

# 模式识别

## 第6章：非线性判别函数分类器

主讲人：张治国

[zhiguo Zhang@hit.edu.cn](mailto:zhiguo Zhang@hit.edu.cn)



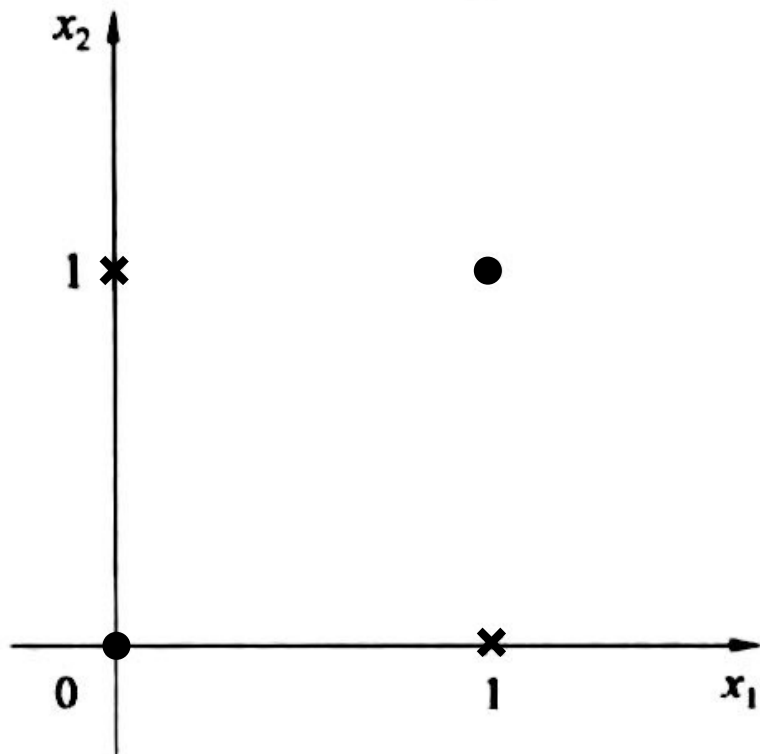
# 本章内容

---

- 广义线性判别函数分类器
  - 异或问题的非线性判别函数
  - 广义线性判别函数分类器一般原理
- 多层感知器网络
  - 解决异或问题的多层感知器
  - 多层感知器的结构
  - 多层感知器的学习
  - 多层感知器学习算法的改进
- 支持向量机
  - 最优线性判别函数分类器
  - 支持向量机的学习
  - 核函数与非线性支持向量机

# 线性分类器的局限

- 很多现实应用中，不同类别的样本并不是线性可分的。例如，简单而基本的异或问题。

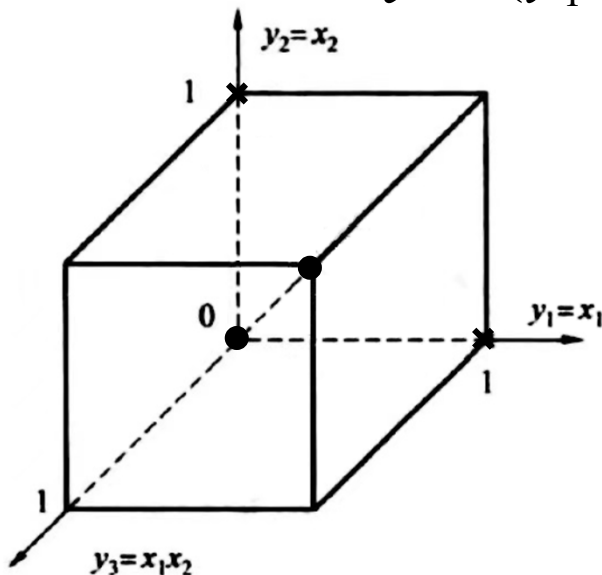


$\omega_1$  类样本:  $\{(0, 0)^T, (1, 1)^T\}$   
 $\omega_2$  类样本:  $\{(1, 0)^T, (0, 1)^T\}$

# 异或问题的非线性判别函数

- 建立一个非线性判别函数可以按某种变换将其转化为线性判别函数。
- 例：异或问题中将2维矢量扩展为3维矢量，可以引入一个新特征  $x_1x_2$ ，使得

$$\mathbf{y} = (y_1, y_2, y_3)^T = (x_1, x_2, x_1x_2)^T$$

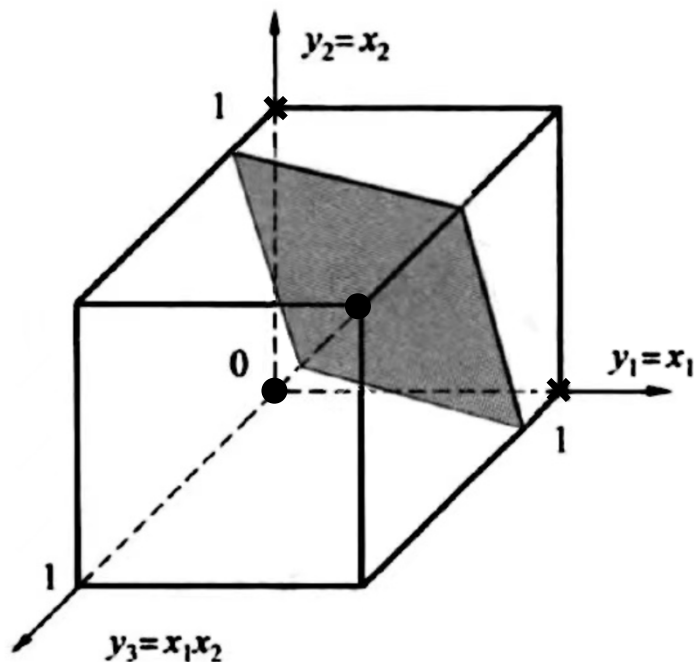


$\omega_1$  类样本:  $\{(0, 0, 0)^T, (1, 1, 1)^T\}$   
 $\omega_2$  类样本:  $\{(1, 0, 0)^T, (0, 1, 0)^T\}$

# 异或问题的非线性判别函数

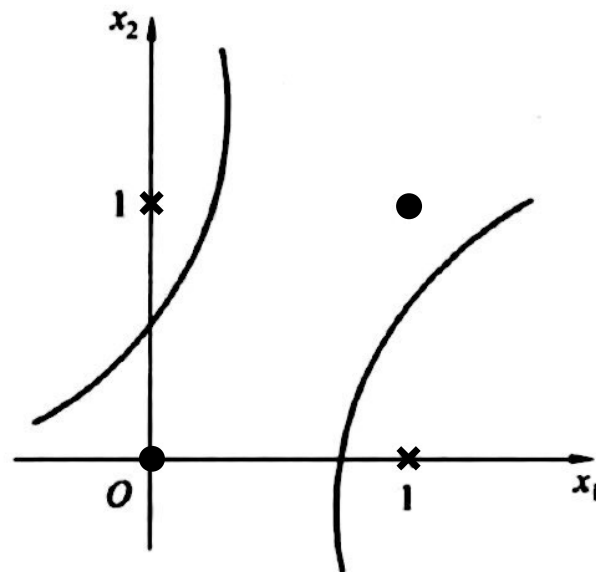
- 2维不可分的样本在3维空间中有可能线性可分。

异或问题的3维空间扩展



$$g(\mathbf{y}) = -2y_1 - 2y_2 + 6y_3 + 1$$

异或问题的二次判别函数  
分类界面



$$g(\mathbf{x}) = -2x_1 - 2x_2 + 6x_1x_2 + 1$$

# 广义线性判别函数分类器

---

- 广义线性判别函数分类器：通过增加定义于原空间上的非线性函数作为特征，实现从低维空间向高维空间的映射，从而提升线性可分能力。
- 在高维空间中得到的线性判别函数对应着低维空间中的非线性函数。
- 常用的非线性函数是多项式（见课本6.1.2），也可以是指数函数、对数函数等。

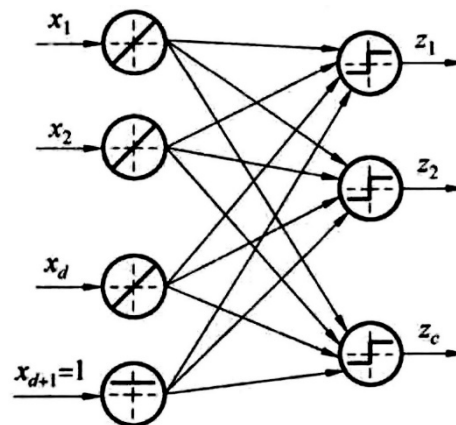
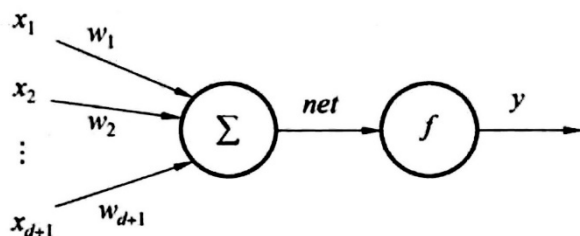
# 广义线性判别函数的问题

---

- **非线性映射的选择**：非线性映射的方式很多，参数设置也不同；没有方法可以确保选择到合适的非线性映射使得映射后的样本可分。
- **特征维数灾难**：特征维数增加有助于提高线性可分能力，一般地， $r \geq n + 1$  维空间中的  $n$  个两类训练样本都是线性可分的（证明略）。但过高的维度会带来“维数诅咒”。

# 多层感知器网络

- 由人工神经元可以构成两层的感知器网络，但两层感知器仍是线性分类器。



- 多个感知器级联可以实现非线性的判别能力。与广义线性判别函数人为设定非线性映射不同，多层感知器从样本集中学习得到非线性映射。



# 多层感知器网络

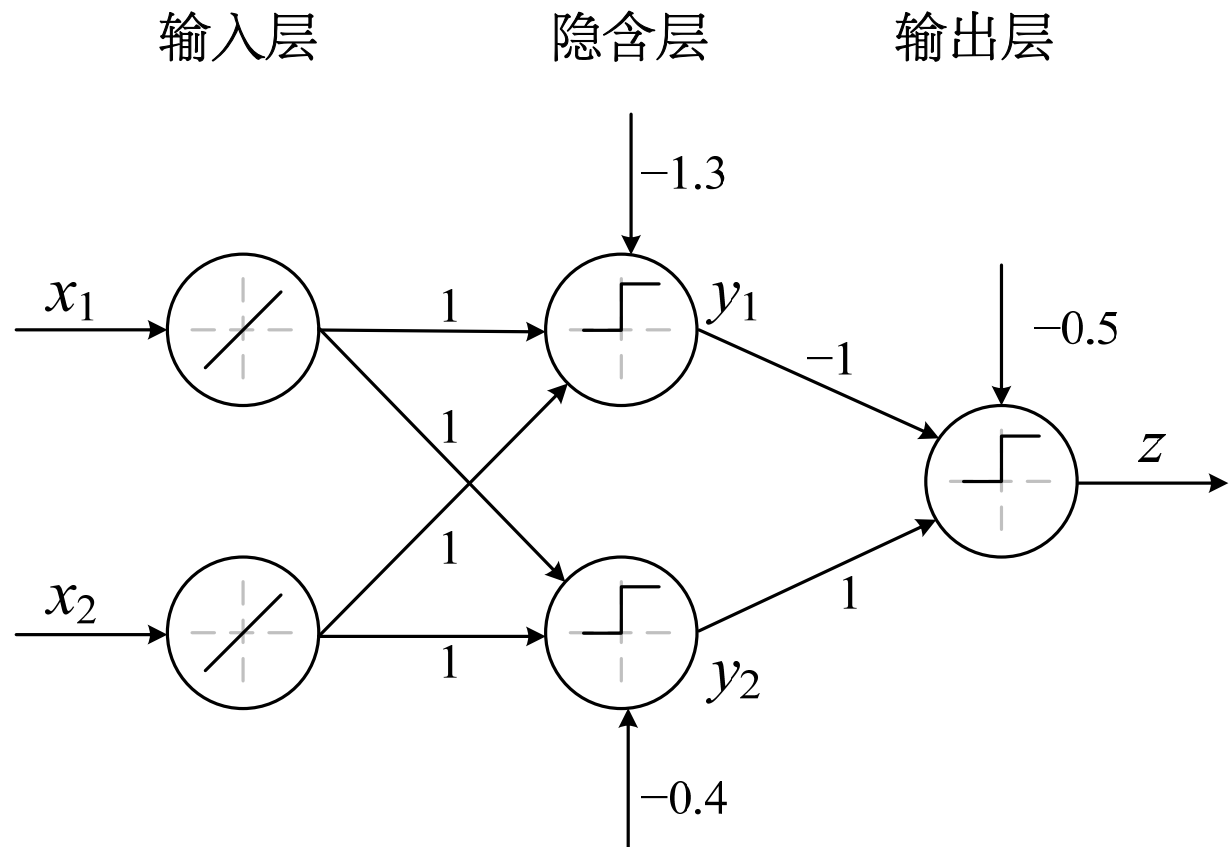
- 例：可解决异或问题的多层感知器

输入层使用  
线性激活函数

$$f(u) = u$$

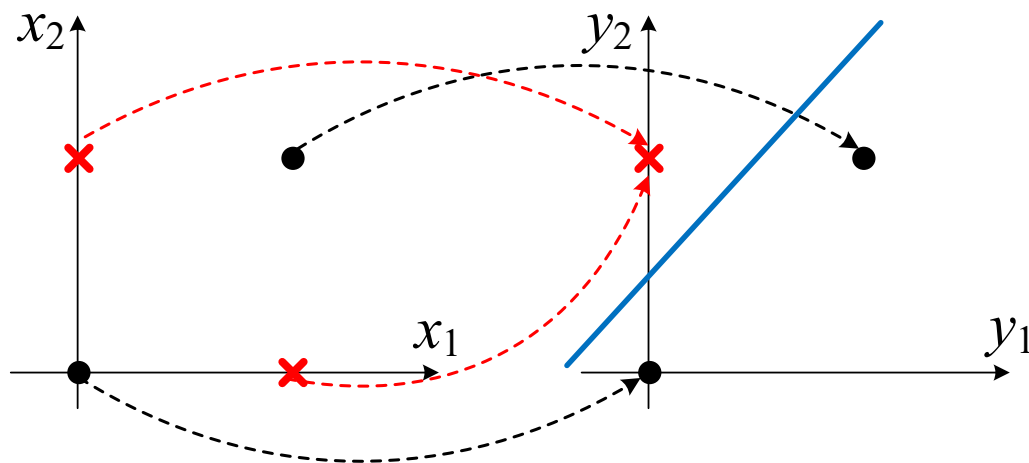
隐含层和输出层使用  
0-1阶跃激活函数

$$f(u) = \begin{cases} 1, & u > 0 \\ 0, & u \leq 0 \end{cases}$$



# 多层感知器网络

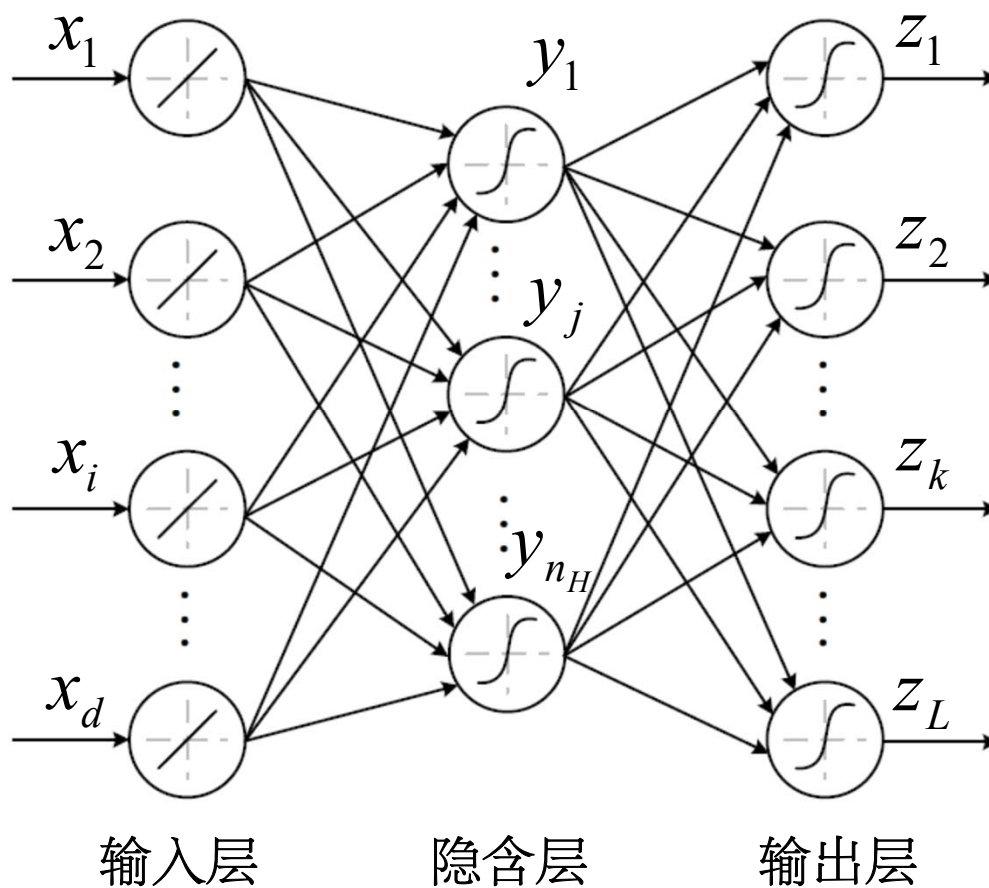
- 隐含层实现对输入空间的非线性映射，输出层实现线性分类判别。
- 隐含层的非线性映射（下图红黑虚线）和输出层的线性判别函数（蓝色实线）可以同时学习。



- 问题：如何设计网络结构？如何学习映射参数？

# 多层感知器的结构

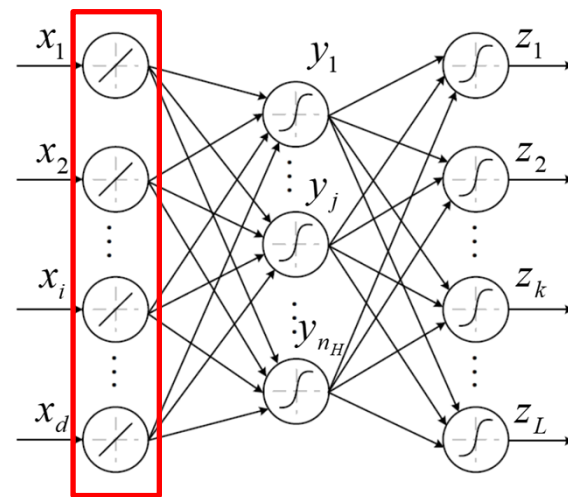
- 三层感知器网络的结构：



为简化计，  
偏置项未显示

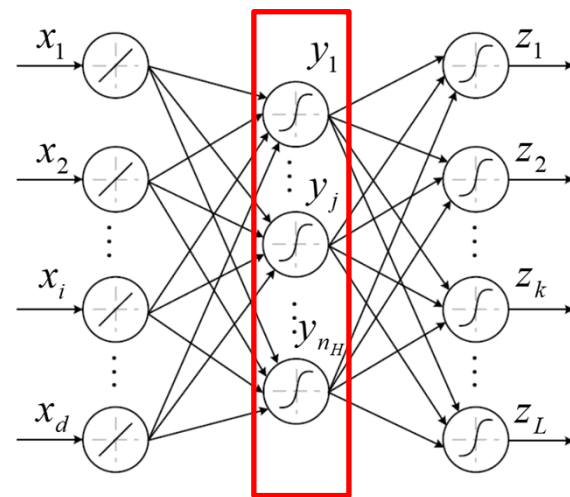
# 多层感知器的结构

- **输入层**：将训练样本的特征输入到多层感知器网络中。
- 输入层的神经元个数为输入样本特征的维数。
- 神经元的输入连接到第一隐含层每个神经元的输入，映射函数采用线性函数。



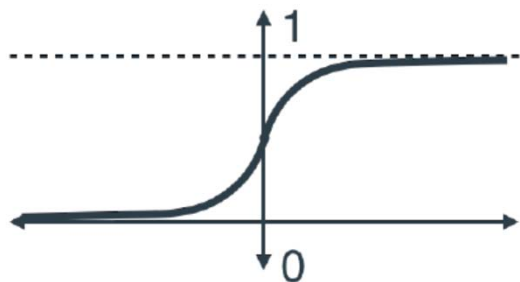
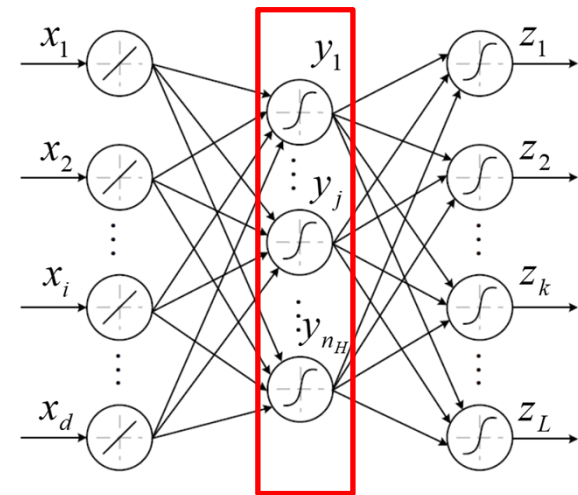
# 多层感知器的结构

- **隐含层**：实现样本从原特征空间向隐含层输出空间的非线性映射。
- 隐含层的数量和神经元数节点数（每层无需相同）需要根据实际问题设定。
- 一般地，隐含层越多和神经元数越多，网络的非线性分类能力越强。



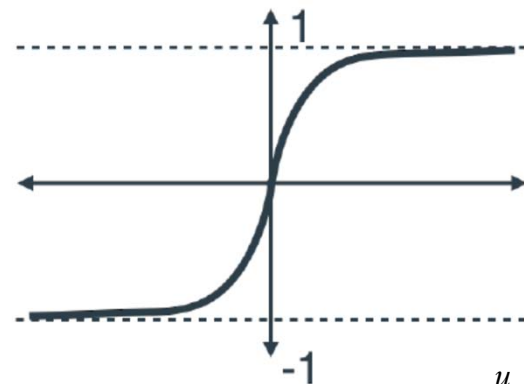
# 多层感知器的结构

- 隐藏层的激活函数：一般采用 Sigmoid型函数。
- 常用的有对数型Sigmoid函数和双曲正切型Sigmoid函数。



注意，激活函数的取值范围或有不同。

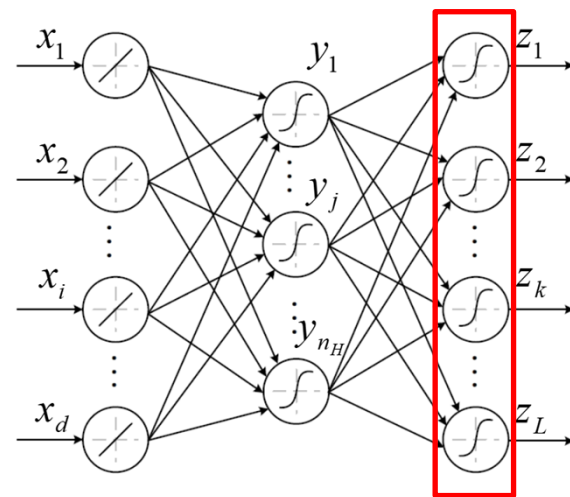
对数型Sigmoid函数  $f(u) = \frac{1}{1 + e^{-u}}$



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

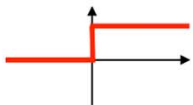
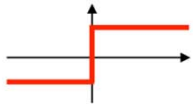

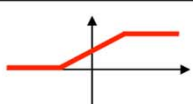
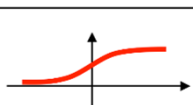
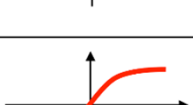


# 多层感知器的结构

- **输出层**：输出层神经元的个数由识别问题的分类数决定：
  - 1) 输出神经元数量等于类别数 $c$
  - 2) 采用编码输出的方式，输出神经元数量为  $\log_2 c$ 。
- 输出层神经元激活函数可以采用线性函数、Sigmoid函数或阶跃函数等。



# 多层感知器的结构

## 常用激活函数

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	



# 三层感知器识别算法

---

- 输入待识别样本  $\mathbf{x}$ ，隐含层和输出层分别有  $n_H$  和  $L$  个神经元，隐含层第  $j$  个节点的权值矢量、偏置和激活函数分别为  $\mathbf{w}_{h,j}$ ， $b_{h,j}$ ，和  $f_h(u)$ ；输出层的第  $k$  个节点权值矢量、偏置和激活函数分别为  $\mathbf{w}_{o,k}$ ， $b_{o,k}$ ，和  $f_o(u)$ 。
  - 计算隐含层神经元的输出：
    - 计算每个神经元的净输入： $net_{h,j} = \mathbf{w}_{h,j}^T \mathbf{x} + b_{h,j}$ ；
    - 计算每个神经元的实际输出： $y_j = f_h(net_{h,j})$ ， $j = 1, \dots, n_H$ ；
  - 计算输出层的输出，隐含层的输出矢量表示为  $\mathbf{y} = (y_1, \dots, y_{n_H})^T$ ：
    - 计算每个神经元的净输入： $net_{o,k} = \mathbf{w}_{o,k}^T \mathbf{y} + b_{o,k}$ ；
    - 计算每个神经元的实际输出： $z_k = f_o(net_{o,k})$ ， $k = 1, \dots, L$ ；
  - 根据网络的输出矢量  $\mathbf{z} = (z_1, \dots, z_L)^T$  确定样本  $\mathbf{x}$  的类别属性。
-

# 多层感知器的学习

---

- 多层感知器的权值学习要转化为对误差平方和的优化问题求解。
- 令训练集包含  $n$  个样本  $D = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ , 根据每个样本类别属性设定相应期望输出矢量  $\{\mathbf{t}_1, \dots, \mathbf{t}_n\}$  (与激活函数取值范围和输出层的设置有关)。
- 将网络中所有权值和偏置表示为统一的矢量  $\mathbf{w}$ , 网络实际输出为  $\{\mathbf{z}_1, \dots, \mathbf{z}_n\}$ , 则需要优化的函数为

$$\min_{\mathbf{w}} J(\mathbf{w}) = \sum_{i=1}^n E_i(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n \|\mathbf{t}_i - \mathbf{z}_i\|^2$$

# 多层感知器的学习

---

- 多层感知器中，实际输出和权值矢量间的关系复杂，难以使用微分求解上述优化函数。
- 可以使用**梯度下降法**从初始的参数矢量  $\mathbf{w}(0)$  开始进行优化迭代直到收敛（ $\kappa$ 为迭代次数）：

$$\mathbf{w}(\kappa) = \mathbf{w}(\kappa) - \eta \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}(\kappa-1)}$$

- 因为  $J(\mathbf{w}) = \sum_{i=1}^n E_i(\mathbf{w})$ ，关键是计算  $E_i(\mathbf{w}) = \frac{1}{2} \|\mathbf{t}_i - \mathbf{z}_i\|^2$  关于输入层和隐含层的每个网络参数的偏导数。

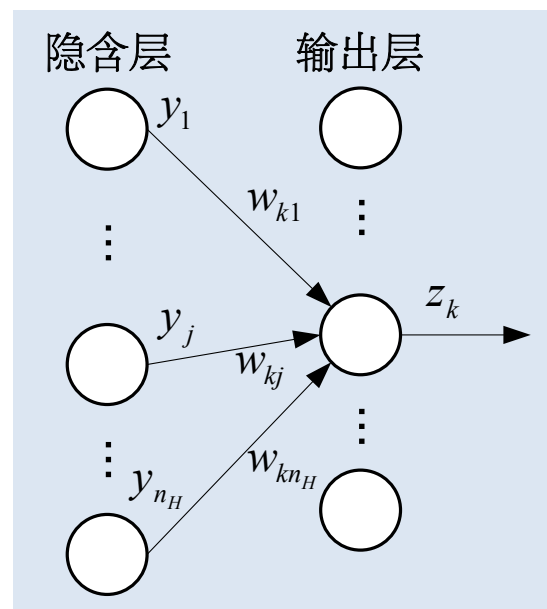
# 多层感知器的学习

- **输出层**：考虑包含  $n_H$  个节点的隐含层的第  $j$  个神经元与包含  $L$  个节点的输出层的第  $k$  个节点的权值矢量  $w_{kj}$  和偏置  $b_k$ 。输出层的净输入为  $\{net_1, \dots, net_L\}$ ，输出层和隐含层节点激活函数分别为  $f_o(u)$  和  $f_h(u)$ 。

- 应用隐函数求导法有：

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

$$\frac{\partial E}{\partial b_k} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial b_k}$$



# 多层感知器的学习

- 由于  $E = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2 = \frac{1}{2} \sum_{k=1}^L (t_k - z_k)^2$ ,  $\frac{\partial E}{\partial z_k} = -(t_k - z_k)$

$$z_k = f_o(net_k), \quad \frac{\partial z_k}{\partial net_k} = f_o'(net_k)$$

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j + b_k, \quad \frac{\partial net_k}{\partial w_{kj}} = y_j, \quad \frac{\partial net_k}{\partial b_k} = 1$$

- 因此得到：

$$\begin{aligned} \frac{\partial E}{\partial w_{kj}} &= -(t_k - z_k) f_o'(net_k) y_j \\ \frac{\partial E}{\partial b_k} &= -(t_k - z_k) f_o'(net_k) \end{aligned} \Rightarrow \begin{aligned} \frac{\partial E}{\partial w_{kj}} &= -\delta_k y_j \\ \frac{\partial E}{\partial b_k} &= -\delta_k \end{aligned}$$

# 多层感知器的学习

- **隐含层**：考虑隐含层的第  $j$  个节点与输入层的第  $m$  个节点的权值矢量  $w_{jm}$  和偏置  $b_j$ 。

- 应用隐函数求导法有：

$$\frac{\partial E}{\partial w_{jm}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{jm}}$$

$$\frac{\partial E}{\partial b_j} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial b_j}$$

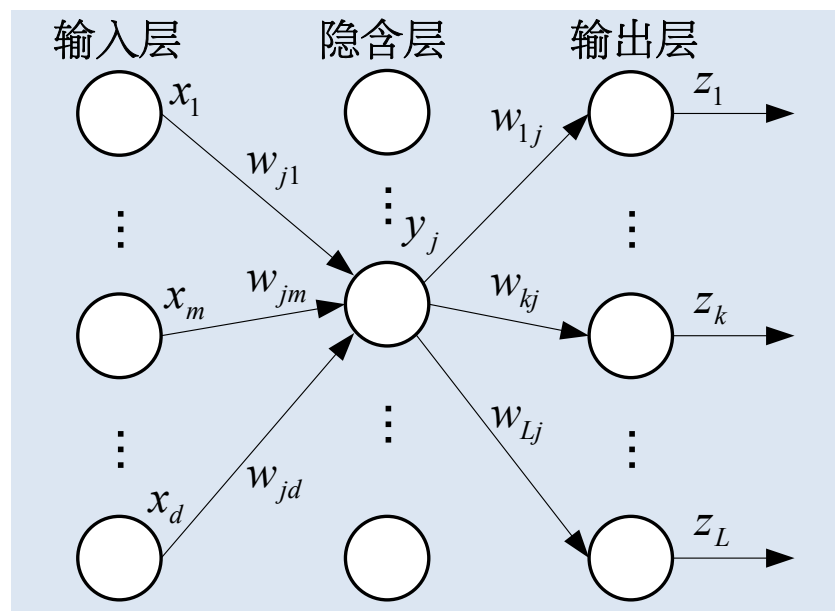
其中

$$y_j = f_h(net_j),$$

$$net_j = \sum_{i=1}^d w_{ji} x_i + b_j,$$

$$\frac{\partial y_j}{\partial net_j} = f'_h(net_j)$$

$$\frac{\partial net_j}{\partial w_{jm}} = x_m, \frac{\partial net_j}{\partial b_j} = 1$$



# 多层感知器的学习

- $\partial E / \partial y_j$  的计算可以通过  $z_k$  进行:

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

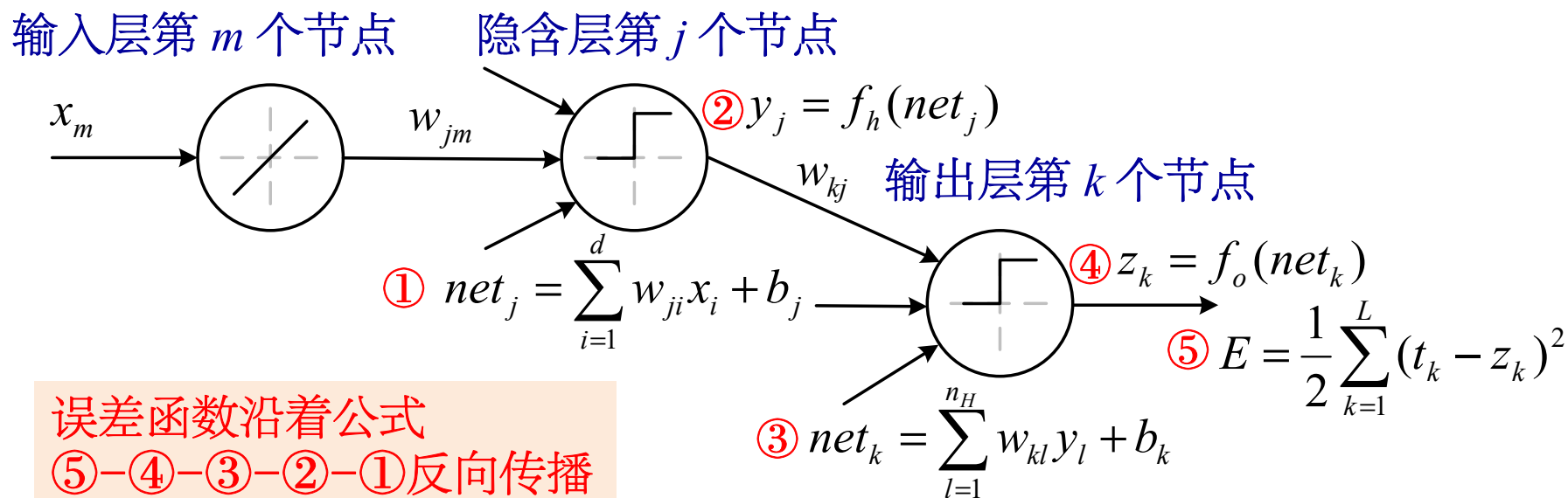
• 可得:

$$\begin{aligned}\frac{\partial E}{\partial w_{jm}} &= - \left[ \sum_{k=1}^L \delta_k w_{kj} \right] f'_h(net_j) x_m \\ \frac{\partial E}{\partial b_j} &= - \left[ \sum_{k=1}^L \delta_k w_{kj} \right] f'_h(net_j)\end{aligned}$$

$\Rightarrow \delta_j$

$$\begin{aligned}\frac{\partial E}{\partial w_{jm}} &= -\delta_j x_m \\ \frac{\partial E}{\partial b_j} &= -\delta_j\end{aligned}$$

# 多层感知器的学习



- 链式求导：计算误差函数对“隐含—输出”连接权值  $w_{kj}$  和“输入—隐含层”连接权值  $w_{jm}$  的导数。

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}, \quad \frac{\partial E}{\partial w_{jm}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{jm}} = \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{jm}}$$

⑤      ④      ③      ②      ①



# BP算法

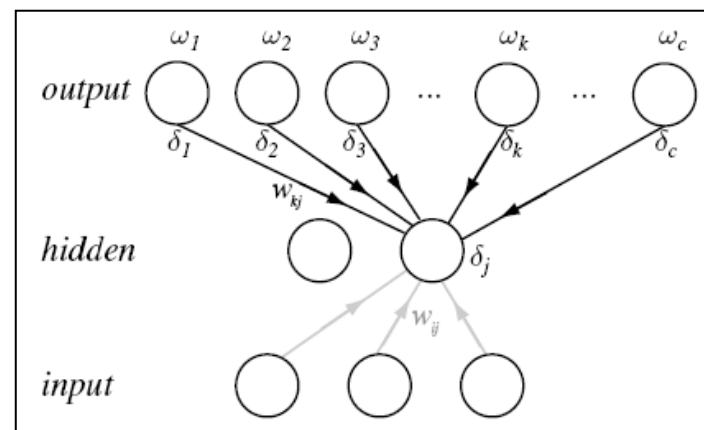
- 迭代公式

- 输出层:  $\frac{\partial E}{\partial w_{kj}} = -\delta_k y_j, \quad \delta_k = (t_k - z_k) f'_o(net_k)$

- 隐含层:  $\frac{\partial E}{\partial w_{jm}} = -\delta_j x_m, \quad \delta_j = \left[ \sum_{k=1}^L \delta_k w_{kj} \right] f'_h(net_j)$

隐含层节点的误差需要  
输出层节点的误差反向计算

- 多层感知器网络参数的学习  
算法称为**误差反向传播算法**  
(back propagation, BP)。



# BP算法

---

- BP算法的实现需要输入信号的前馈传递和误差信号的反向传递两个过程。
  - 前馈传递：依次计算隐含层和输出层每个节点的净输入  $net_j$  和  $net_k$ ，以及实际输出  $y_j$  和  $z_k$ 。
  - 反向传递：首先计算输出层的误差  $\delta_k$ ，然后反向计算隐含层的误差  $\delta_j$ 。
- BP算法有两种实现方式：批量学习和在线学习。

# BP算法

- BP算法（批量学习） 详见课本129-130页

```
1.      begin initialize  $n_H, \mathbf{w}, \theta, \eta, \kappa=0$ 
2.      do  $\kappa = \kappa + 1$ 
3.           $\Delta w_{jm} = 0; \Delta w_{kj} = 0$ 
4.          for  $i = 1$  to  $n$ 
5.               $\mathbf{x}_i \leftarrow$  select input sample
6.              forward propagation:  $net_j, y_j, net_k, z_k$ 
7.              back propagation:  $\delta_k, \delta_j$ 
8.               $\Delta w_{jm} \leftarrow \Delta w_{jm} + \delta_j x_{im}; \Delta w_{kj} \leftarrow \Delta w_{kj} + \delta_k y_j$ 
9.          end for
10.          $w_{jm} \leftarrow w_{jm} + \eta \Delta w_{jm}; w_{kj} \leftarrow w_{kj} + \eta \Delta w_{kj}$ 
11.         while  $\|\nabla J(\mathbf{w})\| > \theta$ 
12.         return  $\mathbf{w}$ 
13.     end
```

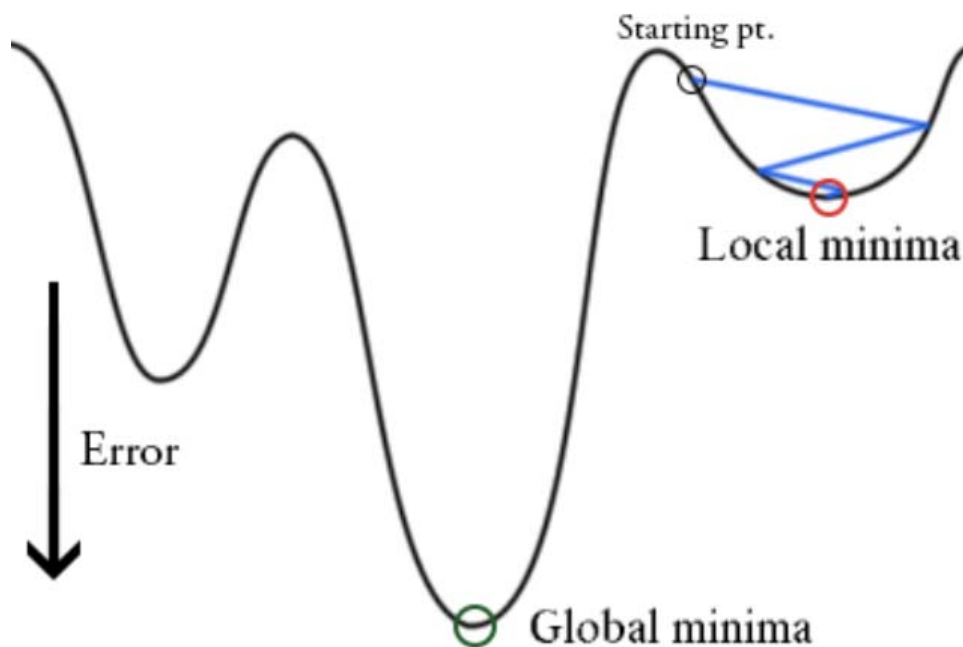
# BP算法注意事项

---

- **激活函数的选择**：一般可以选择双曲型的Sigmoid函数，深度网络可用ReLU等；
- **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**：不能过大或过小，也不可设为统一常量或0，可使用一些初始化方法如Xavier法或者预训练；
- **学习率的选择**：太大容易发散，太小收敛较慢。

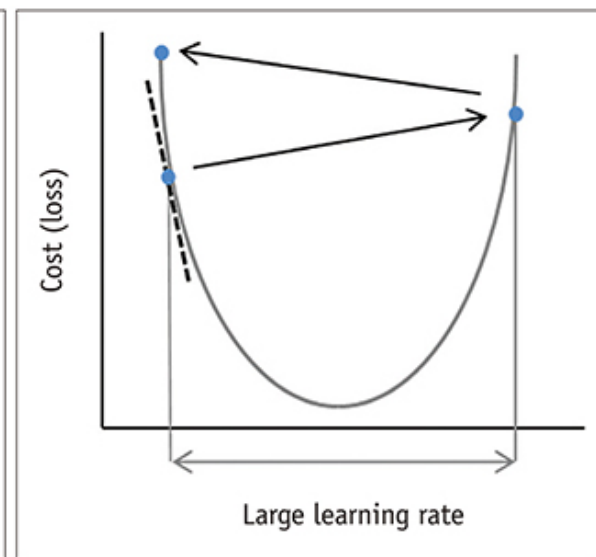
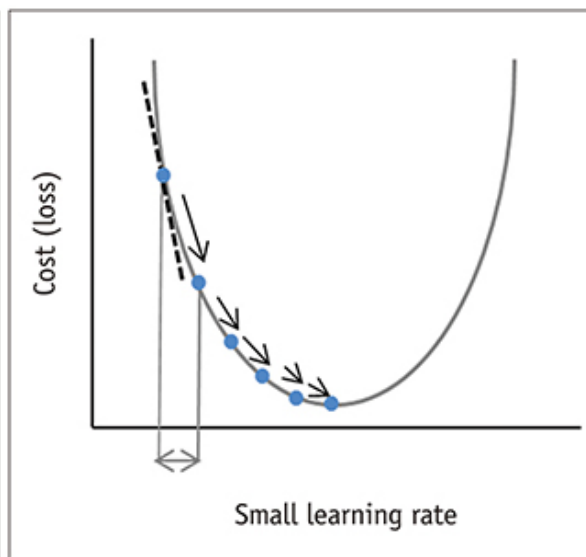
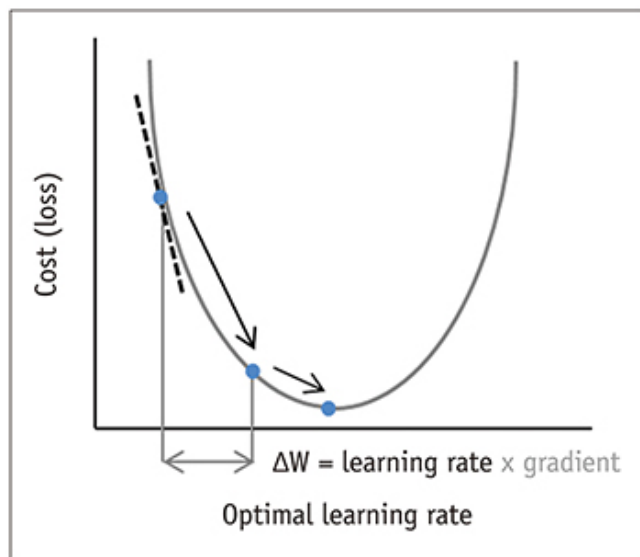
# 多层感知器存在的问题

- BP算法（基于梯度下降法）只能收敛于局部最优解，不能保证收敛于全局最优解。



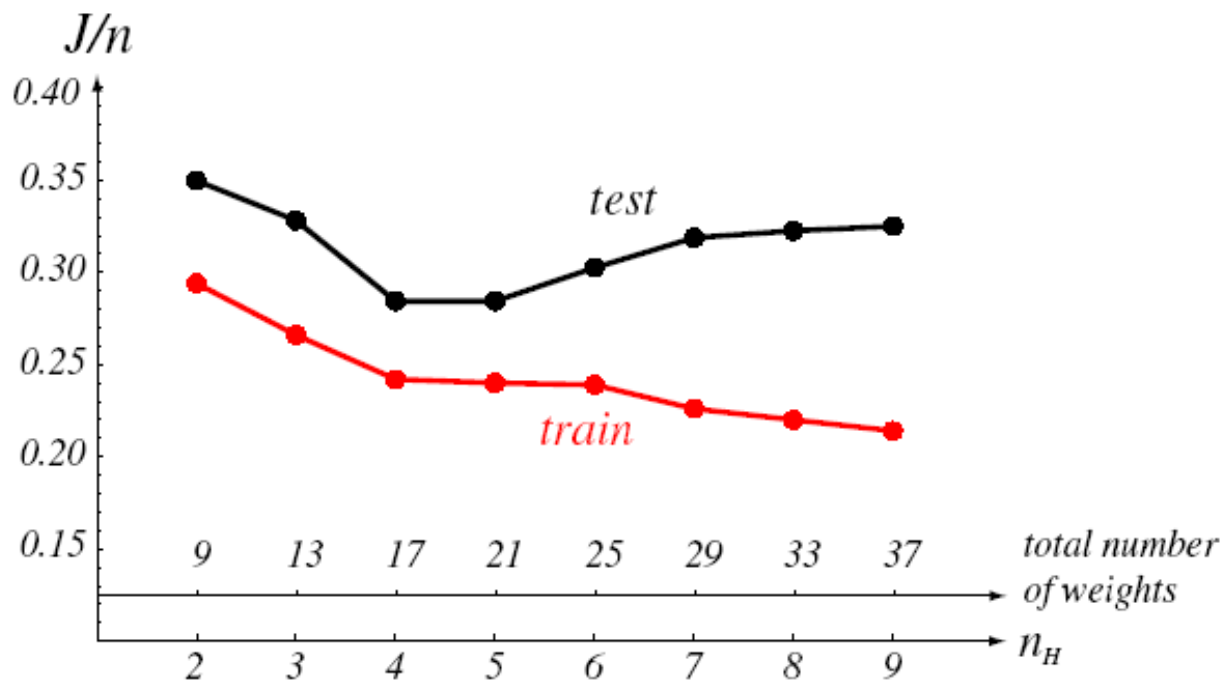
# 多层感知器存在的问题

- BP算法的收敛速度一般来说比较慢。可以通过增大学习率来提高收敛速度，但也有可能造成算法发散。



# 多层感知器存在的问题

- 当隐含层神经元的数量足够多时，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，即网络的泛化能力可能较差。



# 多层感知器学习算法的改进

---

- 多层感知器学习算法的改进途径：主要通过调整增量方向和学习率以达到既快又准收敛的目的；此外，也要考虑在降低运算复杂度和避免局部收敛方面改进。
  - 冲量项：调整增量方向
  - 自适应学习率：调整学习率
  - 二阶优化技术（拟牛顿、共轭梯度等）：用二阶函数降低近似误差（与线性函数相比），进而降低复杂度，增加收敛速度等。



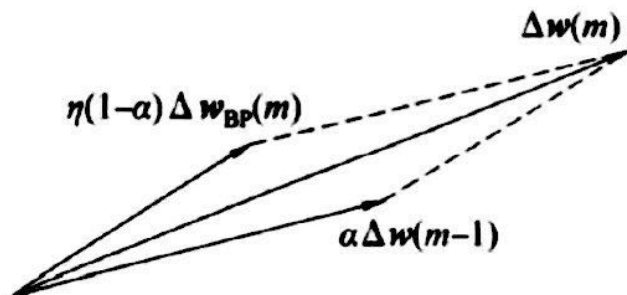
# 多层感知器学习算法的改进

- **冲量项法**：第  $m$  轮迭代的权值增量  $\Delta w(m)$  不只决定于当前的梯度（用  $\Delta w_{BP}(m)$  表示），还与上一轮迭代的权值增量  $\Delta w(m-1)$  有关：

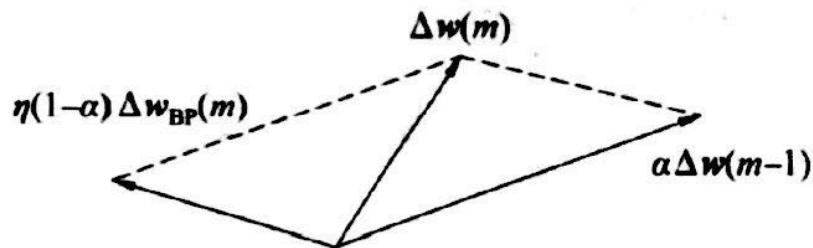
$$\Delta w(m) = \alpha \Delta w(m-1) + \eta(1-\alpha) \Delta w_{BP}(m)$$

其中  $0 < \alpha < 1$ （如取0.9）。

- 如果当前梯度和上一轮权值增量方向相近，就会增大增量，反之减小增量。



(a) 梯度与上一轮的增量方向相近



(b) “过冲”的冲量项权值增量

# 多层感知器学习算法的改进

---

- **自适应学习率**：通过比较每一轮权值调整前和调整后的优化函数值来决定学习率的增大或减小。
  - 如果权值调整能够使得优化函数值下降，则下一轮迭代中可以尝试更大学习率；
  - 如果权值调整使得优化函数值增加，说明上一轮的学习太冒进，应该减小学习率；
  - 首先计算权值调整前后优化函数的比值，然后根据该比值决定学习率的增大或减小。

# 多层感知器学习算法的改进

- BP算法是基于梯度下降法，其中目标函数的一阶泰勒级数展开为

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

- 为了使目标函数尽快下降，应该使  $\Delta\mathbf{w}_k$  取值为负的梯度方向，即

$$\Delta\mathbf{w}_k = - \left( \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_k} \right) / \left\| \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_k} \right\|$$

- 迭代公式为

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \left( \left. \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \right|_{\mathbf{w}=\mathbf{w}_k} \right)$$

详见附录B.2

# 多层感知器学习算法的改进

- BP算法的改进可以使用**牛顿法**，即基于二阶泰勒级数近似获得更优的权值增量：

$$\Delta J(\mathbf{w}_k) = \left( \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right)^T \Delta \mathbf{w}_k + \frac{1}{2} \Delta \mathbf{w}_k^T \mathbf{H} \Delta \mathbf{w}_k$$

其中  $\mathbf{H}$  是目标函数的二阶海森Hessian矩阵

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 J}{\partial w_1 \partial w_1} & \frac{\partial^2 J}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 J}{\partial w_1 \partial w_d} \\ \frac{\partial^2 J}{\partial w_2 \partial w_1} & \frac{\partial^2 J}{\partial w_2 \partial w_2} & \cdots & \frac{\partial^2 J}{\partial w_2 \partial w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 J}{\partial w_d \partial w_1} & \frac{\partial^2 J}{\partial w_d \partial w_2} & \cdots & \frac{\partial^2 J}{\partial w_d \partial w_d} \end{bmatrix}$$

详见附录B.3

# 多层感知器学习算法的改进

---

- 为了使  $\Delta J(\mathbf{w}_k)$  最小，对其微分求极值点，可得

$$\left( \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right) + \mathbf{H} \Delta \mathbf{w}_k = 0 \Rightarrow \Delta \mathbf{w}_k = -\mathbf{H}^{-1} \left( \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} \bigg|_{\mathbf{w}=\mathbf{w}_k} \right)$$

- $\mathbf{H}^{-1}$  相当于是最优学习率，对每个参数的学习率是不同的。但  $\mathbf{H}^{-1}$  的计算比较困难，因此牛顿法一般不直接使用。
- 基于牛顿法，发展了近似的二阶优化技术，如拟牛顿法和共轭梯度法等。

详见附录B.3

# 多层感知器学习算法的改进

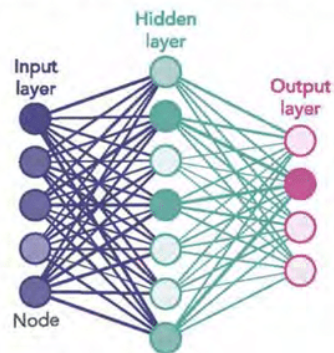
---

- 拟牛顿法（Quasi-Newton）：不直接计算 $H$ ，用另一个矩阵近似，该矩阵可以迭代计算，从而解决运算复杂度的问题。
- 共轭梯度法（Conjugate Gradient）：对于二次优化函数，权值沿初始方向移动到最小点，再沿着该方向关于 $H$ 的共轭可移动到全局最小点。
- Levenberg-Marquardt算法：针对均方误差准则函数的二阶方法，执行时修改参数达到结合高斯-牛顿算法以及梯度下降法的优点。

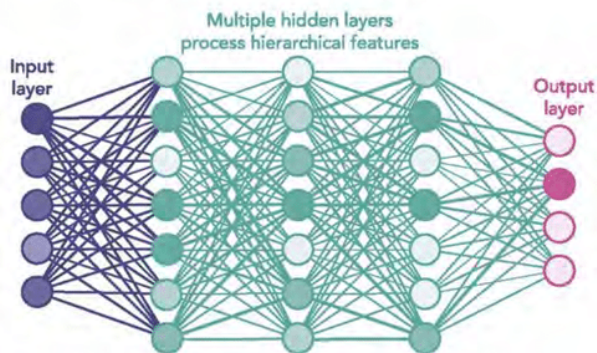
# 多层感知器与反向传播

- 多层感知器是深度神经网络的基本形式之一。
- 反向传播是深度神经网络训练方法的基础。

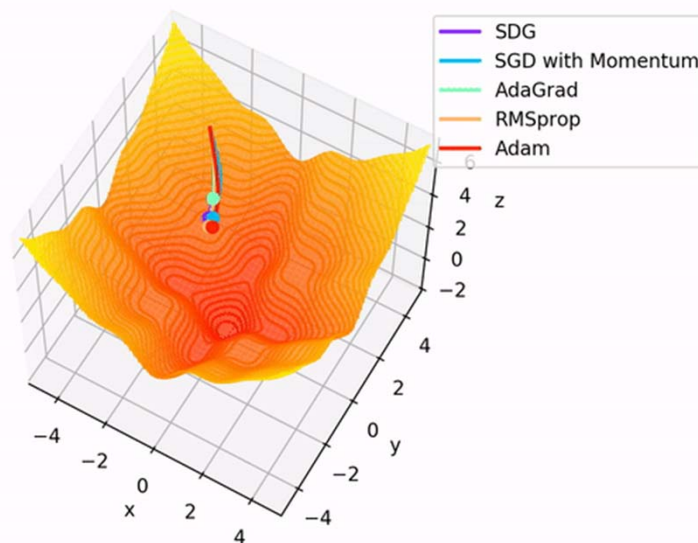
SHALLOW NEURAL NETWORK



DEEP NEURAL NETWORK



Optimizer Comparison





# 多层感知器与反向传播

- 深度神经网络技术的发展已经突飞猛进，但基本的问题和思路仍与多层感知器和反向传播类似：如何设计网络架构？如何优化求解？如何更快更准地优化求解？

## Learning representations by back-propagating errors



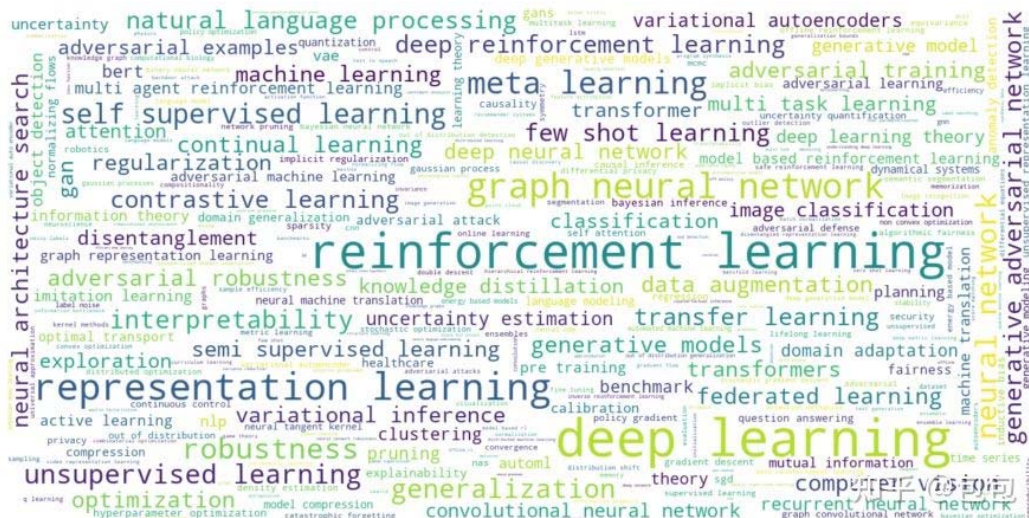
**David E. Rumelhart\*, Geoffrey E. Hinton†  
& Ronald J. Williams\***

\* Institute for Cognitive Science, C-015, University of California, San Diego, La Jolla, California 92093, USA

† Department of Computer Science, Carnegie-Mellon University,  
Pittsburgh, Philadelphia 15213, USA

nature 1986

We describe a new learning procedure, back-propagation, for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure<sup>1</sup>.





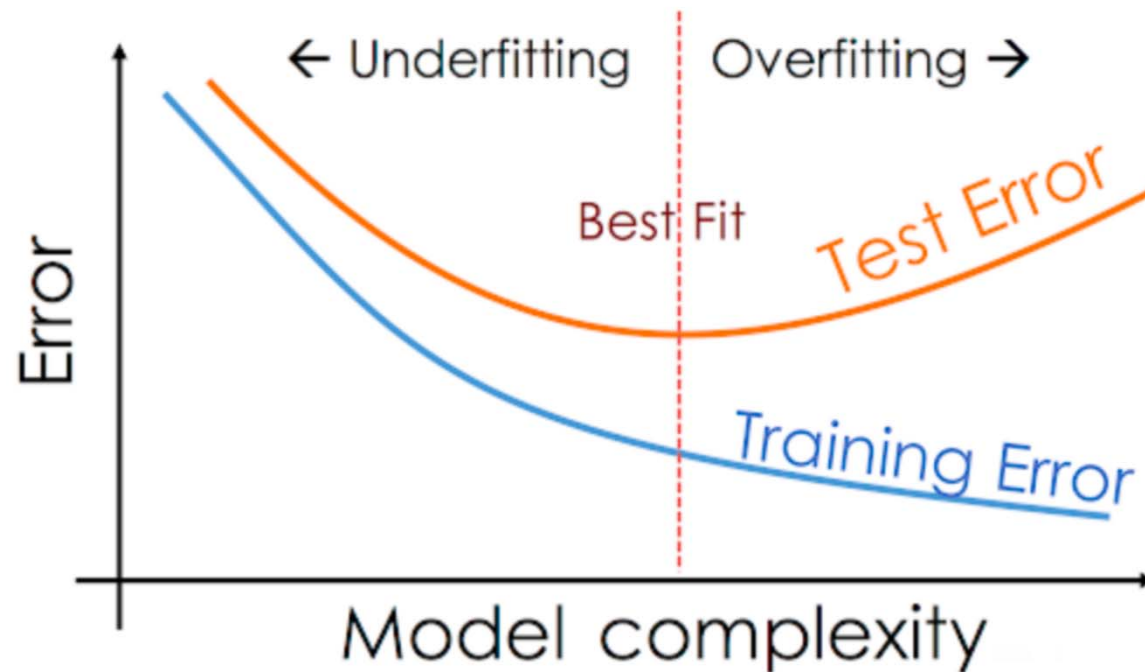
# 分类器性能评价准则

---

- 什么样的分类器是一个好的分类器？
- 模式识别的任务是确定能够提供最佳泛化性能的分类器，而不是提供最佳训练性能的分类器。
  - 训练性能：分类器识别训练数据类别的能力
  - 泛化性能（预测性能）：分类器识别未知标签的测试数据类别的能力

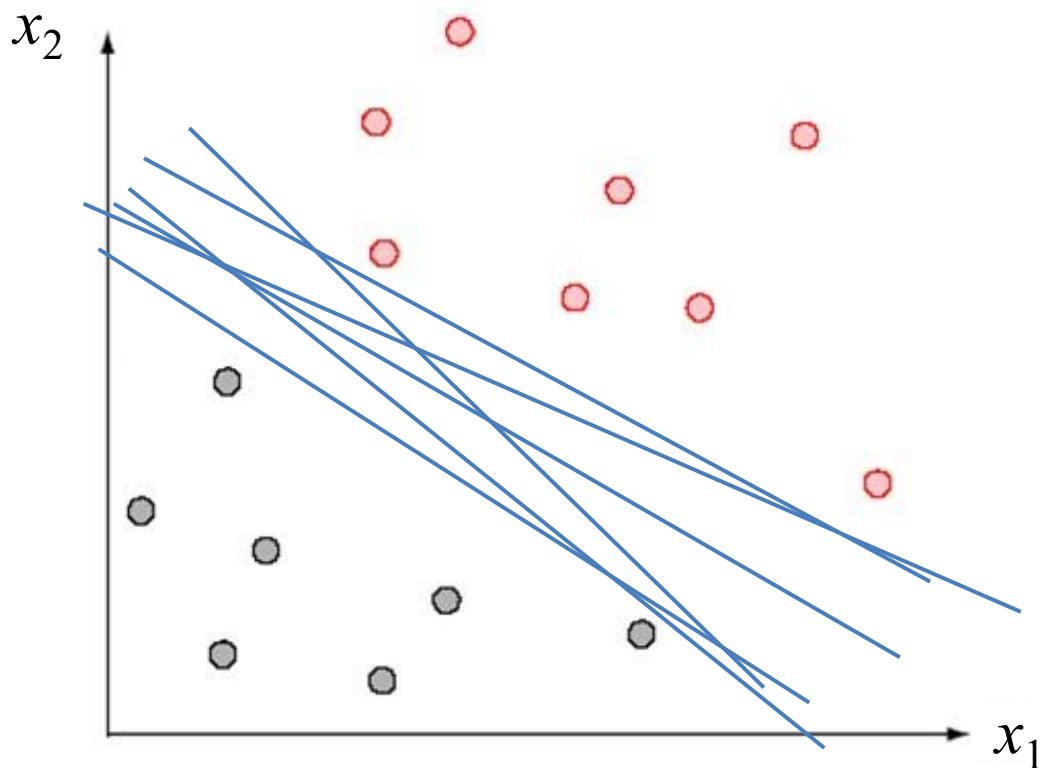
# 分类器性能评价准则

- 最佳分类器应该在测试数据上的表现最好（如，错误处于最小值）。



# 最优线性判别分类器

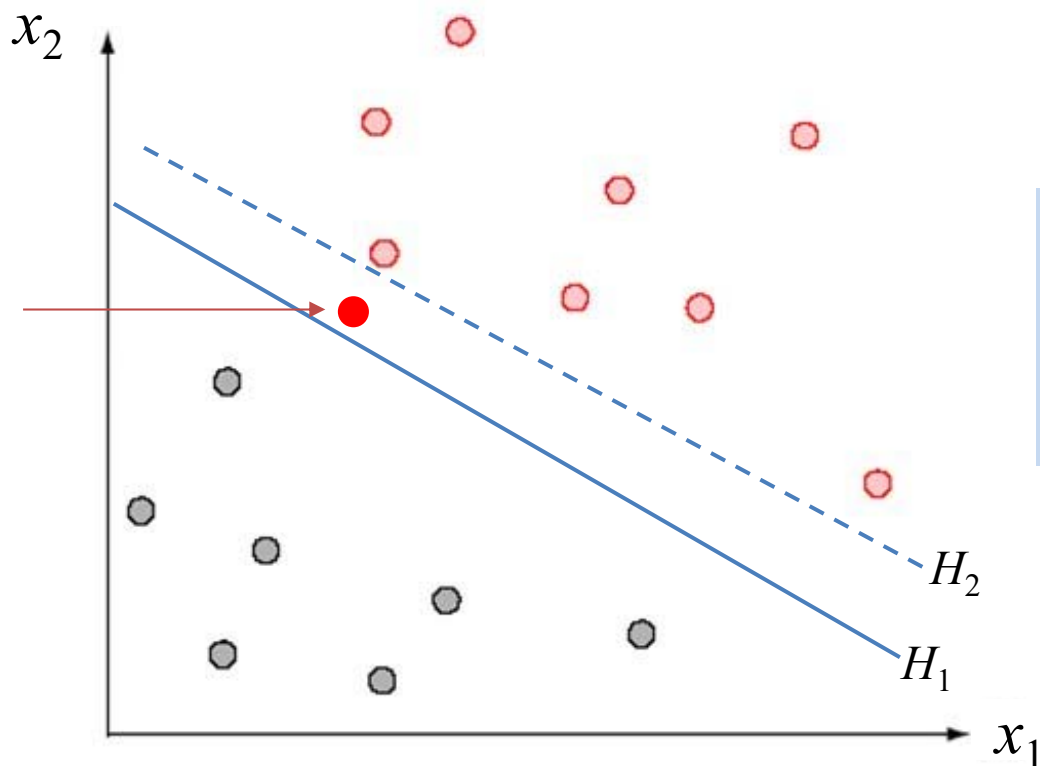
- 什么样的线性分类界面是泛化能力意义上的最优分类器？



# 最优线性判别分类器

- 什么样的线性分类界面是泛化能力意义上的最优分类器？

考虑一个新样本，属于红色类别。由于 $H_2$ 距离红色训练样本太近，引起误判； $H_1$ 仍可正确识别。



猜想：是否距离训练样本越远的分类界面泛化能力最强？

# 最优线性判别分类器

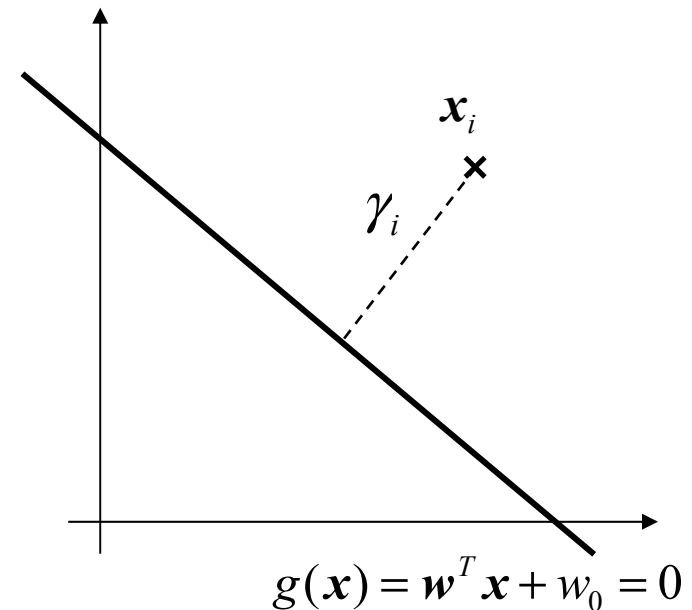
- 令分类界面对应的线性函数为  $g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0$  , 定义样本点  $\mathbf{x}_i$  到线性分类界面的两种间隔。

- 函数间隔:

$$b_i = |g(\mathbf{x}_i)| = |\mathbf{w}^T \mathbf{x}_i + w_0|$$

- 几何间隔:

$$\gamma_i = \frac{b_i}{\|\mathbf{w}\|}$$



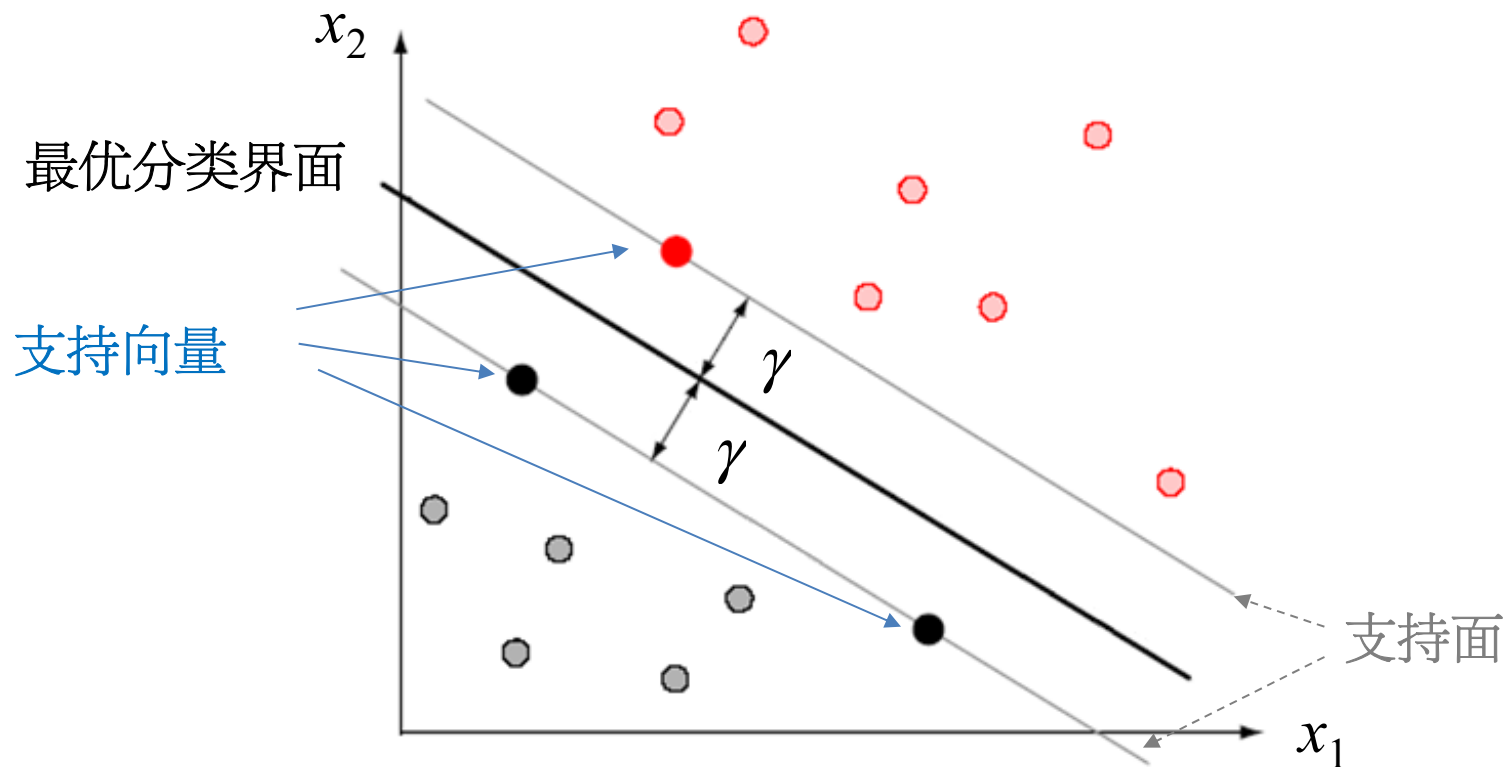
- 样本集与分类界面之间的间隔  $\gamma$  定义为样本与分类界面之间几何间隔的最小值。

# 最优线性判别分类器

---

- **最优分类界面**：给定线性可分样本集，能够将样本分开的**最大几何间隔**超平面，也即是在所有可正确区分样本的超平面中距离训练样本最远的。
- **支持面**：最优分类界面两侧各一个包含距离分类界面最近的样本并与分类界面平行的超平面。
- **支持向量**（support vector）：距离最优分类界面最近的训练样本。
- **支持向量机**（support vector machine, SVM）：使用支持向量定义最优分类界面的分类方法。

# 最优线性判别分类器



- 问题：如何寻找支持向量并确定最优分类界面？

# 支持向量机的学习

- 首先考虑线性可分的情况。给定两个类别的训练样本集  $D = \{(\mathbf{x}_1, z_1), \dots, (\mathbf{x}_n, z_n)\}$ ,  $\mathbf{x}_i$  为训练样本的特征矢量,  $z_i$  为样本的类别标号:

$$z_i = \begin{cases} +1, & \mathbf{x}_i \in \omega_1 \\ -1, & \mathbf{x}_i \in \omega_2 \end{cases}$$

- 能够将样本线性分开的分类界面满足:

$$z_i(\mathbf{w}^T \mathbf{x}_i + w_0) > 0, \quad i = 1, \dots, n$$

类似感知器的规范化

- 设  $b_{\min} = \min_{1 \leq j \leq n} [z_j(\mathbf{w}^T \mathbf{x}_j + w_0)] > 0$  为训练样本中距  
离分类超平面最近的函数间隔, 则有

$$z_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq b_{\min}$$



# 支持向量机的学习

---

- 当权值矢量  $\mathbf{w}$  和偏置  $w_0$  同时乘以  $1/b_{\min}$  时，超平面位置不变。因此超平面需要满足的条件变为

$$z_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i = 1, \dots, n$$

- 即，可以通过调权值矢量  $\mathbf{w}$  和偏置  $w_0$  将样本集的最小函数间隔调整为1。
- 当函数间隔为1时，样本集到分类界面的几何间隔为  $\gamma = \frac{1}{\|\mathbf{w}\|}$ 。因此最大化  $\gamma$  即为最小化  $\|\mathbf{w}\|$ 。

# 支持向量机的学习

- 训练样本集可分条件下，SVM学习最优分类超平面的优化准则和约束条件：

$$\min_{\mathbf{w}, w_0} J_{SVM}(\mathbf{w}, w_0) = \frac{1}{2} \|\mathbf{w}\|^2$$

$$\text{约束: } z_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1, \quad i = 1, \dots, n$$

- 根据上述有约束的优化问题构建Lagrange函数：

见附录B.6，“不等式约束优化问题”

$$L(\mathbf{w}, w_0, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i [z_i(\mathbf{w}^T \mathbf{x}_i + w_0) - 1]$$

其中  $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_n)^T$ ， $\alpha_i \geq 0$  是Lagrange系数。

# 支持向量机的学习

- 先考虑Lagrange函数与  $\alpha$  的关系。当  $L(\mathbf{w}, w_0, \alpha)$  取最大值时，函数求和式中的两个乘积项  $\alpha_i$  和  $z_i(\mathbf{w}^T \mathbf{x}_i + w_0) - 1$  必有一项为0，所以

$$\max_{\alpha} L(\mathbf{w}, w_0, \alpha) = J_{SVM}(\mathbf{w}, w_0) = \frac{1}{2} \|\mathbf{w}\|^2$$

- 原始优化问题等价于一个min-max问题，

详见课本  
143页

$$\min_{\mathbf{w}, w_0} J_{SVM}(\mathbf{w}, w_0) = \min_{\mathbf{w}, w_0} \max_{\alpha} L(\mathbf{w}, w_0, \alpha) = \max_{\alpha} \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \alpha)$$

- 以下首先计算Lagrange函数针对  $\mathbf{w}$  和  $w_0$  的优化问题。

# 支持向量机的学习

---

- 对  $L(\mathbf{w}, w_0, \alpha)$  求关于  $\mathbf{w}$  和  $w_0$  的导数:

$$\frac{\partial L(\mathbf{w}, w_0, \alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i = 0$$

$$\frac{\partial L(\mathbf{w}, w_0, \alpha)}{\partial w_0} = -\sum_{i=1}^n \alpha_i z_i = 0$$

- 因此可得

$$\mathbf{w} = \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i$$

$$\sum_{i=1}^n \alpha_i z_i = 0$$

# 支持向量机的学习

---

- 将上两式回代入Lagrange函数中：

$$\begin{aligned} L(\mathbf{w}, w_0, \boldsymbol{\alpha}) &= \frac{1}{2} \left( \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i \right)^T \left( \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i \right) \\ &\quad - \sum_{i=1}^n [\alpha_i z_i \left( \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i \right)^T \mathbf{x}_i + \alpha_i z_i w_0 - \alpha_i] \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j \mathbf{x}_i^T \mathbf{x}_j \end{aligned}$$

- 此时Lagrange函数只与优化矢量  $\boldsymbol{\alpha}$  相关，而与  $\mathbf{w}$  和  $w_0$  无关。

# 支持向量机的学习

- 因此SVM的原始优化问题可以转化为一个对偶优化问题：

$$\begin{aligned}\max_{\alpha} L(\mathbf{w}, w_0, \alpha) &= \max_{\alpha} L(\alpha) \\ &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j \mathbf{x}_i^T \mathbf{x}_j\end{aligned}$$

约束：  $\alpha_i \geq 0, i = 1, \dots, n$

$$\sum_{i=1}^n \alpha_i z_i = 0$$

线性不等式  
约束条件下的  
二次优化问题

注意：关于对偶优化的约束问题，详见课本143-144页

# 支持向量机的学习

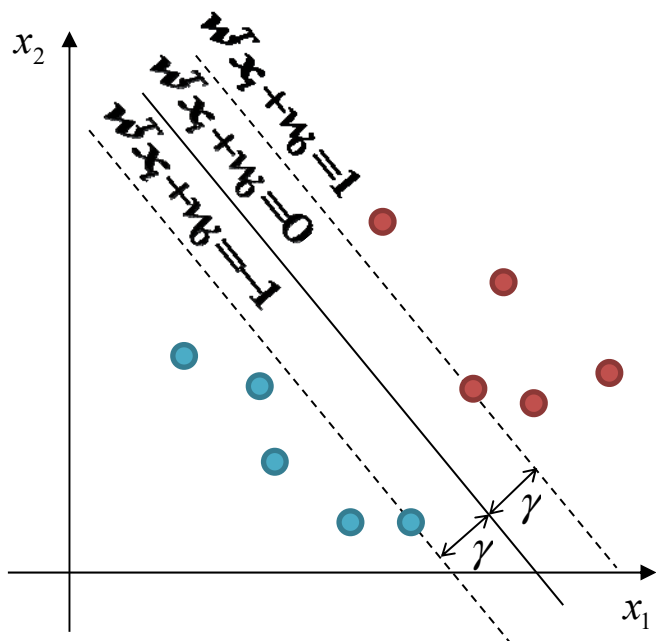
---

- SVM一般求解的是上页的对偶优化问题，因为：
  - 1) 对偶问题不直接优化权值矢量，与样本的特征维数无关，只与样本的数量有关。当样本特征维数很高时，对偶问题更容易求解。
  - 2) 对偶优化中，训练样本以任意两个矢量内积的形式出现。因此只要计算矢量间的内积，而不需要知道样本的每一维特征就可以进行求解。
- 以上两个特点可以很容易地将“核函数”引入到算法中，实现非线性的SVM分类。

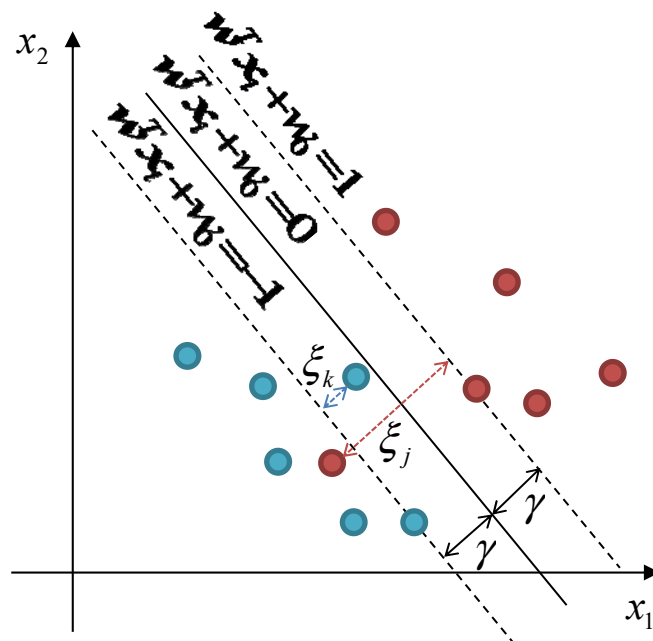
# 支持向量机的学习

- 在**线性不可分**的情况下，需要引入松弛变量  $\xi_i$ ，使得不等式变为

$$z_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i, \quad \xi_i \geq 0$$



线性可分，硬间隔



线性不可分，软间隔



# 支持向量机的学习

---

- 训练样本不可分的情况下，也希望分类器尽可能正确识别尽量多的样本，即希望尽可能多的松弛变量  $\xi_i = 0$ 。
- 非零松弛变量的数量较难直接优化，一般转而优化松弛变量之和  $\sum_{i=1}^n \xi_i$ 。
- 目标函数需要同时考虑两方面的优化：
  - 1) 分类界面与样本集之间的间隔，
  - 2) 松弛变量之和。

# 支持向量机的学习

---

- 优化问题:

$$\min_{\mathbf{w}, w_0} J_{SVM}(\mathbf{w}, w_0) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

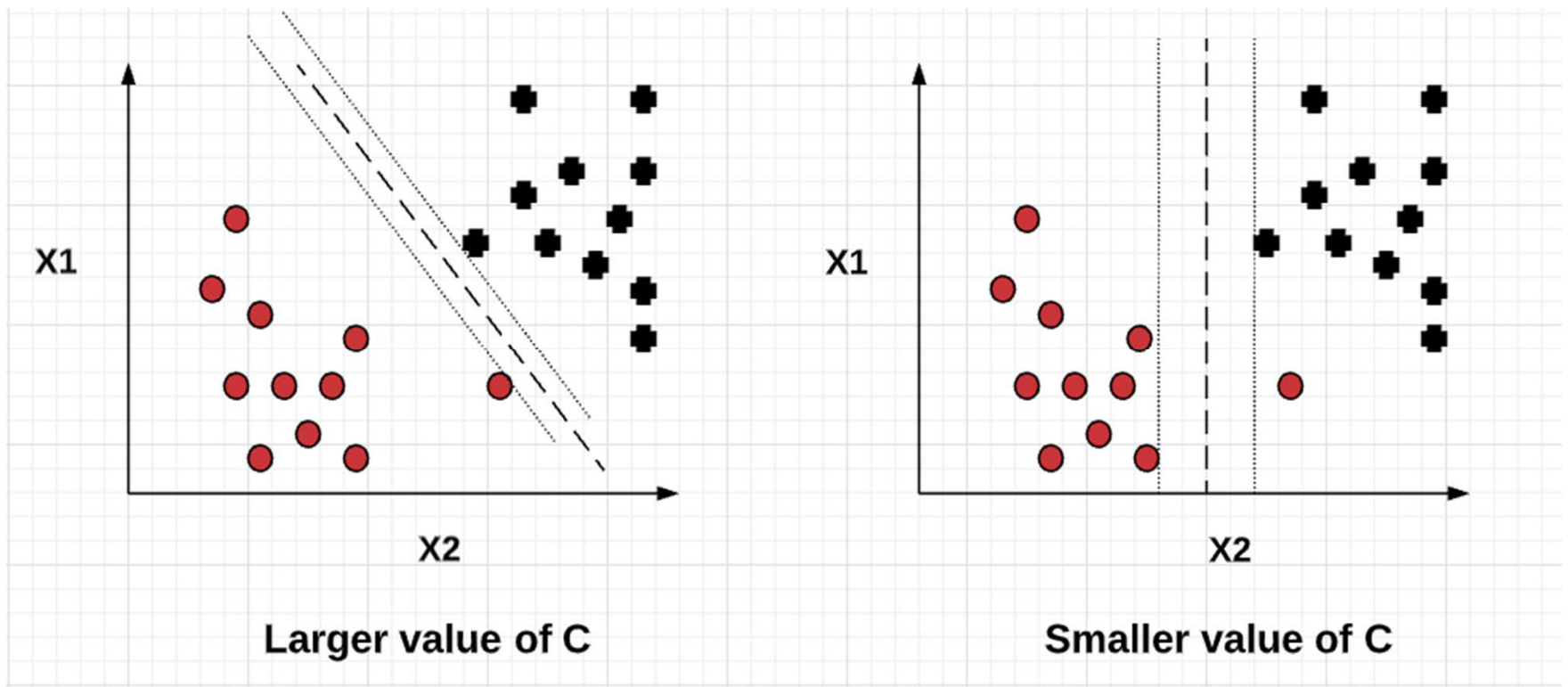
$$\text{约束: } z_i(\mathbf{w}^T \mathbf{x}_i + w_0) \geq 1 - \xi_i, \quad i = 1, \dots, n$$

$$\xi_i \geq 0, \quad i = 1, \dots, n$$

其中参数  $C > 0$  用以协调两个目标在总体目标中的比例。  $C$  值越大表示希望更少的样本被错误识别，  $C$  值越小表示希望分类界面与样本集之间的间隔越大。

# 支持向量机的学习

- 参数  $C$  的变化对分类器的影响:



# 支持向量机的学习

---

- 类似地，可以得到线性不可分情况下的对偶优化问题：

$$\max_{\alpha} L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j \mathbf{x}_i^T \mathbf{x}_j$$

约束：  $C \geq \alpha_i \geq 0, i = 1, \dots, n$

$$\sum_{i=1}^n \alpha_i z_i = 0$$

详见课本  
145-146页。

# 支持向量机的学习

---

- 线性SVM 分类器的学习，就是采用二次规划算法对上页对偶优化问题的求解，得到最优系数  $\alpha$ 。
- 最优化方法的研究已经证明此类问题属于凸规划问题，存在唯一的最优解，并且可由常用的二次规划算法（如内点法、有效集法、椭球算法等）求解（略）。

# 支持向量机的学习

- 不等式约束条件下的二次优化问题解法的基础是 Karush-Kuhn-Tucker (KKT) 定理。 详见课本236页 (附录B.6)
- 线性可分情况下，根据KKT定理，有：

$$\begin{cases} z_i(\mathbf{w}^T \mathbf{x}_i + w_0) > 1, \alpha_i = 0 \\ z_i(\mathbf{w}^T \mathbf{x}_i + w_0) = 1, \alpha_i > 0 \end{cases}$$

其中满足上面第二个条件的  $\mathbf{x}_i$  是支持向量。

- 线性不可分的情况下，略为复杂，详见课本146页公式(6.34)和上下文。

# 支持向量机的学习

---

- 以下介绍计算得到最优系数  $\alpha$  之后，如何计算  $\mathbf{w}$  和  $w_0$  以构造最有判别函数。
- 根据找到的支持向量  $\mathbf{x}_i$  以及相应的Lagrange乘子  $\alpha_i$ ，计算权值矢量  $\mathbf{w}$ :

$$\mathbf{w} = \sum_{i=1}^n \alpha_i z_i \mathbf{x}_i$$

- 偏置  $w_0$  可以用支持向量满足的条件从任意一个支持向量  $\mathbf{x}_i$  求得:

$$z_i (\mathbf{w}^T \mathbf{x}_i + w_0) = 1 \rightarrow w_0 = z_i - \mathbf{w}^T \mathbf{x}_i$$

# 支持向量机的学习

---

- 例：线性SVM分类器在参数  $C = 4$  的条件下得到如下训练结果：
  - 第一类支持向量：  $\mathbf{x}_1 = (1,0)^T$ ,  $\mathbf{x}_2 = (2,2)^T$
  - 第二类支持向量：  $\mathbf{x}_3 = (1,2)^T$
  - 对应 Lagrange 系数：  $\alpha_1 = 0.5$ ,  $\alpha_2 = 2$ ,  $\alpha_3 = 2.5$
- 写出对应的线性判别函数，并判别下列样本的类别，以及相对于支持面和判别界面的位置关系：  
 $\mathbf{y}_1 = (0.5,2)^T$ ,  $\mathbf{y}_2 = (1.5,1)^T$ ,  $\mathbf{y}_3 = (1,1.5)^T$



# 支持向量机的学习

---

- 计算权值矢量  $\mathbf{w}$ :

$$\mathbf{w} = \sum_{i=1}^3 \alpha_i z_i \mathbf{x}_i = 0.5 \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 2 \times \begin{pmatrix} 2 \\ 2 \end{pmatrix} - 2.5 \times \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

- 选择  $\mathbf{x}_1$  作为支持向量, 计算偏置  $w_0$ :

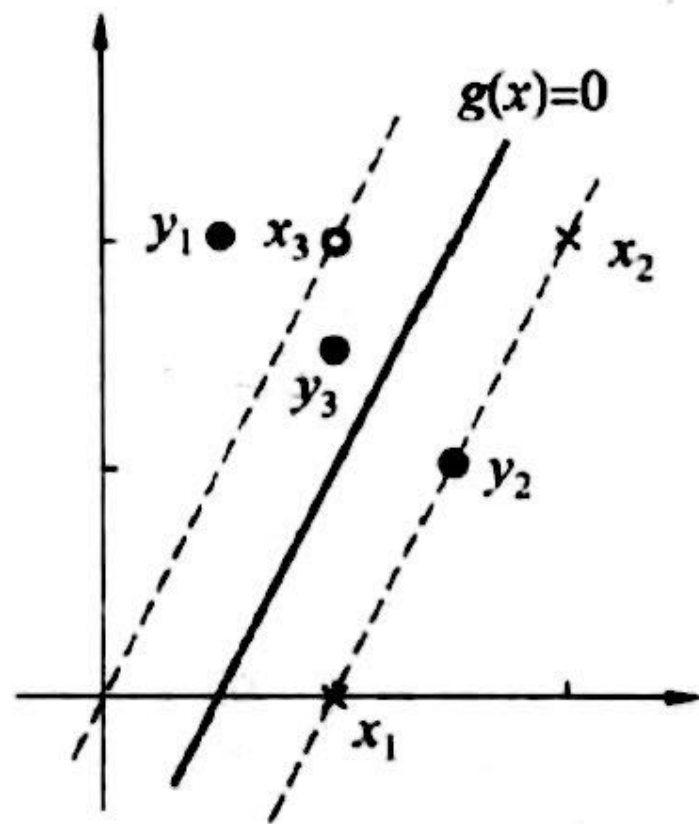
$$w_0 = z_1 - \mathbf{w}^T \mathbf{x}_1 = 1 - (2, -1) \times \begin{pmatrix} 1 \\ 0 \end{pmatrix} = -1$$

- 因此线性判别函数为

$$g(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + w_0 = 2x_1 - x_2 - 1$$

# 支持向量机的学习

- $g(y_1) = -2$   
因为  $g(y_1) < -1$ ，所以  $y_1$  属于第2类，且在第2类支持面之外。
- $g(y_2) = 1$   
 $y_2$  属于第1类，且在第1类的支持面上。
- $g(y_3) = -0.5$   
因为  $0 > g(y_3) > -1$ ，所以  $y_3$  属于第2类，且在第2类支持面与分类界面之间。



# 核函数

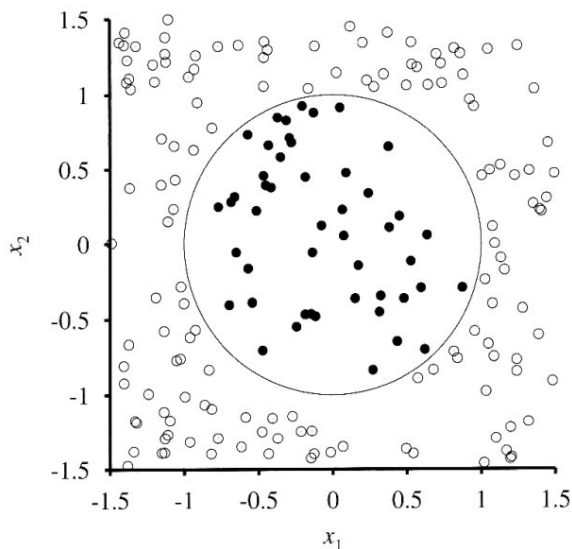
---

- 支持向量机本身是线性判别分类器，但可以扩展为非线性分类器。
- 广义线性判别函数通过定义一个低维特征到高维特征的映射，在高维特征空间学习一个线性判别函数，就可得到低维空间中的非线性判别函数。
- 支持向量机使用类似思路实现非线性判别，通过使用核函数（Kernel Function）避免在高维空间中的计算。

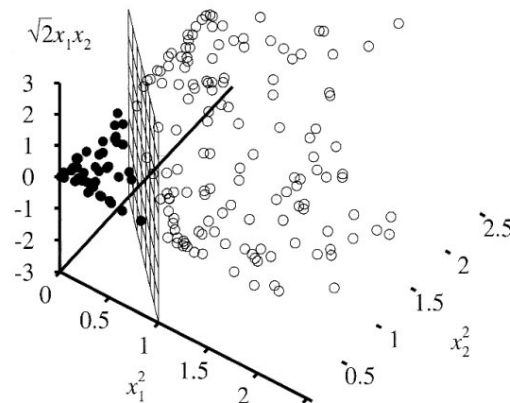
# 核函数

- 例：定义建立一个  $\mathbf{R}^2 \rightarrow \mathbf{R}^3$  的非线性映射  $\Phi$ :

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbf{R}^2 \rightarrow \mathbf{y} = \Phi(\mathbf{x}) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \in \mathbf{R}^3$$



线性不可分



线性可分

# 核函数

---

- 计算  $\mathbf{R}^3$  空间中两个矢量  $\mathbf{y}_i$  和  $\mathbf{y}_j$  的内积:

$$\begin{aligned}\mathbf{y}_i^T \mathbf{y}_j &= \boldsymbol{\Phi}^T(\mathbf{x}_i) \boldsymbol{\Phi}(\mathbf{x}_j) = [x_{i1}^2, \sqrt{2}x_{i1}x_{i2}, x_{i2}^2] \begin{bmatrix} x_{j1}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{bmatrix} \\ &= (x_{i1}x_{j1} + x_{i2}x_{j2})^2 = (\mathbf{x}_i^T \mathbf{x}_j)^2\end{aligned}$$

- 如果定义  $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2$ , 那么  $\mathbf{y}_i^T \mathbf{y}_j$  不必在计算  $\mathbf{x}_i$  和  $\mathbf{x}_j$  的映射  $\mathbf{y}_i$  和  $\mathbf{y}_j$  后再计算内积, 而是可以由  $\mathbf{y}_i^T \mathbf{y}_j = K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j)^2$  计算得出。

# 核函数

---

- **核函数方法**：通过在  $R^d$  空间（输入空间）定义一个核函数  $K(\mathbf{x}_i, \mathbf{x}_j)$ ，等价地计算经映射后  $R^r$  空间（特征空间）中两个矢量  $\Phi(\mathbf{x}_i)$  和  $\Phi(\mathbf{x}_j)$  的内积，从而可以不必真正定义和计算非线性映射  $\Phi$ 。
- 将核函数的方法应用于某个线性算法，将其转化为非线性算法，需要满足两个条件：
  - 1) 核函数必须能够等价于映射之后空间中的矢量内积；
  - 2) 算法不需计算矢量本身，只需计算两个矢量内积。

# 核函数

- 什么样的函数可以作为核函数，能够与映射之后空间中矢量的内积对应？
- Mercer定理**：令  $\mathbf{x}, \mathbf{y} \in \mathbf{R}^d$ ，如果一个对称函数  $K(\mathbf{x}, \mathbf{y})$  针对任意的平方可积函数  $g(\mathbf{x})$  满足半正定条件，即
$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \quad \text{Mercer条件}$$
$$\int g^2(\mathbf{x}) d\mathbf{x} < +\infty$$

则存在一个由  $\mathbf{R}^d$  空间到Hilbert空间  $H$  的映射  $\Phi: \mathbf{x} \rightarrow \Phi(\mathbf{x}) \in H$ ，使得  $K(\mathbf{x}, \mathbf{y})$  等于  $\Phi(\mathbf{x})$  和  $\Phi(\mathbf{y})$  空间  $H$  中的内积。

# 核函数

---

- 只要定义的函数  $K(\mathbf{x}, \mathbf{y})$  满足Mercer条件就可以作为核函数，与某个  $\Phi$  映射后空间中内积对应。
- 模式识别中常用的核函数：

Gaussian: 
$$K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{\sigma}\right)$$

Polynomial: 
$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T \mathbf{y} + 1)^d$$

Sigmoidal: 
$$K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}^T \mathbf{y} + \theta)$$

Inverse Multiquadric: 
$$K(\mathbf{x}, \mathbf{y}) = \frac{1}{\sqrt{\|\mathbf{x} - \mathbf{y}\|^2 + c^2}}$$



# 非线性支持向量机

- 如何在线性SVM中使用核函数将其转化为非线性分类器?
- 引入非线性映射  $\Phi$  后, SVM的学习过程主要是优化下列函数:

$$\max_{\alpha} L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j \Phi^T(\mathbf{x}_i) \Phi(\mathbf{x}_j)$$

- 使用核函数可以将上式变为

$$\max_{\alpha} L(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j z_i z_j K(\mathbf{x}_i, \mathbf{x}_j)$$

# 非线性支持向量机

---

- 与前类似，可以求解得到  $\alpha$ ，进而计算权值

$$\mathbf{w} = \sum_{i=1}^n \alpha_i z_i \Phi(\mathbf{x}_i)$$

- 但是核方法没有定义  $\Phi$ ，无法直接计算  $\Phi(\mathbf{x}_i)$ 。
- 考虑测试样本  $\mathbf{x}$  在特征空间的线性判别函数为

$$\begin{aligned} g(\mathbf{x}) &= \mathbf{w}^T \Phi(\mathbf{x}) + w_0 = \left[ \sum_{i=1}^n \alpha_i z_i \Phi(\mathbf{x}_i) \right]^T \Phi(\mathbf{x}) + w_0 \\ &= \sum_{i=1}^n \alpha_i z_i K(\mathbf{x}, \mathbf{x}_i) + w_0 \end{aligned}$$

- 即，无需计算  $\mathbf{w}$  也可根据上式得到判别器输出。

# 非线性支持向量机

---

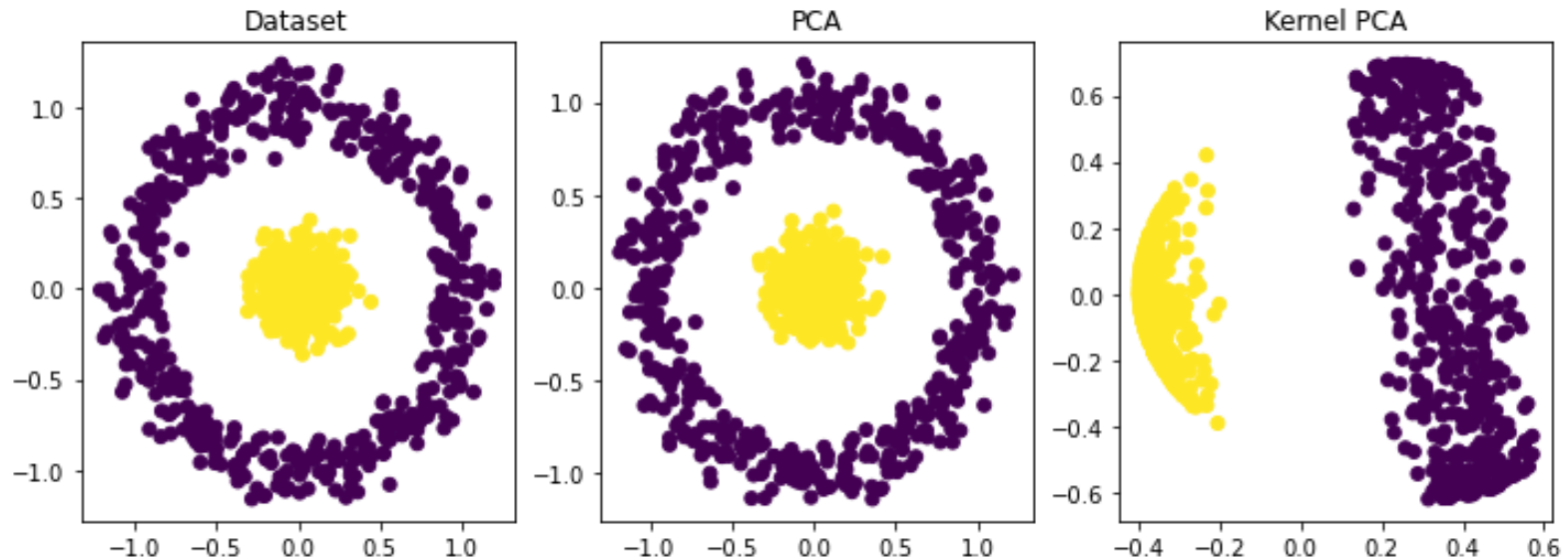
- 偏置  $w_0$  同样可以从某个支持向量  $\mathbf{x}_j$  计算：

$$w_0 = z_j - \sum_{i=1}^n \alpha_i z_i K(\mathbf{x}_j, \mathbf{x}_i)$$

- 因此，通过引入核函数可以实现非线性的SVM分类。非线性SVM无法像线性SVM一样直接计算权值矢量  $\mathbf{w}$ ，而是需要识别的时候在线性判别函数中利用核函数计算测试样本与训练样本在特征空间中的内积，从而得到判别函数输出。

# 核方法其他应用

- 核方法不仅用于SVM中，事实上它已经被用于将多种线性模式识别方法扩展为非线性方法。
- Kernel PCA：利用核函数，将原始特征“映射到高维空间中”进行PCA，从而实现非线性降维。



# 支持向量机的讨论

---

- **最优性**：SVM在对测试样本分类性能的意义下是“最优分类器”，但这种最优性只是相对于线性分类器而言的；在非线性SVM中，最优性存在于特征空间，而不是输入空间。
- **算法效率**：传统的二次优化算法求解大规模问题存在困难，当训练样本很多时，算法的存储计算量很大。随着序列最小化等系列算法的提出，大规模SVM的效率已经得到了很大的提高。

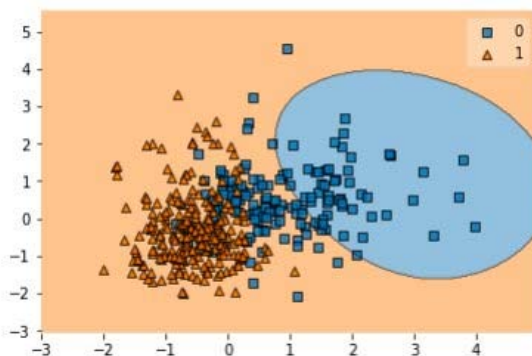
# 支持向量机的讨论

---

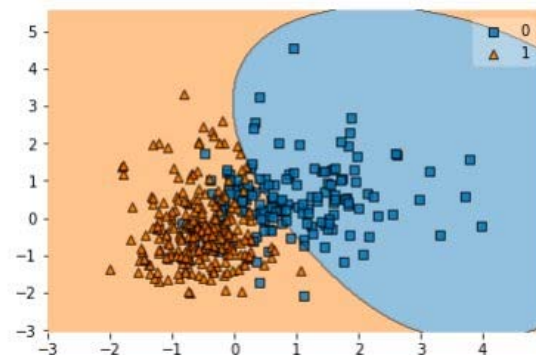
- **收敛性**：无论线性或非线性SVM，优化的都是正定二次函数，存在唯一极值点，无论什么样的初始值都会收敛于最优解，具有良好的收敛性。
- **参数选择**：SVM参数选择主要体现在优化函数的参数  $C$ ，核函数的选择以及相应的核函数参数的选择。不同参数的选择对判别界面的学习结果有着很大影响，实际应用中需根据具体问题进行一定的尝试，选择出一组较优的参数。

# 支持向量机的讨论

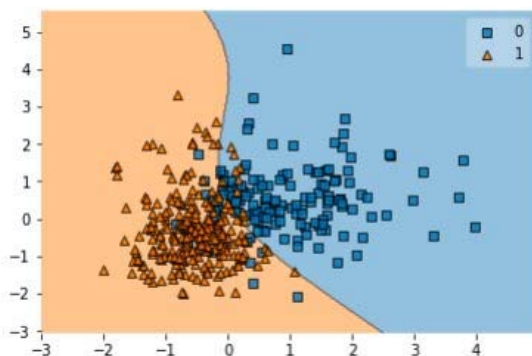
- 参数  $C$  选择



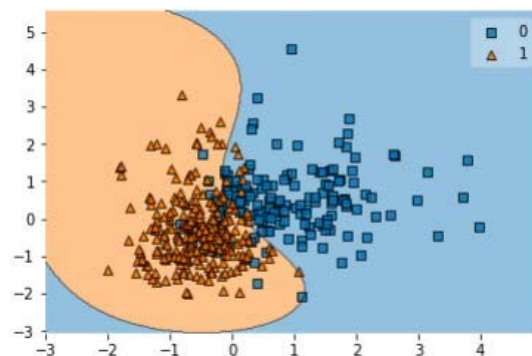
$C = 0.02$   
Accuracy: 81.3%



$C = 0.03$   
Accuracy: 88.3%



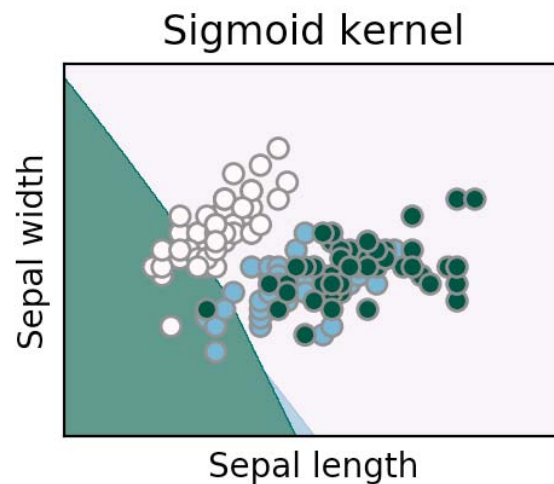
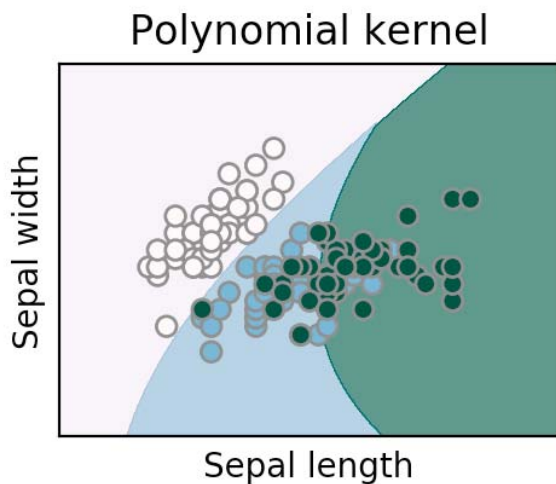
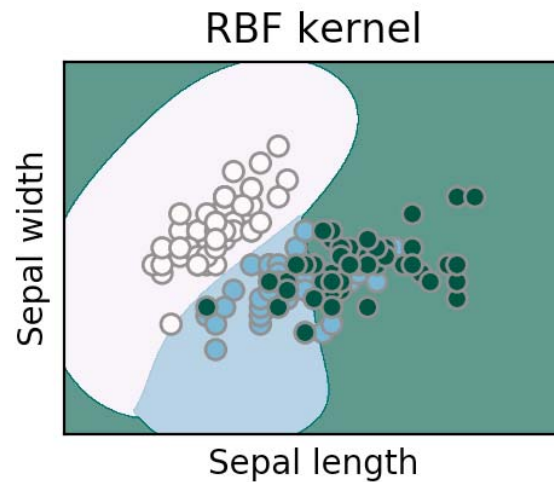
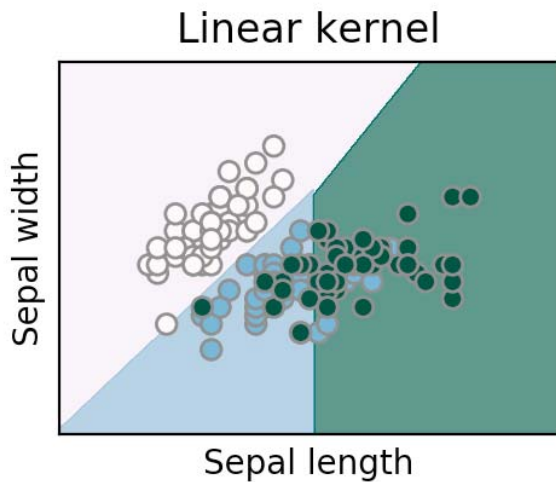
$C = 1.0$   
Accuracy: 90.6%



$C = 10.0$   
Accuracy: 90.1%

# 支持向量机的讨论

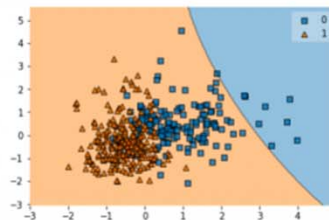
- 核函数选择



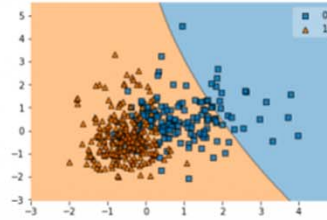


# 支持向量机的讨论

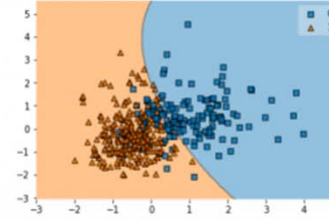
- 参数选择 ( RBF )  $K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{\sigma}\right) = \exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$



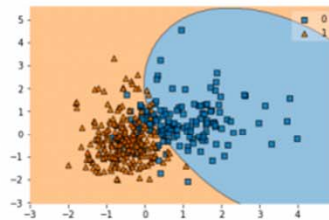
Gamma = 0.008  
Accuracy: 63.7%



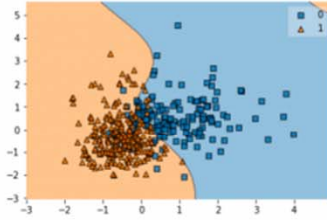
Gamma = 0.01  
Accuracy: 68.4%



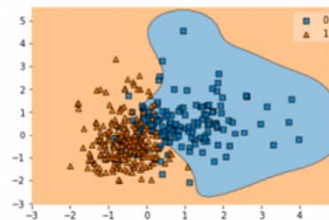
Gamma = 0.05  
Accuracy: 88.9%



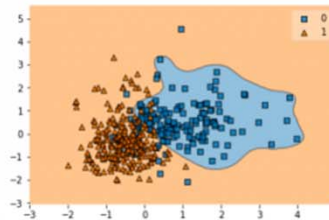
Gamma = 0.1  
Accuracy: 90.1%



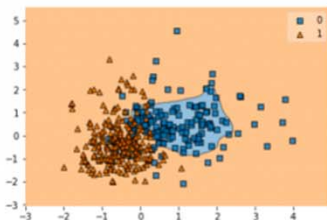
Gamma = 0.5  
Accuracy: 91.6%



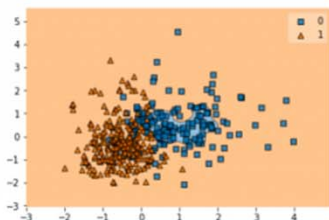
Gamma = 1.0  
Accuracy: 90.1%



Gamma = 3.0  
Accuracy: 88.9%



Gamma = 7.0  
Accuracy: 84.8%



Gamma = 11.0  
Accuracy: 74.9%

# 本章小结

---

- 介绍了线性判别函数分类器的局限和广义线性判别函数分类器的一般原理
- 着重介绍了多层感知器网络的结构和反向传播学习算法
- 简单介绍了多层感知器网络学习算法的改进思路 and 多种改进形式

# 本章小结

---

- 介绍了作为最优线性判别函数分类器的支持向量机的基本原理
- 介绍了线性支持向量机在线性可分与不可分两种情况下的优化函数和学习过程
- 介绍了核函数的基本原理与非线性支持向量机的实现与学习
- 介绍了支持向量机应用中的注意事项