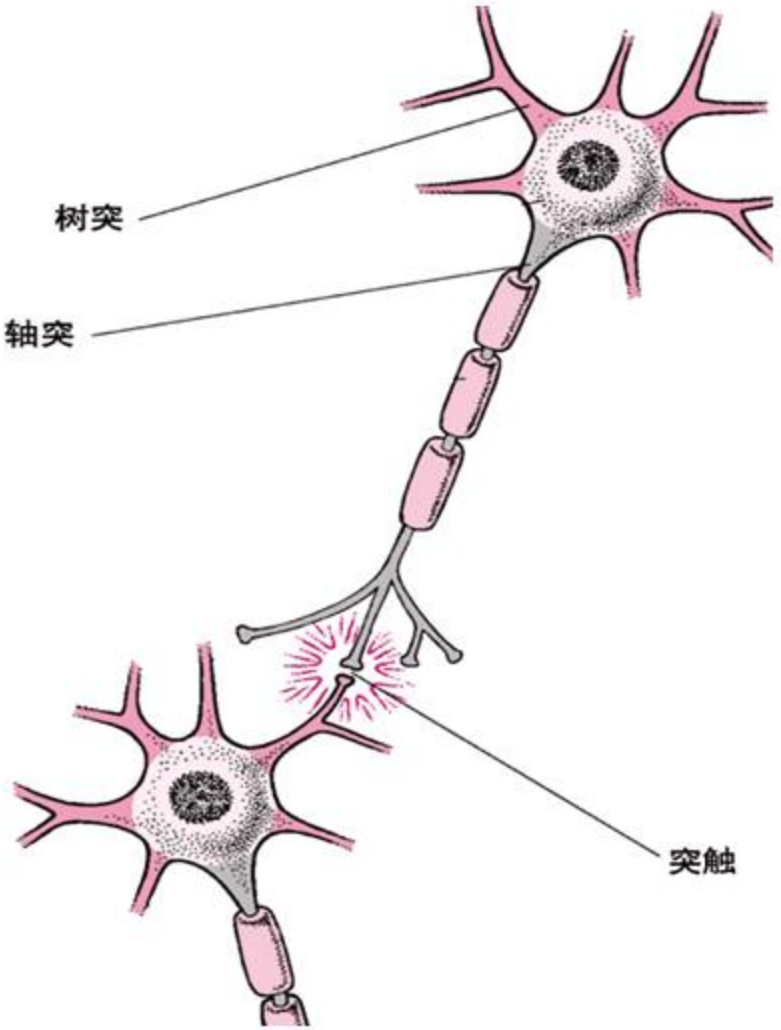
神经网络分类器

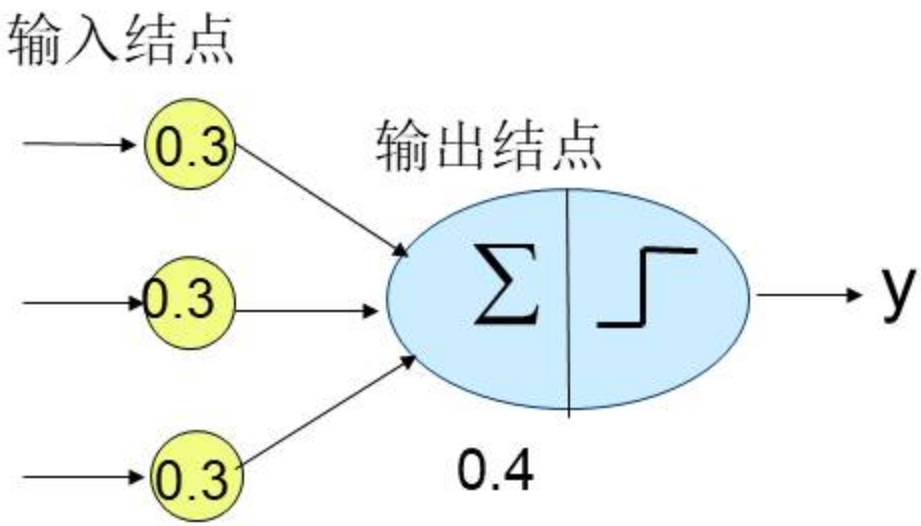
人工神经网络

- 大脑是如何学习的？
  - 神经元传导
  - 神经键改变
- 为什么要模拟人脑？
  - 快速、智能
  - 稳定
- 我们如何模拟人脑的神经元？
- https://blog.csdn.net/zengxiantao1994/article/details/72787849



感知机

x_1	x_2	x_3	Y
1	0	0	−1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	−1
0	1	0	−1
0	1	1	1
0	0	0	−1



$$\hat{y} = \begin{cases} 1 \ , 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 > 0 \\ -1, 0.3x_1 + 0.3x_2 + 0.3x_3 - 0.4 < 0 \end{cases}$$

感知机

上一个例子的模型：

$$\hat{y}=\begin{cases}1 & ,0.3x_1+0.3x_2+0.3x_3-0.4>0\\ -1 & ,0.3x_1+0.3x_2+0.3x_3-0.4<0\end{cases}$$

化为一般的形式：

$$\hat{y}=sign(w_1x_1+w_2x_2+\cdots+w_dx_d-t)$$

$$\hat{y}=sign(\sum_i w_{ij}x_i-t)$$

感知机

初始：用随机值初始化权值向量$w^{(0)}$

循环：直到输出和实例一致

{

对每个训练样例(x_i,y_i)，按照$w^{(k)}$计算输出值$y_i^{(k)}$;

for每个权值$w^{(k)}$

更新权值$w^{(k+1)}= w^{(k)}+\lambda(y_i-y_i^{(k)})x_i$;

}

感知机

更新权值：$w^{(k+1)}= w^{(k)}+\lambda(y_i-y_i^{(k)})x_i$

- 如果 $y_i=y_i^{(k)}$， 权值w不变；
- 如果 $y_i=1$（实际类），$y_i^{(k)}=-1$，预测误差为2；

  为了补偿这个误差，需要提高正输入链的权值，降低负输入链的；
- 如果 $y_i=-1$（实际类），$y_i^{(k)}=1$，预测误差为-2；

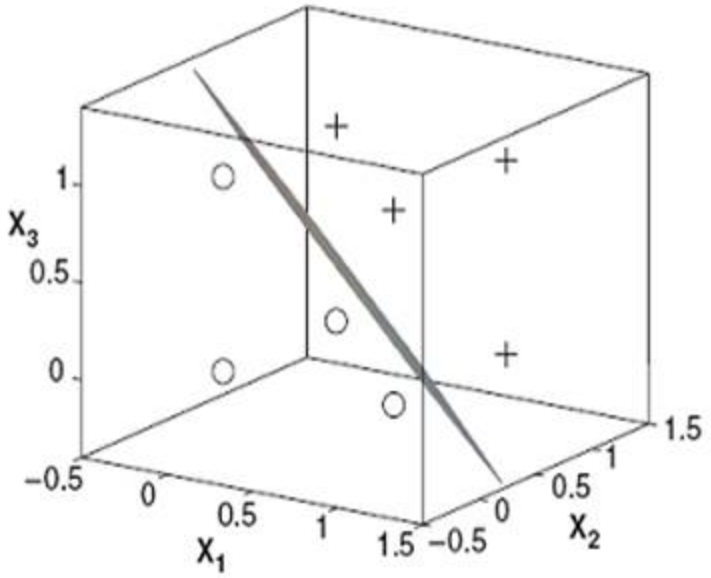
  为了补偿这个误差，需要降低正输入链的权值，提高负输入链的；

感知机

F. Rosenblatt证明了对"线性可分"的问题

感知机通过有限次训练就能学会正确的行为

x_1	x_2	x_3	Y
1	0	0	−1
1	0	1	1
1	1	0	1
1	1	1	1
0	0	1	−1
0	1	0	−1
0	1	1	1
0	0	0	−1

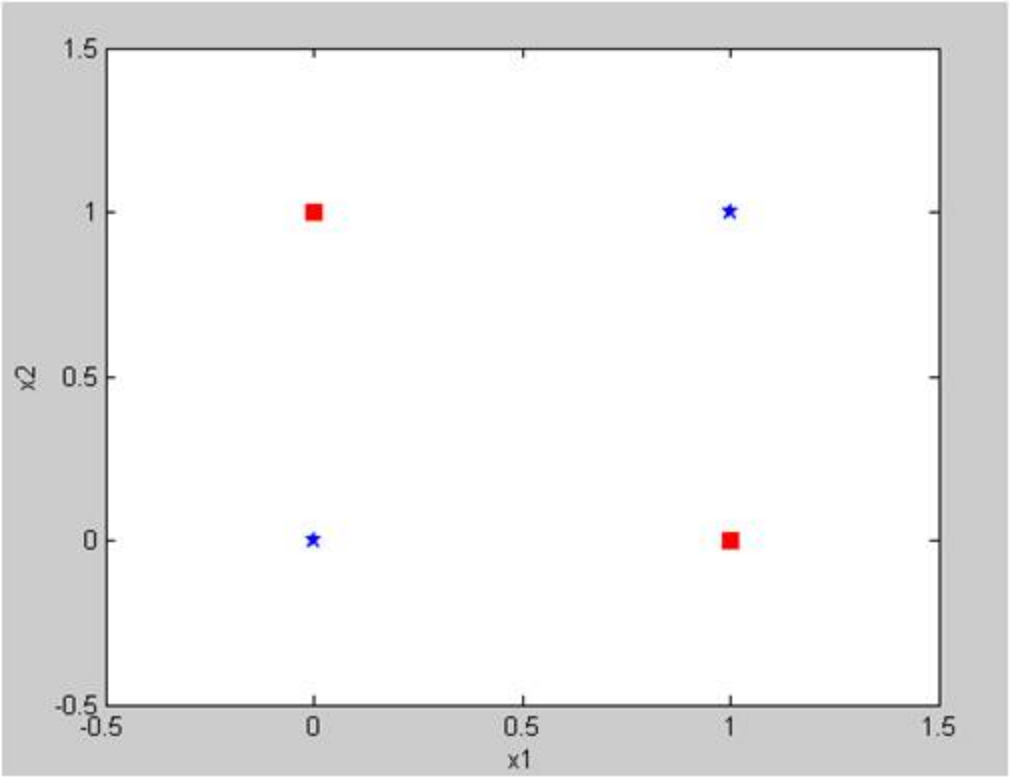


感知机

- Marvin Minsky, *Perceptrons*

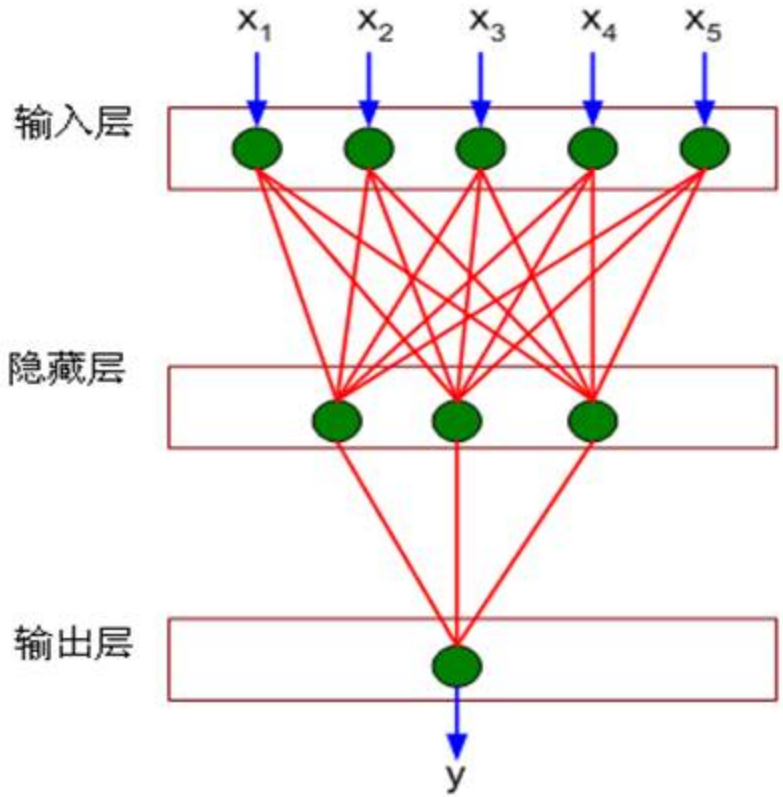
  证明了感知器无法执行“异或问题”

x_1	x_2	Y
0	0	−1
1	0	1
0	1	1
1	1	−1



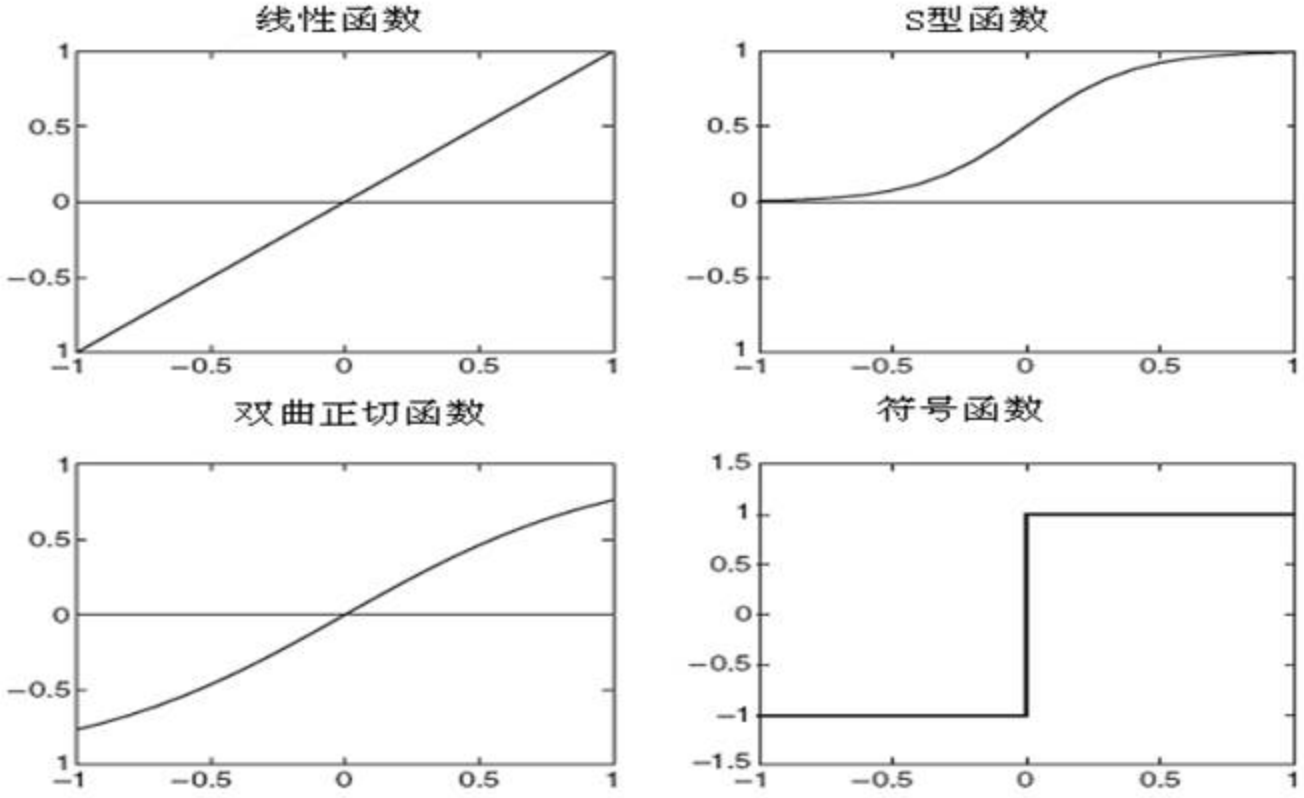
多层人工神经网络

- 人工神经网络比感知机模型复杂
  - 输入层和输出层之间包含隐藏层



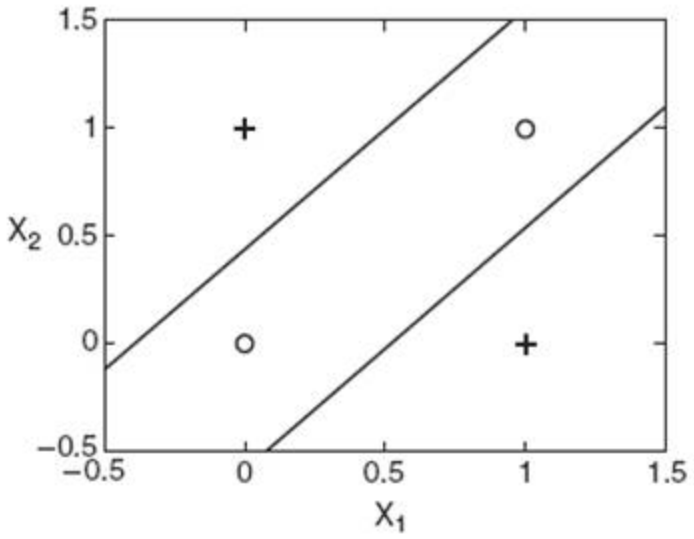
多层人工神经网络

- 人工神经网络比感知机模型复杂
  - 激活函数可以是多种函数

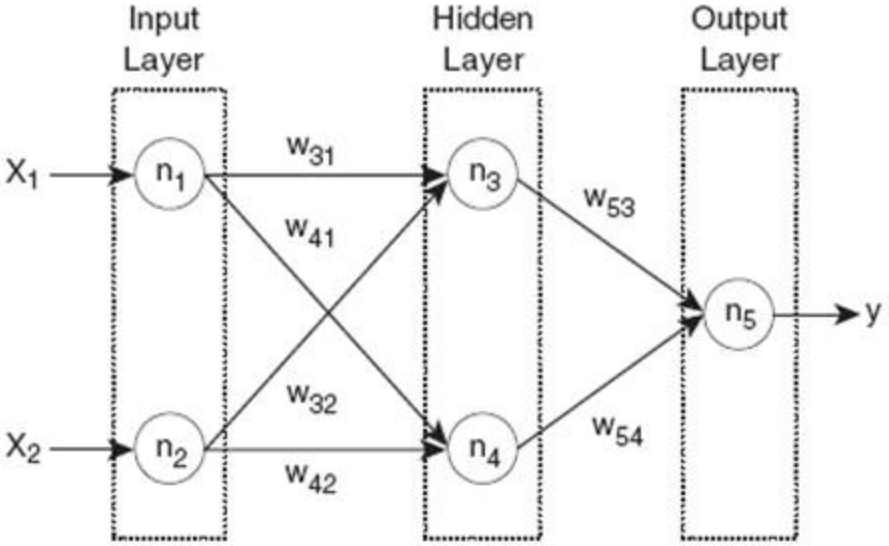


多层人工神经网络

- 例：用两层前馈神经网络解决异或问题



(a) 决策边界



(b) 神经网络拓扑结构

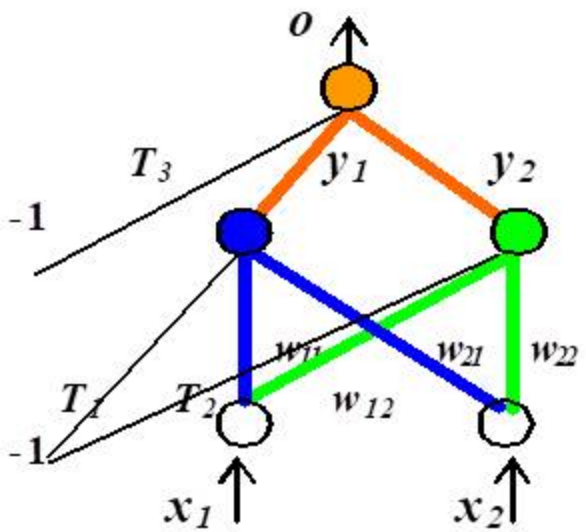
多层感知器

用两计算层感知器解决“异或”问题。

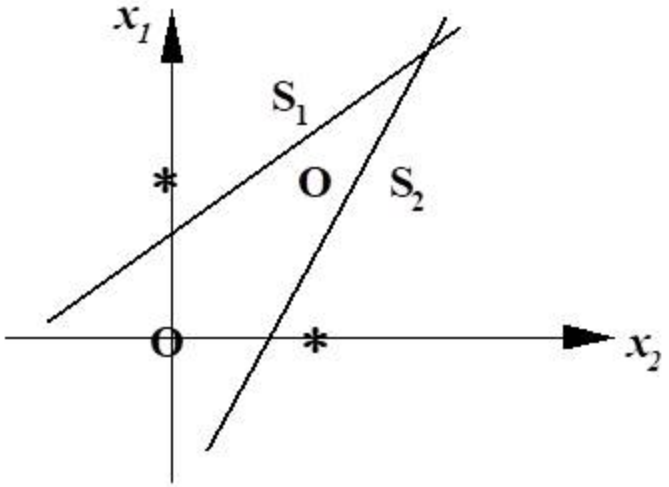
“异或”的真值表

x1	x2	y1	y2	o
0	0	1		
0	1	1		
1	0	0		
1	1	1		

双层感知器



“异或”问题分类



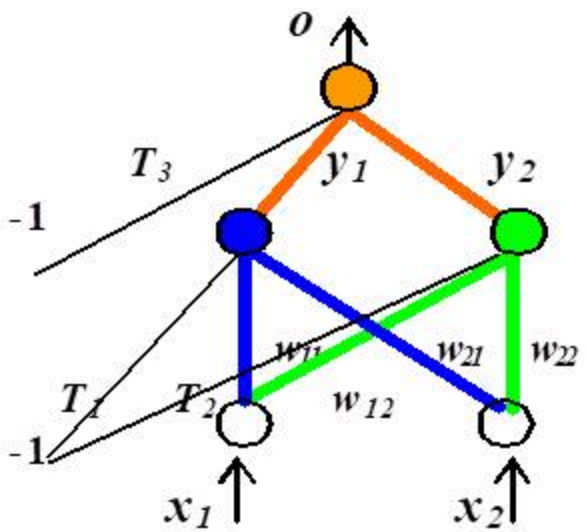
多层感知器

用两计算层感知器解决“异或”问题

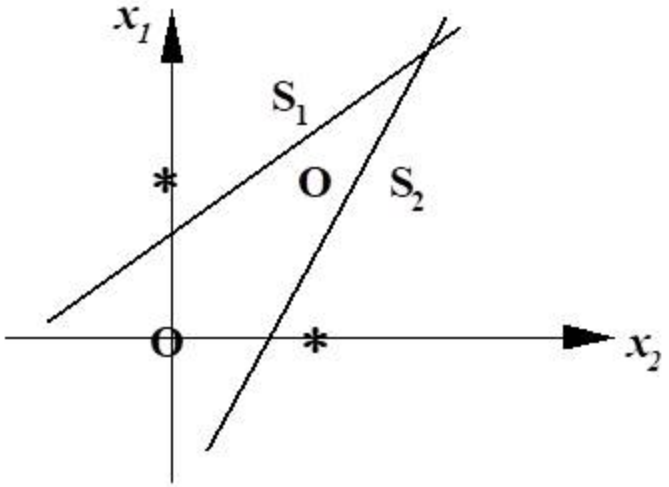
“异或”的真值表

x1	x2	y1	y2	o
0	0		1	
0	1		0	
1	0		1	
1	1		1	

双层感知器



“异或”问题分类



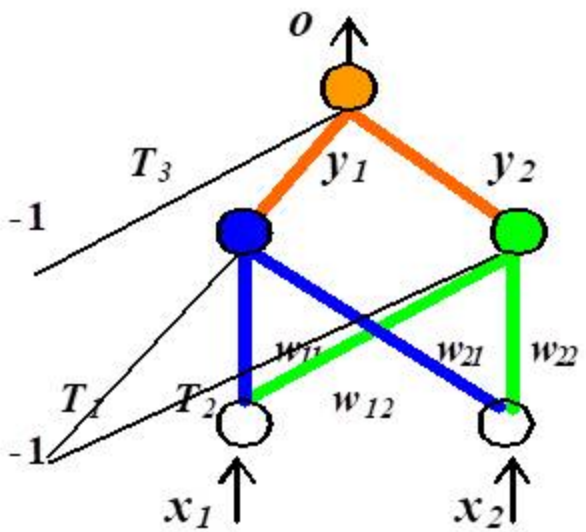
多层感知器

用两计算层感知器解决“异或”问题。

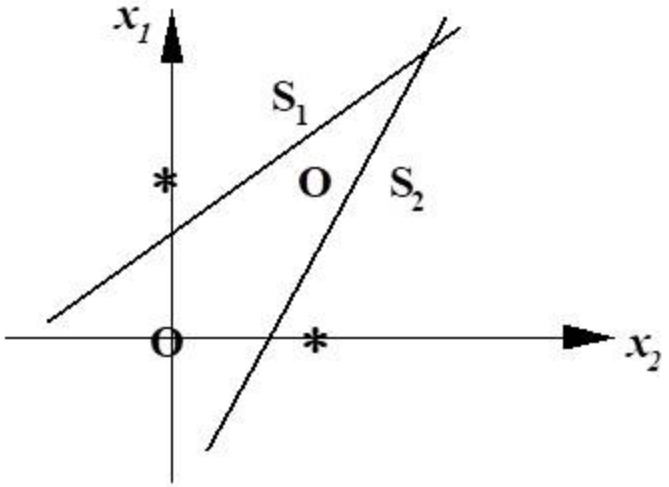
“异或”的真值表

x1	x2	y1	y2	o
0	0	1	1	
0	1	1	0	
1	0	0	1	
1	1	1	1	

双层感知器



“异或”问题分类



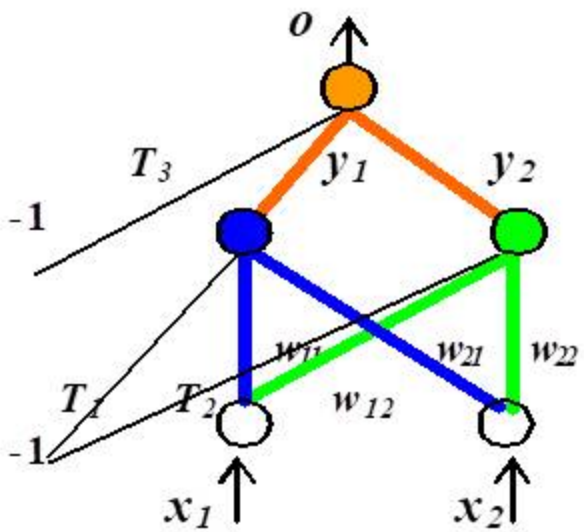
多层感知器

例四 用两计算层感知器解决“异或”问题。

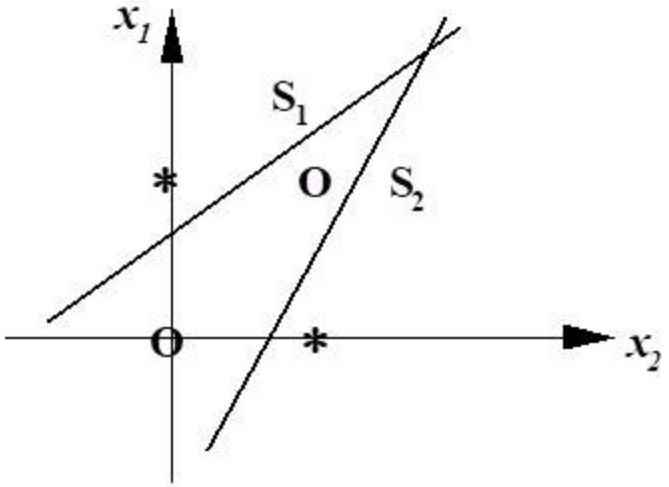
“异或”的真值表

x1	x2		o
0	0		0
0	1		1
1	0		1
1	1		0

双层感知器

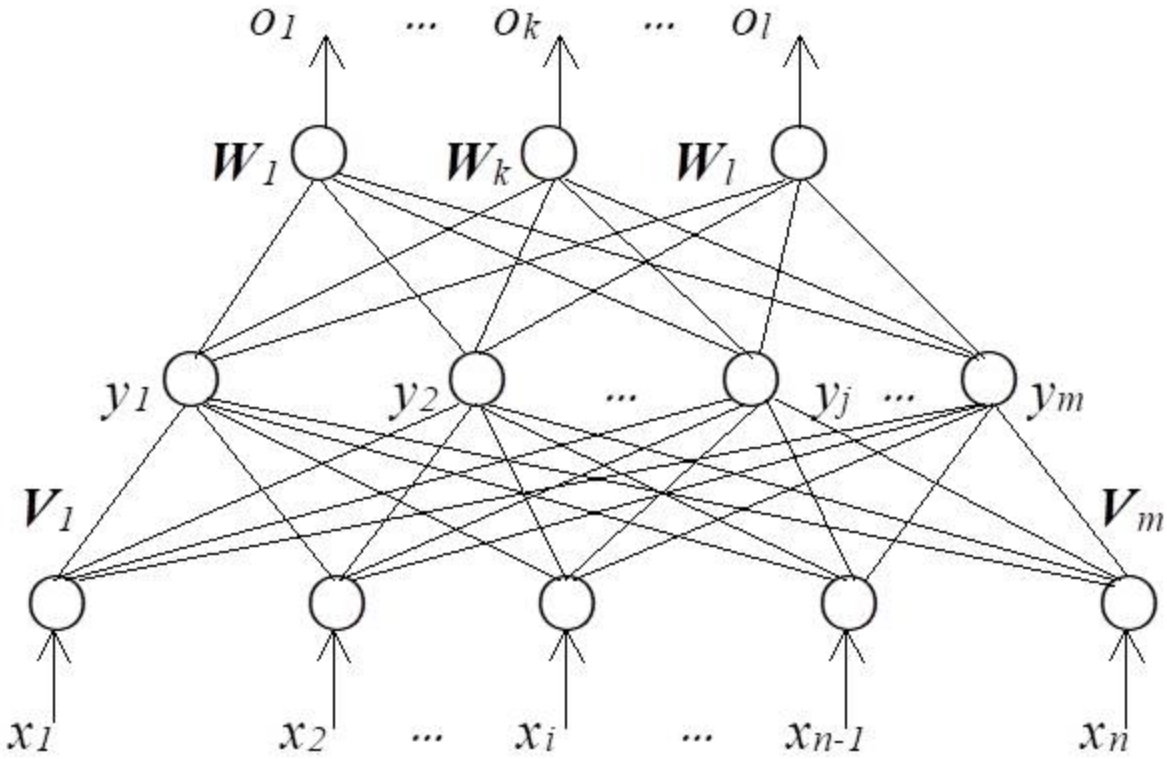


“异或”问题分类



误差反向传播（BP）网络

基于**BP**算法的多层前馈网络模型



- 激活函数
  - 必须处处可导
    - 一般都使用S型函数
- 使用S型激活函数时BP网络输入与输出关系
  - 输入 $net = x_1 w_1 + x_2 w_2 + \ldots + x_n w_n$
  - 输出 $y = \mathrm{f}(net) = \dfrac{1}{1 + \mathrm{e}^{-net}}$

BP网络的标准学习算法

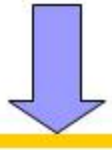
- 学习的过程：
  - 神经网络在外界输入样本的刺激下不断改变网络的连接权值,以使网络的输出不断地接近期望的输出。
- 学习的本质：
  - 对各连接权值的动态调整
- 学习规则：
  - 权值调整规则，即在学习过程中网络中各神经元的连接权变化所依据的一定的调整规则。

- 学习的类型：有监督式学习
- 核心思想：
  - 将输出误差*以某种形式*通过隐层向输入层逐层反传

将误差分摊给各层的所有单元———各层单元的误差信号 → 修正各单元权值

- 学习的过程：
  - 信号的正向传播　　误差的反向传播

- 正向传播：
  - 输入样本－－－输入层－－－各隐层－－－输出层
- 判断是否转入反向传播阶段：
  - 若输出层的实际输出与期望的输出（教师信号）不符
- 误差反传
  - 误差以某种形式在各层表示－－－－修正各层单元的权值



- 网络输出的误差减少到可接受的程度
  进行到预先设定的学习次数为止

后向传播算法

1. 初始化权重

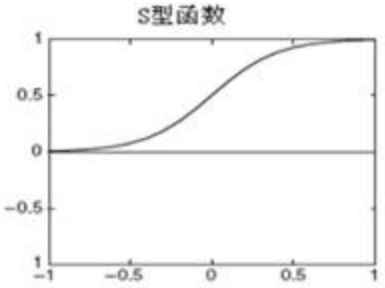
循环以下两步，直到满足条件

$$I_j = \sum_i w_{ij} x_i - t_j \qquad y_j = \frac{1}{1+e^{-I_j}}$$

1. 向前传播输入

在每个节点加权求和，再代入激活函数

$$\hat{y} = sign(\sum_i w_{ij} x_i - t_j)$$



2. 向后传播误差

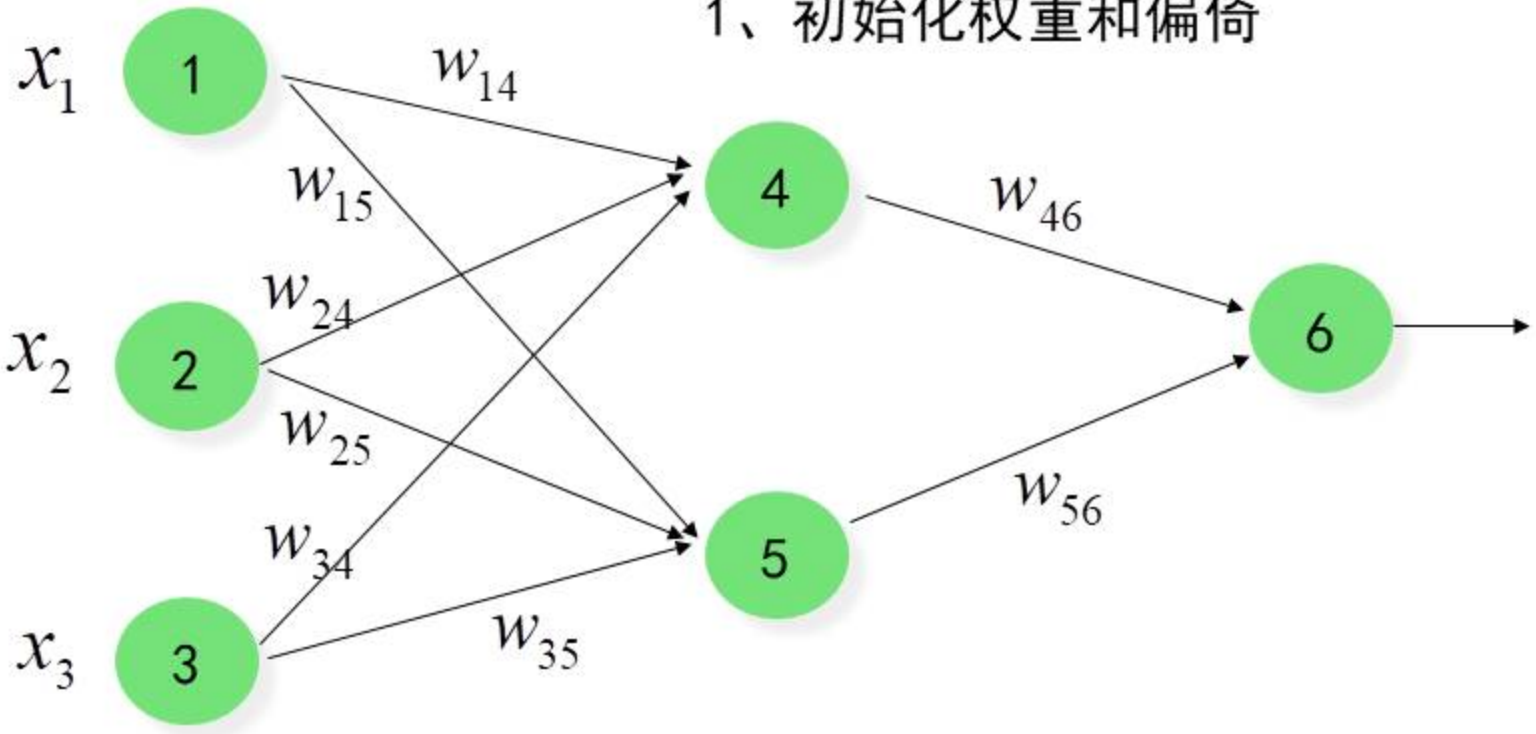
$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$w_{ij} = w_{ij} + \lambda Err_j y_i$$

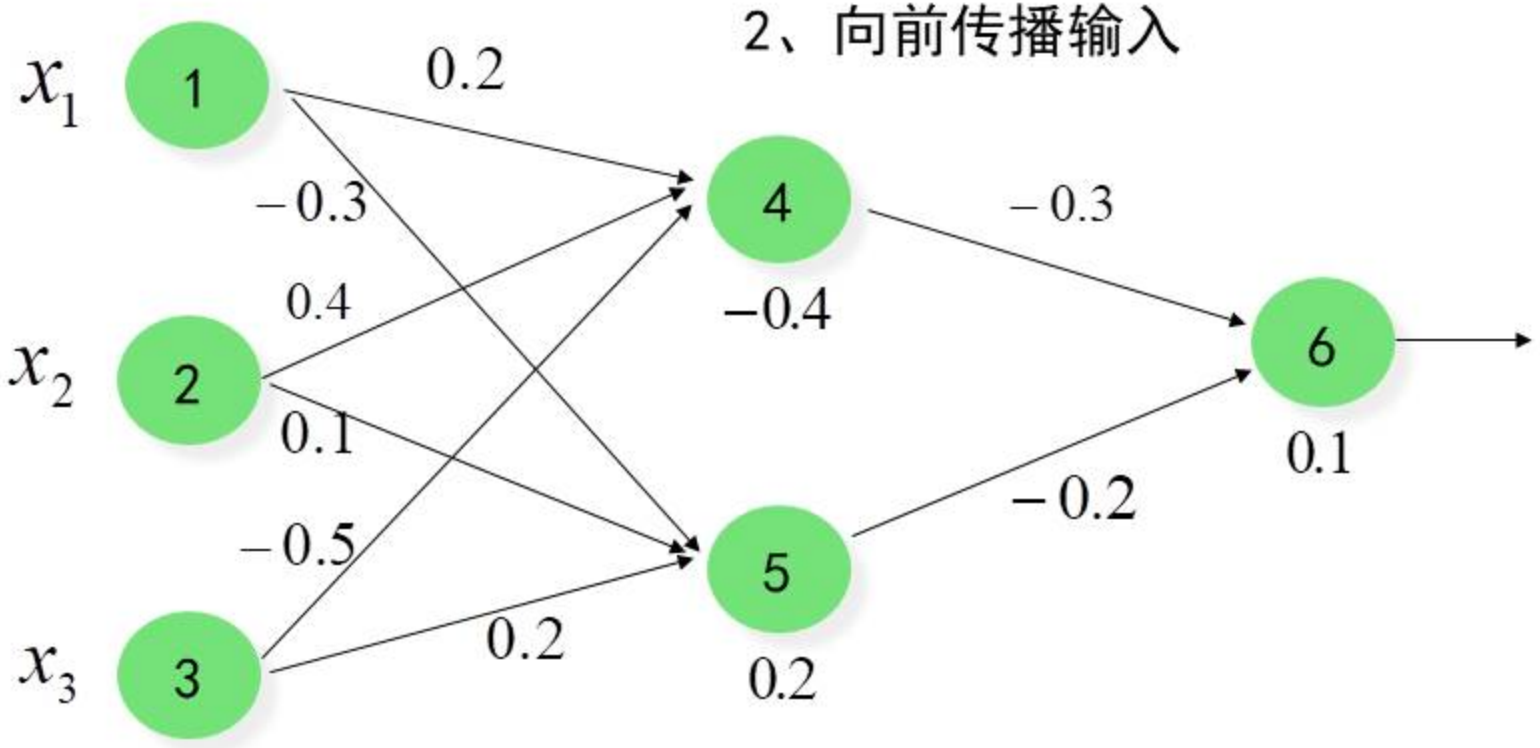
$$t_j = t_j + \lambda Err_j$$

1、初始化权重和偏倚



后向传播算法

2、向前传播输入



后向传播算法

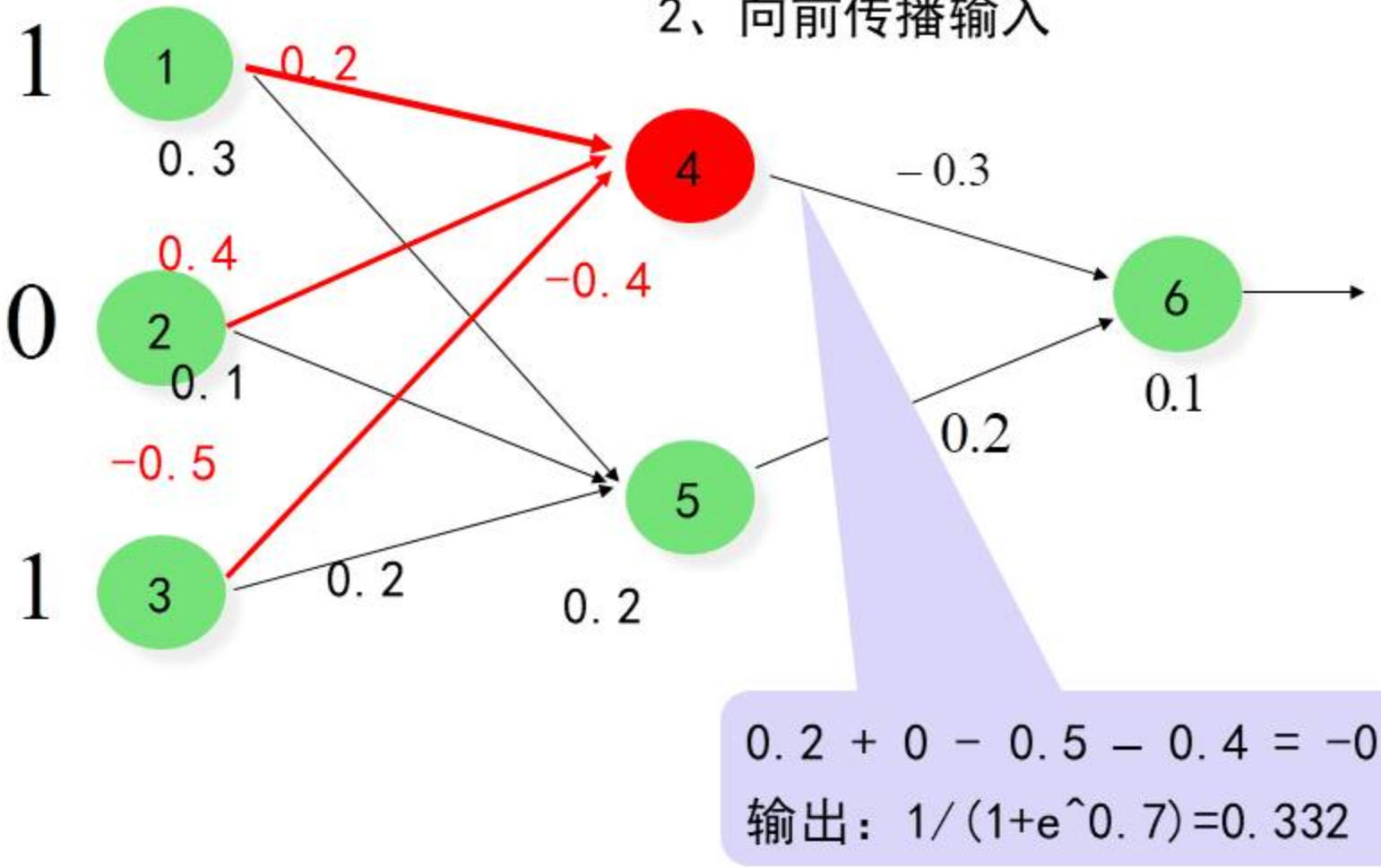
初始随机设置参数

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	−0.3	0.4	0.1	−0.5	0.2	−0.3	−0.2	−0.4	0.2	0.1

静输入和输出的表格

Unit j	*Net input,* I_j	*Output,* O_j
4	$0.2+0-0.5-0.4=-0.7$	$1/(1+e^{0.7})=0.332$
5	$-0.3+0+0.2+0.2=0.1$	$1/(1+e^{-0.1})=0.525$
6	$(-0.3)(0.332)-(0.2)(0.525)+0.1=-0.105$	$1/(1+e^{0.105})=0.474$

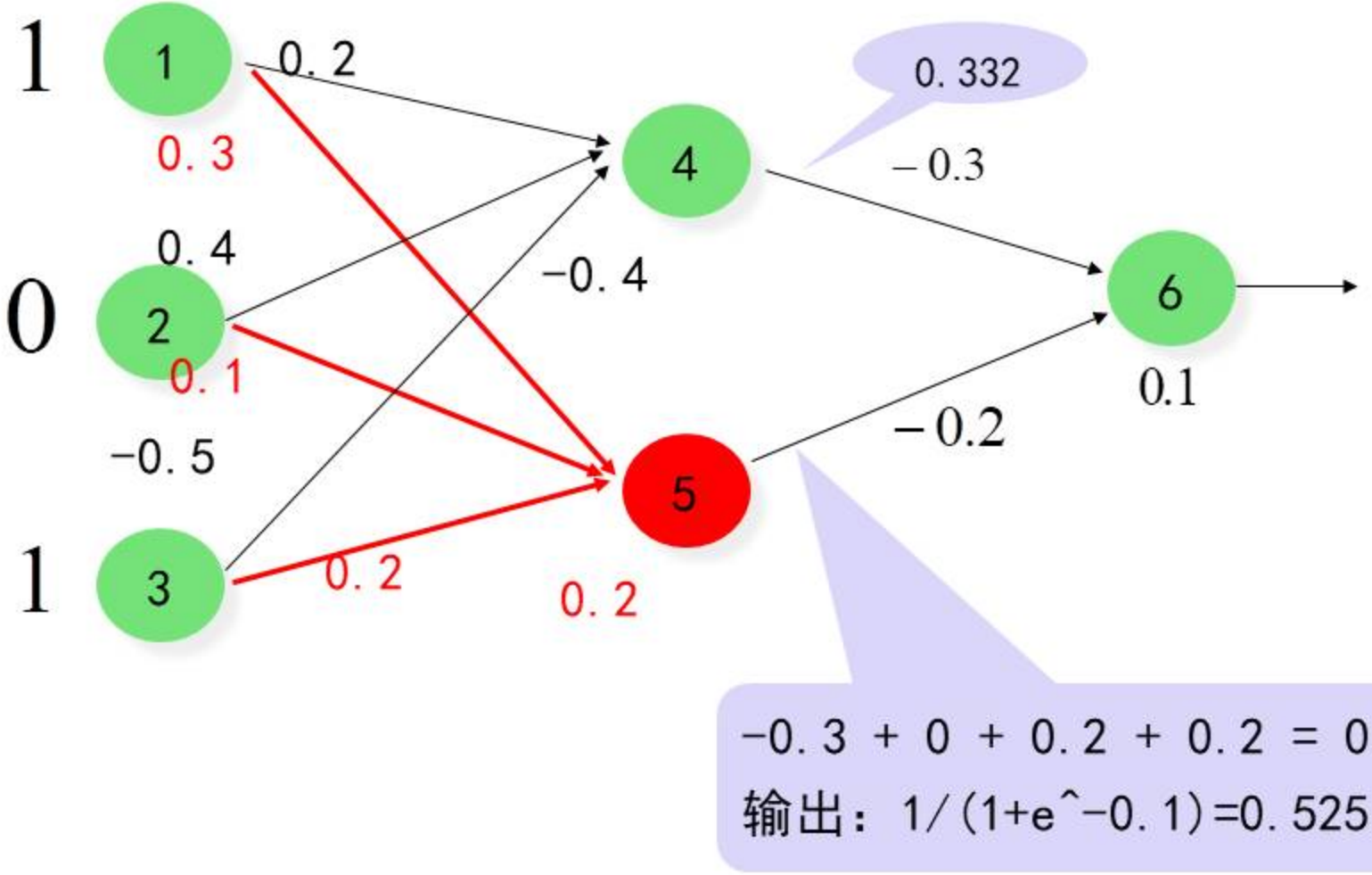
2、向前传播输入



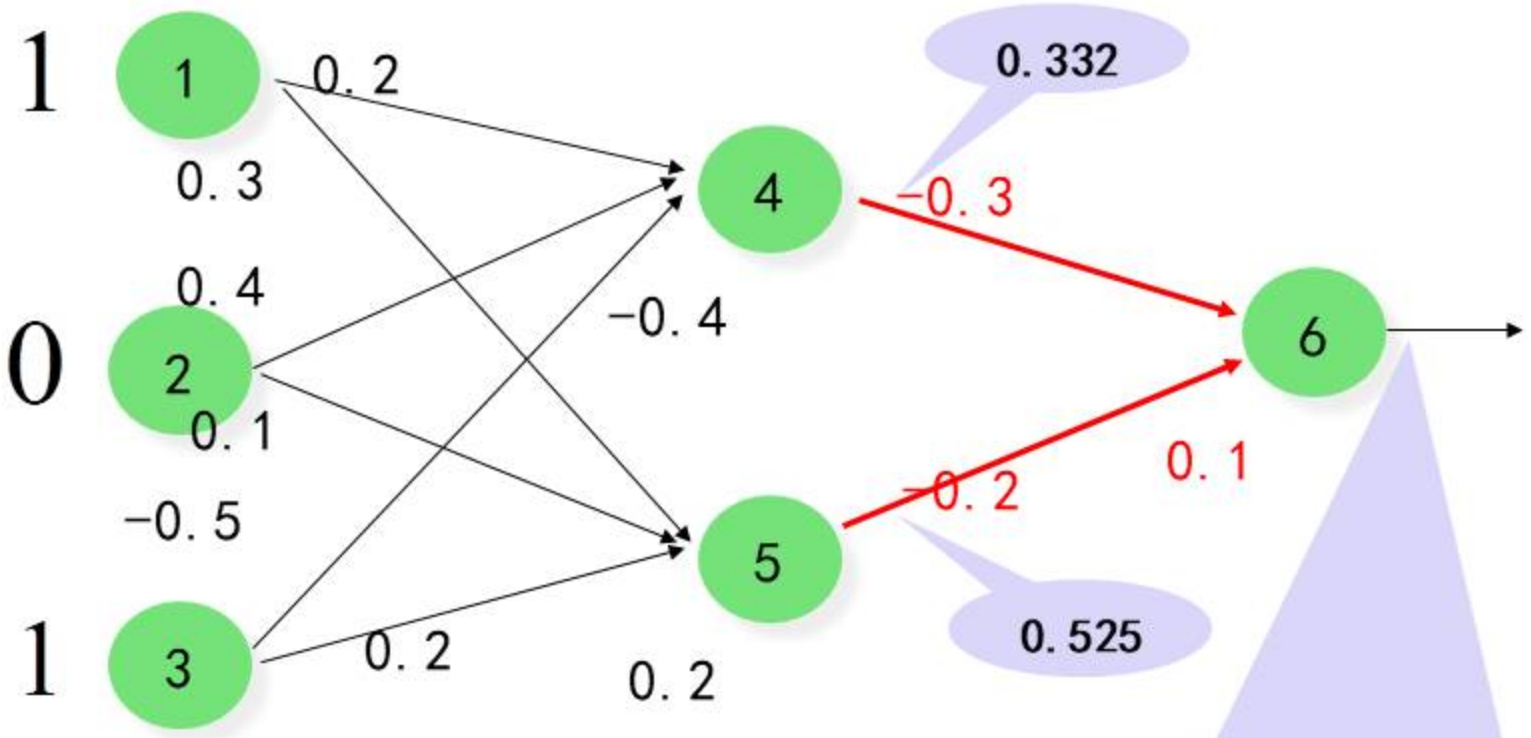
0.2 + 0 − 0.5 − 0.4 = −0.7

输出：1/(1+e^0.7)=0.332

后向传播算法


1
0
1
1
2
3
4
5
6
0.2
0.3
0.4
0.1
-0.5
-0.4
0.2
0.2
0.332
-0.3
-0.2
0.1
-0.3 + 0 + 0.2 + 0.2 = 0.1
输出：1/(1+e^-0.1)=0.525

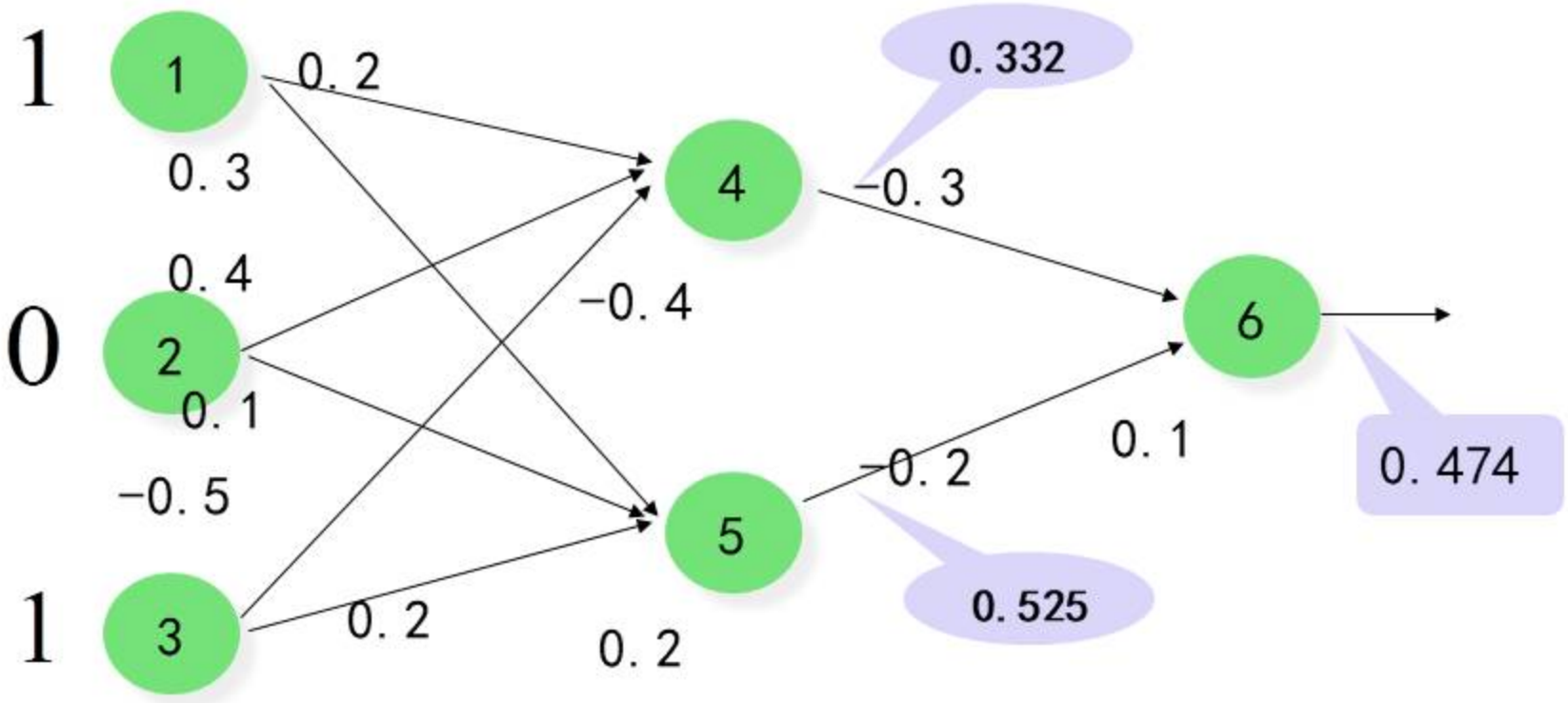
后向传播算法



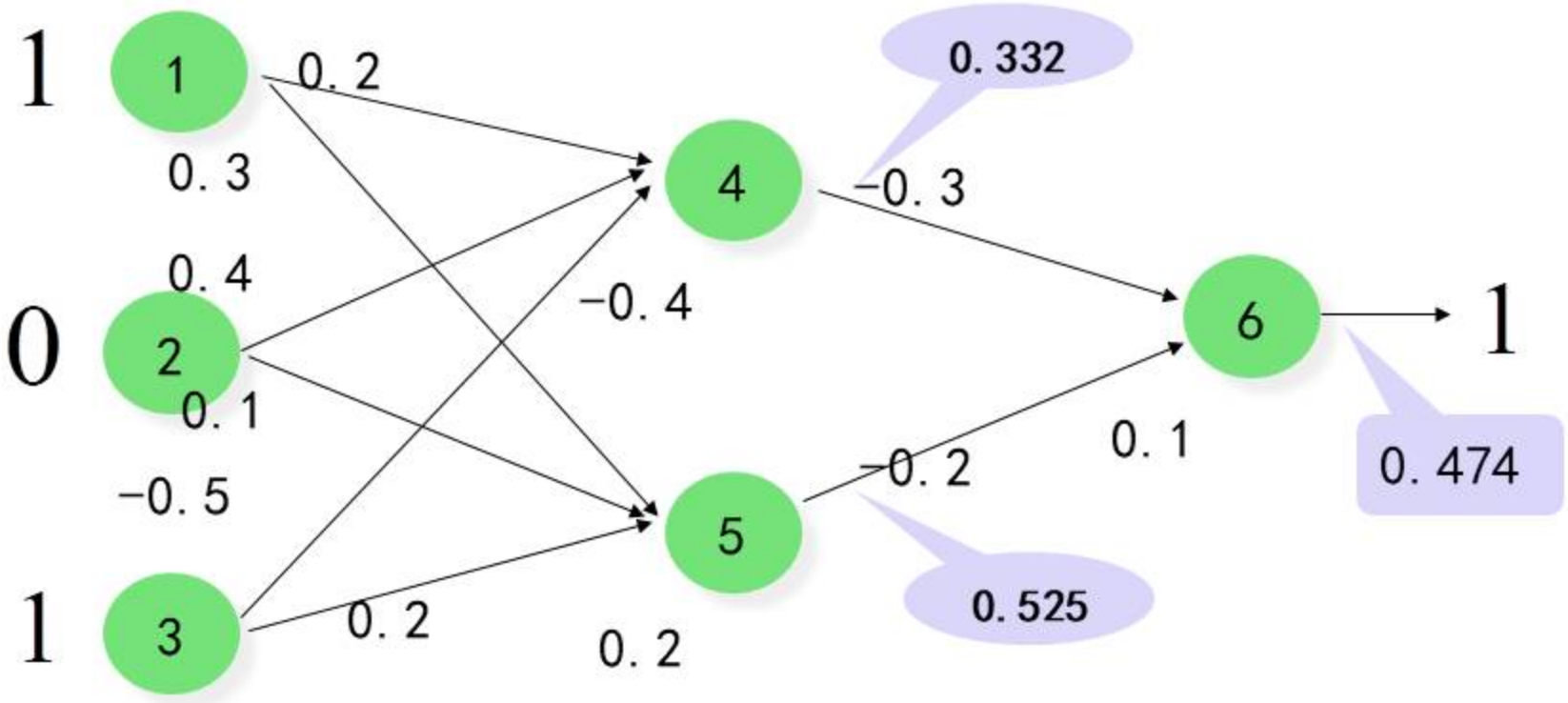
$(-0.3)(0.332)-(0.2)(0.525)+0.1=$ **-0.105**

输出：$1/(1+e^{-0.105})=0.474$

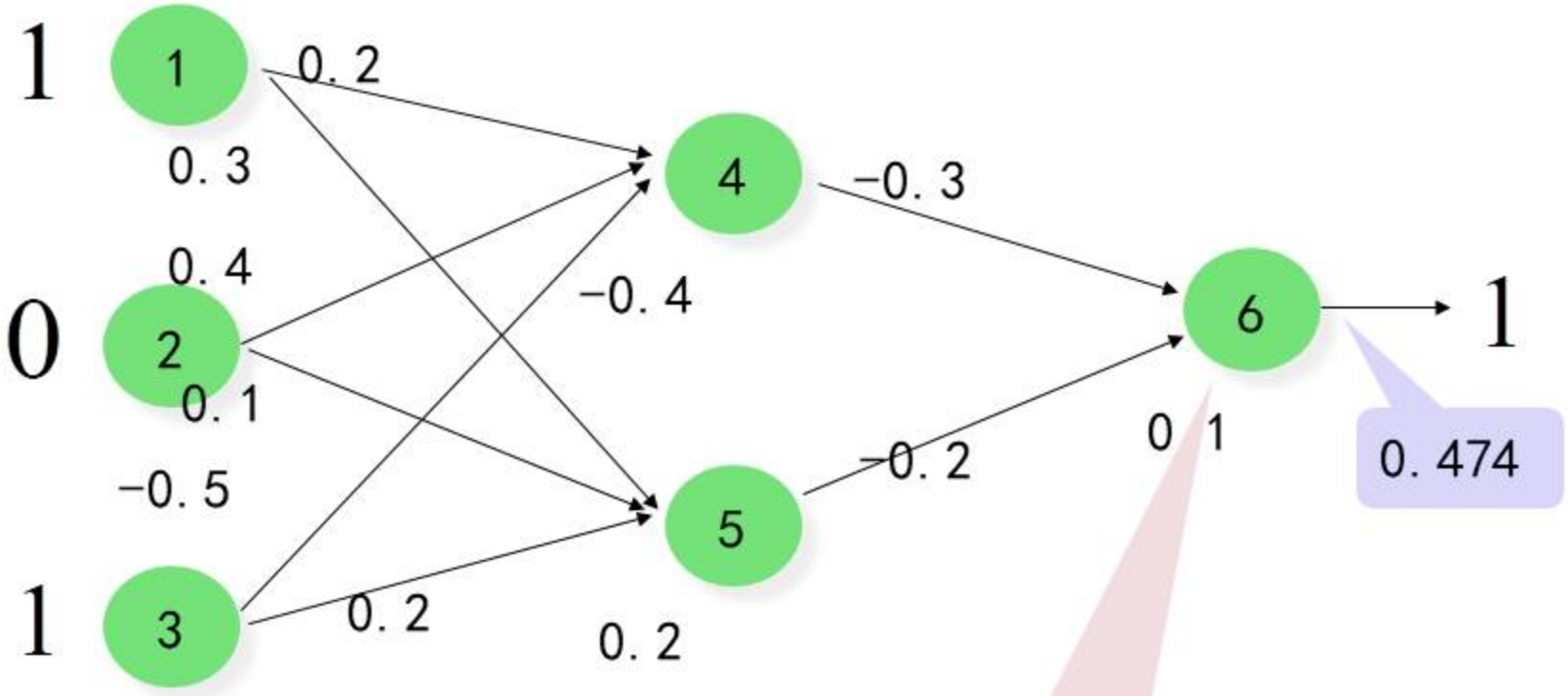
后向传播算法



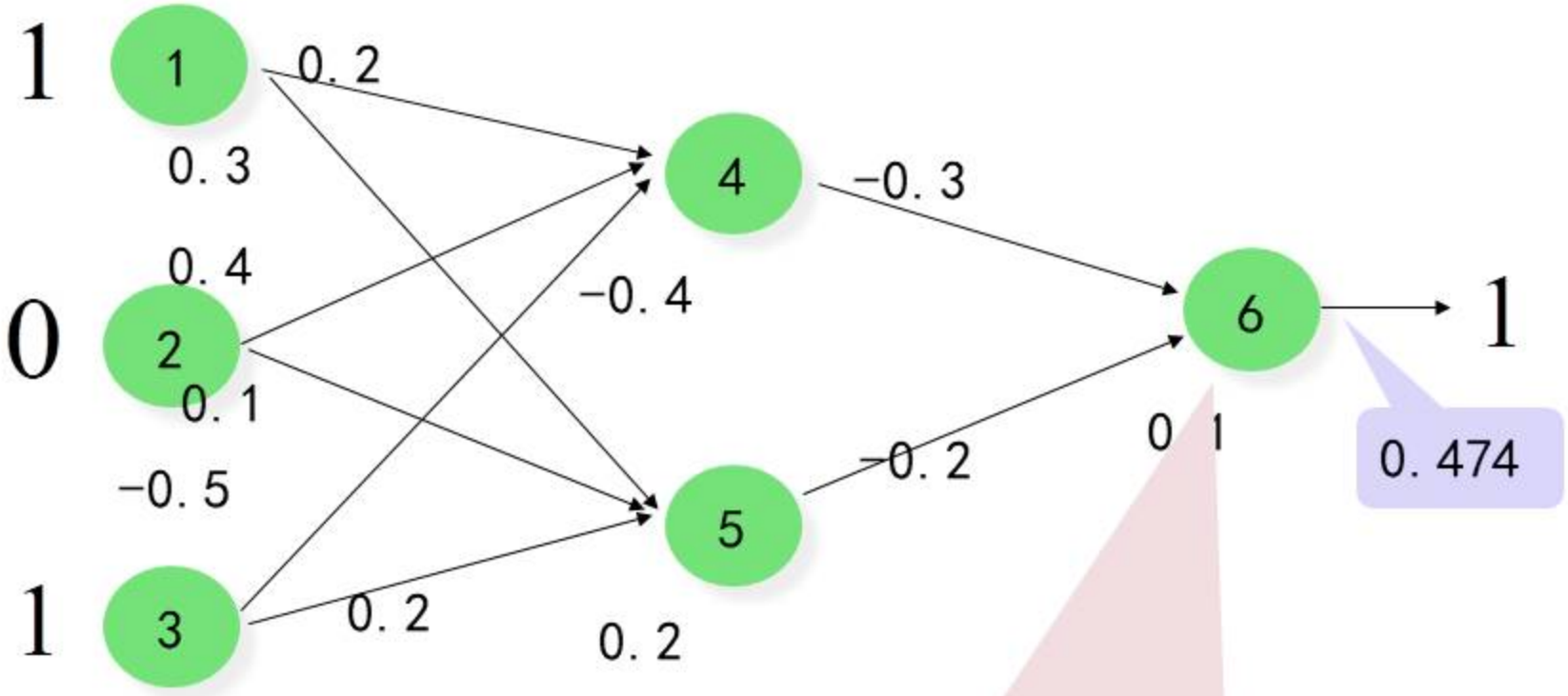
后向传播算法



后向传播算法



$$Err_j = O_j(1-O_j)(T_j-O_j)$$

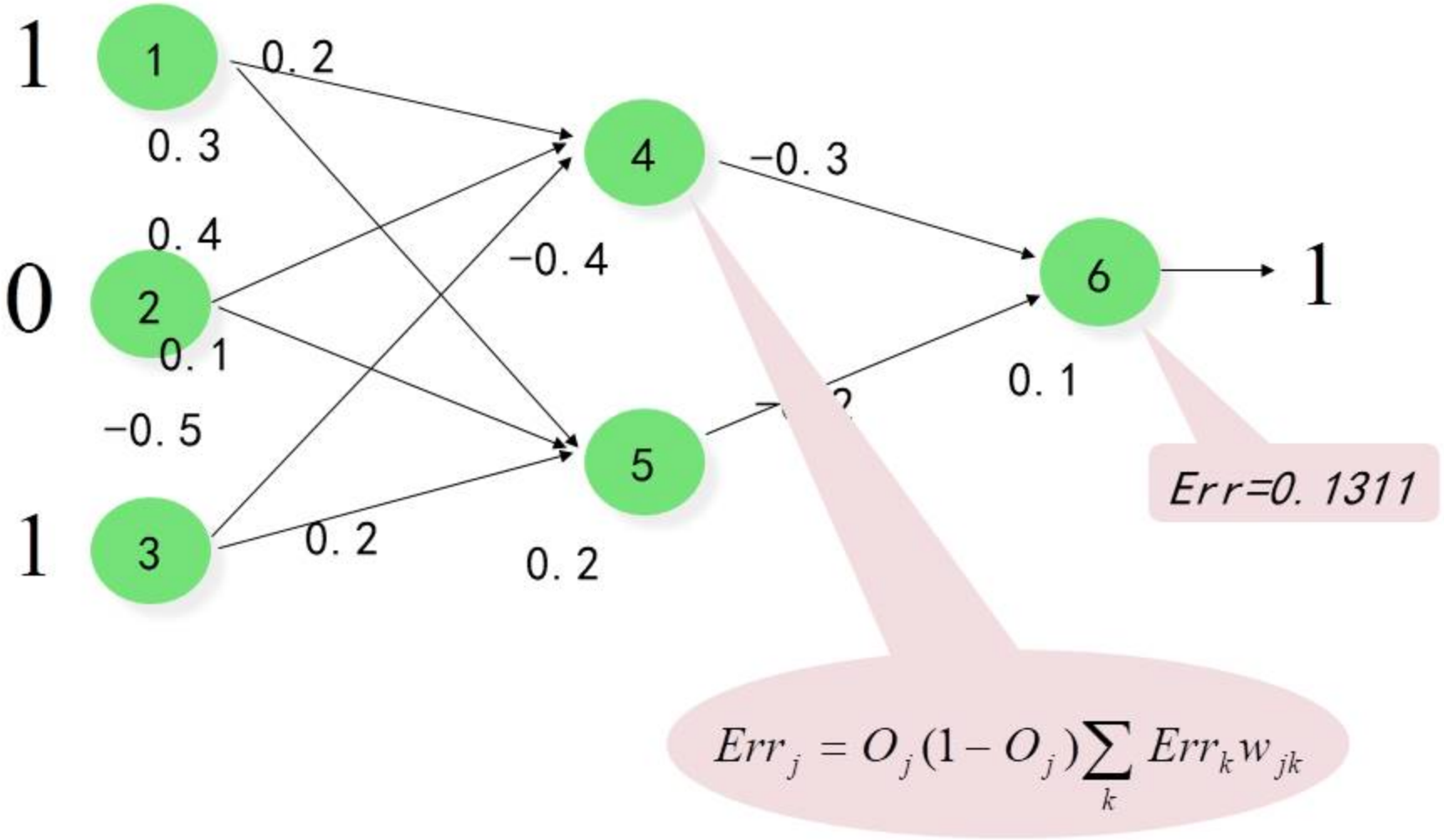


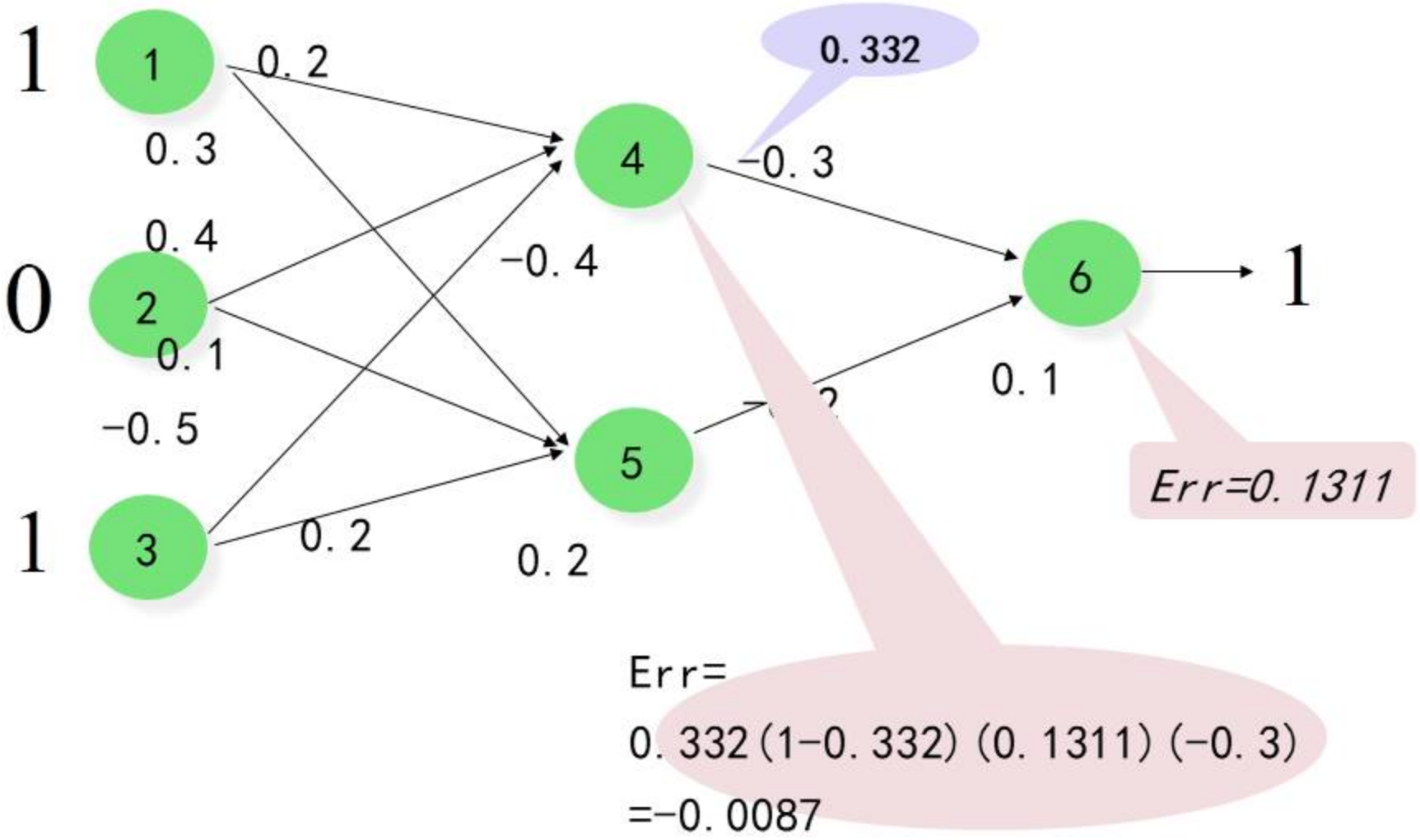
Err =

0.474(1−0.474)(1−0.474)=

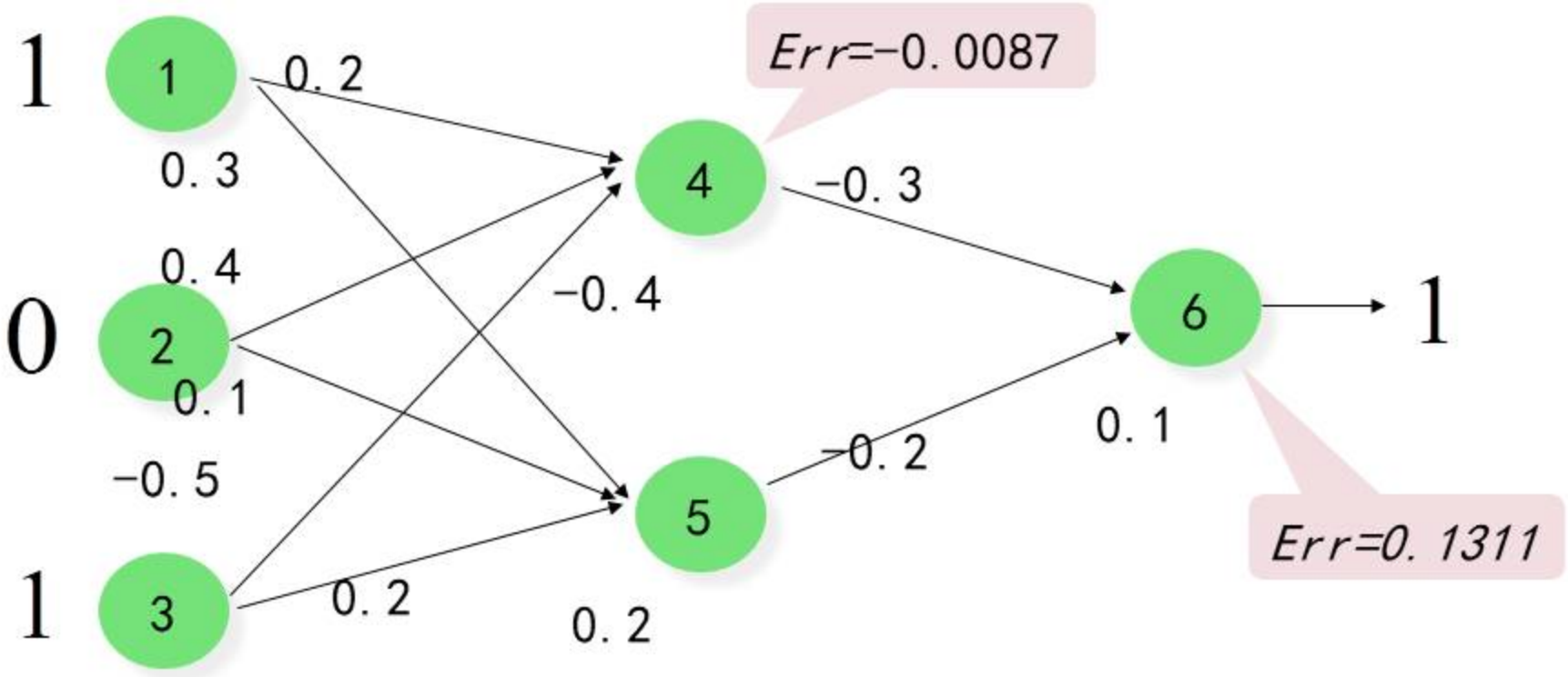
0.1311

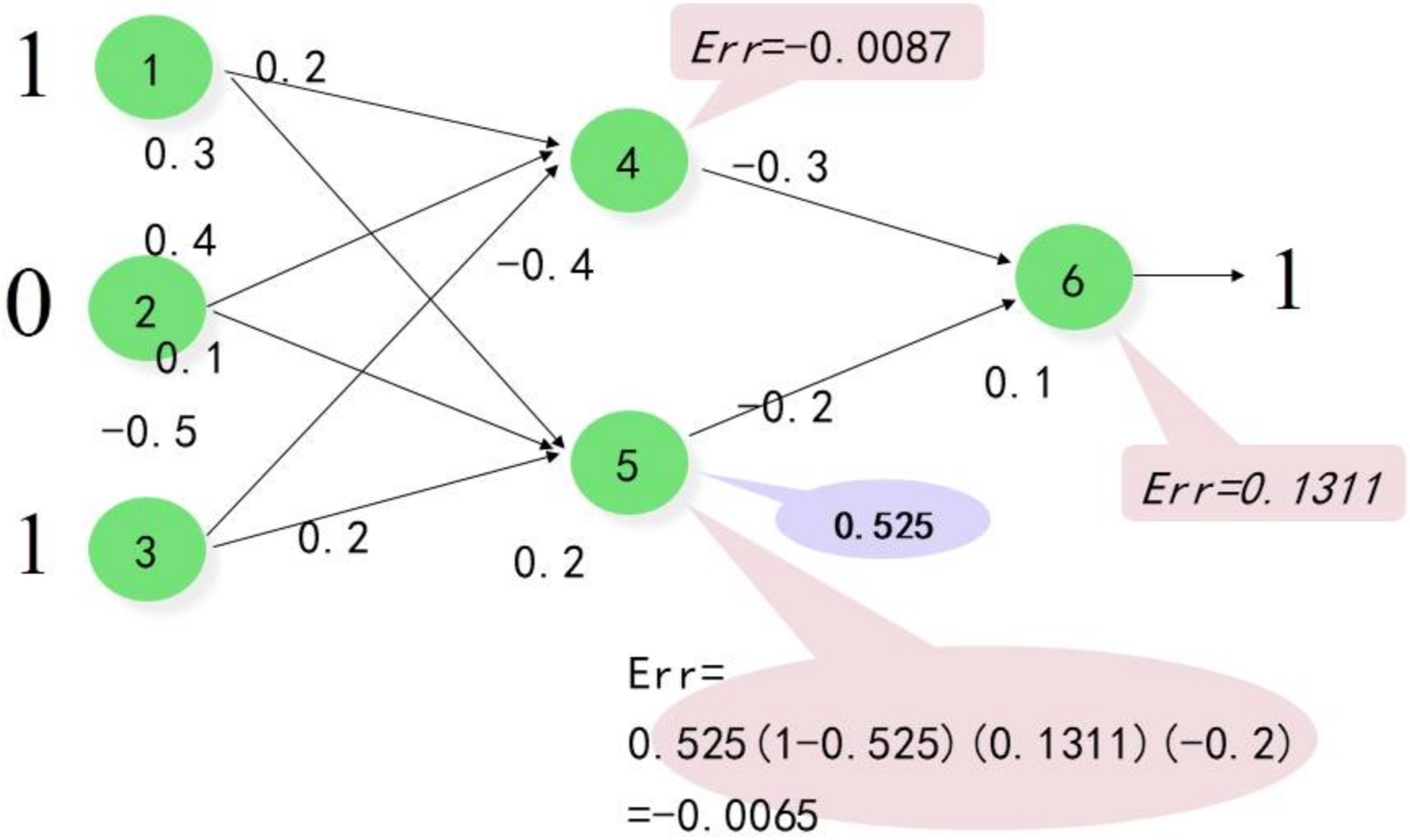
后向传播算法


1
0
1
1
2
3
4
5
6
0. 2
0. 3
0. 4
0. 1
−0. 5
−0. 4
0. 2
0. 2
−0. 3
0. 1
1
Err=0. 1311
$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$

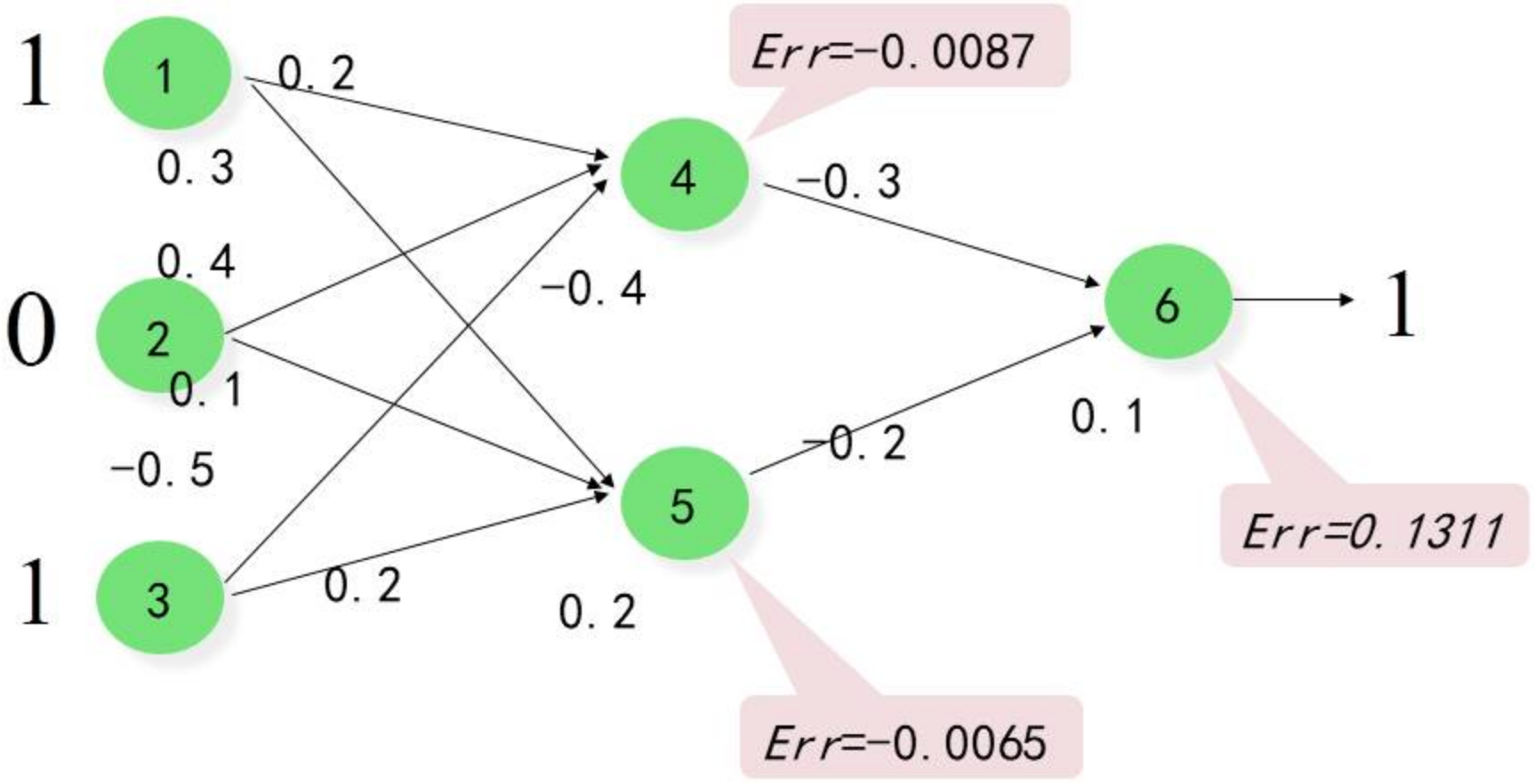

1
0
1
1
2
3
0.2
0.3
0.4
0.1
-0.5
0.2
-0.4
0.2
4
5
0.332
-0.3
-2
6
0.1
1
Err=0.1311
Err=
0.332(1-0.332)(0.1311)(-0.3)
=-0.0087

后向传播算法




1
0
1
1
2
3
0.2
0.3
0.4
0.1
-0.5
0.2
4
5
-0.4
0.2
Err=-0.0087
-0.3
-0.2
6
0.1
1
Err=0.1311
0.525
Err=
0.525(1-0.525)(0.1311)(-0.2)
=-0.0065

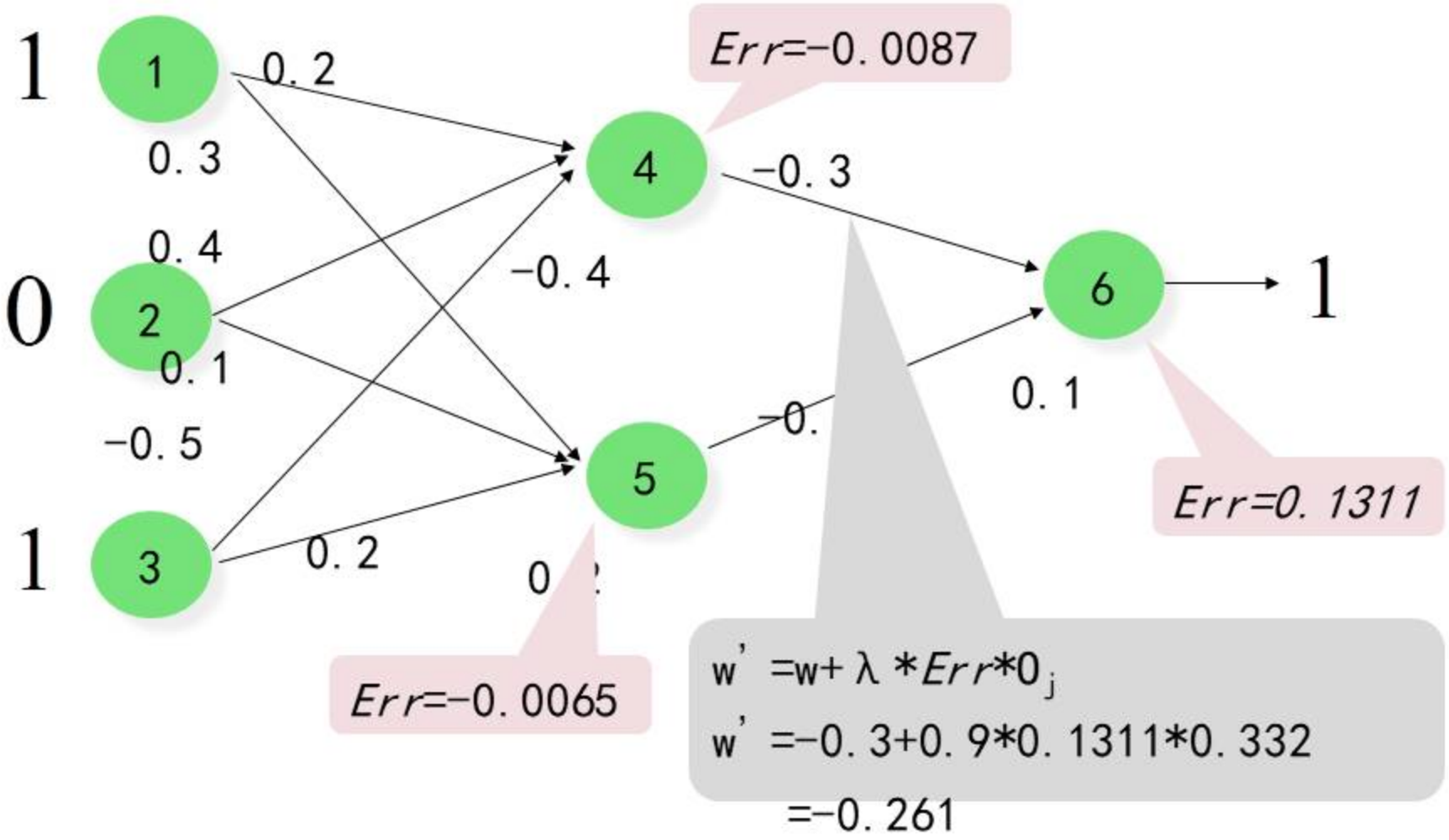
后向传播算法


1
0
1
1
2
3
4
5
6
0.2
0.3
0.4
0.1
-0.5
0.2
-0.4
0.2
-0.3
-0.2
0.1
1
Err=-0.0087
Err=0.1311
Err=-0.0065

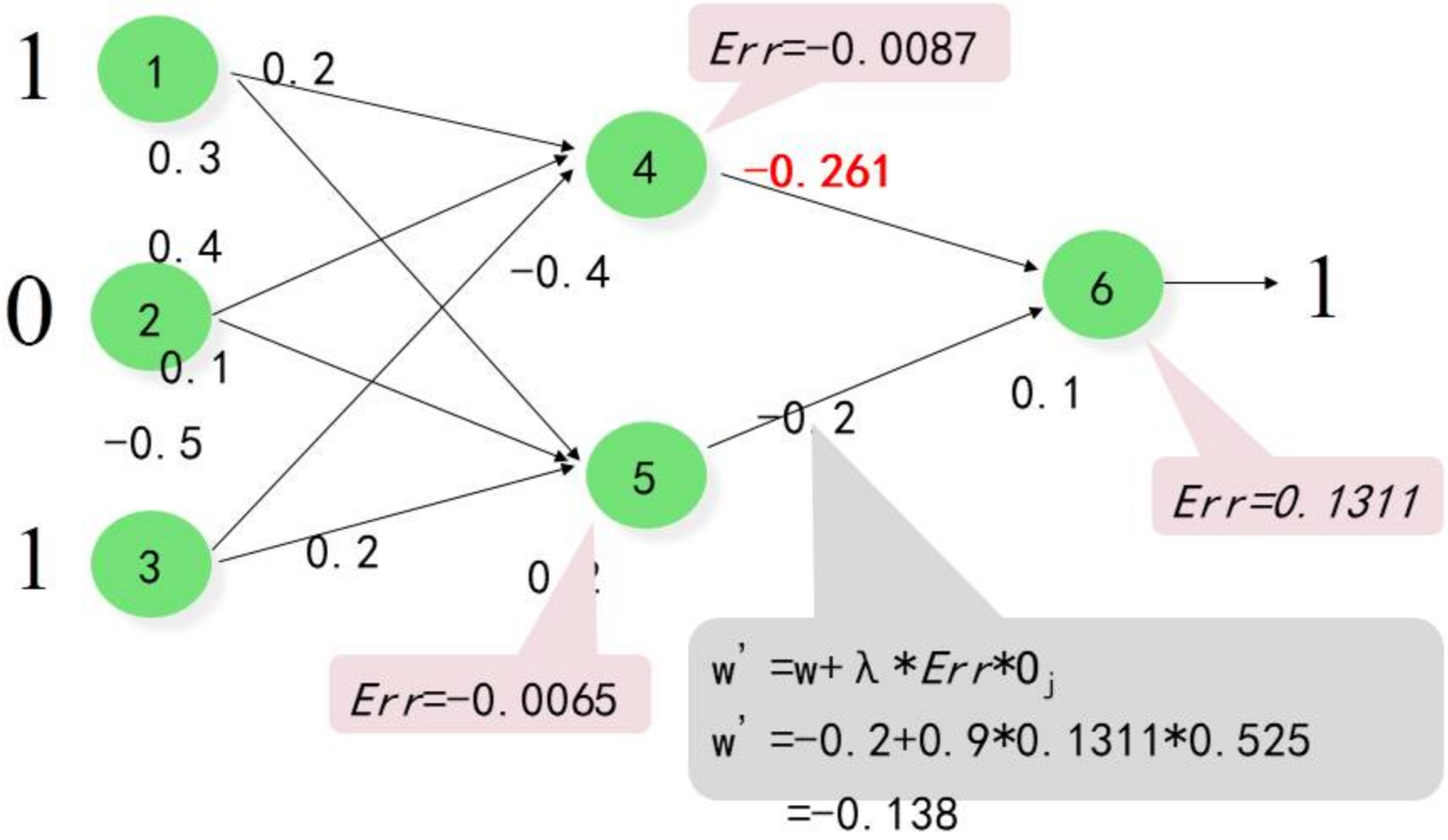
计算各结点的误差

Unit j	Err_j
6	$(0.474)(1-0.474)(1-0.474)=0.1311$
5	$(0.525)(1-0.525)(0.1311)(-0.2)=-0.0065$
4	$(0.332)(1-0.332)(0.1311)(-0.3)=-0.0087$

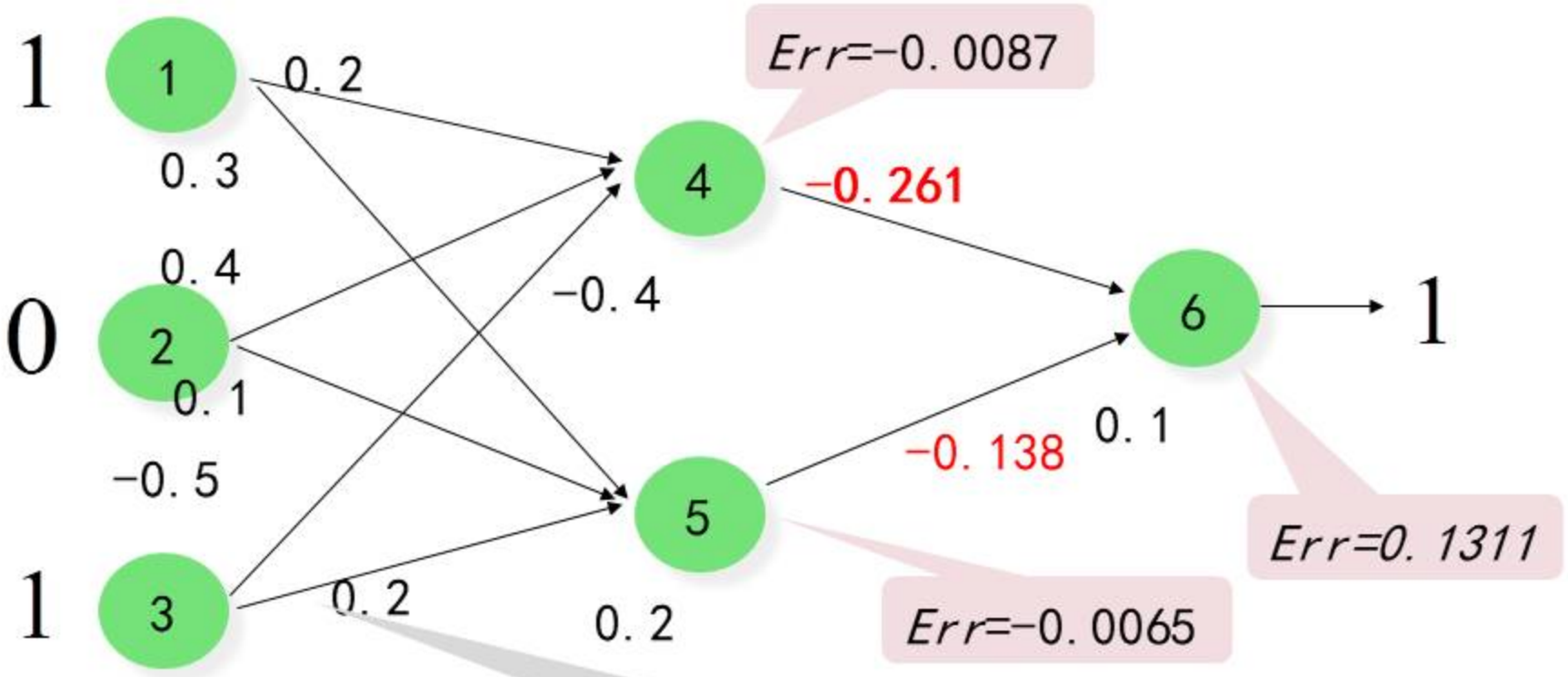
后向传播算法


1
0
1
1
2
3
4
5
6
0. 2
0. 3
0. 4
0. 1
−0. 5
−0. 4
0. 2
−0. 3
0. 1
1
Err=−0. 0087
Err=0. 1311
Err=−0. 0065
w' =w+ λ *Err*0j
w' =−0. 3+0. 9*0. 1311*0. 332
=−0. 261

后向传播算法


1
0
1
1
2
3
4
5
6
0.2
0.3
0.4
0.1
-0.5
0.2
-0.4
0.2
-0.261
0.1
1
Err=-0.0087
Err=0.1311
Err=-0.0065
w' =w+λ*Err*0j
w' =-0.2+0.9*0.1311*0.525
=-0.138

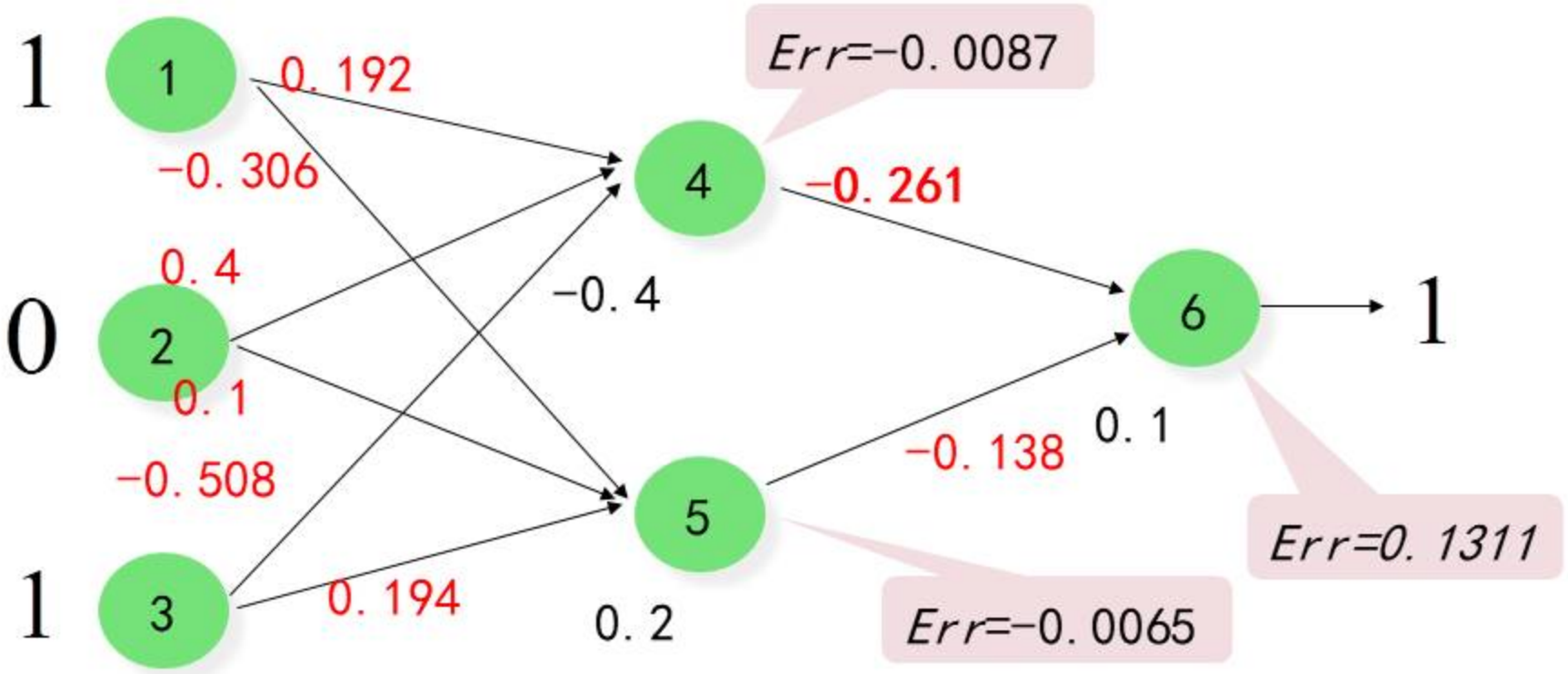
后向传播算法



$w' = w + \lambda * Err * O_j$

$w' = 0.2 + 0.9 * -0.0065 * 1 = 0.194$

后向传播算法


1
0
1
1
2
3
4
5
6
0. 192
−0. 306
0. 4
0. 1
−0. 508
0. 194
−0. 4
0. 2
−0. 261
−0. 138
0. 1
Err=−0. 0087
Err=0. 1311
Err=−0. 0065
1

根据误差调整新的权重 和偏倚

Weight or bias	*New value*
w_{46}	$-0.3+(0.9)(0.1311)(0.332)=-0.261$
w_{56}	$-0.2+(0.9)(0.1311)(0.525)=-0.138$
w_{14}	$0.2+(0.9)(-0.0087)(1)=0.192$
w_{15}	$-0.3+(0.9)(-0.0065)(1)=-0.306$
w_{24}	$0.4+(0.9)(-0.0087)(0)=0.4$
w_{25}	$0.1+(0.9)(-0.0065)(0)=0.1$
w_{34}	$-0.5+(0.9)(-0.0087)(1)=-0.508$
w_{35}	$0.2+(0.9)(-0.0065)(1)=0.194$
θ_6	$0.1+(0.9)(0.1311)=0.218$
θ_5	$0.2+(0.9)(-0.0065)=0.194$
θ_4	$-0.4+(0.9)(-0.0087)=-0.408$

后向传播算法

1. 初始化权重

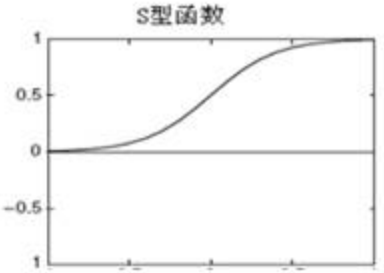
循环以下两步，直到满足条件

$$I_j = \sum_i w_{ij} x_i - t_j \qquad y_j = \frac{1}{1+e^{-I_j}}$$

1. 向前传播输入

在每个节点加权求和，再代入激活函数

$$\hat{y} = sign(\sum_i w_{ij} x_i - t_j)$$



2. 向后传播误差

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$w_{ij} = w_{ij} + \lambda Err_j y_i$$

$$t_j = t_j + \lambda Err_j$$

均方误差

$$J(w,b) = \frac{1}{n}\sum \frac{1}{2}(Y - Y^{hat})^2$$

权值迭代，梯度下降法

推导参考：

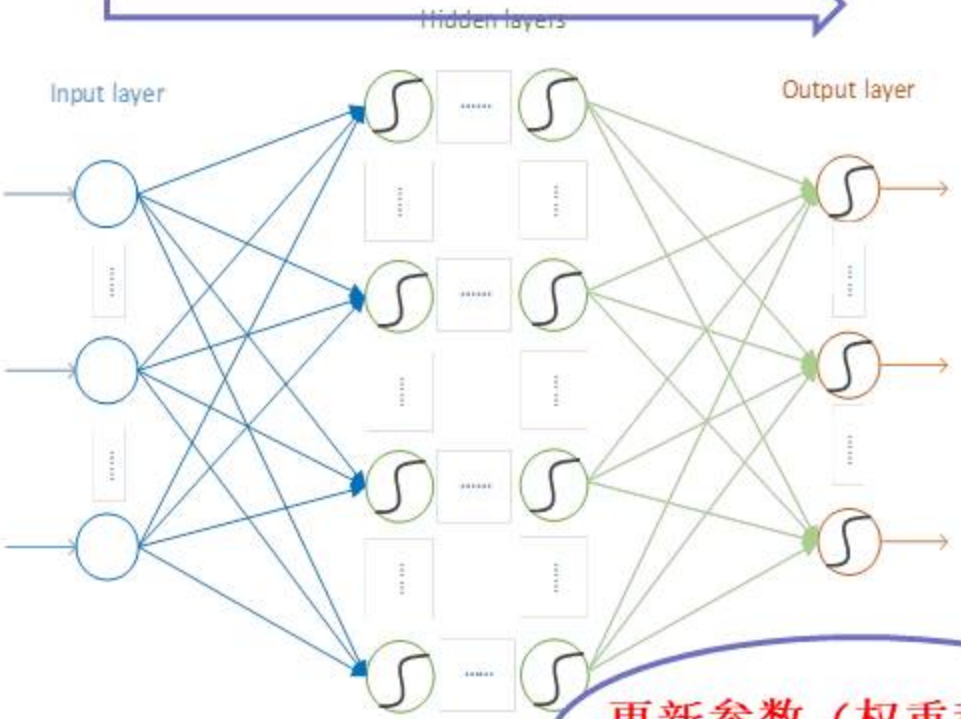
https://baijiahao.baidu.com/s?id=1589633691348109038&wfr=spider&for=pc

训练过程

$$D = \{(\vec{x_1}, \vec{y_1}), (\vec{x_2}, \vec{y_2}), \cdots, (\vec{x_m}, \vec{y_m})\}, \vec{x_i} \in R^d, \vec{y_i} \in R^l$$

训练集　　验证集

前向传播



从训练集中选择一部分样本（Batch）

每个样本产生的预测和其真实标签存在一定误差

预测标签 ≠ 真实标签

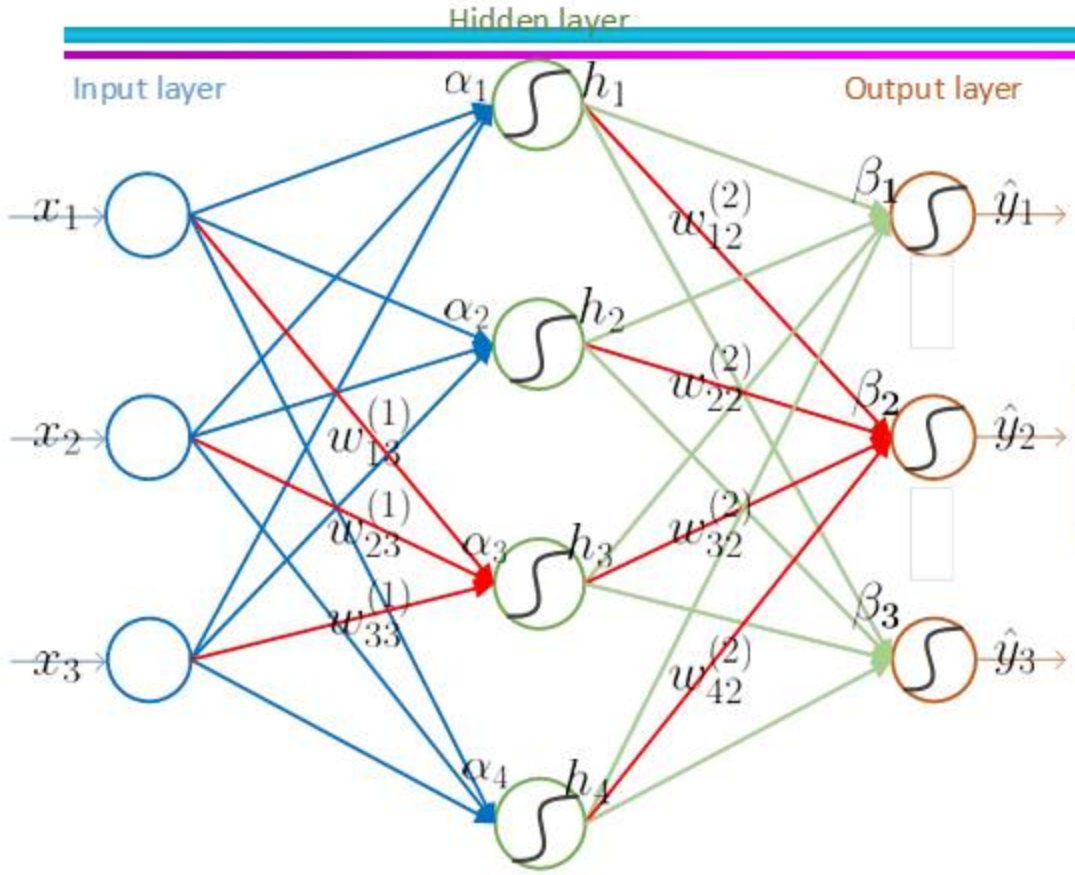
损失函数

最小化损失

更新参数（权重和阈值）

迭代

示例



超参数

- 激活函数：Sigmoid
- 损失函数：均方差损失
- Batch size: 1

训练的参数

Weights $\vec{w}$, Bias $\vec{b}$ 随机初始化

最小化 $E = \frac{1}{2}\sum_{j=1}^{3}(\hat{y}_j - y_j)^2$

梯度下降法（Gradient Descent）

$$D = \{(\vec{x}_1, \vec{y}_1), (\vec{x}_2, \vec{y}_2), \cdots, (\vec{x}_{1000}, \vec{y}_{1000})\}, \vec{x} \in R^3, \vec{y} \in R^3$$

Input: $\vec{x} = (x_1, x_2, x_3)$

Output: $\vec{\hat{y}} = (\hat{y}_1, \hat{y}_2, \hat{y}_3)$

Real: $\vec{y} = (y_1, y_2, y_3)$

$$\alpha_3 = \sum_{k=1}^{3} w_{k3}^{(1)} x_k \qquad \beta_2 = \sum_{i=1}^{4} w_{i2}^{(2)} h_i$$

$$h_3 = f(\alpha_3 - b_3^{(1)}) \qquad \hat{y}_2 = f(\beta_2 - b_2^{(2)})$$

梯度下降法

- 什么是梯度?

For example: $f(\theta_0, \theta_1) \xrightarrow{\nabla} (\frac{\partial f}{\partial \theta_0}, \frac{\partial f}{\partial \theta_1})^T$

- 梯度 ∇f 指向函数增长最快的方向

- $-\nabla f$ 指向函数下降最快的方向

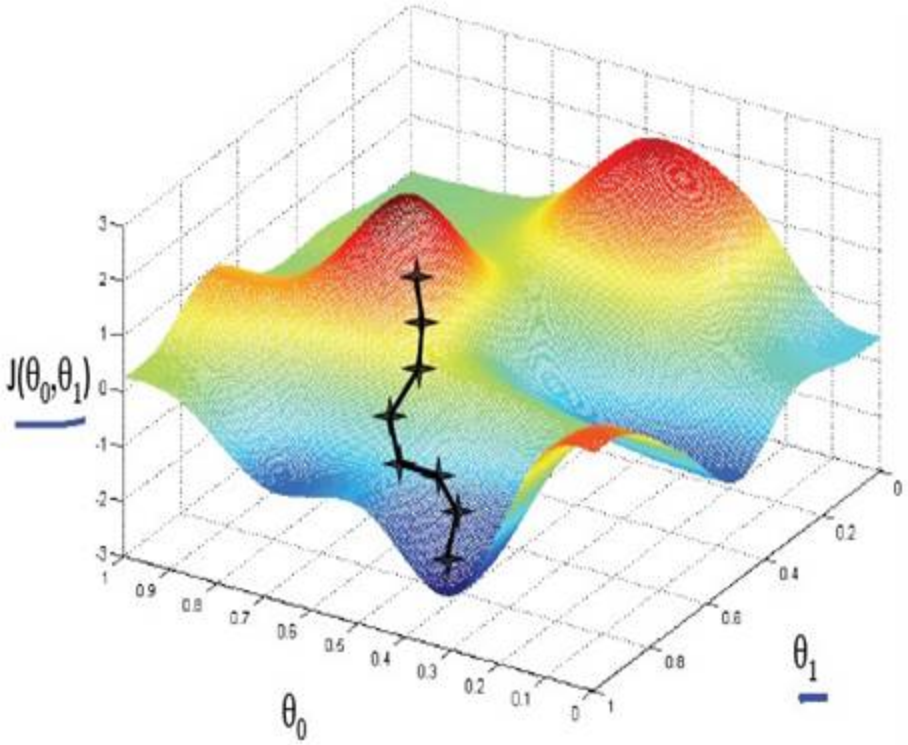
最小化 $f(\theta_0, \theta_1)$ $\Longrightarrow$ $\boxed{\vec{\theta} \leftarrow \vec{\theta} - \eta \nabla f(\vec{\theta})}$ $\Longrightarrow$

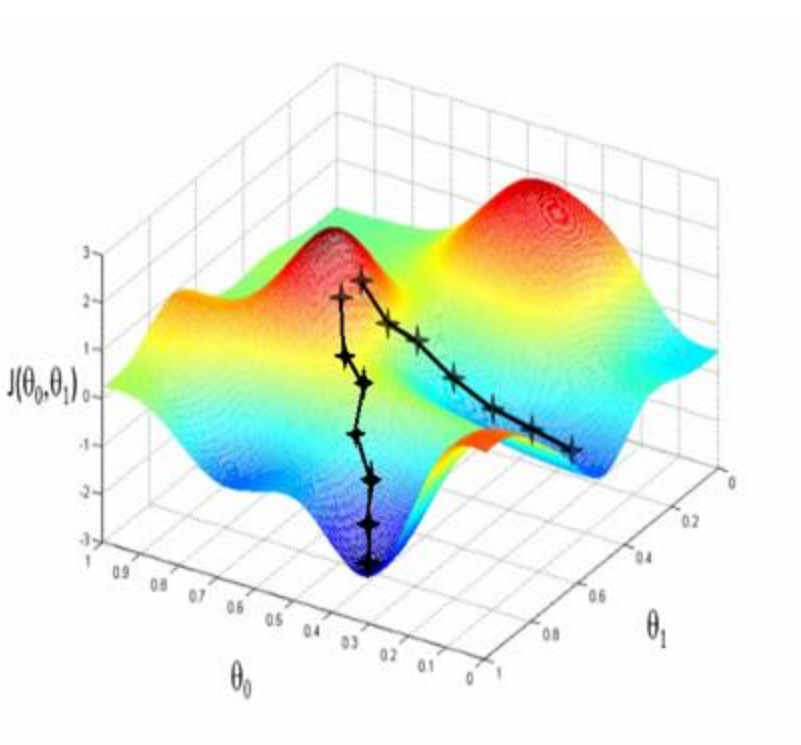
$$\theta_0 \leftarrow \theta_0 - \eta \frac{\partial f}{\partial \theta_0}$$

$$\theta_1 \leftarrow \theta_1 - \eta \frac{\partial f}{\partial \theta_1}$$

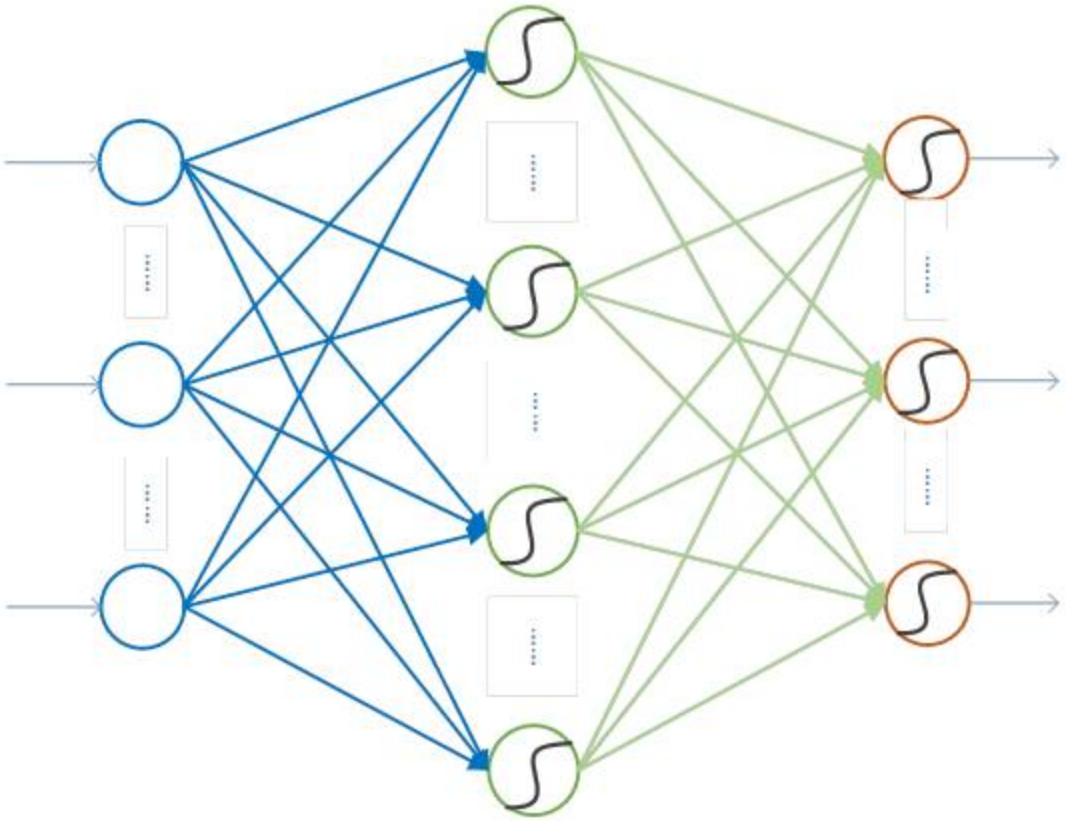
梯度下降法

梯度下降是求一个函数最小值的一阶迭代优化算法





更新神经网络中的参数



minimize loss function

$$E = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2$$

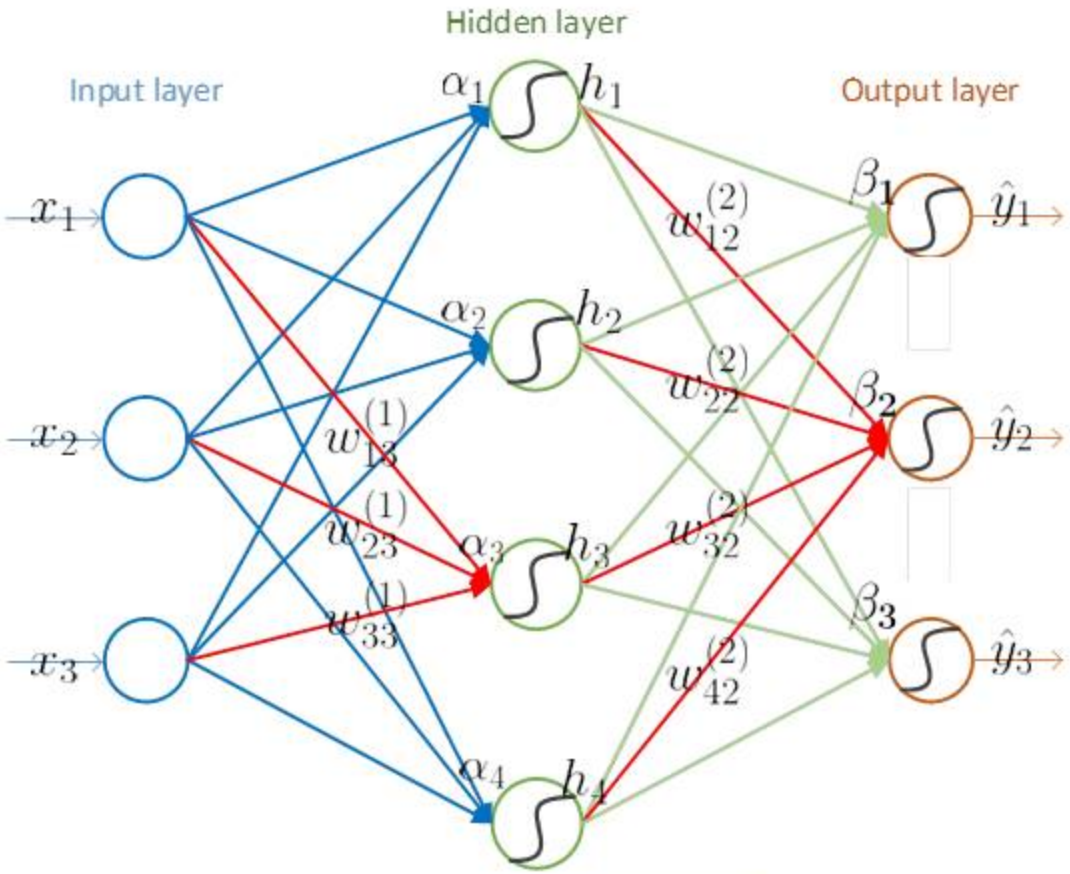
Update Parameters:

$$\vec{w} \leftarrow \vec{w} - \eta \nabla E$$

$$\vec{b} \leftarrow \vec{b} - \eta \nabla E$$

如何高效地更新参数？　　误差反传播算法（**Backpropagation**）

Backpropagation



$$\alpha_3 = \sum_{k=1}^{3} w_{k3}^{(1)} x_k$$

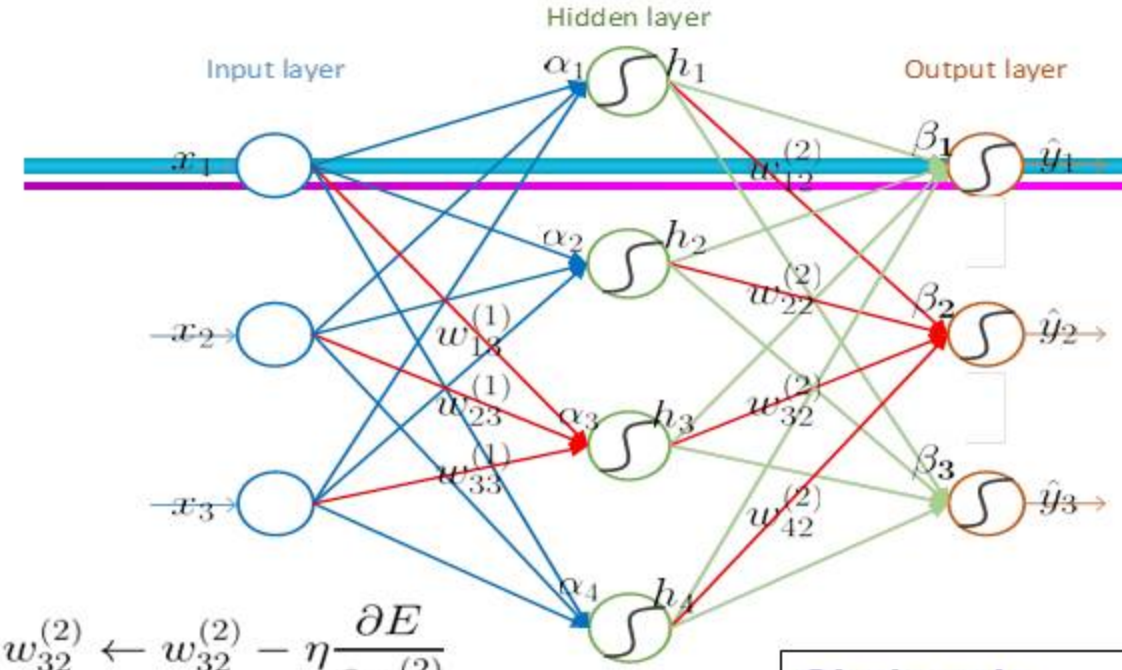
$$h_3 = f(\alpha_3 - b_3^{(1)})$$

$$\beta_2 = \sum_{i=1}^{4} w_{i2}^{(2)} h_i$$

$$\hat{y}_2 = f(\beta_2 - b_2^{(2)})$$

$$E = \frac{1}{2} \sum_{j=1}^{3} (\hat{y}_j - y_j)^2$$

$$w_{32}^{(2)} \leftarrow w_{32}^{(2)} - \eta \boxed{\frac{\partial E}{\partial w_{32}^{(2)}}}$$



$$\beta_2 = \sum_{i=1}^{4} w_{i2}^{(2)} h_i$$

$$\hat{y}_2 = f(\beta_2 - b_2^{(2)})$$

$$E = \frac{1}{2}\sum_{j=1}^{3}(\hat{y}_j - y_j)^2$$

$$w_{32}^{(2)} \leftarrow w_{32}^{(2)} - \eta \frac{\partial E}{\partial w_{32}^{(2)}}$$

$$w_{32}^{(2)} \leftarrow w_{32}^{(2)} - \eta g_2 h_3$$

$$w_{21}^{(2)} \leftarrow w_{21}^{(2)} - \eta g_1 h_2$$

Chain rule

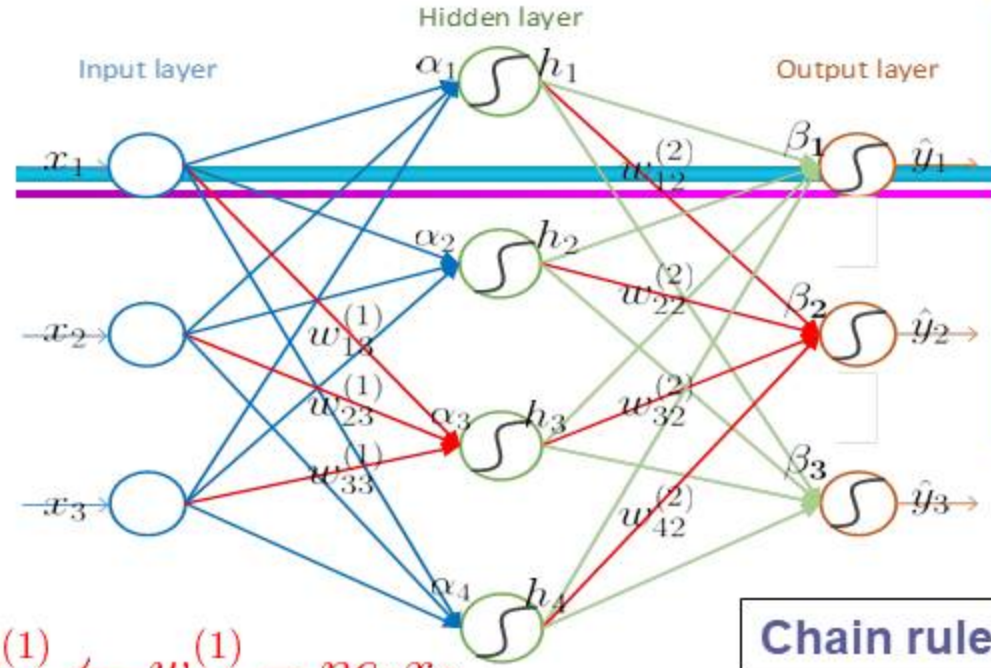
$$\frac{\partial E}{\partial w_{32}^{(2)}} = \frac{\partial E}{\partial \hat{y}_2} \cdot \frac{\partial \hat{y}_2}{\partial \beta_2} \cdot \frac{\partial \beta_2}{\partial w_{32}^{(2)}} = \hat{y}_2(1-\hat{y}_2)(\hat{y}_2 - y_2)h_3$$

梯度项 g_2

$$\frac{\partial E}{\partial \hat{y}_2} = \hat{y}_2 - y_2$$

$$\frac{\partial \hat{y}_2}{\partial \beta_2} = f'(\beta_2 - b_2^{(2)}) = f(\beta_2 - b_2^{(2)})(1 - f(\beta_2 - b_2^{(2)})) = \hat{y}_2(1-\hat{y}_2)$$

$$\frac{\partial \beta_2}{\partial w_{32}^{(2)}} = h_3$$



$$g_j = \frac{\partial E}{\partial \beta_j} = \hat{y}_j(1-\hat{y}_j)(\hat{y}_j - y_j)$$

$$w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} - \eta g_j h_i$$

$$b_j^{(2)} \leftarrow b_j^{(2)} + \eta g_j$$

输出层总结

$$w_{23}^{(1)} \leftarrow w_{23}^{(1)} - \eta \frac{\partial E}{\partial w_{23}^{(1)}}$$

$$w_{23}^{(1)} \leftarrow w_{23}^{(1)} - \eta e_3 x_2$$

$$w_{12}^{(1)} \leftarrow w_{12}^{(1)} - \eta e_2 x_1$$

Chain rule

梯度项 e_3

$$\frac{\partial E}{\partial w_{23}^{(1)}} = \frac{\partial E}{\partial h_3} \cdot \frac{\partial h_3}{\partial \alpha_3} \cdot \frac{\partial \alpha_3}{\partial w_{23}^{(1)}} = h_3(1-h_3)\left(\sum_{j=1}^{3} w_{3j}^{(2)} g_j\right) x_2$$

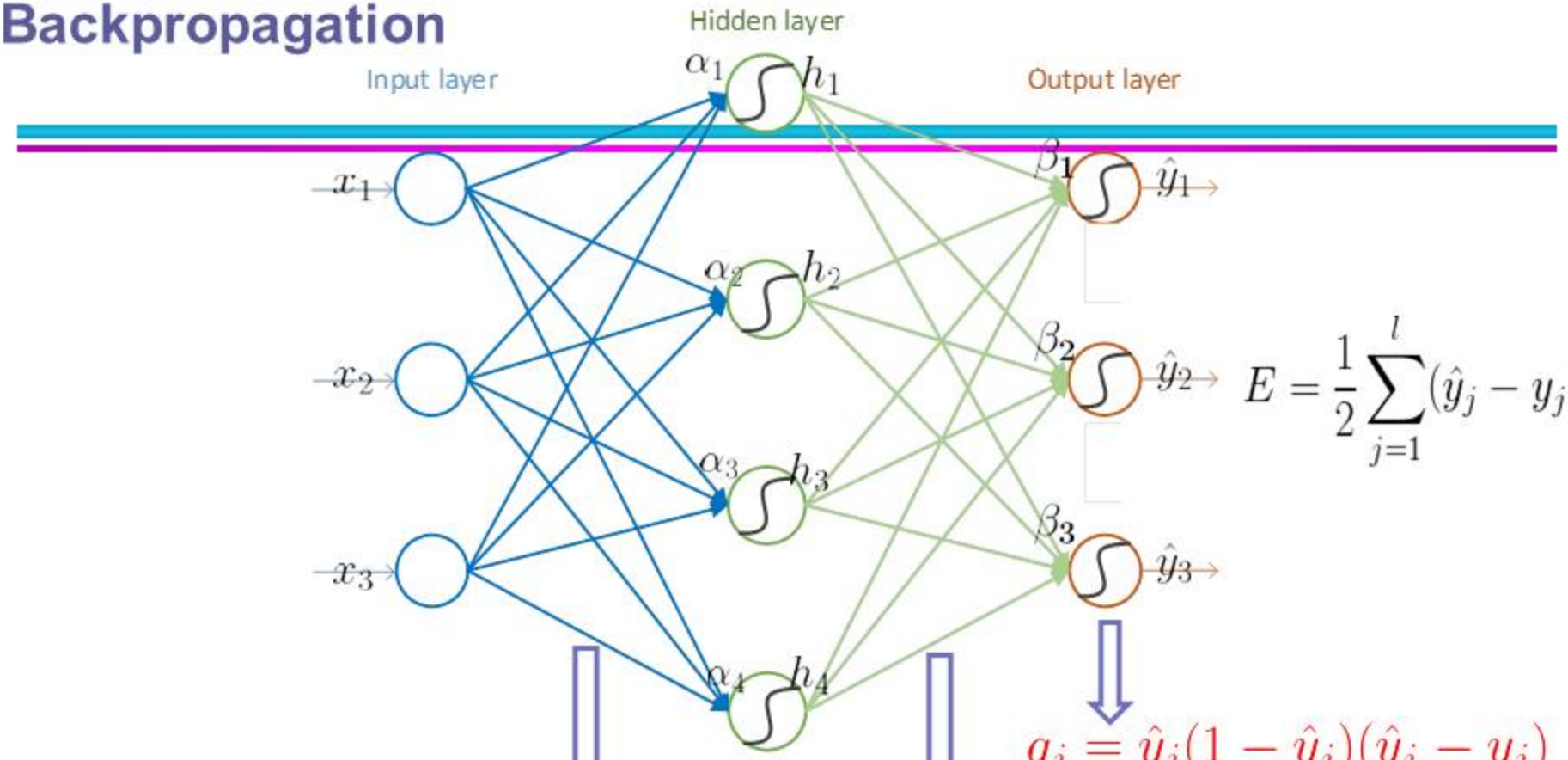
Chain rule

$$\frac{\partial E}{\partial h_3} = \sum_{j=1}^{3} \frac{\partial \beta_j}{\partial h_3} \cdot \frac{\partial E}{\partial \beta_j} = \sum_{j=1}^{3} w_{3j}^{(2)} g_j$$

$$\frac{\partial h_3}{\partial \alpha_3} = f'(\alpha_3 - b_3^{(1)}) = f(\alpha_3 - b_3^{(1)})(1 - f(\alpha_3 - b_3^{(1)})) = h_3(1-h_3)$$

$$\frac{\partial \alpha_3}{\partial w_{23}^{(1)}} = x_2$$

Backpropagation



$$E = \frac{1}{2}\sum_{j=1}^{l}(\hat{y}_j - y_j)^2$$

$$g_j = \hat{y}_j(1-\hat{y}_j)(\hat{y}_j - y_j)$$

$$w_{ki}^{(1)} \leftarrow w_{ki}^{(1)} - \eta e_i x_k$$
$$b_i^{(1)} \leftarrow b_i^{(1)} + \eta e_i$$

$$w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} - \eta g_j h_i$$
$$b_j^{(2)} \leftarrow b_j^{(2)} + \eta g_j$$

$$e_i = h_i(1-h_i)\sum_{j=1}^{3} w_{ij}^{(2)} g_j$$

原来的连接权

后向传播算法

1. 初始化权重

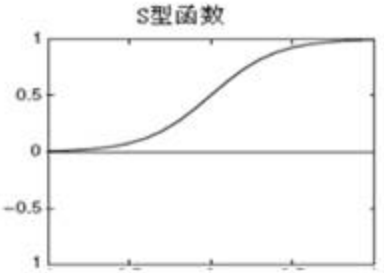
循环以下两步，直到满足条件

$$I_j = \sum_i w_{ij} x_i - t_j \qquad y_j = \frac{1}{1+e^{-I_j}}$$

1. 向前传播输入

在每个节点加权求和，再代入激活函数

$$\hat{y} = sign(\sum_i w_{ij} x_i - t_j)$$



2. 向后传播误差

$$Err_j = O_j(1-O_j)(T_j - O_j)$$

$$Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$$

$$w_{ij} = w_{ij} + \lambda Err_j y_i$$

$$t_j = t_j + \lambda Err_j$$

均方误差

$$J(w,b) = \frac{1}{n}\sum \frac{1}{2}(Y - Y^{hat})^2$$

权值迭代，梯度下降法

推导参考：

https://baijiahao.baidu.com/s?id=1589633691348109038&wfr=spider&for=pc

人工神经网络

总结

- 普适近似，精度较高
- 噪声敏感
- 训练非常耗时，但对目标分类较快
- 缺少完整、成熟的理论体系

神经网络分类编程实践

- https://scikit-learn.org/stable/modules/neural_networks_supervised.html

- 同学们可以尝试利用python读入本地iris数据集，来完成神经网络分类，分析其分类效果

第13次课后作业

- 第十三次课后作业-在educoder平台上完成作业
- https://www.educoder.net/shixuns/tf9qcivo/challenges

提交作业截至时间：2020年4月2日

Any Questions?

谢谢！