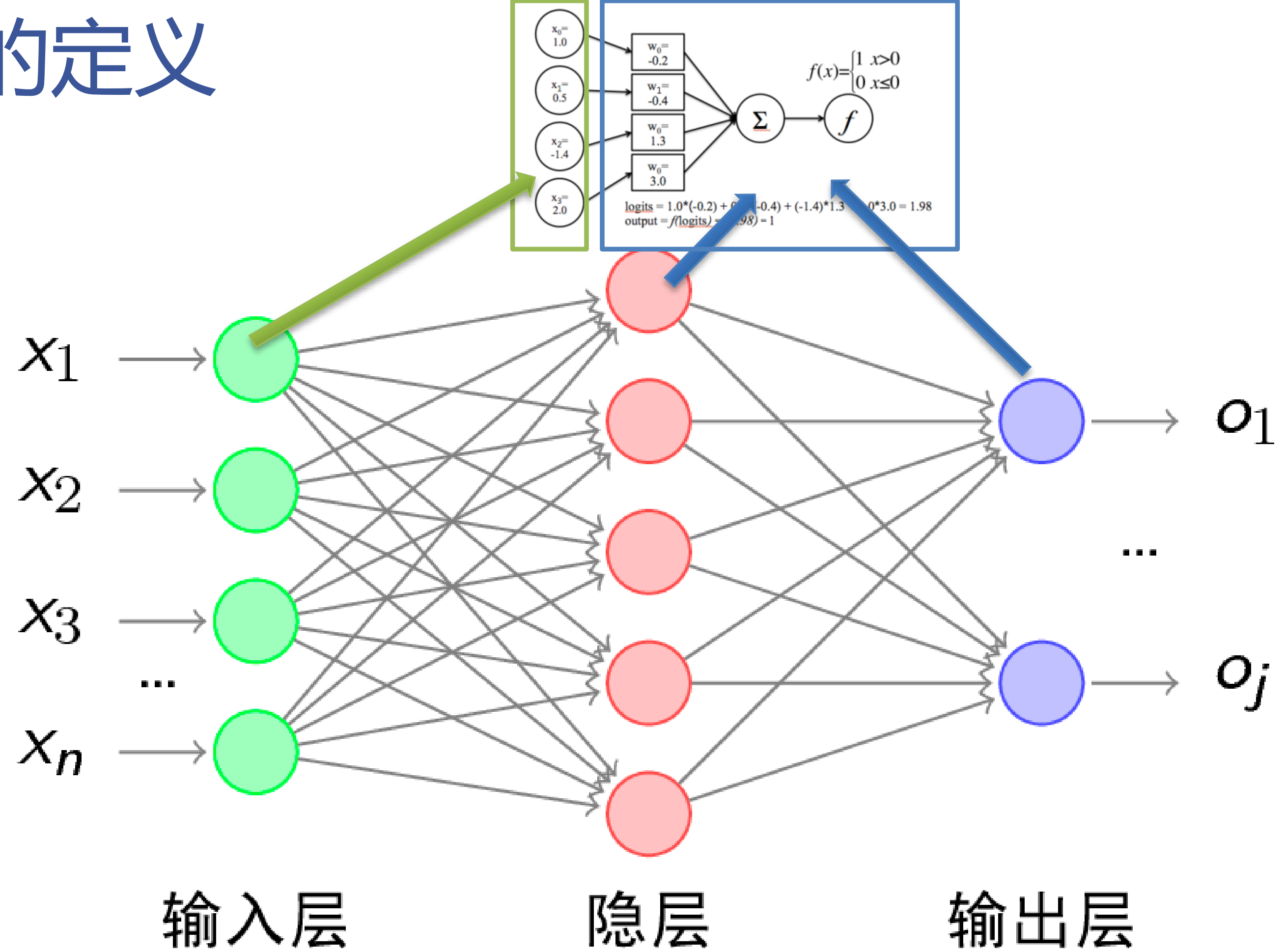
多层感知机

现代神经网络的原型

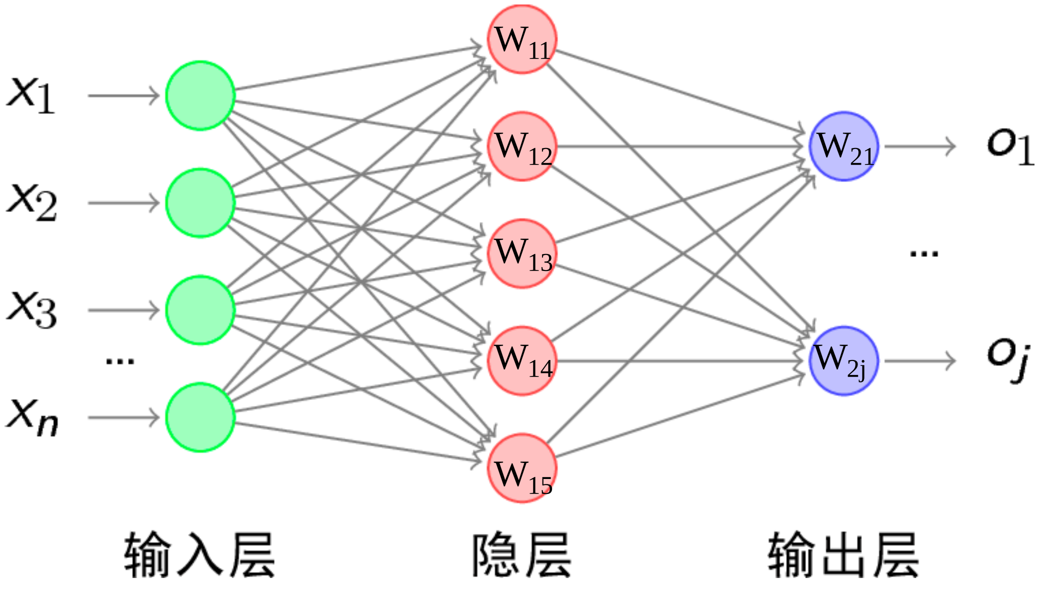
多层感知机

- 隐层的引入
- 激活函数的改变
- 反向传播算法
- 优化算法
- 其实它就是神经网络

▶ 隐层的定义


x_0= 1.0
x_1= 0.5
x_2= -1.4
x_3= 2.0
w_0= -0.2
w_1= -0.4
w_0= 1.3
w_0= 3.0
Σ
f
$f(x)=\begin{cases}1 & x>0\\ 0 & x\le 0\end{cases}$
logits = 1.0*(-0.2) +
-0.4) + (-1.4)*1.3
0*3.0 = 1.98
output = f(logits)
98) = 1
x_1
x_2
x_3
...
x_n
o_1
...
o_j
输入层
隐层
输出层

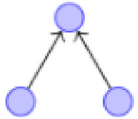
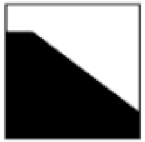
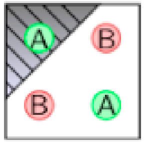
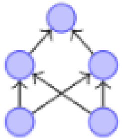
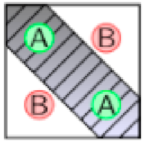
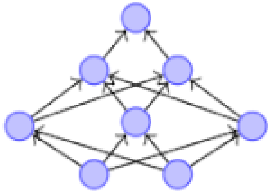
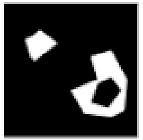
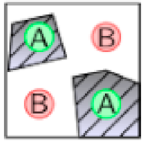
隐层



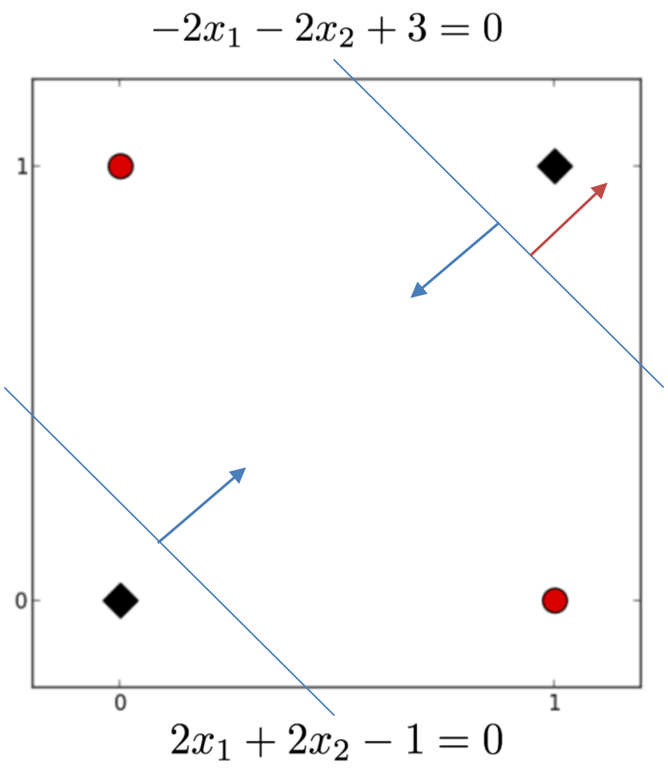
- 所有同一层的神经元都与上一层每个输出相连
- 同一层的神经元之间不互相连接
- 各神经元的输出为数值

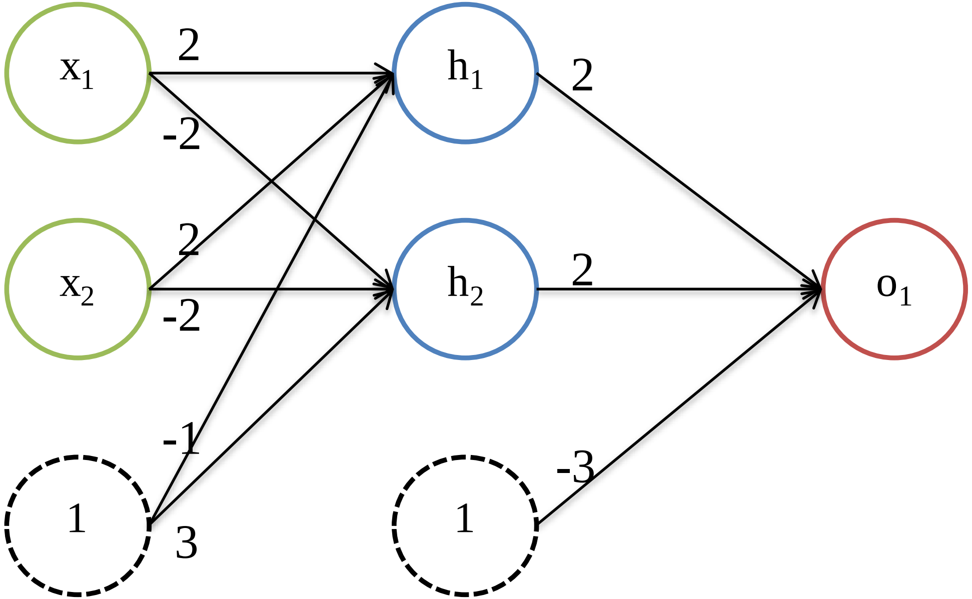
- 每一个输入数据为n维向量
- $w_{ij}^{(l)}$ 为第 l 层第 i 个神经元与第j个输入相连的权重
- 第l层中每个神经元的权重为k维向量，k为（l-1）层神经元数量
- 第l个隐层中所有p个神经元的权重组成矩阵W^l
- $W^{(l)} \in \mathbb{R}\left[p,k\right]$
- $y^{(l)} = output^{(l)} = f(W \cdot output^{(l-1)} + b)$

隐层

结构	决策区域类型	区域形状	异或问题
无隐层 	由一超平面分成两个		
单隐层 	开凸区域或闭凸区域		
双隐层 	任意形状（其复杂度由单元数目确定）		

▶ 隐层



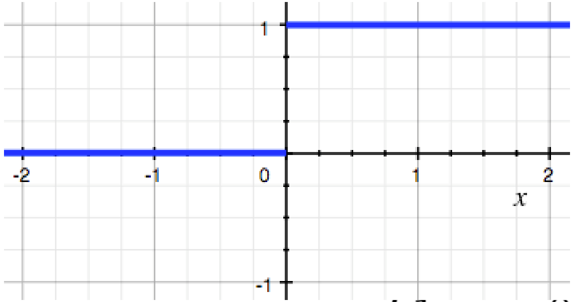


$$h_1 = f(2x_1 + 2x_2 - 1)$$

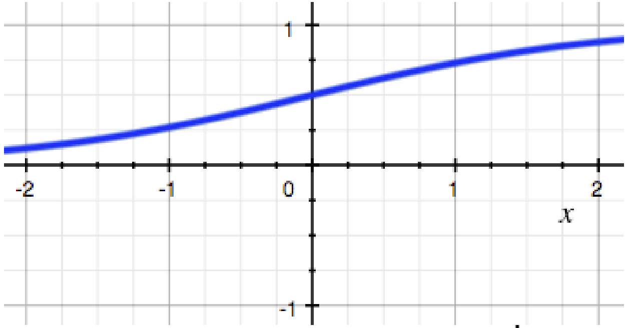
$$h_2 = f(-2x_1 + -2x_2 + 3)$$

$$o_1 = f(2h_1 + 2h_2 - 3)$$

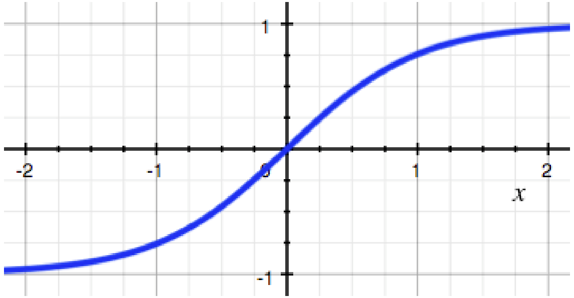
▶ 新的激活函数



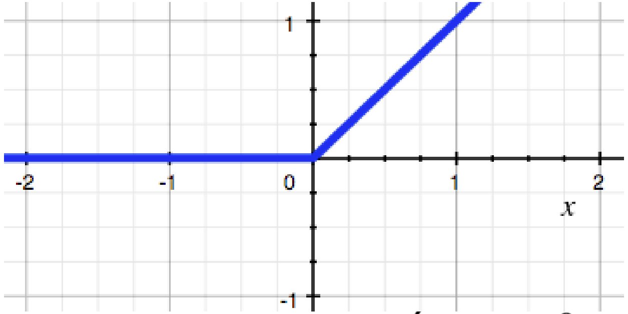
$$f(x) = unit(x) = \begin{cases} 1, x > 0 \\ 0, x \le 0 \end{cases}$$



$$f(x) = sigmoid(x) = \frac{1}{1 + e^{-x}}$$



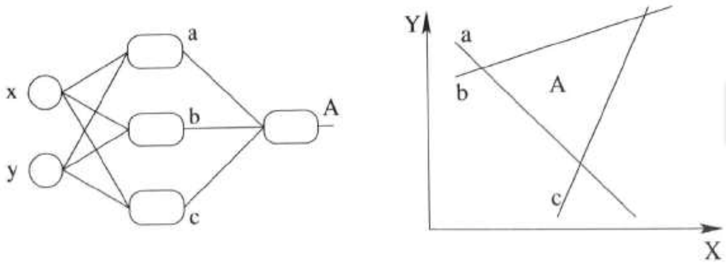
$$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



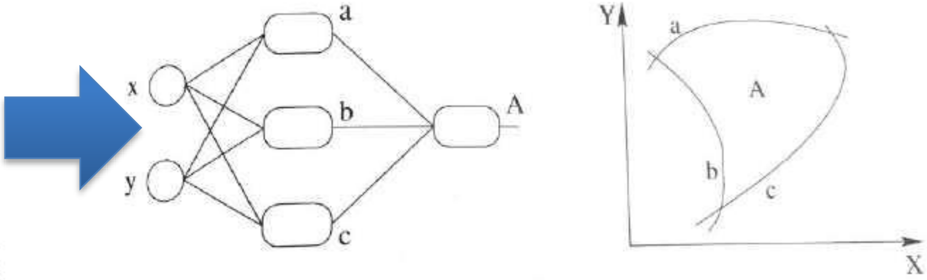
$$f(x) = ReLU(x) = \begin{cases} x, x > 0 \\ 0, x \le 0 \end{cases}$$

▶ 激活函数带来了什么？

- 非线性

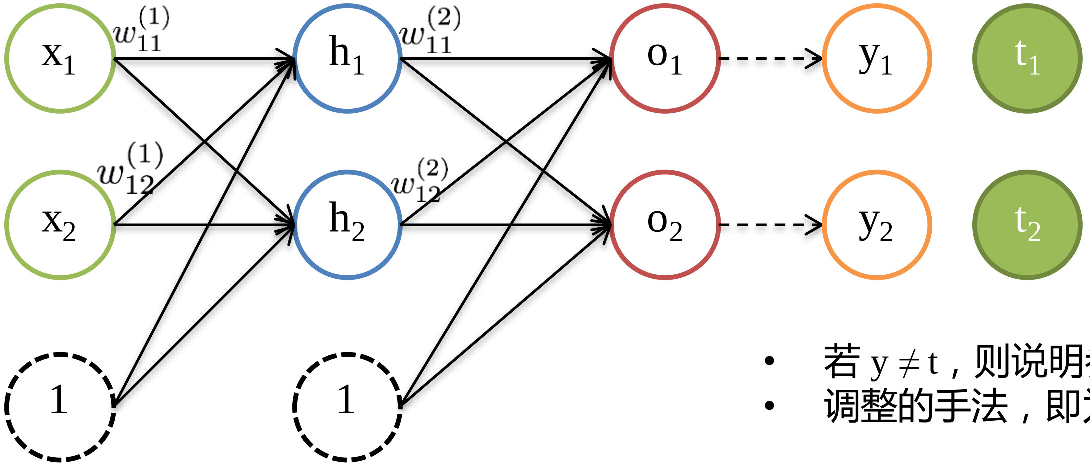


with step activation function



with sigmoid activation function

反向传播（back propagation）算法



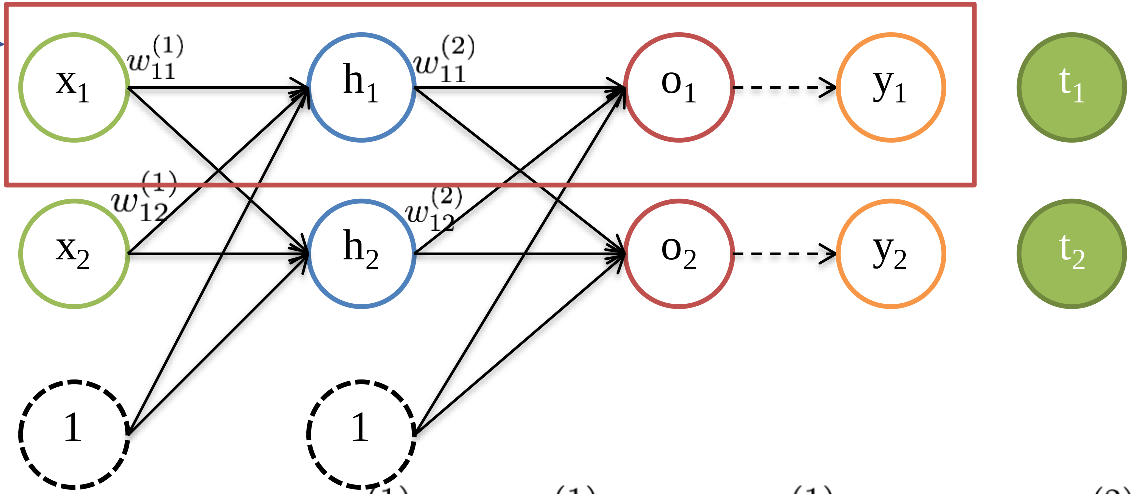
- 有一层隐层
- 输入为2维向量x
- 输出为2维向量y
- x所对应的期望输出(ground truth)为 t

- 若 $y \neq t$，则说明参与计算的权重w不恰当，需要进行调整。
- 调整的手法，即为反向传播。

- 反向传播算法的核心，是通过比较输出y和真值t，对参与计算的w进行调整。
- 其计算方法是从网络的输出层开始，向输入层方向逐层计算梯度并更新权重，与前馈运算正好相反。

在开始之前，大家需要做一些准备工作：

- 损失函数（以二次损失函数为例）
- 损失函数对权重参数的偏导数
- 梯度
- 链式法则
- 激活函数的导数 $y = sigmoid(x), y' = y(1-y)$
- 以及，相信这东西有点难于理解



$$Cost(x) = \frac{1}{2}(t-y)^2$$

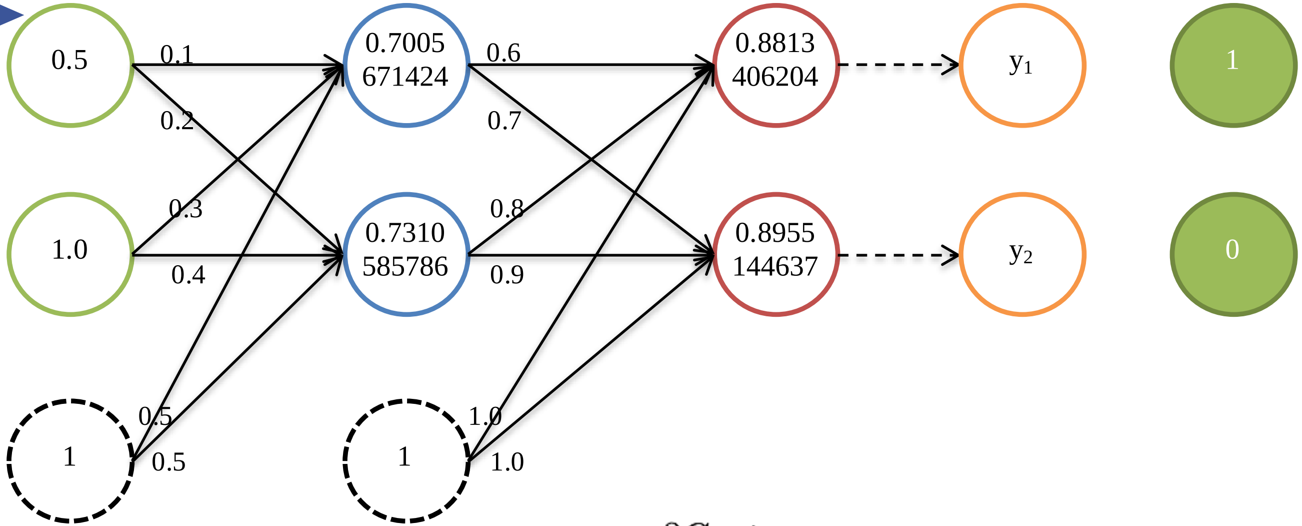
$$\sigma(x) = sigmoid(x)$$

$$h_1 = sigmoid(w_{11}^{(1)}x_1 + w_{12}^{(1)}x2 + bias_1^{(1)}) \qquad logit_1^{(2)} = w_{11}^{(2)}h_1 + w_{12}^{(2)}h_2 + bias_1^{(2)}$$

则 $y1 = sigmoid(logit_1^{(2)}) = sigmoid(w_{11}^{(2)}h_1 + w_{12}^{(2)}h_2 + bias_1^{(2)})$

根据链式法则：
$$\frac{\partial Cost}{\partial w_{11}^{(2)}} = \frac{\partial Cost}{\partial y_1} \times \frac{\partial y_1}{\partial logit_1^{(2)}} \times \frac{\partial logit_1^{(2)}}{\partial w_{11}^{(2)}}$$
$$= \frac{\partial \frac{1}{2}(t_1-y_1)^2}{\partial y_1} \times \frac{\partial sigmoid(logit_1^{(2)})}{\partial logit_1^{(2)}} \times \frac{\partial (w_{11}^{(2)}h_1 + w_{12}^{(2)}h_2 + bias_1^{(2)})}{\partial w_{11}^{(2)}}$$
$$= (y_1 - t_1) \times y_1(1-y_1) \times h_1$$

我们定义 $\delta_1^{(2)} = \frac{\partial Cost}{\partial logit_1^{(2)}}$ 则 $\frac{\partial Cost}{\partial w_{11}^{(2)}} = \delta_1^{(2)} \times h_1$ 扩展：$\frac{\partial Cost}{\partial w_{ij}^{(l)}} = h_j^{(l-1)}\delta_i^{(l)}$



$$\delta_1^{(2)} = \frac{\partial Cost}{\partial logit_1^{(2)}} = \frac{\partial Cost}{\partial y_1} \times \frac{\partial y_1}{\partial logit_1^{(2)}} = \frac{\partial \frac{1}{2}(t_1 - y_1)^2}{\partial y_1} \frac{\partial sigmoid(logit_1^{(2)})}{logit_1^{(2)}} = (y_1 - t_1) \times y_1(1 - y_1) = (0.88134062 - 1) \times (0.88134062 \times (1 - 0.88134062)) = -0.01240932$$

$$\frac{\partial Cost}{\partial w_{11}^{(2)}} = \delta_1^{(2)} \times h_1 = -0.01240932 \times 0.7005671424 = -0.00869356$$

$$\partial w_{11\_new}^{(2)} = w_{11\_old}^{(2)} - \eta \frac{\partial Cost}{\partial w_{11}^{(2)}} = 0.6 - (-0.00869356) = 0.60869356$$

$$Cost = \frac{1}{2}(t-y)^2$$

$$\sigma(x) = sigmoid(x)$$

$$y = sigmoid(x), \frac{\partial y}{\partial x} = y(1-y)$$

$$\frac{\partial Cost}{\partial w_{ij}^{(l)}} = h_j^{(l-1)} \delta_i^{(l)}$$

$$\delta_i^{(L)} = \frac{\partial Cost}{\partial logit_i^{(L)}} = \frac{\partial Cost}{\partial y_i} \times \frac{\partial y_i}{\partial logit_i^{(L)}} = \nabla_y Cost \times \sigma'(logit_i^{(L)})$$

对于 $w^{(1)}$，我们来计算损失函数对于它的偏导数（也就是梯度）：

$$\frac{\partial Cost}{\partial w^{(l)}} = h^{(l-1)} \delta^{(l)}$$

$$\begin{aligned}
\delta^{(l)} &= \frac{\partial Cost}{\partial logit^{(l)}} \\
&= \frac{\partial Cost}{\partial logit^{(l+1)}} \times \frac{\partial logit^{(l+1)}}{\partial logit^{(l)}} \\
&= \delta^{(l+1)} \times \frac{\partial logit^{(l+1)}}{\partial logit^{(l)}} \\
&= \delta^{(l+1)} \times \frac{\partial (w^{(l+1)} \sigma(logit^{(l)}))}{\partial logit^{(l)}} \\
&= \delta^{(l+1)} w^{(l+1)} \sigma'(logit^{(l)})
\end{aligned}$$

对于偏置项：我们有：

$$\frac{\partial Cost}{\partial bias_i^{(l)}} = \delta_i^{(l)}$$

BP四项基本原则

$$\delta_i^{(L)} = \nabla_y Cost \times \sigma'(logit_i^{(L)}) \quad (BP1)$$

$$\delta_i^{(l)} = \sum_j \delta_j^{(l+1)} w_{ji}^{(l+1)} \sigma'(logit_i^{(l)}) \quad (BP2)$$

$$\frac{\partial Cost}{\partial bias_i^{(l)}} = \delta_i^{(l)} \quad (BP3)$$

$$\frac{\partial Cost}{\partial w_{ij}^{(l)}} = h_j^{(l-1)} \delta_i^{(l)} \quad (BP4)$$

BP四项基本原则（矩阵形态）

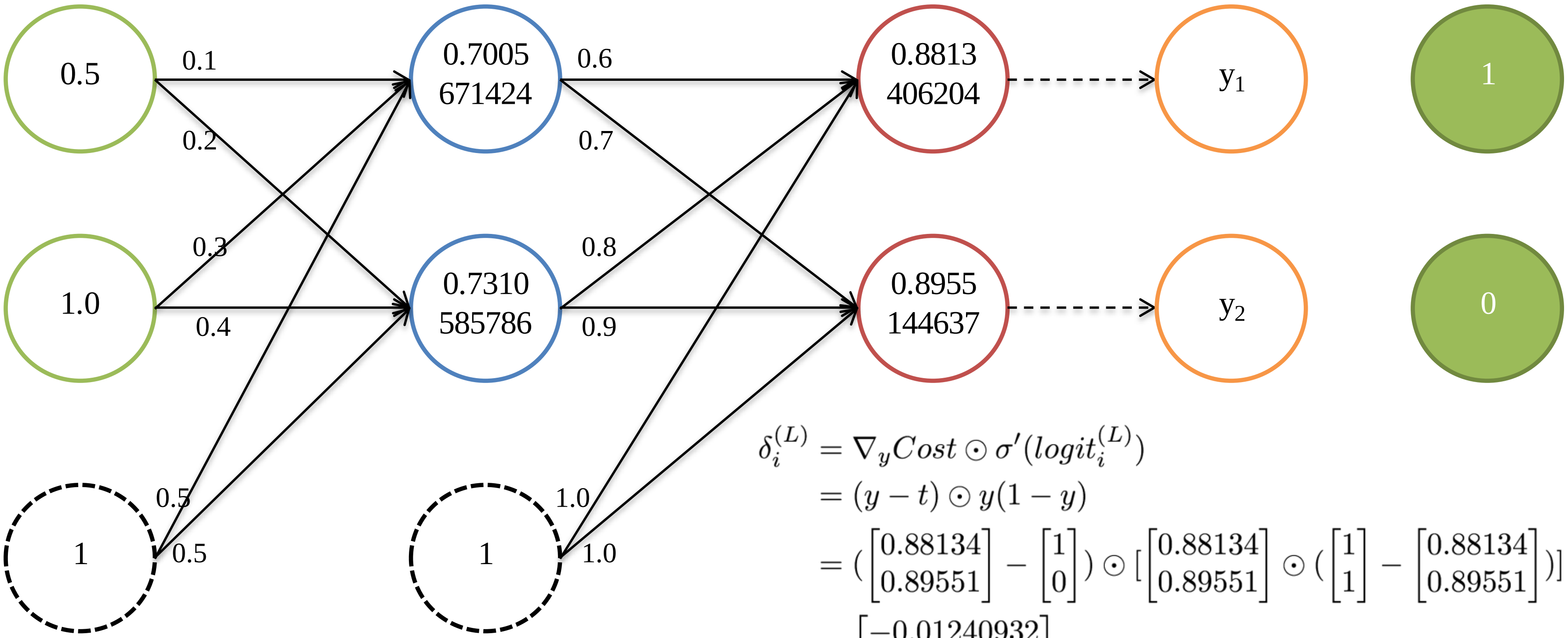
Hadamard乘积，element-wise product

$$\delta_i^{(L)} = \nabla_y Cost \odot \sigma'(logit_i^{(L)}) \qquad (BP1)$$

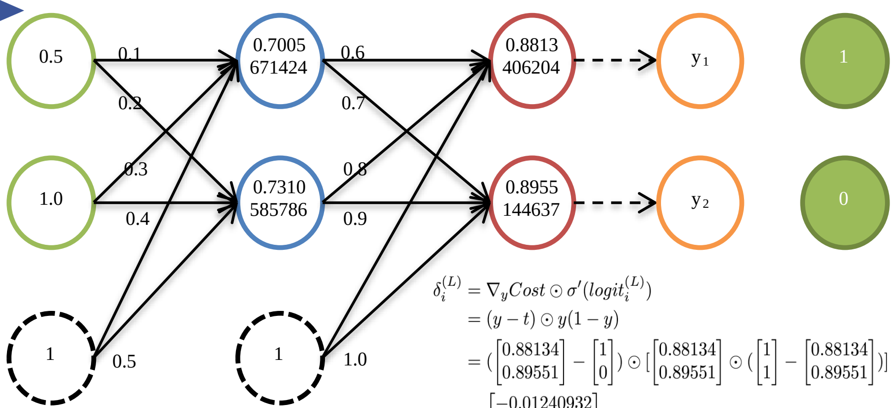
$$\delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(logit^{(l)}) \qquad (BP2)$$

$$\frac{\partial Cost}{\partial bias^{(l)}} = \delta^{(l)} \qquad (BP3)$$

$$\frac{\partial Cost}{\partial w^{(l)}} = \delta^{(l)} \cdot (h^{(l-1)})^T \qquad (BP4)$$



$$
\begin{aligned}
\delta_i^{(L)} &= \nabla_y Cost \odot \sigma'(logit_i^{(L)}) \\
&= (y-t) \odot y(1-y) \\
&= (\begin{bmatrix}0.88134\\0.89551\end{bmatrix} - \begin{bmatrix}1\\0\end{bmatrix}) \odot [\begin{bmatrix}0.88134\\0.89551\end{bmatrix} \odot (\begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}0.88134\\0.89551\end{bmatrix})] \\
&= \begin{bmatrix}-0.01240932\\0.08379177\end{bmatrix}
\end{aligned}
$$



$$\delta_i^{(L)} = \nabla_y Cost \odot \sigma'(logit_i^{(L)})$$
$$= (y-t)\odot y(1-y)$$
$$= (\begin{bmatrix}0.88134\\0.89551\end{bmatrix} - \begin{bmatrix}1\\0\end{bmatrix}) \odot [\begin{bmatrix}0.88134\\0.89551\end{bmatrix} \odot (\begin{bmatrix}1\\1\end{bmatrix} - \begin{bmatrix}0.88134\\0.89551\end{bmatrix})]$$
$$= \begin{bmatrix}-0.01240932\\0.08379177\end{bmatrix}$$

$$\Delta w^{(2)} = \delta^{(2)} \cdot h^{(1)}$$
$$= \begin{bmatrix}-0.01240932\\0.08379177\end{bmatrix} \cdot \begin{bmatrix}0.70056714\\0.73105858\end{bmatrix}^T$$
$$= \begin{bmatrix}-0.00869356 & 0.00907194\\0.05870176 & 0.0612567\end{bmatrix}$$

$$\delta^{(1)} = ((w^{(2)})^T\delta^{(2)}) \odot \sigma'(logit^{(1)})$$
$$= ((\begin{bmatrix}0.6 & 0.8\\0.7 & 0.9\end{bmatrix}^T \cdot \begin{bmatrix}-0.01240932\\0.08379177\end{bmatrix})) \odot \begin{bmatrix}0.20977282\\0.19661193\end{bmatrix}$$
$$= \begin{bmatrix}0.01074218\\0.01287516\end{bmatrix}$$

$$\Delta w^{(1)} = \delta(1) \cdot x^T$$
$$= \begin{bmatrix}0.01074218\\0.01287516\end{bmatrix} \cdot \begin{bmatrix}0.5\\1\end{bmatrix}^T$$
$$= \begin{bmatrix}0.00537109 & 0.01074218\\0.00643758 & 0.01287516\end{bmatrix}$$

$$w_{new}^{(1)} = w_{old}^{(1)} - \Delta w^{(1)}$$
$$= \begin{bmatrix}0.1 & 0.3\\0.2 & 0.4\end{bmatrix} - \begin{bmatrix}0.00537109 & 0.01074218\\0.00643758 & 0.01287516\end{bmatrix}$$
$$= \begin{bmatrix}0.09462891 & 0.28925782\\0.19356242 & 0.38712484\end{bmatrix}$$

$$b_{new}^{(1)} = b_{old}^{(1)} - \Delta b^{(1)}$$
$$= \begin{bmatrix}0.5\\0.5\end{bmatrix} - \begin{bmatrix}0.01074218\\0.01287516\end{bmatrix}$$
$$= \begin{bmatrix}0.48925782\\0.48712484\end{bmatrix}$$

$$w_{new}^{(2)} = w_{old}^{(2)} - \Delta w^{(2)}$$
$$= \begin{bmatrix}0.60869356 & 0.80907194\\0.64129824 & 0.8387433\end{bmatrix}$$

$$b_{new}^{(2)} = b_{old}^{(2)} - \Delta b^{(2)}$$
$$= \begin{bmatrix}1.01240932\\0.91620823\end{bmatrix}$$

梯度消失（弥散） vanishing gradient

$$\delta_i^{(L)} = \nabla_y Cost \odot \sigma'(logit_i^{(L)}) \qquad (BP1)$$

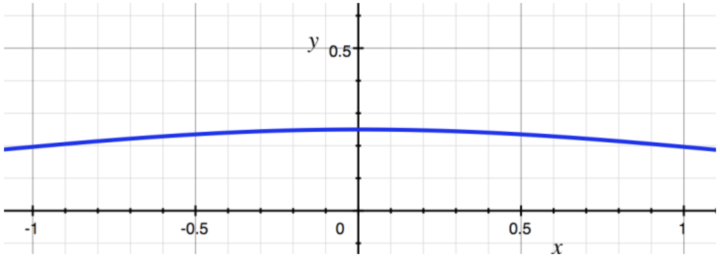
$$\delta^{(l)} = ((w^{(l+1)})^T \delta^{(l+1)}) \odot \sigma'(logit^{(l)}) \qquad (BP2)$$

$$\frac{\partial Cost}{\partial bias^{(l)}} = \delta^{(l)} \qquad (BP3)$$

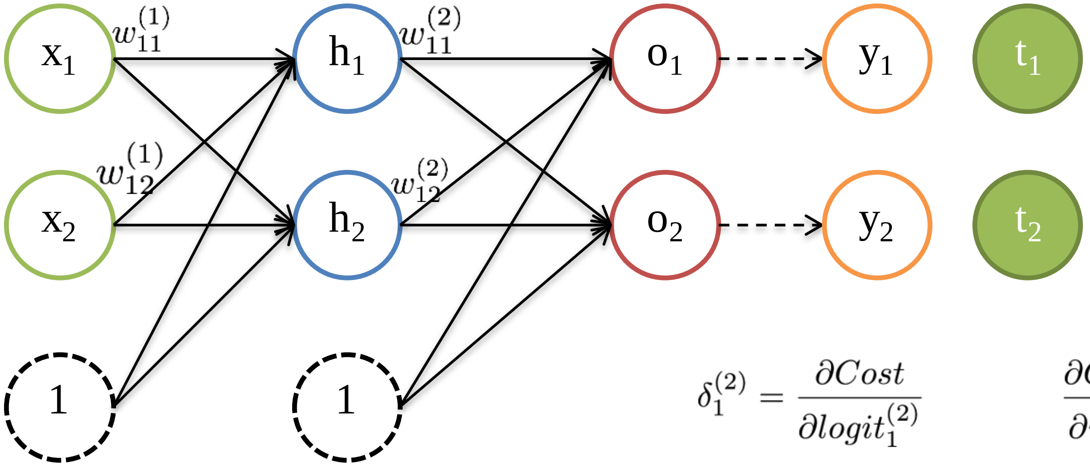
$$\frac{\partial Cost}{\partial w^{(l)}} = \delta^{(l)} \dot{(} h^{(l-1)})^T \qquad (BP4)$$

BP2中我们可以看到，计算梯度时包含了激活函数的导数。
如果使用sigmoid函数，那么它的导数为sigmoid’(x)=sigmoid(x)*(1-sigmoid(x))

其最大值为0.25，而越往两侧，越接近0
在反向传播时，每一层的δ都逐层减小
最终消失



zig zag



思考一下o_1的两个权重

$w_{11}^{(2)}$ $w_{12}^{(2)}$

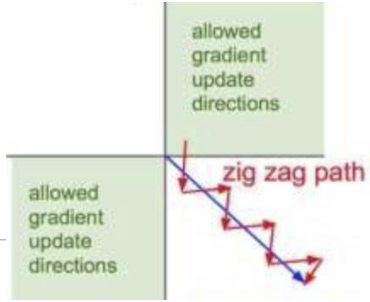
$$\delta_1^{(2)} = \frac{\partial Cost}{\partial logit_1^{(2)}} \qquad \frac{\partial Cost}{\partial w_{11}^{(2)}} = \delta_1^{(2)} \times h_1 \qquad \frac{\partial Cost}{\partial w_{12}^{(2)}} = \delta_1^{(2)} \times h_2$$

若h_1、h_2都是sigmoid函数的输出，则h_1、$h_2>0$

那么$w_{11}^{(2)}$ $w_{12}^{(2)}$ 两个权重得到的更新值Δ要么同时为正，要么同时为负

如果这两个权重恰好要求一个增加，另一个减小，那么……

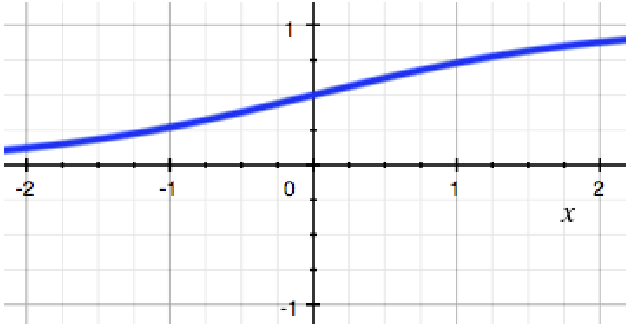
$$(\Delta w_1, \Delta w_2)$$



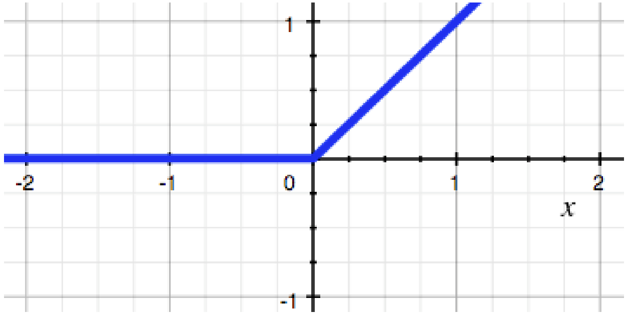
回过头来再看看激活函数

- 饱和：函数的梯度趋向或者等于0
  - 软饱和：无线趋近于0
  - 硬饱和：等于0

Sigmoid：两侧软饱和，梯度消失
zigzag



ReLU：左侧硬饱和，右侧无饱和，
zigzag，神经元死亡



各种激活函数的特点

- unit：线性分界
  - 几乎已经不用了
- sigmoid：非线性分界
  - 两端软饱和，输出为(0,1)区间
  - 两端有梯度消失问题
  - 因为输出恒为正，可能有zigzag现象
- tanh：非线性分界
  - 两端软饱和，输出为(-1, 1)区间
  - 仍然存在梯度消失问题
  - 没有zigzag，收敛更快(LeCun 1989)
- ReLU：非线性分界
  - 左侧硬饱和，右侧无饱和，输出为[0,+∞)区间
  - 左侧会出现梯度一直为0的情况，导致神经元不再更新（神经元死亡）
  - 改善了梯度弥散
  - 同样存在zigzag

一些新的激活函数

- PReLU：$prelu(x) = \begin{cases} x & x>0 \\ \alpha x & x\leq 0 \end{cases}$
- MaxOut：$\max(w_1^T x + b_1,\ w_2^T x + b_2,\ \ldots,\ w_n^T x + b_n)$
- ELU：$elu(x)=\begin{cases} x & x>0 \\ \alpha\cdot(\exp(x)-1) & x\leq 0 \end{cases}$
- SELU：$selu(x) = selu_{scale}\cdot\begin{cases} x & x>0 \\ selu_{\alpha}\cdot(\exp(x)-1) & x\leq 0 \end{cases}$

  $selu_{\alpha}=1.6732632423543772848170429916717$

  $selu_{scale}=1.0507009873554804934193349852946$
- Swish：$swish(x) = x\cdot sigmoid(x)$

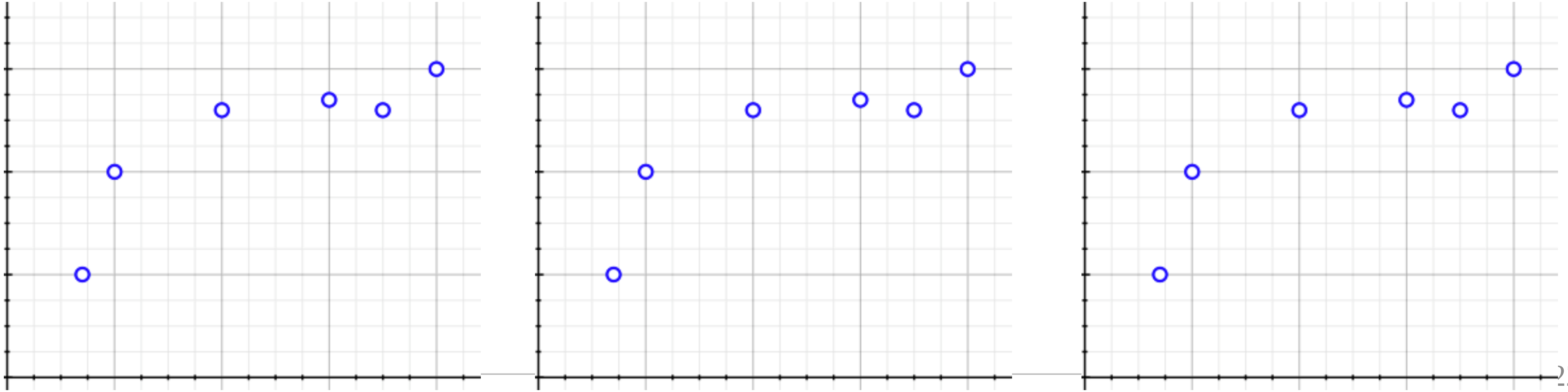
正则项

$$L_{reg}(t, f(x,W)) = L(t, f(x,W)) + \lambda\Omega(W)$$

$$\Omega_{L1}(W) = |W|$$

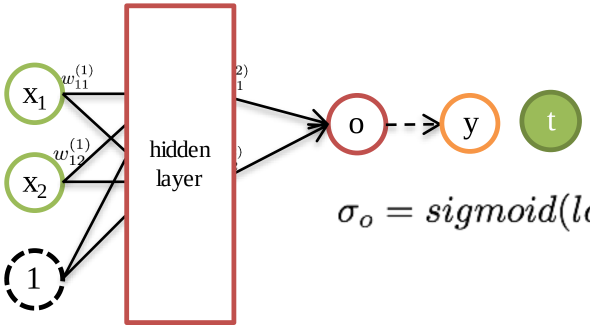
$$\Omega_{L2}(W) = \|W\|^2$$

- 正则项：
- L1和L2都可以实现稀疏
- L2会对过大的权重施加更严厉的惩罚



▶ 交叉熵损失

二分类问题



$$\sigma_o = sigmoid(logit_o) = \frac{1}{1+e^{-logit_o}}$$

$$\delta^{(L)} = \nabla_y Cost \cdot \sigma'(logit^{(L)})$$

$$\nabla_y Cost = \frac{\partial\{-[t \cdot ln(y) + (1-t)ln(1-y)]\}}{\partial y}$$

$$= -(\frac{t}{y} - \frac{1-t}{1-y})$$

$$= -\frac{(t-ty)-(y-ty)}{y(1-y)}$$

$$= -\frac{t-y}{y(1-y)}$$

$$\sigma'(logit^{(L)}) = y(1-y)$$

$$\delta_i^{(L)} = -(t-y)$$

$$= y - t$$

交叉熵损失： $L = -[t \cdot ln(y) + (1-t)ln(1-y)]$

梯度：

$$\frac{\partial cost}{\partial w_j} = x_j * \delta^{(L)}$$

$$= x_j * (y-t)$$

$$or \; -x_j * (t-y)$$

softmax

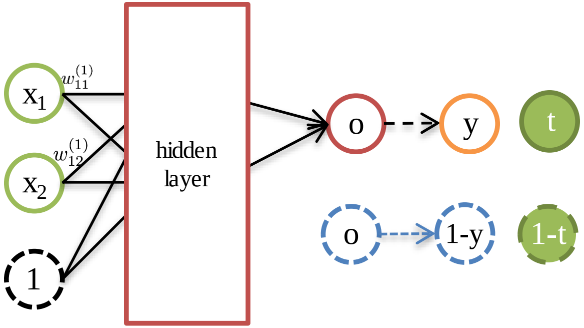
$$softmax_j^{(L)} = \frac{e^{logit_j^{(L)}}}{\sum_k e^{logit_k^{(L)}}}$$

$$Cost = \sum_i t_i ln(y_i)$$

$$\frac{\partial Cost}{\partial b_j^L} = -(t_j - y_j^L)$$

$$\frac{\partial Cost}{\partial w_{jk}^L} = -y_k^{L-1}(t_j^L - y_j)$$

$$L = -[t \cdot ln(y) + (1-t)ln(1-y)]$$



加上正则项：

$$Cost = \sum_i t_i ln(y_i) + \sum_l \sum_j \sum_k \frac{\lambda}{2} \left\| w_{jk}^L \right\|^2$$

$$\frac{\partial Cost}{\partial w_{jk}^L} = -y_k^{L-1}(t_j^L - y_j) + w_{jk}^L$$

$$\begin{aligned} w_{jk\_new}^L &= w_{jk\_old}^L - \eta(-y_k^{L-1}(t_j^L - y_j) + \lambda w_{jk\_old}^L) \\ &= (1 - \eta\lambda) w_{jk\_old}^L + \eta y_k^{L-1}(t_j^L - y_j) \end{aligned}$$

权重的初始化

最原始的初始化：全部为固定值

$$W_{ij} = 0.1$$

稍好些的初始化：服从固定方差的独立高斯分布

$$W \sim G(0, \alpha^2)$$

Xavier初始化：服从动态方差的独立高斯分布

$$W \sim G(0, \sqrt{\frac{1}{n_{in}}}^{2})$$

MSRA初始化：服从动态方差的独立高斯分布

$$W \sim G(0, \sqrt{\frac{2}{n_{in}}}^{2})$$

不 止 于 代 码

THANK YOU