深度学习

# 神经网络反向传播

## 之数学原理

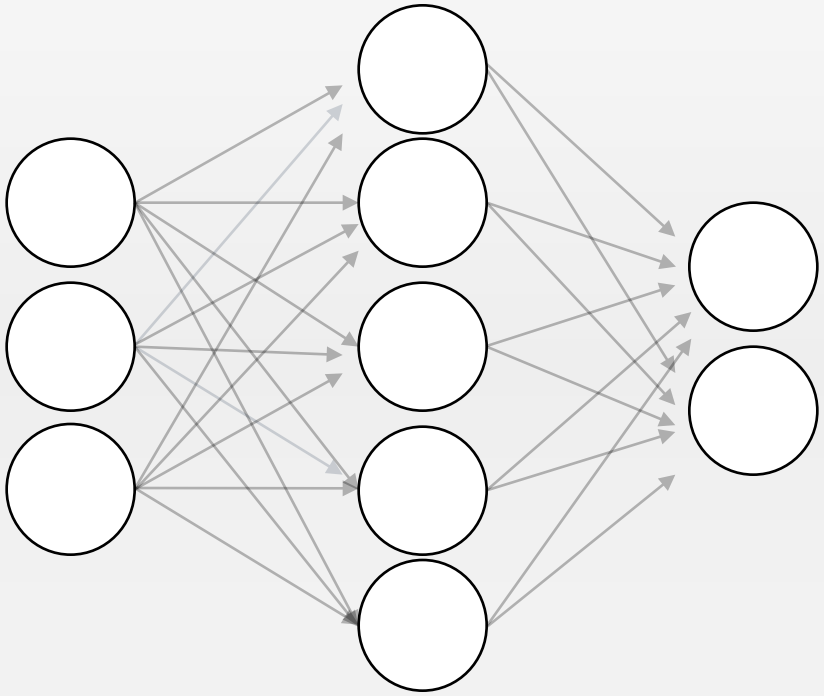
张江

集智AI学园
campus.swarma.org

# 主要内容

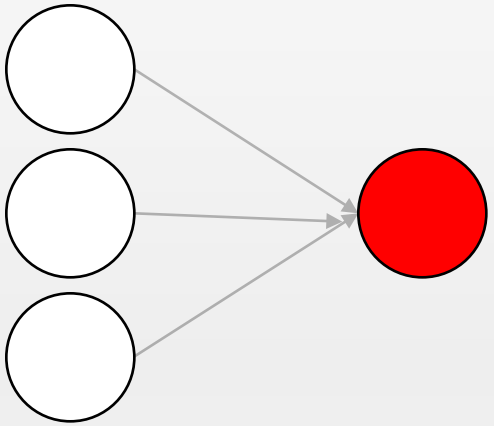
- 神经元网络模型
- 反向传播算法

# (前馈) 人工神经网络

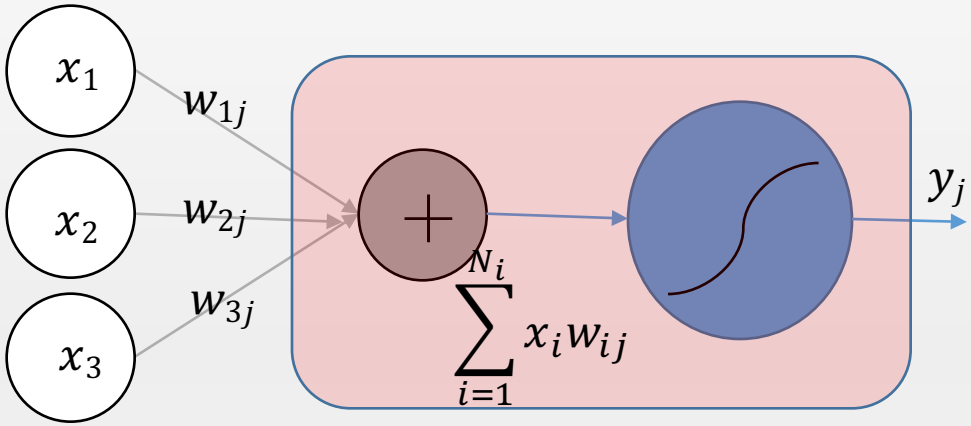


输入　　隐含　　输出

# 神经元



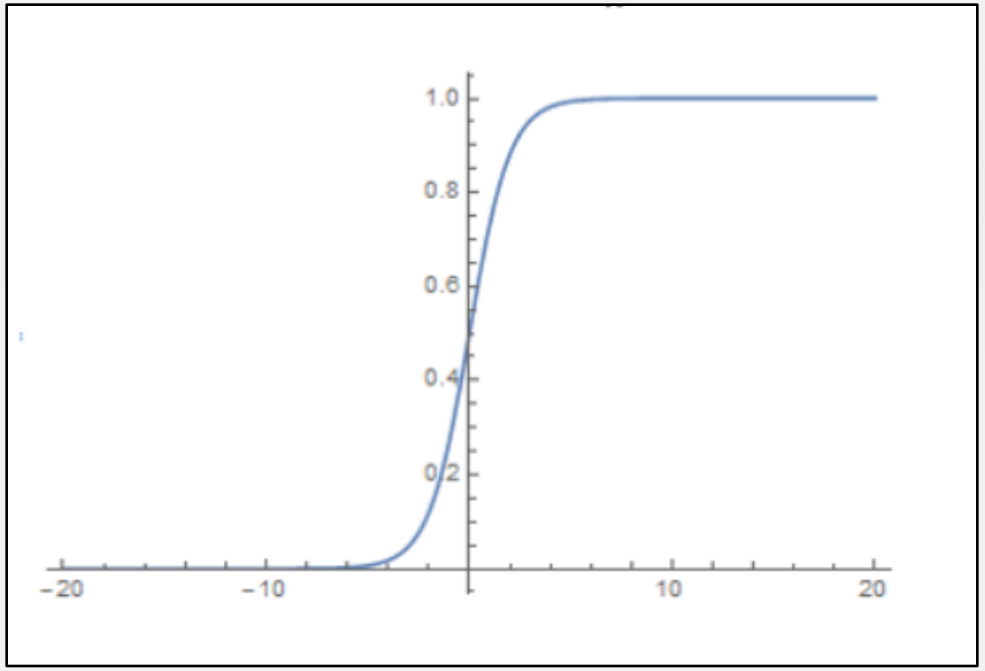
一个神经元细胞




$$y_j = \sigma\left(\sum_{i=1}^{N_i} x_i w_{ij}\right)$$

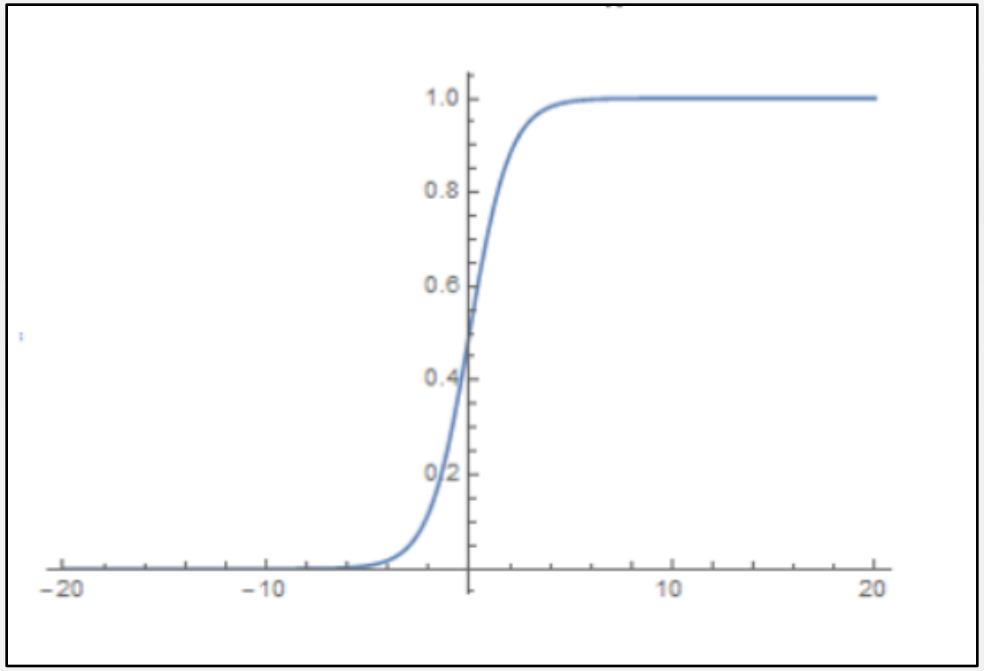
# 激活函数

Sigmoid $\sigma(x) = \dfrac{1}{1 + e^{-x}}$



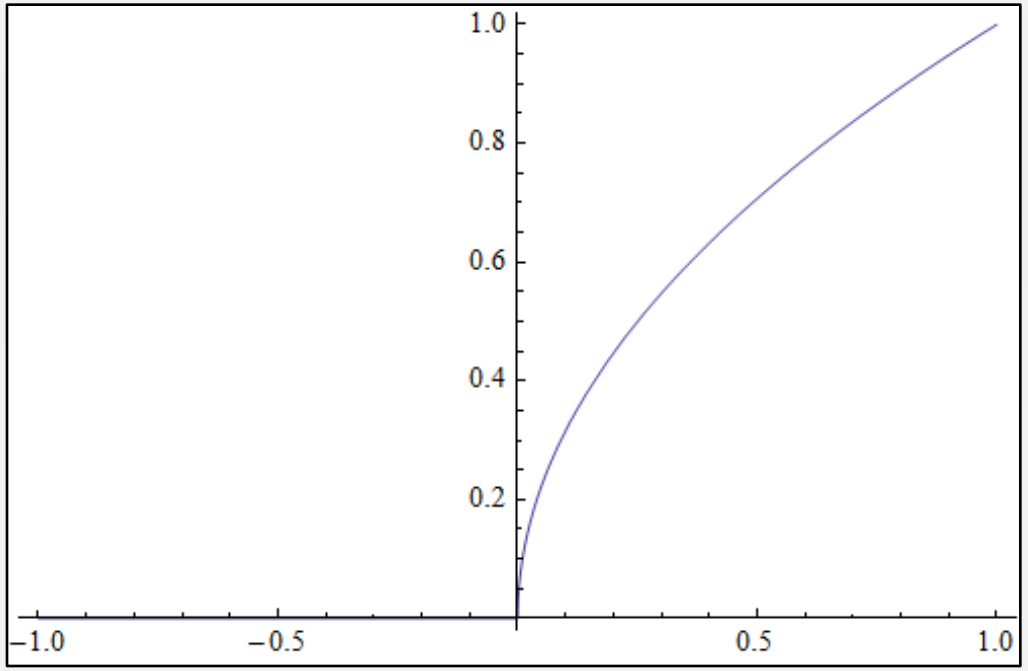
# 激活函数

Sigmoid $\sigma(x) = \frac{1}{1+e^{-x}}$

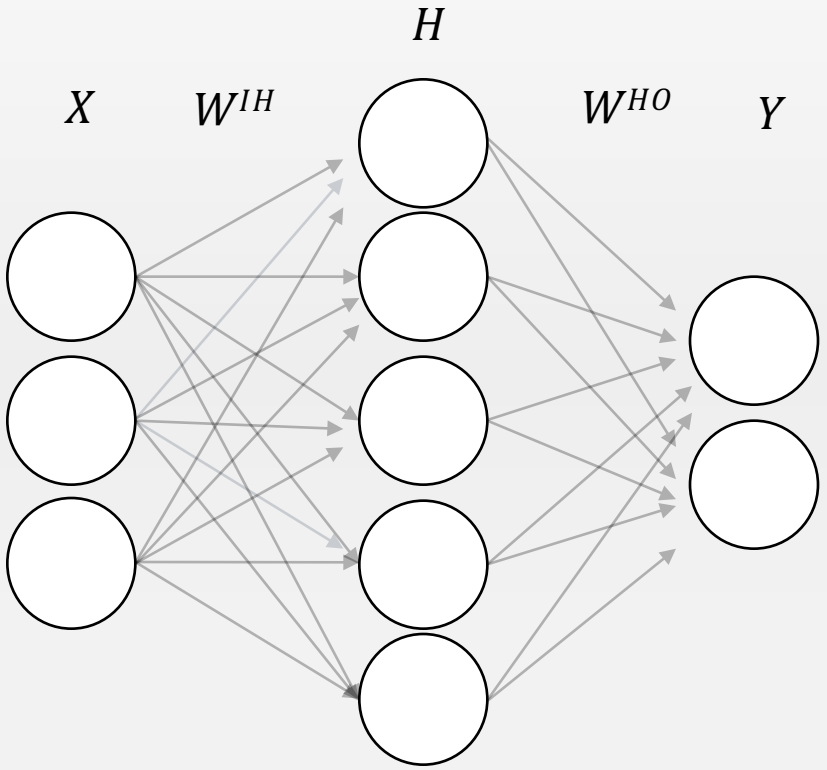



ReLU

$$\sigma(x) = ReLU(x) = \begin{cases} x^a & x > 0 \\ 0 & x \leq 0 \end{cases}$$



# 简便表示

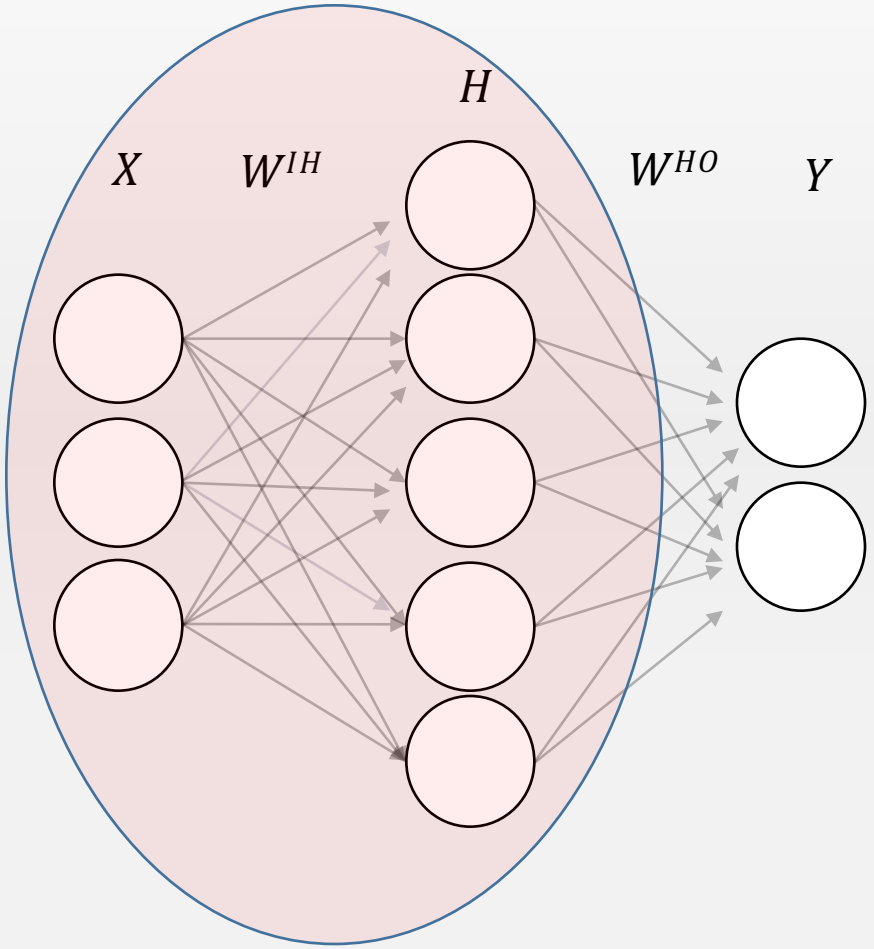



$$X = (-0.1, 0.2, 0.3)$$

$$W^{IH} = \begin{pmatrix} 0.1 & 1.2 & 0.3 & 1 & 1 \\ -0.1 & 0.01 & 0.4 & 2 & 0.1 \\ 0.3 & 0.9 & 0.2 & 2 & 0.9 \end{pmatrix}$$

$$W^{HO} = \begin{pmatrix} 0.1 & 0.1 \\ -0.3 & -0.5 \\ 0.1 & 3 \\ 0.02 & 0.01 \\ -0.4 & 0.6 \end{pmatrix}$$

# 简便表示



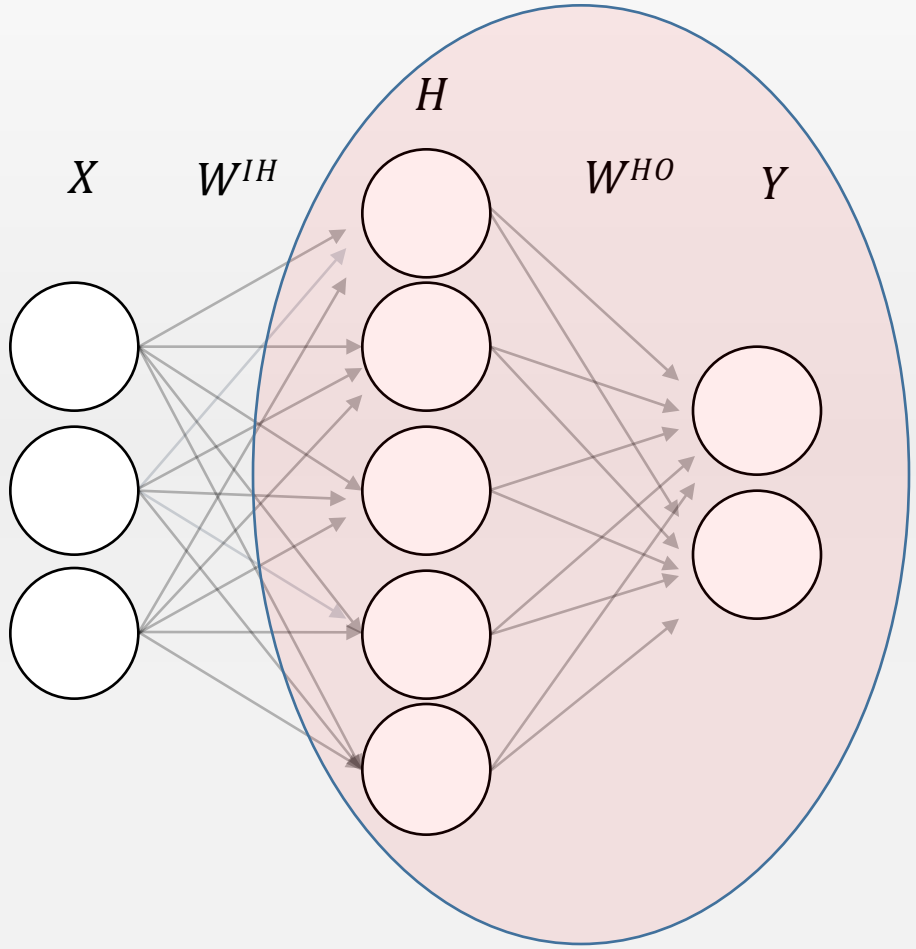

$$H = \sigma(X \cdot W^{IH})$$

$$X = (-0.1, 0.2, 0.3)$$

$$W^{IH} = \begin{pmatrix} 0.1 & 1.2 & 0.3 & 1 & 1 \\ -0.1 & 0.01 & 0.4 & 2 & 0.1 \\ 0.3 & 0.9 & 0.2 & 2 & 0.9 \end{pmatrix}$$

$$W^{HO} = \begin{pmatrix} 0.1 & 0.1 \\ -0.3 & -0.5 \\ 0.1 & 3 \\ 0.02 & 0.01 \\ -0.4 & 0.6 \end{pmatrix}$$

# 简便表示



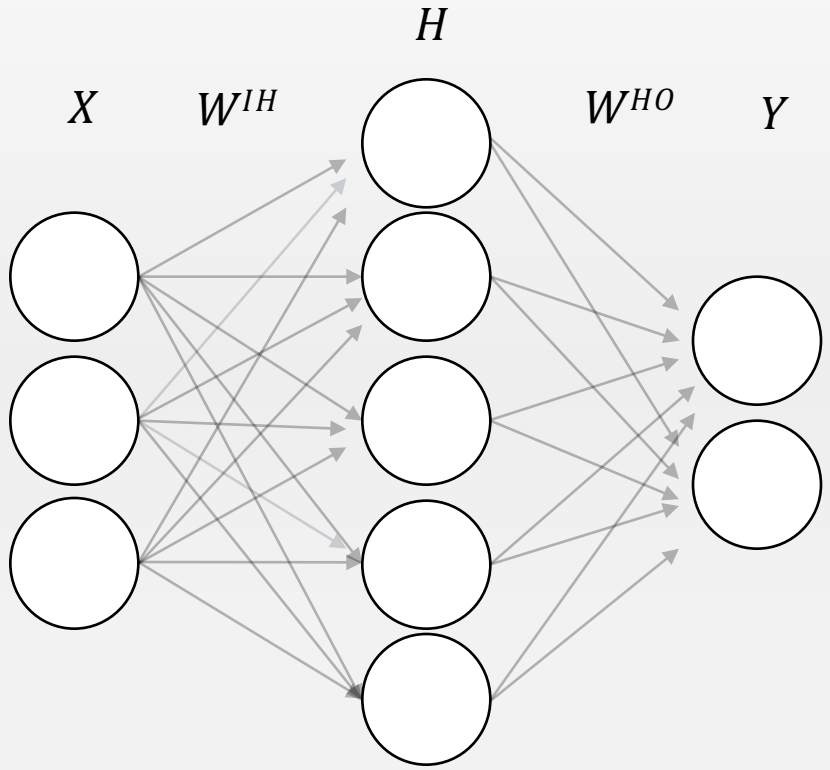

$$H = \sigma(X \cdot W^{IH}) \qquad Y = \sigma(H \cdot W^{HO})$$

$$X = (-0.1, 0.2, 0.3)$$

$$W^{IH} = \begin{pmatrix} 0.1 & 1.2 & 0.3 & 1 & 1 \\ -0.1 & 0.01 & 0.4 & 2 & 0.1 \\ 0.3 & 0.9 & 0.2 & 2 & 0.9 \end{pmatrix}$$

$$W^{HO} = \begin{pmatrix} 0.1 & 0.1 \\ -0.3 & -0.5 \\ 0.1 & 3 \\ 0.02 & 0.01 \\ -0.4 & 0.6 \end{pmatrix}$$

# 分类问题




$$Y = \sigma(\sigma(X \cdot W^{IH}) \cdot W^{HO})$$

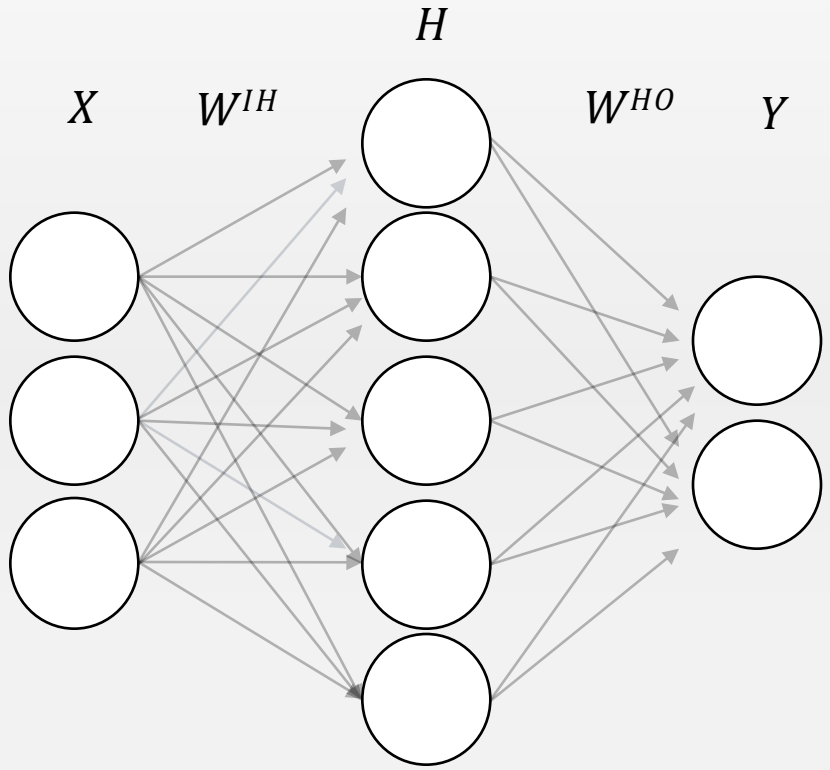
$$H = \sigma(X \cdot W^{IH}) \qquad Y = \sigma(H \cdot W^{HO})$$

$$X = (-0.1, 0.2, 0.3)$$

$$W^{IH} = \begin{pmatrix} 0.1 & 1.2 & 0.3 & 1 & 1 \\ -0.1 & 0.01 & 0.4 & 2 & 0.1 \\ 0.3 & 0.9 & 0.2 & 2 & 0.9 \end{pmatrix}$$

$$W^{HO} = \begin{pmatrix} 0.1 & 0.1 \\ -0.3 & -0.5 \\ 0.1 & 3 \\ 0.02 & 0.01 \\ -0.4 & 0.6 \end{pmatrix}$$

# 预测问题




$$Y = \sigma(X \cdot W^{IH}) \cdot W^{HO}$$

$$H = \sigma(X \cdot W^{IH})$$

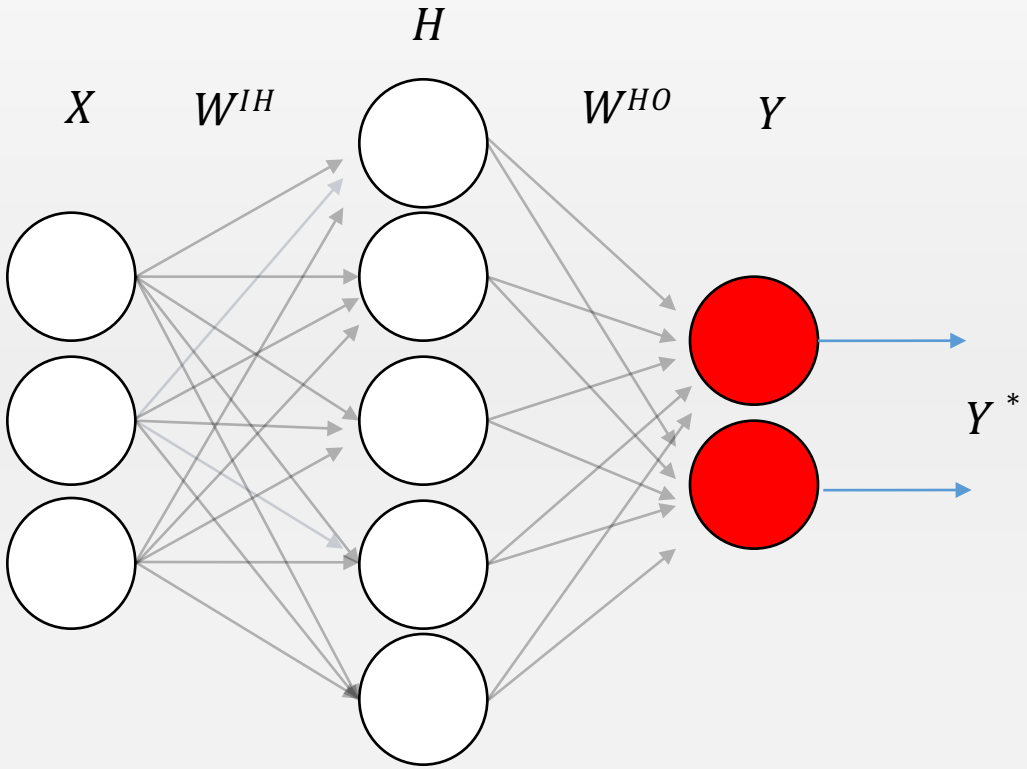
$$Y = H \cdot W^{HO}$$

$$X = (-0.1, 0.2, 0.3)$$

$$W^{IH} = \begin{pmatrix} 0.1 & 1.2 & 0.3 & 1 & 1 \\ -0.1 & 0.01 & 0.4 & 2 & 0.1 \\ 0.3 & 0.9 & 0.2 & 2 & 0.9 \end{pmatrix}$$

$$W^{HO} = \begin{pmatrix} 0.1 & 0.1 \\ -0.3 & -0.5 \\ 0.1 & 3 \\ 0.02 & 0.01 \\ -0.4 & 0.6 \end{pmatrix}$$

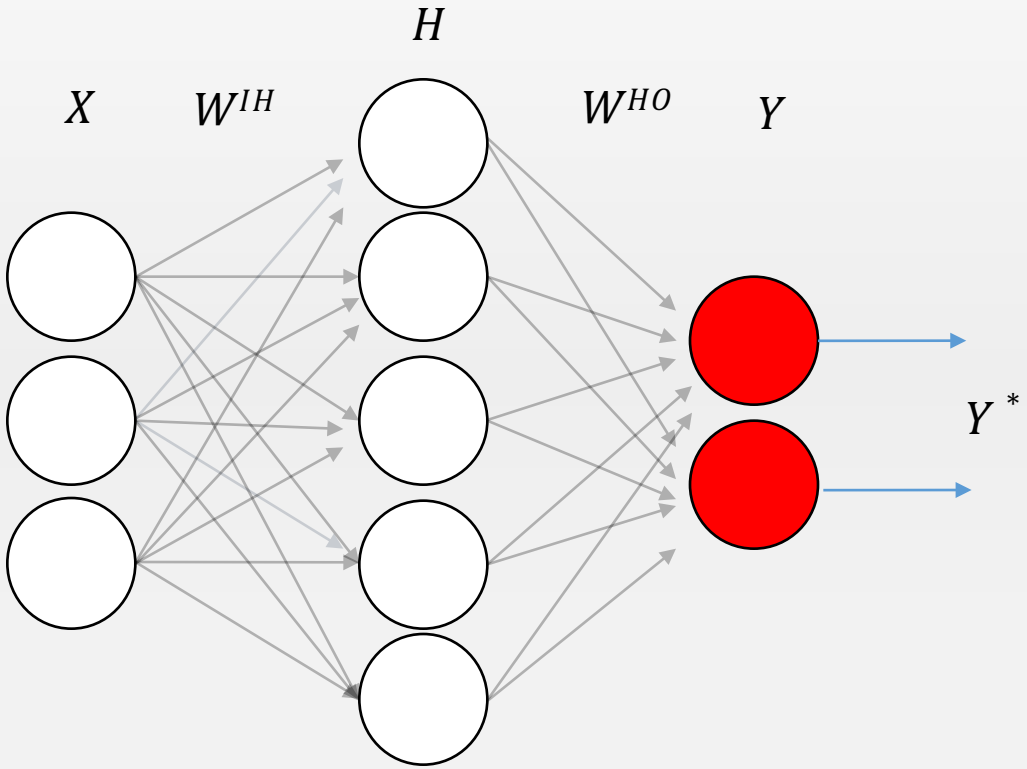
# 误差计算




$e_j = y_j^* - y_j$ ,　　$E = (Y^* - Y)$

总误差：　$\frac{1}{N_O}\sum_{j=1}^{N_O} e_j^2$

# 学习问题



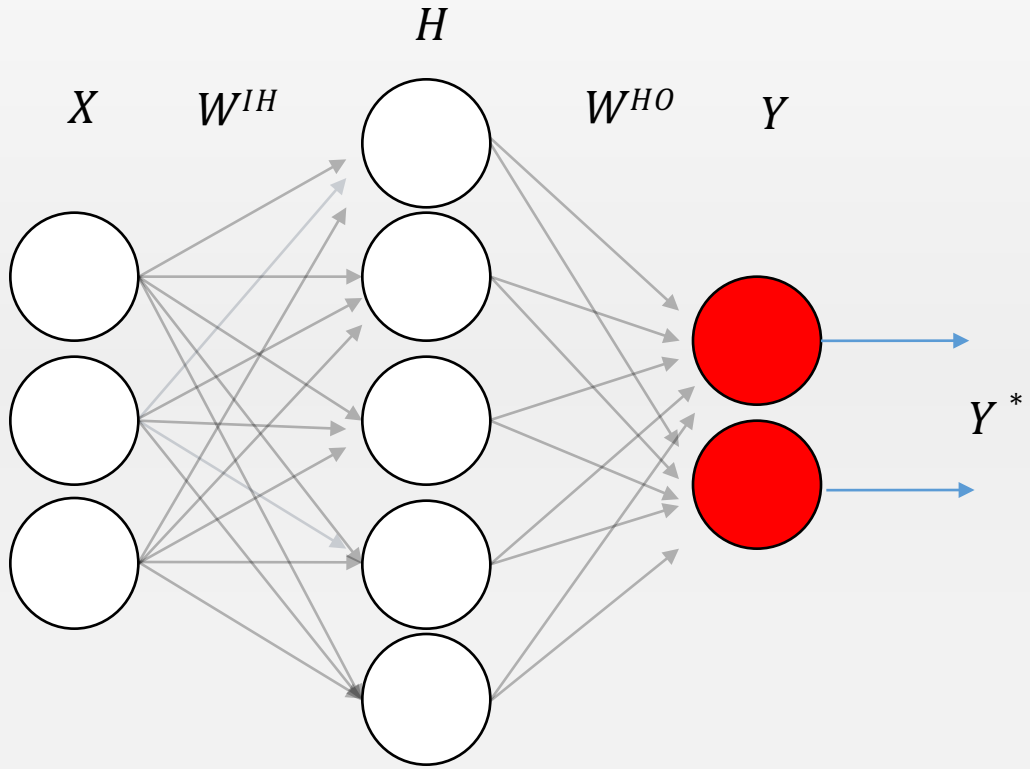

$$e_j = y_j^* - y_j, \qquad E = (Y^* - Y)$$

总误差：
$$\varepsilon = \frac{1}{N_O}\sum_{j=1}^{N_O} e_j^2$$

当神经网络的结构给定，只有$W^{IH}, W^{HO}$是需要调节的，故而神经网络的学习就是要找到一组最优的W*

$$W^* = \operatorname{argmin}_W \varepsilon(W)$$

# 梯度下降




求解优化问题

$$W^* = \text{argmin}_W \, \mathcal{E}(W)$$

$$W_{t+1} = W_t - \alpha \Delta W$$

$$\Delta W = \frac{\partial \mathcal{E}}{\partial W}$$ 梯度大小

人工神经网络这种特殊的结构使得梯度计算可以很方便

# 反向传播算法（Back propagation algorithm）

- BP算法是一种神经网络极其重要的算法
- 它使得神经网络在诸多机器学习算法中胜出
- 对于任意的可微分结构（所有的函数都是可微的）都可以应用BP算法
- 起源于导数的链式求导法则

# BP的一般性数学原理

- 广义的人工神经网络

$y = f_1(w_1, f_2(w_2, f_3 \cdots f_n(w_n, x)))$, 其中第i层神经元是：

$$x_i = f_i(w_i, f_{i+1}(w_{i+1}, f_{i+2} \cdots f_n(w_n, x) \cdots)),$$

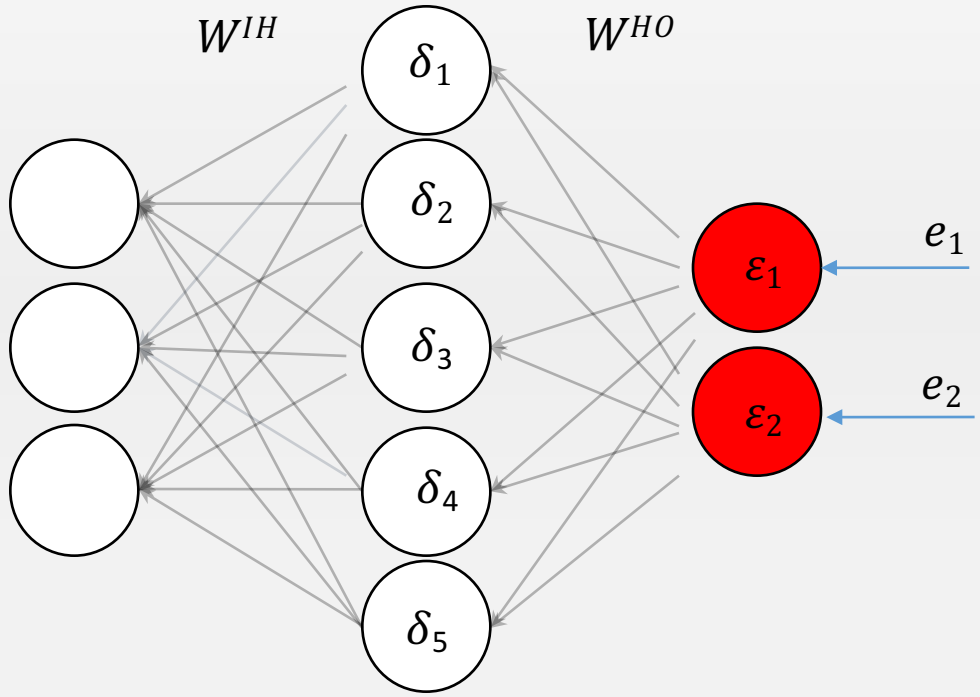
- 求梯度

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_4} \cdots \frac{\partial x_{i-1}}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_i} \qquad \frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial x_2} \cdot \frac{\partial x_2}{\partial x_3} \cdot \frac{\partial x_3}{\partial x_4} \cdots \frac{\partial x_{i-1}}{\partial x_i}$$

$$\frac{\partial y}{\partial w_i} = \frac{\partial y}{\partial x_i} \cdot \frac{\partial x_i}{\partial w_i}$$

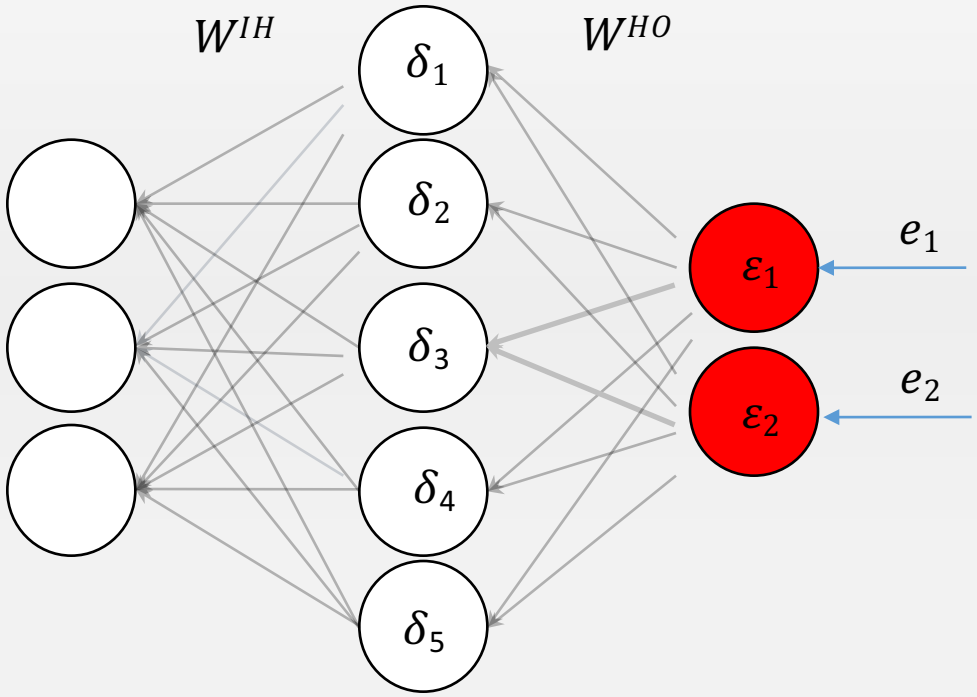
$$\frac{\partial y}{\partial x_i} = \frac{\partial y}{\partial x_{i-1}} \cdot \frac{\partial x_{i-1}}{\partial x_i}$$

# 反向传播算法（Back propagation algorithm）



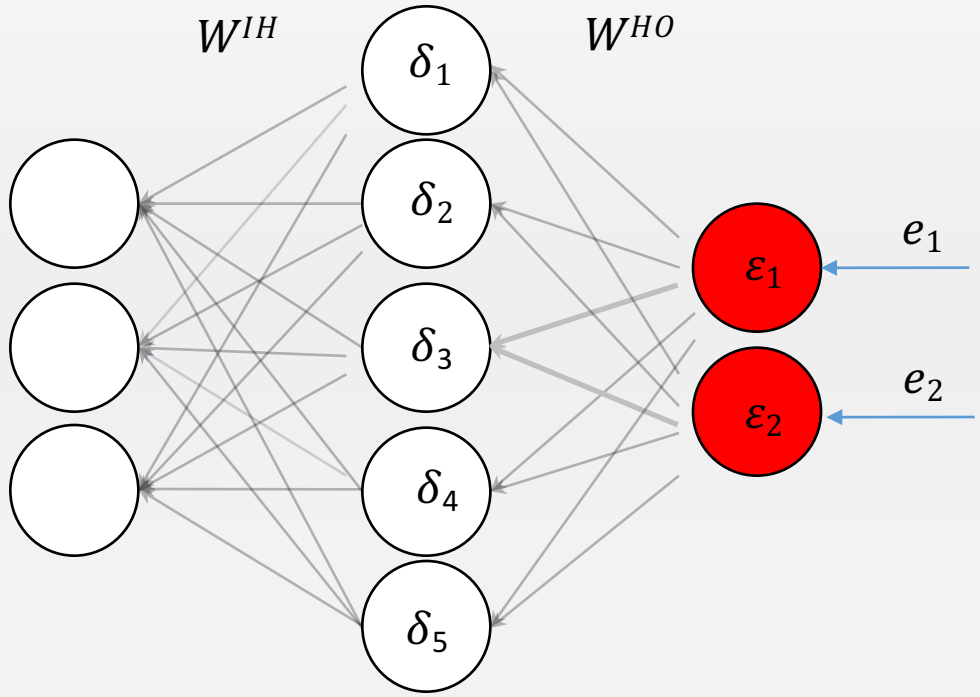

- 反向传播算法可以这样理解：
- 1、将神经网络所有的链接全部反向
- 2、将误差从输出层节点反向沿所有连边传播到下一层节点
- 3、激活函数$\sigma$全部改为：$\sigma'$
- 4、根据连边连通的两个节点的数值和误差调整连边权重

# 单个神经元的误差



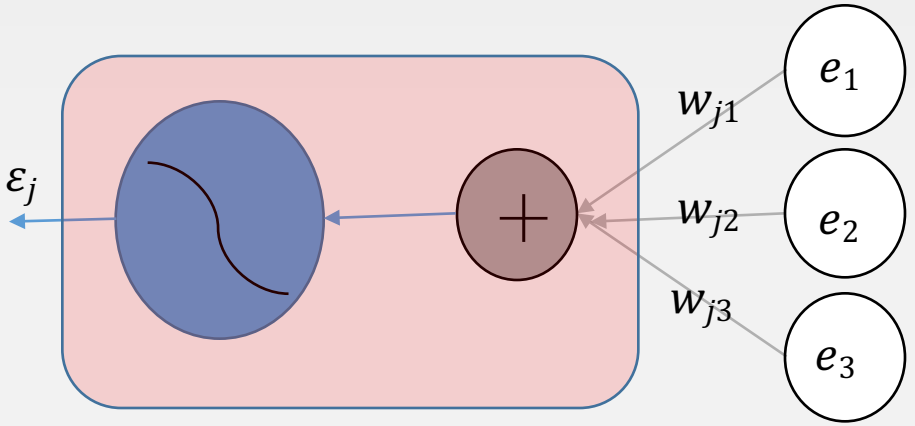

让我们考虑红色节点的误差

# 单个神经元的误差

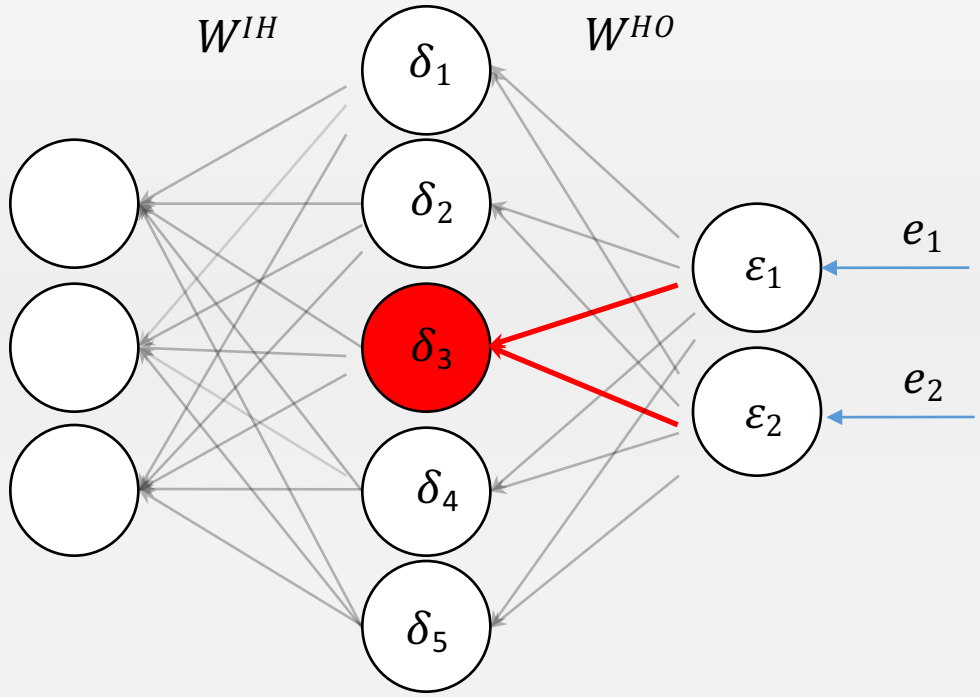



让我们考虑红色节点的误差

$$\varepsilon_i = \sigma'(I_i)e_i, \qquad I_i = \sum_{j=1}^{N_H} w_{ji}^{HO} h_j$$



# 单个神经元的误差

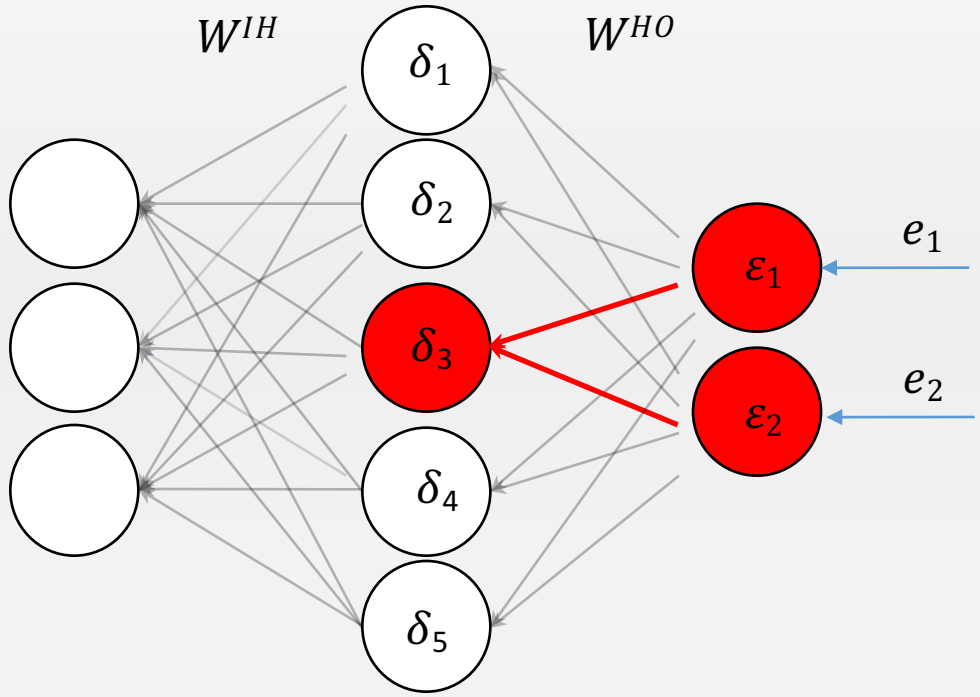



让我们考虑红色节点的误差

$$\varepsilon_i = \sigma'(I_i)e_i, \qquad I_i = \sum_{j=1}^{N_H} w_{ji}^{HO} h_j$$

$$\delta_3 = \sigma'(I_3)(w_{31}^{HO}\varepsilon_1 + w_{32}^{HO}\varepsilon_2) \qquad I_3 = \sum_{i=1}^{N_I} w_{ij}^{IH} x_i$$

# 单个神经元的误差




让我们考虑红色节点的误差

$$\varepsilon_i = \sigma'(I_i)e_i, \qquad I_i = \sum_{j=1}^{N_H} w_{ji}^{HO} h_j$$

$$\delta_3 = \sigma'(I_3)(w_{31}^{HO}\varepsilon_1 + w_{32}^{HO}\varepsilon_2) \qquad I_3 = \sum_{i=1}^{N_I} w_{ij}^{IH} x_i$$

一般情况下

$$\Delta^l = \sigma'(I^l) \odot (W^{l,l+1} \cdot \Delta^{l+1})$$

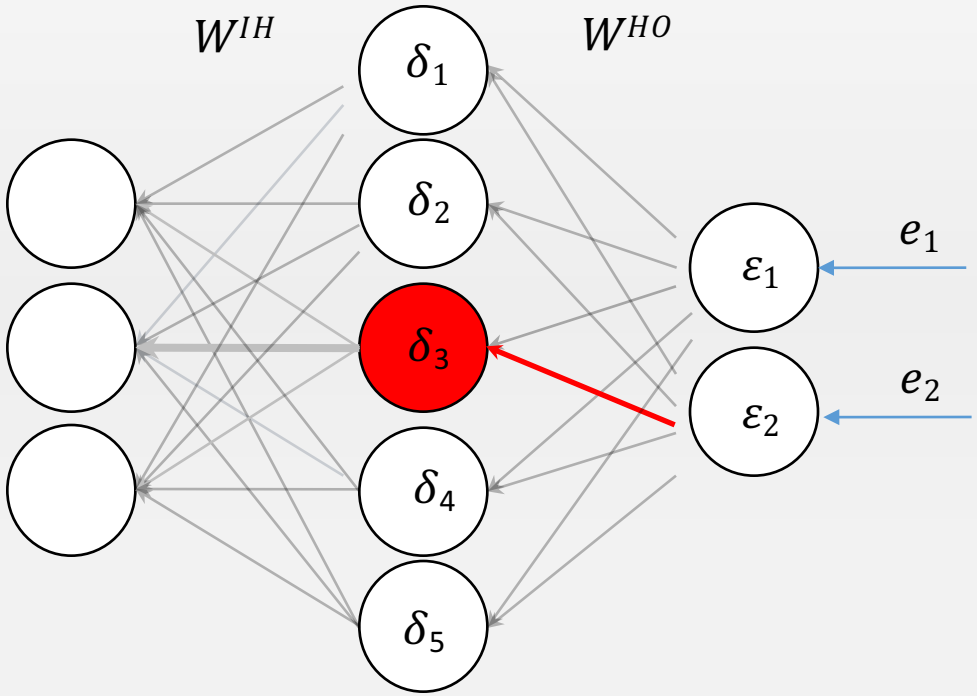
# 矩阵按元素乘积（element-wised product）

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad B = \begin{pmatrix} x & y \\ z & t \end{pmatrix} \qquad A \cdot B = \begin{pmatrix} ax + bz & ay + bt \\ cx + dz & cy + dt \end{pmatrix}$$

$$A \odot B = \begin{pmatrix} ax & by \\ cz & dt \end{pmatrix}$$

在Python中，A*B就会自动按照元素的方式进行乘积

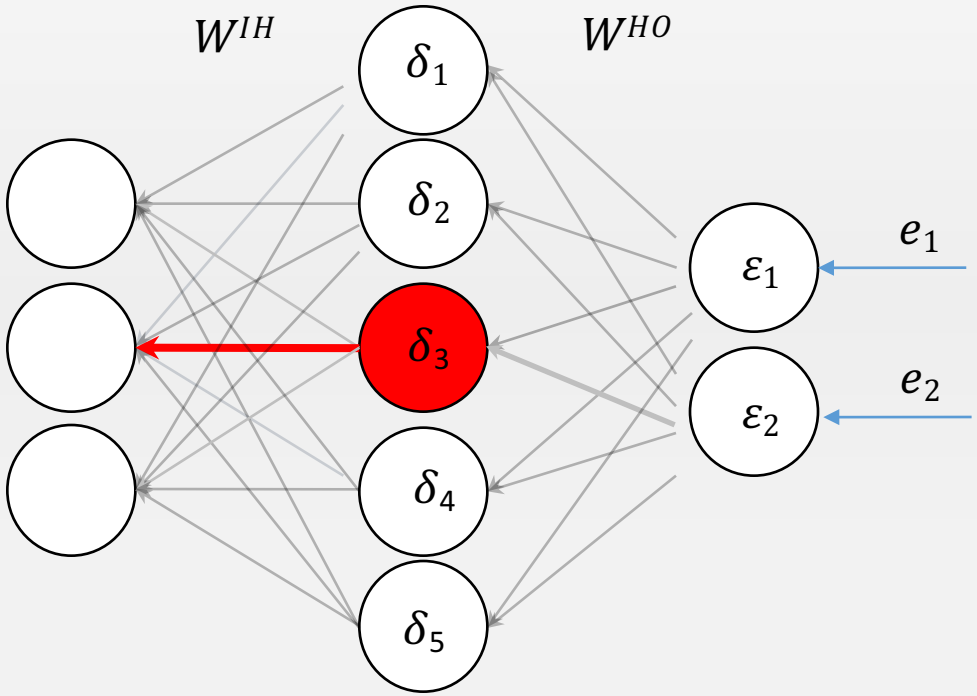
# 权重的更新




让我们考虑红色权重的更新

$$\Delta w_{32}^{HO} = \epsilon_2 h_3$$

# 权重的更新

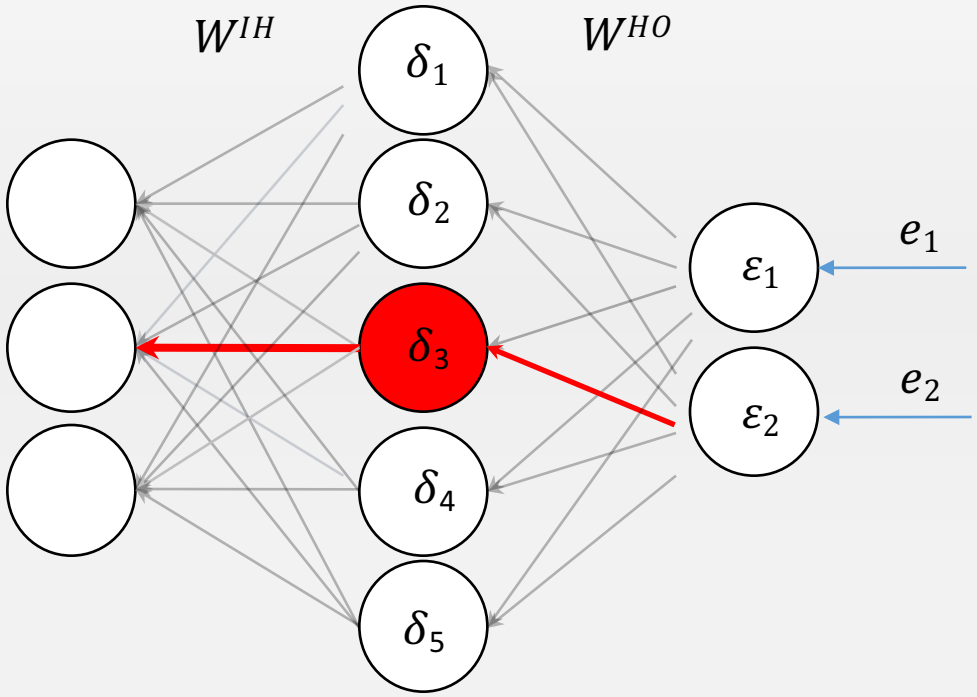



让我们考虑红色权重的更新

$$\Delta w_{32}^{HO} = \epsilon_2 h_3$$

$$\Delta w_{23}^{IH} = \delta_3 x_2$$

# 权重的更新




让我们考虑红色权重的更新

$$\Delta w_{32}^{HO} = \epsilon_2 h_3$$

$$\Delta w_{23}^{IH} = \delta_3 x_2$$

一般情况下

$$\Delta W^{l,l+1} = h^l \otimes \Delta^{l+1}$$

# 科罗内克积(Kroneker product)

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \qquad B = \begin{pmatrix} x & y \\ z & t \end{pmatrix}$$

$$A \otimes B = \begin{pmatrix} a\begin{pmatrix} x & y \\ z & t \end{pmatrix} & b\begin{pmatrix} x & y \\ z & t \end{pmatrix} \\ c\begin{pmatrix} x & y \\ z & t \end{pmatrix} & d\begin{pmatrix} x & y \\ z & t \end{pmatrix} \end{pmatrix} = \begin{pmatrix} ax & ay & bx & by \\ az & at & bz & bt \\ cx & cy & dx & dy \\ cz & ct & dz & dt \end{pmatrix}$$

$$C = \begin{pmatrix} a \\ b \end{pmatrix}, D = (x, y) \qquad C \odot D = C \otimes D = \begin{pmatrix} ax & ay \\ bx & by \end{pmatrix}$$

在Python中，如果C是1*N, D是N*1，则C*D就会自动科罗内克积

# 整个神经网络的运行

- 对于输入的X，逐层进行前向传播
- 得到最后一层输出，计算误差Y-Y*
- 逐层进行误差的反向传播:

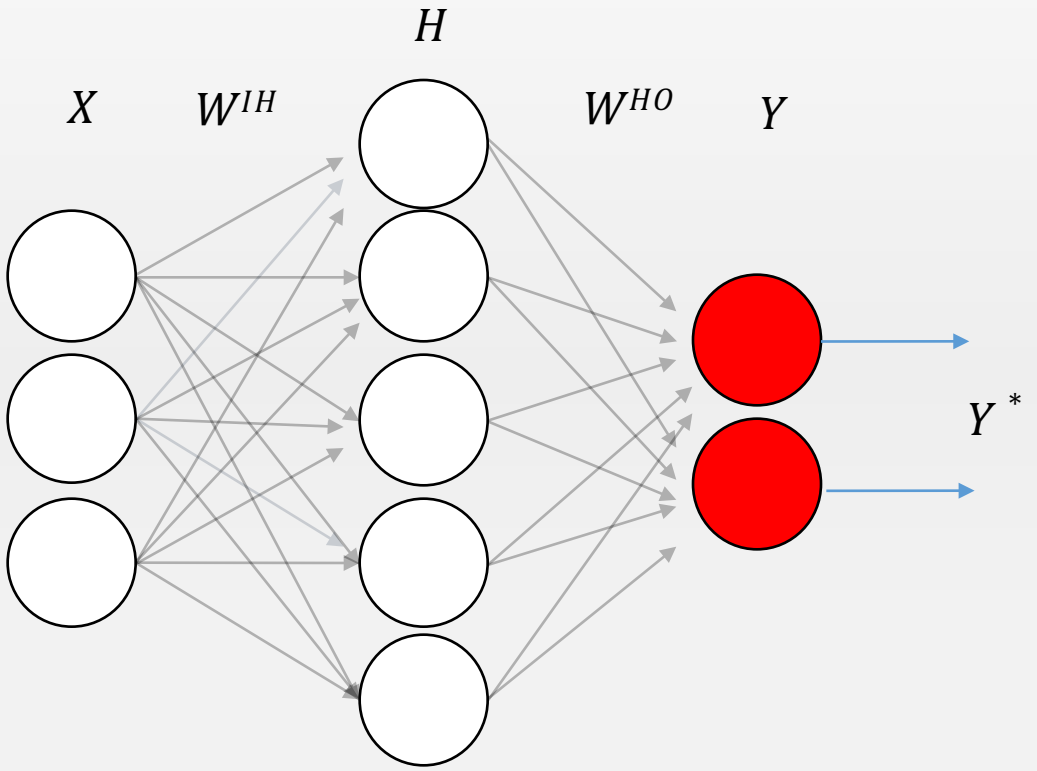
$$\Delta^l = \sigma'(I^l) \odot (W^{l,l+1} \cdot \Delta^{l+1})$$

- 每一层链接权重的调整按如下方式:

$$\Delta W^{l,l+1} = I^l \otimes \Delta^{l+1}$$

$$W'^{l,l+1} = W^{l,l+1} - \alpha \Delta W^{l,l+1}$$

- $\alpha$为学习率，越大则权重调整越大



# 要点重述

- 人工神经元: 线性 + 非线性
- 线性代数不过是一种简化描述的手段
- 反向传播算法是链式求导法则的衍生物
- 神经网络的递归结构使得BP算法具有特殊性:
  - 节点误差反传
  - 权重更新