

模式识别

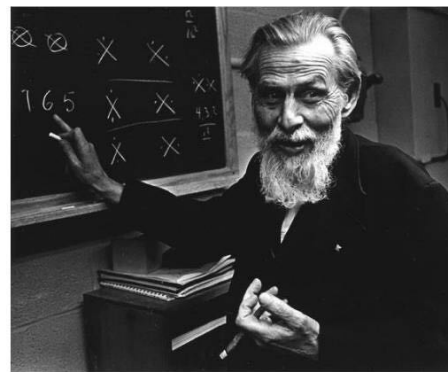
Pattern Recognition

第6讲 神经网络I

ANN history (1)

■ 萌芽期（--1949年）

- 1943年M-P模型：阈值加权和模型
 - 精神病学家、生理学家 McCulloch
 - 数学家 Pitts
 - M-P模型、神经网络逻辑演算
 - ANN原则上可以计算任何可计算函数
 - 标志着 ANN & AI 的诞生
- 1949年Hebb学习规则
 - 自学习

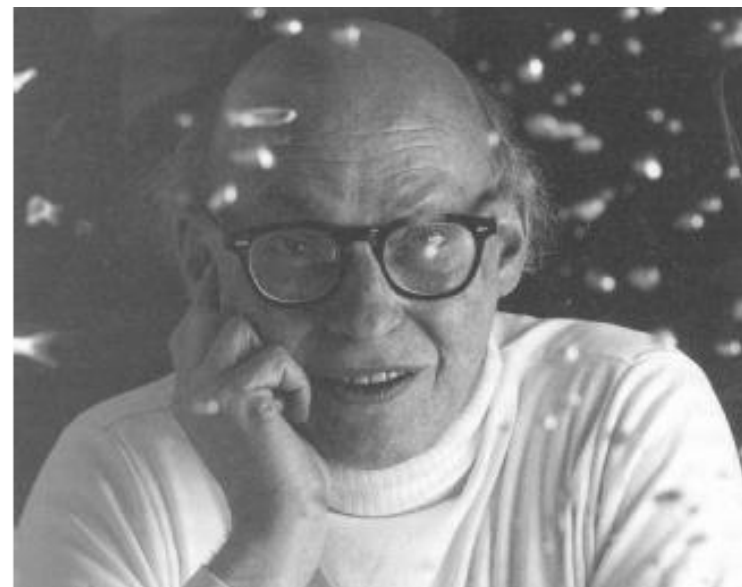


■ 第一次高潮（1950–68年）

- 1954: Minsky与Princeton大学“神经网络”博士论文
 - 1967: 《Computation: Finite and Infinite Machine》
 - 第一本以书的形式扩展了 M-P 的结果
- 1958: F. Rosenblatt
 - 单层感知器及其电路模拟
 - 感知器收敛定理
 - 学习定理
 - ANN可以学会它可以表达的任何东西
- Widrow提出了自适应线性元件Adaline
 - 成功用于雷达天线等连续可调过程的控制

■ 反思期（1969–1981年）

- Marvin Minsky & S. Papert
- 《Perceptron》，MIT, 1969
 - 著名人工智能学家、ANN专家
 - 造成了人工智能跌宕起伏的发展历程
 - 促成了三种不同的认知观的均衡发展和相互融合



2000多人 • 寥寥无几

■ 线性不可分问题

- XOR问题

■ 第二次高潮（1982–89年）

■ 1982年 Hopfield网

- 美国加州理工学院生物物理学家J. Hopfield
- 将Lyapunov函数、动力学观点引入到神经网络中来
- Hopfield网络的电路实现，使实用化成为可能

■ 1982年 Kohonen网

■ 1985年 Boltzmann机

■ 1986年 BP算法

- Rumelhart (86), Paker (82), Werbos (74), ...

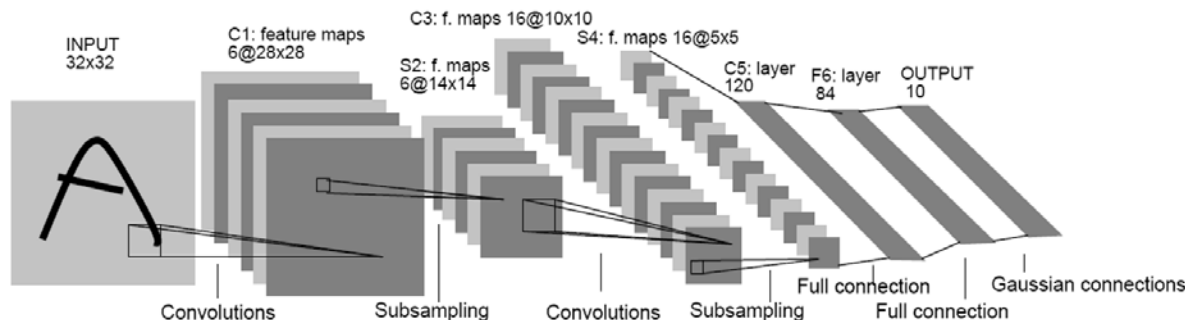
■ 1987年美国加州第一届神经网络国际会议

- 1000多学者参加

■ 国内首届神经网络大会1990年，北京

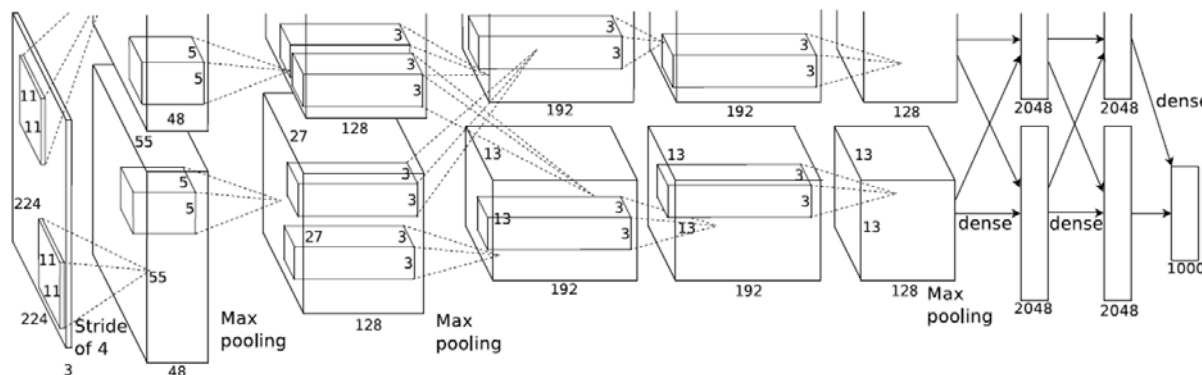
第三次高潮 2006-今天

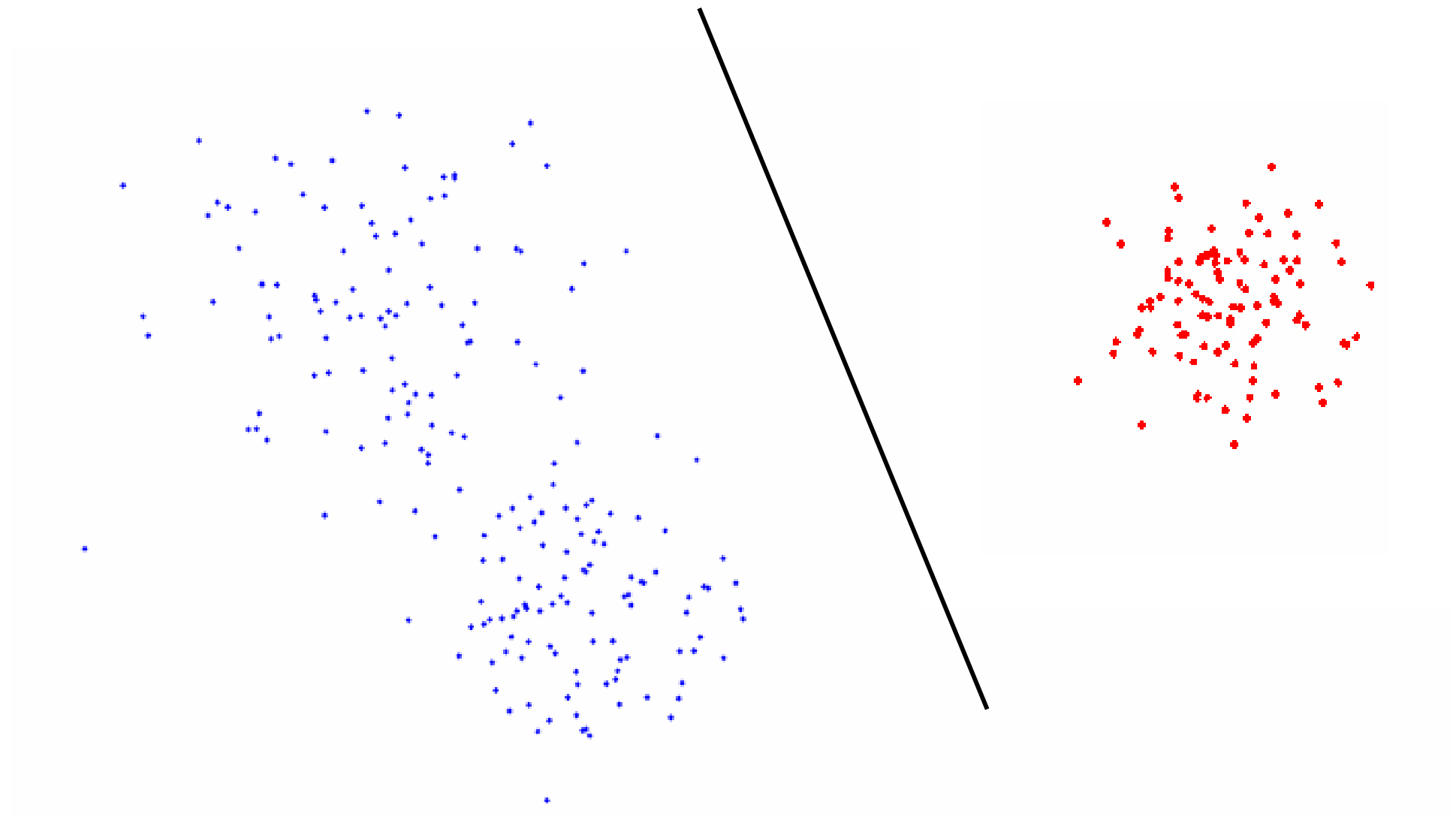
1989年Yann LeCun提出小型CNN 字符识别,

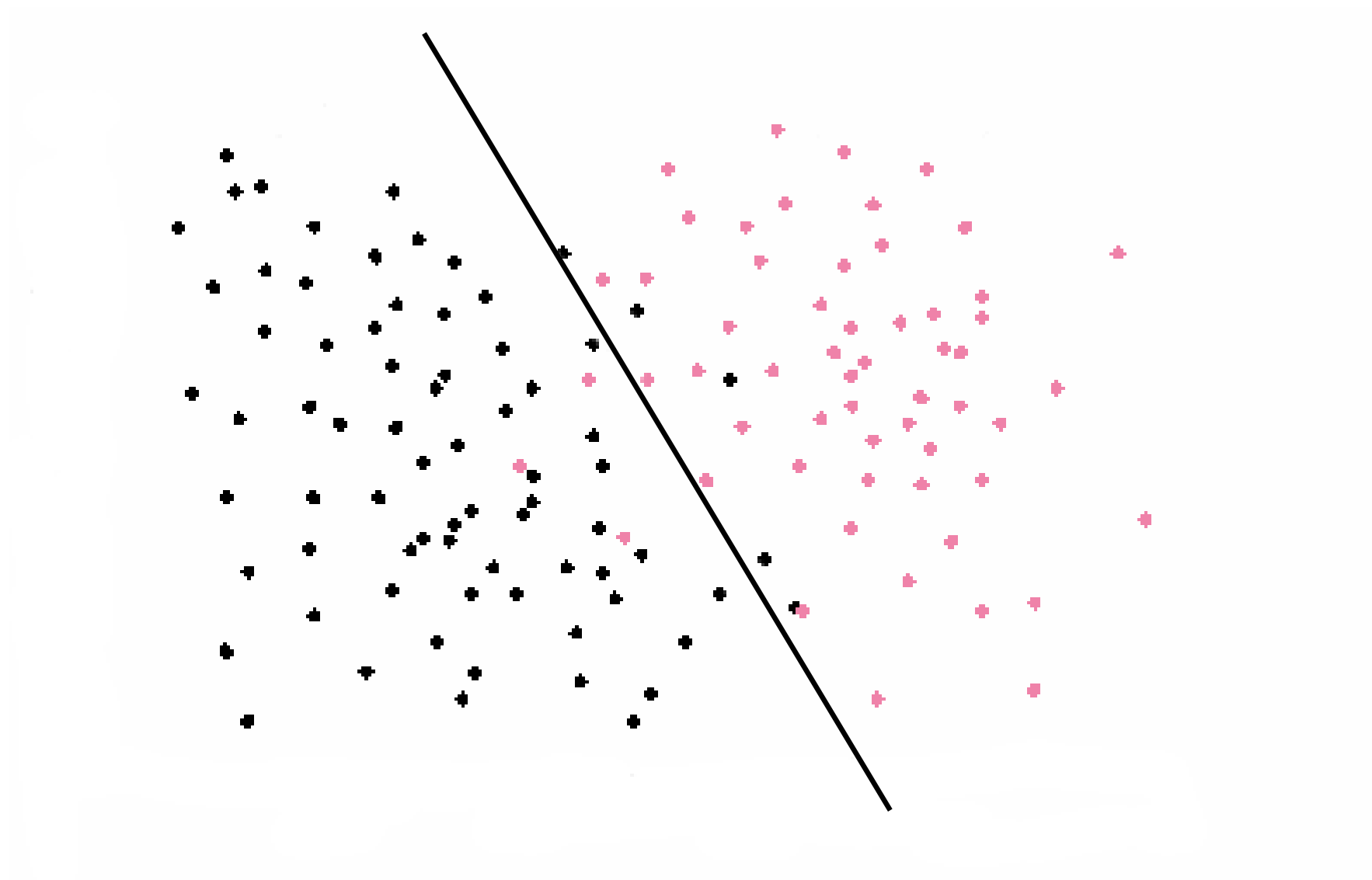


2006年Geoffrey Hinton提出了深度学习（多层神经网络）

2012年10月，Geoffrey Hinton和他的两个学生在著名的
ImageNet问题上用更深的CNN







1 线性判别函数和线性分类界面

□ 线性判别函数

	3维空间平面	d维空间超平面
代数形式	$w_1x_1 + w_2x_2 + w_3x_3 + w_0 = 0$	$w_1x_1 + w_2x_2 + \cdots + w_dx_d + w_0 = 0$
向量形式	$(w_1, w_2, w_3) \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + w_0 = 0$	$(w_1, w_2, \cdots, w_d) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} + w_0 = 0$
	$\mathbf{w}^T \mathbf{x} + w_0 = 0$ 其中 \mathbf{w} : 权值矢量 w_0 : 偏置	

d维特征空间中的超平面方程 $H: g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$

➤ 特征矢量: $\mathbf{x} = (x_1, x_2, \dots, x_d)^T$

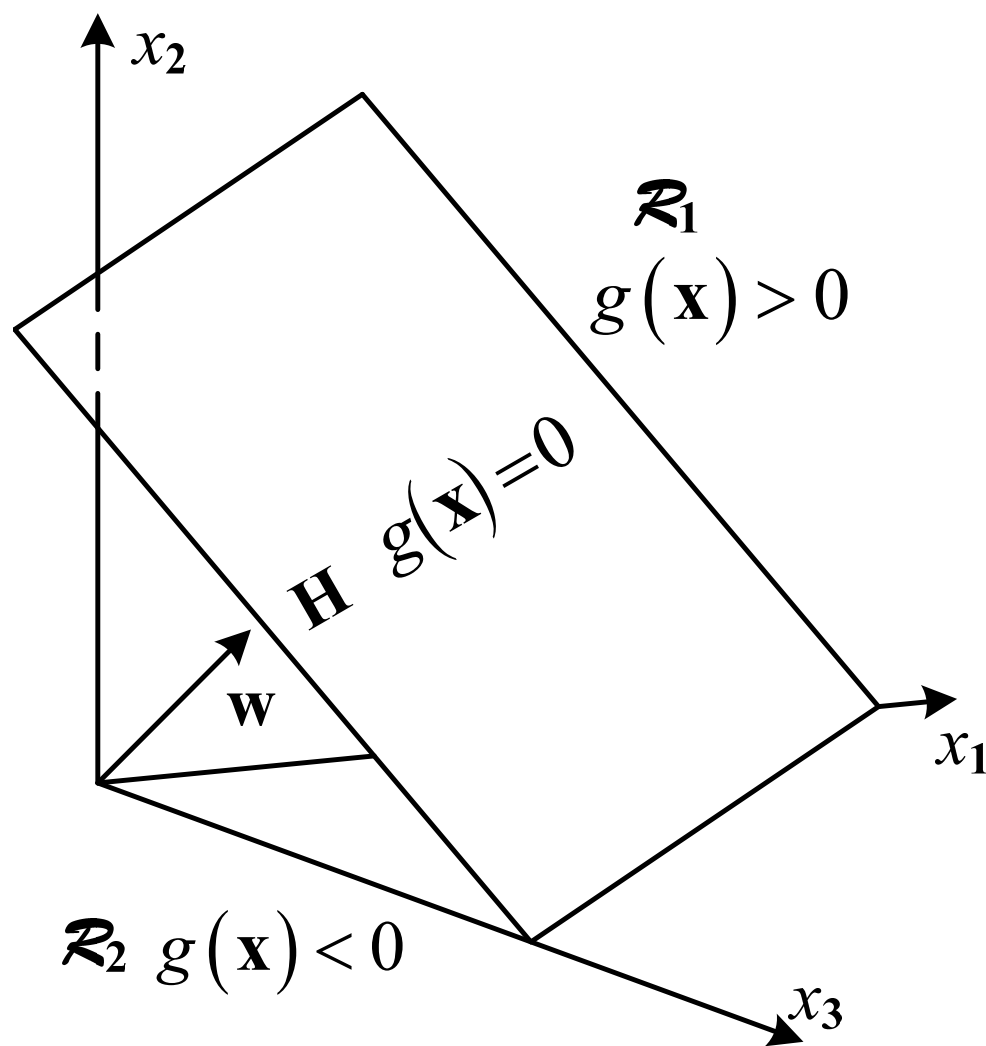
➤ 偏置(bias): w_0

➤ 权矢量 $\mathbf{w} = (w_1, w_2, \dots, w_d)^T$:

1 线性判别函数和线性分类界面

□ 线性判别函数

➤ H 将特征空间划分为两区域 R_1, R_2



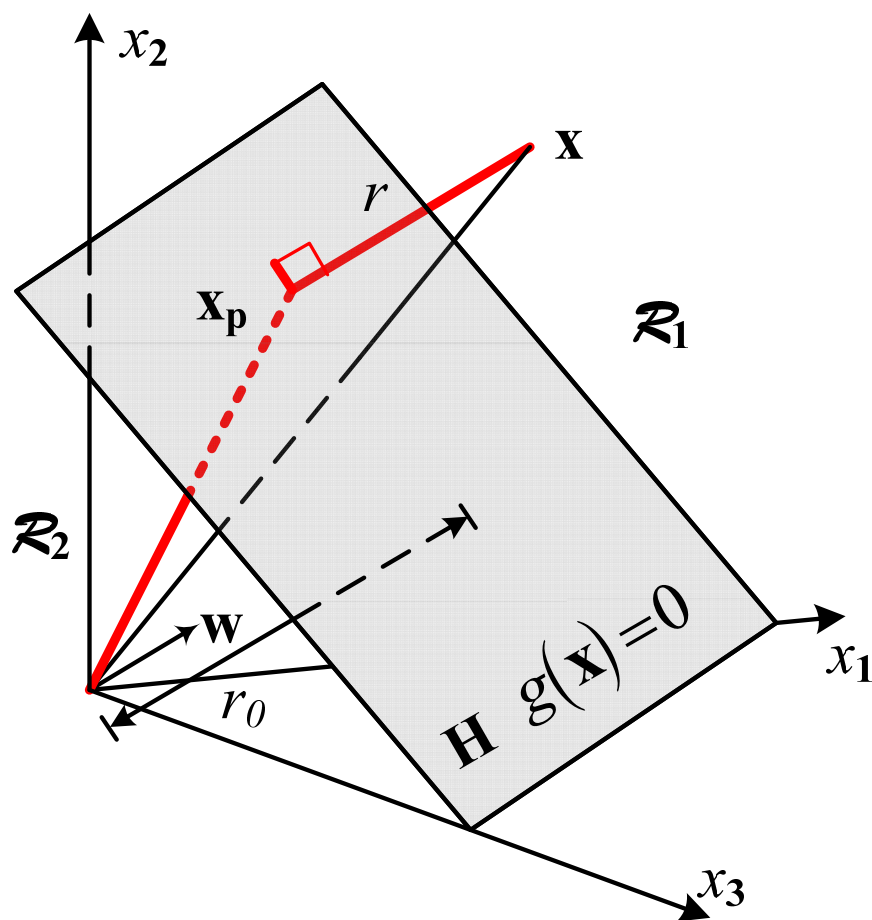
$$g(\mathbf{x}) \begin{cases} > 0, & \mathbf{x} \text{ 处于 } H \text{ 上方 } R_1 \text{ 区域} \\ = 0, & \mathbf{x} \text{ 在 } H \text{ 上} \\ < 0, & \mathbf{x} \text{ 处于 } H \text{ 下方 } R_2 \text{ 区域} \end{cases}$$

➤ 权矢量 \mathbf{w} 垂直于分类面 H , 指向 R_1 区域

1 线性判别函数和线性分类界面

□ 线性判别函数——断言

➤ 点到平面的距离



平面 H 方程： $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 0$

点到平面 H 距离： $r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$

原点到平面距离：

$$r_0 = \frac{g(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{0} + w_0}{\|\mathbf{w}\|} = \frac{w_0}{\|\mathbf{w}\|}$$

1 线性判别函数和线性分类界面

□ 两类问题线性判别准则

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 \begin{cases} > 0, & \mathbf{x} \in \omega_1 \\ < 0, & \mathbf{x} \in \omega_2 \\ = 0, & \text{拒识} \end{cases}$$

已知一组训练样本,
如何计算出合适的
 \mathbf{w} , w_0 ?

——分类器的
学习

1. 线性分类界面H是d维空间中的一个超平面;
2. 分类界面将d维空间分成两部分, R_1 , R_2 分别属于两个类别;
3. 判别函数的权矢量 \mathbf{w} 是一个垂直于分类界面H的矢量, 其方向指向区域 R_1 ;
4. 偏置 w_0 与原点 to 分类界面H的距离有关: $r_0 = \frac{w_0}{\|\mathbf{w}\|}$

2感知器算法

□线性判别函数分类器的学习

- 现有属于两个类别的训练样本集 $D=\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{1n}\}$ 分别标记为 ω_1, ω_2 用这些样本来确定一个判别函数 $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$ 的权矢量 \mathbf{w} 及偏置 w_0 。
- 在线性可分的情况下，希望得到的判别函数能够将所有的训练样本正确分类：

$$\begin{cases} \mathbf{w}^T \mathbf{x} + w_0 > 0, & \forall \mathbf{x} \in \omega_1 \\ \mathbf{w}^T \mathbf{x} + w_0 < 0, & \forall \mathbf{x} \in \omega_2 \end{cases}$$

- 线性不可分的情况下，判别函数产生错误的概率最小。

2感知器算法

	线性判别函数	线性判别函数的增广形式
函数形式	$g(\mathbf{x}) = (w_1, w_2, \dots, w_d) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix} + w_0$	$g(\mathbf{x}) = (w_0, w_1, w_2, \dots, w_d) \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}$
	$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$	增广权矢量: $\mathbf{a} = [\mathbf{w}^T, w_0]^T$
		$g(\mathbf{y}) = \mathbf{a}^T \mathbf{y}_0$
判别规则	$g(\mathbf{x}) = \begin{cases} > 0, & \mathbf{x} \in \omega_1 \\ < 0, & \mathbf{x} \in \omega_2 \\ = 0, & \text{拒识} \end{cases}$	样本规范化: $\begin{cases} \mathbf{y} = [\mathbf{x}^T, 1]^T, & \forall \mathbf{x} \in \omega_1 \\ \mathbf{y} = [-\mathbf{x}^T, -1]^T, & \forall \mathbf{x} \in \omega_2 \end{cases}$
		统一形式: $\mathbf{a}^T \mathbf{y}_i > 0, \quad i = 1, \dots, n$

线性判别函数的增广形式

$$g(\mathbf{y}) = \mathbf{a}^T \mathbf{y}$$

- ▣ $\mathbf{y} = (1, x_1, x_2, \dots, x_d)^T$: 增广的特征矢量;
- ▣ $\mathbf{a} = (w_0, w_1, w_2, \dots, w_d)^T$: 增广的权矢量;
- ▣ 在线性可分的情况下, 希望得到的判别函数能够将所有的训练样本 \mathbf{y}_i 正确分类:

$$\mathbf{a}^T \mathbf{y}_i > 0, \quad i = 1, \dots, n$$

不等式组的求解?

不等式组求解方法—梯度下降法

□求解不等式组采用的最优化的方法：

- 定义一个准则函数 $J(\mathbf{a})$ ，当 \mathbf{a} 是解向量时， $J(\mathbf{a})$ 为最小；
- 采用最优化方法求解标量函数 $J(\mathbf{a})$ 的极小值。

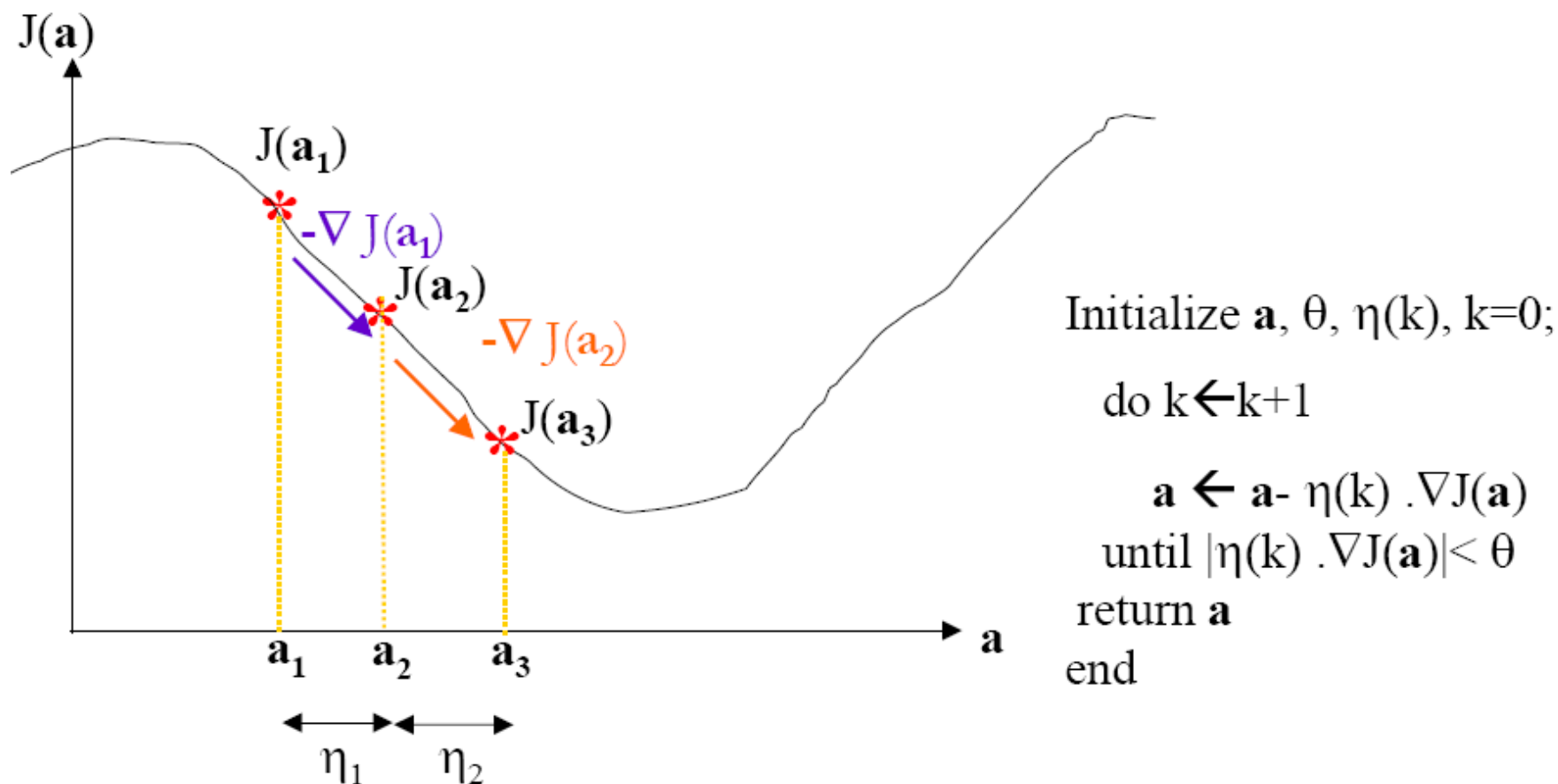
□最优化方法采用最多的是梯度下降法，

- 设定初始权值矢量 $\mathbf{a}(1)$ ，然后沿梯度的负方向迭代计算：

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \nabla J(\mathbf{a}(k))$$

其中 $\eta(k)$ 称为学习率，或称步长。

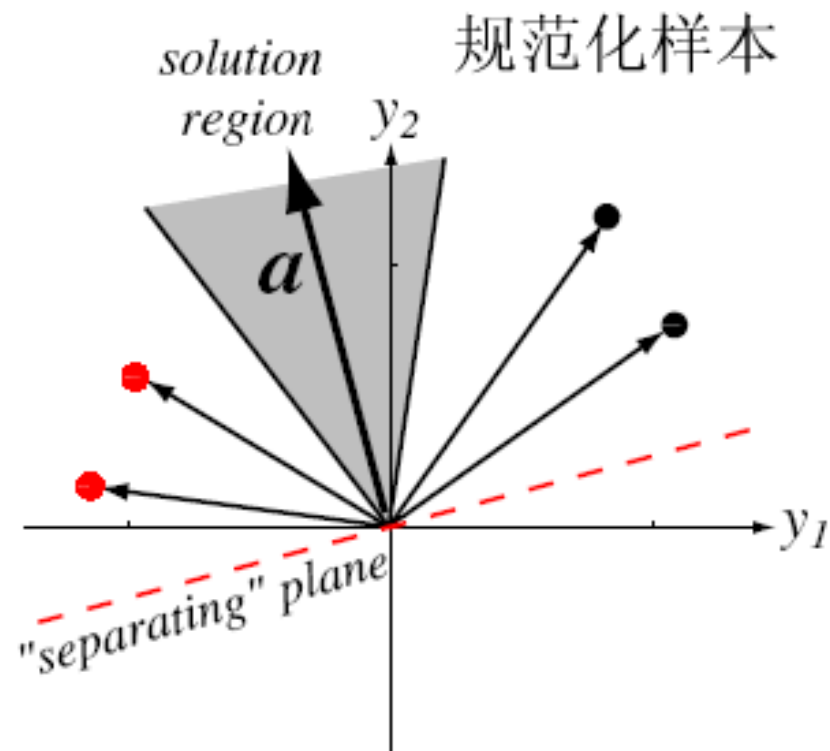
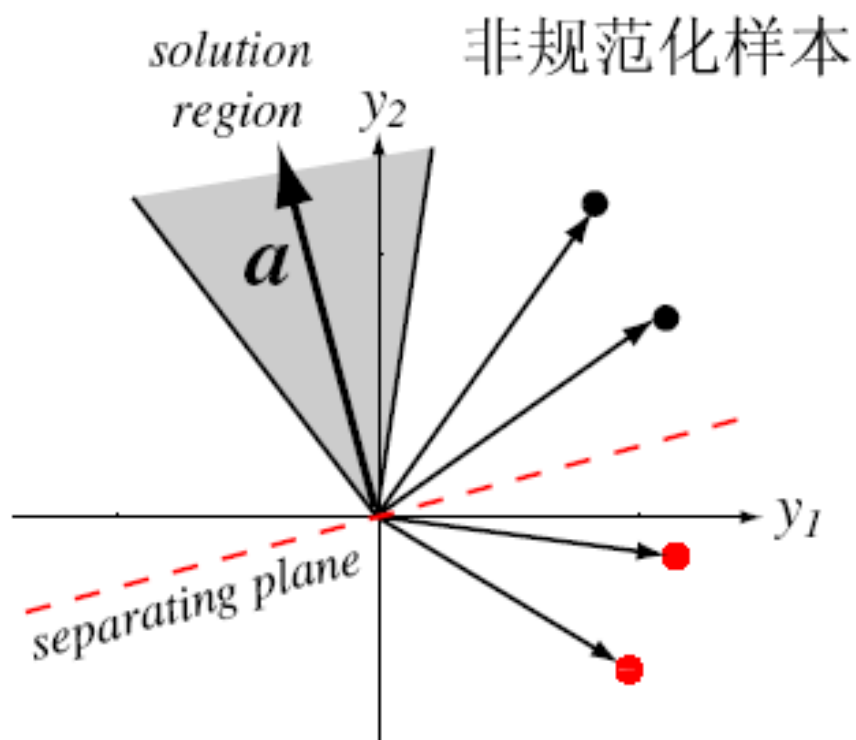
不等式组求解方法—梯度下降法



$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \nabla J(\mathbf{a}(k))$$

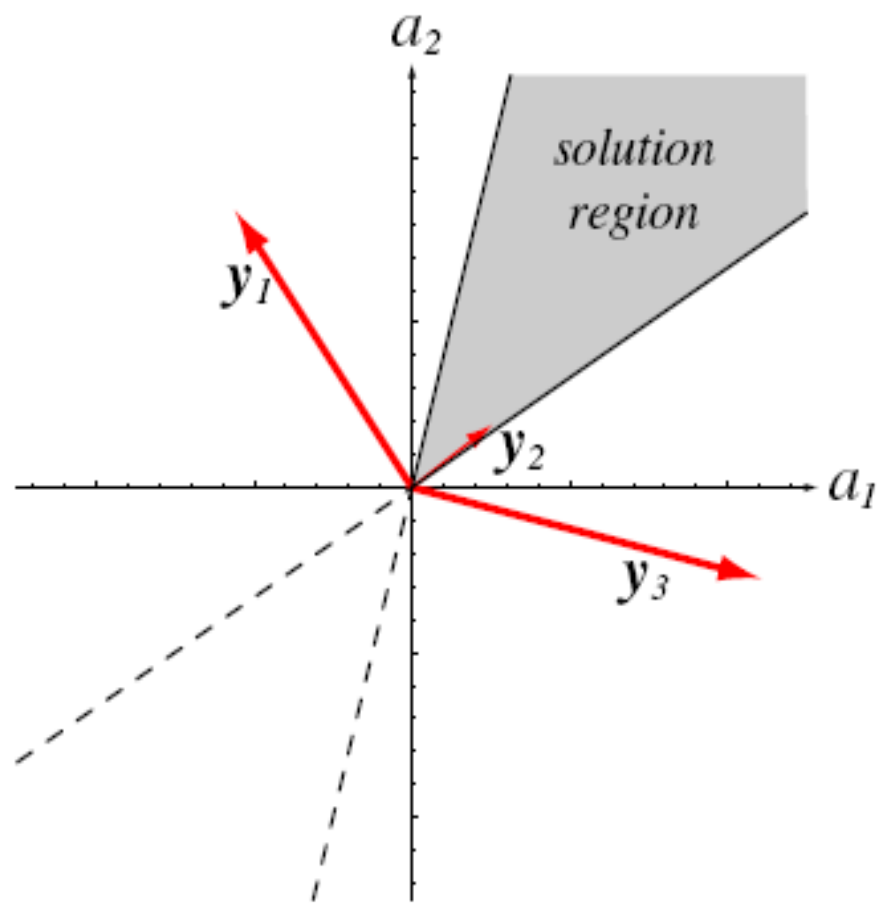
解区域的几何解释(特征空间中)

- **特征空间中：** 矢量 a 是垂直于分类界面的矢量：



解区域的几何解释(权空间中)

权空间中： $\mathbf{a}^T \mathbf{y}_i = 0$ 是一个通过原点的超平面， \mathbf{y}_i 是法向量， \mathbf{a} 是空间中一个点。

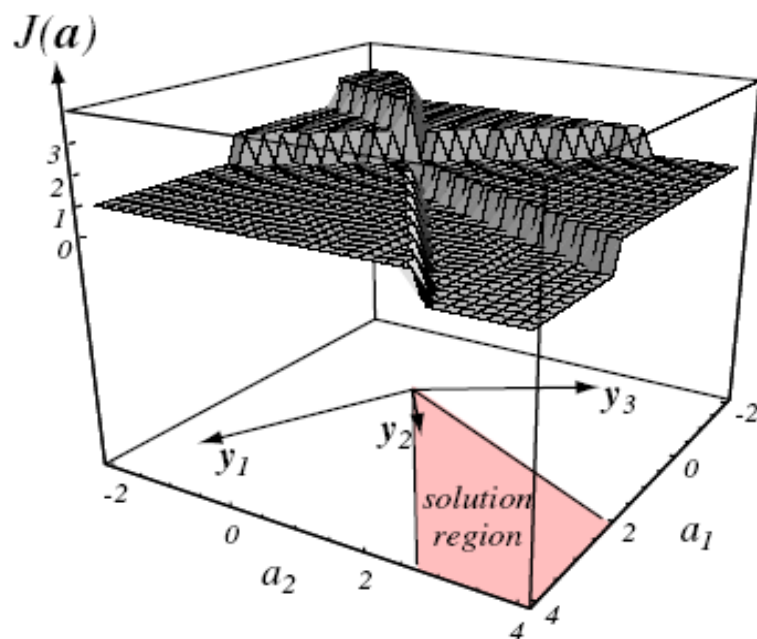


2感知器算法

➤最少错分样本数准则:

$$J_N(\mathbf{a}) = \sum_{\mathbf{y} \in Y} 1$$

不连续、梯度为0
无法进行迭代优化!

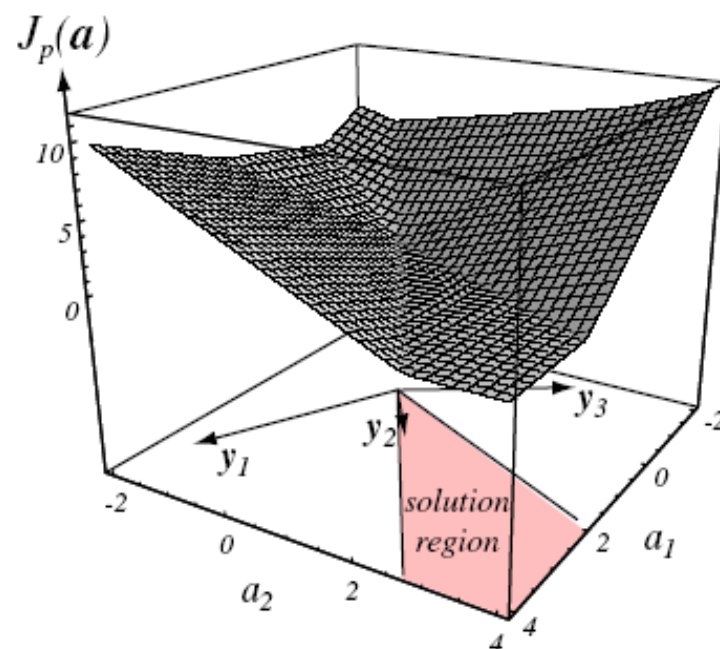


➤感知器准则: 错分样本到分类界面“距离”之和

$$J_P(\mathbf{a}) = \sum_{\mathbf{y} \in Y} (-\mathbf{a}^T \mathbf{y})$$

$$\nabla J_P = \sum_{\mathbf{y} \in Y} (-\mathbf{y})$$

连续、
梯度不为0
可迭代优化!



感知器算法(批量调整版本)

1. begin initialize $\mathbf{a}(0)$, $\eta(\cdot)$, θ , $k \leftarrow 0$
2. do $k \leftarrow k+1$
3.
$$\mathbf{a}(k+1) \leftarrow \mathbf{a}(k) + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$$
4. until $\left| \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} \right| < \theta$
5. return \mathbf{a}
6. end

感知器算法(单样本调整版本)

1. begin initialize $\mathbf{a}(0)$, $k \leftarrow 0$
2. do $k \leftarrow (k+1) \bmod n$
3. if y_k is misclassified by \mathbf{a} then
$$\mathbf{a}(k+1) \leftarrow \mathbf{a}(k) + \mathbf{y}_k$$
4. until all patterns properly classified
5. return \mathbf{a}
6. end

作业:

□ 有两类模式的训练样本:

$$\omega_1: \{ (0,0), (0,1) \}$$

$$\omega_2: \{ (1,0), (1,1) \}$$

- 用感知器算法求取判别函数，将两类样本分开，
- 初始权矢量为 $\mathbf{a}(0) = (-2, 0, -1)^T$ ，第3维为偏置，
- 学习率 $\eta = 1$

- 五十年代Rosenblatt提出的一种自学习判别函数生成方法，
- 企图将其用于脑模型感知器，因此被称为感知准则函数。
- 随意确定的判别函数初始值，在对样本分类训练过程中逐步修正直至最终确定。

感知器算法收敛证明

- 如果训练样本线性可分，固定增量算法给出的权向量序列必定终止于某个解向量
 - 每次校正，都使权向量更靠近解区域
 - 如果 $\hat{\mathbf{a}}$ 是解向量，随着训练样本的增加有：

$$\|\mathbf{a}(k+1) - \hat{\mathbf{a}}\| < \|\mathbf{a}(k) - \hat{\mathbf{a}}\|$$

感知器算法收敛定理

设 $\hat{\mathbf{a}}$ 是解向量则 $\hat{\mathbf{a}}^T \mathbf{y}_i > 0$, 令 α 为一比例因子, 有:

$$\begin{aligned}
 \|\mathbf{a}(k+1) - \alpha \hat{\mathbf{a}}\|^2 &= \|\mathbf{a}(k) + \mathbf{y}_k - \alpha \hat{\mathbf{a}}\|^2 \\
 &= \|\mathbf{a}(k) - \alpha \hat{\mathbf{a}}\|^2 + 2 \left(\underbrace{\mathbf{a}(k)^T \mathbf{y}_k}_{< 0} - \alpha \hat{\mathbf{a}}^T \mathbf{y}_k \right) + \|\mathbf{y}_k\|^2 \quad \mathbf{y}_k \text{ 为错分样本} \\
 &\leq \|\mathbf{a}(k) - \alpha \hat{\mathbf{a}}\|^2 - 2\alpha \underbrace{\hat{\mathbf{a}}^T \mathbf{y}_k}_{> 0} + \|\mathbf{y}_k\|^2 \quad \beta^2 = \max_i \|\mathbf{y}_i\|^2 \\
 &\leq \|\mathbf{a}(k) - \alpha \hat{\mathbf{a}}\|^2 - 2\alpha \gamma + \underbrace{\beta^2}_{\text{设 } \alpha = \beta^2 / \gamma} \quad \gamma = \min_i [\hat{\mathbf{a}}^T \mathbf{y}_i] > 0 \\
 &\leq \|\mathbf{a}(k) - \alpha \hat{\mathbf{a}}\|^2 - \beta^2
 \end{aligned}$$

每次校正后, 从 $\mathbf{a}(k+1)$ 到 $\alpha \hat{\mathbf{a}}$ 的平方距离减少了 β^2

感知器算法收敛定理

每次校正后，从 $\mathbf{a}(k+1)$ 到 $\alpha\hat{\mathbf{a}}$ 的平方距离减少了 β^2

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(k) - \hat{\mathbf{a}}\|^2 - \beta^2$$

经过k步校正后

$$\|\mathbf{a}(k+1) - \alpha\hat{\mathbf{a}}\|^2 \leq \|\mathbf{a}(1) - \hat{\mathbf{a}}\|^2 - k\beta^2$$

$$\alpha = \beta^2 / \gamma$$

$$\beta^2 = \max_i \|\mathbf{y}_i\|$$

经过不超过 k_0 步校正后，校正终止

$$\gamma = \min_i [\hat{\mathbf{a}}^t \mathbf{y}_i] > 0$$

$$k_0 = \|\mathbf{a}(1) - \alpha\hat{\mathbf{a}}\|^2 / \beta^2 \quad \text{if : } \mathbf{a}(1) = 0$$

$$= \|\alpha\hat{\mathbf{a}}\|^2 / \beta^2 = \alpha^2 \|\hat{\mathbf{a}}\|^2 / \beta^2 = \frac{\|\hat{\mathbf{a}}\|^2 \max_i \|\mathbf{y}_i\|}{\min_i [\hat{\mathbf{a}}^t \mathbf{y}_i]}$$

与解向量最接近正交的样本

(收敛的难点)

感知器算法的特点

□ 学习率的选择

当样本线性可分情况下，学习率 $\eta(\cdot)$ 合适时，算法具有收敛性；

□ 收敛速度

$$k_0 = \frac{\|\hat{\mathbf{a}}\|^2 \max_i \|\mathbf{y}_i\|}{\min_i [\hat{\mathbf{a}}^t \mathbf{y}_i]}$$

□ 线性不可分的训练样本集

当样本线性不可分情况下，算法不收敛，且无法判断样本是否线性可分。

感知器算法推广

错误分类点数: $J_{mis}(\mathbf{a}) = \sum_{\mathbf{y} \in Y} (1)$

感知器判据: $J_p(\mathbf{a}) = \sum_{\mathbf{y} \in Y} -\mathbf{a}^T \mathbf{y}$

平方误差判据: $J_q(\mathbf{a}) = \sum_{\mathbf{y} \in Y} (\mathbf{a}^T \mathbf{y})^2$

规范化平方（裕量）误差判据:

$$J_r(\mathbf{a}) = \sum_{\mathbf{y} \in Y} \frac{(\mathbf{a}^T \mathbf{y} - b)^2}{\|\mathbf{y}\|^2}$$

不连续



梯度不连续

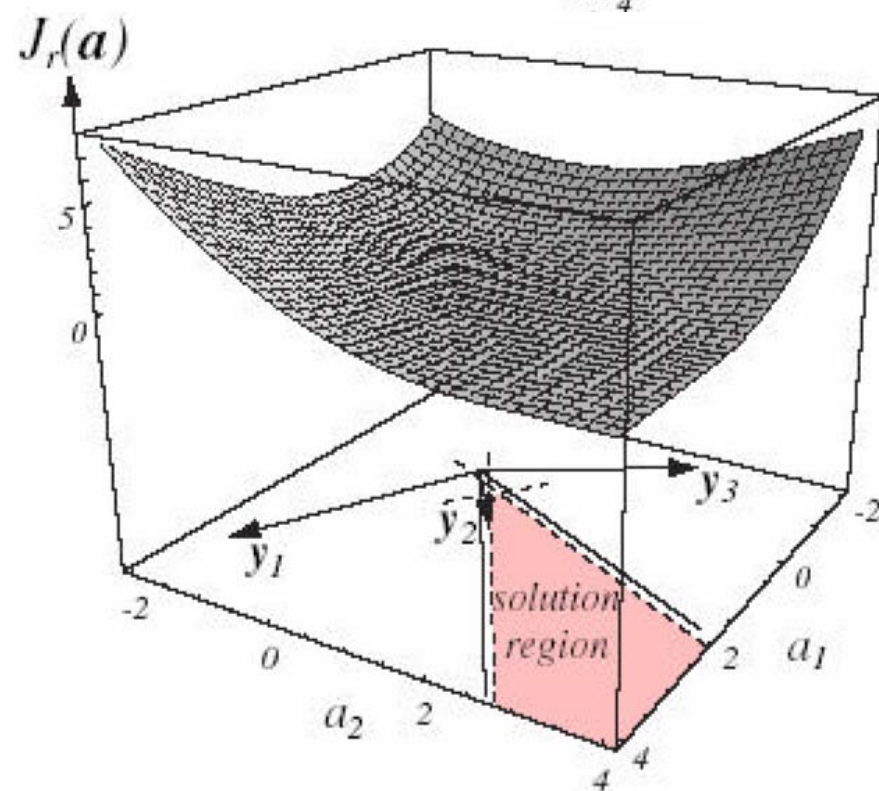
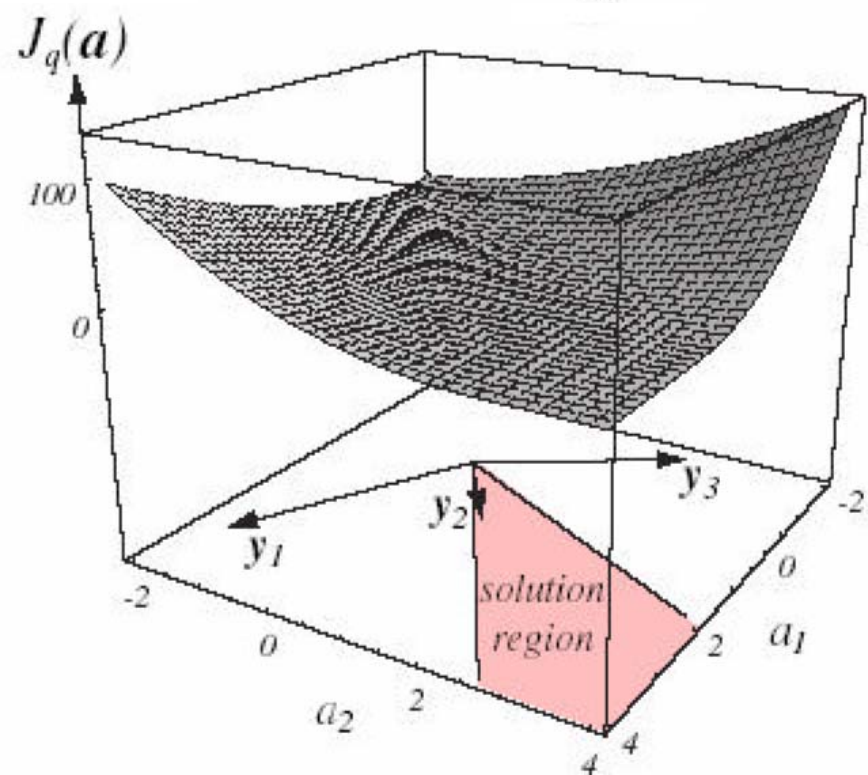
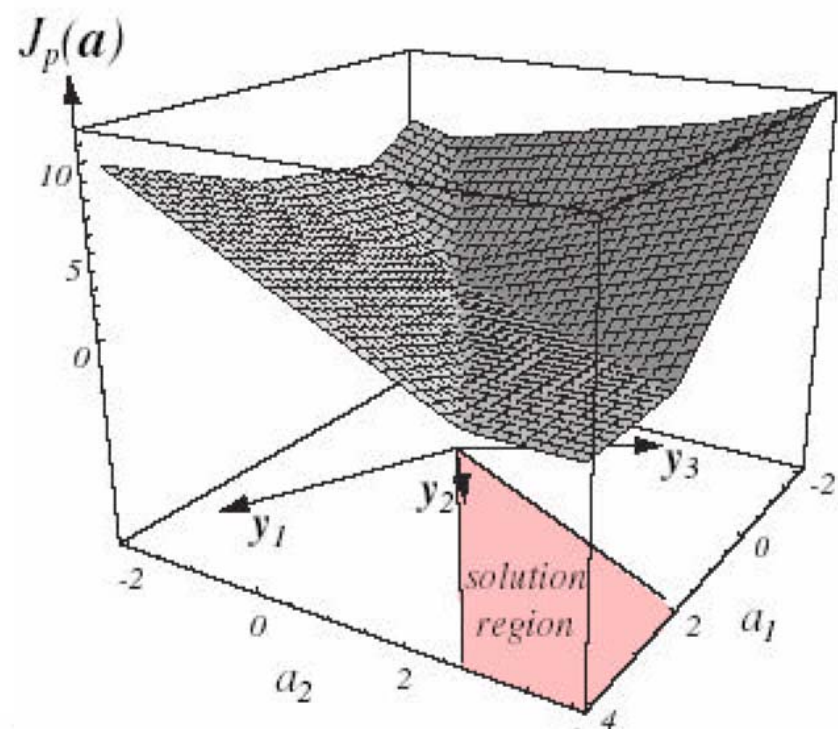
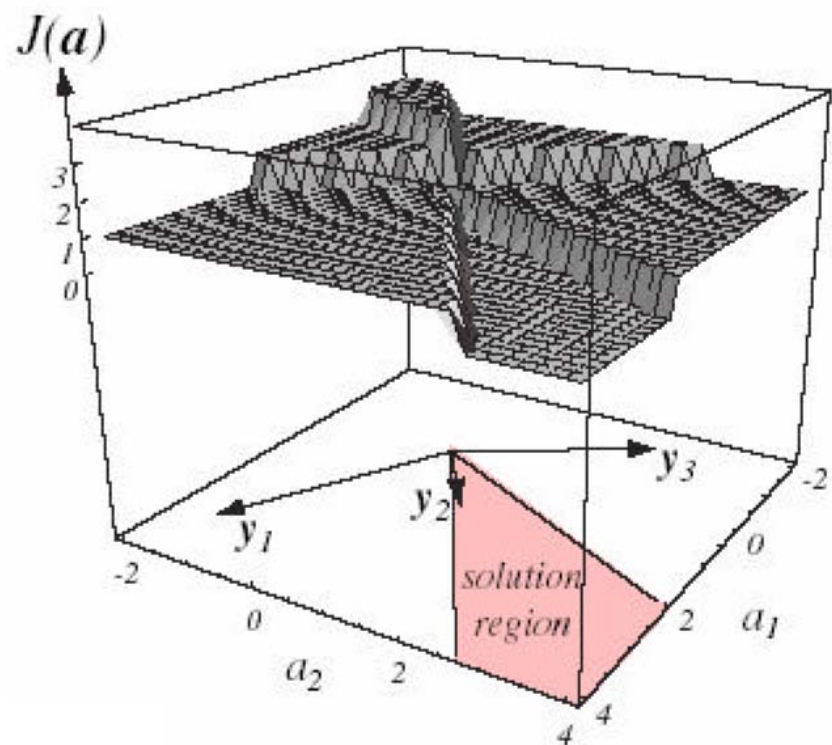


梯度连续



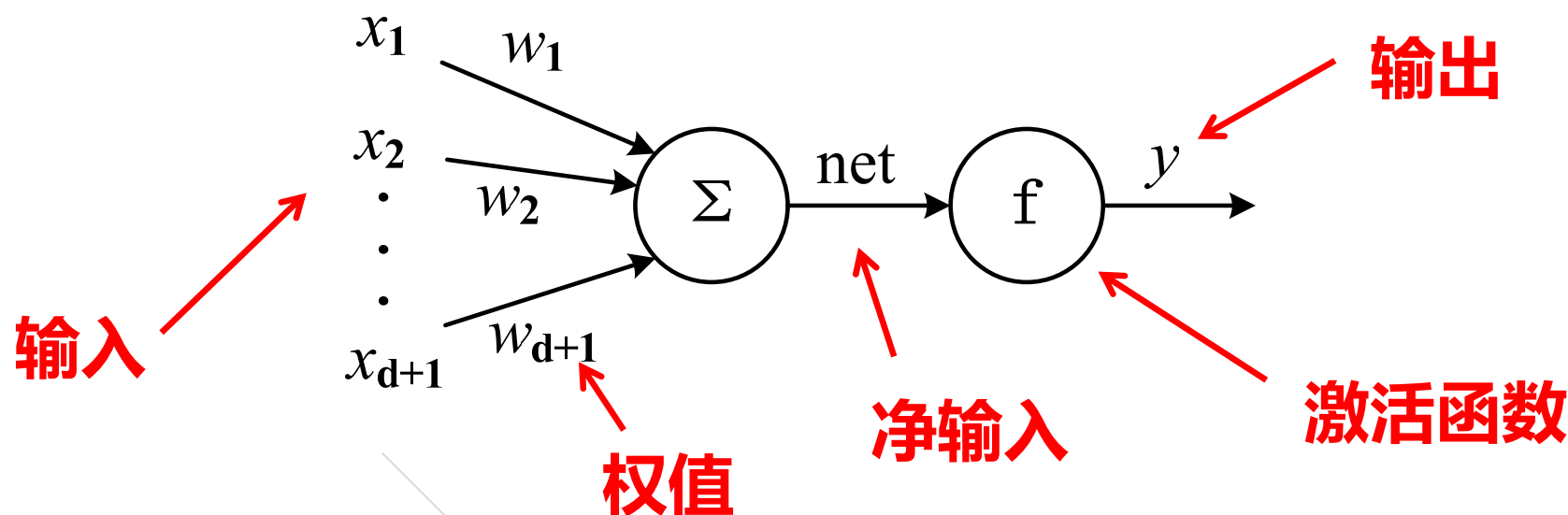
克服边界点及
 $\|\mathbf{y}\|$ 的影响

松弛算法



3 感知器网络

□1943年McCulloch和Pitt提出了人工神经元模型

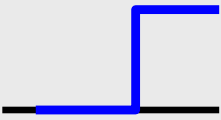
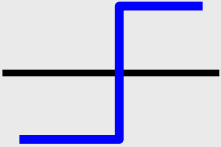
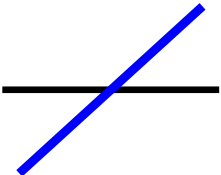
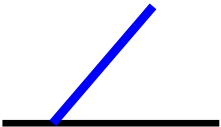
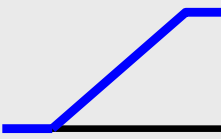
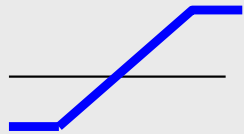
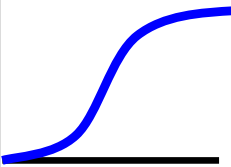
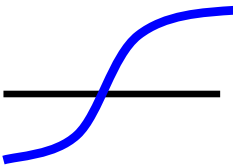
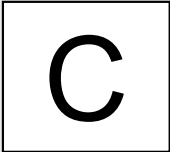


$$net = \sum_{i=1}^{d+1} x_i w_i = \mathbf{w}^t \mathbf{x} \quad y = \underline{f(net)} = f(\mathbf{w}^t \mathbf{x})$$

当 f 为符号函数时，“神经元”等价于“线性判别函数”

$$f(net) = \begin{cases} -1, & net < 0 \\ +1, & net \geq 0 \end{cases}$$

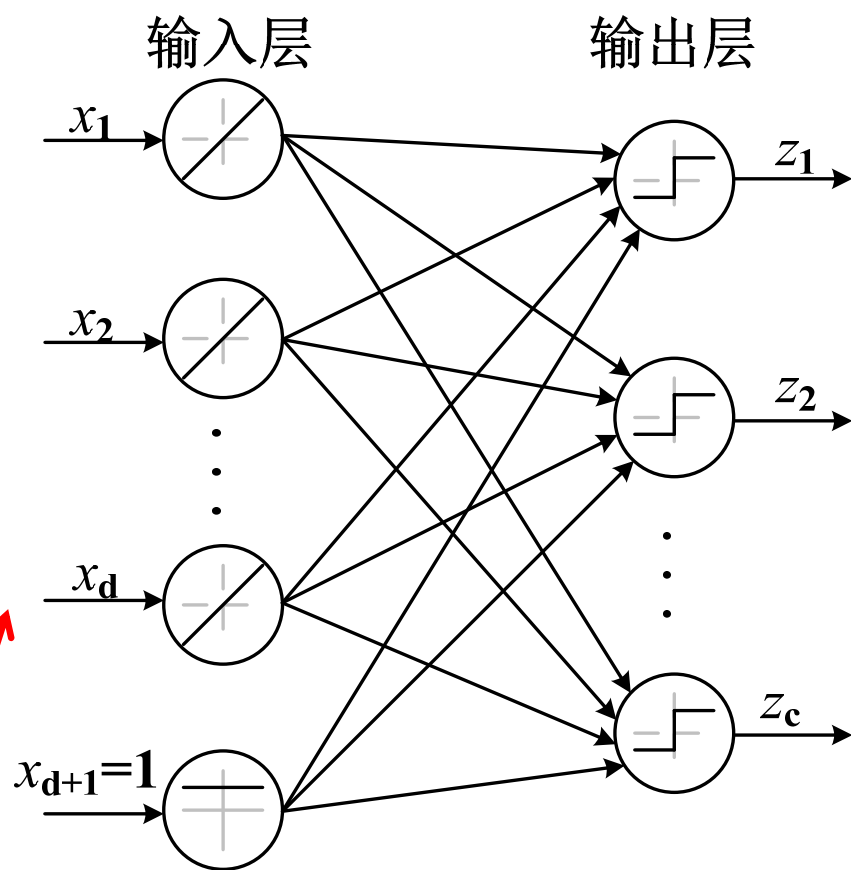
激活函数（传输函数）

名称	输入 n /输出 y	图标	名称	输入 n /输出 y	图标
硬极限	$y = 0, n < 0$ $y = 1, n \geq 0$		对称硬极限	$y = -1, n < 0$ $y = 1, n \geq 0$	
线性	$y = n$		正线性	$y = 0, n < 0$ $y = n, n \geq 0$	
饱和线性	$y = 0, n < 0$ $y = n, 0 \leq n < 1$ $y = 1, n \geq 1$		对称饱和线性	$y = 0, n < 0$ $y = n, 0 \leq n < 1$ $y = 1, n \geq 1$	
对数-S型	$y = \frac{1}{1 + e^{-n}}$		双曲正切S型	$y = \frac{e^n - e^{-n}}{e^n + e^{-n}}$	
竞争	$y = 1$, 具有最大 n 的神经元 $y = 0$, 所有其他神经元				

3 感知器网络

□ 两层感知器网络（线性网络）

➤ 需要解决多个类别分类问题，由多个神经元构成一个神经网络



输出：可用多种方式

- 1) 直接编码
- 2) 进制编码
- 3) 最大值

学习：网络结构、激活函数已确定，如何学习连接权值 w ？

——最小平方误差法

输入：d+1维增广特征矢量

3 感知器网络

□ 两层感知器网络的学习

- 将所有的训练样本写成“样本矩阵”，以及对应的“期望输出矩阵”

$$\mathbf{Y} = \begin{pmatrix} x_{11} & \cdots & x_{1d} & 1 \\ x_{12} & \cdots & x_{2d} & 1 \\ \vdots & \ddots & \vdots & \vdots \\ x_{n1} & \cdots & x_{nd} & 1 \end{pmatrix} \quad \mathbf{T} = \begin{pmatrix} t_{11} & \cdots & t_{1c} \\ \vdots & \ddots & \vdots \\ t_{n1} & \cdots & t_{nc} \end{pmatrix}$$

- 输出层神经元权值写成矩阵形式，每列对应一个神经元的权值矢量

$$\mathbf{W} = \begin{pmatrix} w_{11} & \cdots & w_{c1} \\ \vdots & \ddots & \vdots \\ w_{1d+1} & \cdots & w_{cd+1} \end{pmatrix}$$

- 构建矩阵方程 $\mathbf{YW} = \mathbf{T}$ ，采用最小平方误差法求近似解。

伪逆法
梯度下降法

3 感知器网络

□ 两层感知器网络的学习

目标函数		$\mathbf{Y}\mathbf{W} = \mathbf{T}$
最小平方误差代价函数		$\min_{\mathbf{W}} J(\mathbf{W}) = \ \mathbf{Y}\mathbf{W} - \mathbf{T}\ _F^2$ $\ \cdot\ _F \text{ 矩阵的Frobenius范数 } \ \mathbf{A}\ _F = \sqrt{\sum_{i,j} a_{ij}^2}$
代价函数求导		$\nabla J(\mathbf{W}) = \mathbf{Y}^t (\mathbf{Y}\mathbf{W} - \mathbf{T})$
求解	伪逆法	$\nabla J(\mathbf{W}) = 0 \rightarrow \mathbf{W} = (\mathbf{Y}^t \mathbf{Y})^{-1} \mathbf{Y}^t \mathbf{T} = \mathbf{Y}^+ \mathbf{T}$
	梯度下降法	$\mathbf{W}(k+1) = \mathbf{W}(k) - \eta(k) \mathbf{Y}^t [\mathbf{Y}\mathbf{W}(k) - \mathbf{T}]$

4 多层感知器网络

多层感知器网络（MLP, Multi-Layer Perception）

误差反向传播算法（BP, Backpropagation）

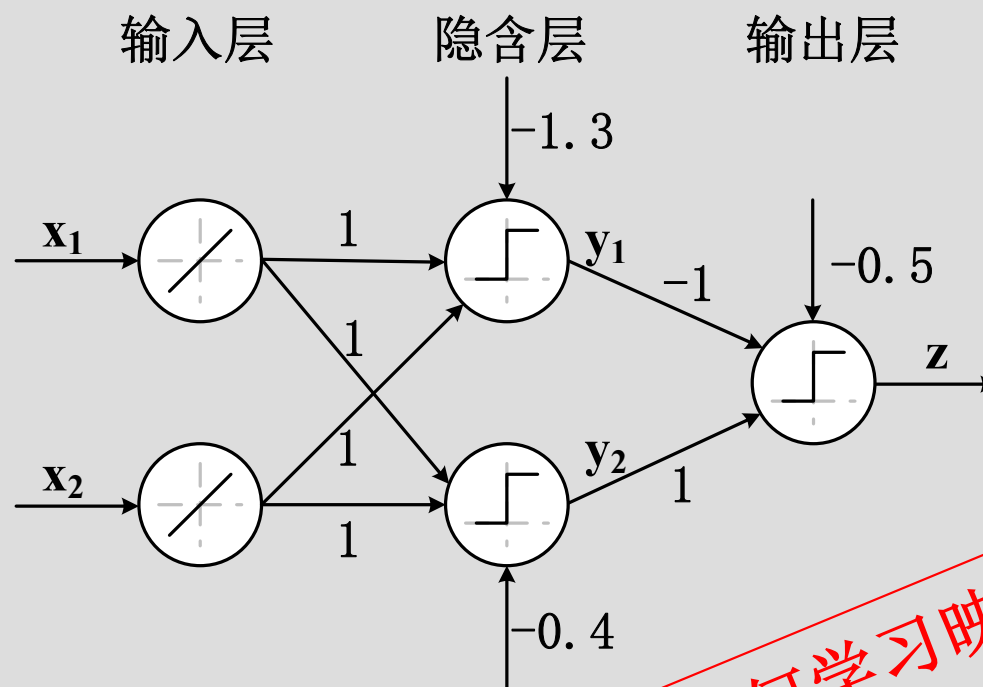
优化方法

实践

4.1 解决XOR问题的多层感知器

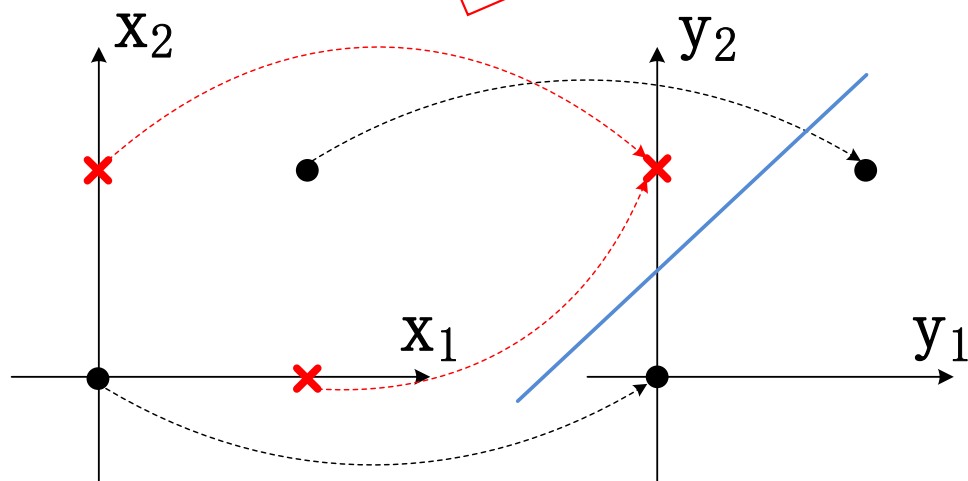
网络结构:

输入层采用线性激活函数，
隐含层和输出层节点采用
0-1阶跃激活函数



如何设计通用
的网络结构?

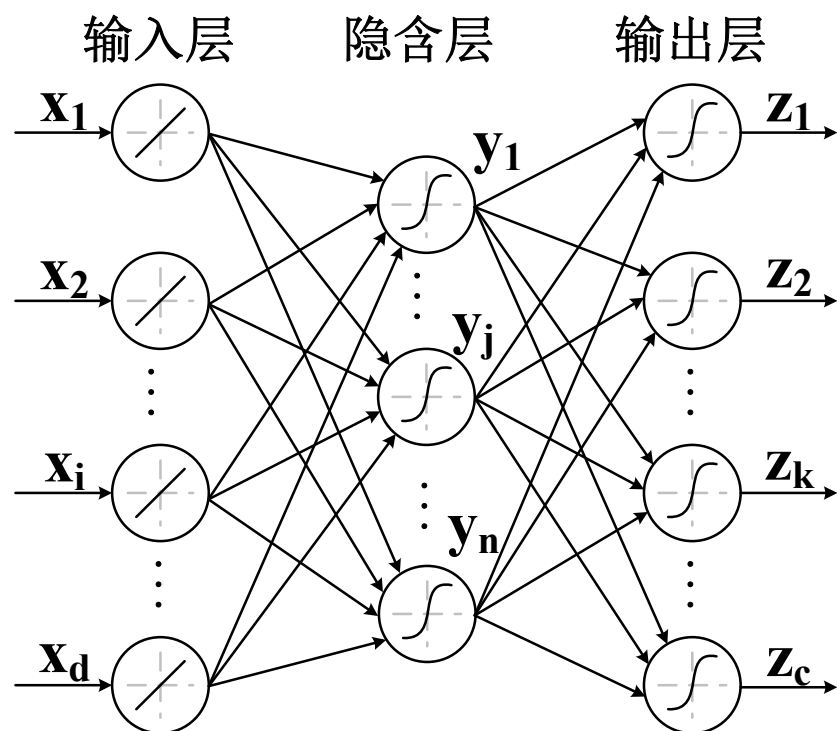
如何学习映
射参数?



非线性映射:

隐含层对原特征进行非线性映射，输出层进行线性判别。

4.2 多层感知器的结构



选定层数：通常采用三层网络，增加网络层数并不能提高网络的分类能力；

输入层：输入层节点数为输入特征的维数 d ，映射函数采用线性函数；

隐含层：隐含层节点数需要设定，一般来说，隐层节点数越多，网络的分类能力越强，映射函数一般采用 Sigmoid 函数；

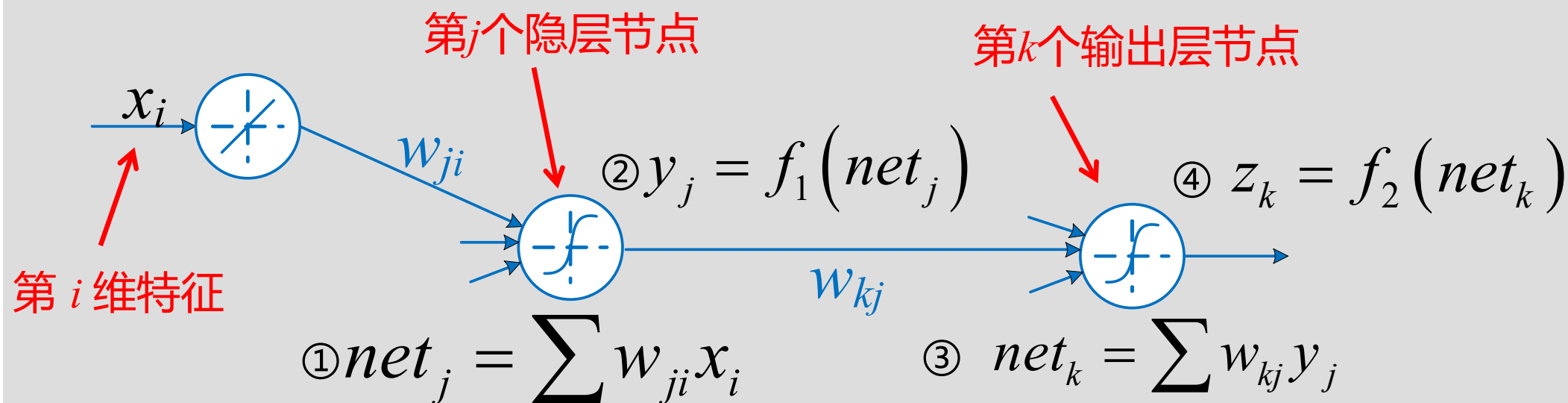
Sigmoid 型激活函数：

对数型
$$f(u) = \frac{1}{1 + e^{-u}}$$

双曲正切型
$$f(u) = \frac{e^u - e^{-u}}{e^u + e^{-u}}$$

输出层：输出层节点数可以等于类别数 c ，也可以采用编码输出的方式，少于类别数 c ，输出函数可以采用线性函数或 Sigmoid 函数。

三层网络的传输过程



输出层第 k 节点输出:

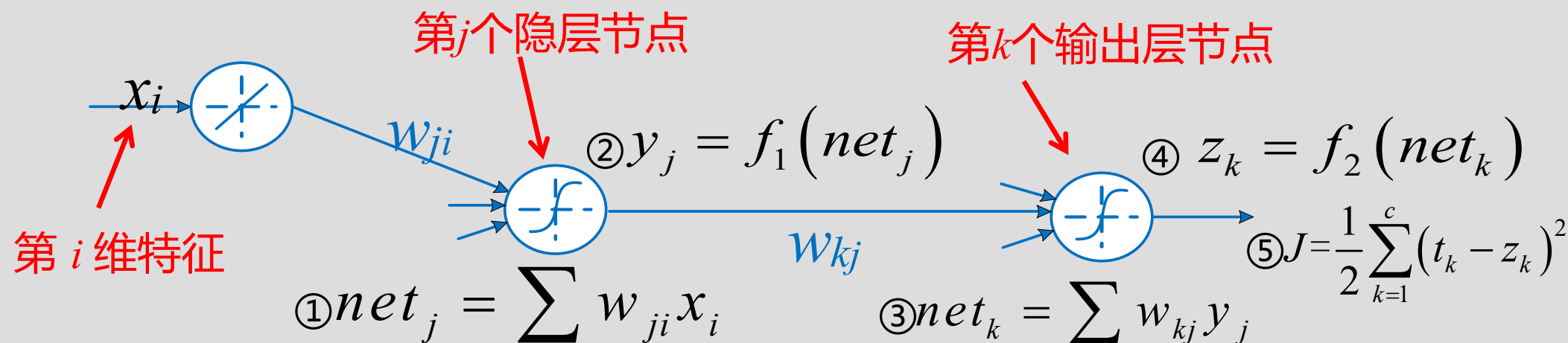
$$z_k(\mathbf{x}) = f_2 \left(\sum_{j=1}^{n_H} w_{kj} \underbrace{f_1 \left(\sum_{i=1}^d w_{ji} x_i + w_{j0} \right)}_{y_j} + w_{k0} \right)$$

Annotations in the diagram:

- 共 n_H 个隐层节点 (Total n_H hidden nodes) points to the summation index j .
- 共 d 维特征 (Total d dimensions of features) points to the summation index i .
- 第 j 隐节点偏置 (Bias of the j -th hidden node) points to w_{j0} .
- 第 k 输出节点偏置 (Bias of the k -th output node) points to w_{k0} .

哈尔滨工业大学 计算机学院
模式识别与智能系统研究中心
ISBN 978-7-5603-4763-9

误差反向传播算法 (BP, Backpropagation algorithm)



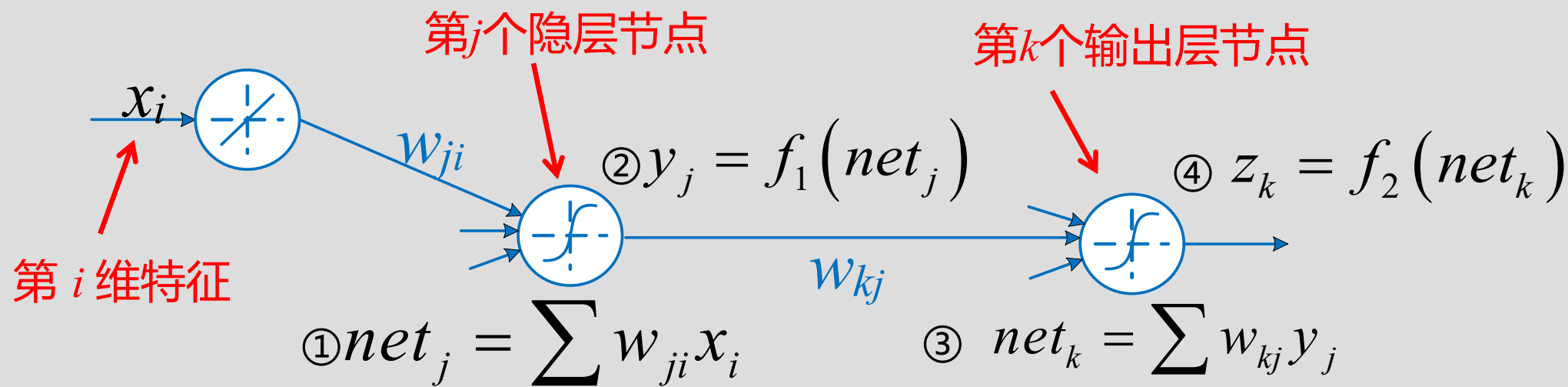
BP算法：训练样本 \mathbf{x} ，期望输出 \mathbf{t} ，网络实际输出 \mathbf{z} ，基于误差最小化的梯度下降法，求解连接权值 $\mathbf{w} = \{w_{ji}, w_{kj}\}$

目标（误差）函数： $J(\mathbf{w}) = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2 = \frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2$

梯度下降： $\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m) = \mathbf{w}(m) - \eta \frac{\partial J}{\partial \mathbf{w}}$

如何求导？

误差反向传播——链式求导



链式求导：计算误差函数对“隐含—输出”连接权值 w_{kj} ，“输入—隐含层”连接权值 w_{ji} 的导数

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

——误差函数 J 沿着公式⑤-④-③-②-①反向传播

BP算法基本思路

1) 为了使目标函数 $J(\mathbf{w}) = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2$ 最小化, 需要采用梯度下降法搜索最佳 \mathbf{w} :

$$\mathbf{w}(m+1) = \mathbf{w}(m) - \eta \frac{\partial J}{\partial \mathbf{w}}$$

2) 为了采用梯度下降法, 需要计算导数: $\frac{\partial J}{\partial \mathbf{w}}$

3) \mathbf{w} 分为两部分, 采用链式法则求导:

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

Sigmoid激活函数求导

对数Sigmoid 函数: $f(x) = \frac{1}{1 + e^{-x}}$

导数: $f'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \frac{1}{1 + e^{-x}} - \frac{1}{(1 + e^{-x})^2} = f(x)(1 - f(x))$

应用该激活函数时，输出层及隐含层的导数形式为：

输出层: $f'(net_k) = f(net_k)(1 - f(net_k)) = z_k(1 - z_k)$

隐含层: $f'(net_j) = f(net_j)(1 - f(net_j)) = y_j(1 - y_j)$

Sigmoid激活函数

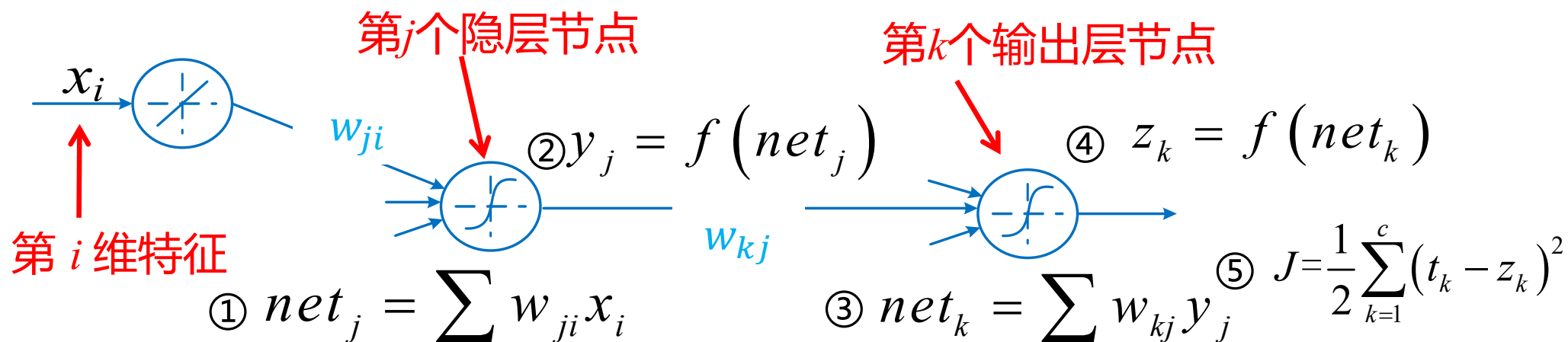
- 双曲Sigmoid激活函数

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

请证明应用该激活函数时，输出层及隐含层的导数形式为

输出层: $f'(net_k) = 1 - f^2(net_k) = 1 - z_k^2$

隐含层: $f'(net_k) = 1 - f^2(net_j) = 1 - y_j^2$



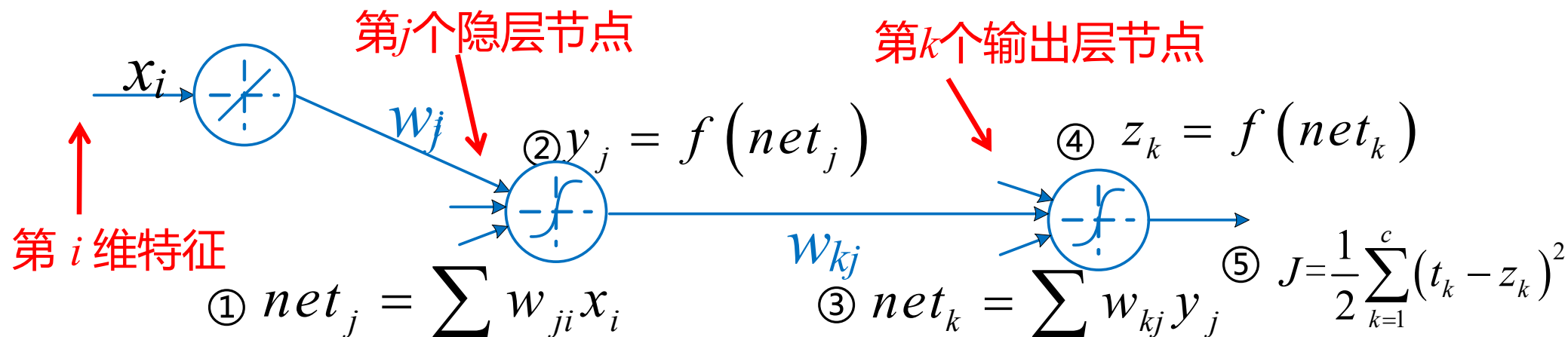
输出层求导：

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

由⑤得： $\frac{\partial J}{\partial z_k} = -(t_k - z_k)$ 由④得： $\frac{\partial z_k}{\partial net_k} = f'(net_k)$ 由③得： $\frac{\partial net_k}{\partial w_{kj}} = y_j$

$$\frac{\partial J}{\partial w_{kj}} = -(t_k - z_k) f'(net_k) y_j = -\delta_k y_j$$

令 $\delta_k = (t_k - z_k) f'(net_k)$



隐含层

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \quad \xrightarrow{\text{由①得:}} \quad \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{m=1}^d w_{jm} x_m \right) = x_i$$

$$\xrightarrow{\text{由②得:}} \quad \frac{\partial y_j}{\partial net_j} = f'(net_j)$$

$$\begin{aligned} \frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j} \left[\frac{1}{2} \sum_{k=1}^c (t_k - z_k)^2 \right] = - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial y_j} \\ &= - \sum_{k=1}^c (t_k - z_k) \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial y_j} = - \sum_{k=1}^c \underbrace{(t_k - z_k) f'(net_k)}_{\delta_k} w_{kj} = - \sum_{k=1}^c \delta_k w_{kj} \end{aligned}$$

隐含层 $\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$

$$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^c \delta_k w_{kj} \quad \frac{\partial y_j}{\partial net_j} = f'(net_j) \quad \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{m=1}^d w_{jm} x_m \right) = x_i$$

$$\frac{\partial J}{\partial w_{ji}} = - \left[\sum_{k=1}^c \delta_k w_{kj} \right] f'(net_j) x_i = -\delta_j x_i$$

$$\text{令 } \delta_j = \left(\sum_{k=1}^c \delta_k w_{kj} \right) f'(net_j)$$

迭代公式

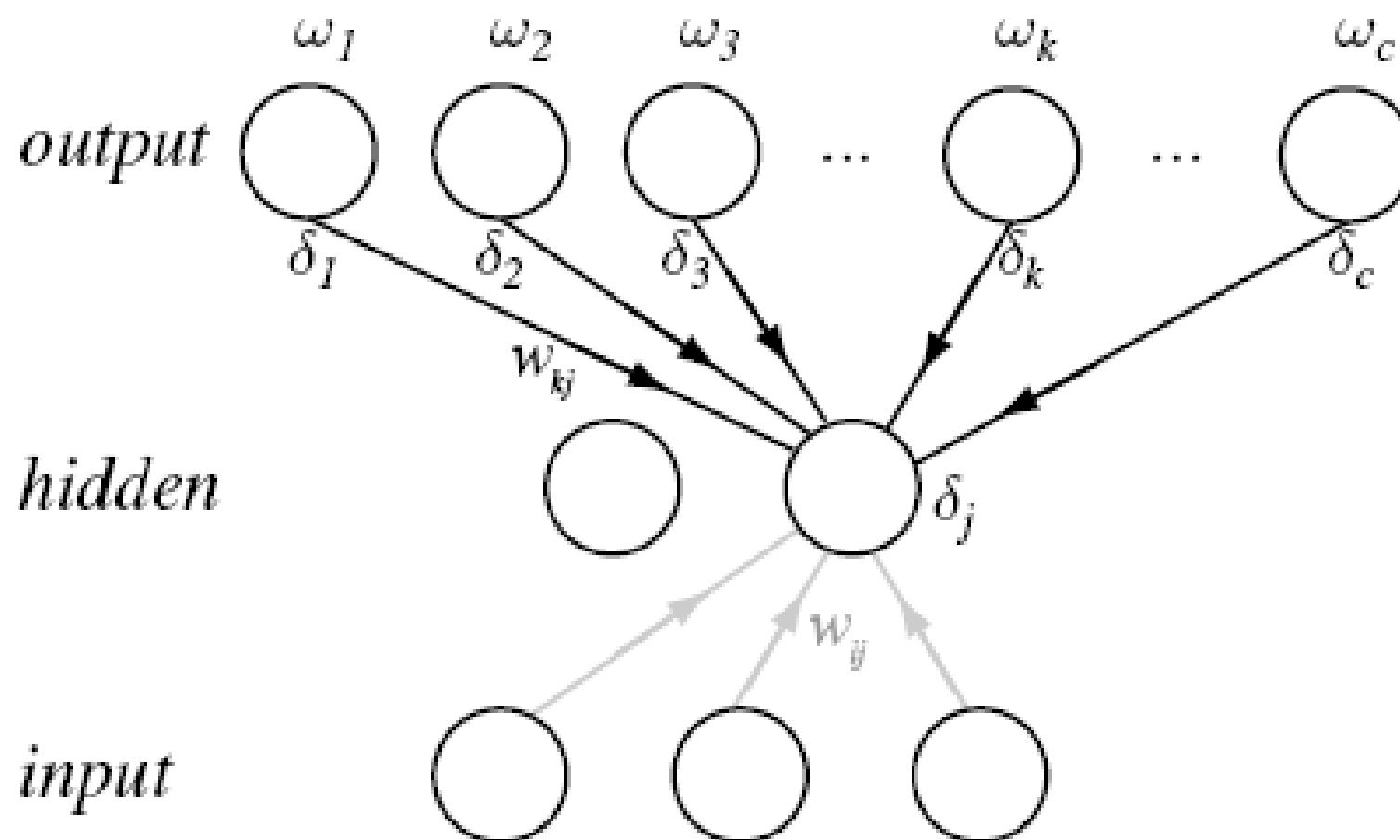
- 输出层: $\delta_k = (t_k - z_k) f'(net_k)$

$$\frac{\partial J}{\partial w_{kj}} = -\delta_k y_j,$$

- 隐含层: $\frac{\partial J}{\partial w_{ji}} = -\delta_j x_i,$

$$\delta_j = f'(net_j) \sum_{k=1}^c \delta_k w_{kj}$$

误差反向传播

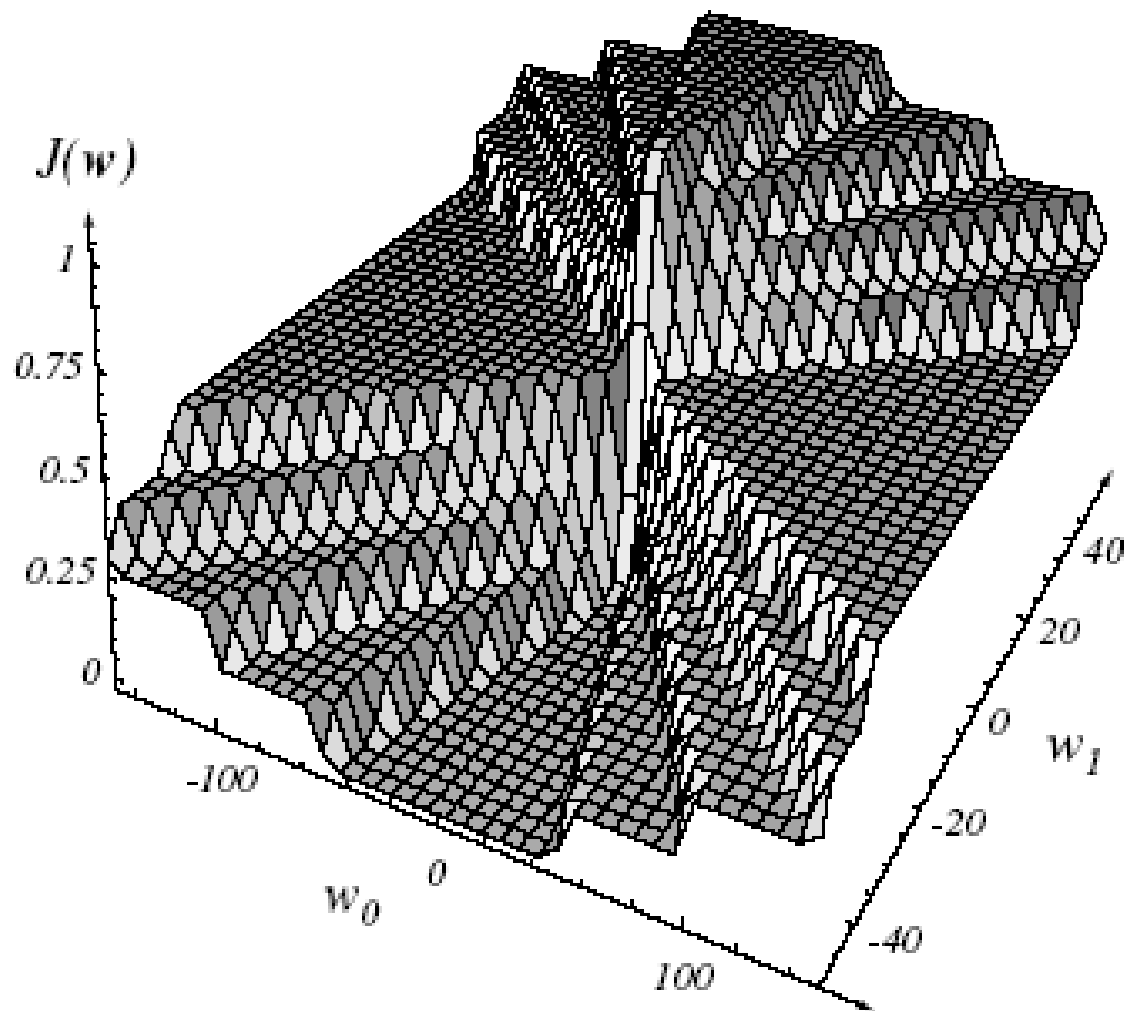


BP算法—批量修改

1. begin initialize n_H , \mathbf{w} , θ , η , $r \leftarrow 0$
2. do $r \leftarrow r + 1$
3. $m \leftarrow 0; \Delta w_{ji} \leftarrow 0; \Delta w_{kj} \leftarrow 0$
4. do $m \leftarrow m + 1$
5. $\mathbf{x}_m \leftarrow \text{select pattern}$
6. $\Delta w_{ji} \leftarrow \Delta w_{ji} + \eta \delta_j x_i; \Delta w_{kj} \leftarrow \Delta w_{kj} + \eta \delta_k y_j$
7. until $m = n$
8. $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}; w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$
9. until $\|\nabla J(\mathbf{w})\| < \theta$
10. return \mathbf{w}
11. end

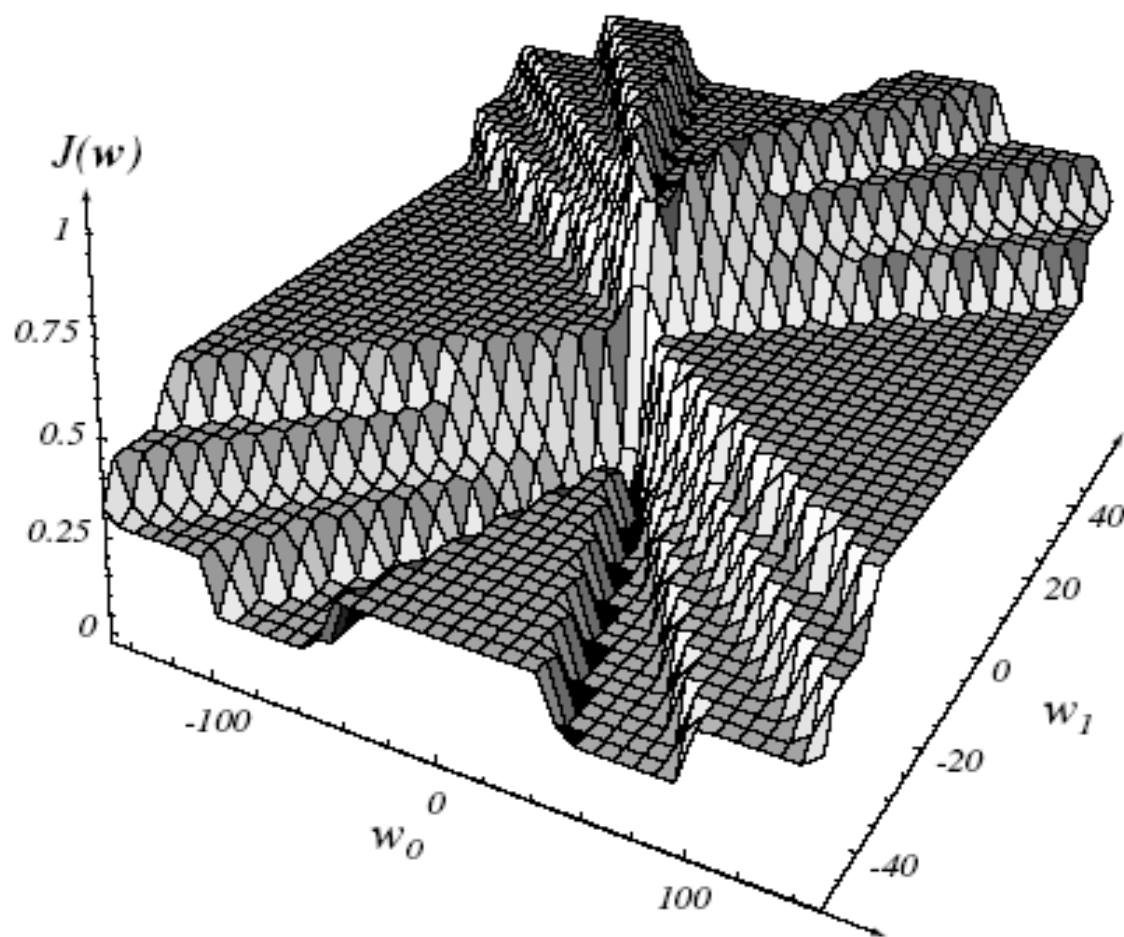
多层感知器网络存在的问题

1. BP算法的收敛速度一般来说比较慢;



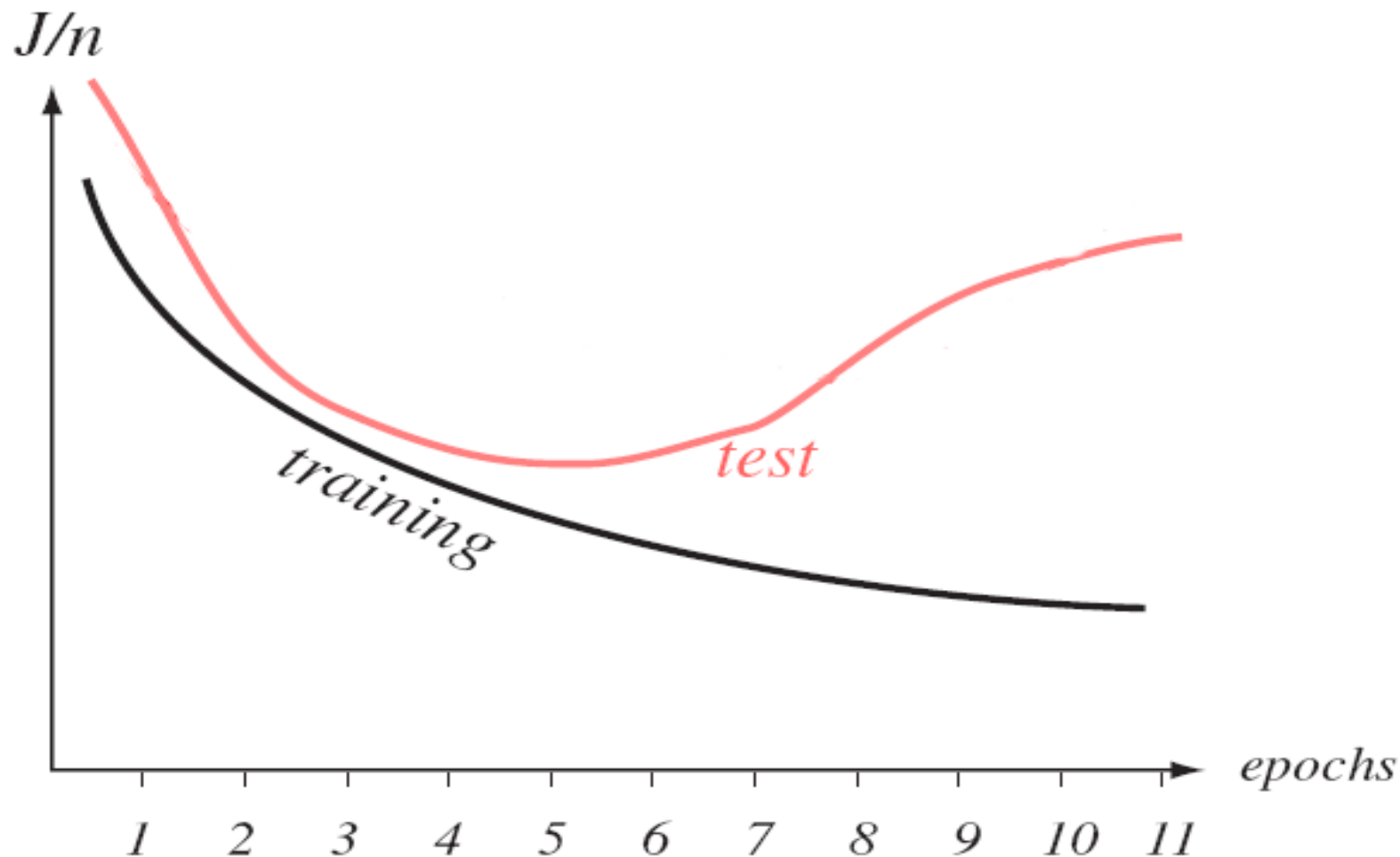
多层感知器网络存在的问题

2. BP算法只能收敛于局部最优解，不能保证收敛于全局最优解；



多层感知器网络存在的问题

3. 当隐层元的数量足够多时，网络对训练样本的识别率很高，但对测试样本的识别率有可能很差，即网络的

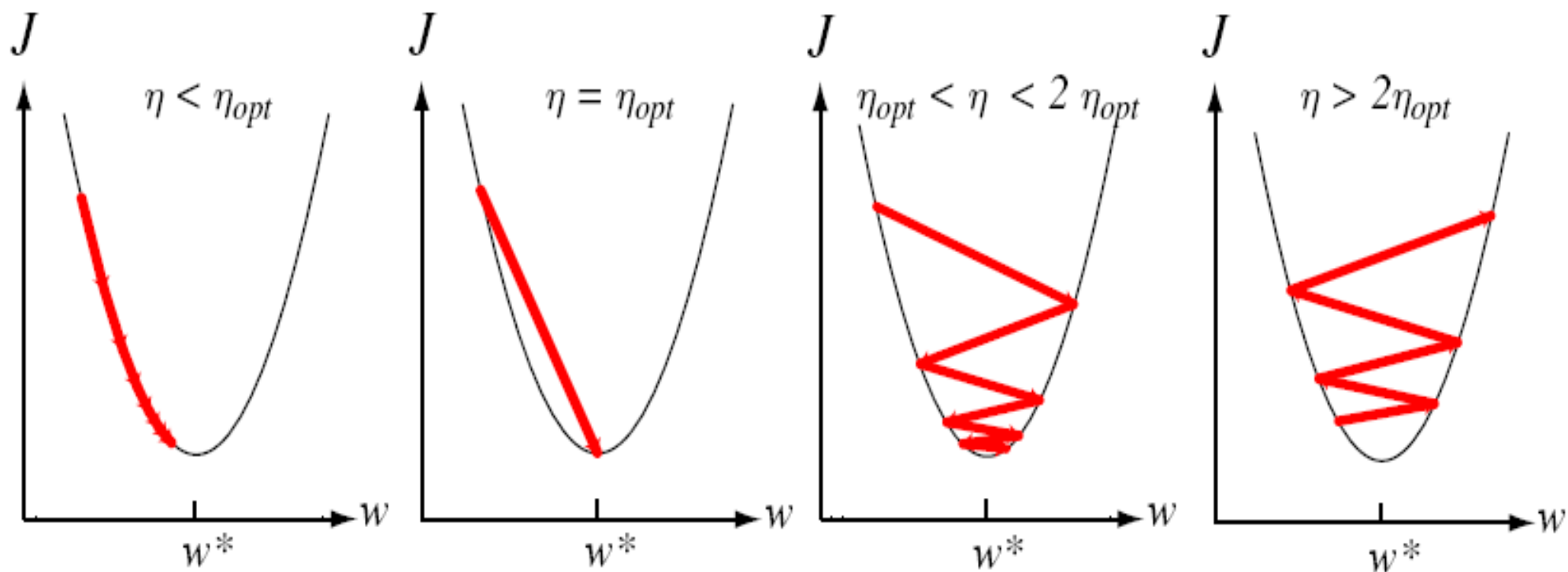


BP算法的一些实用技术

- **激活函数的选择**：一般可以选择双曲型的Sigmoid函数；
- **目标值**：期望输出一般选择 $(-1, +1)$ 或 $(0, 1)$ ；
- **规格化**：训练样本每个特征一般要规格化为0均值和标准差；
- **权值初始化**：期望每个神经元的 $-1 < \text{net} < +1$ ，因此权值一般初始化为 $-1/\sqrt{d} < w < 1/\sqrt{d}$ ；
- **学习率的选择**：太大容易发散，太小则收敛较慢；

提高收敛速度的方法

■ 一个比较直观的想法是通过增大学习率来提高收敛速度，但这样有可能造成算法发散。



一阶技术——梯度下降法

- 目标函数的一阶泰勒级数展开：

$$J(\mathbf{w}_{k+1}) = J(\mathbf{w}_k + \Delta\mathbf{w}_k) \approx J(\mathbf{w}_k) + \left(\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right)^T \Delta\mathbf{w}_k$$

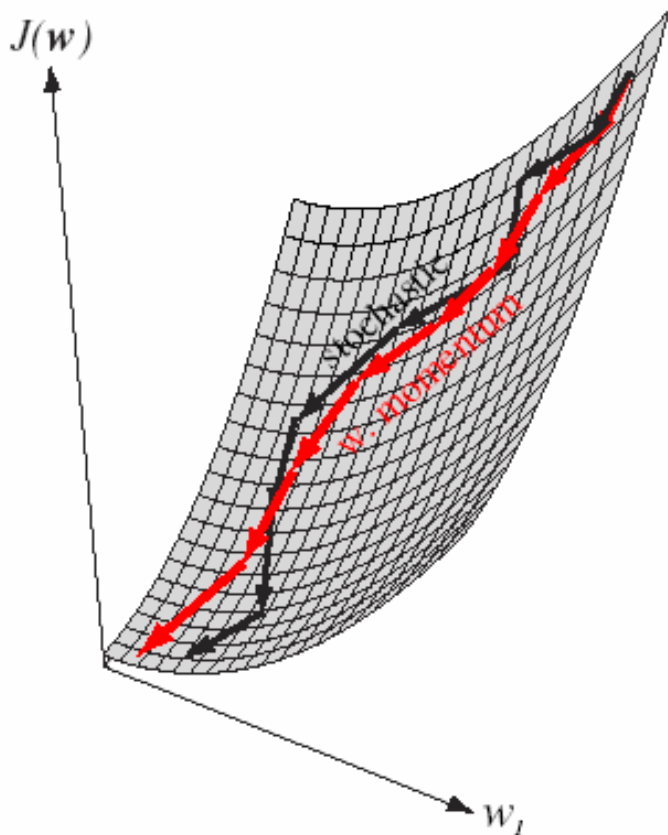
目标函数增量：

$$\Delta J(\mathbf{w}_k) = \left(\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right)^T \Delta\mathbf{w}_k \leq 0$$

使目标函数下降最大：

$$\Delta\mathbf{w}_k = - \left(\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right)^T$$

冲量项



$$\Delta \mathbf{w}(m) = \mathbf{w}(m) - \mathbf{w}(m-1)$$

减少总体梯度方向的偏离，
加快学习速度

$$\mathbf{w}(m+1) = \mathbf{w}(m) + (1-\alpha)\Delta \mathbf{w}_{bp}(m) + \alpha\Delta \mathbf{w}(m-1)$$

权值衰减

对误差函数不起作用的权值，将逐步减小。等价于引入正则项：

$$J_{ef} = J(\mathbf{w}) + \frac{2\varepsilon}{\eta} \mathbf{w}^t \mathbf{w}$$

正则项，优先惩罚较大的权值

$$\mathbf{w}^{new} = \mathbf{w}^{old} (1 - \varepsilon) \quad 0 < \varepsilon < 1$$

$$J_{ef} = J(\mathbf{w}) + \frac{2\varepsilon}{\eta} \sum_{i,j} \frac{w_{i,j}^2 / \mathbf{w}^t \mathbf{w}}{1 + w_{i,j}^2 / \mathbf{w}^t \mathbf{w}}$$

使用衰减参数，将惩罚分散于整个网络

线索 (hint)

增加输出单元来执行附加问题，有助于提高分类能力

