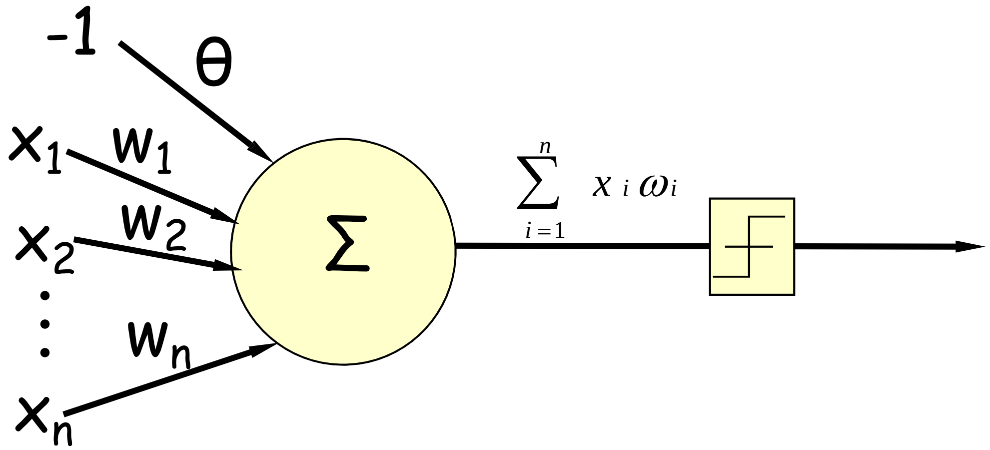
Ch.4 Computational Intelligence
第四章　智能计算 I

1. 概述
2. 神经计算
   - ANN
   - TLU
   - Backpropagation Learning
   - Hebb Learning
   - Perceptron Learning
3. 进化计算
4. 模糊计算
5. 群智能

4.2.3 Learning of ANN



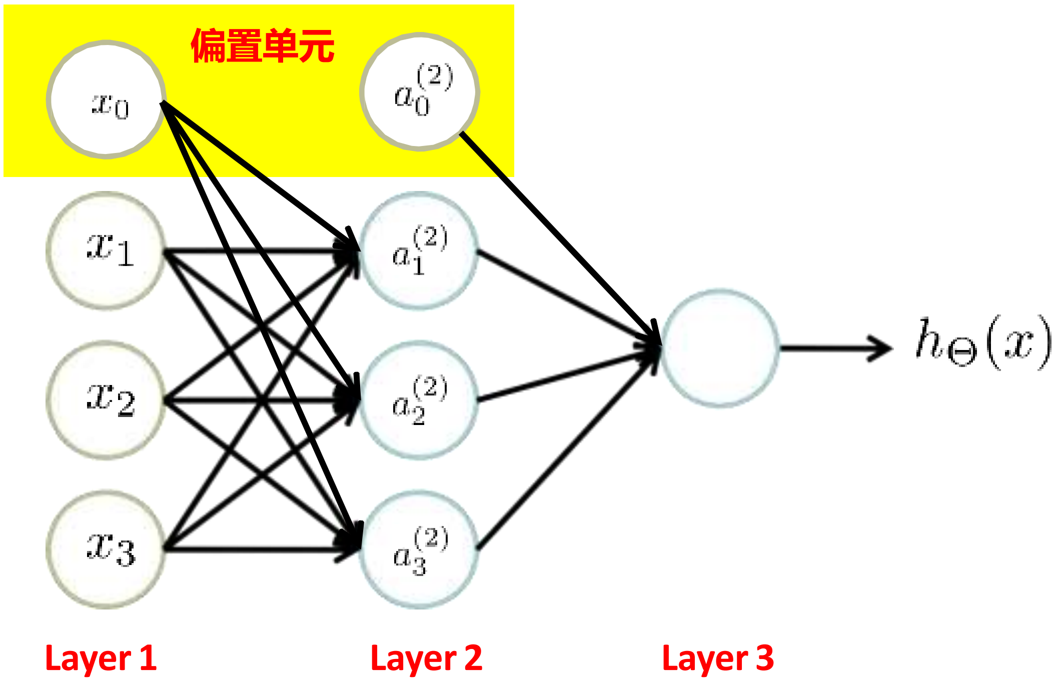
- **感知器学习**（Perceptron Learning Rule）
  - 有师学习
  - 权值调整公式

    $w_{ij}(t+1)=w_{ij}+\eta\ (t_i - O_i)\ O_j$

    w_{ij}： i 到 j 的 权值　O_i： i 的实际输出

    $\eta>0$： 学习常数　t_i： i 的期望输出
  - Equivalent to rules:
    - If output is correct do nothing.
    - If output is high, lower weights on active inputs
    - If output is low, increase weights on active inputs

全连接前馈网络（Fully Connect Feedforward Network)

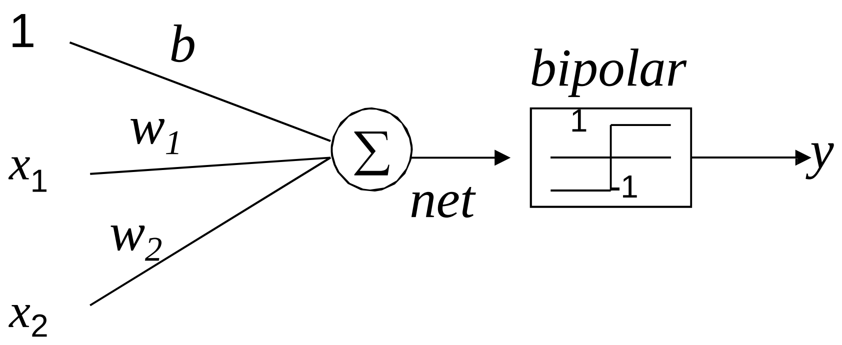


Learning Strategies——神经学习

Hebb Learning Rule

“条件反射”
“用进废退”

- supervised learning



hebb learning rule :

$w_i\,(new) = w_i\,(old) + \boldsymbol{x_i\ t}$ $\quad\quad i$=1,2

$b\ (new) = b\ (old) + t$

t——desired response or target

$\boldsymbol{x_i\ x_j}$

- e.g. Training an AND gate

x_1	x_2	t_1
1	1	1
1	-1	-1
-1	1	-1
-1	-1	-1

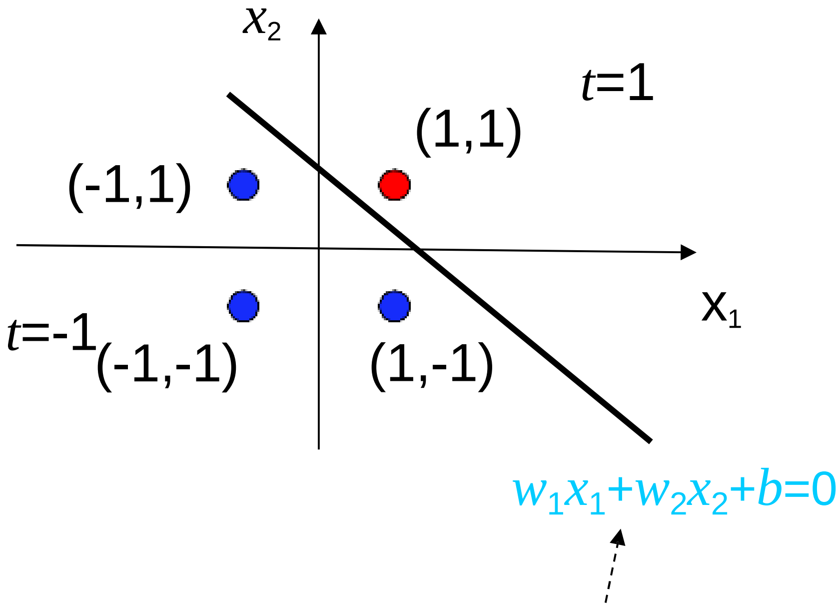


Figure out the function, confirm the value of w_1, w_2 , b

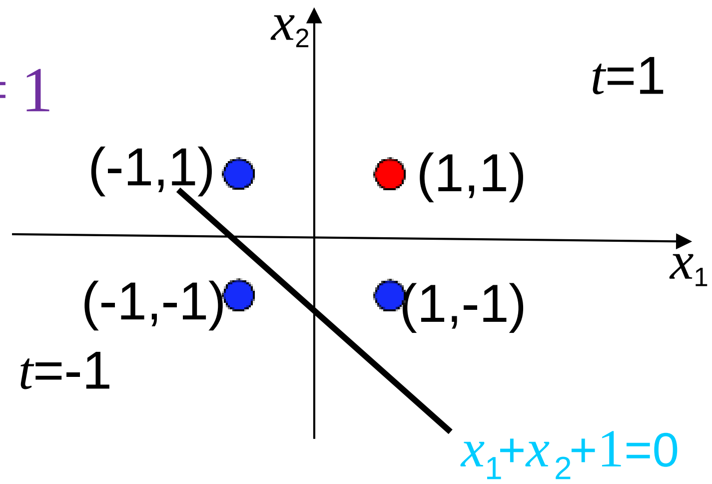
Hebb Learning Rule

step 1 set initial $b = 0$

$$w_1 = \sum_{x\in P} x + \sum_{x\in N} x = 0 \qquad w_2 = \sum_{x\in P} x - \sum_{x\in N} x = 0$$

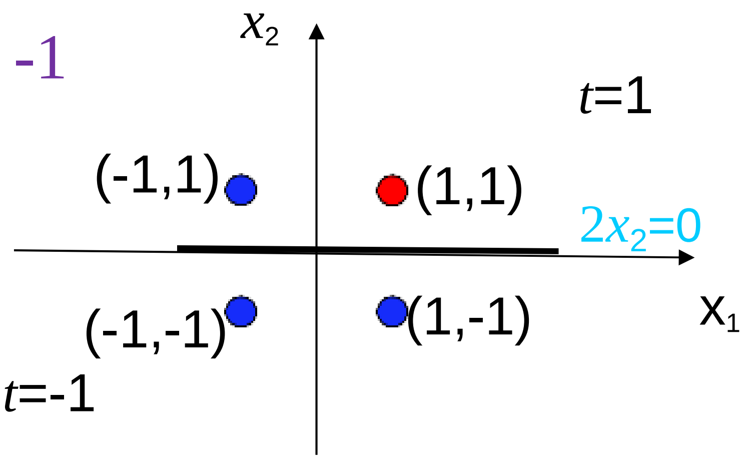
step 2 input $(x_1,x_2) = (1,1)$, $t = 1$

$$\begin{cases} w_1 = 0 + 1\cdot 1 = 1 \\ w_2 = 0 + 1\cdot 1 = 1 \\ b = 0 + 1 = 1 \end{cases}$$



step 2 input $(x_1,x_2) = (1,-1)$, $t = -1$

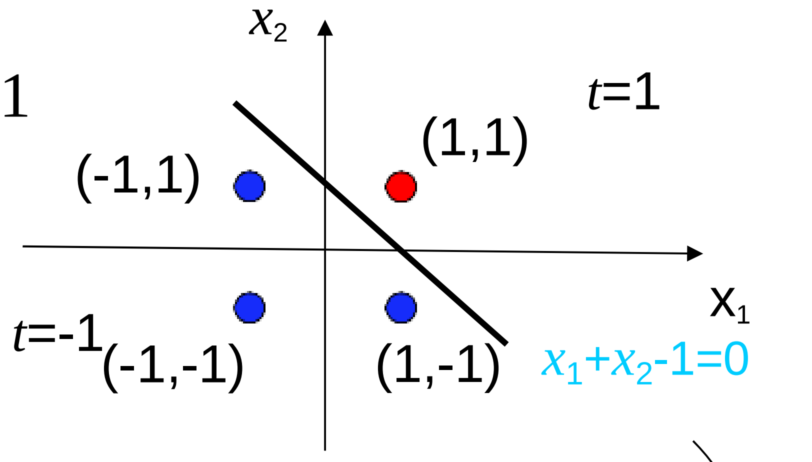
$$\begin{cases} w_1 = 1 + 1\cdot(-1) = 0 \\ w_2 = 1 + (-1)\cdot(-1) = 2 \\ b = 1 + (-1) = 0 \end{cases}$$



Hebb Learning Rule

step 4 input (x_1,x_2) = (-1,1) , t = -1

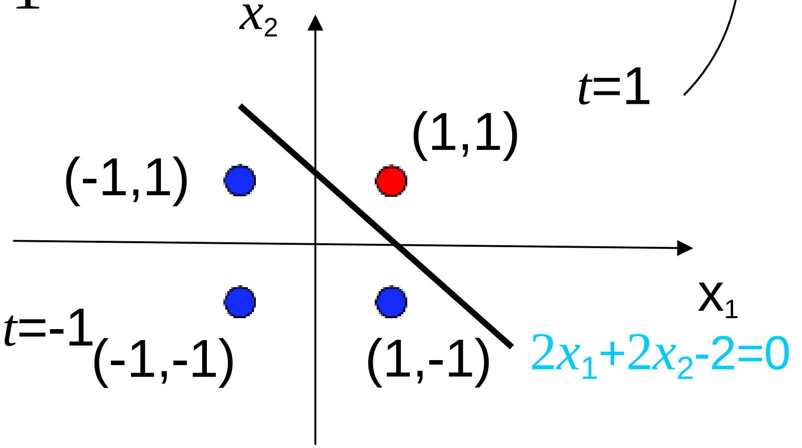
$$\begin{cases} w_1 = 0 + -1 \cdot (-1) = 1 \\ w_2 = 2 + 1 \cdot (-1) = 1 \\ b = 0 + (-1) = -1 \end{cases}$$



AND gate

step 5 input (x_1,x_2) = (-1,-1) , t = -1

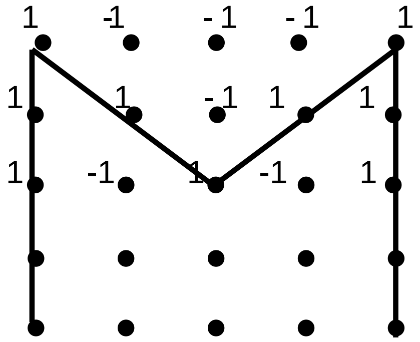
$$\begin{cases} w_1 = 1 + (-1) \cdot (-1) = 2 \\ w_2 = 1 + (-1) \cdot (-1) = 2 \\ b = -1 + (-1) = -2 \end{cases}$$



Example: Design a Neural Network to recognize “M” and “L”

1. Represent ”M” and “L” as vector

(1) “M”

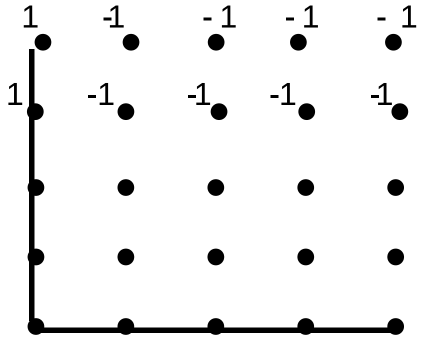


5×5

M = (1 -1 -1 -1 1, 1 1 -1 1 1, 1 -1 1 -1 1, 1 -1 -1 -1 1, 1 -1 -1 -1 1)

desired output = target = t = 1

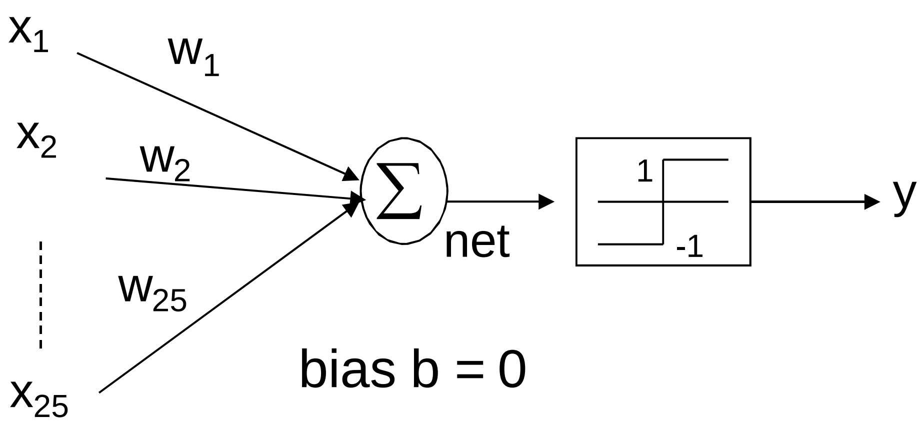
(2) “L”



L = (1 -1 -1 -1 -1, 1 -1 -1 -1 -1, 1 -1 -1 -1,1 -1 -1 -1 -1, 1 1 1 1 1)

desired output = target = t = -1

2. Use Hebb learning rule to train the Neural Network



Hebb learning rule:

(1) set initial $w_i = 0 \quad i=1\sim25$

(2) input x = M 且 t = 1

$$w = 0 + x \cdot 1$$

$$\therefore w = M$$

(3) input x = L 且 t = -1

$$w = M + x \cdot (-1)$$

$$\therefore w = M - L$$

w = M-L = (0 0 0 0 2,0 2 0 0 2, 0 0 2 0 2,0 0 0 0 2,0 -2 -2 -2 0)

$w = M - L = (0\ 0\ 0\ 0\ 2, 0\ 2\ 0\ 0\ 2, 0\ 0\ 2\ 0\ 2, 0\ 0\ 0\ 0\ 2, 0\ -2\ -2\ -2\ 0)$

3. Recognition

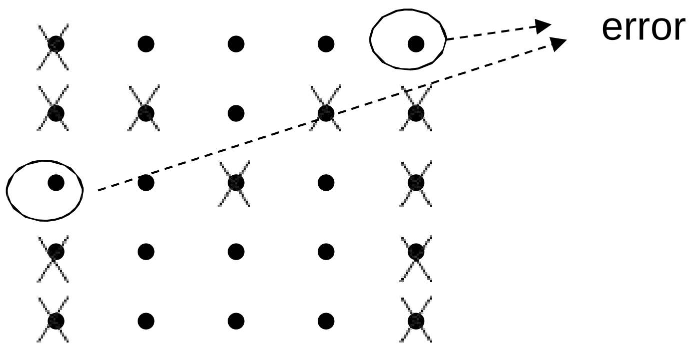
if input $x = M = (1\ -1\ -1\ -1\ 1, 1\ 1\ -1\ 1\ 1, 1\ -1\ 1\ -1\ 1, 1\ -1\ -1\ -1\ 1, 1\ -1\ -1\ -1\ 1)$

$net = w^T x = 20 > 0 \quad \Rightarrow \quad y = 1$

if input $x = L = (1\ -1\ -1\ -1\ -1, 1\ -1\ -1\ -1\ -1, 1\ -1\ -1\ -1\ -1, 1\ -1\ -1\ -1\ -1, 1\ 1\ 1\ 1\ 1)$

$net = w^T x = -20 < 0 \quad \Rightarrow \quad y = -1$

if there is a not perfect "M"

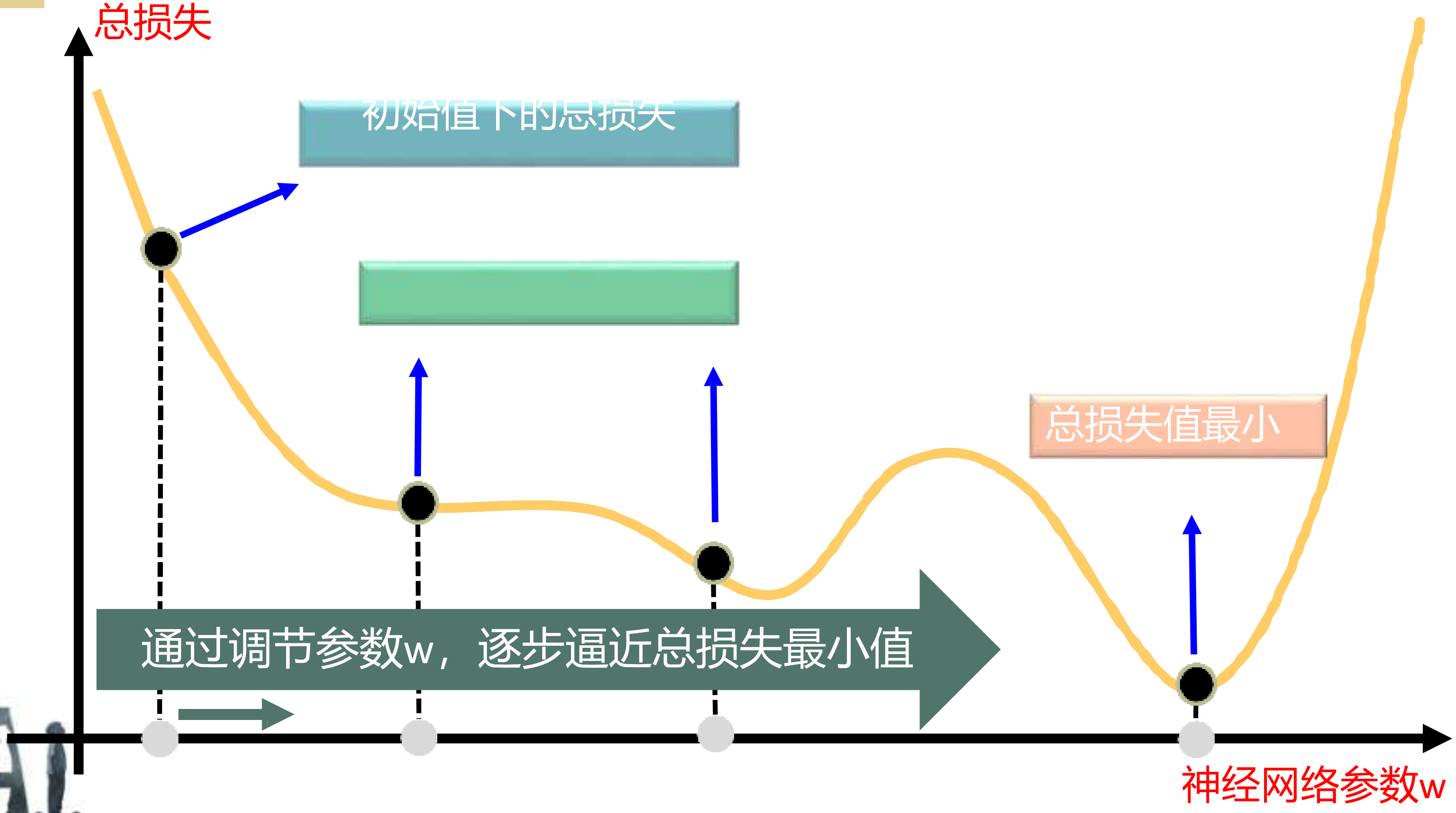


$x = (1\ -1\ -1\ -1\ -1, 1\ 1\ -1\ 1\ 1, -1\ -1\ 1\ -1\ 1, 1\ -1\ -1\ -1\ 1, 1\ -1\ -1\ -1\ 1)$

$net = w^T x = 16 > 0 \quad \Rightarrow y = 1$

The result still "M"

梯度下降法



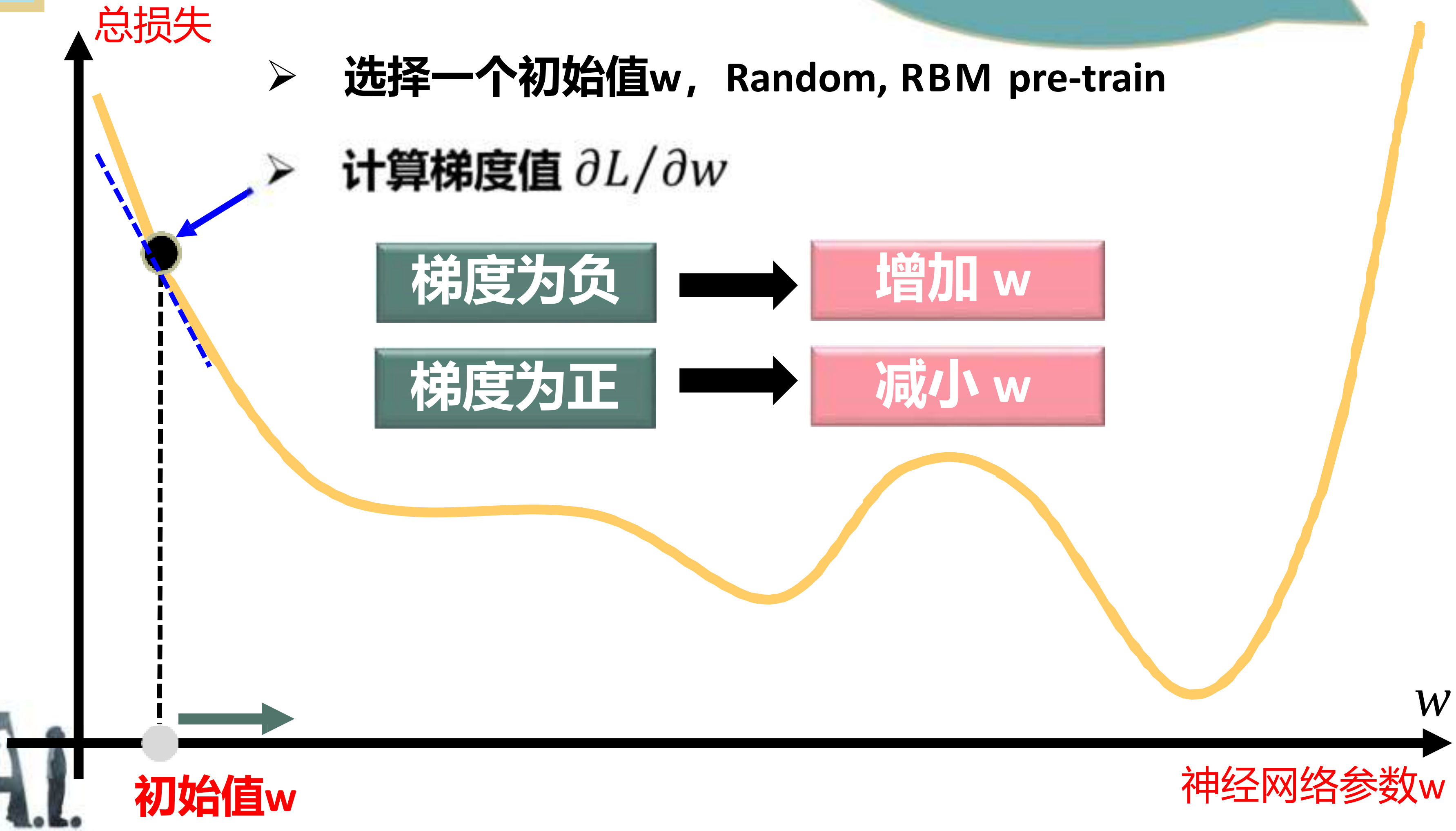
梯度下降法

神经网络参数$\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

- 选择一个初始值w，Random, RBM pre-train
- 计算梯度值 $\partial L / \partial w$

梯度为负 → 增加 w

梯度为正 → 减小 w



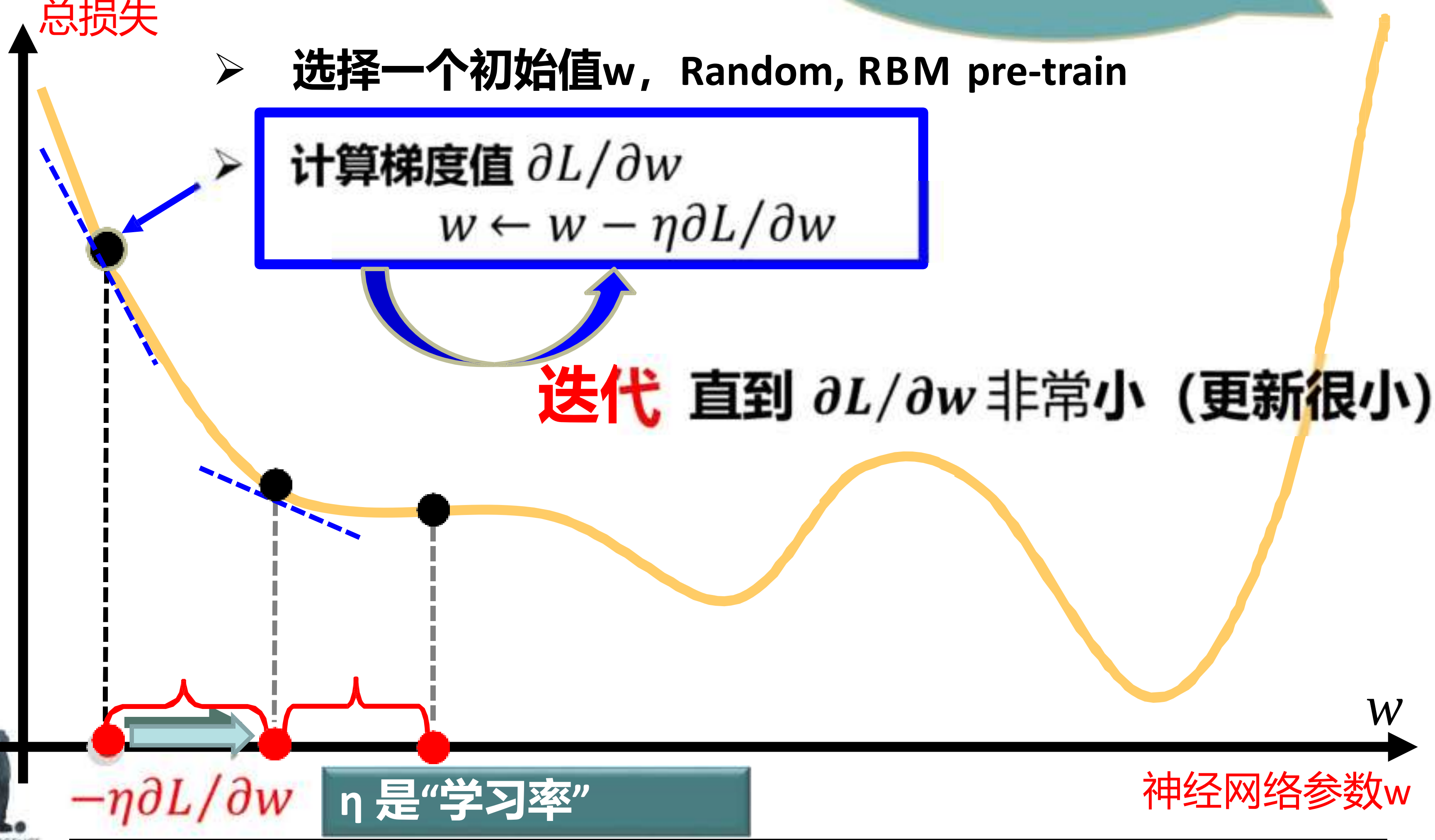
梯度下降法

神经网络参数$\theta = \{w_1, w_2, \cdots, b_1, b_2, \cdots\}$

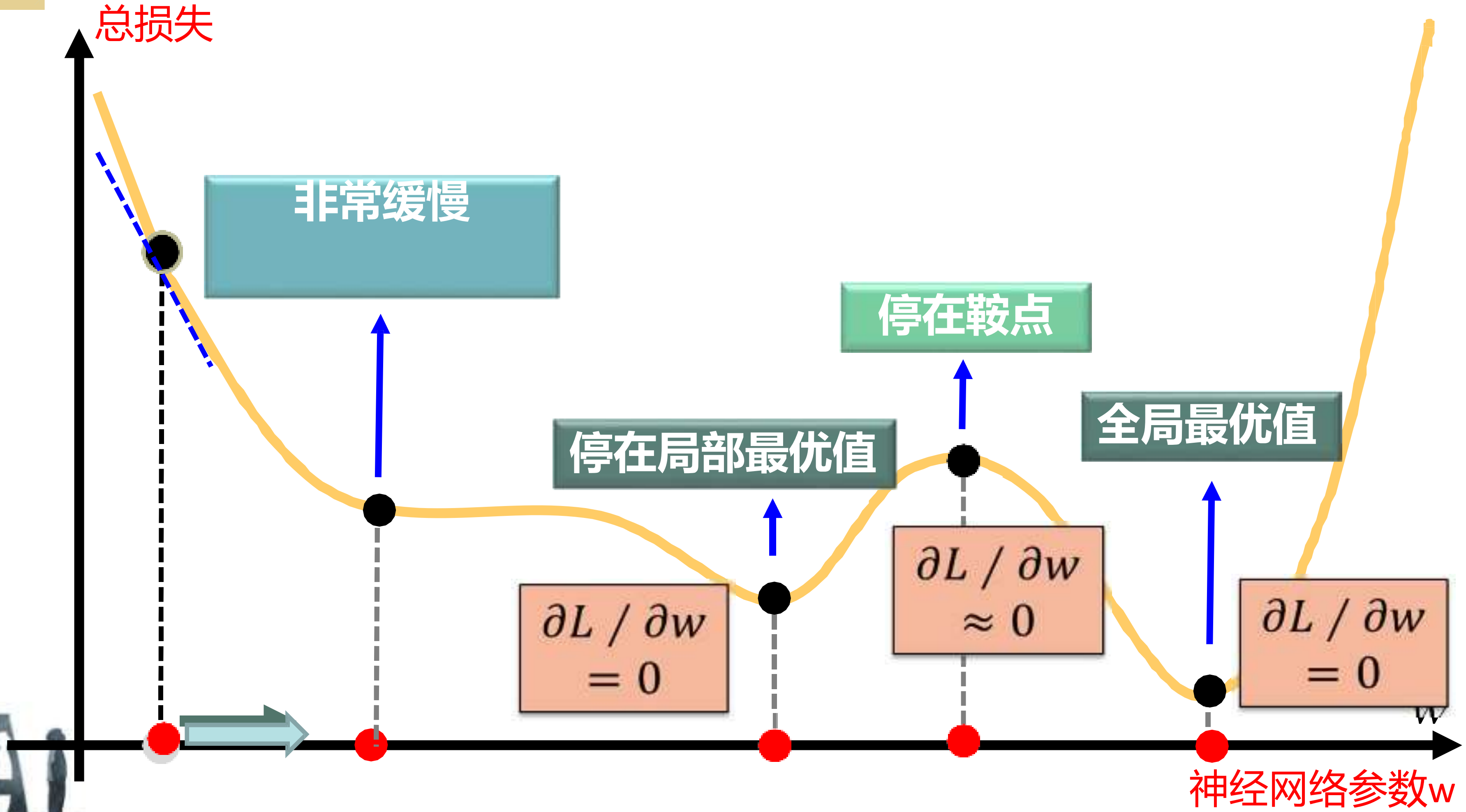
- 选择一个初始值w，Random, RBM pre-train
- 计算梯度值 $\partial L/\partial w$

$$w \leftarrow w - \eta \partial L/\partial w$$

迭代 直到 $\partial L/\partial w$ 非常小（更新很小）



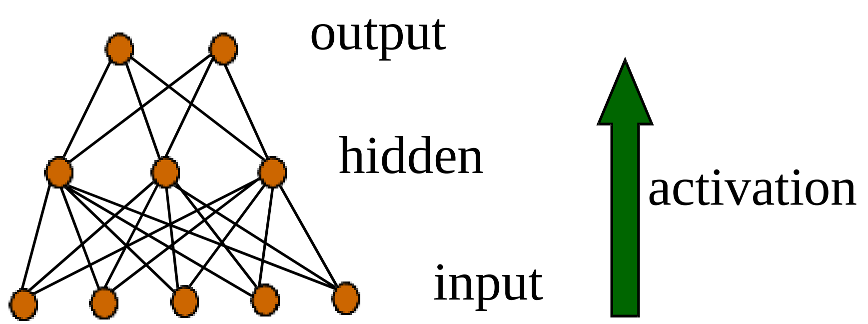
梯度下降法


总损失
非常缓慢
停在局部最优值
停在鞍点
全局最优值
$\partial L / \partial w = 0$
$\partial L / \partial w \approx 0$
$\partial L / \partial w = 0$
w
神经网络参数w

Backpropagation Learning

多层网络 Multi-Layer Networks

- 多层网络可以表示任意函数，但是要构造高效的学习算法曾被认为非常困难.
- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.



- The weights determine the function computed. Given an arbitrary number of hidden units, any Boolean function can be computed with a single hidden layer.

Backpropagation Learning

- 反向传播算法
  - 一种将输出层误差反向传播给隐藏层进行参数更新的方法
  - BP算法过程包含从**输出节点**开始，将误差从后向前传递，将误差分摊给各层所有单元，从而获得各层单元所产生的误差，进而依据这个误差来让各层单元负起各自责任、修正各单元参数。

Backpropagation Learning idea

- 学习的类型：有导师学习
- 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

反向传播算法

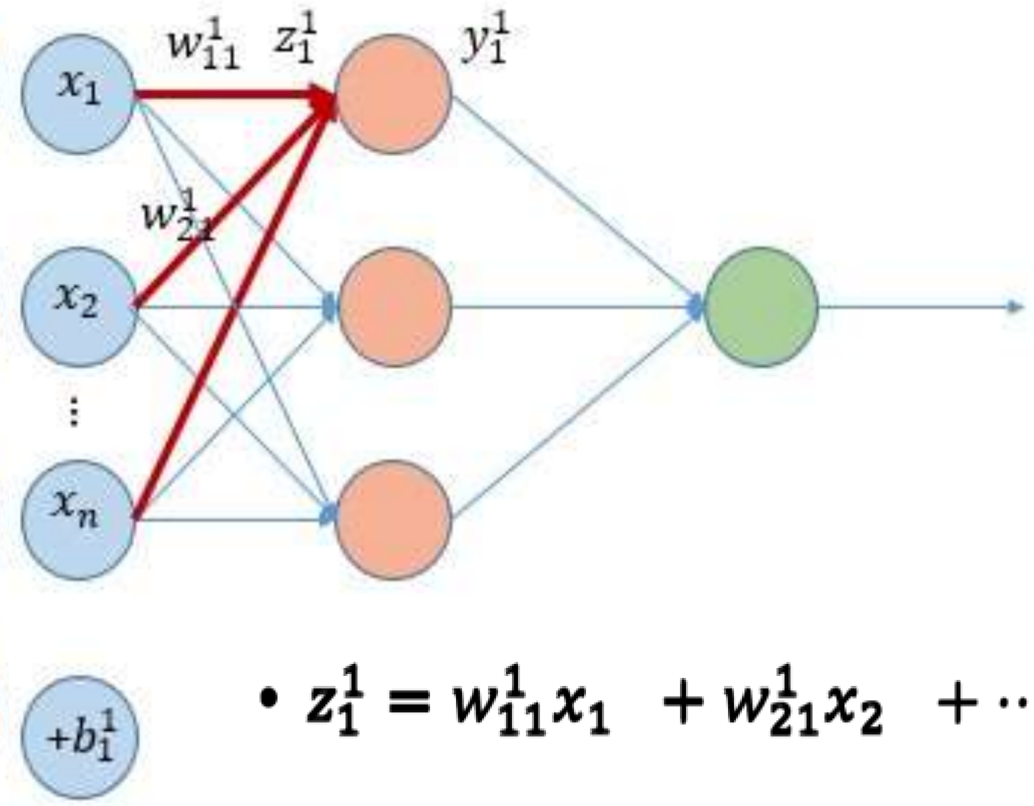
- 应用于多层前馈网络（BP神经网络）的一种学习算法
- BP算法过程包含从输出节点开始，反向地向第一隐含层传播由总误差引起的权值修正
- BP网络学习过程是一个对给定训练模式，利用传播公式，沿着减小误差的方向不断调整网络连接权值和阈值的过程
- 采用梯度下降法
  - 要求变换函数连续可导
  - 采用Sigmoid函数

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

$$f'(x) = \frac{df(x)}{dx} = f(x)(1 - f(x))$$

信号正向传播

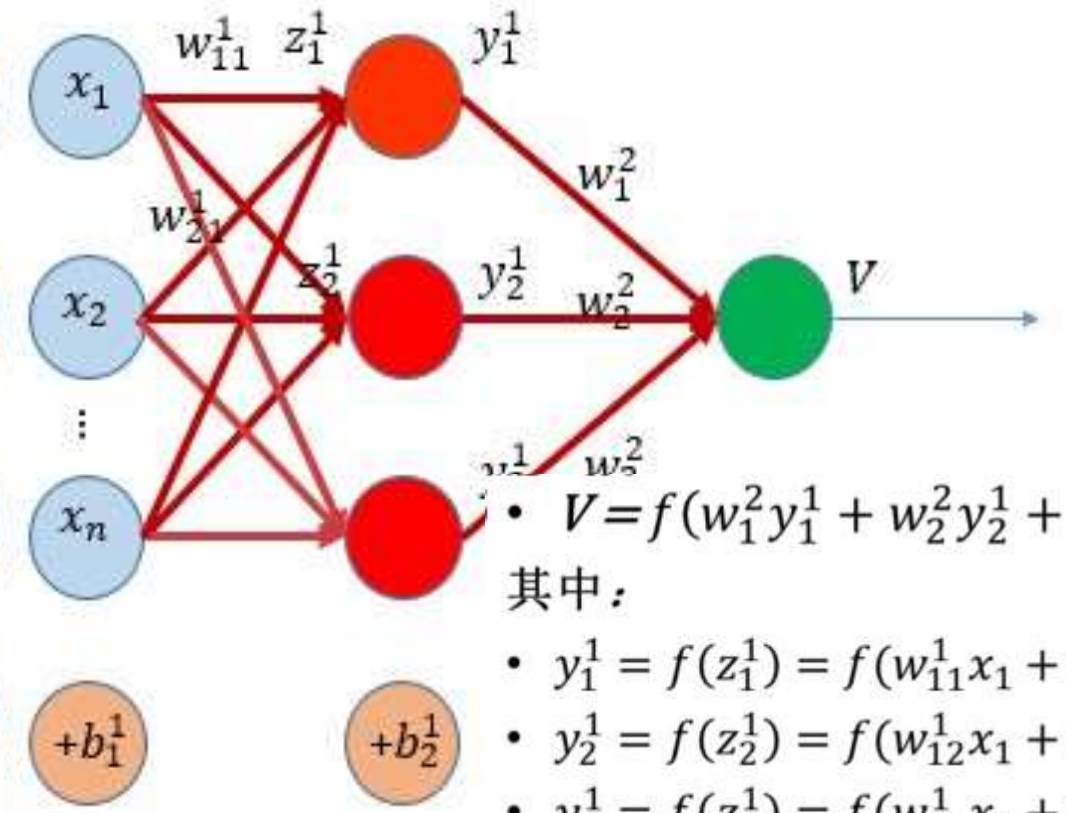
➢ **假定第l层神经元状态为z^l，经**激活函数后的输出值为y^l



- $z_1^1 = w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1$

信号正向传播

➢ **假定第l层神经元状态为z^l，经**激活函数后的输出值为y^l



- $V = f(w_1^2 y_1^1 + w_2^2 y_2^1 + \cdots + w_3^2 y_3^1 + b_2)$

其中：

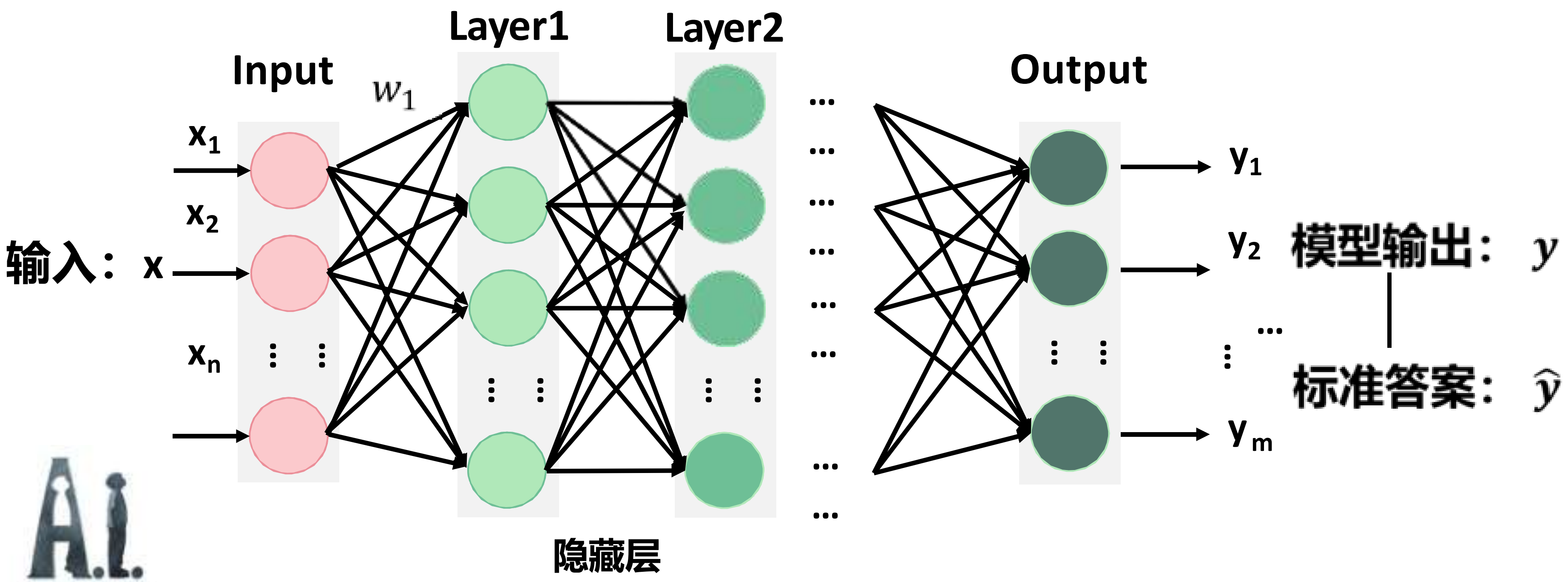
- $y_1^1 = f(z_1^1) = f(w_{11}^1 x_1 + w_{21}^1 x_2 + \cdots + w_{n1}^1 x_n + b_1^1)$
- $y_2^1 = f(z_2^1) = f(w_{12}^1 x_1 + w_{22}^1 x_2 + \cdots + w_{n2}^1 x_n + b_2^1)$
- $y_3^1 = f(z_3^1) = f(w_{13}^1 x_1 + w_{23}^1 x_2 + \cdots + w_{n3}^1 x_n + b_2^1)$

反向传播算法

只剩一个问题：
怎么求 梯度$\partial L/\partial w$?

$w_1 - \eta\,\partial L/\partial w_1$　　计算 $\partial L/\partial w_1$　　总误差 L

更新 w_1　　以此类推更新 w_2 w_3 ………



Backpropagation Learning idea

- 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

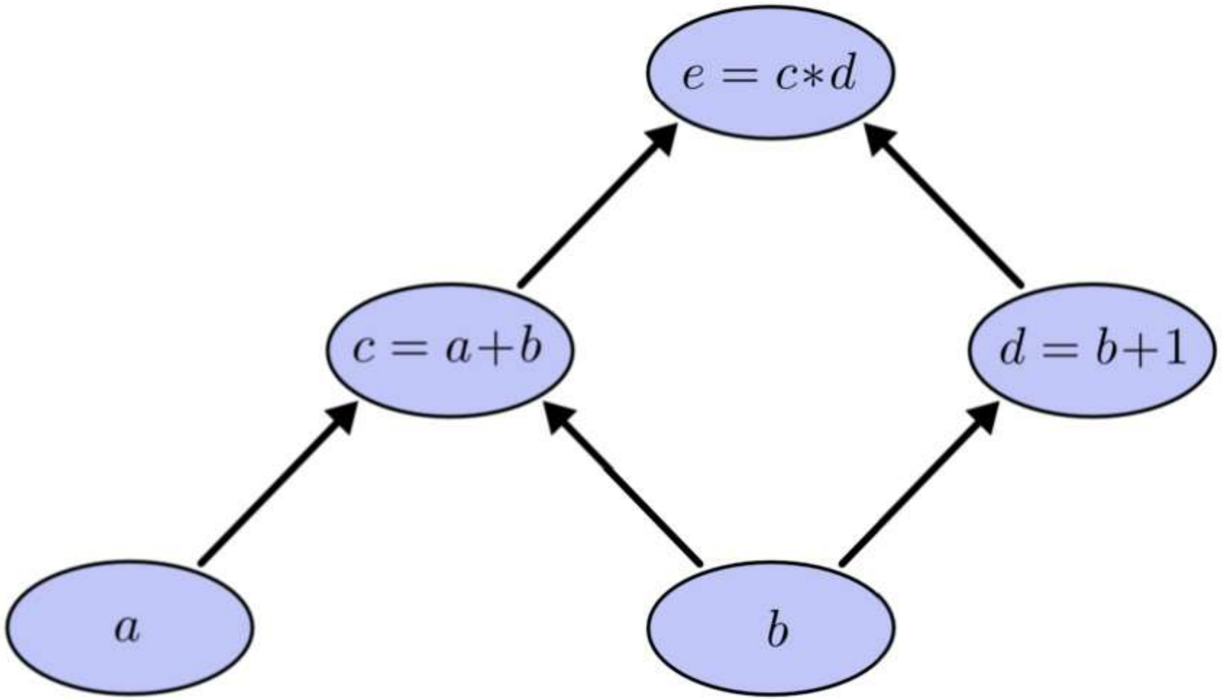
网络输出的误差减少到可接受的程度

- 进行到预先设定的学习次数为止

反向传播算法

- **算法示例**： e=(a+b)*(b+1), 求$\partial e/\partial a$, $\partial e/\partial b$

  引入两个中间变量c和d： c=a+b， d=b+1， e=c∗d

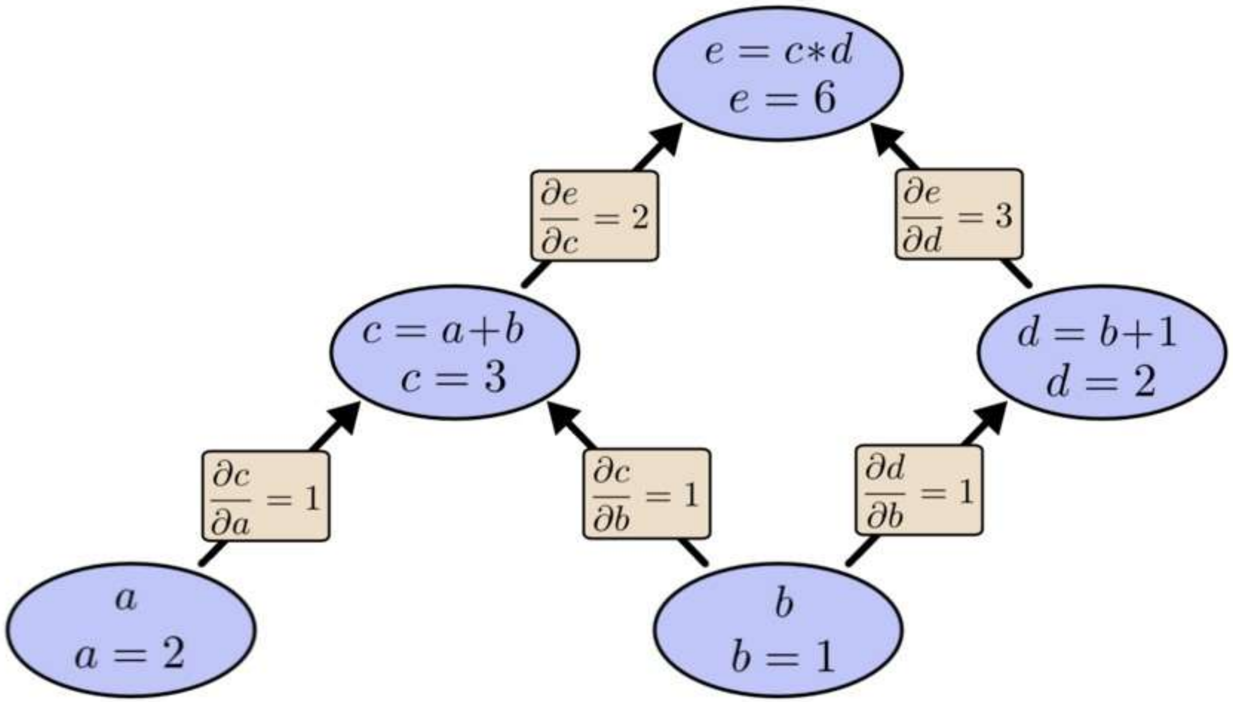


ref： http://colah.github.io/posts/2015-08-Backprop/

反向传播算法

- **算法示例：** e=(a+b)*(b+1), 求$\partial e/\partial a$, $\partial e/\partial b$

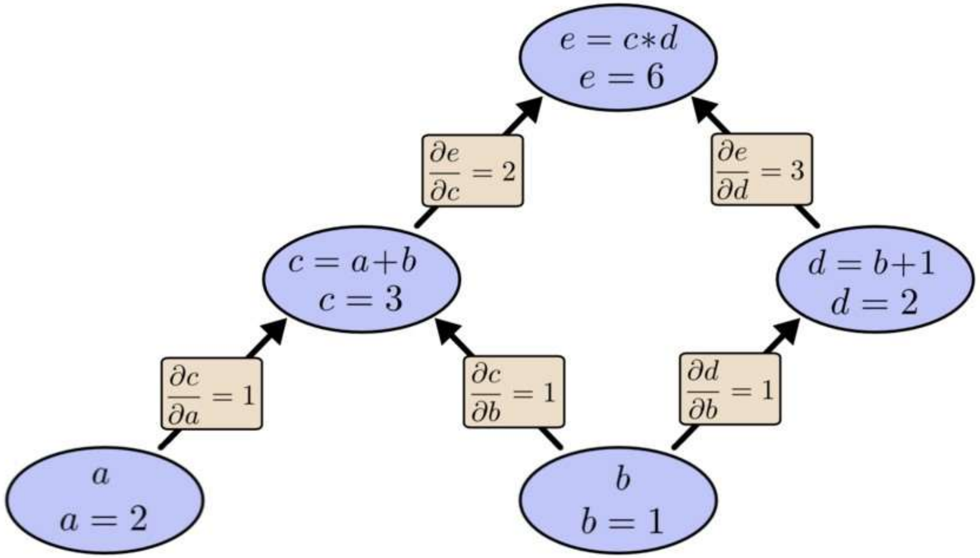
  引入两个中间变量c和d： c=a+b， d=b+1， e=c∗d



ref： http://colah.github.io/posts/2015-08-Backprop/

反向传播算法

◆ 链式法则



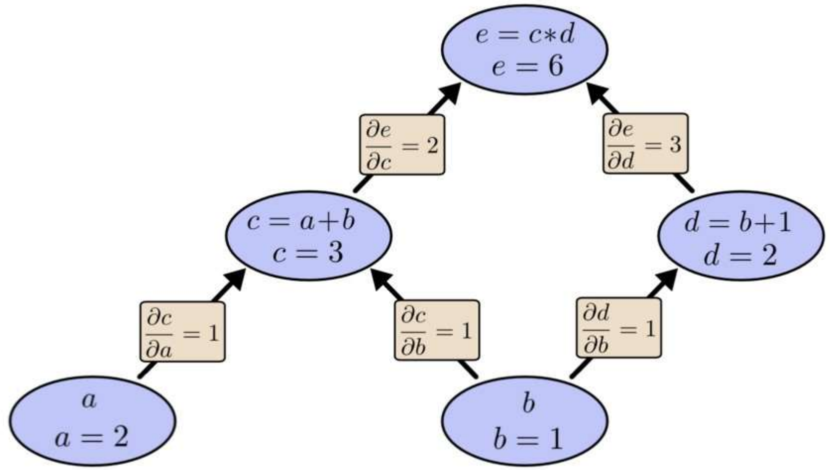
- $\frac{\partial e}{\partial a} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial a}$， 图中$\frac{\partial e}{\partial a}$的值等于从a到e的路径上的偏导值的乘积

- $\frac{\partial e}{\partial b} = \frac{\partial e}{\partial c} \cdot \frac{\partial c}{\partial b} + \frac{\partial e}{\partial d} \cdot \frac{\partial d}{\partial b}$， 上图中$\frac{\partial e}{\partial b}$的值等于从b到e的路径 (b-c-e)上的偏导值的乘积加上路径(b-d-e)上的偏导值的乘积。

- 若自下而上求解，很多路径被重复访问了。

比如图中，求$\frac{\partial e}{\partial a}$需要计算路径a-c-e， 求$\frac{\partial e}{\partial b}$都需要计算路径b-c-e和b-d-e, 路径c-e被访问了两次。

反向传播算法



◆ 链式法则

自上而下：从最上层的节点e开始，对于e的下一层的所有子节点，将e的值（e是最顶点，值=1）乘以e到某个节点路径上的偏导值，并将结果发送到该子节点中。 该子节点的值被设为“发送过来的值”，继续此过程向下传播.

第一层：节点e初始值为1

第二层：节点e向节点c发送1*2，节点e向节点得d发送1*3，

节点c值为2.　节点d值为3.

第三层： 节点c向a发送2*1，节点c向b发送2*1，节点d向b发送3*1

节点a值为2.　节点b值为为2*1+3*1=5.

即顶点e对a的偏导数为2　 顶点e对b的偏导数为5

反向传播算法

- 学习准则
  - 目的

$$\min E(w) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

  - 学习方法

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}$$

- 计算梯度

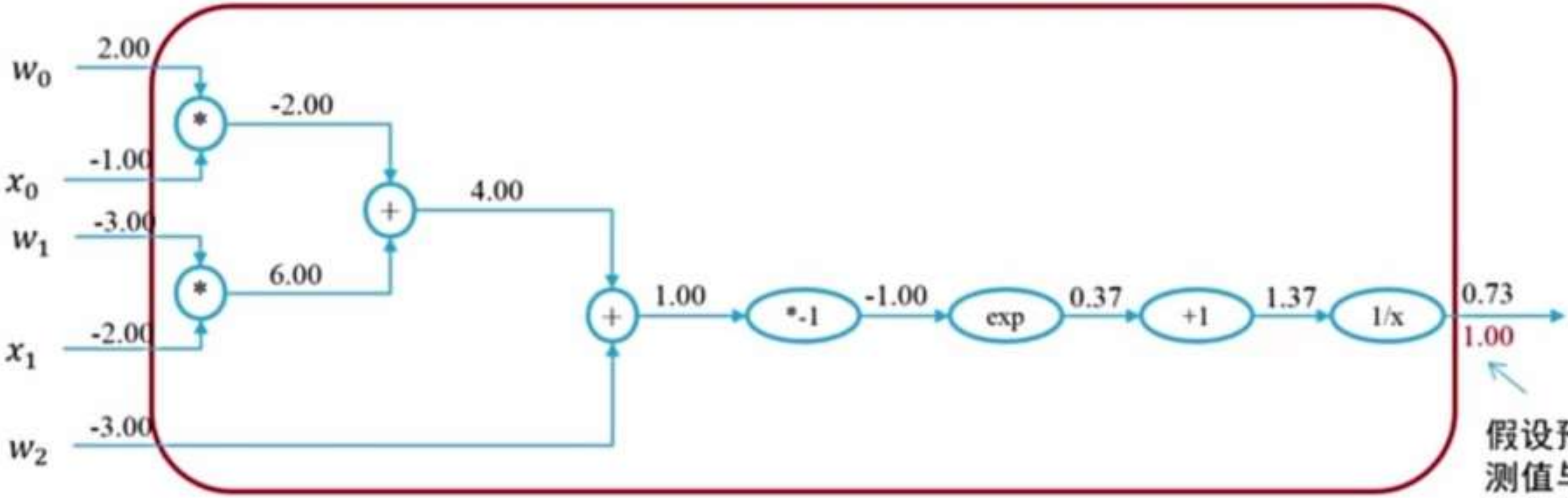
$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot x_{ji}$$

其中$net_j = \sum_j w_{ji} x_{ji}$

w_{ji}—与单元j的第i个输入相关连的权值

误差反向传播例子

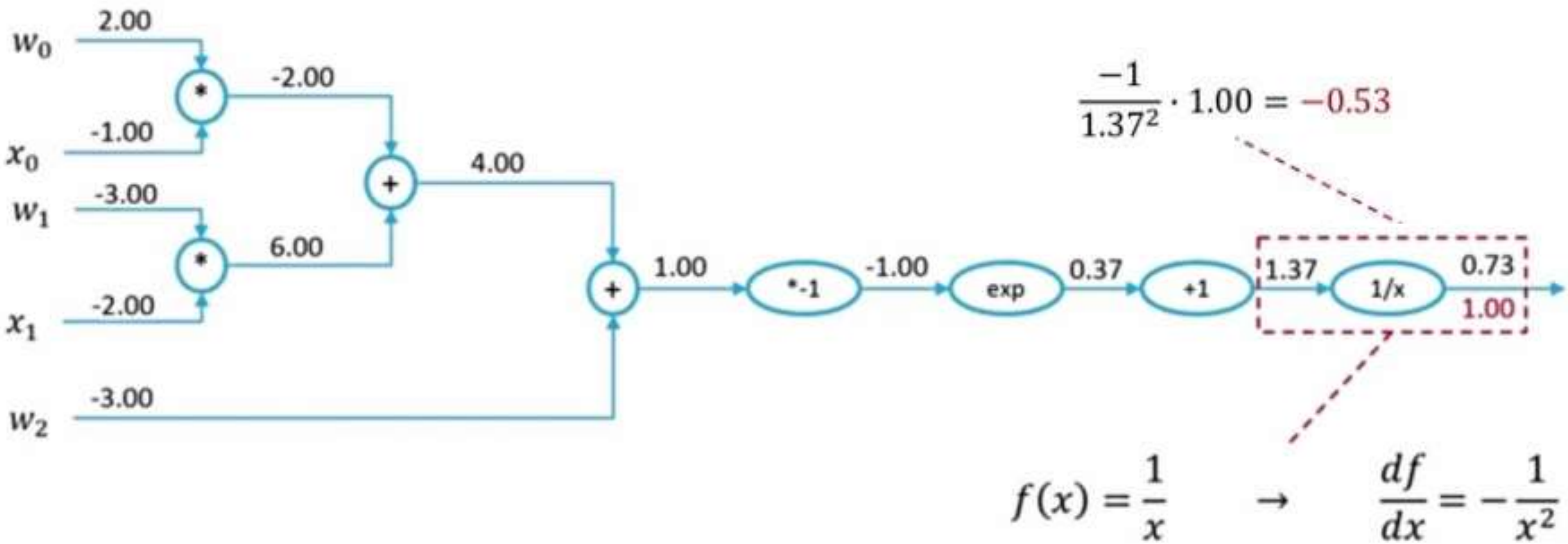
$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



假设预测值与真实值误差为1

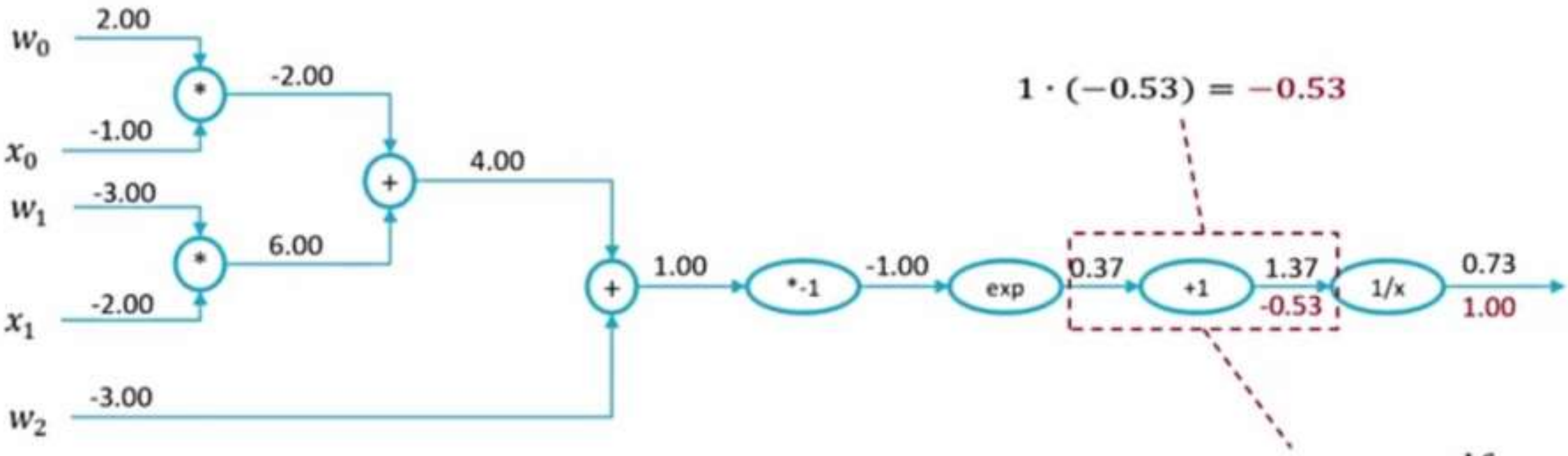
误差反向传播例子

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



误差反向传播例子

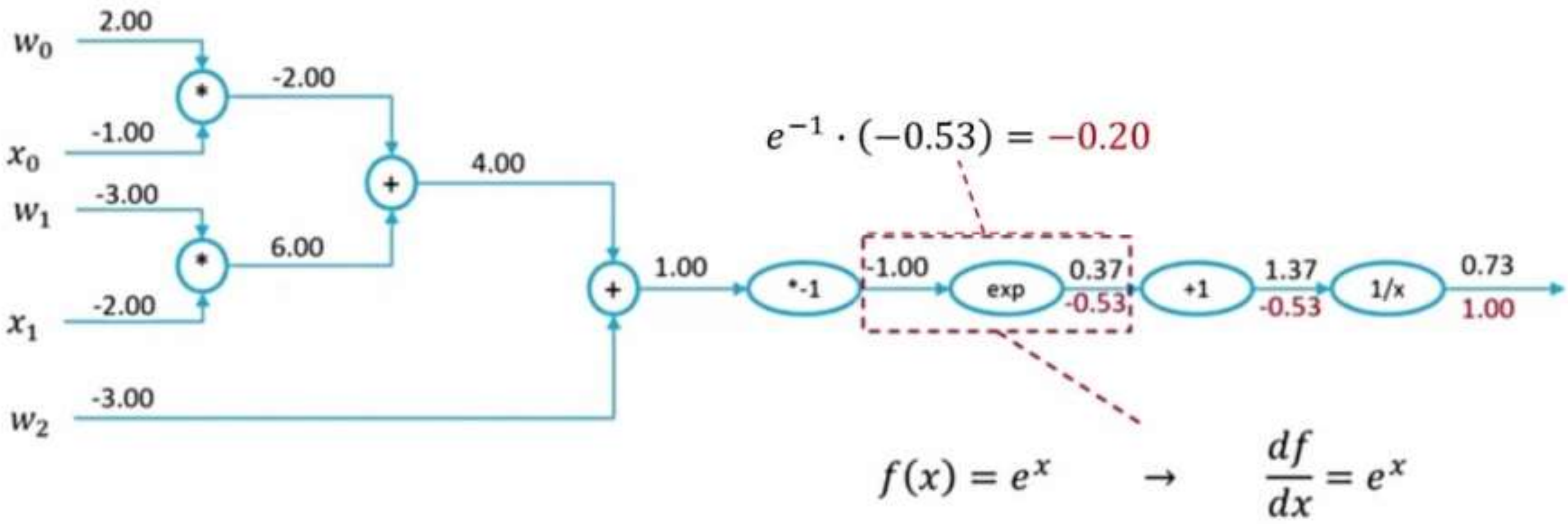
$$f(w,x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



$$f(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

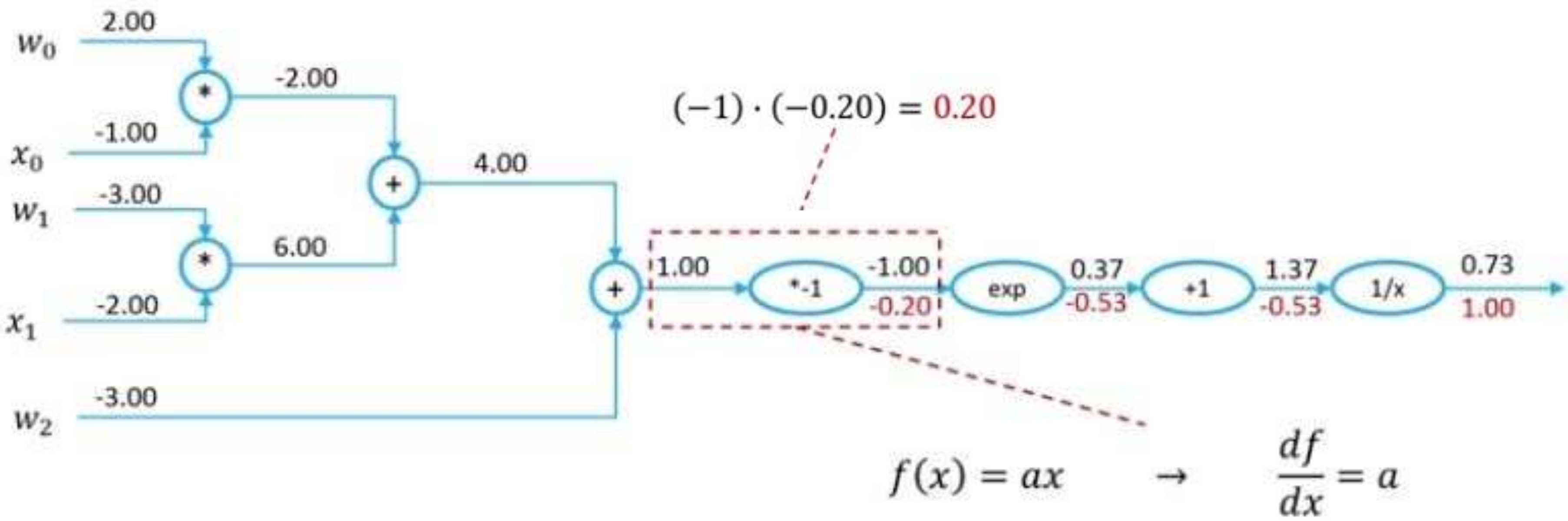
误差反向传播例子

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



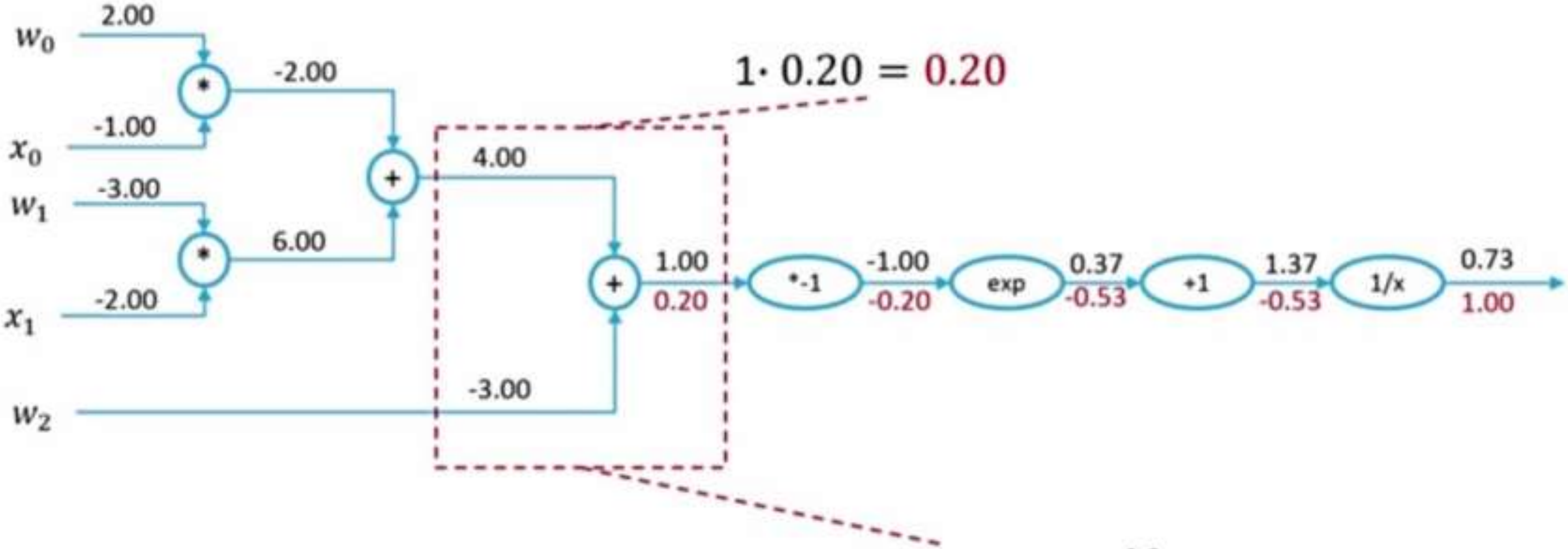
误差反向传播例子

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



误差反向传播例子

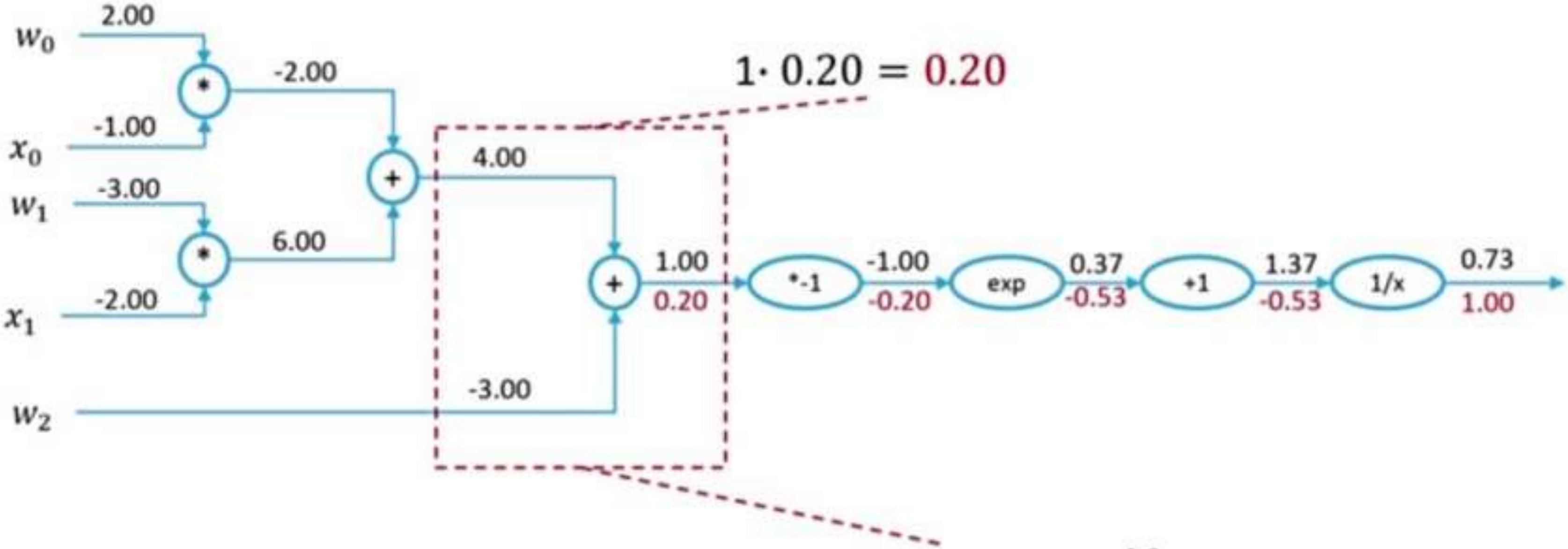
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$$

误差反向传播例子

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

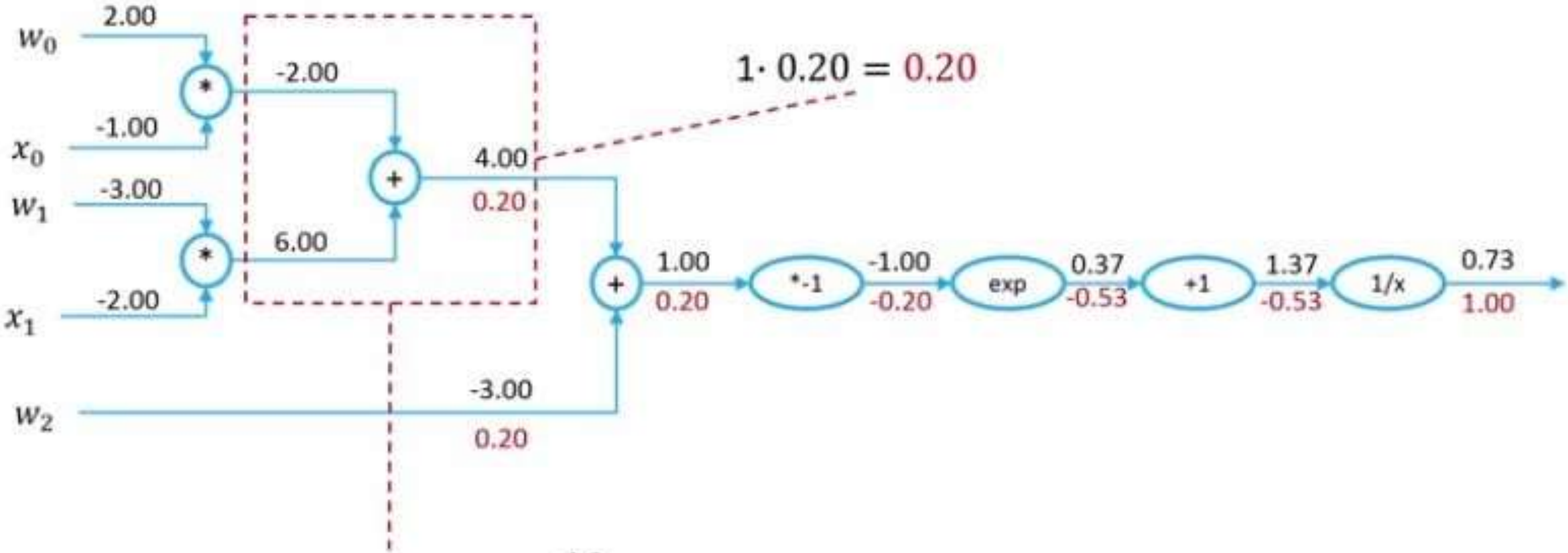


$1 \cdot 0.20 = 0.20$

$f(x) = x + x' \quad \rightarrow \quad \frac{df}{dx} = 1$

误差反向传播例子

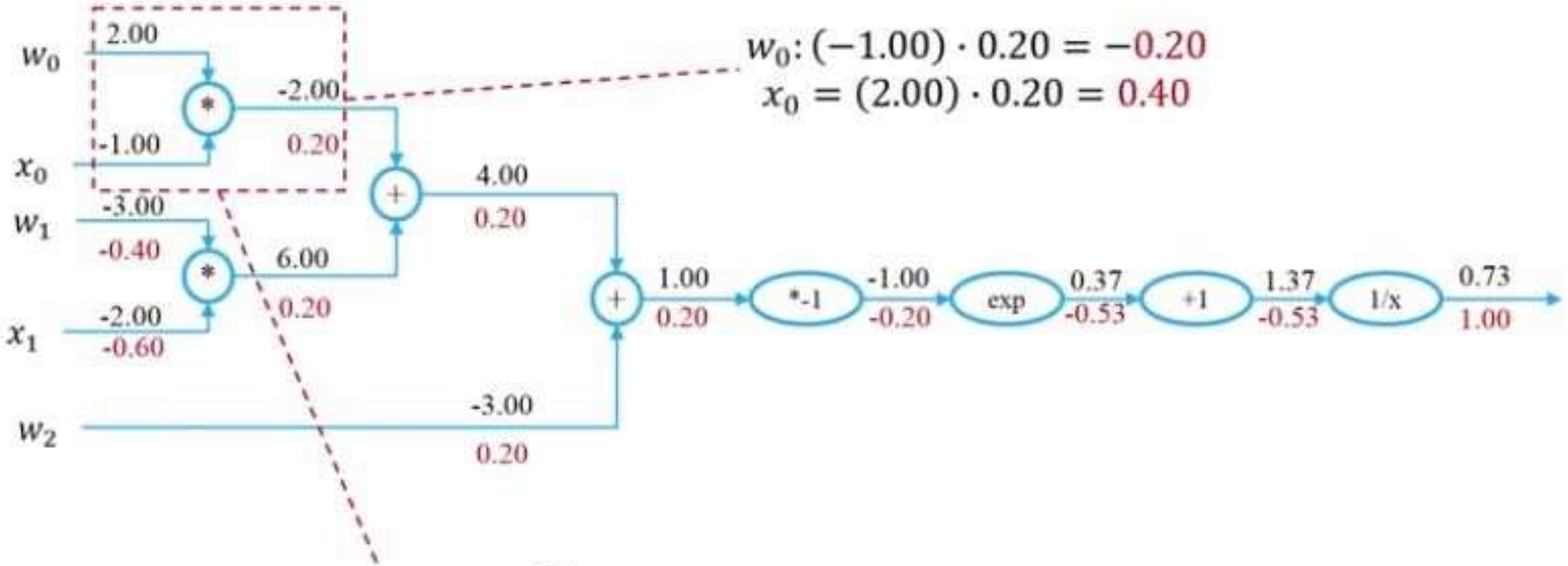
$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x)=x+x' \quad \rightarrow \quad \frac{df}{dx}=1$$

误差反向传播例子

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

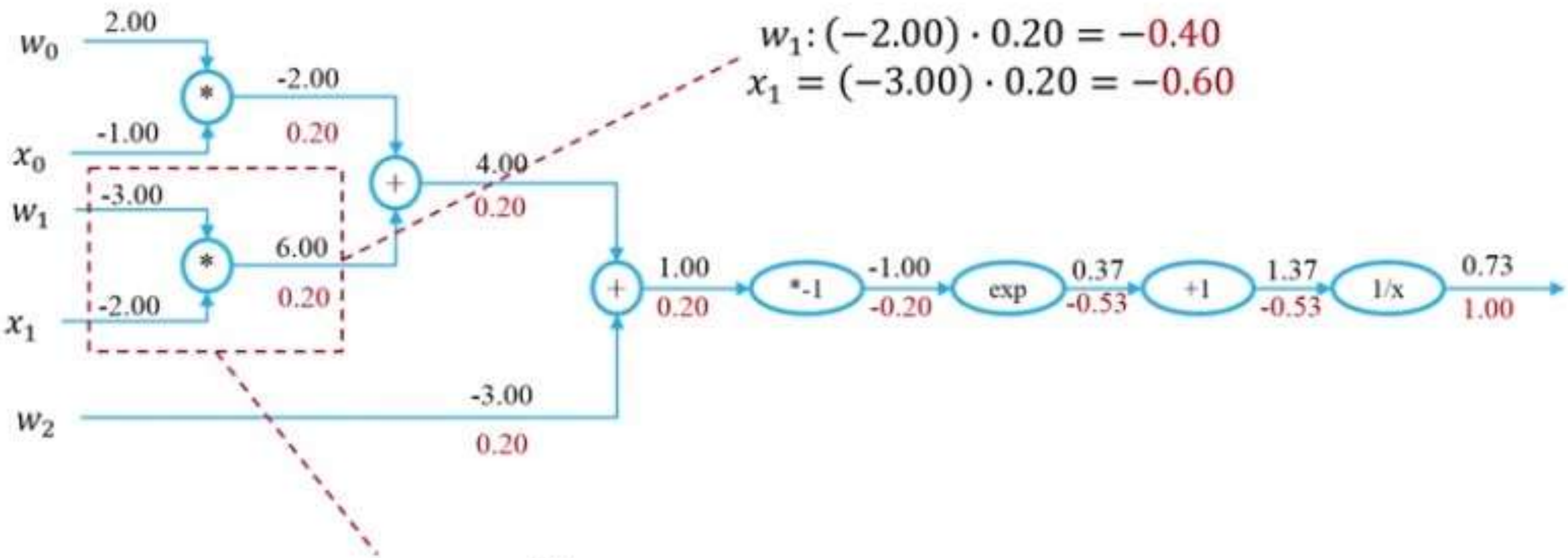


$$w_0: (-1.00)\cdot 0.20 = -0.20$$
$$x_0 = (2.00)\cdot 0.20 = 0.40$$

$$f(x) = x\cdot x' \quad\rightarrow\quad \frac{df}{dx} = x'$$

误差反向传播例子

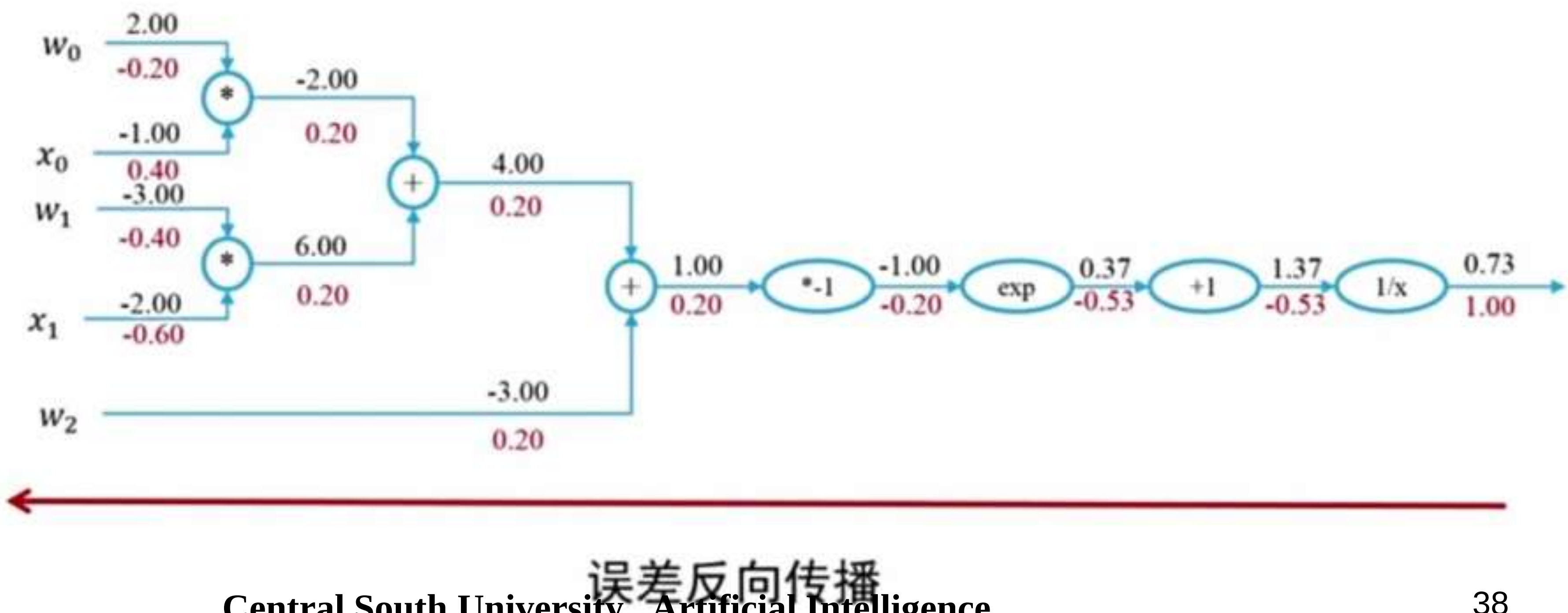
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



$$f(x) = x\cdot x' \quad \rightarrow \quad \frac{df}{dx} = x'$$

误差反向传播例子

$$f(w,x)=\frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



反向传播算法

- 计算梯度

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

分两种情况计算$\frac{\partial E_d}{\partial net_j}$

- 情况1 :输出单元

$$\frac{\partial E_d}{\partial net_j} = \boxed{\frac{\partial E_d}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k\in outputs}(t_k - o_k)^2 \cdot \frac{\partial f(net_j)}{\partial net_j}$$

除了$k=j$外，其他输出单元k的导数$\frac{\partial}{\partial o_j}(t_k - o_k)^2$为0

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 = \frac{1}{2} \times 2(t_j - o_j)\frac{\partial}{\partial o_j}(t_j - o_j) = -(t_j - o_j)$$

$$\frac{\partial f(net_j)}{\partial net_j} = o_j(1 - o_j)$$

所以： $\frac{\partial E_d}{\partial net_j} = \boxed{-(t_j - o_j)o_j(1 - o_j)}$ 令 $\frac{\partial E_d}{\partial net_j} = -\delta_j$

则： $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot x_{ji} = -\delta_j x_{ji}$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \delta_j x_{ji}$$

反向传播算法

- 计算梯度
  - 情况2 :隐层单元

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$\frac{\partial E_d}{\partial net_j} = \sum_k \frac{\partial E_d}{\partial net_k} \cdot \frac{\partial net_k}{\partial net_j} = \sum_k -\delta_k \cdot \frac{\partial net_k}{\partial net_j} = \sum_k -\delta_k \cdot \frac{\partial net_k}{\partial o_j} \cdot \frac{\partial o_j}{\partial net_j}$$

$$= \sum_k -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} = \sum_k -\delta_k w_{kj} o_j (1 - o_j)$$

注意：这里激活函数为Sigmoid函数

令 $\frac{\partial E_d}{\partial net_j} = -\delta_j$

k为所有与j连接的下游神经元

即： $\delta_j = o_j (1 - o_j) \sum_k \delta_k w_{kj}$

则： $\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \cdot x_{ji} = -\delta_j x_{ji}$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta \delta_j x_{ji}$$

Backpropagation Learning

- BP算法中权值的修正公式

$$\Delta w_{ji} = \eta \delta_j o_i$$

$$\delta_j = o_j(1-o_j)(t_j - o_j) \qquad \text{if } j \text{ is an output unit}$$

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj} \qquad \text{if } j \text{ is a hidden unit}$$

- η is a constant called the learning rate

  t_j is the correct (expected-teacher) output for unit j

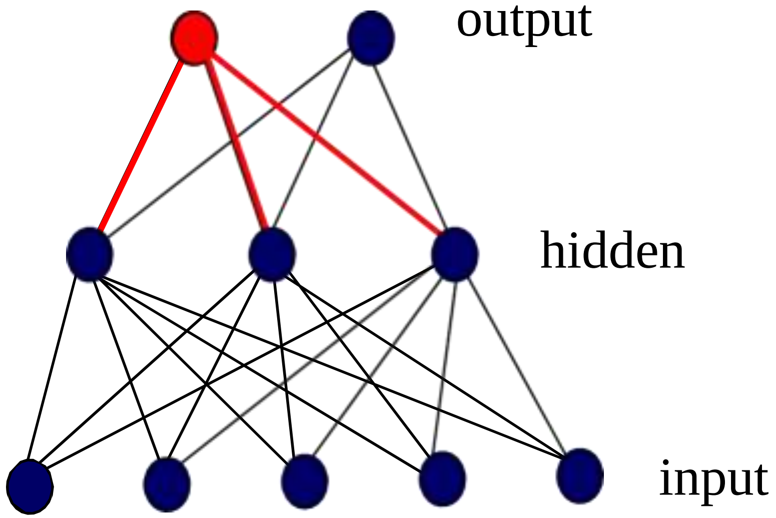
  o_j is the computed (current) output for unit j

  δ_j is the error measure for unit j

Error Backpropagation step 1

- 首先计算输出层单元的误差，并用该误差调整输出层的权值

Current output: o_j=0.2
Correct output: t_j=1.0
Error $\delta_j = o_j(1-o_j)(t_j-o_j)$
0.2(1–0.2)(1–0.2)=0.128



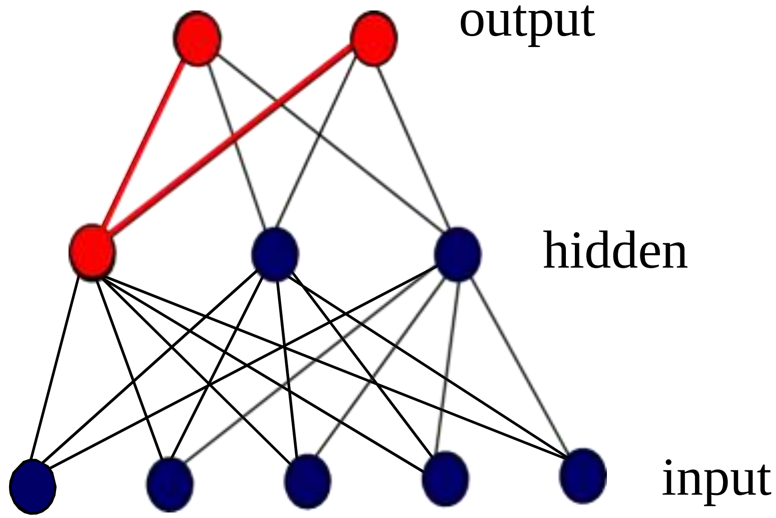
Update weights into j

$$\Delta w_{ji} = \eta \delta_j o_i$$

Error Backpropagation step 2

- 接着根据输出层的误差计算隐层单元的误差.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$



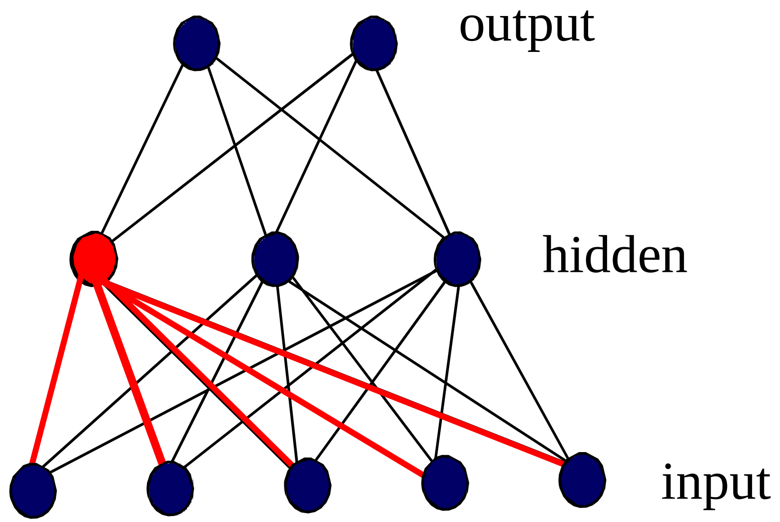
Error Backpropagation step 3

- 最后根据隐层单元的误差调整下层的权值.

$$\delta_j = o_j(1-o_j)\sum_k \delta_k w_{kj}$$

Update weights into *j*

$$\Delta w_{ji} = \eta \delta_j o_i$$

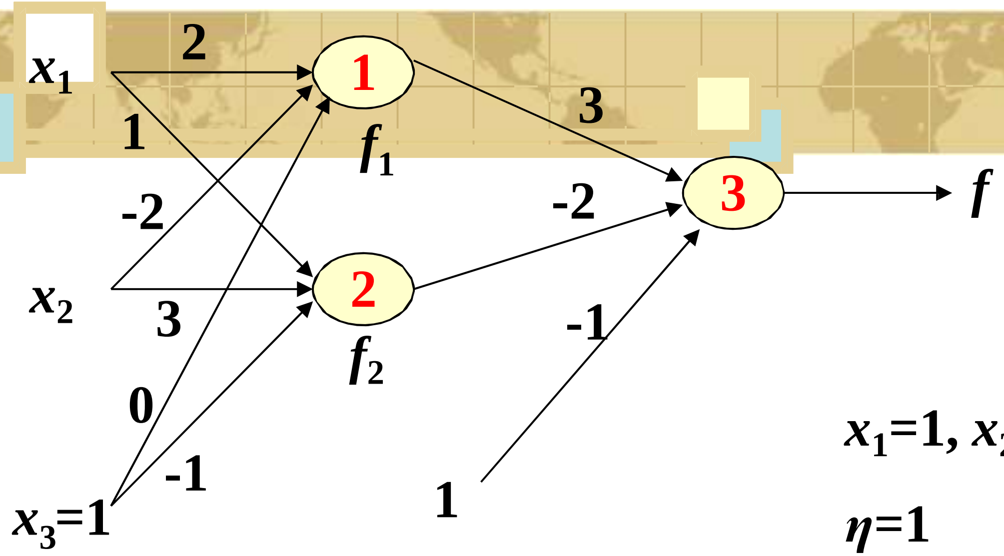


Backpropagation Learning

- BP算法

1. 初始化权及阈值为小的随机数
2. 给出输入x_0，$x_1 \ldots x_{n-1}$及期望输出d_0，d_1，d_{n-1}
3. 逐层计算输出
4. 修正权值
5. 重复（3）~（5），直到对所有样本权值不变

Example:



x_1=1, x_2 =0, x_3 =1, t=0

η=1

$$f_1=s(1\times2+0\times(-2)+1\times0)=s(2)=1/(1+e^{-2})=0.881$$

$$f_2=s(1\times1+0\times3+1\times(-1))=s(0)=1/(1+e^{0})=0.5$$

$$f=s(0.881\times3+0.5\times(-2)+1\times(-1))=s(0.643)=0.655$$

$$\delta=0.655\times(1-0.655)\times(0-0.655)=-0.148$$

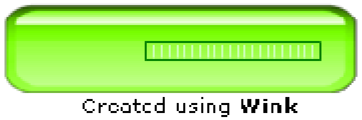
$$\delta_1=0.881\times(1-0.881)\times(-0.148\times3)=-0.047$$

$$\delta_2=0.5\times(1-0.5)\times(-0.148\times-2)=0.074$$

$$w_{11}=2+1\times(-0.047)=1.953 \qquad w_{12}=-2+0\times(-0.047)=-2$$

$$w_{21}=1+1\times0.074=1.074 \qquad w_{22}=3+0\times0.074=3$$

$$w_{31}=3+0.881\times(-0.148)=2.870 \qquad w_{32}=-2+0.5\times(-0.148)=-2.074$$


Created using Wink

Sample: Matlab & BP

- 药品销量预测
  - 下表为某药品的销售情况
  - 预测方法采用滚动预测方式，即用前三个月的销售量来预测第四个月的销售量，如此反复直至满足预测精度要求为止。
  - 构建三层BP神经网络：输入层有三个结点，隐含层 结点数为5，隐含层的激活函数为tansig；输出层 结点数为1个，输出层的激活函数为logsig

月份	1	2	3	4	5	6
销量	2056	2395	2600	2298	1634	1600
月份	7	8	9	10	11	12
销量	1873	1478	1900	1500	2046	1556

Sample: Matlab & BP

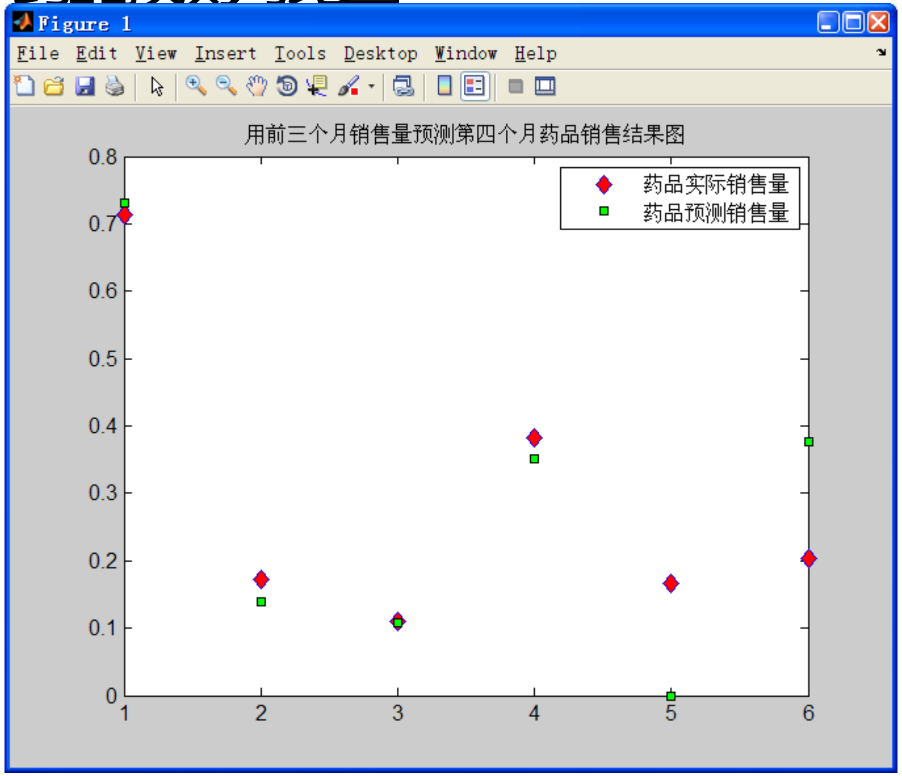
- %以每三个月的销售量经归一化处理后作为输入
  - P=[0.5152 0.8173 1.0000 ;
    0.8173 1.0000 0.7308;
    1.0000 0.7308 0.1390;
    0.7308 0.1390 0.1087;
    0.1390 0.1087 0.3520;
    0.1087 0.3520 0.0000;]';
- %以第四个月的销售量归一化处理后作为目标向量=[0.7308 0.1390
  - 0.1087 0.3520 0.0000 0.3761];

Sample: Matlab & BP

- %创建一个BP神经网络，每一个输入向量的取值范围为 [0 ,1]，隐含层有5个神经元，输出层有一个神经元， 隐含层的激活函数为tansig，输出层的激活函数为logsig，训练函数为梯度下降函数
  - net=newff([0 1;0 1;0 1], [5,1], {'tansig','logsig'},'traingd');
  - net.trainParam.epochs=15000;
- %设置学习速率为0.1
  - LP.lr=0.1;
  - net.trainParam.goal=0.01;
- net=train(net,P,T);

Sample: Matlab & BP

- BP网络应用于药品预测对比图



- 由对比图可以看出预测效果与实际存在一定误差，此误差可以通过增加运行步数和提高预设误差精度业进一步缩小

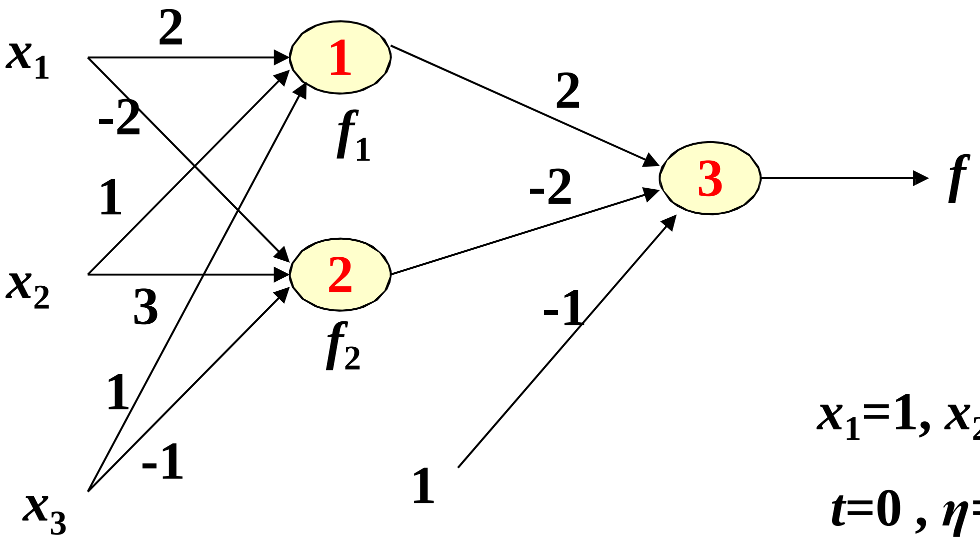
Successful Applications

- Text to Speech (NetTalk)
- Fraud detection
- Financial Applications
  - HNC (eventually bought by Fair Isaac)
- Chemical Plant Control
  - Pavillion Technologies
- Automated Vehicles
- Game Playing
  - Neurogammon
- Handwriting recognition

- 焦李成等. **神经网络七十年：回顾与展望**[J]. 计算机学报, 2016, 39(8): 1697-1716.
- AI Studio[EB/OL]. https://aistudio.baidu.com/aistudio/course

- 从浅层网络到深度网络
- 卷积神经网络

作业（30分）



x_1=1, x_2 =0, x_3 =1,

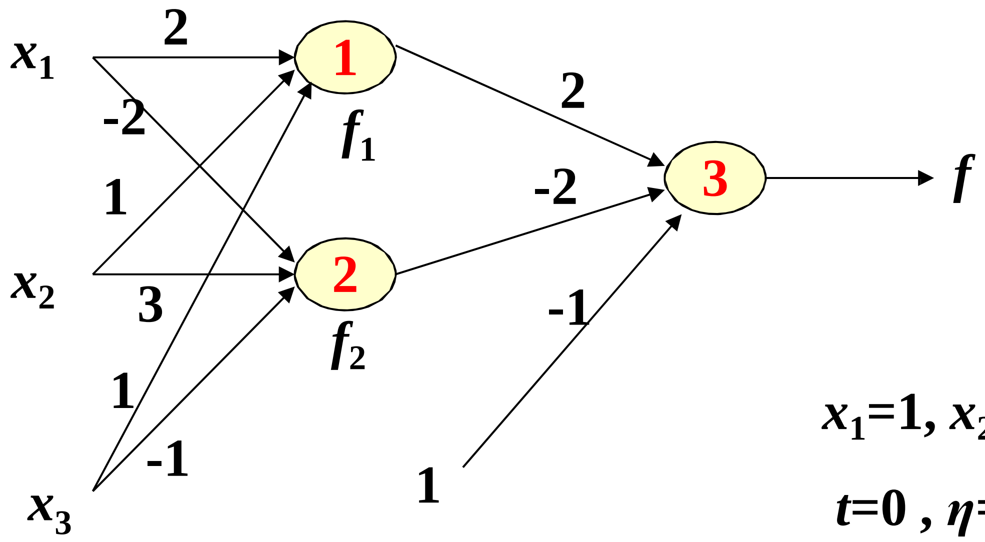
t=0 , η=0.1

假设有如上结构的神经网络，其中每个神经元为**Sigmoid**函数。

（1）如果初始值为x_1=1,x_2=0,x_3=1, 试给出所有隐藏层和输出层神经元的输入；

（2）利用反向传播算法更新参数，学习率为0.1，试给出第一次更新后的参数值，

作业（30分）



x_1=1, x_2 =0, x_3 =1,

t=0 , η=0.1

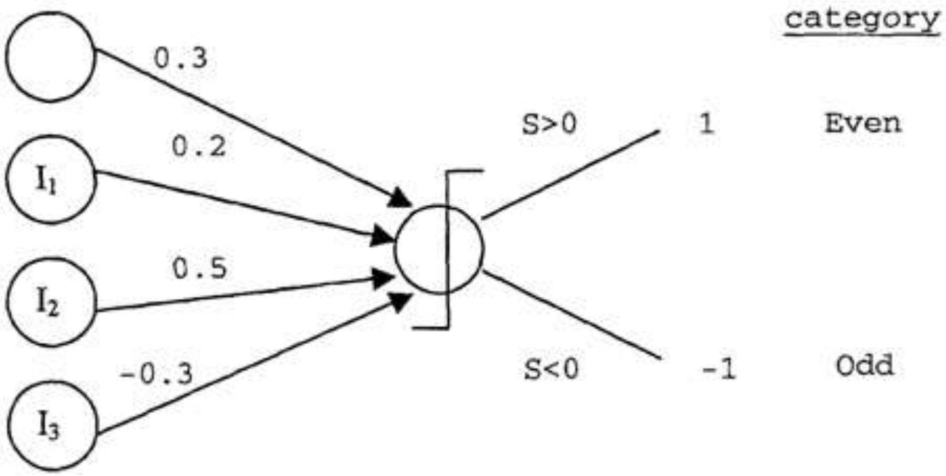
假设有如上结构的神经网络，其中每个神经元为**ReLU**函数。

（1）如果初始值为x_1=1,x_2=0,x_3=1, 试给出所有隐藏层和输出层神经元的输入；

（2）利用反向传播算法更新参数，学习率为0.1，试给出第一次更新后的参数值，

注意：权值的修正公式 δ_j 变化

作业（40分）



- 训练一个感知器用以判断输入的三个整数之积是偶数还是奇数。该感知器有三个输入I_1~I_3分别对应输入三个整数，若是偶数，则输入值为+1，否则为-1。

1 为什么需要第4个输入端？其输入值应该设为多少？

2 对于2*3*4，该感知判断其结果是奇数还是偶数？

实验三 神经网络

一、实验目的

理解反向传播网络的结构和原理，掌握反向传播算法对神经元的训练过程，了解反向传播公式。通过构建 BP 网络实例，熟悉前馈网络的原理及结构。

二、实验内容

1. 通过 BP 网络各项参数的不同设置，观察 BP 算法的学习效果。观察比较 BP 网络 拓扑结构及其它各项参数变化对于训练结果的影响。观察并分析不同训练数据对相同拓扑结构的 BP 网络建模的影响。
2. 设计简单的感知器，实现简单的逻辑运算（与、或）等，也可做其他更复杂的问题。

实验三 神经网络

三、实验环境

以下两种实验环境可供选择：

1. 神经网络可视化实验环境，如图 3 所示。

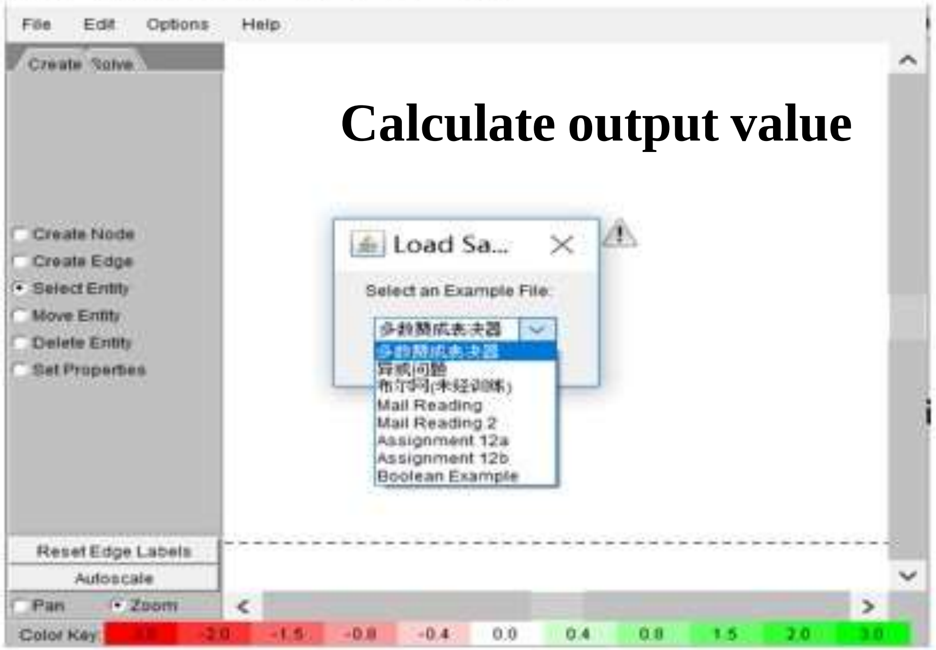


图 3 神经网络可视化实验环境

2. C++语言编程环境，语言环境可以自选。

总训练误差变化图

训练50步后的误差

训练多于50步时的误差

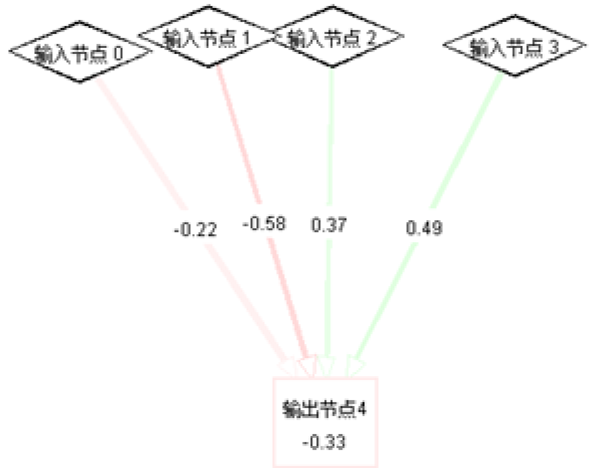
模拟的问题或函数

观察结果

学生结论

实验目的	实现3位数乘积奇偶判别器。 通过改变数据集，观察对于实验的影响。

网络拓朴图



训练数据集	T: 输入节点 0, 输入节点 1, 输入节点 2, 输入节点 3, 输出节点 4; A: 1, 1, 0, 1, 1; A: 1, 0, 0, 0, 0; A: 1, 0, 1, 0, 1; A: 1, 0, 1, 1, 1; A: 1, 1, 1, 1, 1; A: 1, 1, 1, 0, 1;	T: 输入节点 0, 输入节点 1, 输入节点 2, 输入节点 3, 输出节点 4; A: 1, 1, 0, 1, 1; A: 1, 0, 0, 0, 0; A: 1, 0, 1, 0, 1; A: 1, 0, 1, 1, 1; A: 1, 1, 1, 1, 1; A: 1, 1, 1, 0, 1;	T: 输入节点 0, 输入节点 1, 输入节点 2, 输入节点 3, 输出节点 4; A: 1, 1, 0, 1, 1; A: 1, 0, 0, 0, 0; A: 1, 0, 1, 0, 1; A: 1, 0, 1, 1, 1;

rror: 0.53

005 after 250 steps

训练)

向传播和

可选

- **任务内容：**本次利用飞桨动态图搭建一个全连接神经网络，对包含不同车辆的图像进行分类。
- **实践平台：**百度AI实训平台-AI Studio
- **实践环境：**Python3.7，飞桨2.0

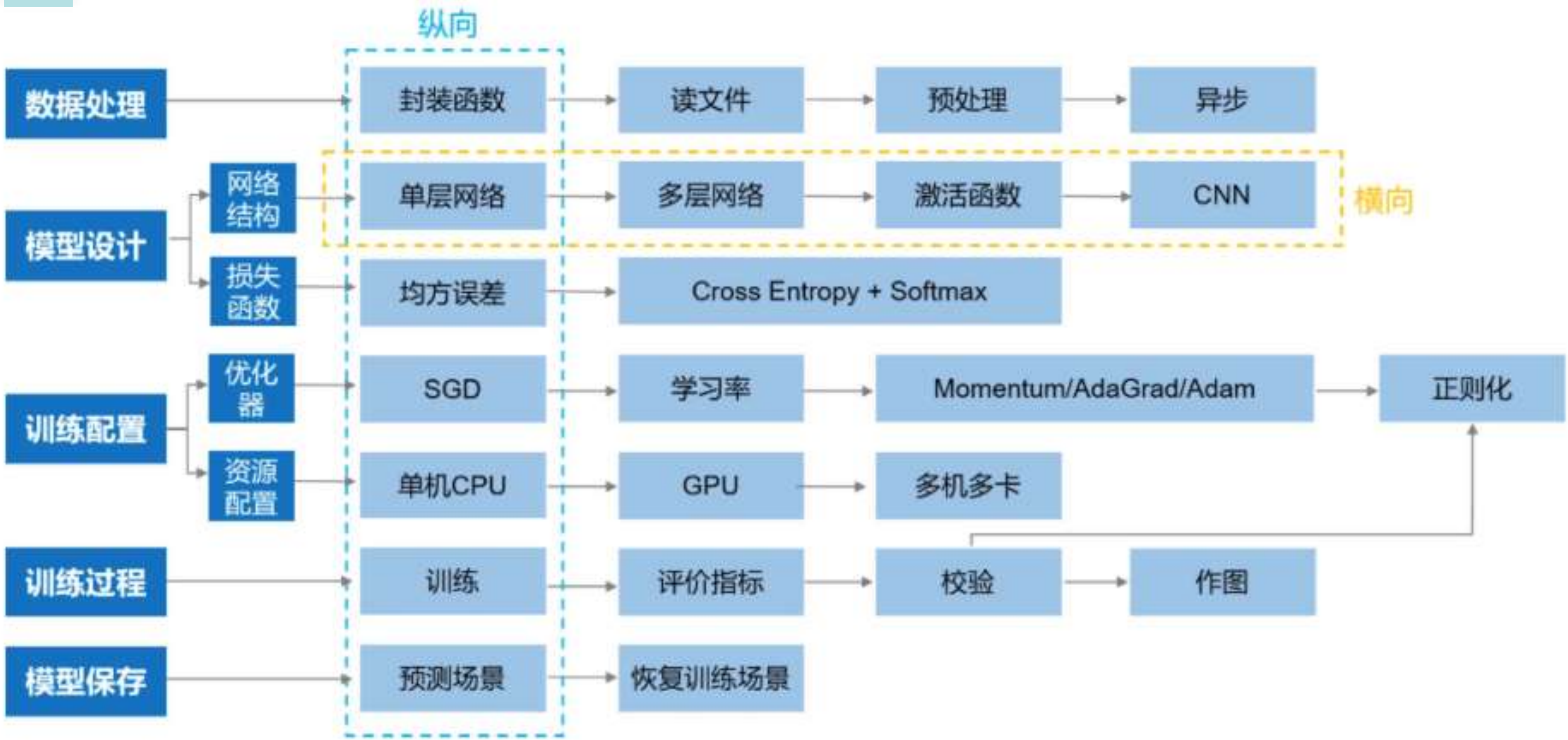
数据说明里 标签值说明：1=“汽车”'，2=“摩托车”，3=“货车”

https://aistudio.baidu.com/aistudio/projectdetail/1799981?channelType=0&channel=0

◆ 数据集介绍

- 网上公开的车辆图像数据集：
  - ✓ 数据来源：摩托车、家用汽车、货车数据来自2005 PASCAL视觉类挑战赛（VOC2005）所使用的数据的筛选处理结果，货车图片来自网络收集，后期通过筛选处理得到。
  - ✓ 数据格式：3*120*120

- 特别提示：本实践所用数据集均来自互联网，请勿用于商务用途。


纵向
数据处理
封装函数
读文件
预处理
异步
模型设计
网络结构
单层网络
多层网络
激活函数
CNN
横向
损失函数
均方误差
Cross Entropy + Softmax
训练配置
优化器
SGD
学习率
Momentum/AdaGrad/Adam
正则化
资源配置
单机CPU
GPU
多机多卡
训练过程
训练
评价指标
校验
作图
模型保存
预测场景
恢复训练场景