

# 神经元网络

高 阳, 李文斌

<http://cs.nju.edu.cn/rl>, 2022.3.7

# 大纲

多层感知机 MLP

自动编码器 AUTOENCODER

径向基网络 RBF

# 大纲

多层感知机 MLP

回顾

反向传播

其他议题

自动编码器 AUTOENCODER

径向基网络 RBF

# 回顾

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

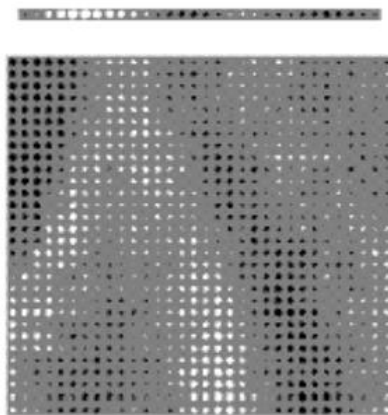
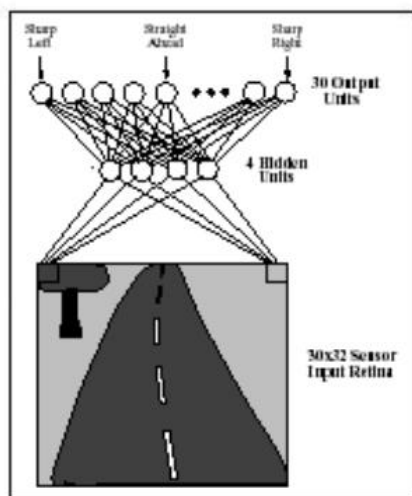


□ 1993年, CMU研发

□ 输入: 30\*32的像素

□ 4个隐藏节点

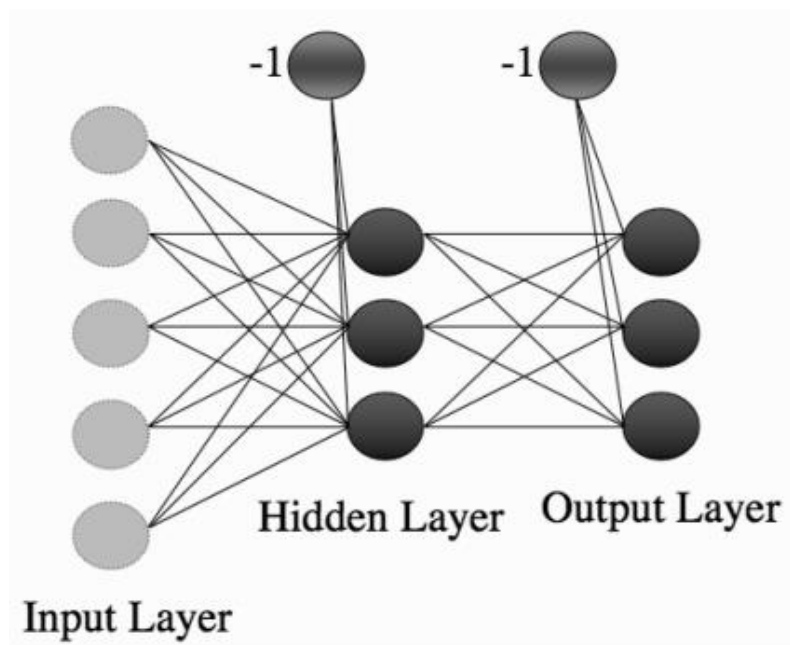
□ 输出: 30个驾驶动作(急剧左转、急剧右转, 正前方行进.....)



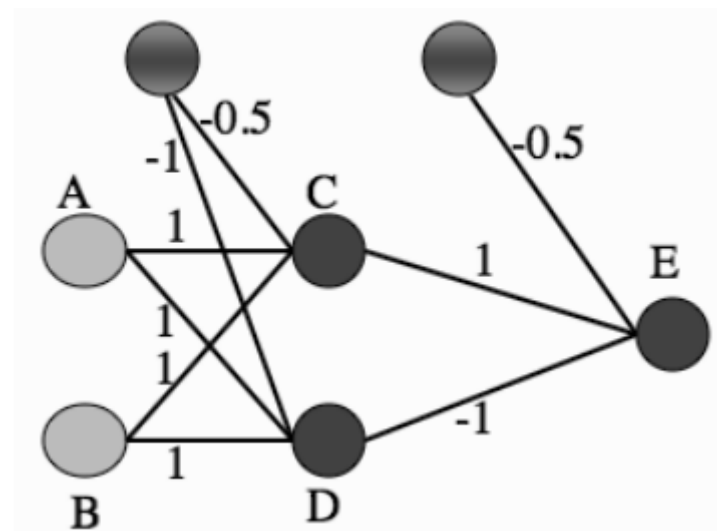
有向/无向? 有环/无环?

结构? 多层(浅/深)?

# 回顾

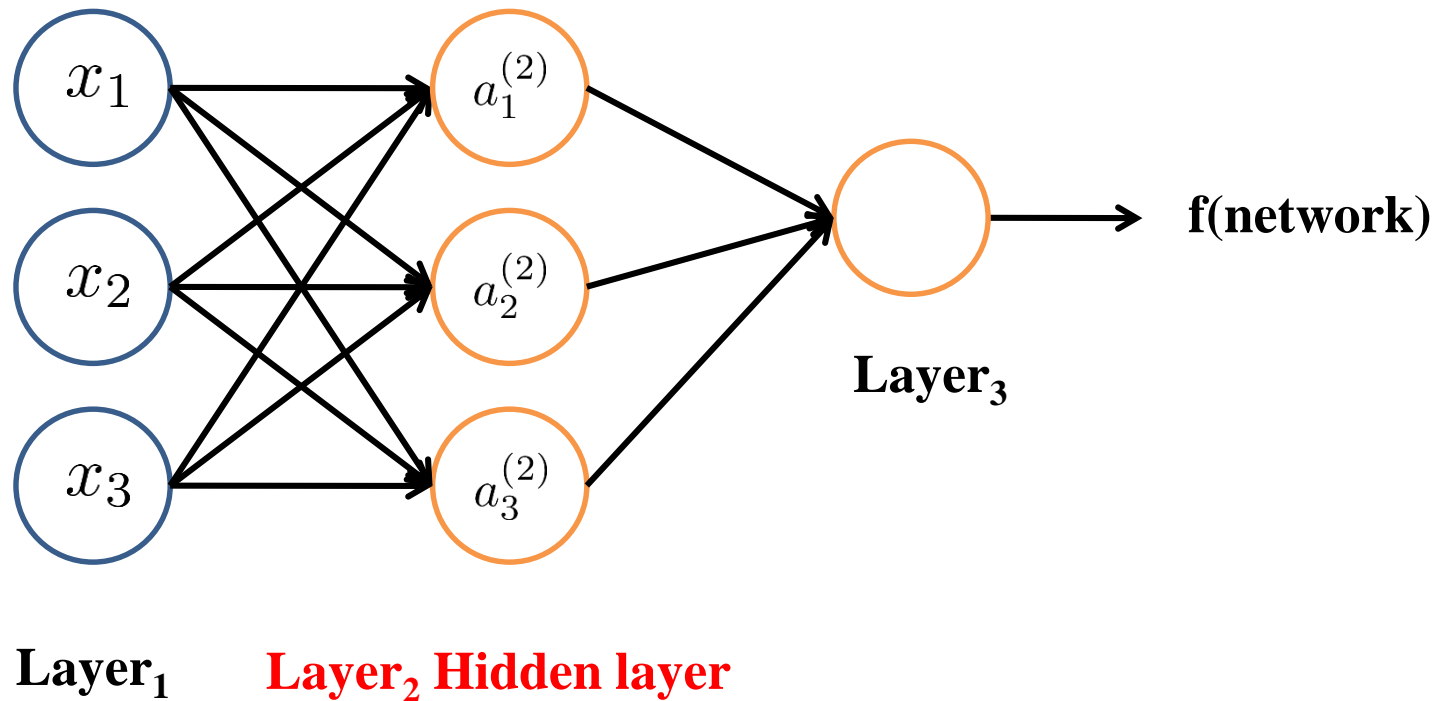


多层感知器网络

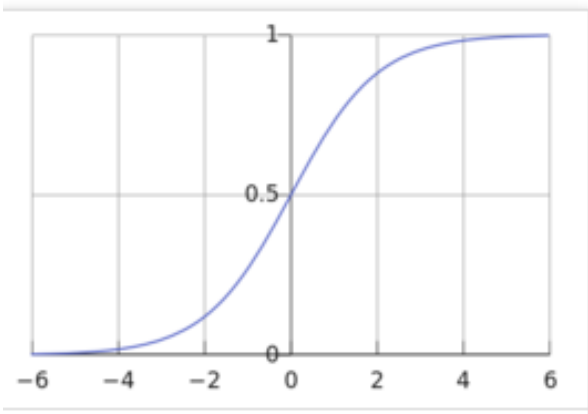
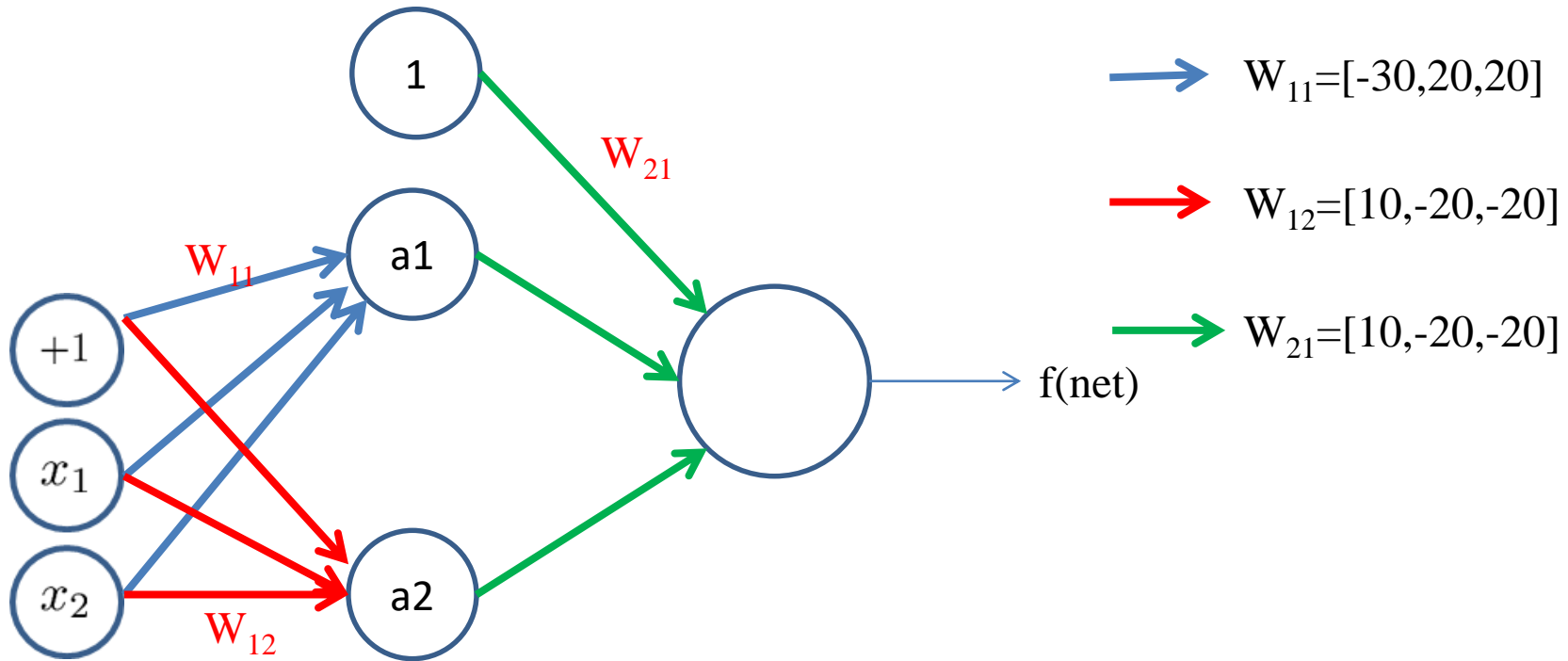


解决XOR的多层感知器网络权值

# 多层感知机



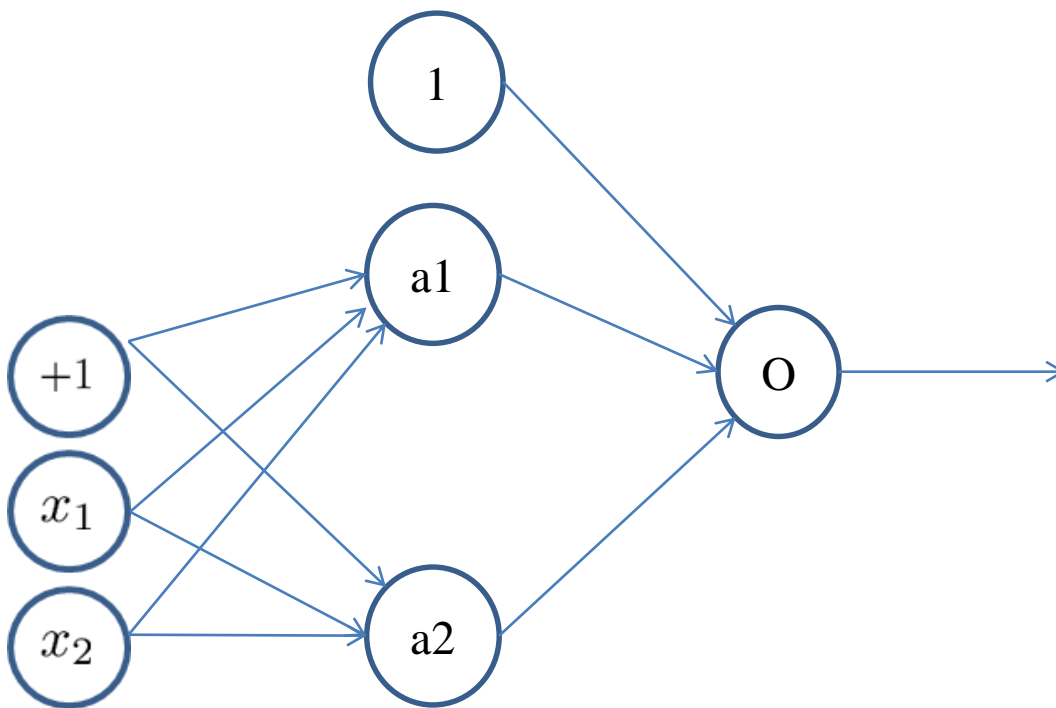
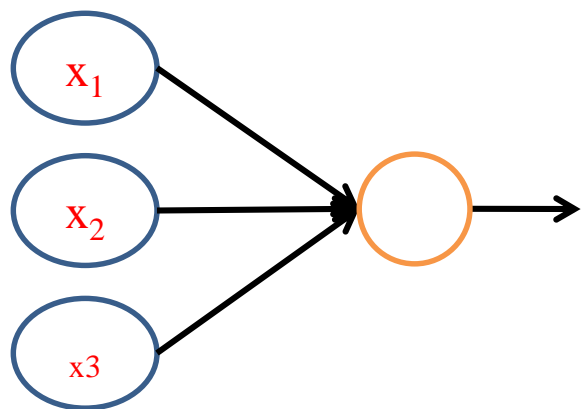
# 多层感知机



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

# 多层感知机的隐层

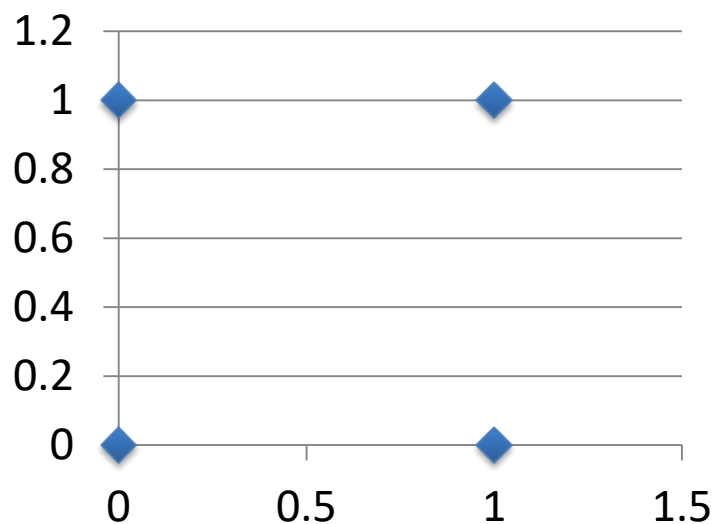
为什么要使用隐层神经元



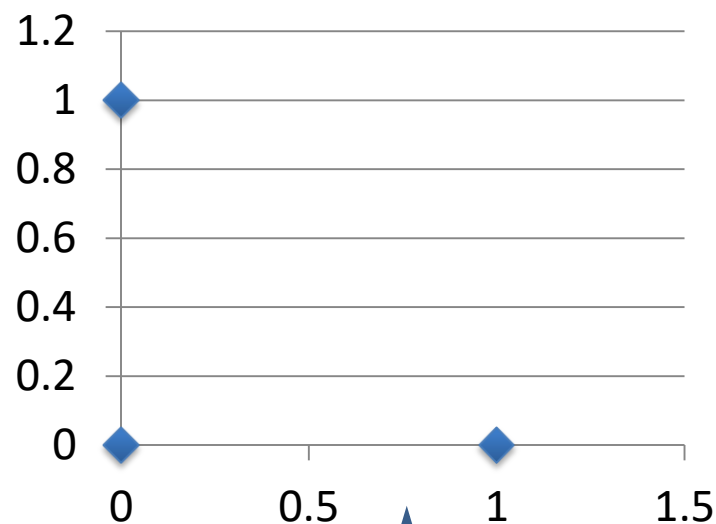
Q1:为什么需要隐藏层?

Q2:如何计算隐藏层的权值?





单层神经网络的特征空间

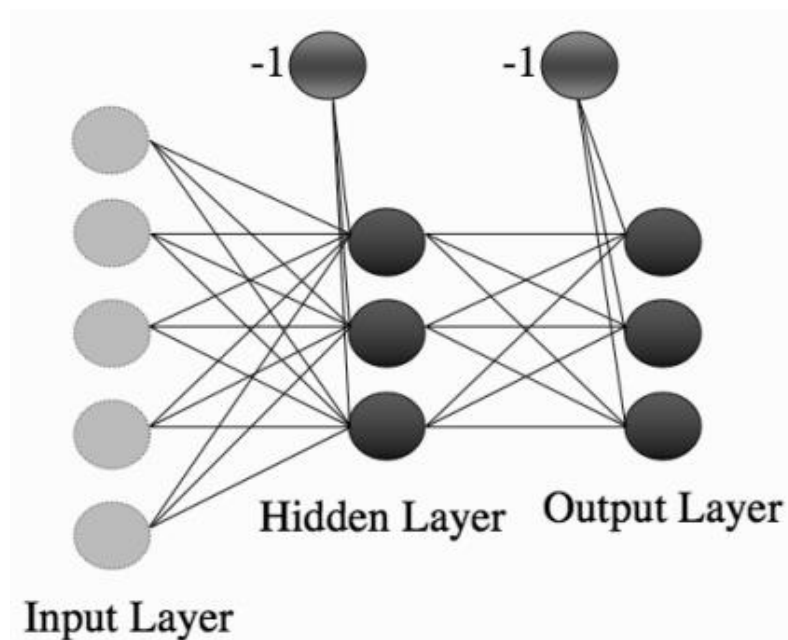
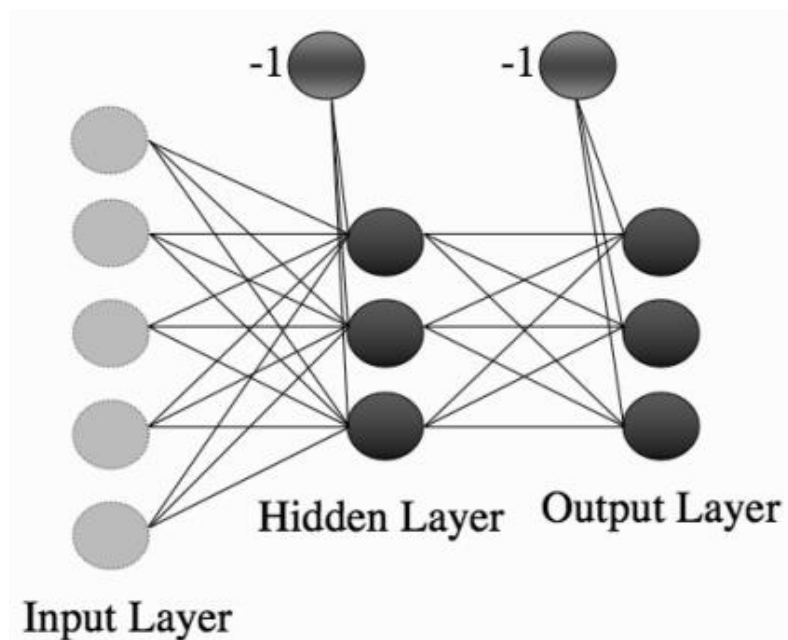


多层神经网络隐层处理后的特征空间

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

# 工作机制：向前和向后



计算误差



向前：分阶段，

逐层计算输入和输出

向后：分阶段，

逐层调整权值

# 大纲

多层感知机MLP

回顾

反向传播

其他议题

自动编码器AUTOENCODING

径向基网络RBF

# 向后：误差的反向传播

## □ 误差反传(Back-propagation of error)要素

- ✓ 误差定义

- ✓ Delta规则

- ✓ 激活函数

- ✓ 反传学习的推导(链式法则)


# 误差定义

经典感知机中  $N = 1$

## □ 感知器

$$\sum_{k=1}^N E_k = \sum_{k=1}^N (y_k - t_k)$$


## □ 多层感知机(BP神经网络)

$$E(\mathbf{t}, \mathbf{y}) = \frac{1}{2} \sum_{k=1}^N (y_k - t_k)^2$$


$\frac{1}{2}$  非必须

# Delta规则

- Delta规则是基于误差平面的。
- 误差平面是神经网络所表示的函数在(训练)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。

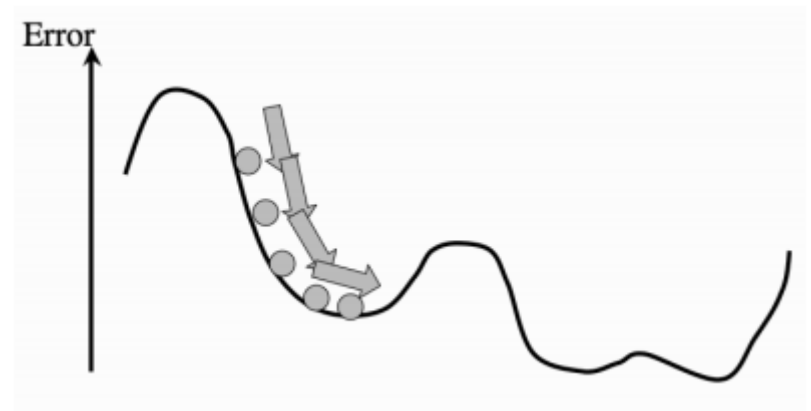
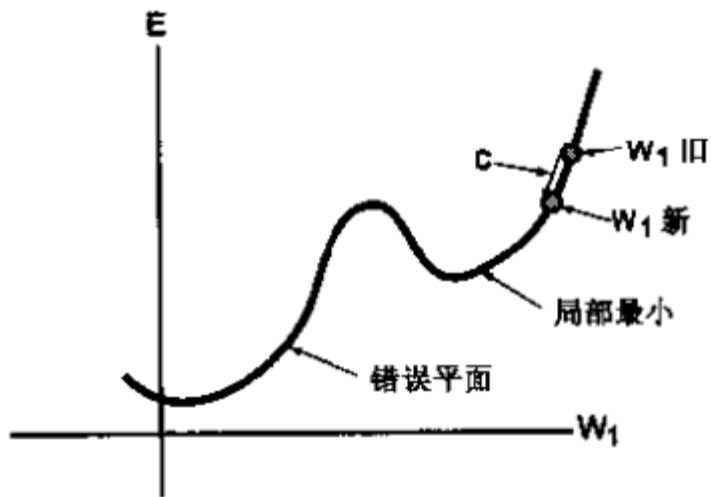


图 10-8 二维坐标中的错

常数  $c$  指示了学习步幅的大小

# Delta规则

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

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

此页符号标记前后有差异，  
学习时请注意

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

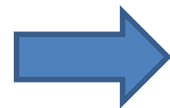
因为输出层中的节点的误差并不影响其他节点，因此

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

与感知机调节权值方法一致！

# Delta规则

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


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

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

$$= c(d_i - O_i) f'(\text{net}_i) * X_k$$

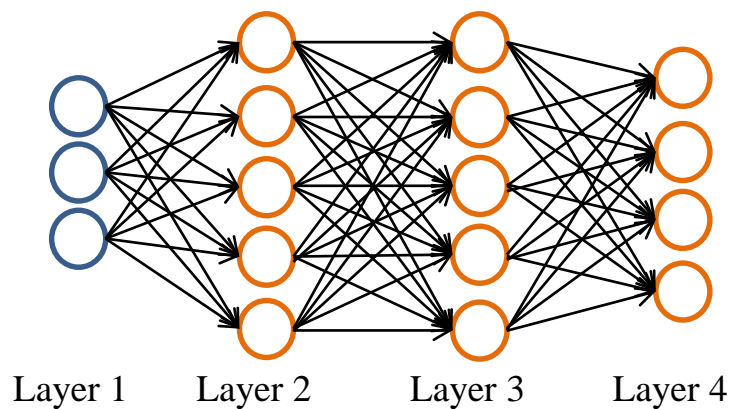
思考与感知机学习规则的区别！



# 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误差反传算法(BP)的核心，反传算法是多层神经网络中的学习算法。

# 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

第一个就是对 $\frac{1}{2}x^2$ 求导(对 $x$ )，它是 $x$ ；

推导所使用的  
数学技巧

第二个是链式法则，即 $\frac{dy}{dx} = \frac{dy}{dt} \frac{dt}{dx}$ ；

第三个是： $\frac{dy}{dx} = 0$ 如果 $y$ 不是 $x$ 的函数。

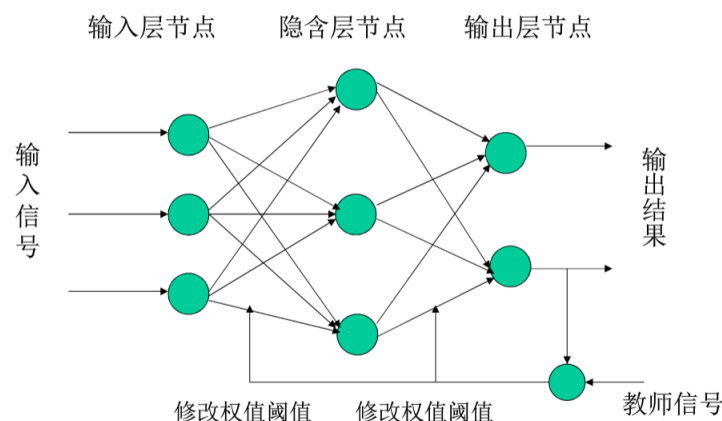
# 反向传播算法Back Propagation

- 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 (机器学习中的核心难题)
  - ✓ 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？

# 多层感知机→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

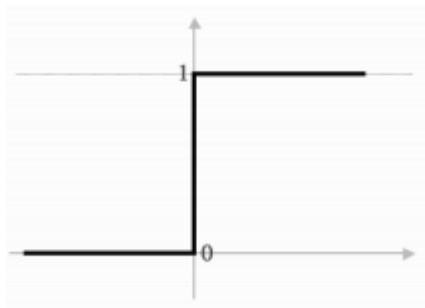


BP本质是学习算法，被广泛应用到MLP, ADE, KBF, DNN中。

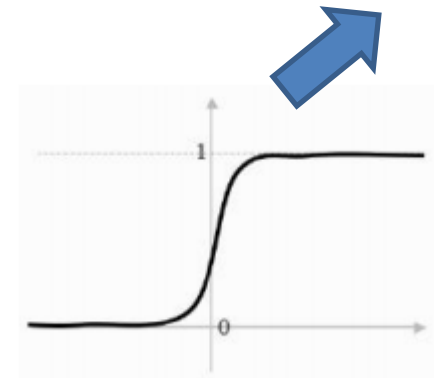
但很多时候，又把MLP称为BP神经网络。

# 激励函数

饱和型函数



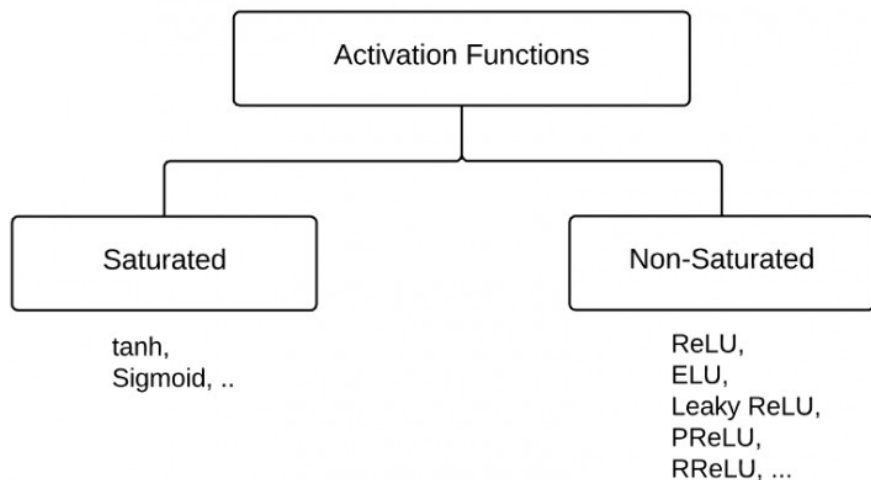
感知机中所使用的阶  
跃函数



BP神经网络中  
所常用的Sigmod函数

$$a = g(h) = \frac{1}{1 + \exp(-\beta h)}$$

# 激励函数的分类

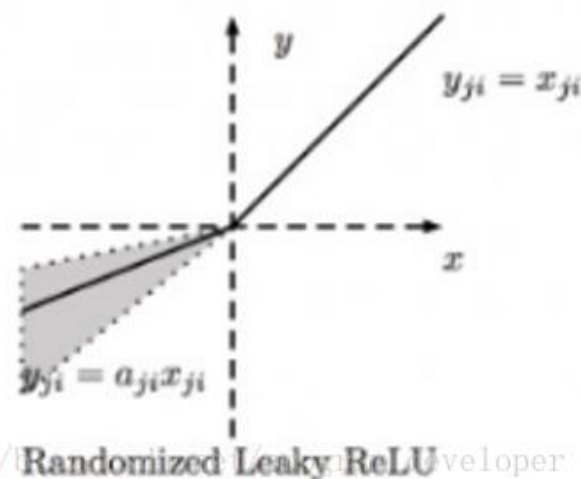
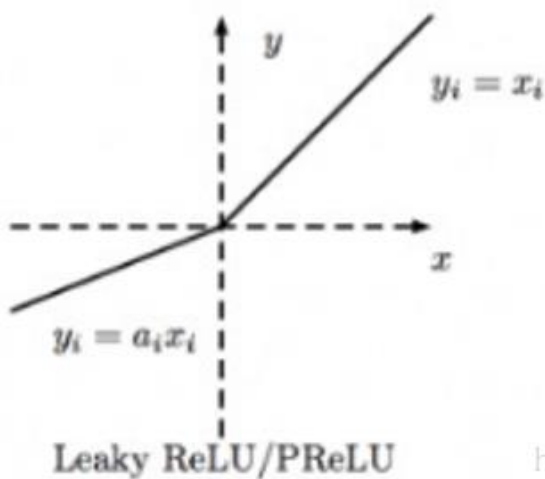
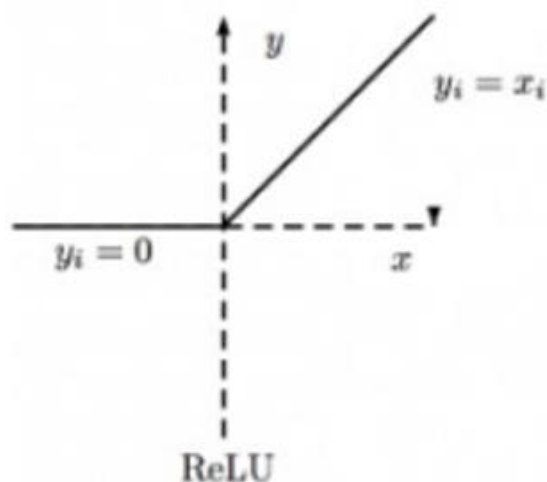


饱和型激励函数的缺点：

1、梯度消失

2、非以0为中心

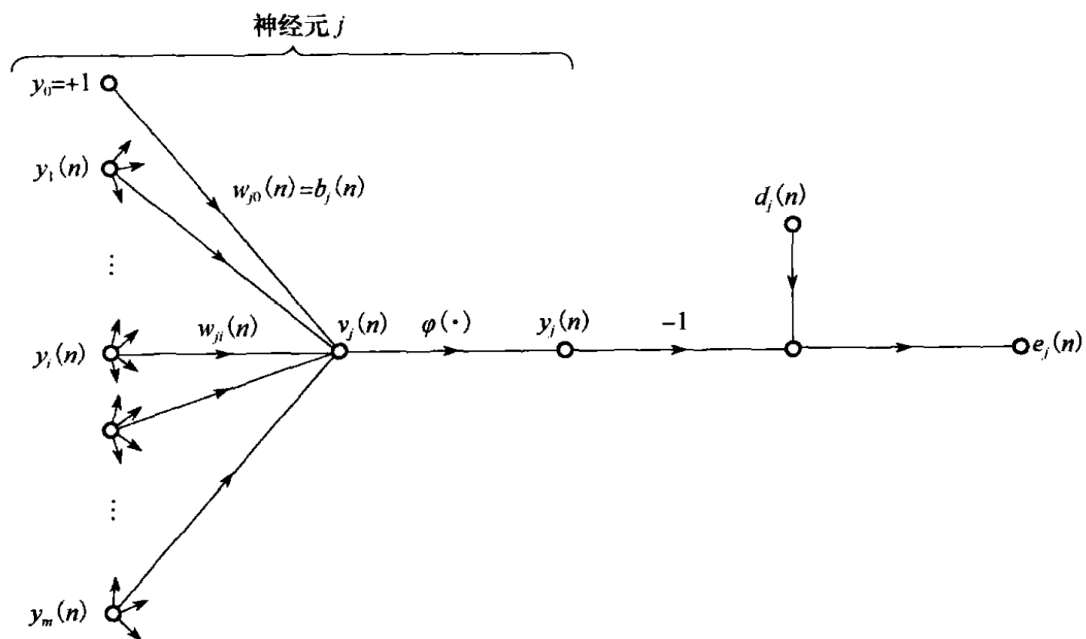
3、指数计算代价大



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

# BP误差反传学习推导

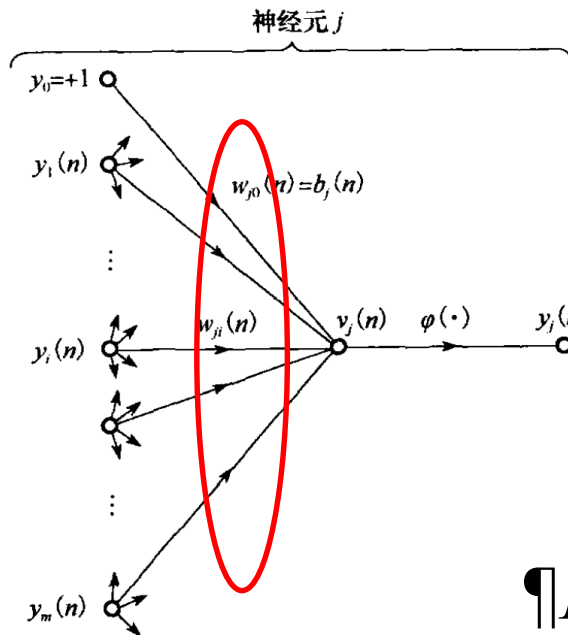
## 图解



- ✓  $y_i$  第 $j$ 个神经元的第 $i$ 个输入
- ✓  $n$  第 $n$ 个训练样本
- ✓  $j$  第 $j$ 个神经元
- ✓  $w_{ji}$  第 $j$ 个神经元和第 $i$ 个输入间的权值
- ✓  $v$  输入加权和
- ✓  $\Phi$  激活函数
- ✓  $y_j$  第 $j$ 个神经元的输出
- ✓  $d$  样本真实值
- ✓  $e$  误差值

后续推导符号标记与教材4.6节有差异，学习时请注意

# 突触权值修正



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

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

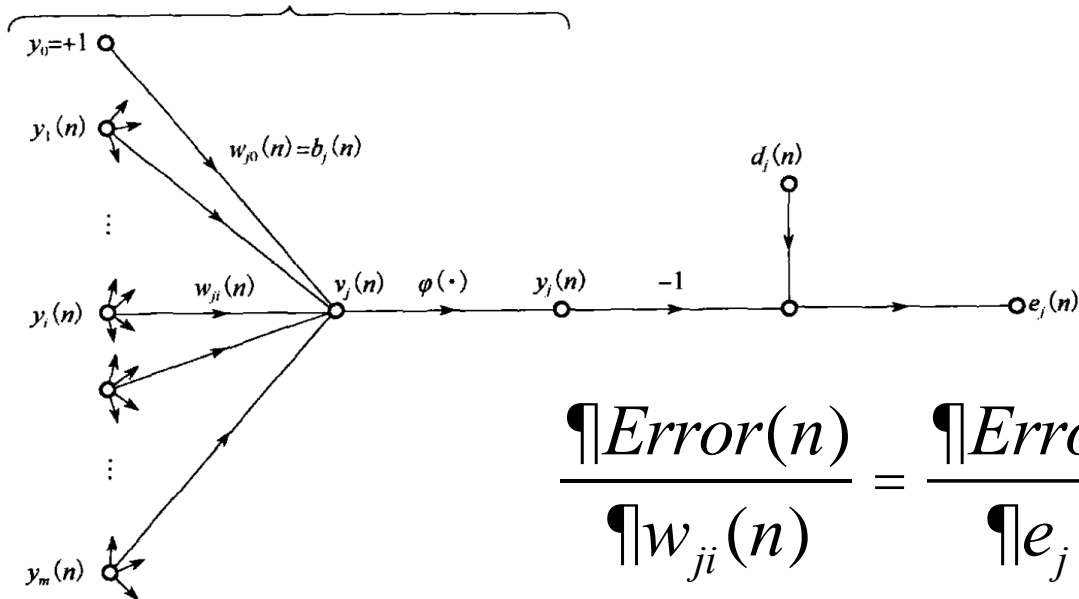
根据链式规则

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

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。



神经元j



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

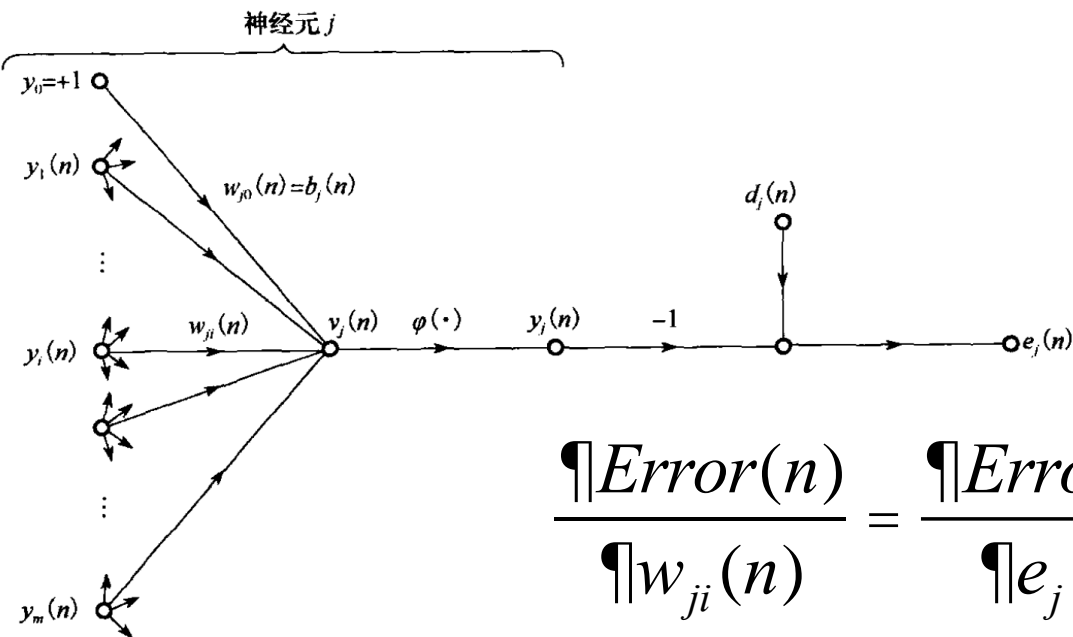
$$y_j(n) = f_j(v_j(n))$$

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

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

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

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

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

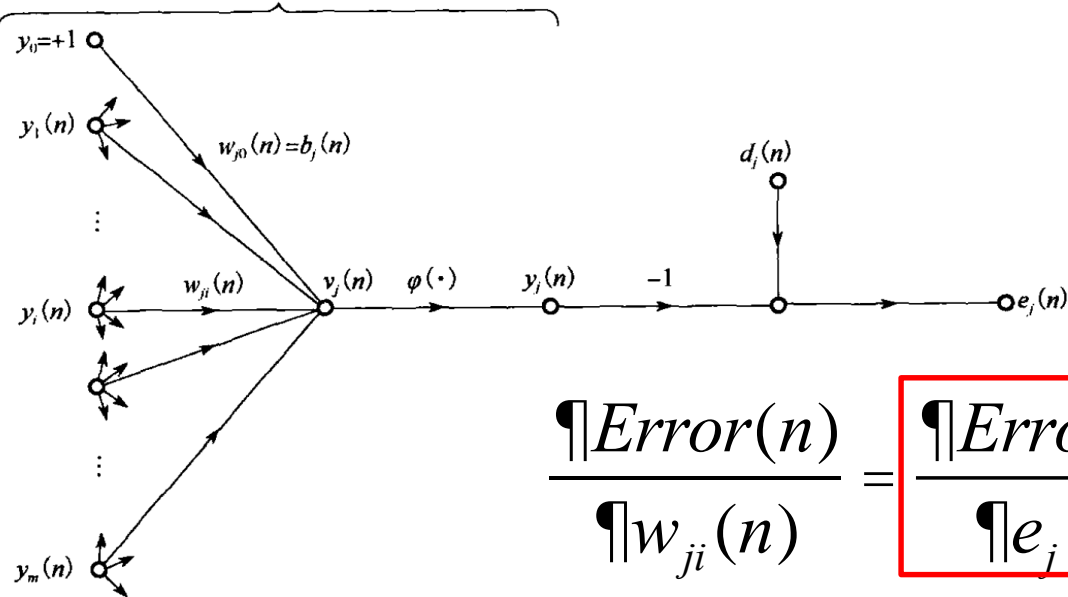
因此可求的偏导数为

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



和Delta规则没区别

神经元  $j$



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

定义局域梯度  $\delta_j(n)$

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

因此权值修正定义为

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

# 权值修正的两种情况

□ Case1: 神经元j是输出层节点

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

求解局域梯度 $\delta_j(n)$ :

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

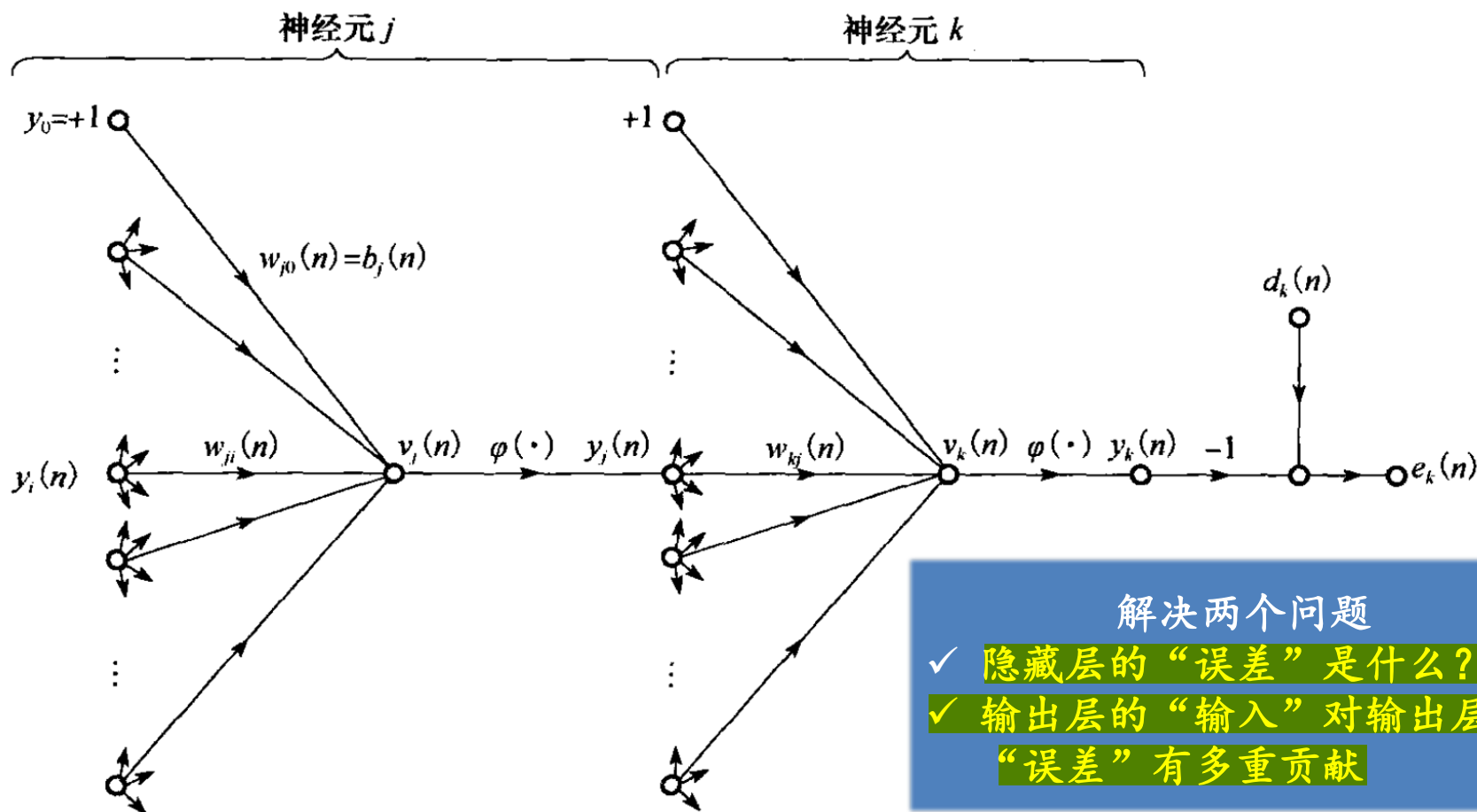
因此权值修正值为:

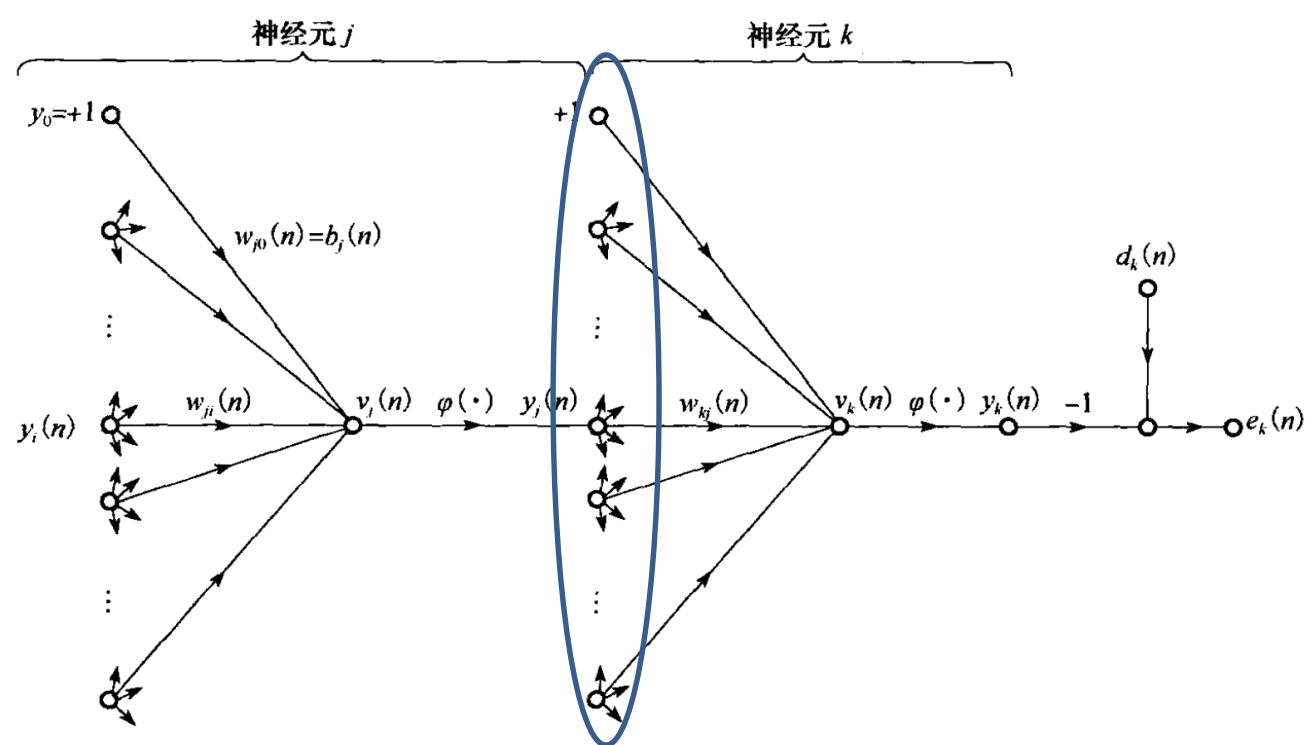
$$\Delta w_{ji}(n) = a * e_j(n) * f_j'(v_j(n)) * y_i(n)$$

# 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。





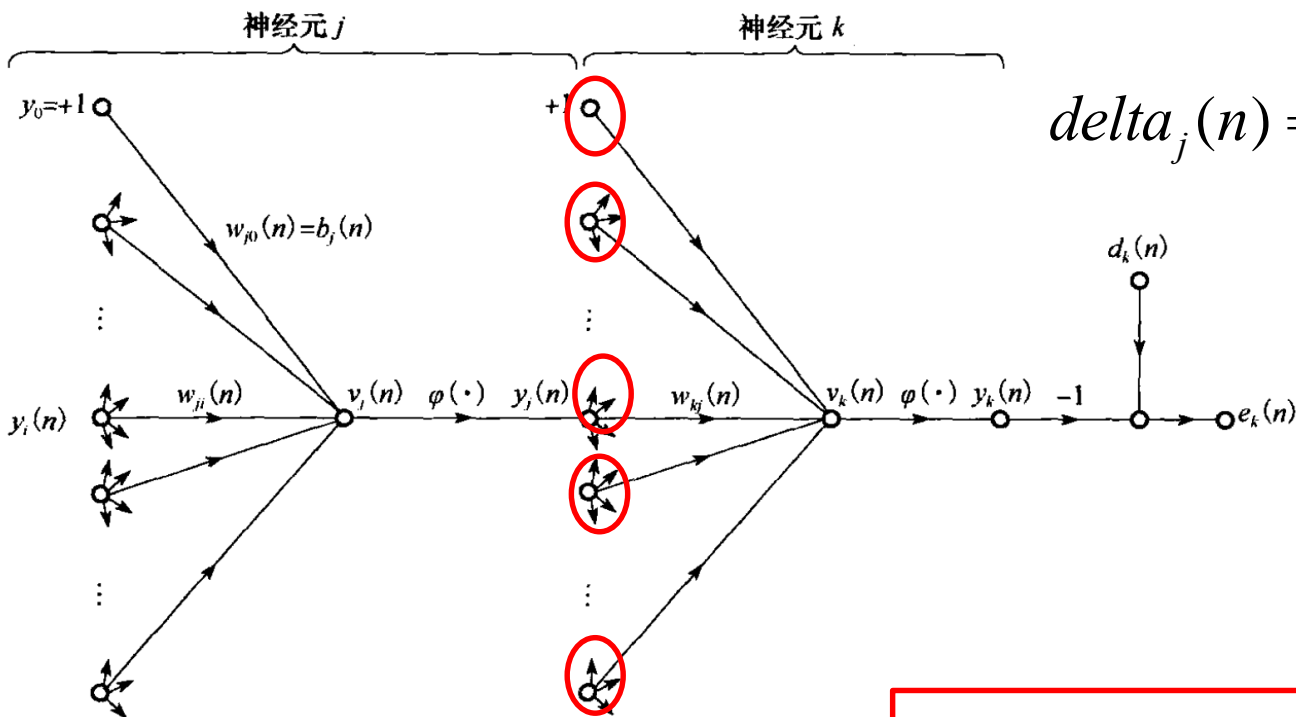
隐藏层神经元不能直接访问，但是它们必须分担对网络输出的误差责任。

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  $\delta_j(n)$

注意分子部分

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



$$\delta_j(n) = -\frac{\|Error(n)\|}{\|y_j(n)\|} j'_j(v_j(n))$$

假设神经元k为输出层神经元

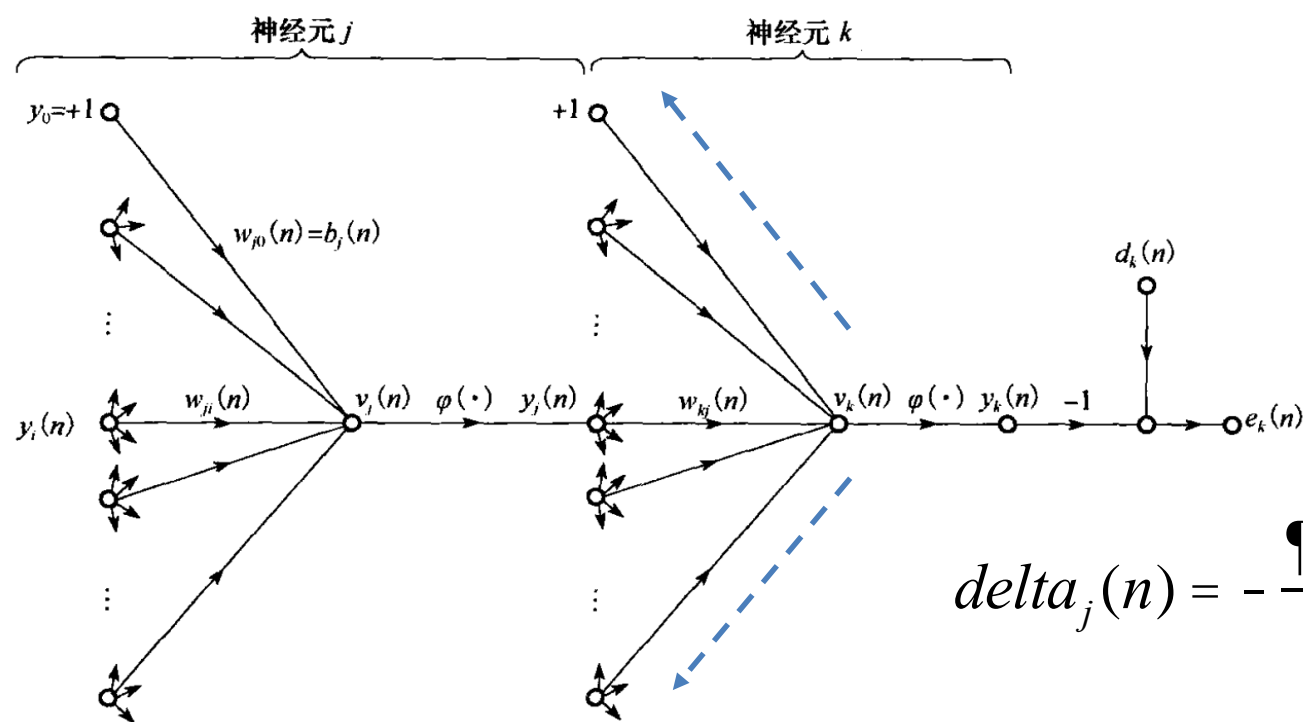
计算隐藏层神经元j，对网络输出层  
神经元k(有若干个k)的误差的责任

$$Error(n) = \frac{1}{2} \sum_k \dot{a}_k e_k^2(n)$$

则可求解

注意为什么这样

$$\frac{\|Error(n)\|}{\|y_j(n)\|} = \sum_k \dot{a}_k e_k \frac{\|e_k(n)\|}{\|y_j(n)\|} = \sum_k \dot{a}_k e_k \frac{\|e_k(n)\|}{\|v_k(n)\|} \frac{\|v_k(n)\|}{\|y_j(n)\|}$$



$$\text{delta}_j(n) = - \frac{\nabla \text{Error}(n)}{\nabla y_j(n)} j'_j(v_j(n))$$

又因为

$$e_k(n) = d_k(n) - y_k(n) = d_k(n) - j'_k(v_k(n))$$

所以

$$\frac{\nabla \text{Error}(n)}{\nabla y_j(n)} = - \dot{\sum}_k e_k(n) j'_k(v_k(n)) w_{kj}(n)$$

根据局域梯度的定义，可得

$$\frac{\nabla \text{Error}(n)}{\nabla y_j(n)} = - \dot{\sum}_k \text{delta}_k(n) w_{kj}(n)$$



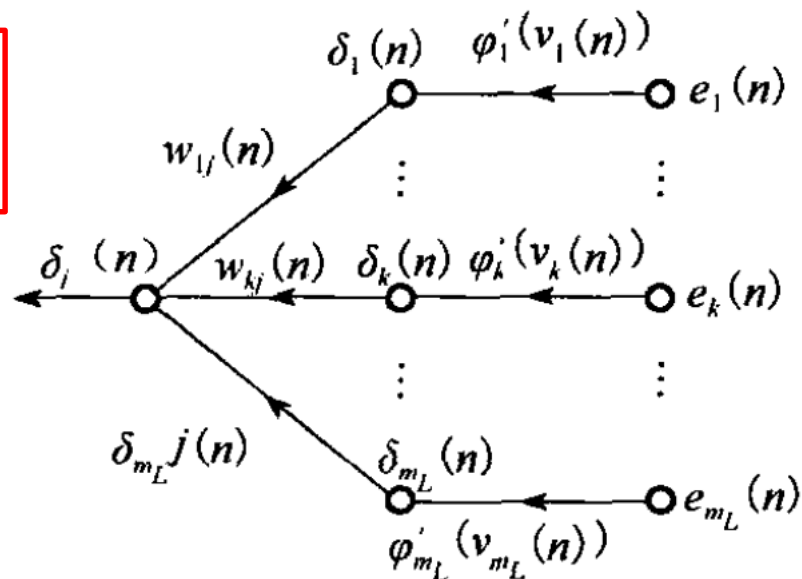
认真思考此公式的物理意义  
这个公式是反向传播的“真谛”



因此

$$\delta_j(n) = f_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

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

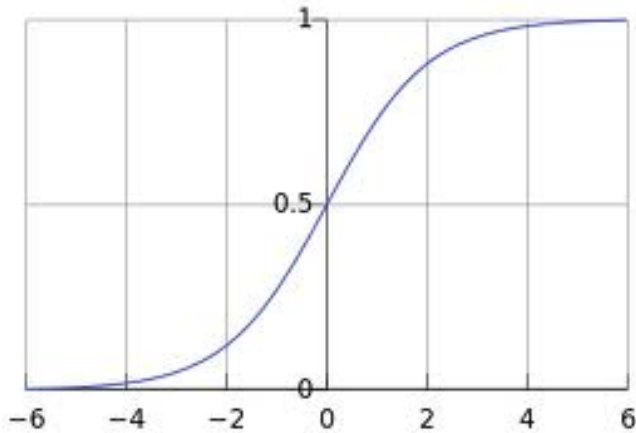


❑ 反向传播的误差信号的转变——局域梯度delta:

❑ 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激活函数



## □ 特点

- ✓  $\lambda$ 是挤压参数，值越大，区间 $[0,1]$ 上越接近直线。
- ✓ 求导计算简化为加乘运算(但仍需要计算指数项)。

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

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

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

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

# 反向传播算法

初始化：随机挑选突触权值

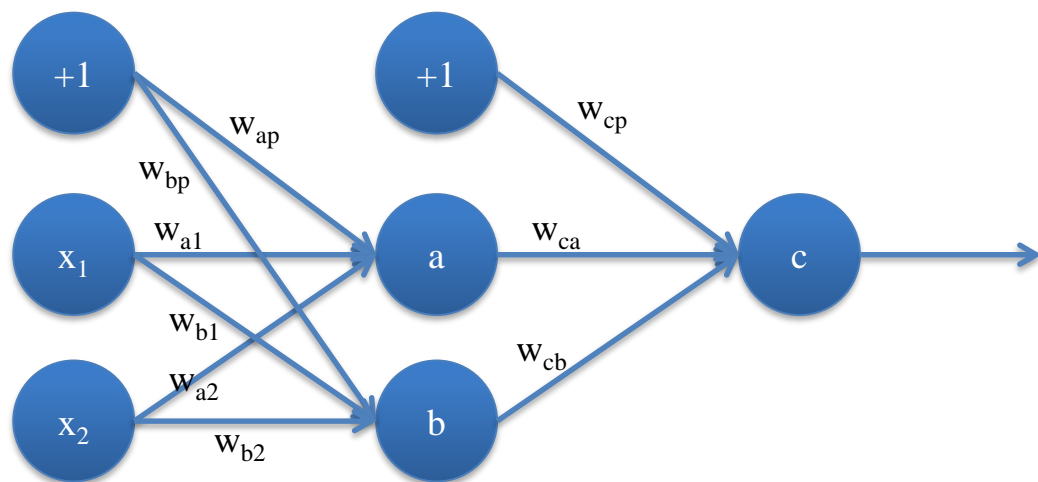
训练样本呈现：对训练集中的样本  
进行后续计算

前向计算：每次迭代输入一个训练  
样本，并得到网络输出

反向计算：首先反向传播计算每一层  
神经元的局域梯度 $\delta$ ，然后计算修  
正值 $\Delta w$ 修正突触权值



# 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。

# 大纲

多层感知机MLP

回顾

反向传播

其他议题

自动编码器AUTOENCODING

径向基网络RBF

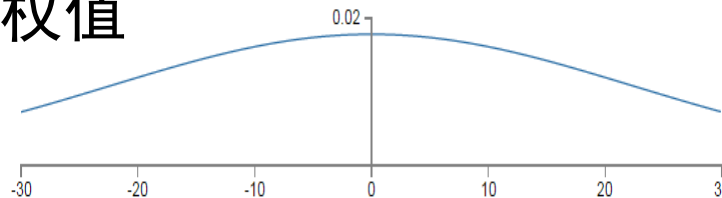
# 其他议题

- 初始权值
- 激活函数选择
- 顺序和批量训练
- 局部极小和冲量
- 停止机制

# 初始权值

□ 模型选择，需要多次设定初始随机权值

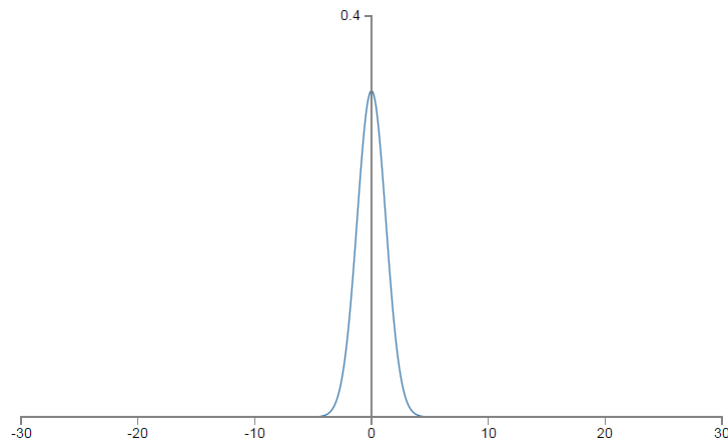
□ 简易方法：权值  $w \sim (0, 1)$



□ 独立变量和的方差 = 独立变量方差的和，输出值  $\sim (0, n)$

□ 导致输出值过大，进一步导致激活函数饱和，梯度趋近于0，学习失效

□ 因此：权值  $w \sim (0, 1/\sqrt{n})$



# 激活函数选择

□ 分类问题：sigmoid函数

□ 回归问题：输出节点的激活函数， $y_k = g(h) = h$

□ Soft-max激活函数  $y_k = g(h_k) = \frac{\exp(h_k)}{\sum_{k=1}^N \exp(h_k)}$

□ 其他深度学习中用的非饱和激活函数



# 顺序和批量训练

## □ 批量训练(梯度下降 gradient descent)

- ✓ 计算所有训练样本的误差和。缺点：计算代价大/收敛速度快/局部极小

## □ 顺序训练

- ✓ 按次序计算每个训练样本的误差。不需要计算总误差，快/局部极小/噪声敏感

## □ 小批量训练(随机梯度下降 stochastic gradient descent)

- ✓ Mini-Batch → SGD。适用于大规模数据。

# 局部极小和冲量

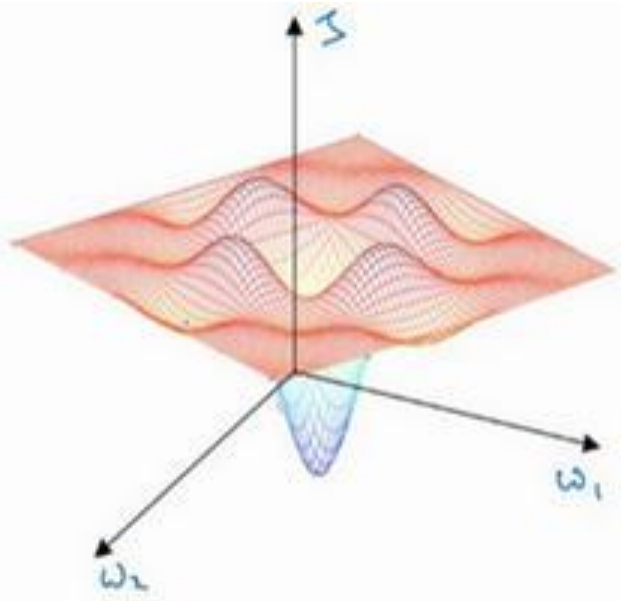
□ **冲量** (此部分学习可结合第7讲)

- ✓ 加大搜索步长的效果，能更快的进行收敛。
- ✓ 越过某些狭窄的局部极小值，达到更小的地方

$$w_{\zeta\kappa}^t \leftarrow w_{\zeta\kappa}^{t-1} + \eta \delta_o(\kappa) a_{\zeta}^{hidden} + \alpha \Delta w_{\zeta\kappa}^{t-1}$$

梯度为0的点，并不是全局最优。  
那么，对学习算法如何调整呢？

$$\Delta w_{\zeta\kappa}^t = \eta \delta_o(\kappa) a_{\zeta}^{hidden} + \alpha \Delta w_{\zeta\kappa}^{t-1}$$



# 停止机制

- 设定固定迭代步数
- 设定误差小于某个固定阈值
- 以上两者的结合



易导致过拟合

- 利用验证集观察误差的变化，找到合适的参数(模型选择)

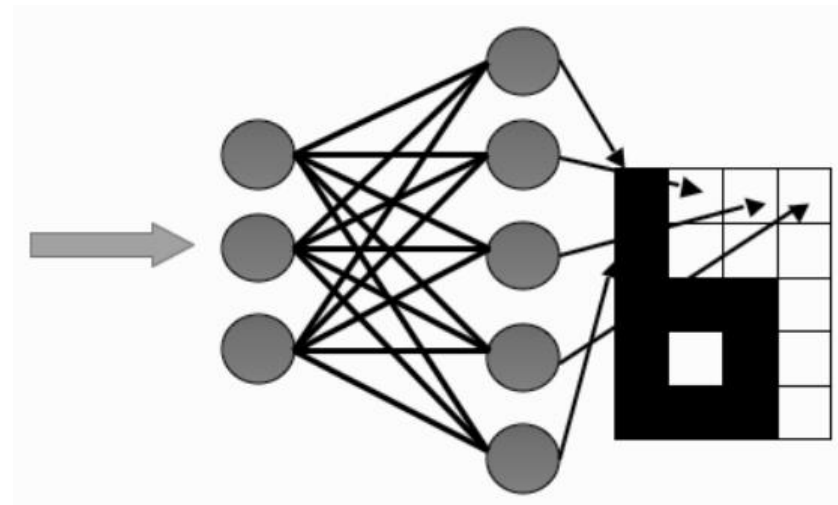
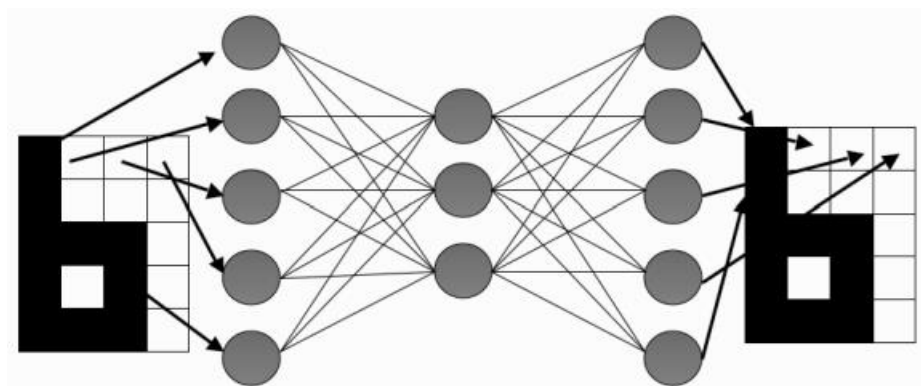
# 大纲

多层感知机MLP

自动编码器AUTOENCODER

径向基网络RBF

# 自动编码器

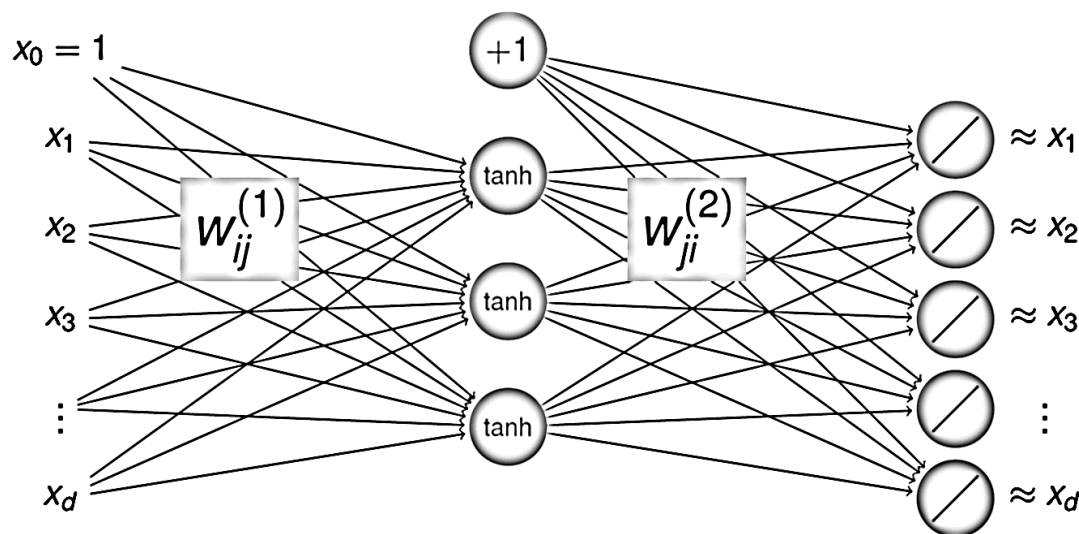


$$\phi : \mathcal{X} \rightarrow \mathcal{F}$$

$$\psi : \mathcal{F} \rightarrow \mathcal{X}$$

$$\phi, \psi = \arg \min_{\phi, \psi} \|X - (\psi \circ \phi)X\|^2$$

# 自动编码器



- ✓  $W_{ij}$  作为  $W_{ji}$  的逆函数
- ✓ 中间层输出加上正则化项(例如稀疏约束)

$$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

$$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

# 大纲

多层感知机MLP

自动编码器AUTOENCODING

径向基网络RBF

# 感受野

编辑

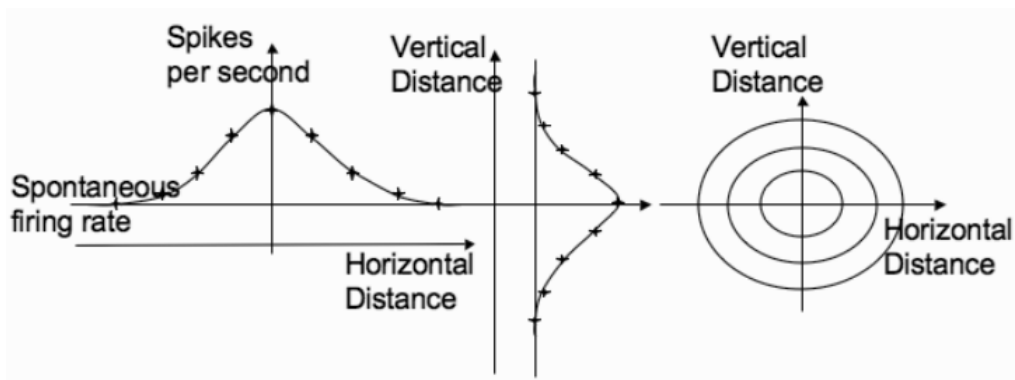
讨论

上传视频

感受野，感受器受刺激兴奋时，通过感受器官中的向心神经元（支配）的刺激区域就叫做神经元的感受野（receptive field）。感觉区的神经元都有各自的感受野。随感觉种类不同，感受野的性质

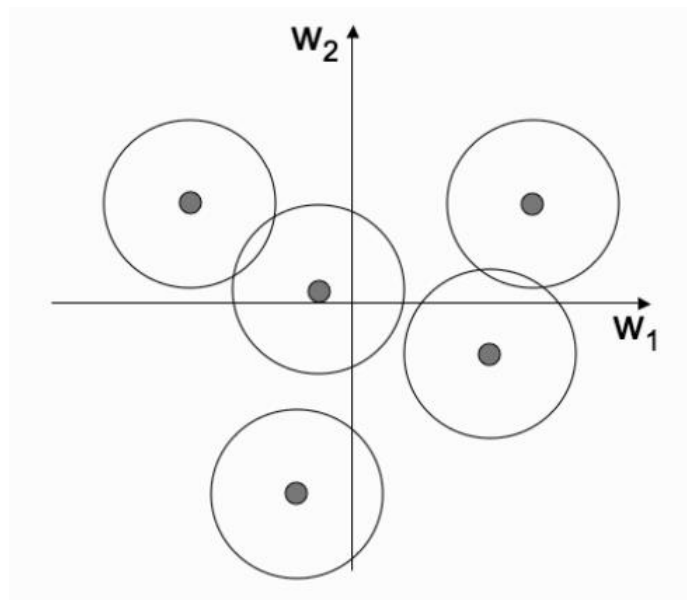
在视觉通路上，视网膜上的光感受器（杆体细胞和锥体细胞）细胞，外膝状体细胞以及视觉皮层中的神经细胞。反过来，任何一种依赖于视网膜上的许多光感受器。我们称直接或间接影响某一特定（receptive field）。

# 感受野



$$g(\mathbf{x}, \mathbf{w}, \sigma) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{w}\|^2}{2\sigma^2}\right)$$

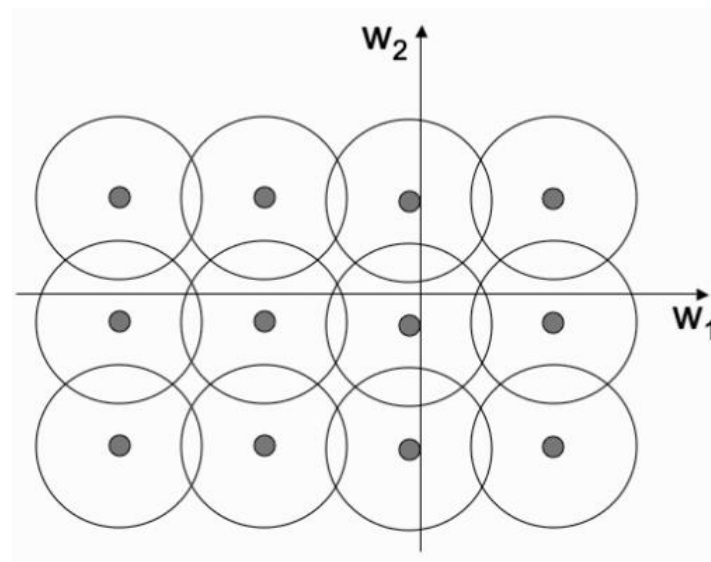
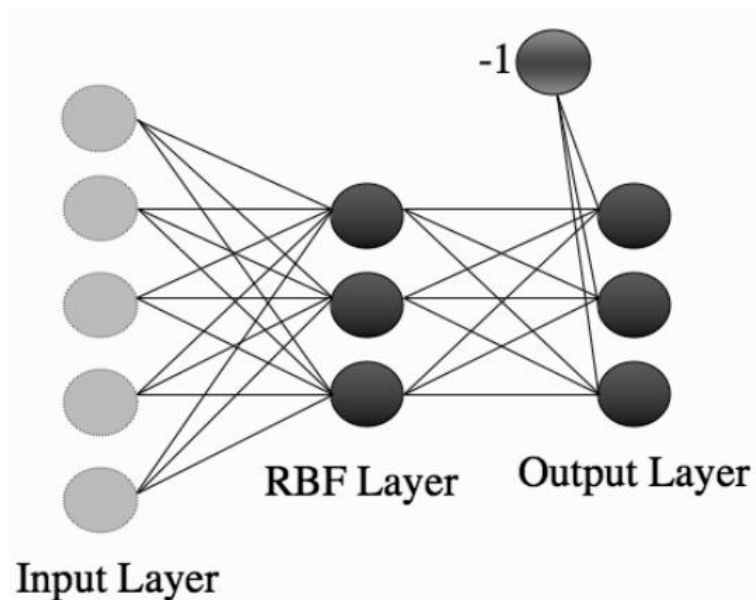
注意：径向基函数的选择有很多种，  
例如最简单的欧式距离



中心点：径向基位置；  
圆圈大小：感受野尺寸。

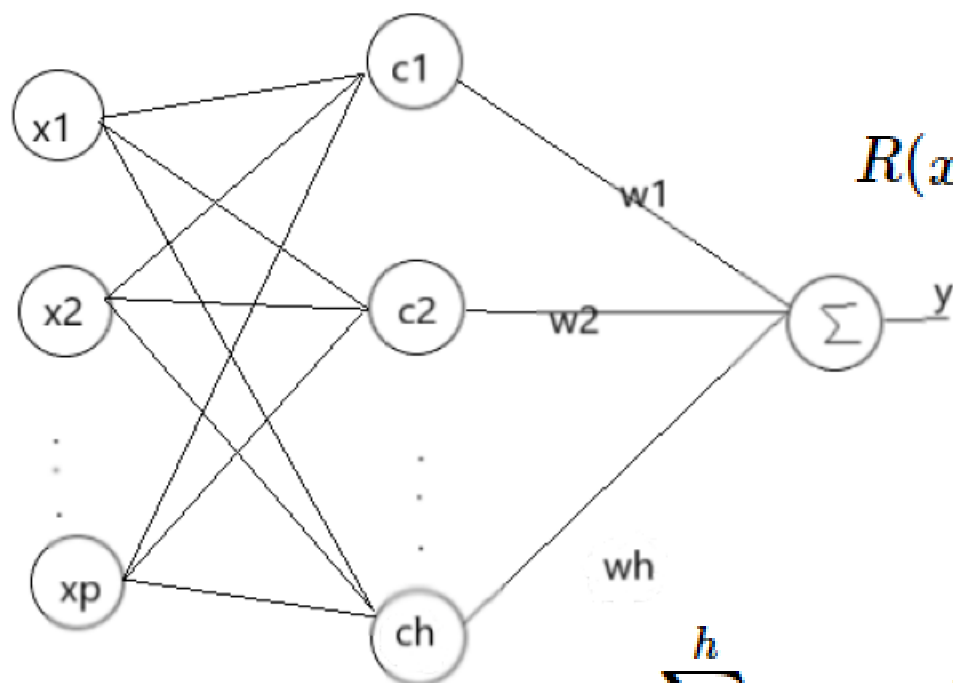


# 径向基网络



- RBF层激活规则：基于距离，局部匹配
- MLP激活规则：基于内积，全局匹配

# 径向基网络的学习



$$R(x_p - c_i) = \exp\left(-\frac{1}{2\sigma^2} \|x_p - c_i\|^2\right)$$

$$\sigma_i = \frac{c_{max}}{\sqrt{2h}} \quad i = 1, 2, \dots, h$$

$$y_j = \sum_{i=1}^h w_{ij} \exp\left(-\frac{1}{2\sigma^2} \|x_p - c_i\|^2\right) \quad j = 1, 2, \dots, n$$

损失

$$ERROR = \sum_j^n (d_j - \phi_j(y_j))$$

# 径向基网络的学习

求解的参数有3个：基函数的中心、方差以及隐含层到输出层的权值。

## 径向基函数算法

□ 以任一种方式放置RBF的中心：

- ✓ 用均值算法初始化RBF中心的位置 或
- ✓ 用随机选择的数据点作为RBF的中心

□ 用公式计算RBF节点的行为  $g(\mathbf{x}, \mathbf{w}, \sigma) = \exp\left(\frac{-\|\mathbf{x} - \mathbf{w}\|^2}{2\sigma^2}\right)$

□ 用任一种方式训练输出的权值

- ✓ 用感知机 或
- ✓ 计算RBF中心活化的伪逆

$$\mathbf{G}^+ = (\mathbf{G}^T \mathbf{G})^{-1} \mathbf{G}^T$$

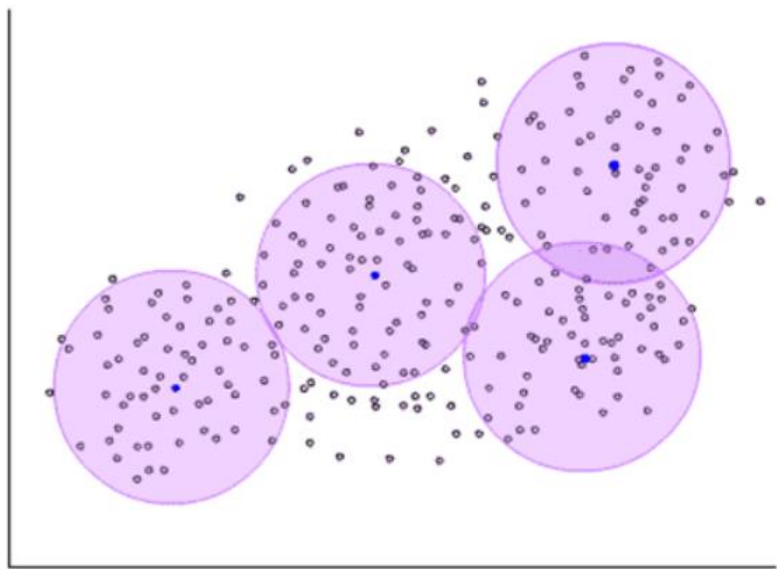
# 径向基网络的原理

- 用RBF作为隐单元的“基”构成隐含层空间，通过确定RBF的中心点，直接将输入矢量映射到隐空间，不需要通过权连接。
- 隐含层空间到输出空间的映射是线性的，网络的输出是隐单元输出的线性加权和，此处的权即为网络可调参数。
- 隐含层的作用是把向量从低维度的 $p$ 映射到高维度的 $h$ ，这样低维度线性不可分的情况到高维度就可以变得线性可分，主要就是核函数的思想。
- 优点
  - ✓ 网络由输入到输出的映射是非线性的，而网络输出对可调参数而言却又是线性的。
  - ✓ 网络的权就可由线性方程组直接解出，从而大大加快学习速度并避免局部极小问题。

# 径向基网络 vs 多层感知机

## □ 局部逼近与全局逼近

- ✓ BP神经网络是对非线性映射的**全局逼近**。
- ✓ RBF神经网络具有“**局部映射**”特性。



所谓局部逼近是指目标函数的逼近  
仅仅根据查询点附近的数据。

# 径向基网络 vs 多层感知机

## □ 中间层数的区别

- ✓ BP神经网络可以有多个隐含层，但是RBF只有一个隐含层。

## □ 训练速度的区别

- ✓ 使用RBF的训练速度快，一方面是因为隐含层较少，另一方面，局部逼近可以简化计算量。对于一个输入 $x$ ，只有部分神经元会有响应，其他的都近似为0，对应的 $w$ 就不用调参了。

## □ 最优性

- ✓ Poggio和Girosi证明：RBF网络是连续函数的最佳逼近，而BP网络不是。

# 径向基网络 vs SVM

SVM中的高斯核函数可以看作与每一个输入点的距离，不太适应于大样本和大的特征数的情况

RBF神经网络对输入点做了一个聚类。RBF神经网络用高斯核函数时,其数据中心C可以是训练样本中的抽样，此时与SVM的高斯核函数是完全等价的。

RBF神经网络

$$G(X, X^p) = \exp\left(-\frac{1}{2\sigma^2} \|X - X^p\|^2\right)$$

SVM

$$\kappa(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

# 思考和讨论

1. BP学习算法中的隐藏参数学习？
2. 局部梯度域。
3. 自动编码器的结构和学习。
4. RBF与MLP的区别和联系。



# 必做实验一：神经网络

- ❑ 任务描述：用python语言，实现神经网络学习，完成“手写体识别”任务。建议：使用图像预处理技术（去噪，归一化，分割等），再使用CNN进行特征提取。
- ❑ 数据集：Mnist数据集（60000个训练样本和10000个测试样本组成，每个样本都是一张  $28 * 28$  像素的灰度手写数字图片），数据集下载：  
<http://yann.lecun.com/exdb/mnist/> （一共4个文件，训练集、训练集标签、测试集、测试集标签）
- ❑ 提交检查内容(包括但不限于):
  - ✓ 代码，关键代码需要注释；
  - ✓ PDF文档，文档中要有明确的过程说明、样例截图等。
- ❑ 评分标准：判断标准主要采用准确率，即精度。在精度指标以外可扩展其他指标，作为加分项。
- ❑ 作业提交：3月24日前至（文件名：学号-姓名-神经网络）

谢谢！