深度学习基础

BP神经网络

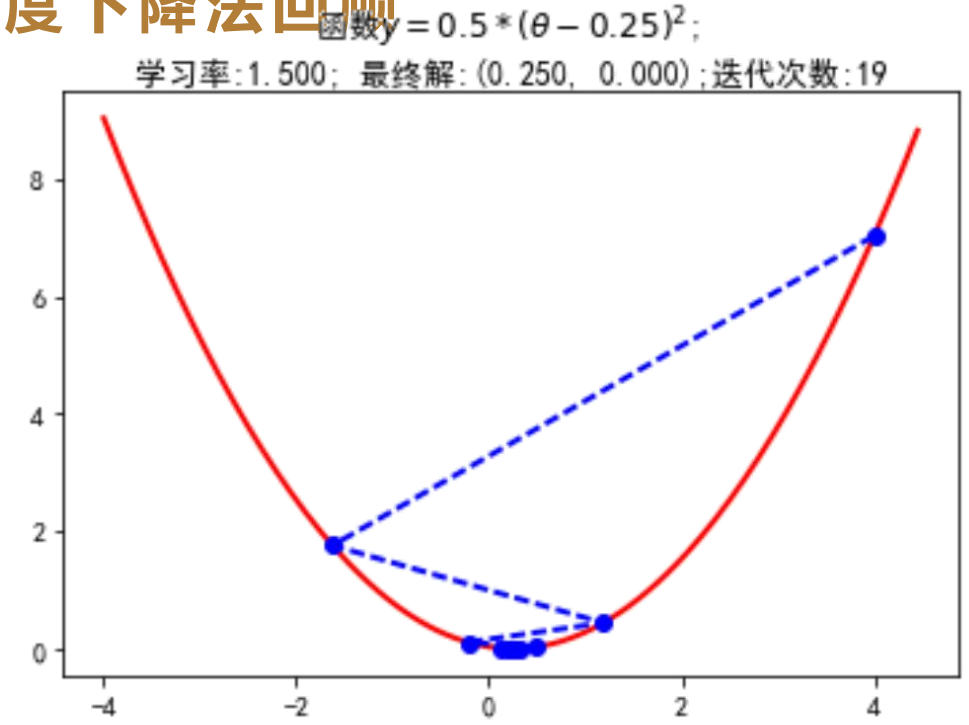
梯度下降法回顾

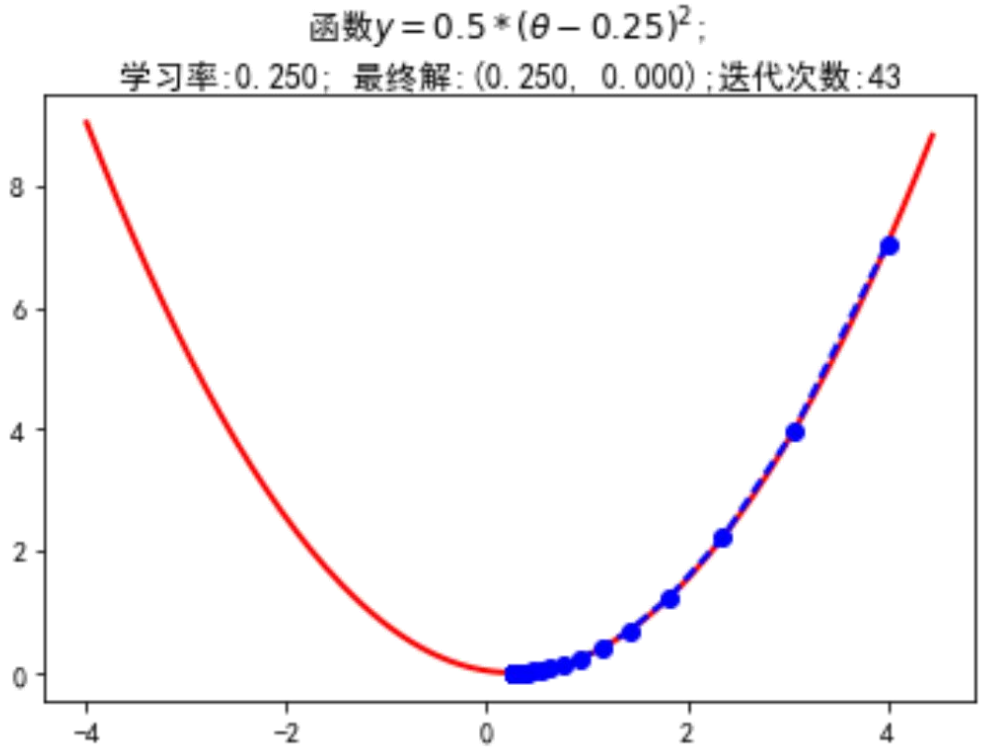
梯度下降法(Gradient Descent， GD)常用于求解**无约束**情况下**凸函数(Convex Function)的极小值**，是一种迭代类型的算法，因为凸函数只有一个极值点，故求解出来的极小值点就是函数的**最小值点**。

$$J(\theta)=\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right)-y^{(i)}\right)^2$$

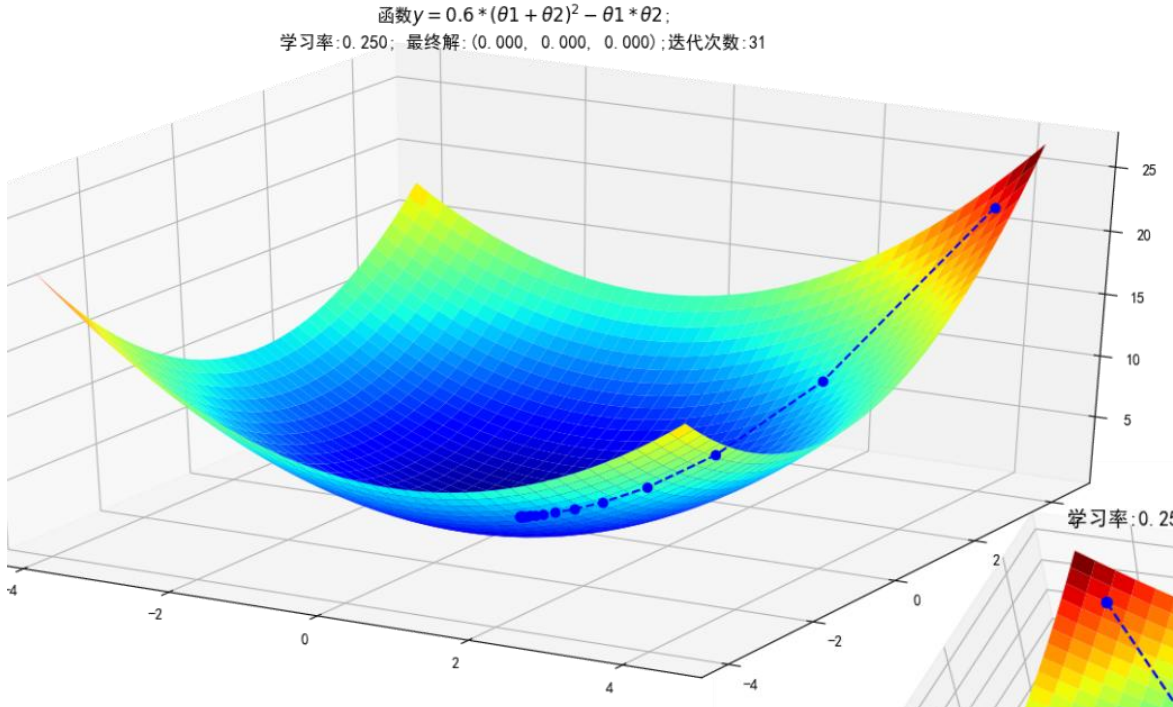
$$\theta^*=\arg\min_{\theta} J(\theta)$$

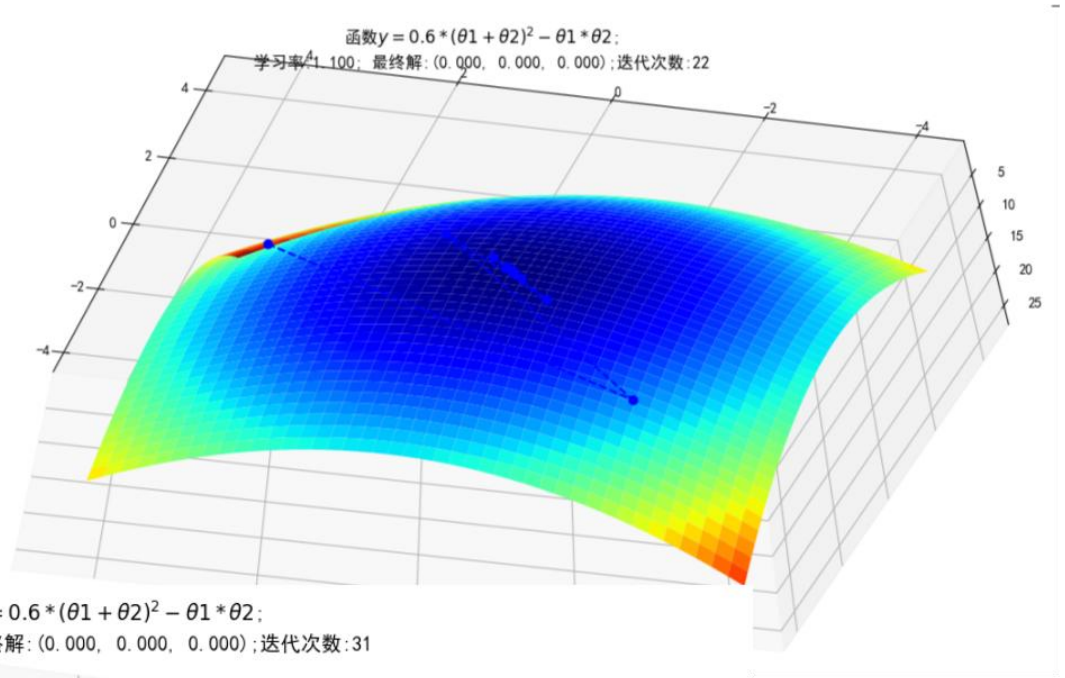
梯度下降法回顾

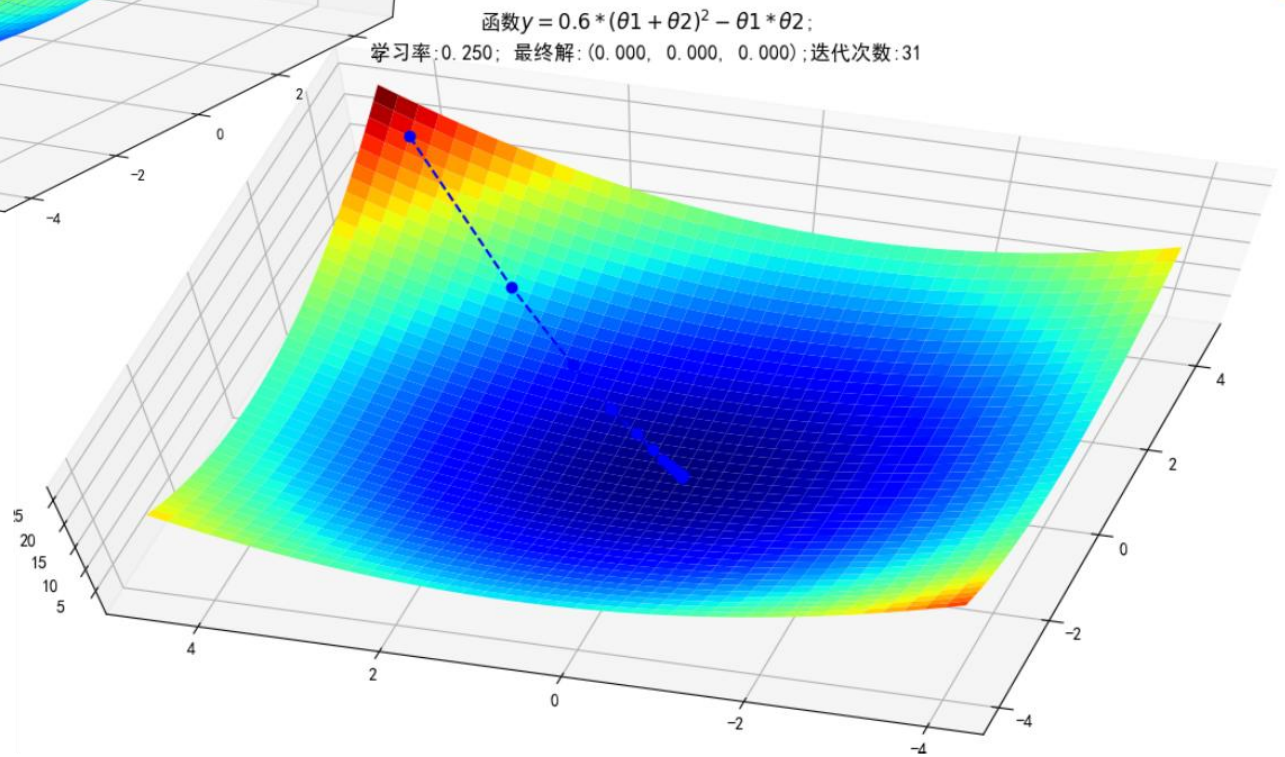

函数$y=0.5*(\theta-0.25)^2$;
学习率:1.500; 最终解:(0.250, 0.000);迭代次数:19
8
6
4
2
0
-4
-2
0
2
4


函数$y=0.5*(\theta-0.25)^2$;
学习率:0.250; 最终解:(0.250, 0.000);迭代次数:43
8
6
4
2
0
-4
-2
0
2
4

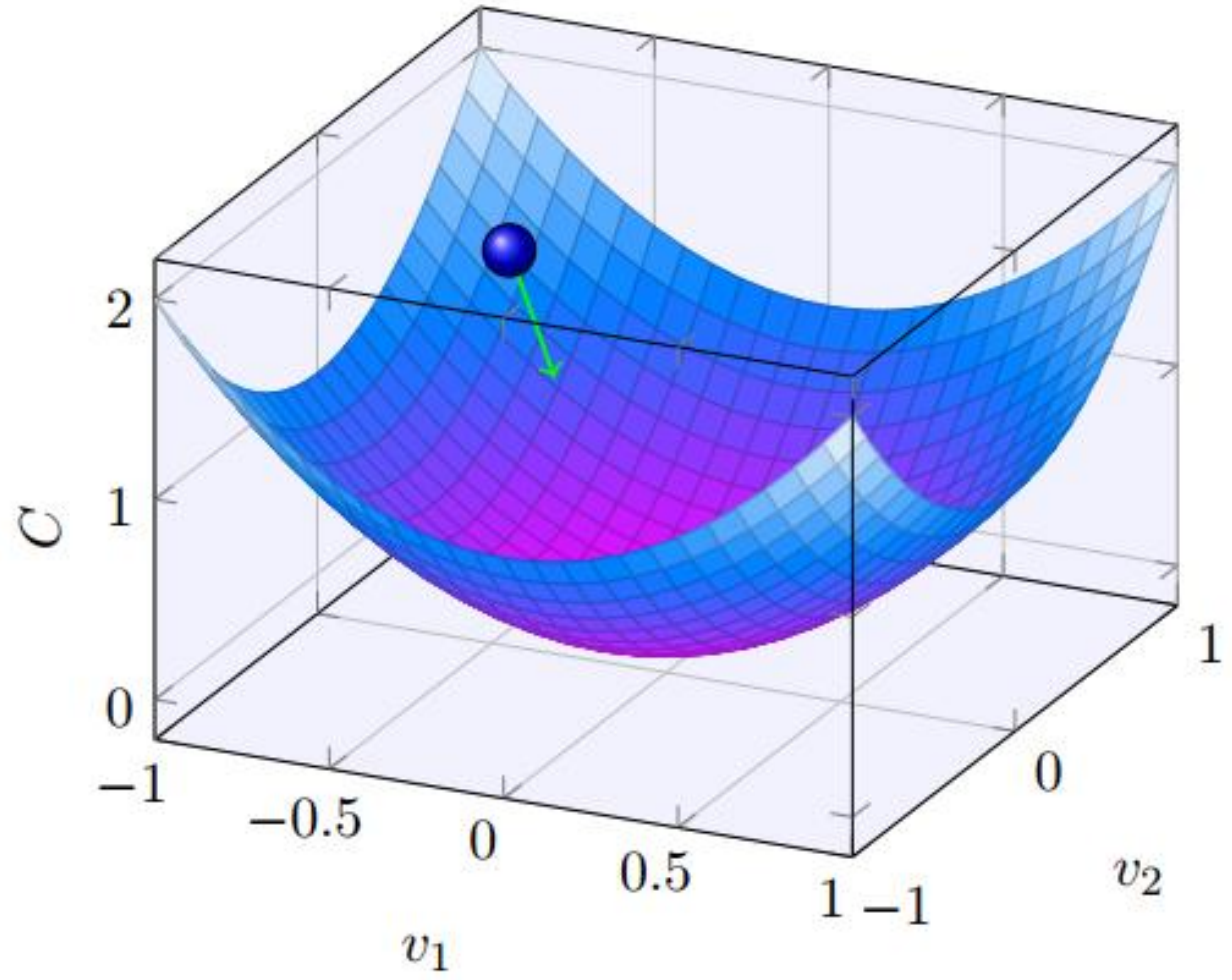
梯度下降法回顾


函数$y=0.6*(\theta1+\theta2)^2-\theta1*\theta2$;
学习率:0.250; 最终解:(0.000, 0.000, 0.000);迭代次数:31


函数$y=0.6*(\theta1+\theta2)^2-\theta1*\theta2$;
学习率:1.100; 最终解:(0.000, 0.000, 0.000);迭代次数:22


函数$y=0.6*(\theta1+\theta2)^2-\theta1*\theta2$;
学习率:0.250; 最终解:(0.000, 0.000, 0.000);迭代次数:31

梯度下降法回顾



线性回归回顾

最基础的一种机器学习算法，常用来拟合线性关系的模型;

$$y^{(i)} = \theta^T x^{(i)} + \varepsilon^{(i)}$$

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$$

$$\theta = \theta - \alpha \bullet \frac{\partial J(\theta)}{\partial \theta}$$

Logistic回归回顾

基于线性回归扩展的一种分类算法。

$$p = h_\theta(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$loss = -\ell(\theta) = -\sum_{i=1}^{m}\left(y^{(i)} \ln h_\theta(x^{(i)}) + (1-y^{(i)}) \ln(1-h_\theta(x^{(i)}))\right)$$

$$= \sum_{i=1}^{m}\left[-y^{(i)} \ln(h_\theta(x^{(i)})) - (1-y^{(i)}) \ln(1-h_\theta(x^{(i)}))\right]$$

$$g'(z) = \left(\frac{1}{1+e^{-z}}\right)' = \frac{e^{-z}}{(1+e^{-z})^2}$$

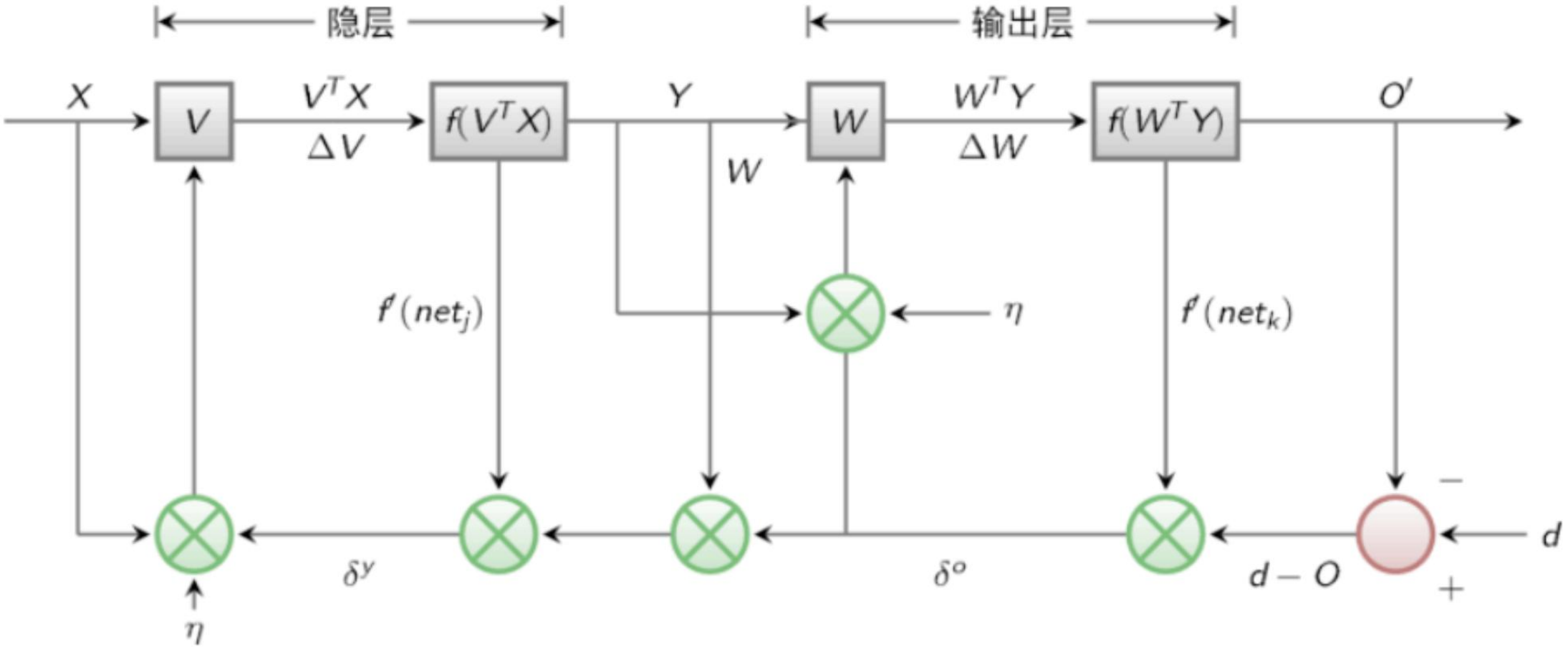
$$= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$= g(z) \cdot (1-g(z))$$

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)}$$

神经网络之BP算法

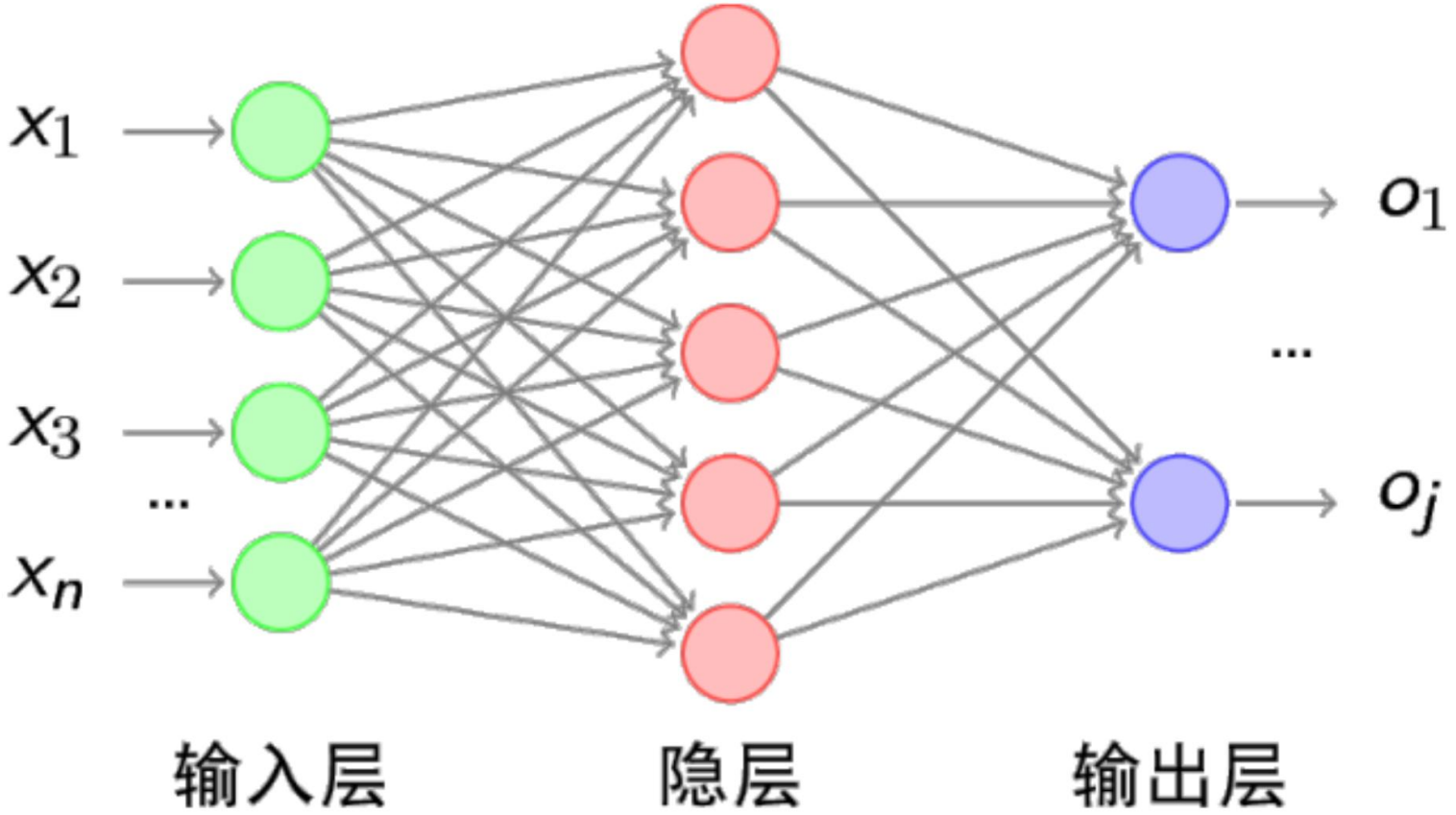
神经网络的一种求解W的算法，分为信号“正向传播(FP)”求损失，“反向传播(BP)”回传误差；根据误差值修改每层的权重，继续迭代。



神经网络之BP算法

BP算法也叫做δ算法

以三层的感知器为例（假定现在隐层和输出层均存在相同类型的激活函数）



神经网络之BP算法

输出层误差

$$E=\frac{1}{2}(d-O)^2=\frac{1}{2}\sum_{k=1}^{\ell}(d_k-O_k)^2$$

隐层的误差

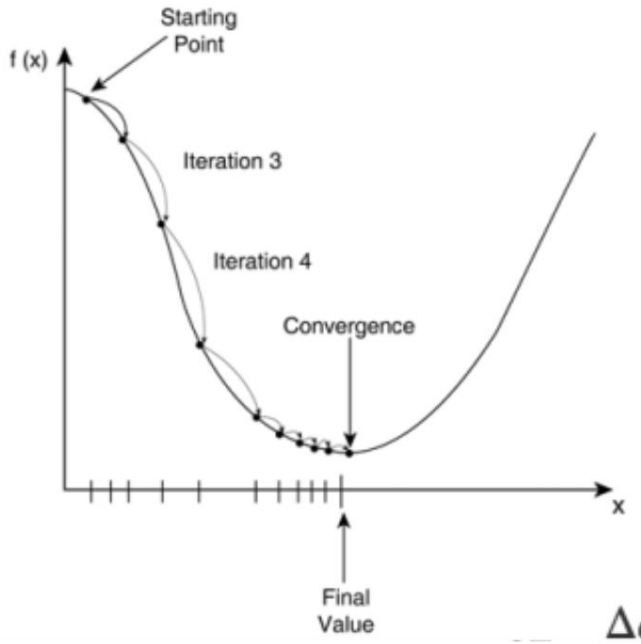
$$E=\frac{1}{2}\sum_{k=1}^{\ell}(d_k-f(net_k))^2=\frac{1}{2}\sum_{k=1}^{\ell}\left(d_k-f\left(\sum_{j=1}^{m}w_{jk}y_j\right)\right)^2$$

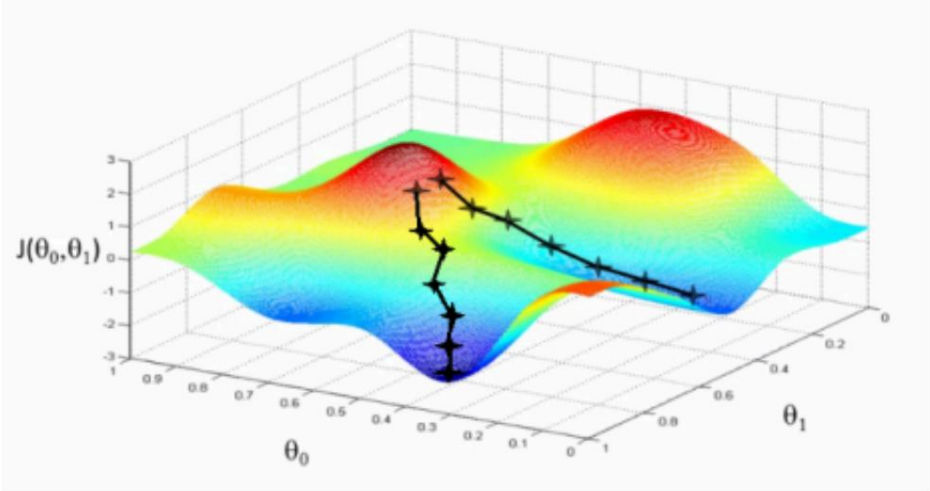
输入层误差

$$E=\frac{1}{2}\sum_{k=1}^{\ell}\left(d_k-f\left[\sum_{j=0}^{m}w_{jk}f(net_j)\right]\right)^2=\frac{1}{2}\sum_{k=1}^{\ell}\left(d_k-f\left[\sum_{j=0}^{m}w_{jk}f\left(\sum_{i=1}^{n}\upsilon_{ij}x_i\right)\right]\right)^2$$

神经网络之SGD

误差E有了，那么为了使误差越来越小，可以采用随机梯度下降的方式进行ω和υ的求解，即求得ω和υ使得误差E最小。

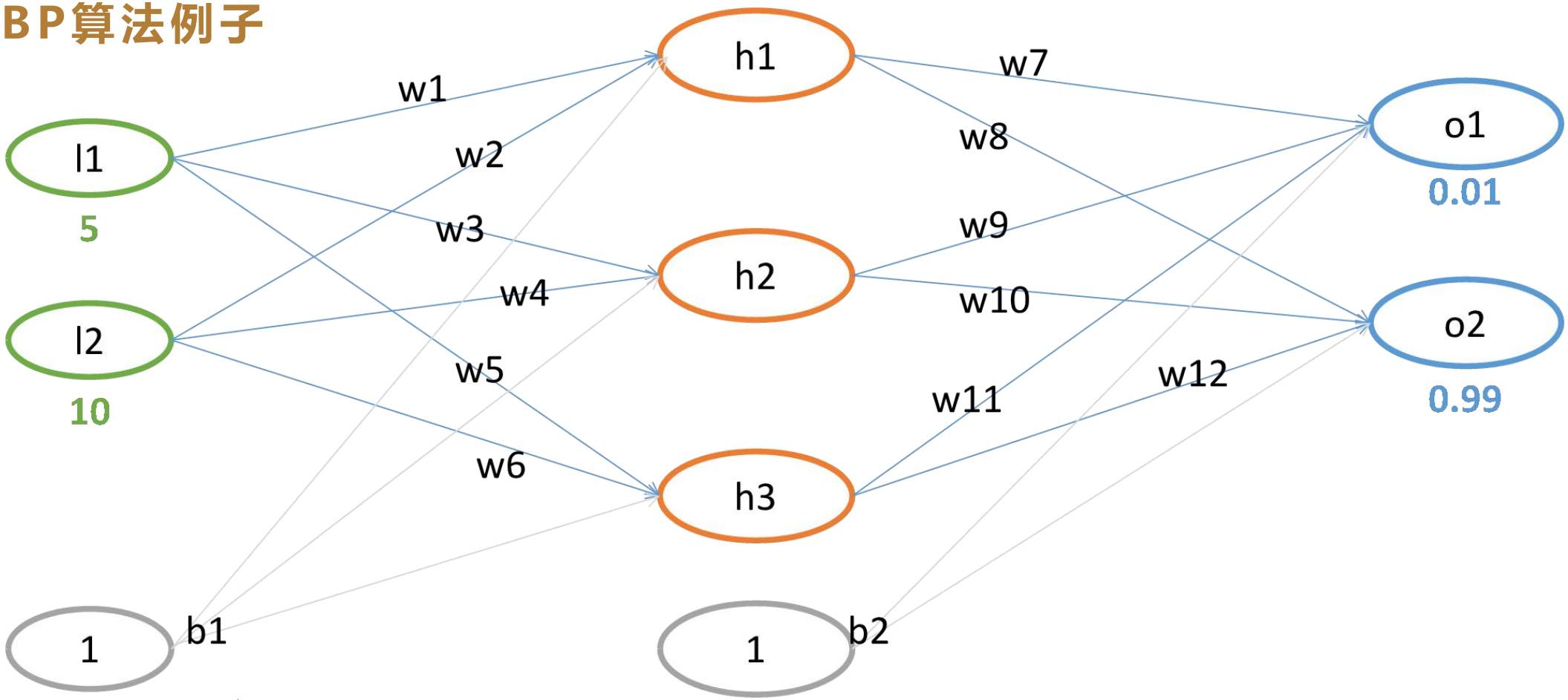




$$\Delta\omega_{j\kappa} = -\eta\frac{\partial E}{\partial\omega_{j\kappa}} j = 0, 1, 2, \ldots, m; \qquad \kappa = 1, 2, \ldots, \ell$$

$$\Delta v_{ij} = -\eta\frac{\partial E}{\partial v_{ij}} i = 0, 1, 2, \ldots, n; \qquad j = 1, 2, \ldots, m$$

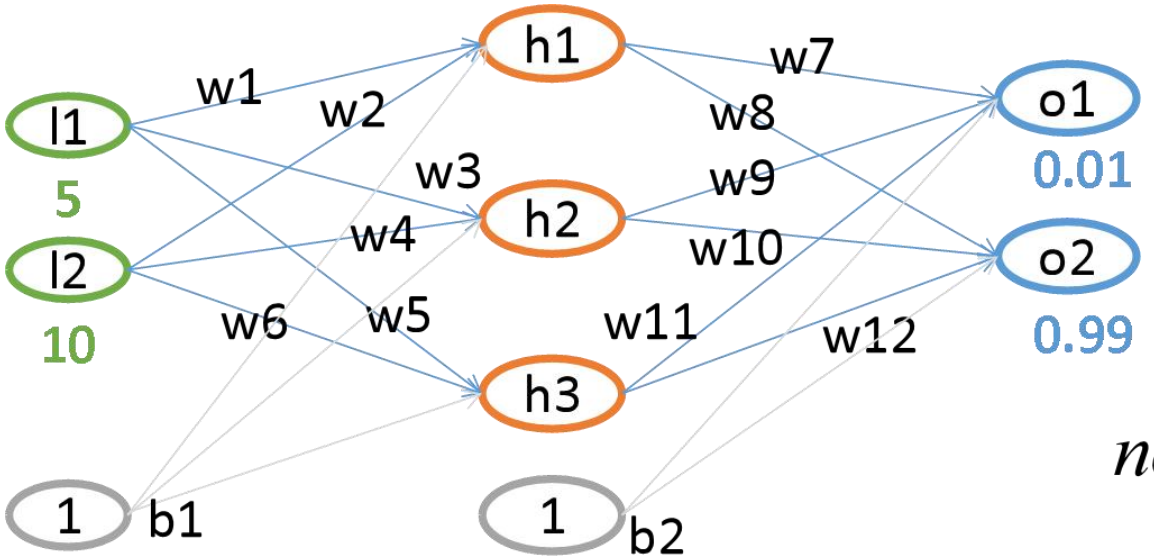
BP算法例子



$$w = (0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65)$$

$$b = (0.35,0.65)$$

BP算法例子-FP过程



$$net_{h1} = w_1 * l_1 + w_2 * l_2 + b_1 * 1$$

$$net_{h1} = 0.1*5 + 0.15*10 + 0.35*1 = 2.35$$

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-2.35}} = 0.912934$$

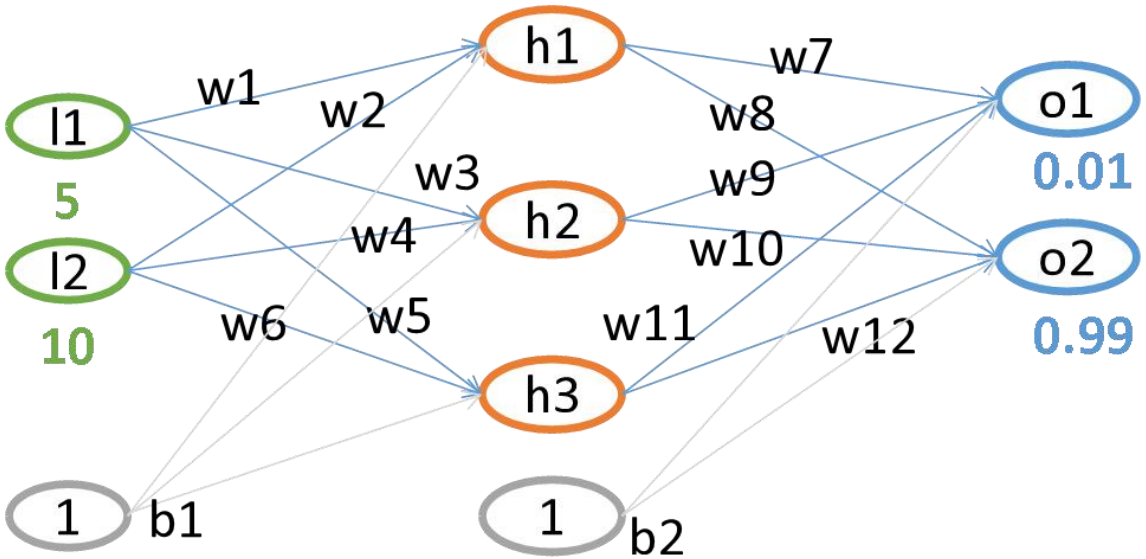
$$b = (0.35, 0.65)$$

$$w = \begin{pmatrix} 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, \\ 0.4, 0.45, 0.5, 0.55, 0.6, 0.65 \end{pmatrix}$$

$$out_{h2} = 0.979164$$

$$out_{h3} = 0.995275$$

BP算法例子-FP过程



$$net_{o1} = w_7 * out_{h1} + w_9 * out_{h2} + w_{11} * out_{h3} + b_2 * 1$$

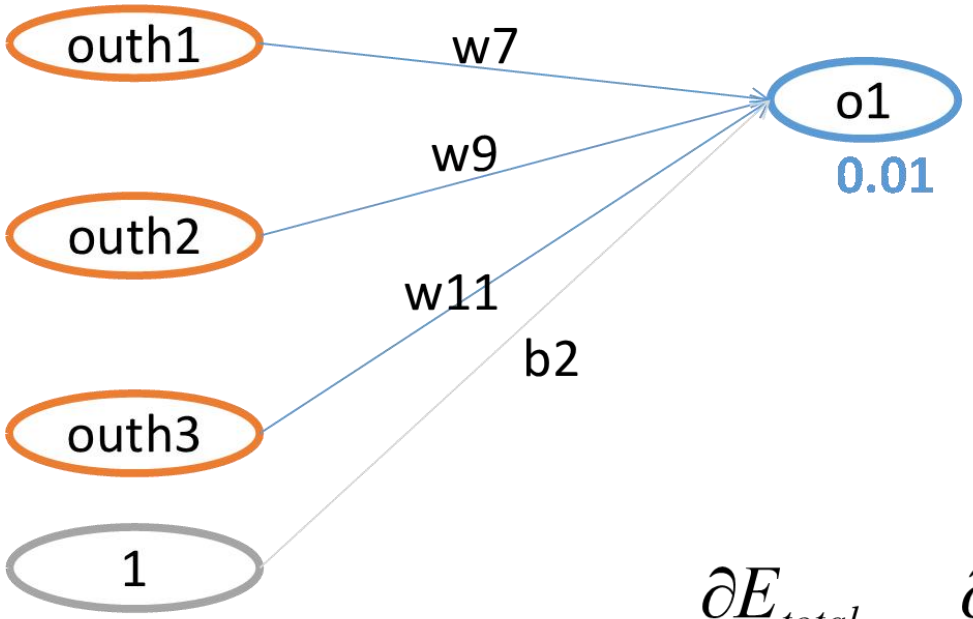
$$net_{o1} = 0.4 * 0.912934 + 0.5 * 0.979164 + 0.6 * 0.995275 = 2.1019206$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-2.1019206}} = 0.891090 \qquad out_{o2} = 0.904330$$

NOTE:正常的神经网络，最后一层(输出层)对应的输出结果是不进行激活操作的(就直接将全连接结果输出)

$$E_{total} = E_{o1} + E_{o2} = \frac{1}{2}(0.01 - 0.891090)^2 + \frac{1}{2}(0.99 - 0.904330)^2 = 0.391829$$

BP算法例子-BP过程(W7)



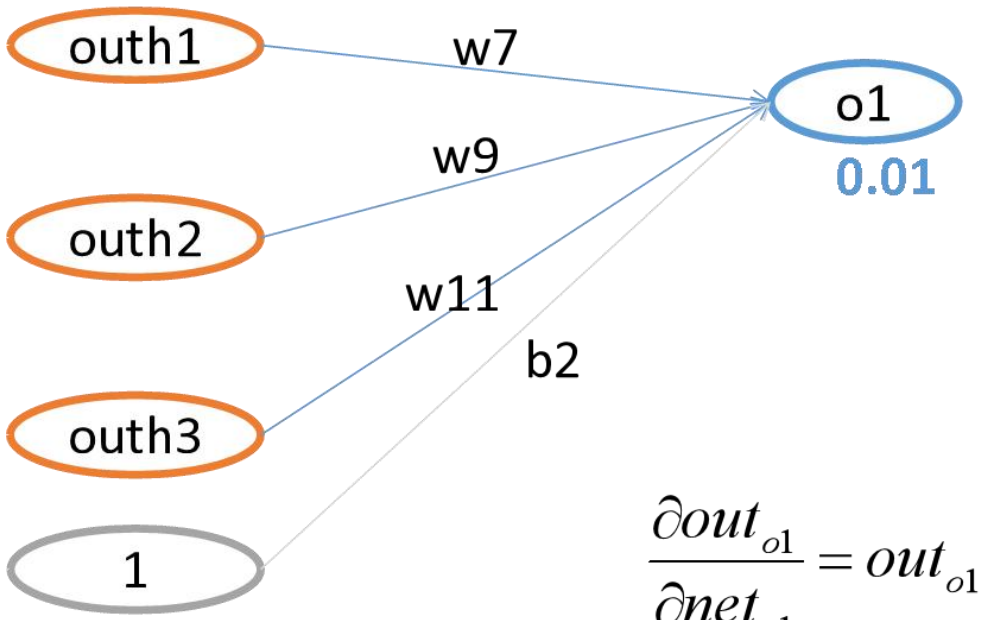
$$E_{o1} = \frac{1}{2}\left(\text{target}_{o1} - out_{o1}\right)^2$$

$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_7}$$

$$\frac{\partial E_{total}}{\partial out_{o1}} = 2 * \frac{1}{2}\left(\text{target}_{o1} - out_{o1}\right)^{2-1} * -1 + 0 = -\left(0.01 - 0.891090\right) = 0.88109$$

BP算法例子-BP过程(W7)

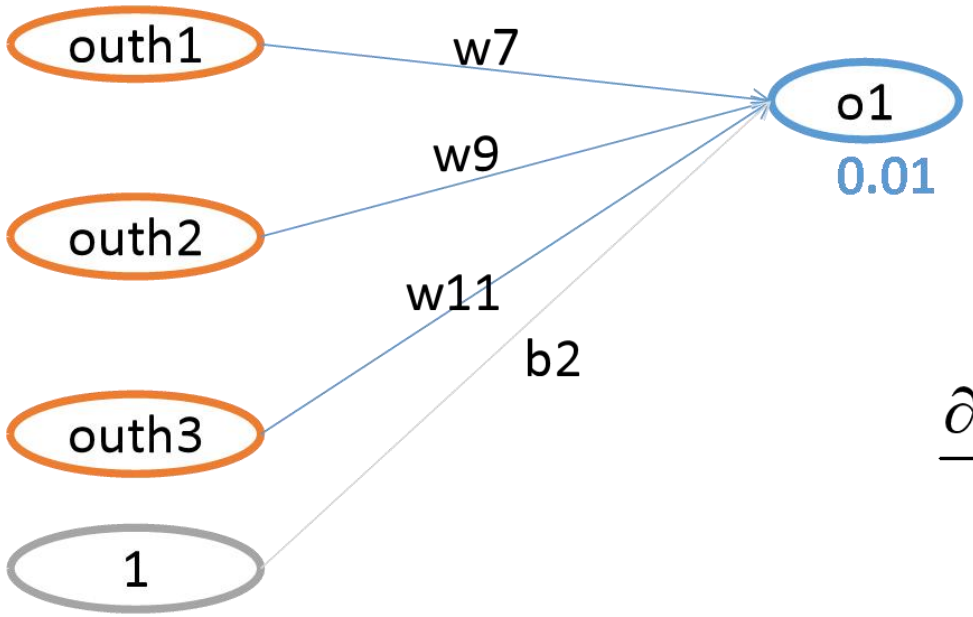


$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1-out_{o1}) = 0.891090(1-0.891090) = 0.097049$$

$$\text{out}_{o1}' = \frac{e^{-net}}{(1+e^{-net})^2} = \frac{1+e^{-net}-1}{(1+e^{-net})^2} = \frac{1}{1+e^{-net}} - \frac{1}{(1+e^{-net})^2} = out_{o1}(1-out_{o1})$$

BP算法例子-BP过程(W7)

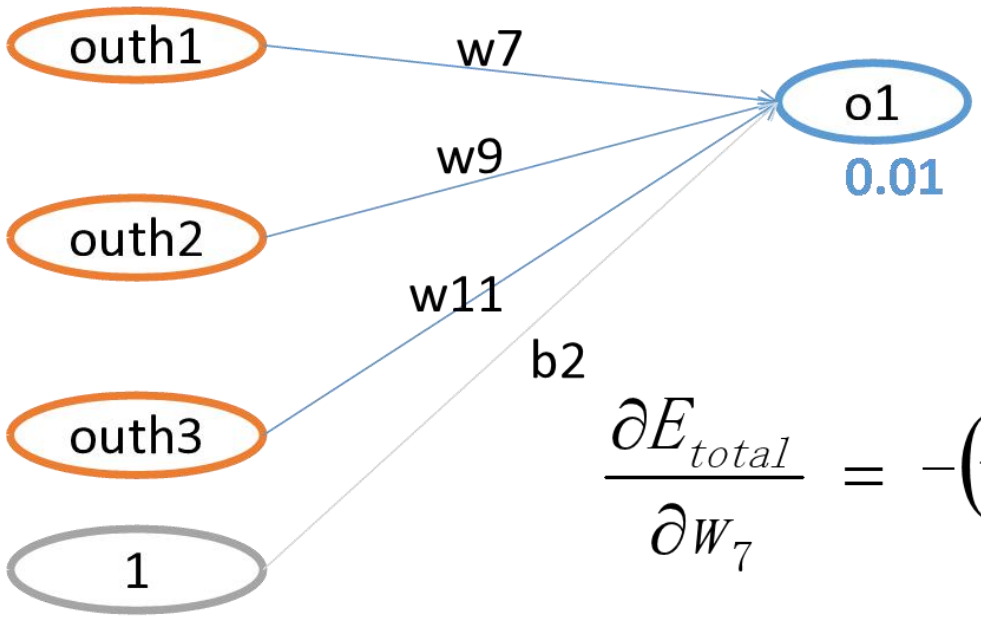


$$\frac{\partial net_{o1}}{\partial w_7} = 1 * out_{h1} * w_7^{(1-1)} + 0 + 0 + 0 = 0.912934$$

$$net_{o1} = w_7 * out_{h1} + w_9 * out_{h2} + w_{11} * out_{h3} + b_2 * 1$$

$$\frac{\partial E_{total}}{\partial w_7} = 0.88109 * 0.097049 * 0.912934 = 0.078064$$

BP算法例子-BP过程(W7...)



$$E_{o1} = \frac{1}{2}(\text{target}_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial w_7} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_7}$$

$$\frac{\partial E_{total}}{\partial w_7} = -(\text{target} - out_{o1}) * out_{o1} * (1 - out_{o1}) * out_{h1}$$

$$w_7^{+} = w_7 + \Delta w_7 = w_7 - \eta \frac{\partial E_{total}}{\partial w_7} = 0.4 - 0.5 * 0.078064 = 0.360968$$

$$w_8^{+} = 0.453383 \qquad w_9^{+} = 0.458137$$

$$w_{10}^{+} = 0.553629 \qquad w_{11}^{+} = 0.557448 \qquad w_{12}^{+} = 0.653688$$

BP算法例子-BP过程(W1)

$$\frac{\partial E_{total}}{\partial w_1}=\frac{\partial E_{total}}{\partial out_{h1}}*\frac{\partial out_{h1}}{\partial net_{h1}}*\frac{\partial net_{h1}}{\partial w_1}=\left(\frac{\partial E_{o1}}{\partial out_{h1}}+\frac{\partial E_{o2}}{\partial out_{h1}}\right)*\frac{\partial out_{h1}}{\partial net_{h1}}*\frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{o1}}{\partial out_{h1}}=\frac{\partial E_{o1}}{\partial out_{o1}}*\frac{\partial out_{o1}}{\partial net_{o1}}*\frac{\partial net_{o1}}{\partial out_{h1}}=-\left(\text{target}_{o1}-out_{o1}\right)*out_{o1}*\left(1-out_{o1}\right)*w_7$$

note：正常逻辑中，在求解梯度使到了w7的数值，一般使用更新前的

$$\frac{\partial E_{o1}}{\partial out_{h1}}=-(0.01-0.891090)*0.891090*(1-0.891090)*0.360968=0.030866$$

BP算法例子-BP过程(W1)

$$\frac{\partial E_{o2}}{\partial out_{h1}} = \frac{\partial E_{o2}}{\partial out_{o2}} * \frac{\partial out_{o2}}{\partial net_{o2}} * \frac{\partial net_{o2}}{\partial out_{h1}} = -(\text{target}_{o2} - out_{o2}) * out_{o2} * (1 - out_{o2}) * w8$$

$$\frac{\partial E_{total}}{\partial w_1} = 0.011204$$

$$w_1^{+} = w_1 + \Delta w_1 = w_1 - \eta \frac{\partial E_{total}}{\partial w_1} = 0.1 - 0.5 * 0.011204 = 0.094534$$

BP算法例子-BP过程

$$w_1^+ = 0.094534 \qquad w_2^+ = 0.139069 \qquad w_3^+ = 0.198211$$

$$w_4^+ = 0.246422 \qquad w_5^+ = 0.299497 \qquad w_6^+ = 0.348993$$

$$w_7^+ = 0.360968 \qquad w_8^+ = 0.453383 \qquad w_9^+ = 0.458137$$

$$w_{10}^+ = 0.553629 \qquad w_{11}^+ = 0.557448 \qquad w_{12}^+ = 0.653688$$

$$b_1 = 0.35 \qquad b_2 = 0.65$$

BP算法例子-多次迭代效果

第10 次迭代结果: $O = (0.662866, 0.908195)$

第100次迭代结果: $O = (0.073889, 0.945864)$

第1000次迭代结果: $O = (0.022971, 0.977675)$

$$w^0 = \begin{pmatrix} 0.1, 0.15, 0.2, 0.25, \\ 0.3, 0.35, 0.4, 0.45, \\ 0.5, 0.55, 0.6, 0.65 \end{pmatrix} \qquad w^{1000} = \begin{pmatrix} 0.214925, & 0.379850, & 0.262855, \\ 0.375711, & 0.323201, & 0.396402, \\ -1.48972, & 0.941715, & -1.50182, \\ 1.049019, & -1.42756, & 1.151881 \end{pmatrix}$$

THANKS!