
12	Levenberg-Marquardt	trainlm	中型网络的首选算法
13	贝叶斯正则化算法	trainbr	中型网络的首选算法



### (3) 神经网络学习参数

1. `net.trainParam.epochs` 最大训练次数 (缺省为10)
2. `net.trainParam.goal` 训练要求精度 (缺省为0)
3. `net.trainParam.lr` 学习率 (缺省为0.01)
4. `net.trainParam.max_fail` 最大失败次数 (缺省为5)
5. `net.trainParam.min_grad` 最小梯度要求 (缺省为 $1e-10$ )
6. `net.trainParam.show` 显示训练迭代过程 (NaN表示不显示, 缺省为25)
7. `net.trainParam.time` 最大训练时间 (缺省为inf)
8. `net.trainParam.mc` 动量因子 (缺省0.9)
9. `net.trainParam.lr_inc` 学习率lr增长比 (缺省为1.05)
10. `net.trainParam.lr_dec` 学习率lr下降比 (缺省为0.7)
11. `net.trainParam.max_perf_inc` 表现函数增加最大比 (缺省为1.04)
12. `net.trainParam.delt_inc` 权值变化增加量 (缺省为1.2)

### (3) 神经网络学习参数

13. `net.trainParam.delt_dec` 权值变化减小量 (缺省为0.5)
14. `net.trainParam.delt0` 初始权值变化 (缺省为0.07)
15. `net.trainParam.deltamax` 权值变化最大值 (缺省为50.0)
16. `net.trainParam.searchFcn` 一维线性搜索方法 (缺省为`srchcha`)
17. `net.trainParam.sigma` 因为二次求导对权值调整的影响参数 (缺省值 $5.0e-5$ )
18. `net.trainParam.lambda` Hessian矩阵不确定性调节参数 (缺省为 $5.0e-7$ )
19. `net.trainParam.mem_reduc` 控制计算机内存/速度的参量, 内存较大设为1, 否则设为2 (缺省为1)
20. `net.trainParam.mu_u`  $u$ 的初始值 (缺省为0.001)
21. `net.trainParam.mu_dec_u`  $u$ 的减小率 (缺省为0.1)
22. `net.trainParam.mu_inc_u`  $u$ 的增长率 (缺省为10)
23. `net.trainParam.mu_max_u`  $u$ 的最大值 (缺省为 $1e10$ )

## (4) 神经网络学习主要函数

### newff功能：建立一个前馈反向传播（BP）网络（不再推荐）

- `net = newff (P, T, S)` P: 输入数据矩阵（归一化[0 1]或[-1 1] ）， $R \times Q_1$ ， $Q_1$ 代表R元的输入向量，即P有 $Q_1$ 列，每一列都是一个样本，而每个样本有R个属性（特征）。T: 目标数据矩阵（归一化[0 1]或[-1 1] ）， $SN \times Q_2$ ， $Q_2$ 代表SN元的目标向量，数据意义参考上面。S: 第i层的神经元个数。（新版本中可以省略输出层神经元个数，因为输出层的神经元个数取决于T）
- `net = newff (P, T, S, TF, BTF, BLF, PF, IPF, OPF, DDF)`（提供了可选择的参数）TF: 相关层的传递函数，默认隐含层使用tansig函数，输出层使用purelin函数。BTF: BP神经网络学习训练函数，默认值为trainlm函数。BLF: 权重学习函数，默认值learnngdm。PF: 性能函数，默认值为mse。PF, OPF, DDF均为默认值即可。

## (4) 神经网络学习主要函数

推荐使用feedforwardnet和fitnet，其中fitnet调用feedforwardnet

### Syntax

```
feedforwardnet(hiddenSizes,trainFcn)
```

### Description

Feedforward networks consist of a series of layers. The first layer has a connection from the network input. Each subsequent layer has a connection from the previous layer. The final layer produces the network's output.

Feedforward networks can be used for any kind of input to output mapping. A feedforward network with one hidden layer and enough neurons in the hidden layers, can fit any finite input-output mapping problem.

Specialized versions of the feedforward network include fitting ([fitnet](#)) and pattern recognition ([patternnet](#)) networks. A variation on the feedforward network is the cascade forward network ([cascadeforwardnet](#)) which has additional connections from the input to every layer, and from each layer to all following layers.

`feedforwardnet(hiddenSizes,trainFcn)` takes these arguments,

<code>hiddenSizes</code>	Row vector of one or more hidden layer sizes (default = 10)
<code>trainFcn</code>	Training function (default = 'trainlm')

## (4) 神经网络学习主要函数

`net = fitnet(hiddenSizes)` returns a function fitting neural network with a hidden layer size of `hiddenSizes`.

`net = fitnet(hiddenSizes, trainFcn)` returns a function fitting neural network with a hidden layer size of `hiddenSizes` and training function, specified by `trainFcn`.

- 隐含层和输出层激活函数的选择对BP神经网络预测精度有较大影响，一般隐含层节点传递函数选用tansig函数或logsig函数，输出层节点转移函数选用tansig函数或purelin函数。
- 如果不用激活函数，每一层输出都是上层输入的线性函数，无论神经网络有多少层，输出都是输入的线性组合。
- 如果使用的话，激活函数给神经元引入了非线性因素，使得神经网络可以任意逼近任何非线性函数，这样神经网络就可以应用到众多的非线性模型中。
- 常见激活函数：sigmoid函数、Tanh函数、ReLU、softmax函数

## (4) 神经网络学习主要函数

- train函数 语法：
  - 网络学习函数： `[net,tr] = train(net,X,Y)`
  - net: 是feedforwardnet或fitnet构建的网络对象；
  - X: 训练数据输入； Y: 训练数据正确输出
  - 输出net是训练后的网络， tr是结构体训练的记录。

This function trains a shallow neural network. For deep learning with convolutional or LSTM neural networks, see [trainNetwork](#) instead.

---

`trainedNet = train(net,X,T,Xi,Ai,EW)` trains a network net according to `net.trainFcn` and `net.trainParam`.

[example](#)

---

`[trainedNet,tr] = train(net,X,T,Xi,Ai,EW)` also returns a training record.

---

`[trainedNet,tr] = train(net,X,T,Xi,Ai,EW,Name,Value)` trains a network with additional options specified by one or more name-value pair arguments.

[example](#)

## (4) 神经网络学习主要函数

- `sim`函数  $Y = \text{sim}(\text{net}, X)$ 
  - $X$ : 输入给网络的 $K \times N$ 矩阵,  $K$ 为网络输入个数,  $N$ 为样本数据量
  - $Y$ : 输出矩阵 $Q \times N$ , 其中 $Q$ 为网络输出个数

### Syntax

```
[Y,Xf,Af] = sim(net,X,Xi,Ai,T)
[Y,Xf,Af] = sim(net,{Q TS},Xi,Ai)
[Y,...] = sim(net,...,'useParallel',...)
[Y,...] = sim(net,...,'useGPU',...)
[Y,...] = sim(net,...,'showResources',...)
[Ycomposite,...] = sim(net,Xcomposite,...)
[Ygpu,...] = sim(net,Xgpu,...)
```

$[Y, Xf, Af] = \text{sim}(\text{net}, X, Xi, Ai, T)$  takes

$\text{net}$	Network
$X$	Network inputs
$Xi$	Initial input delay conditions (default = zeros)
$Ai$	Initial layer delay conditions (default = zeros)
$T$	Network targets (default = zeros)

and returns

$Y$	Network outputs
$Xf$	Final input delay conditions
$Af$	Final layer delay conditions

# 参数设置对神经网络性能的影响

## 1. 隐含层节点个数

隐含层节点的个数对于识别率的影响并不大，但是节点个数过多会增加运算量，使得训练较慢。

## 2. 激活函数的选择

激活函数无论对于识别率或收敛速度都有显著的影响。在逼近高次曲线时，S形函数精度比线性函数要高得多，但计算量也要大得多。

## 3. 学习率的选择

学习率影响着网络收敛的速度，以及网络能否收敛。学习率设置偏小可以保证网络收敛，但是收敛较慢。相反，学习率设置偏大则有可能使网络训练不收敛，影响识别效果。



1. 导入需要处理的数据
2. 对数据进行打乱randperm, 随机获取实验数据、目标数据和测试数据（注意：每一列一个样本），并进行标准化处理(mapminmax)
3. 构建一个训练网络：net = feedforwardnet(hiddenSizes,trainFcn)
4. 对构建的网络进行训练：net = train(net,X,T)
5. 进行数据仿真操作Y = sim(net,testInput)
6. 验证和后评价
7. 绘图直观的显示数据模拟训练的效果

# 案例1：甲状腺疾病分类问题

<http://archive.ics.uci.edu/ml/datasets.php>



Thyroid Disease (thyroid0387) data set			
Type	Classification	Origin	Real world
Features	21	(Real / Integer / Nominal)	(6 / 15 / 0)
Instances	7200	Classes	3
Missing values?			No

Attribute	Domain	Attribute	Domain	Attribute	Domain
Age	[0.01, 0.97]	Thyroid_surgery	[0, 1]	Hypopituitary	[0, 1]
Sex	[0, 1]	l131_treatment	[0, 1]	Psych	[0, 1]
On_thyroxine	[0, 1]	Query_hypothyroid	[0, 1]	TSH	[0.0, 0.53]
Query_on_thyroxine	[0, 1]	Query_hyperthyroid	[0, 1]	T3	[0.0005, 0.18]
On_antithyroid_medication	[0, 1]	Lithium	[0, 1]	TT4	[0.0020, 0.6]
Sick	[0, 1]	Goitre	[0, 1]	T4U	[0.017, 0.233]
Pregnant	[0, 1]	Tumor	[0, 1]	FTI	[0.0020, 0.642]
Class	{1,2,3}				

# 案例1：甲状腺疾病分类问题



```
thyroid_ctest.m  x  +
1  load thyroid.dat
2
3  %% 数据的预处理
4  id = randperm(7200);
5  thyroid = thyroid(id,:);
6  n8 = round(length(thyroid)*0.8);
7  Train = thyroid(1:n8,1:21)';
8  Test = thyroid(n8+1:end,1:21)';
9  T_train = thyroid(1:n8,22)';
10 T_test = thyroid(n8+1:end,22)';
11 [input,PS] = mapminmax(Train);
12
13 %% 类别编码
14 output = zeros(n8,length(unique(T_train)));
15 for i = 1:n8
16     output(i,T_train(i)) = 1;
17 end
18 %net = fitnet([10,5],'traingdx');
19 net = fitnet([10,5],'trainbr');
20 net.trainparam.show = 50; % 显示中间结果的周期
21 net.trainparam.epochs = 1000; %最大迭代次数（学习次数）
22 net.trainparam.goal = 0.000001; %神经网络训练的目标误差
23 net.trainParam.lr = 0.1; %学习速率（Learning rate）
24 net.divideFcn = '';
```

```
25 %% 开始训练
26 [net,tr] = train(net,input,output');
27
28 %% 接下来进行测试
29 testInput = mapminmax('apply',Test,PS); %测试数据归一化
30 %其中net为训练后得到的网络，返回的Y为预测测试数据结果
31 Y = sim(net,testInput);
32
33 %% 统计识别正确率
34 [s1,s2] = size(Y);
35 hitNum = 0;
36 for i = 1:s2
37     [m,Index] = max(Y(:,i));
38     if Index == T_test(i)
39         hitNum = hitNum + 1;
40     end
41 end
42 fprintf(' 识别率是: %3.3f.\n',100 * hitNum / s2 )
```

>> thyroid\_ctest

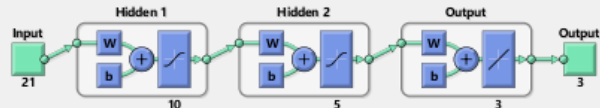
识别率是: 98.056。 %贝叶斯正则化方法

识别率是: 94.583。 %自适应lr动量梯度下降法

# 案例1：甲状腺疾病分类问题

Neural Network Training (nntraintool)

### Neural Network



Input: 21, Hidden 1: 10, Hidden 2: 5, Output: 3

### Algorithms

Training: Bayesian Regularization (trainbr)  
Performance: Mean Squared Error (mse)  
Calculations: MEX

### Progress

Epoch:	0	1000 iterations	1000
Time:	0:00:59		
Performance:	0.738	0.000295	1.00e-06
Gradient:	1.06	7.60e-06	1.00e-07
Mu:	0.00500	50.0	1.00e+10
Effective # Param:	293	212	0.00
Sum Squared Param:	60.5	1.82e+04	0.00
Validation Checks:	0	0	0

### Plots

Performance

(plotperform)

Training State

(plottrainstate)

Error Histogram

(ploterrhist)

Regression

(plotregression)

Fit

(plotfit)

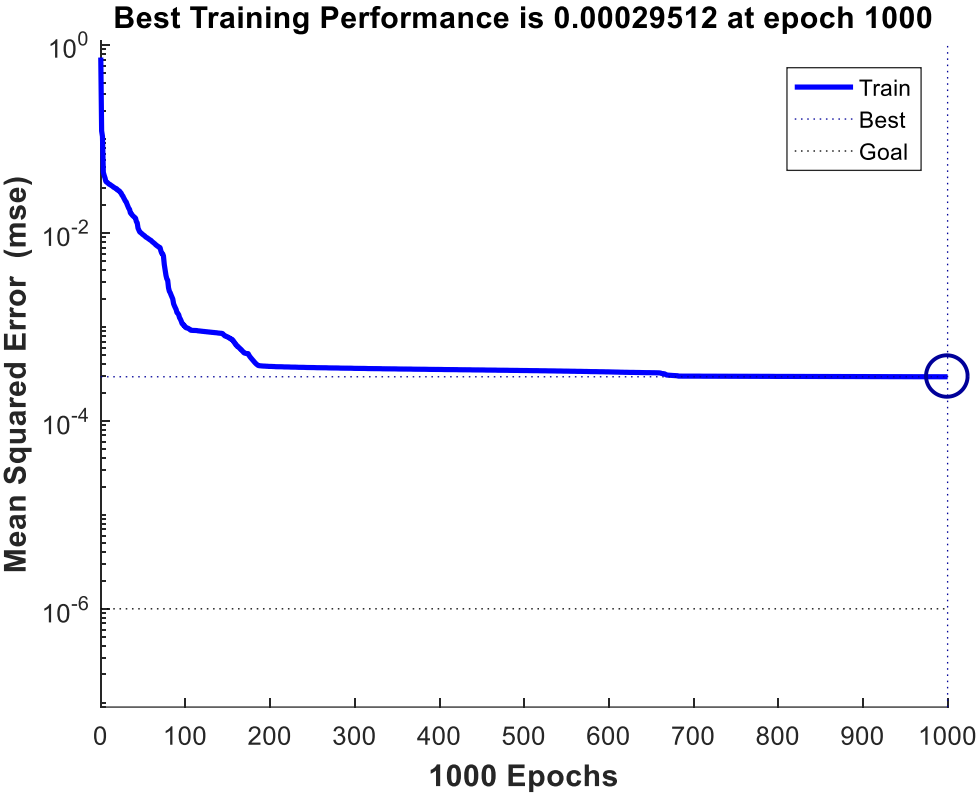
Plot Interval:

50 epochs

Maximum epoch reached.

Stop Training

Cancel



# 分类问题的一般化代码



```
BPtrain_test.m  +
1  function BPtrain_test(data)
2  % data最后一列为类别标签
3  %% 数据的预处理
4  n = size(data,1);
5  id = randperm(n);
6  data = data(id,:);
7  n8 = round(n*0.8);
8  Train = data(1:n8,1:end-1)';
9  Test = data(n8+1:end,1:end-1)';
10 T_train = data(1:n8,end)';
11 T_test = data(n8+1:end,end)';
12 [input,PS] = mapminmax(Train);
13
14 %% 类别编码
15 output = zeros(n8,length(unique(T_train)));
16 for i = 1:n8
17     output(i,T_train(i)) = 1;
18 end
19
20 %net = fitnet([10,5],'traingdx');
21 net = fitnet([10,5],'trainbr');
22 net.trainparam.show = 50; % 显示中间结果的周期
23 net.trainparam.epochs = 1000; %最大迭代次数（学习次数）
24 net.trainparam.goal = 0.000001; %神经网络训练的目标误差
25 net.trainParam.lr = 0.1; %学习速率（Learning rate）
26 %该函数把样本数据三分为训练集、验证集和测试集，默认比例是6:2:2，清除该属性再训练：
27 net.divideFcn = '';
```

```
28 %% 开始训练
29 net = train(net,input,output');
30
31 %% 接下来进行测试
32 testInput = mapminmax('apply',Test,PS); %测试数据归一化
33 %其中net为训练后得到的网络，返回的Y为预测测试数据结果
34 Y = sim(net,testInput);
35
36 %% 统计识别正确率
37 [~,s2] = size(Y);
38 hitNum = 0;
39 class = zeros(1,s2);
40 for i = 1:s2
41     [~,Index] = max(Y(:,i));
42     if Index == T_test(i)
43         hitNum = hitNum + 1;
44         class(i) = T_test(i);
45     end
46 end
47 fprintf('识别率是: %3.3f.\n',100 * hitNum / s2);
48 tabulate(T_test)
49 tabulate(class) % 0表示误识别数据类别
```

# 案例2：页面模块分类数据

Page Blocks Classification data set			
Type	Classification	Origin	Real world
Features	10	(Real / Integer / Nominal)	(4 / 6 / 0)
Instances	5472	Classes	5
Missing values?			No

```
>> BPtrain_test(page_blocks)
识别率是：97.078。
```

Value	Count	Percent
1	964	88.04%
2	78	7.12%
3	6	0.55%
4	17	1.55%
5	30	2.74%

Value	Count	Percent
0	32	2.92%
1	949	86.67%
2	74	6.76%
3	6	0.55%
4	16	1.46%
5	18	1.64%

可见贝叶斯正则化方法比自适应1r动量梯度下降法的识别率较高。

Attribute	Domain
Height	[1, 804]
Lenght	[1, 553]
Area	[7, 143993]
Eccen	[0.0070, 537.0]
P_black	[0.052, 1.0]
P_and	[0.062, 1.0]
Mean_tr	[1.0, 4955.0]
Blackpix	[1, 33017]
Blackand	[7, 46133]
Wb_trans	[1, 3212]
Class	{1, 2, 3, 4, 5}

```
>> load page-blocks.data
>> BPtrain_test(page_blocks)
识别率是：93.242。
```

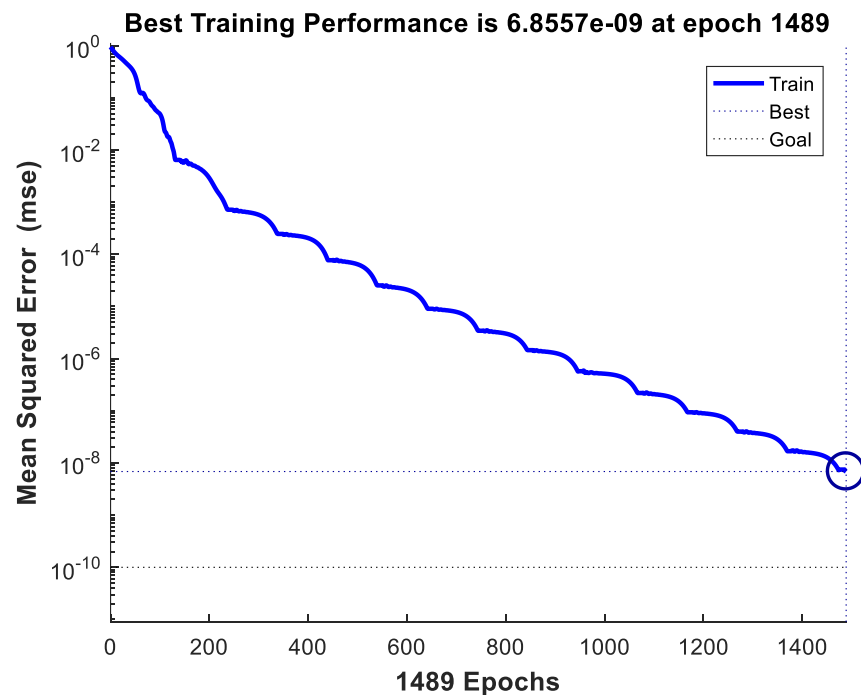
Value	Count	Percent
1	978	89.32%
2	71	6.48%
3	3	0.27%
4	15	1.37%
5	28	2.56%

Value	Count	Percent
0	74	6.76%
1	974	88.95%
2	44	4.02%
3	1	0.09%
5	2	0.18%

## 案例3：神经网络预测

```
%% 调高成绩预测  
hjdata = xlsread('highjump.xlsx');  
Train = hjdata(1:end-1,3:end)'; %训练数据  
T_train = hjdata(1:end-1,2)'; %目标数据  
Test = hjdata(end,3:end)'; %待预测数据  
[input,PS] = mapminmax(Train,-1,1); %训练数据归一化  
[Output,Pso] = mapminmax(T_train,-1,1); %训练数据目标归一化  
net = fitnet([8,4],'traingdx'); %创建前馈BP神经网络  
net.trainParam.epochs = 2000; %设置训练次数  
net.trainParam.goal = 1e-10; %设置收敛误差要求  
net.divideFcn = "";  
[net,tr] = train(net,input,Output); %训练网络
```

```
testInput = mapminmax('apply',Test,PS);  
out = sim(net,testInput); %预测输出  
preOut = mapminmax('reverse',out,Pso) %反归一化  
preOut =  
2.2249
```





---

# 感谢聆听

---