

神经网络

2019年度南京大学“专创融合”特色示范课程培育项目

高 阳

<http://cs.nju.edu.cn/rl>, 2019.11.05

神经网络

从生物神经网络到人工神经网络

高 阳

<http://cs.nju.edu.cn/gaoy>, 2019.11.05

大纲

人工神经网络及其发展

感知机和感知机学习

多层神经网络

反向传播算法BP

大纲

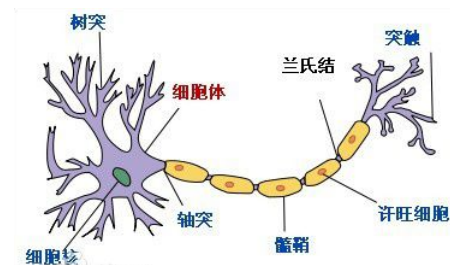
人工神经网络及其发展

感知器学习

多层神经网络

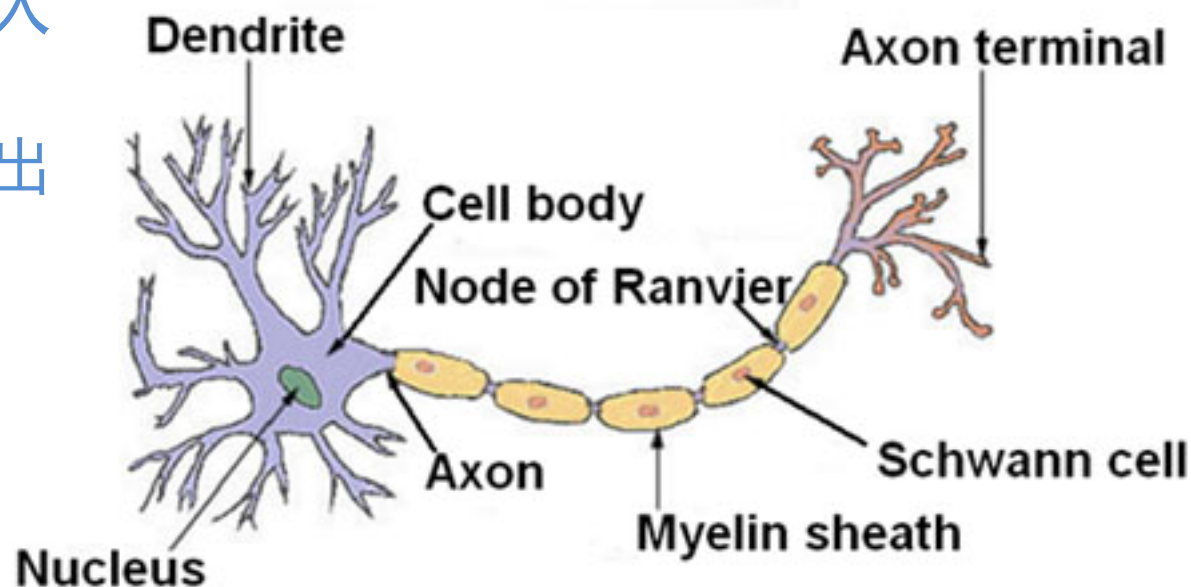
反向传播算法BP

生物学基础



树突：输入

轴突：输出



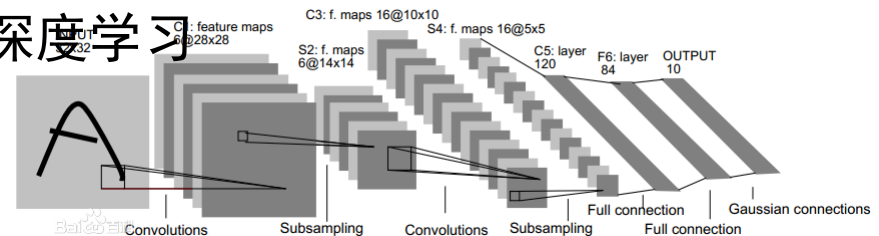
- ✓ 分布/并行计算模型
- ✓ 调整节点间的连接关系

□ 人脑

- ✓ 10^{11} 个神经元
- ✓ 每个神经元处理速度为 $10^{-3}s$
- ✓ 识别人脸，需要约100个神经元计算

发展历史

- ❑ 1943年，心理学家W. McCulloch和数理逻辑学家W. Pitts首次提出神经元的数学模型MP模型
- ❑ 1948年，冯·诺依曼提出相互再生自动机网络结构
- ❑ 1950s，F. Rosenblatt提出感知机模型
- ❑ 1960s，Widrow提出非线性多层自适应网络
- ❑ 1968年，Minsky的《感知机》一书
- ❑ 1982年和1984年，物理学家Hopfield在美国科学院院刊上发表ANN文章
- ❑ 2006年，Hinton发表深度信念网络→深度学习



神经网络表示 - ALVINN

□ ALVINN : **A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork

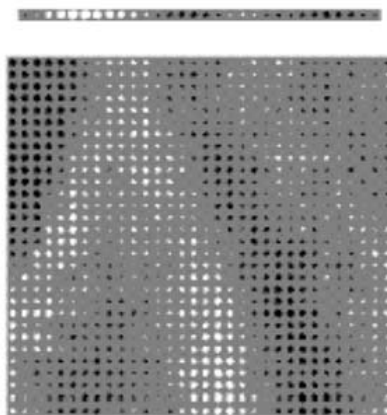
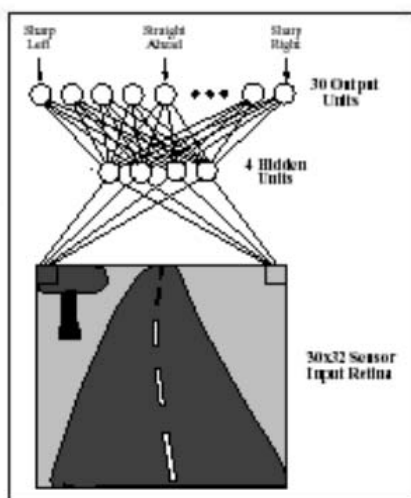


□ 1993年, CMU研发

□ 输入: 30*32的像素

□ 4个隐藏节点

□ 输出: 30个驾驶动作(急剧左转、急剧右转, 正前方行进.....)



有向/无向? 有环/无环?
结构?

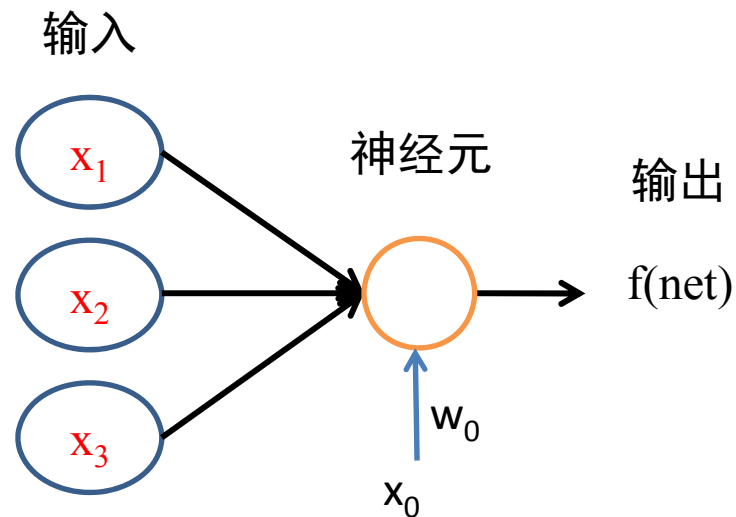
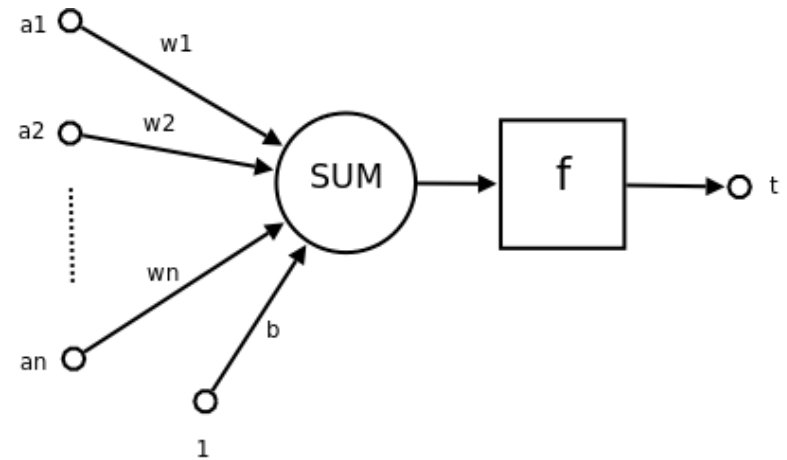
神经元基本结构

□ 输入 $X=[x_1, x_2, x_3, \dots]$

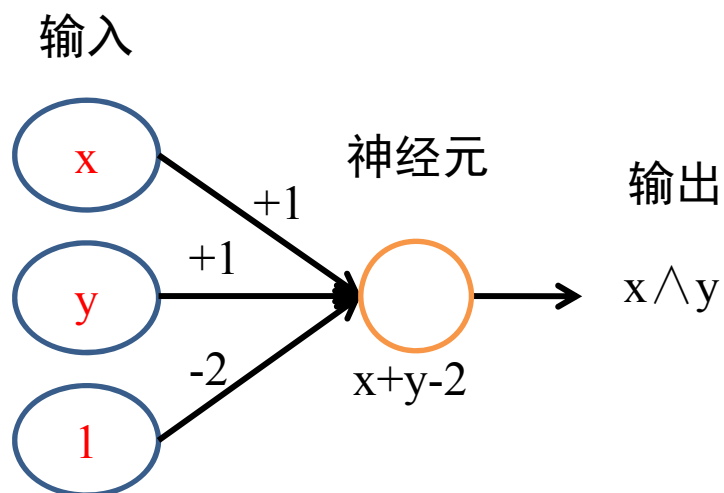
□ 权值 $W=[w_1, w_2, w_3, \dots]$

□ 激励函数 $f(\text{net}) = f(\sum(w_i * x_i))$

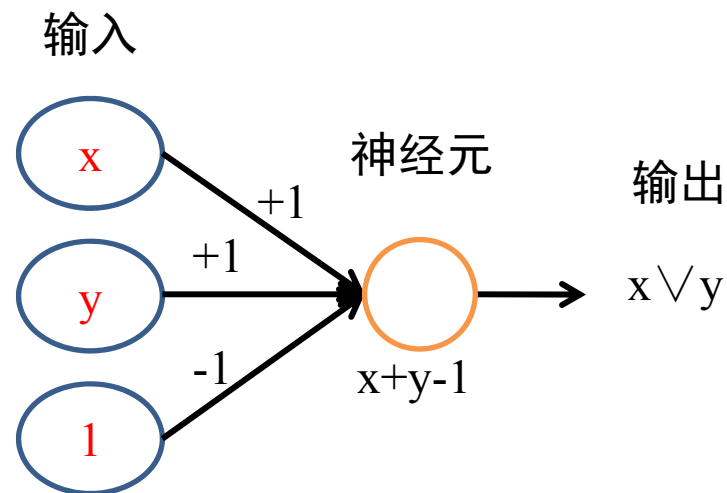
□ 偏置单元(bias unit) x_0 , 其对应权值为 w_0



MP神经元模型

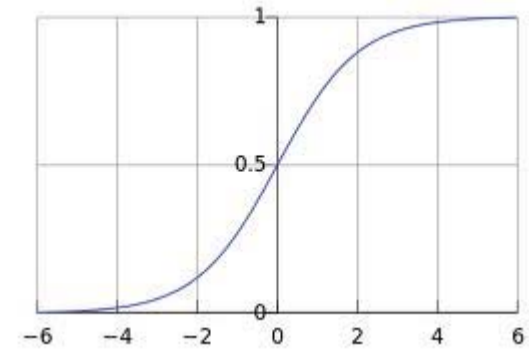
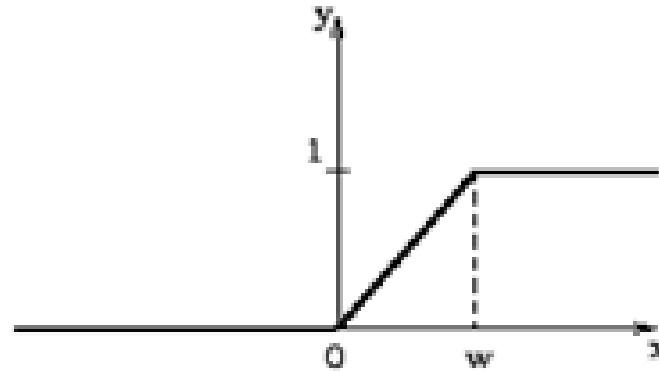
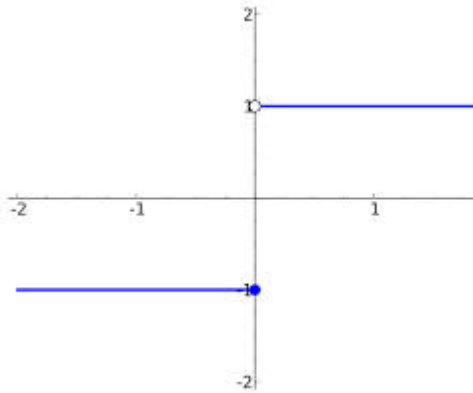


逻辑与的神经元模型
(阈值为 ≥ 0 , 大于等于输出1)



逻辑或的神经元模型
(阈值为 ≥ 0 , 小于输出0)

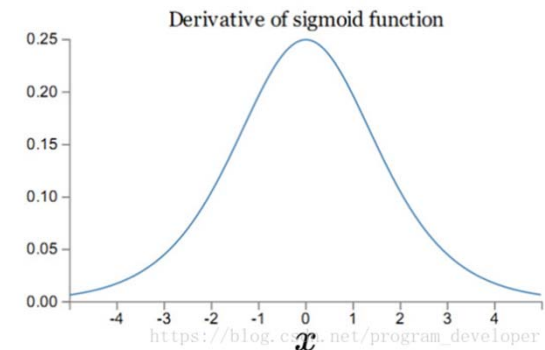
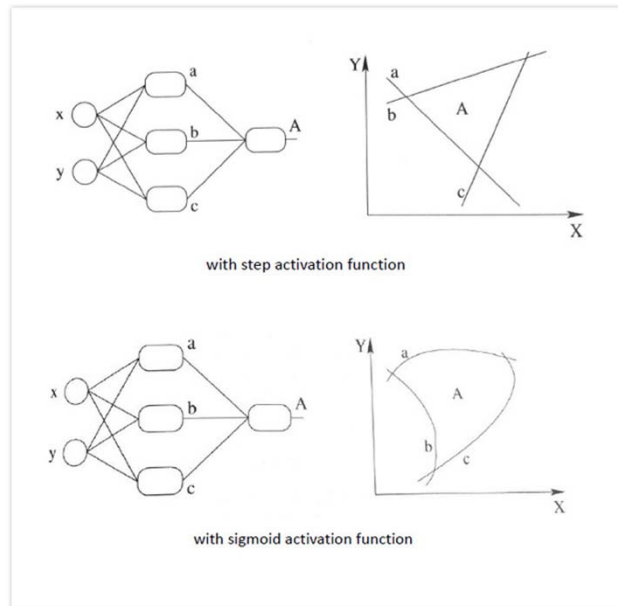
激励函数



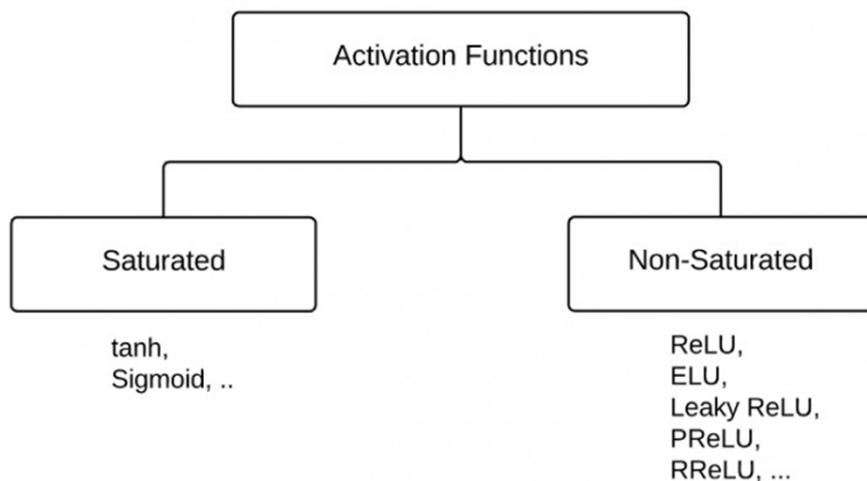
引入非线性激励函数，增强神经网络的表达能力

$$f(\text{net}) = \frac{1}{1 + e^{-\lambda * \text{net}}}$$

λ 是挤压参数，值越大，区间[0,1]上越接近直线。

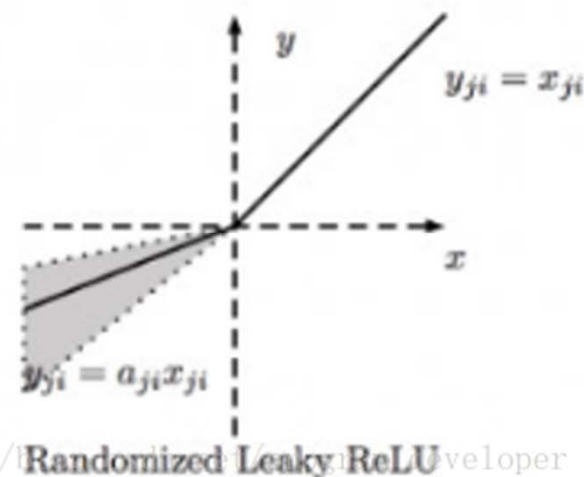
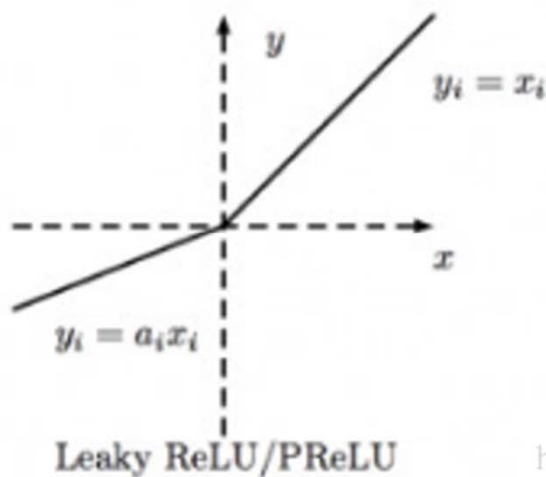
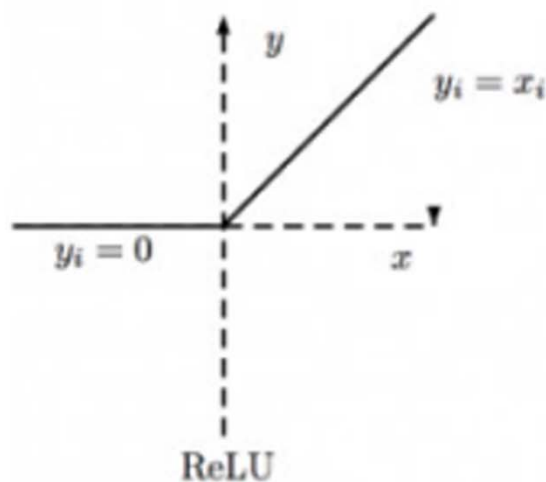


激励函数的作用



饱和型激励函数的缺点：

- 1、梯度消失
- 2、非以0为中心
- 3、指数计算代价大



<https://github.com/leventkaymak/Randomized-Leaky-ReLU-developer>

大纲

人工神经网络及其发展

感知机和感知机学习

多层神经网络

反向传播算法BP

感知机和感知机学习

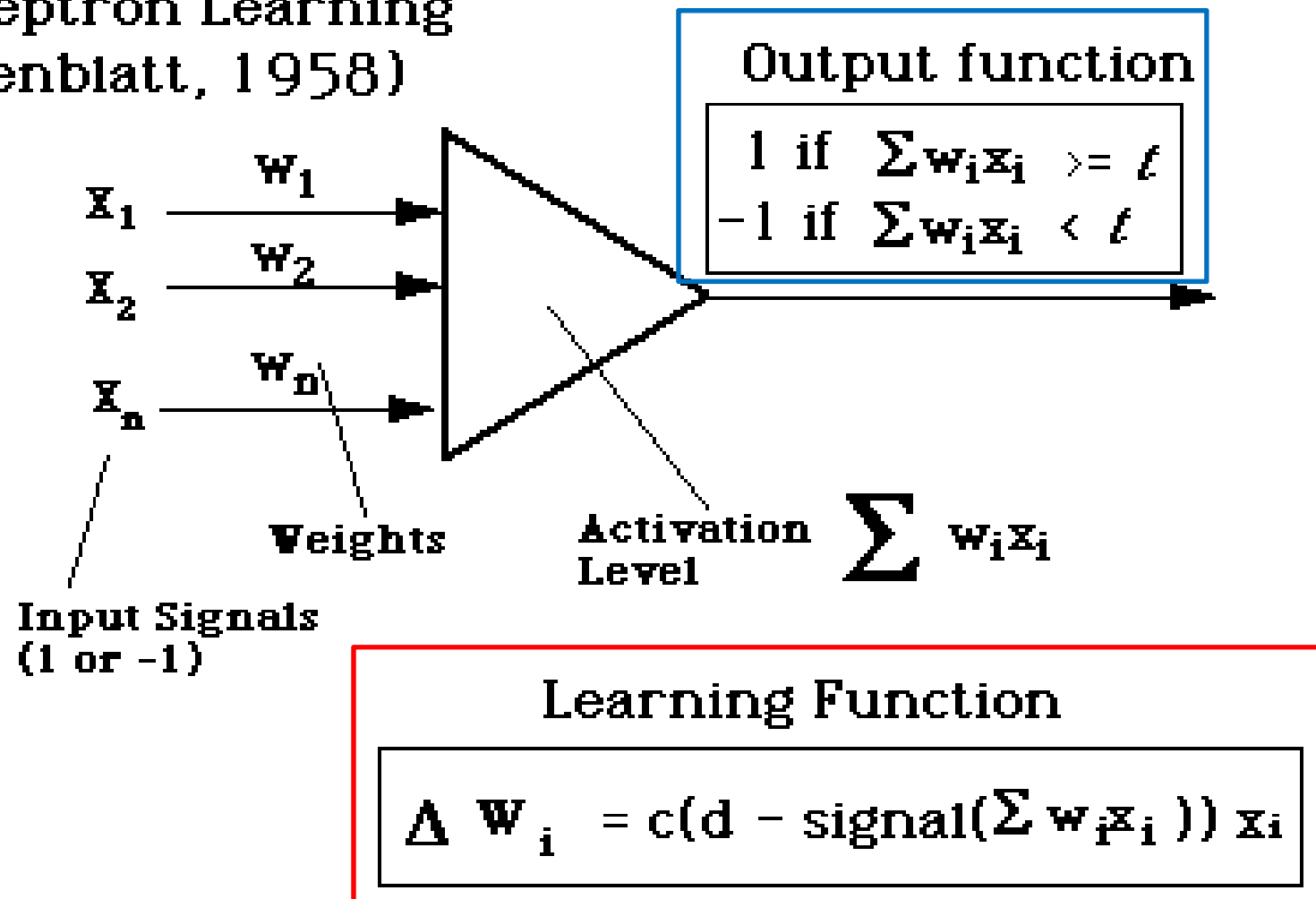
□ Frank Rosenblatt, 1957年, Cornell航空实验室(Cornell Aeronautical Laboratory)

□ 最简单形式的前馈式人工神经网络

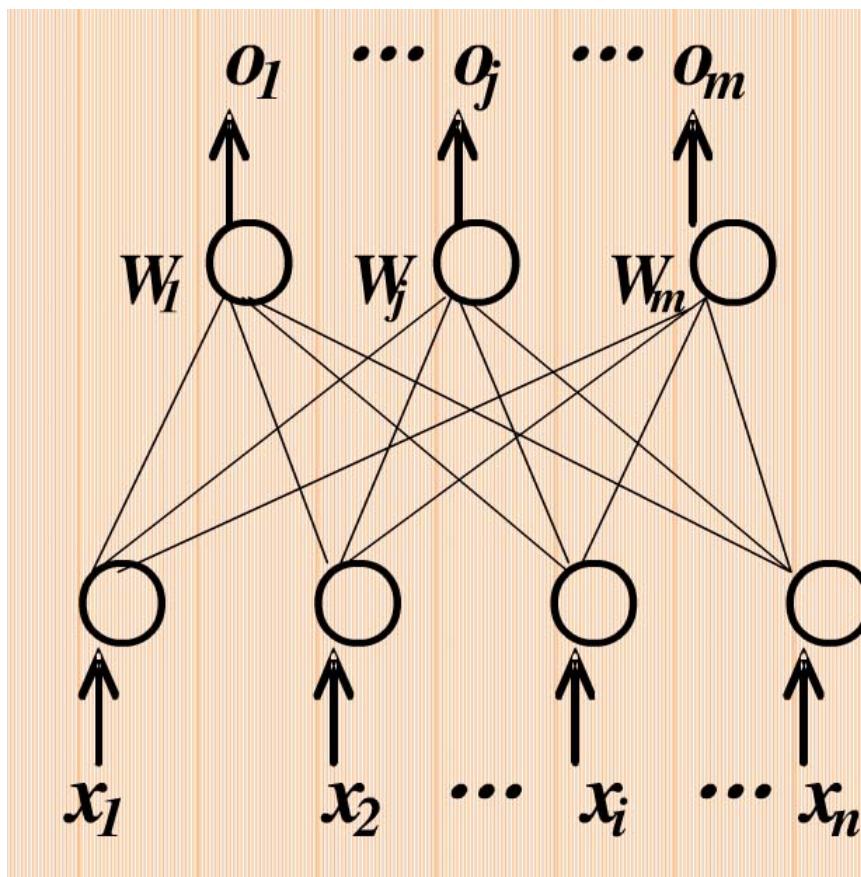
□ 是一种二元线性分类器, 使用特征向量作为输入, 把矩阵上的输入 x (实数值向量) 映射到输出值 $f(x)$ 上 (一个二元的值)

感知器结构

Perceptron Learning
(Rosenblatt, 1958)



感知器结构



□ 非线性前馈网络

□ 同层内无互连

□ 不同层间无反馈

□ 由下层向上层传递

□ 输入输出均为离散值

□ 由阈值函数决定其输出

有监督的学习机制

□ c 是常数，表示学习率

□ d 是期望的输出，取值为1或-1

□ sign 是感知机的输出，取值为1或-1

$$\Delta W_i = c(d - \text{sign}(\sum w_i * x_i)) X_i$$

□ 期望输出和实际输出相同，不改变权值

□ 实际输出为-1，期望输出为+1，则增加 $2cX_i$

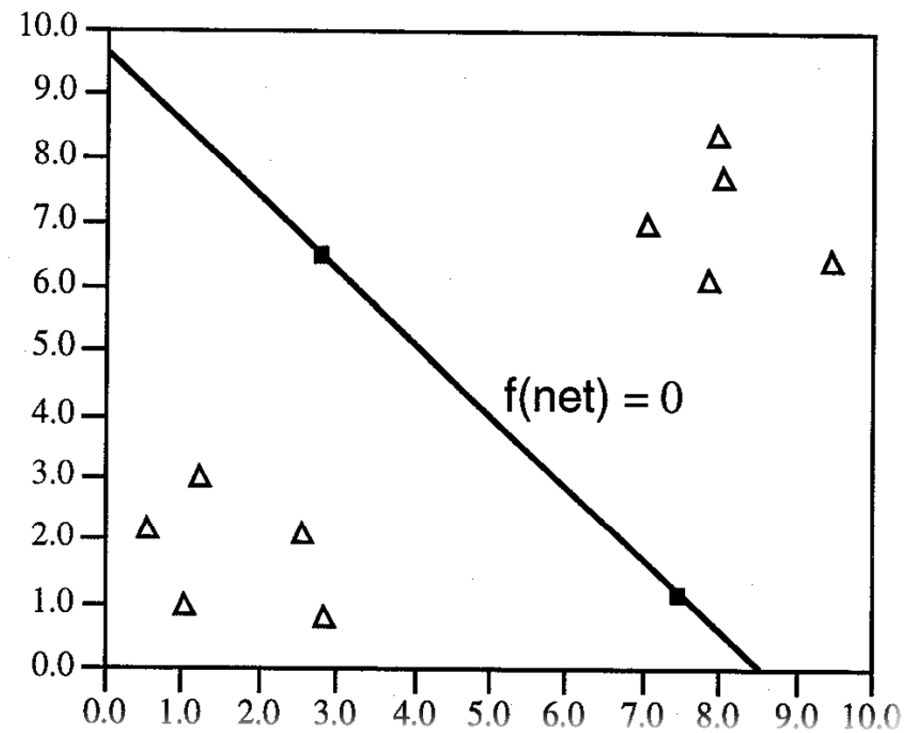
□ 实际输出为+1，期望输出为-1，则减少 $2cX_i$

感知机的学习算法

1. 权值初始化
2. 输入样本对
3. 计算输出
4. 根据感知机学习规则调整权重
5. 返回到步骤2输入到下一对样本，直至对所有样本的实际输出与期望输出相等

例子

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1



x_1	x_2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

$$f(\text{net}) = f(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)$$

x_0 是偏置单元，通常取1

假设初始权值为 $[-0.6, 0.75, 0.5]$

对第一行数据来说

$$f_1 = f(-0.6 * 1 + 0.75 * 1 + 0.5 * 1) = f(0.65) = 1$$

与期望值一样，所以W向量不变， $W_1 = W_0$

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

$$f(\text{net}) = f(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)$$

x_0 是偏置单元，通常取1

假设初始权值为 $[-0.6, 0.75, 0.5]$

对第一行数据来说

$$f_1 = f(-0.6 * 1 + 0.75 * 1 + 0.5 * 1) = f(0.65) = 1$$

与期望值一样，所以W向量不变， $W_1 = W_0$



$$f_2 = f(-0.6 * 1 + 0.75 * 9.4 + 0.5 * 6.4) = f(9.65) = 1$$

期望为-1，所以 $W_2 = W_1 + 0.2 * (-2) * X_2$

$$W_2 = [-0.6, 0.75, 0.5] - 0.4 * [1, 9.4, 6.4] = [-1.00, -3.01, -2.06]$$

x_1	x_2	Output
1.0	1.0	1
9.4	6.4	-1
2.5	2.1	1
8.0	7.7	-1
0.5	2.2	1
7.9	8.4	-1
7.0	7.0	-1
2.8	0.8	1
1.2	3.0	1
7.8	6.1	-1

$$f(\text{net}) = f(w_0 * x_0 + w_1 * x_1 + w_2 * x_2)$$

x_0 是偏置单元，通常取1

假设初始权值为 $[-0.6, 0.75, 0.5]$

对第一行数据来说

$$f_1 = f(-0.6 * 1 + 0.75 * 1 + 0.5 * 1) = f(0.65) = 1$$

与期望值一样，所以W向量不变， $W_1 = W_0$



$$f_2 = f(-0.6 * 1 + 0.75 * 9.4 + 0.5 * 6.4) = f(9.65) = 1$$

期望为-1，所以 $W_2 = W_1 + 0.2 * (-2) * X_2$

$$W_2 = [-0.6, 0.75, 0.5] - 0.4 * [1, 9.4, 6.4] = [-1.00, -3.01, -2.06]$$



$$f_3 = f(-1 * 1 - 3.01 * 2.5 - 2.06 * 2.1) = f(-12.84) = -1$$

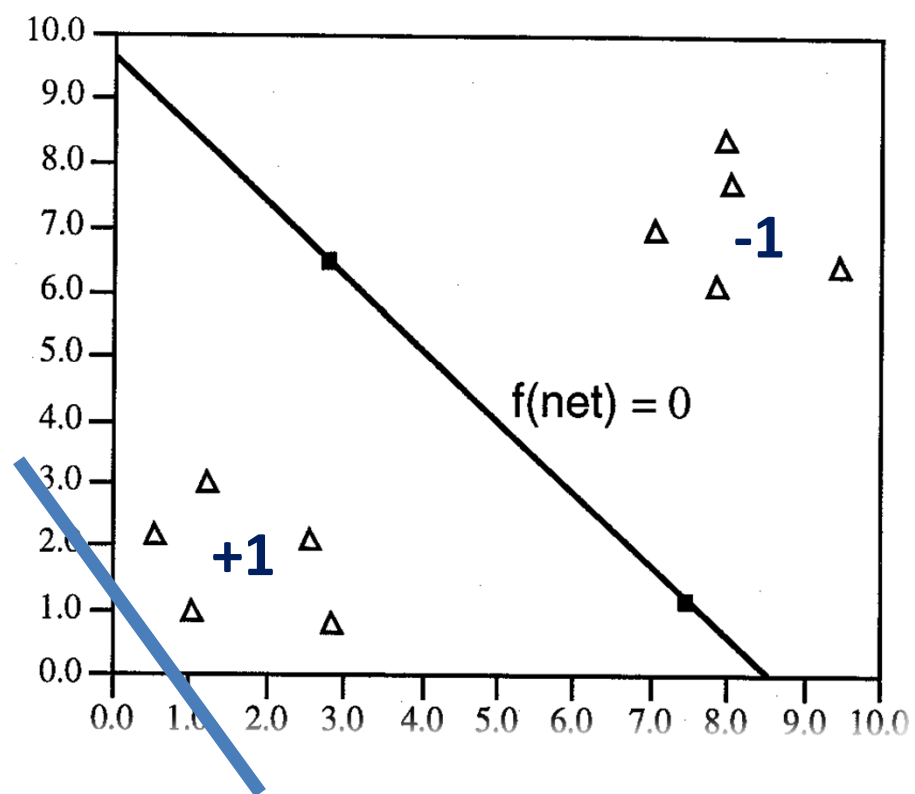
$$W_3 = W_2 + 0.2 * 2 * X_3 = [-0.60, -2.01, -1.22]$$

.....

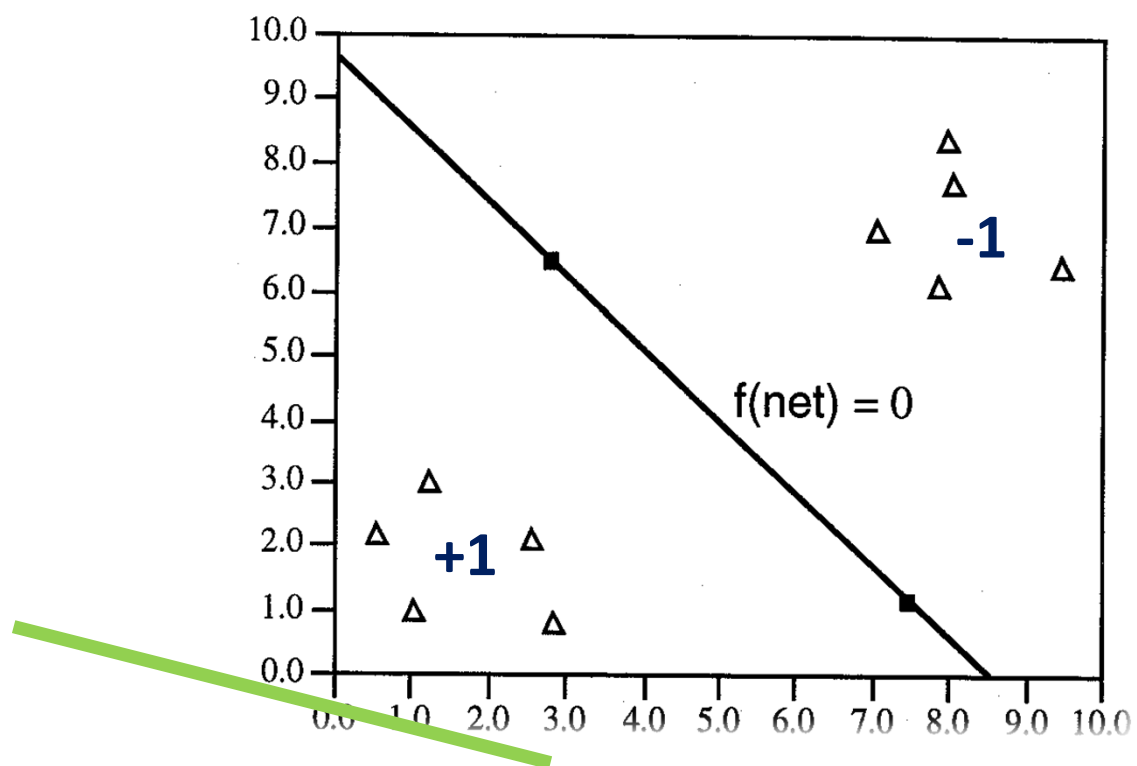
最终结果为 $W = [10.9, -1.3, -1.1]$

X_0 固定为1
a固定为0.2

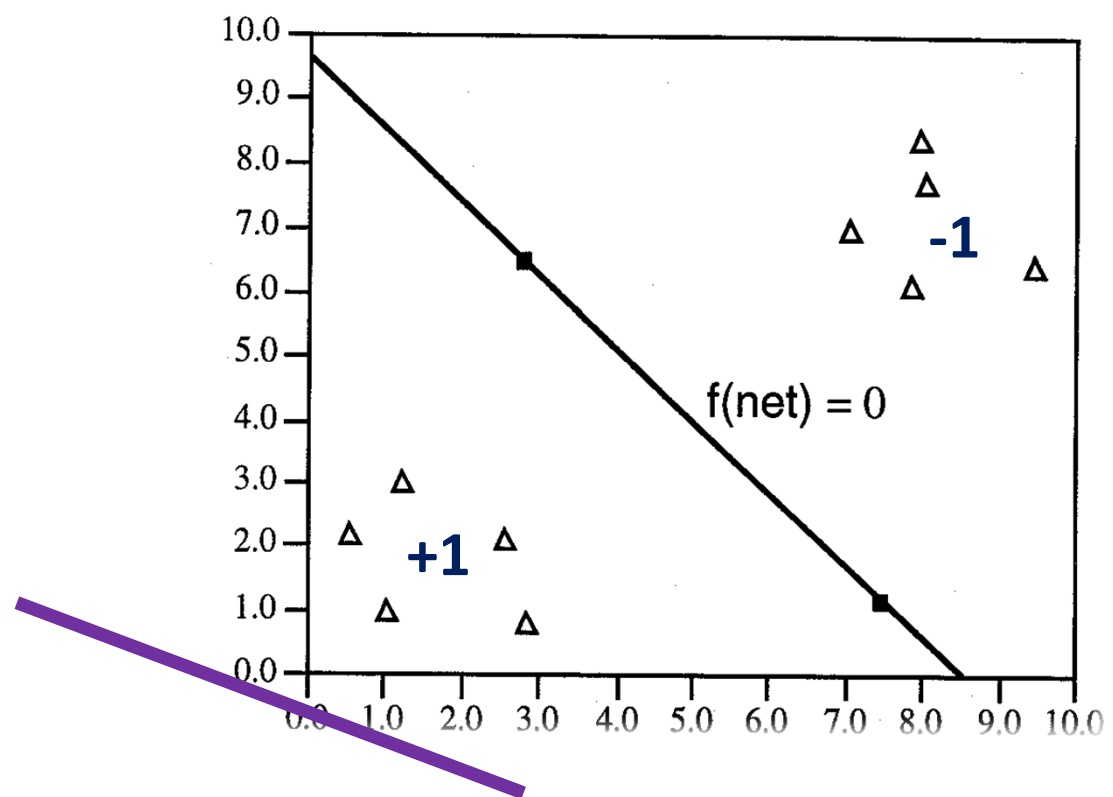
图示



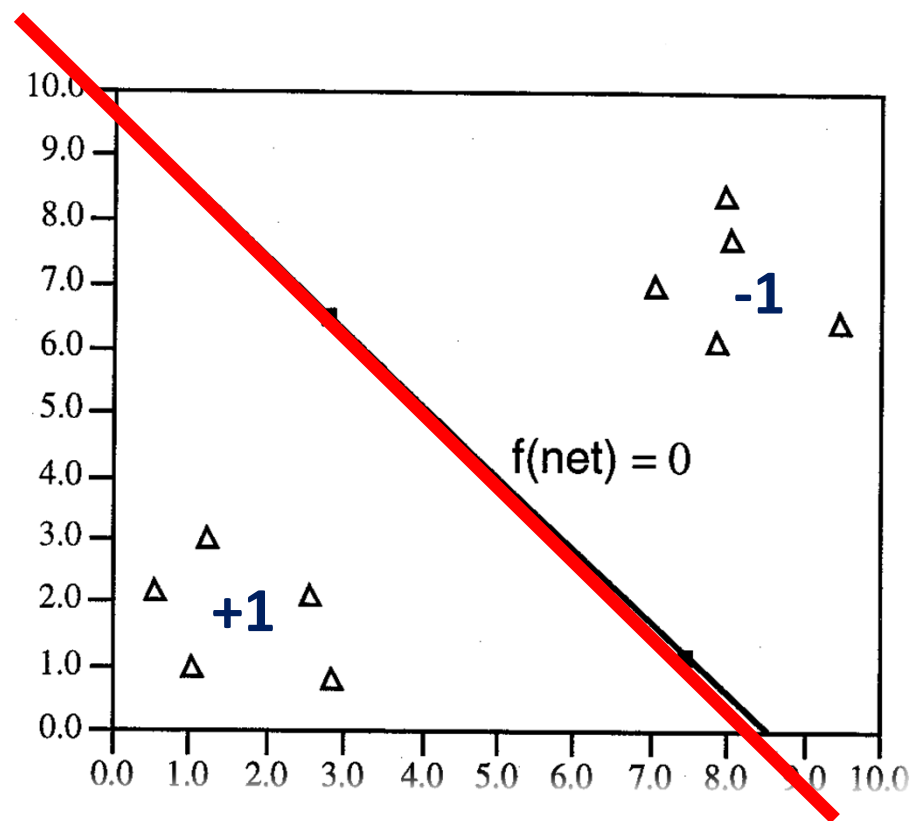
图示



图示



图示



感知机学习缺点

- 感知机模型属于单层神经网络，它不能解决一类非线性可分的问题。
- 典型的例子就是异或

表 10-2 异或真值表

x_1	x_2	输出
1	1	0
1	0	1
0	1	1
0	0	0

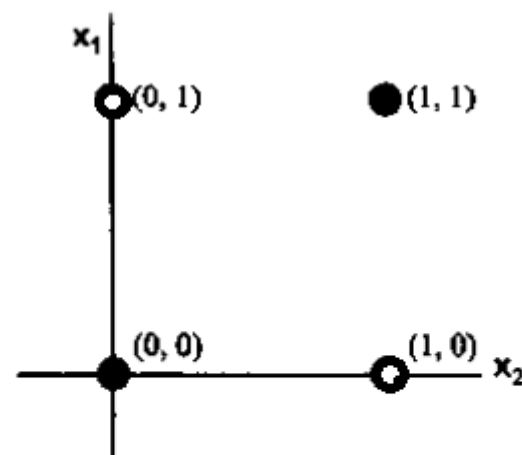


图 10-3 异或问题。在二维空间中
没有可分离点集 $\{(0,0), (1,1)\}$
和 $\{(0,1), (1,0)\}$ 的直线

感知机的表达能力



□ 感知机是 n 维实例空间的超平面决策

□ 候选假设空间 H : 所有可能实数权向量的集合

$$H = \{\vec{W} | \vec{W} \in R^{(n+1)}\}$$

□ 广义布尔函数 m -of- n : n 个输入值至少有 m 个为真，则输出为真

□ 二层神经网络可以表达所有的布尔函数

□ 与、或、与非、或非、异或.....

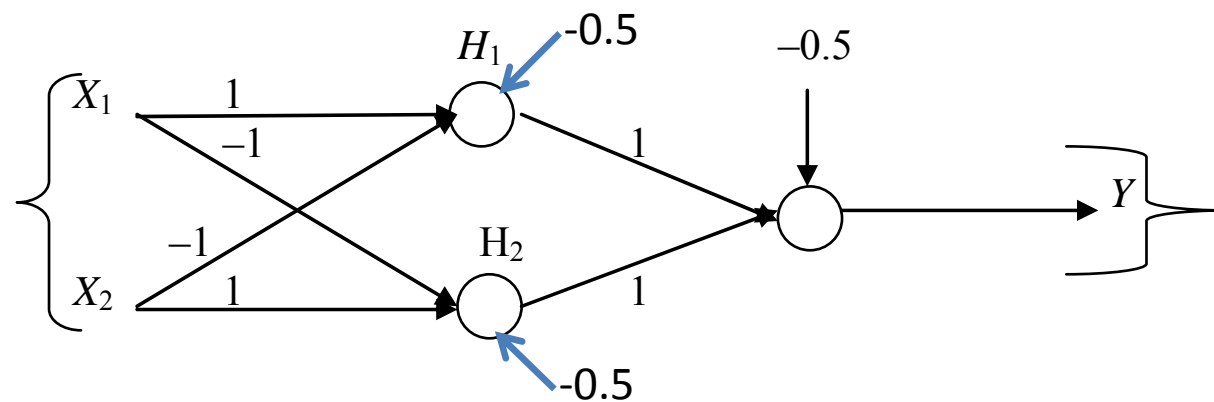
异或问题

输入节点

隐节点

输出节点

输出



□ 权重向量(1,-1)、(-1,1)

输入		隐节点输出		输出节点	$X_1 \text{ XOR } X_2$
X_1	X_2	H_1	H_2		
0	0	0	0	$-0.5 \rightarrow 0$	0
0	1	$-1 \rightarrow 0$	1	$0.5 \rightarrow 1$	1
1	0	1	$-1 \rightarrow 0$	$0.5 \rightarrow 1$	1
1	1	0	0	$-0.5 \rightarrow 0$	0

感知机学习的不足

□ 感知机学习一定可以收敛吗？

- ✓ 前提是训练样例必须是线性可分的！！



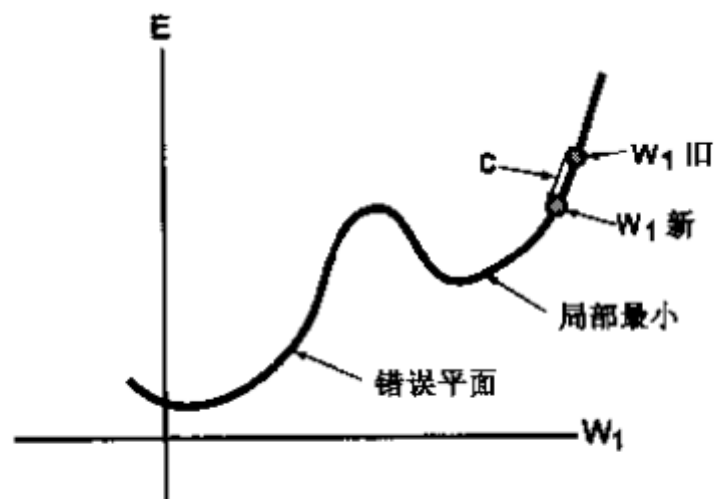
□ 如果训练样例不是线性可分的，怎么办？

- ✓ 只能去找一个学习方法，去收敛到目标概念的最佳近似

感知机学习方法只适用在单层网络！

Delta规则

- Delta规则是基于错误(误差)平面的，错误(误差)平面是神经网络所表示的函数在数据集上的累积误差。每一个神经网络权值向量都对应误差平面中的一个点。
- 应用delta规则时，激励函数必须是连续的和可微分的。



二维坐标中的错误平面。常数 c 指示了学习步幅的大小

Delta规则

$$Error = \frac{1}{2} \sum_i (d_i - O_i)^2 \quad \longrightarrow \quad \vec{o}(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$\Delta W_k = -c \frac{\partial Error}{\partial W_k} = -c \frac{\partial Error}{\partial O_i} * \frac{\partial O_i}{\partial W_k}$$

$$\frac{\partial Error}{\partial O_i} = \frac{\partial (\frac{1}{2} \sum_i (d_i - O_i)^2)}{\partial O_i} = \frac{\partial \frac{1}{2} * (d_i - O_i)^2}{\partial O_i}$$

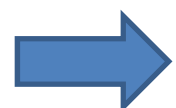
因为输出层中的节点的误差并不影响其他节点，因此

$$\frac{\partial \frac{1}{2} * (d_i - O_i)^2}{\partial O_i} = -(d_i - O_i)$$

与感知机学习方法一致！

Delta规则

$$\frac{\partial O_i}{\partial W_k} = X_k * f'(W_i X_i) = f'(\text{net}_i) * X_k$$


$$\vec{o}(\vec{x}) = \vec{w} \cdot \vec{x}$$

$$\begin{aligned}\Delta W_k &= -c * [-(d_i - O_i) * f'(\text{net}_i) * X_k] \\ &= c(d_i - O_i) \boxed{f'(\text{net}_i)} * X_k\end{aligned}$$

思考与感知机学习规则的区别！

Delta规则分析

- 学习常数 c 对delta规则的性能有很重要的影响， c 决定了在一步学习过程中权值变化的快慢， c 越大，权值朝最优值移动的速度越快。然而， c 过大会越过最优值或在最优值附近震荡。
- 尽管delta规则本身不能克服单层神经网络的局限，但是它的一般形式是反传算法(BP)的核心，反传算法是多层神经网络中的学习算法。

Delta规则分析

□ 梯度下降 gradient descent

- ✓ 搜索无限假设空间的有效策略
- ✓ 无限假设空间：连续的参数/可微
- ✓ 缺点：收敛速度慢/局部极小

□ 随机梯度下降 stochastic gradient descent

- ✓ 不需要计算总误差，快/可以有效避免局部极小

$$\nabla E(\vec{w}) \quad \longrightarrow \quad \nabla_d E(\vec{w})$$

大纲

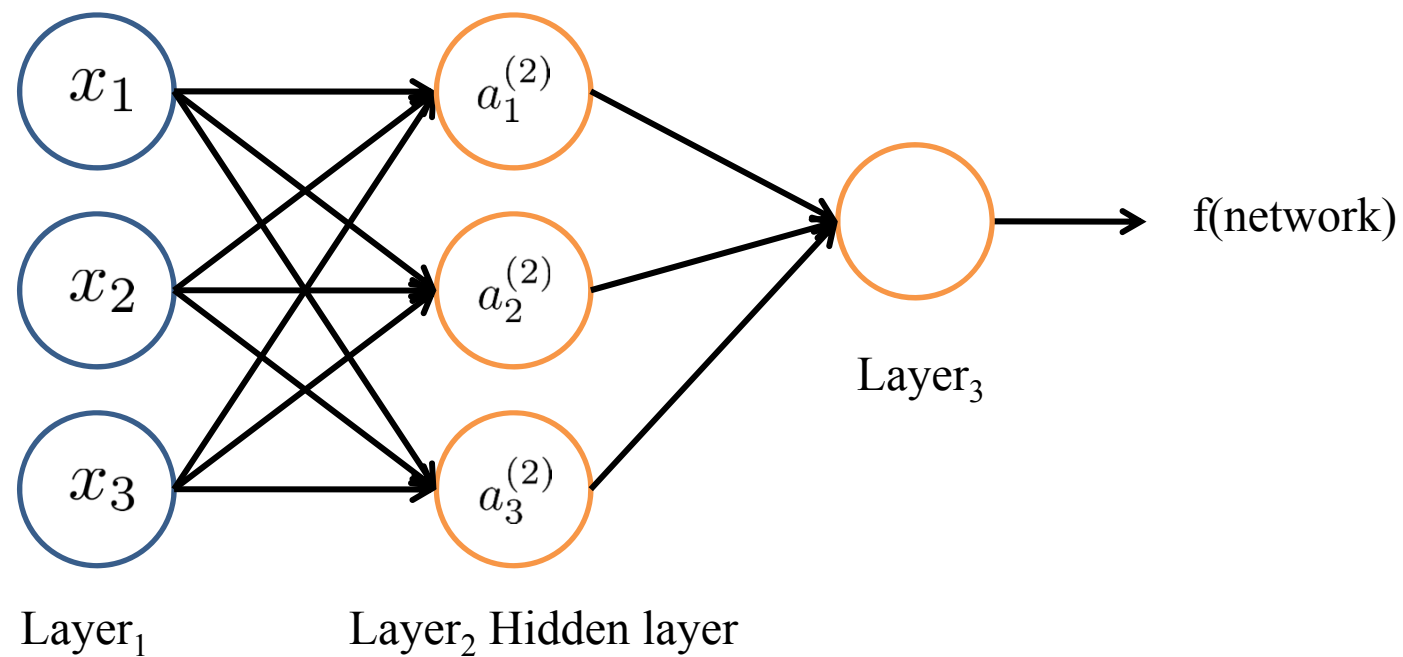
人工神经网络及其发展

感知器学习

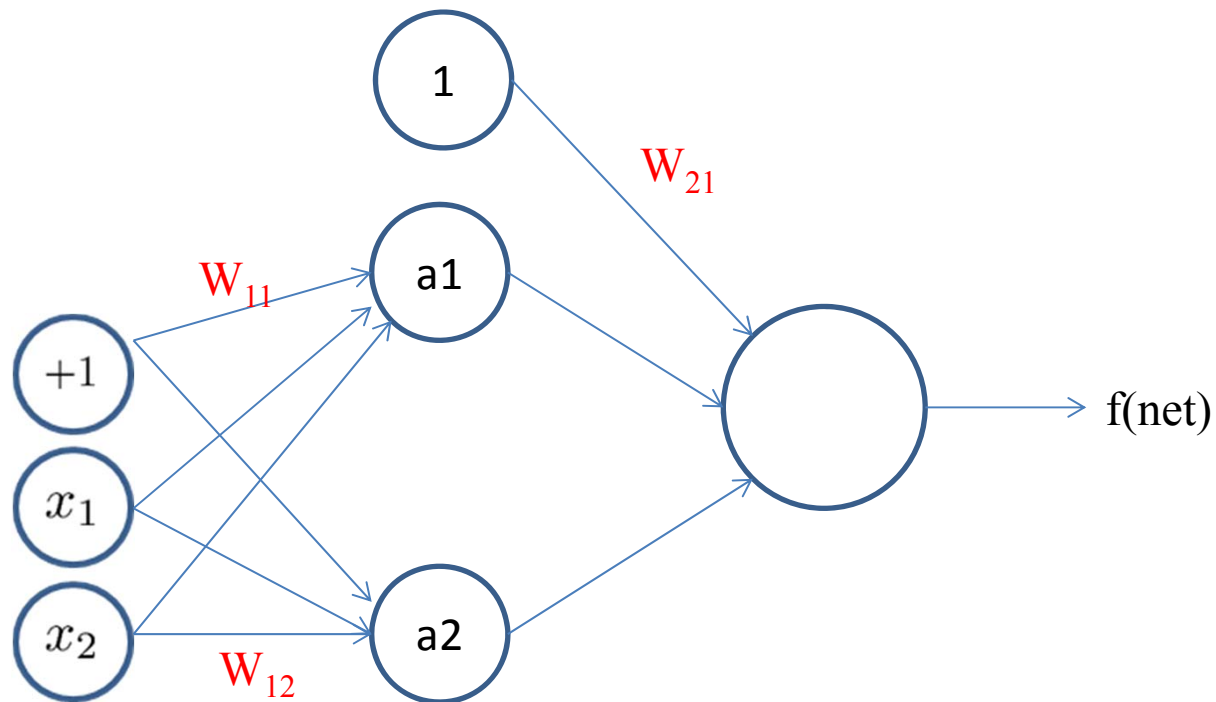
多层神经网络

反向传播算法BP

多层神经网络



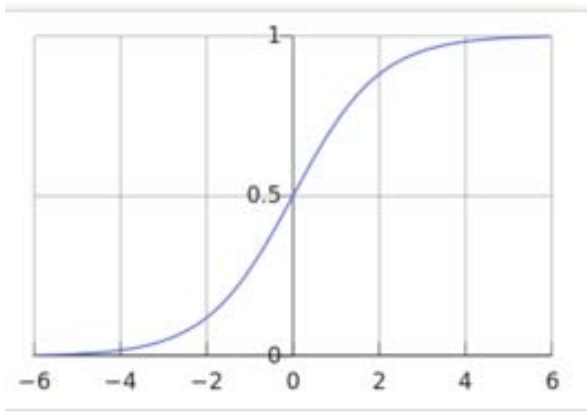
多层神经网络



$$W_{11} = [-30, 20, 20]$$

$$W_{12} = [10, -20, -20]$$

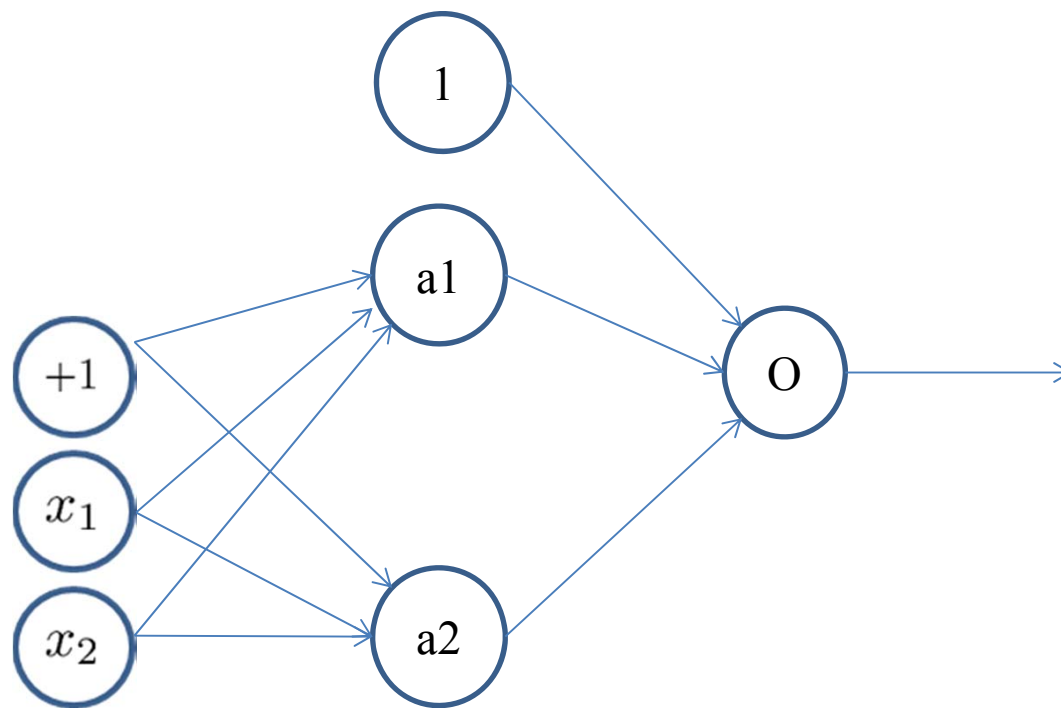
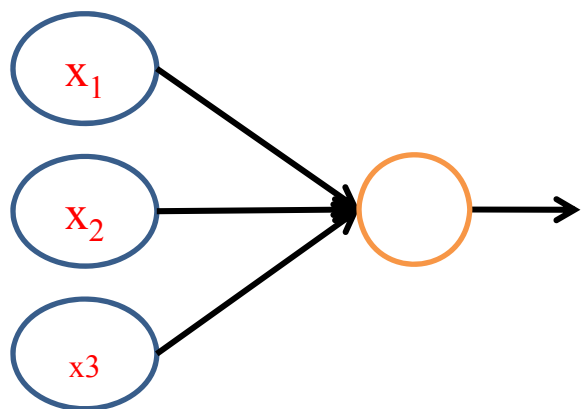
$$W_{21} = [10, -20, -20]$$



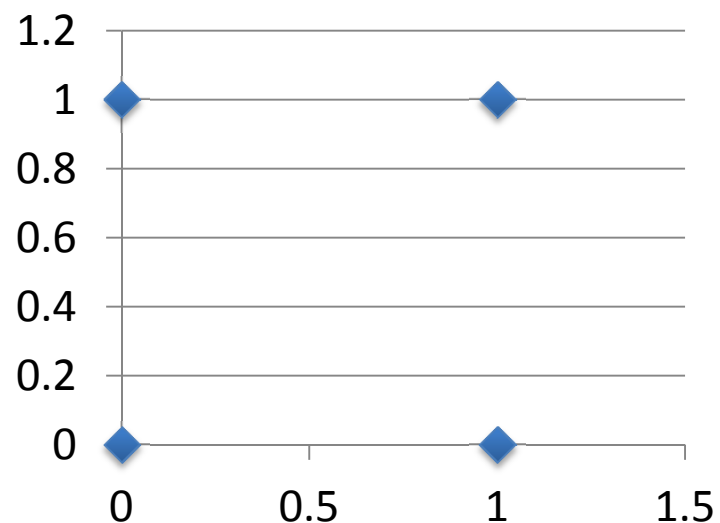
x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$f(\text{net})$
0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

多层神经网络的学习

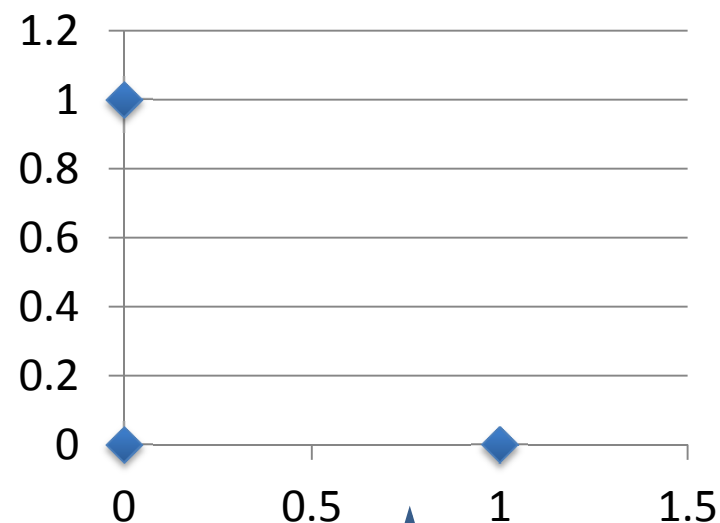
为什么要使用隐层神经元



为什么？？？



单层神经网络的特征空间



多层神经网络隐层处理后的特征空间

隐藏层神经元实际为特征检测算子 (feature detector)，在多层神经网络的学习过程中，隐藏层神经元开始逐步“发现”刻画训练数据的突出特征。

0	0	0	1	0
0	1	0	0	1
1	0	0	0	1
1	1	1	0	0

大纲

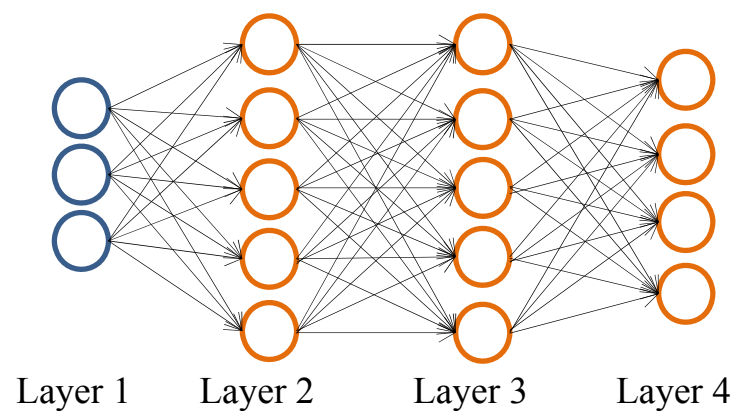
人工神经网络及其发展

感知器学习

多层神经网络

反向传播算法BP

误差传播



误差可通过连续的网络层，以复杂的不可预测的方式传播和变化

反向传播算法Back Propagation

- 前向阶段：网络突触的权值固定，输入信号在网络中正向一层一层传播，直到到达输出端，获得网络的输出。
- 反向阶段：通过比较网络的输出与期望输出，产生一个误差信号。误差信号通过网络反向一层一层传播，在传播过程中对网络突触的权值进行修正。

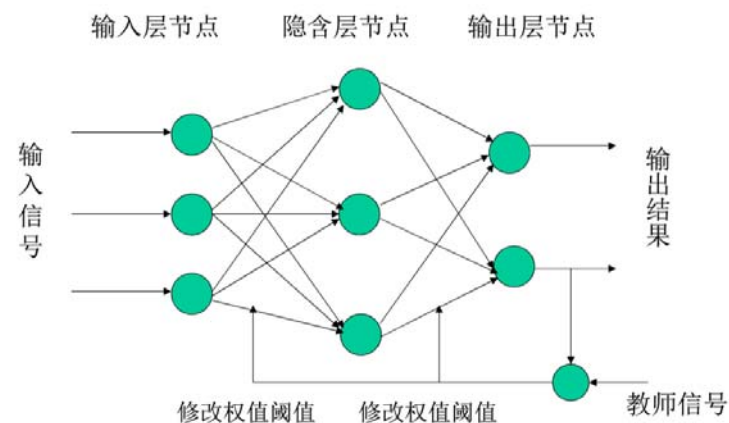
□ 信用分配 Credit assignment

- ✓ 对于输出层的权值修正计算是直接的，因为输出层对于外部世界可见，可以提供一个期望响应来指导神经元的行为。
- ✓ 在修正隐藏层的权值时，如何给隐藏层的神经元分配信用或者责任呢？

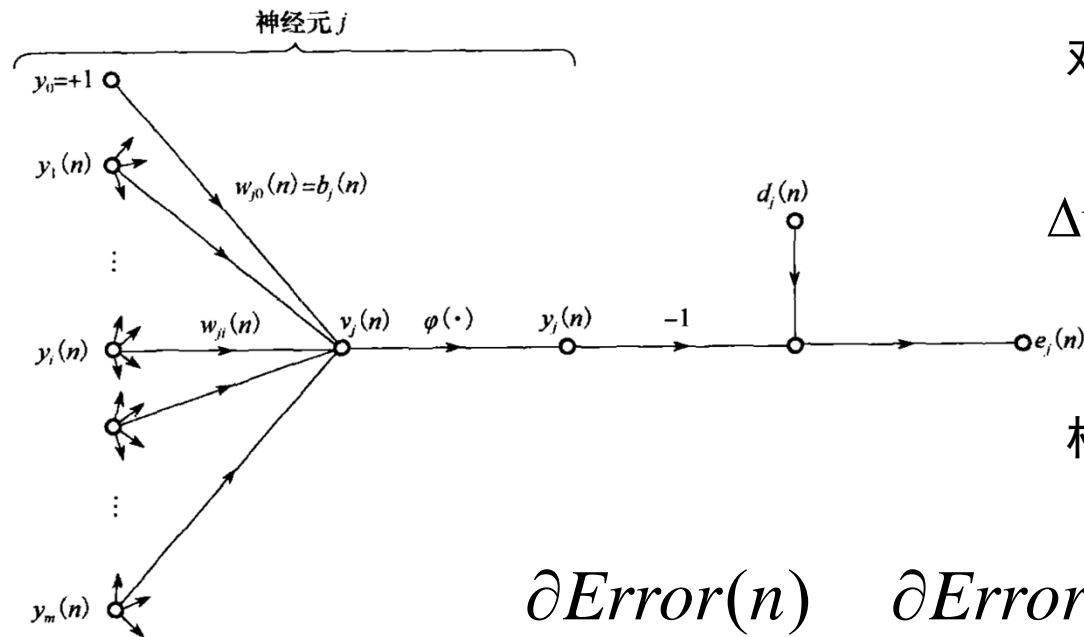
BP神经网络

□BP神经网络

- 三层或三层以上结构
- 无反馈
- 层内无互连
- 输入层+输出层+隐含层
- 采用误差反向传播学习算法



突触权值修正



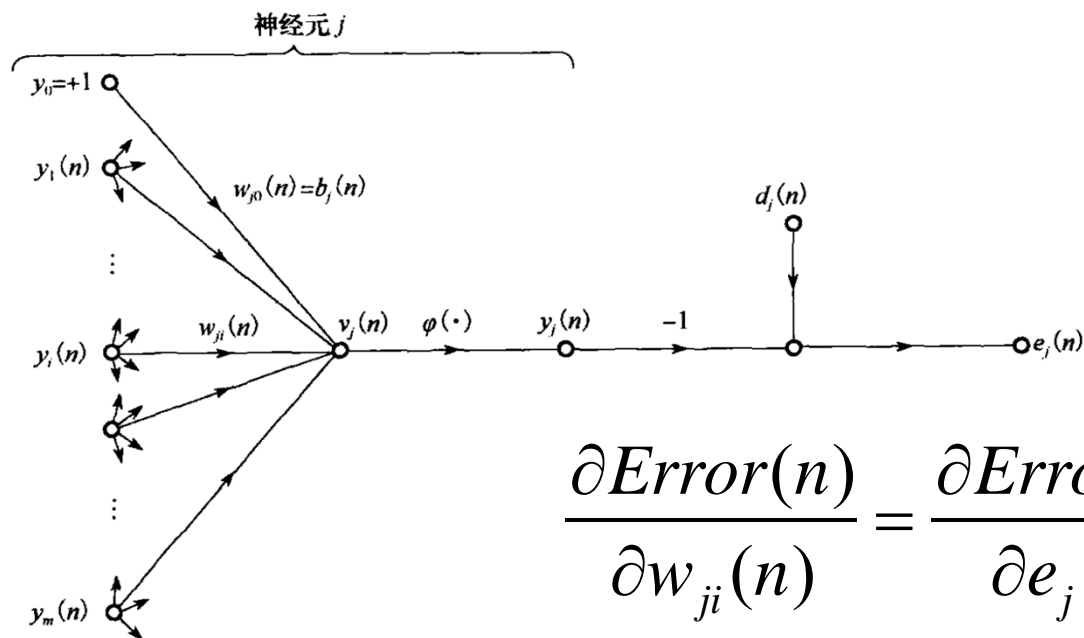
对突触权值 $w_{ji}(n)$ 应用修正值 $\Delta w_{ji}(n)$

$$\Delta w_{ji}(n) = -\alpha (\partial \text{Error}(n) / \partial w_{ji}(n))$$

根据链式规则

$$\frac{\partial \text{Error}(n)}{\partial w_{ji}(n)} = \frac{\partial \text{Error}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

偏导数代表一个敏感因子，决定了突触权值 w_{ji} 在权值空间的搜索方向。



此例权值下标ji

表示是由第i个节点连向第j个节点

$$\frac{\partial Error(n)}{\partial w_{ji}(n)} = \frac{\partial Error(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

对于误差的定义

$$Error(n) = \frac{1}{2} \sum_j (d_j(n) - y_j(n))^2 = \frac{1}{2} \sum_j e_j(n)^2$$

神经元j输出的函数信号

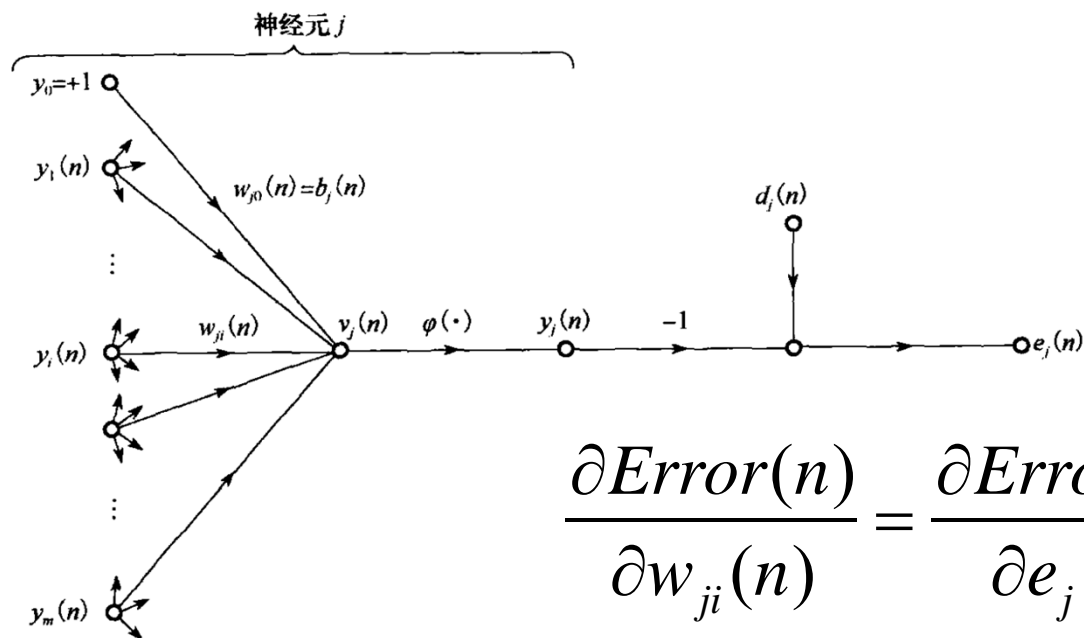
$$y_j(n) = \varphi_j(v_j(n))$$

对于误差的定义

$$e_j(n) = (d_j(n) - y_j(n))$$

诱导局部域

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n)$$



$$\frac{\partial Error(n)}{\partial w_{ji}(n)} = \frac{\partial Error(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

因此我们可以得到

$$\frac{\partial Error(n)}{\partial e_j(n)} = e_j(n)$$

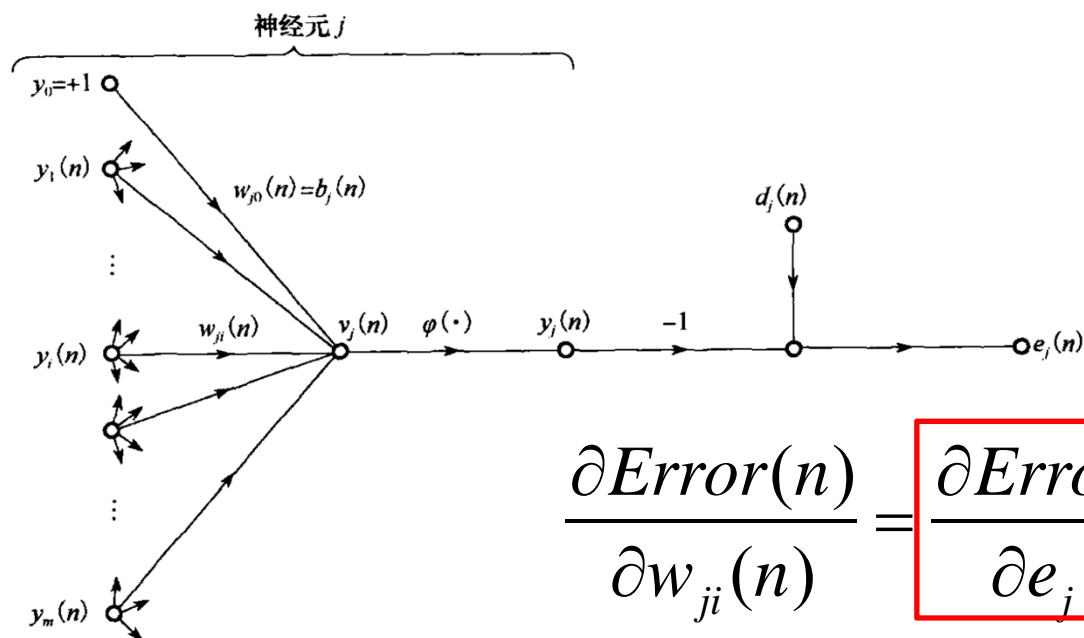
$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi_j'(v_j(n))$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$$

因此可求的偏导数为

$$\frac{\partial Error(n)}{\partial w_{ji}(n)} = -e_j(n) \varphi_j'(v_j(n)) y_i(n)$$



$$\frac{\partial Error(n)}{\partial w_{ji}(n)} = \frac{\partial Error(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

定义局域梯度 $\delta_j(n)$

$$\delta_j(n) = -\frac{\partial Error(n)}{\partial v_j(n)}$$

因此权值修正定义为

$$\Delta w_{ji}(n) = \alpha * \delta_j(n) * y_i(n)$$

权值修正的两种情况

□ Case1: 神经元j是输出层节点

$$\Delta w_{ji}(n) = \alpha * \text{delta}_j(n) * y_i(n)$$

求解局域梯度 $\text{delta}_j(n)$:

$$\text{delta}_j(n) = -\frac{\partial \text{Error}(n)}{\partial v_j(n)} = -\frac{\partial \text{Error}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \varphi_j'(v_j(n))$$

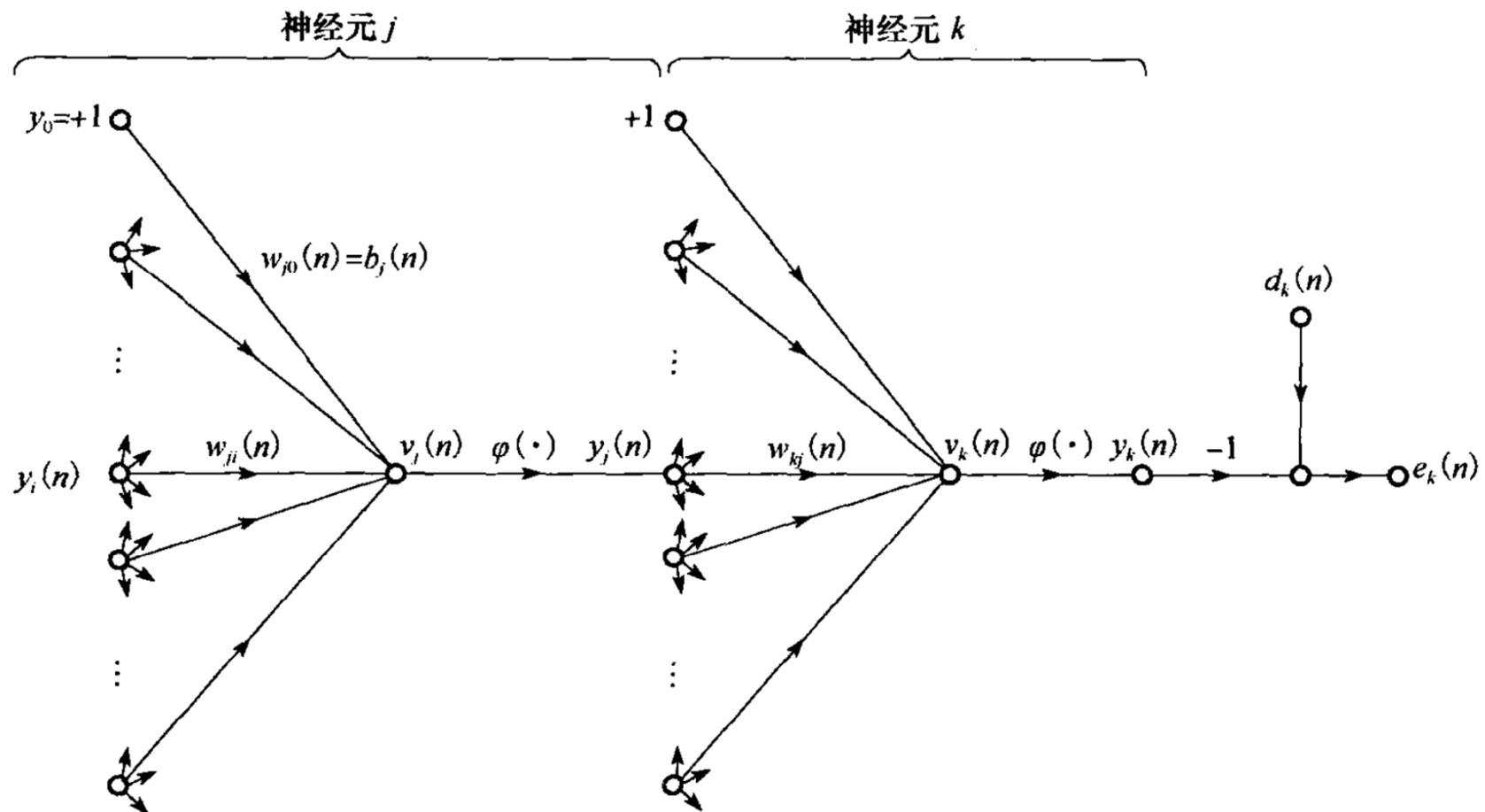
因此权值修正值为:

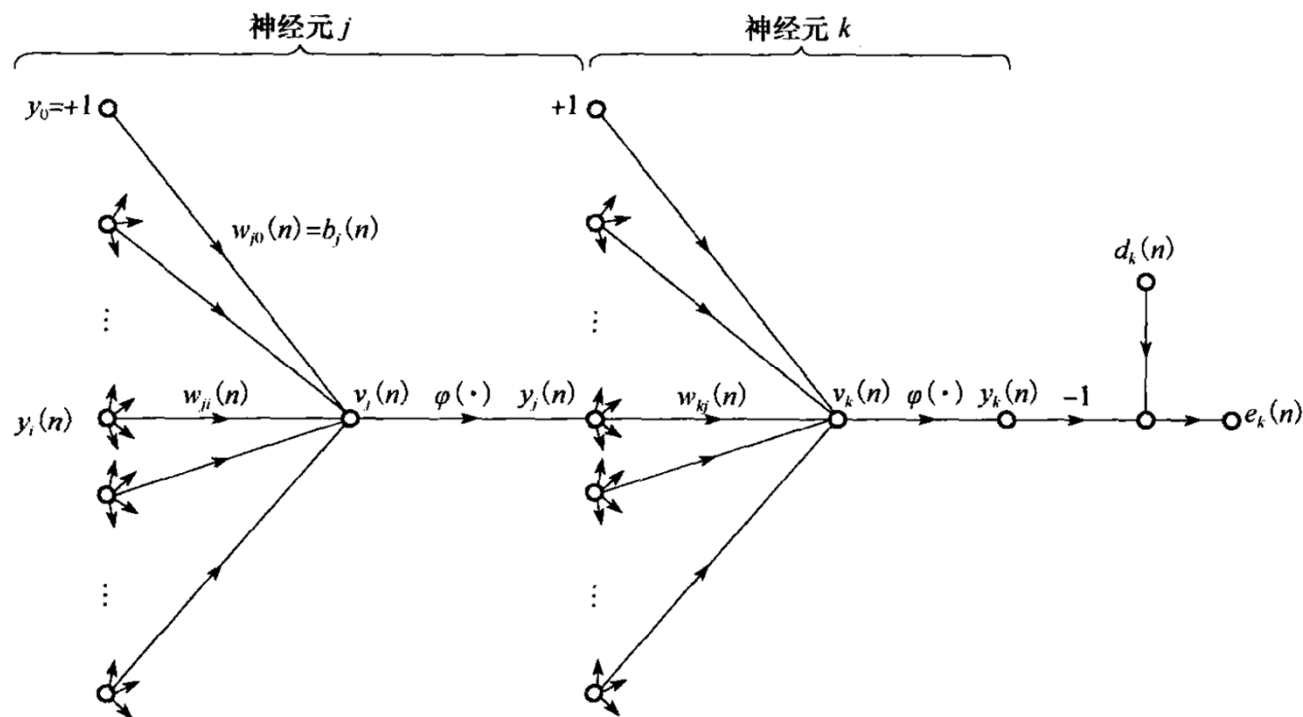
$$\Delta w_{ji}(n) = \alpha * e_j(n) * \varphi_j'(v_j(n)) * y_i(n)$$

神经元是隐藏层节点

□ Case2: 神经元j是隐藏层节点

当神经元 j 位于网络隐藏层时，就没有对于神经元的指定期望输出。



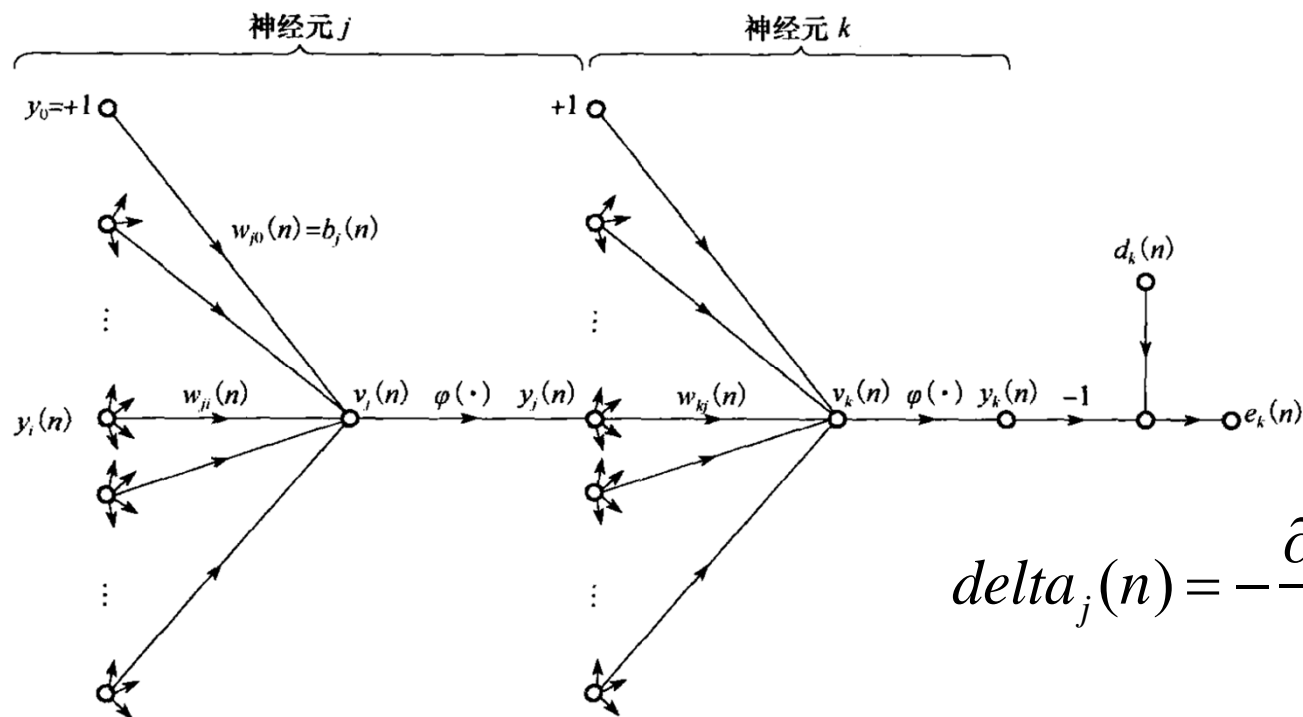


隐藏层神经元不能直接访问，但是它们必须分担对网络输出的误差责任。

如何分配这种共担的责任，就是信用分配问题。

重新求解局域梯度 $\delta_j(n)$

$$\delta_j(n) = -\frac{\partial \text{Error}(n)}{\partial v_j(n)} = -\frac{\partial \text{Error}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} = \boxed{-\frac{\partial \text{Error}(n)}{\partial y_j(n)}} \varphi_j'(v_j(n))$$



$$\delta_j(n) = -\frac{\partial \text{Error}(n)}{\partial y_j(n)} \phi_j'(v_j(n))$$

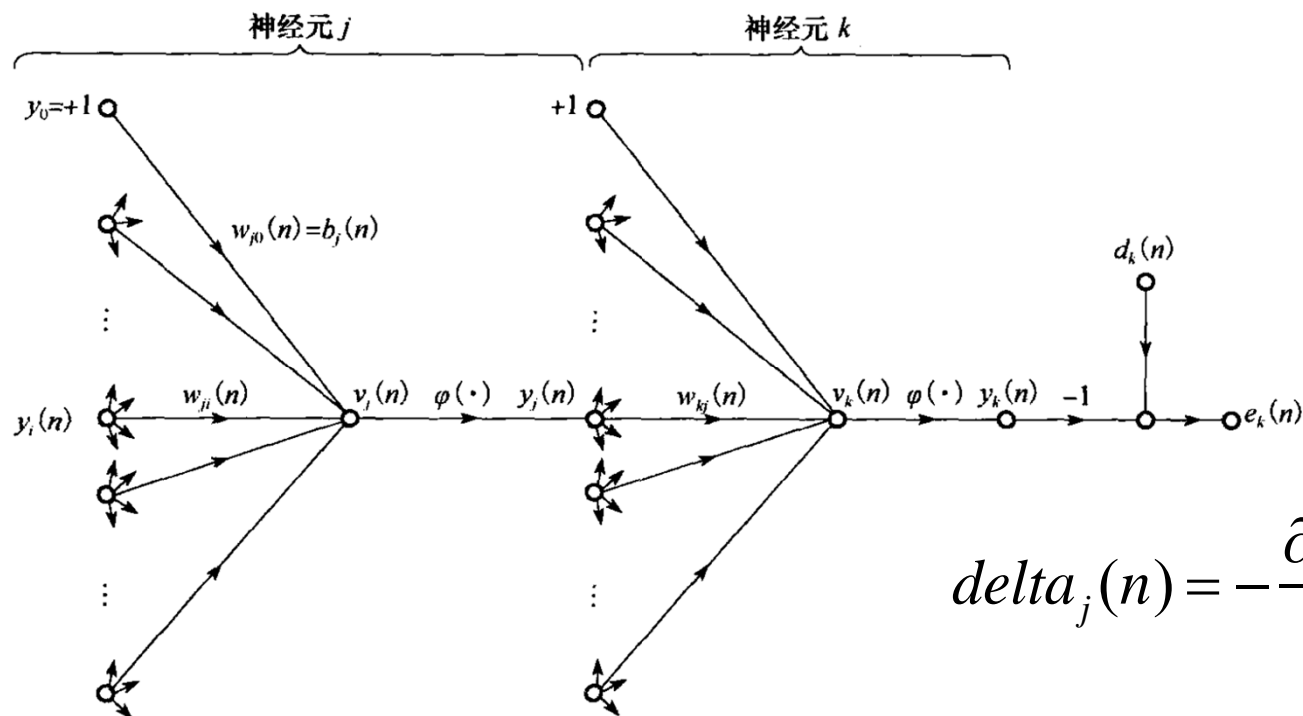
假设神经元k为输出层神经元

$$\text{Error}(n) = \frac{1}{2} \sum_k e_k^2(n)$$

计算隐藏层神经元j，对网络输出层神经元k的误差的责任。

则可求解

$$\frac{\partial \text{Error}(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}$$



$$\delta_j(n) = -\frac{\partial \text{Error}(n)}{\partial y_j(n)} \phi_j'(v_j(n))$$

又因为

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - \phi_k(v_k(n))$$

所以

$$\frac{\partial \text{Error}(n)}{\partial y_j(n)} = -\sum_k e_k(n) \phi_k'(v_k(n)) w_{kj}(n)$$

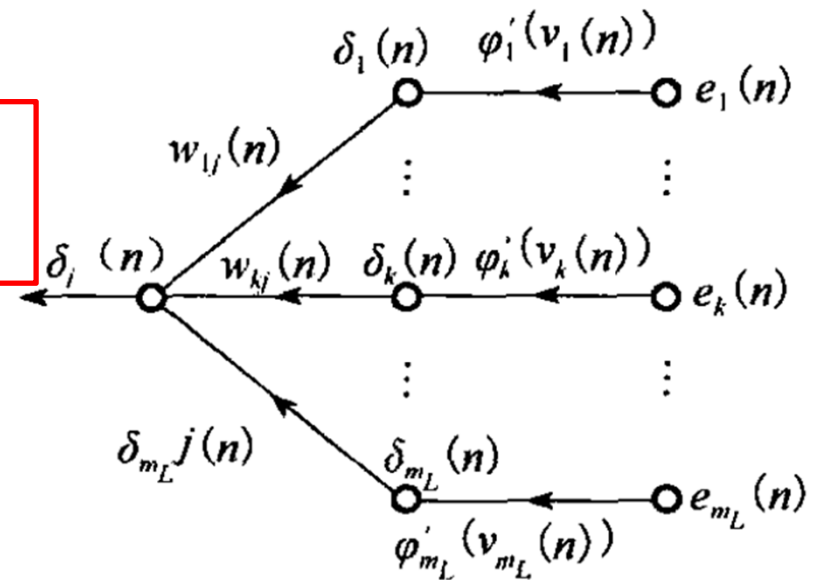
根据局域梯度的定义，可得

$$\frac{\partial \text{Error}(n)}{\partial y_j(n)} = -\sum_k \delta_k(n) w_{kj}(n)$$

因此

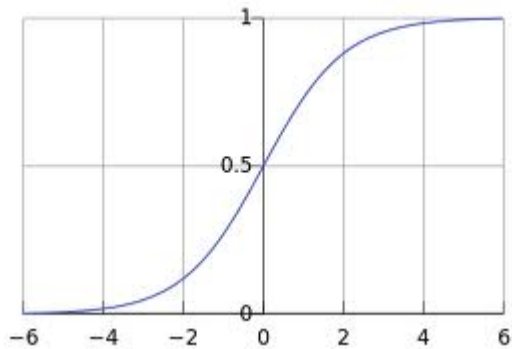
$$\delta a_j(n) = \varphi_j'(v_j(n)) \sum_k \delta a_k(n) w_{kj}(n)$$

$$\Delta w_{ji}(n) = \alpha * \delta a_j(n) * y_i(n)$$



- ❑ 反向传播的误差信号的转变——局域梯度delta:
- ❑ 第一项仅依赖于神经元激励函数
- ❑ 第二项为反向上一层的输入加权和，其中第一项需要误差e的知识，第二项体现了信用分配

sigmoid激励函数



λ 是挤压参数，值越大，区间 $[0,1]$ 上越接近直线。

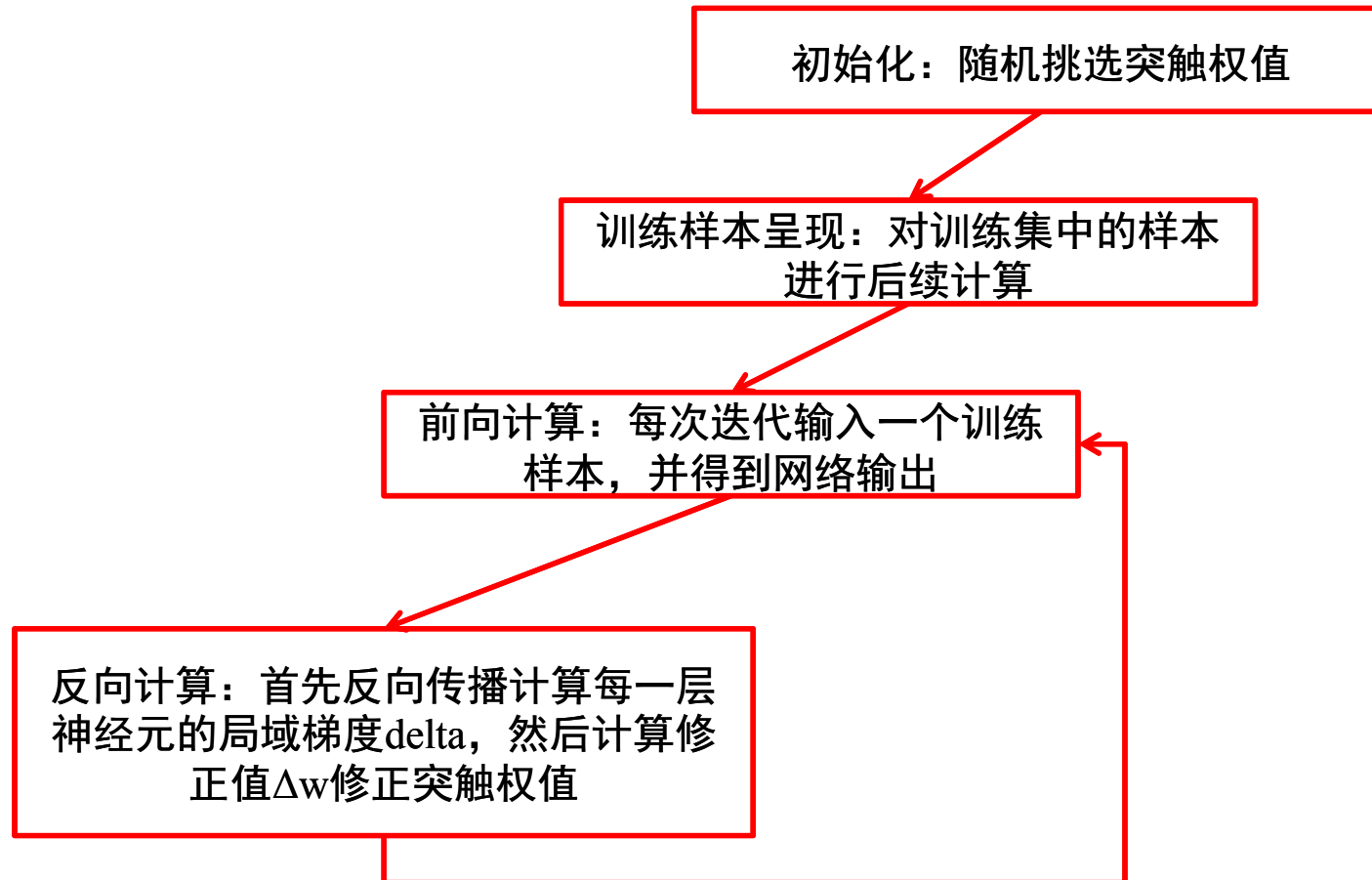
$$f(\text{net}) = \frac{1}{1 + e^{-\lambda * \text{net}}}$$

$$f'(\text{net}) = -\lambda e^{-\lambda} (1 + e^{-\lambda * \text{net}})^{-2}$$

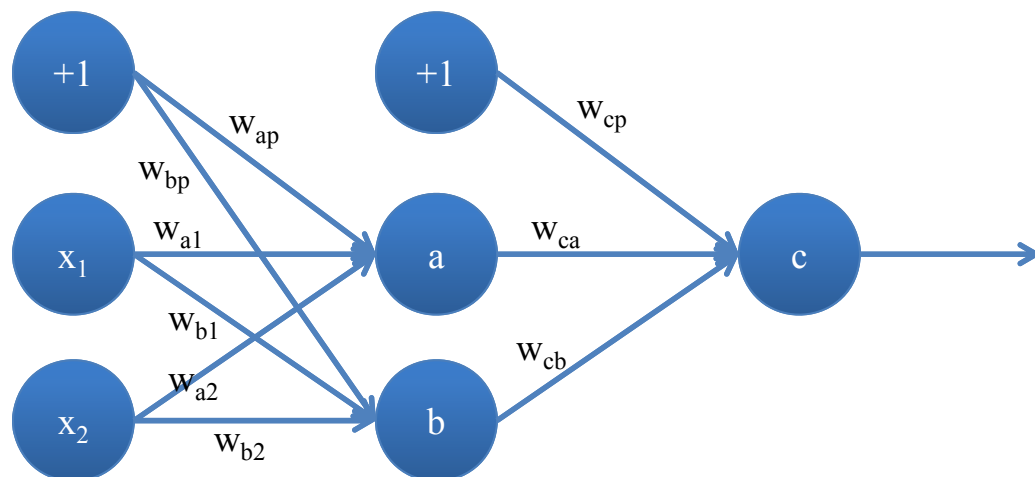
$$= -\lambda (1 + e^{-\lambda * \text{net}})^{-1} \left[1 - (1 + e^{-\lambda * \text{net}})^{-1} \right]$$

$$= -\lambda f(\text{net})(1 - f(\text{net}))$$

反向传播算法



反向传播算法实例：异或



- ❑ 初始化：将所有的权值 w 初始化为0，并选择sigmoid(logistic)函数为神经元的激励函数。
- ❑ 训练样本的呈现：训练样本为异或真值表— $(0,0) \rightarrow 0$; $(0,1) \rightarrow 1$; $(1,0) \rightarrow 1$; $(1,1) \rightarrow 0$, 并进行反复迭代。
- ❑ 后两步迭代过程：以第一个输入样例 $(0,0) \rightarrow 0$ 为例。

适合ANN的学习问题

1. 实例用“属性-值”对表示
2. 目标函数输出可以为离散值/实数值/向量
3. 对训练数据的错误鲁棒
4. 需要长时间的训练
5. 测试时间短
6. 可理解性差

思考和讨论

1. 感知机的表达能力？
2. 隐藏节点的作用？
3. 感知机学习和Delta学习的区别？
4. BP学习算法？

实验

题目：实现BP神经网络学习算法。在UCI数据集中分别选择2个数据集（Audiology (Standardized), Credit Approval），进行学习和分类。

评判预测性能的指标：准确率（precision），召回率(recall)、真阴性（true negative）、真阳性（true positive）（[请查阅资料了解这四个指标的概念](#)）。

谢谢！