神经网络
Neural Networks
第二章

感知器

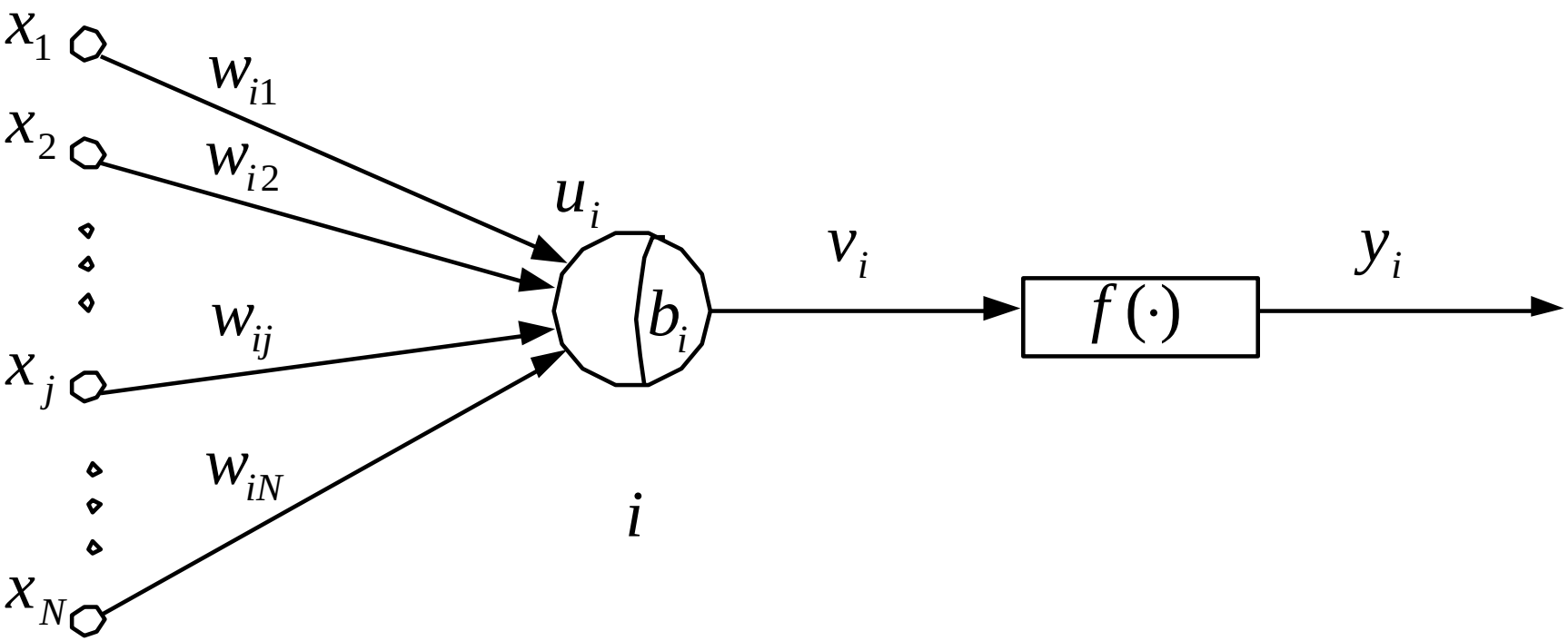
史忠植

中国科学院计算技术研究所
http://www.intsci.ac.cn/

内容提要

- **2.1 感知器的认知观点**
- **2.2 单层感知器**
- **2.3 多层感知器**
- **2.4 学习算法的优化**
- **2.5 最小均方算法**

2.1.1 单层感知器模型



单层感知器模型

感知器模型与MP模型的不同之处是假定神经元的突触权值是可变的，这样就可以进行学习。

$$y = Sgn(\sum_{j=1}^{m} w_{ij}\, x_j + b)$$

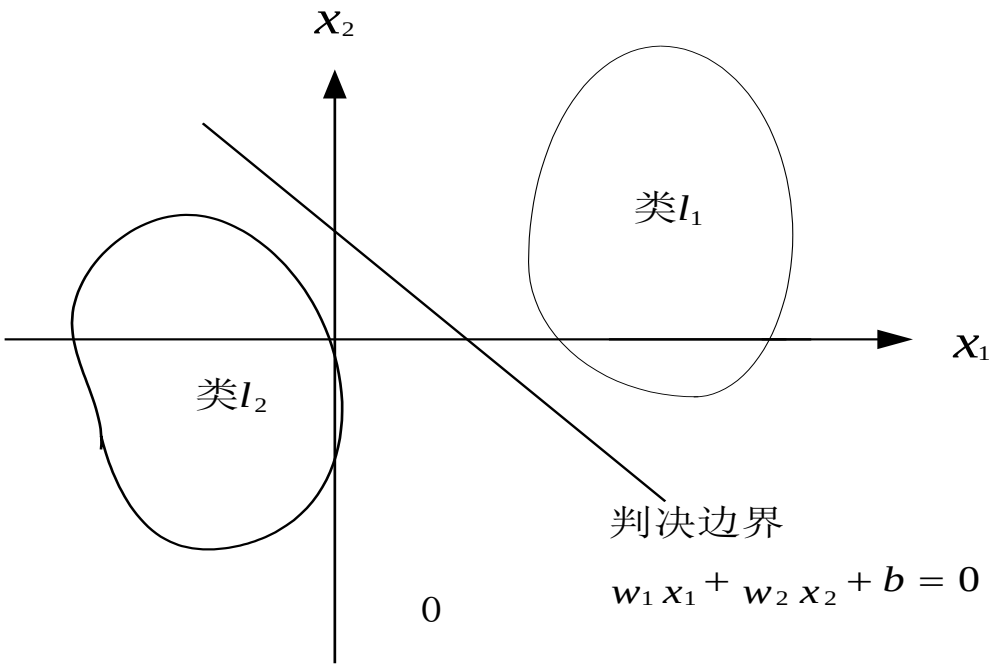
$$y = \begin{cases} 1, & 若(\sum_{j=1}^{m} w_{ij}\, x_j + b) \geq 0 \\ -1, & 若(\sum_{j=1}^{m} w_{ij}\, x_j + b) < 0 \end{cases}$$

单层感知器模型

- 使用单层感知器的目的就是让其对外部输入$x_1, x_2, \ldots, x_m$进行识别分类，单层感知器可将外部输入分为两类l_1和l_2。

$$\sum_{j=1}^{m} w_{ij} x_j + b = 0$$

判别边界



2.1.2 单层感知器的学习算法

- 单层感知器对权值向量的学习算法是基于迭代的思想，通常是采用纠错学习规则的学习算法。
- 为方便起见，将偏差***b***作为神经元突触权值向量的第一个分量加到权值向量中去，那么对应的输入向量也应增加一项，可设输入向量的第一个分量固定为**+1**，这样输入向量和权值向量可分别写成如下的形式：

$$X(n)=\left[+1,x_1(n),x_2(n),\cdots,x_m(n)\right]^T$$

单层感知器的学习算法

$$W(n)=\left[b(n),w_1(n),w_2(n),\cdots,w_m(n)\right]^T$$

- 其中的变量**n**表示迭代次数，其中的**b(n)**可用$w_0(n)$表示，则二值阈值元件的输入可重新写为：

$$v=\sum_{j=0}^{m}w_j(n)x_j(n)=W^T(n)X(n)$$

- 令上式等于零，即可得在**m**维信号空间的单层感知器的判决超平面。

单层感知器的学习算法

- 第一步：设置变量和参量：

  $X(n)= [1, x_1(n), x_2(n), \ldots, x_m(n)]$为输入向量，或称训练样本；

  $W(n)= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]$为权值向量；

  $b(n)$ 为偏差；$y(n)$为实际输出；

  $d(n)$为期望输出；η为学习速率；n为迭代次数。

单层感知器的学习算法

- 第二步：初始化，赋给$W_j(0)$各一个较小的随机非零值，$n = 0$；
- 第三步：对于一组输入样本$X(n)= [1, x_1(n), x_2(n), \ldots, x_m(n)]$，指定它的期望输出（亦称之为导师信号）。

$$if \quad X \in l_1, \ d = 1 \qquad if \quad X \in l_2, \ d = -1$$

- 第四步：计算实际输出：

$$y(n) = Sgn(W^T(n)X(n))$$

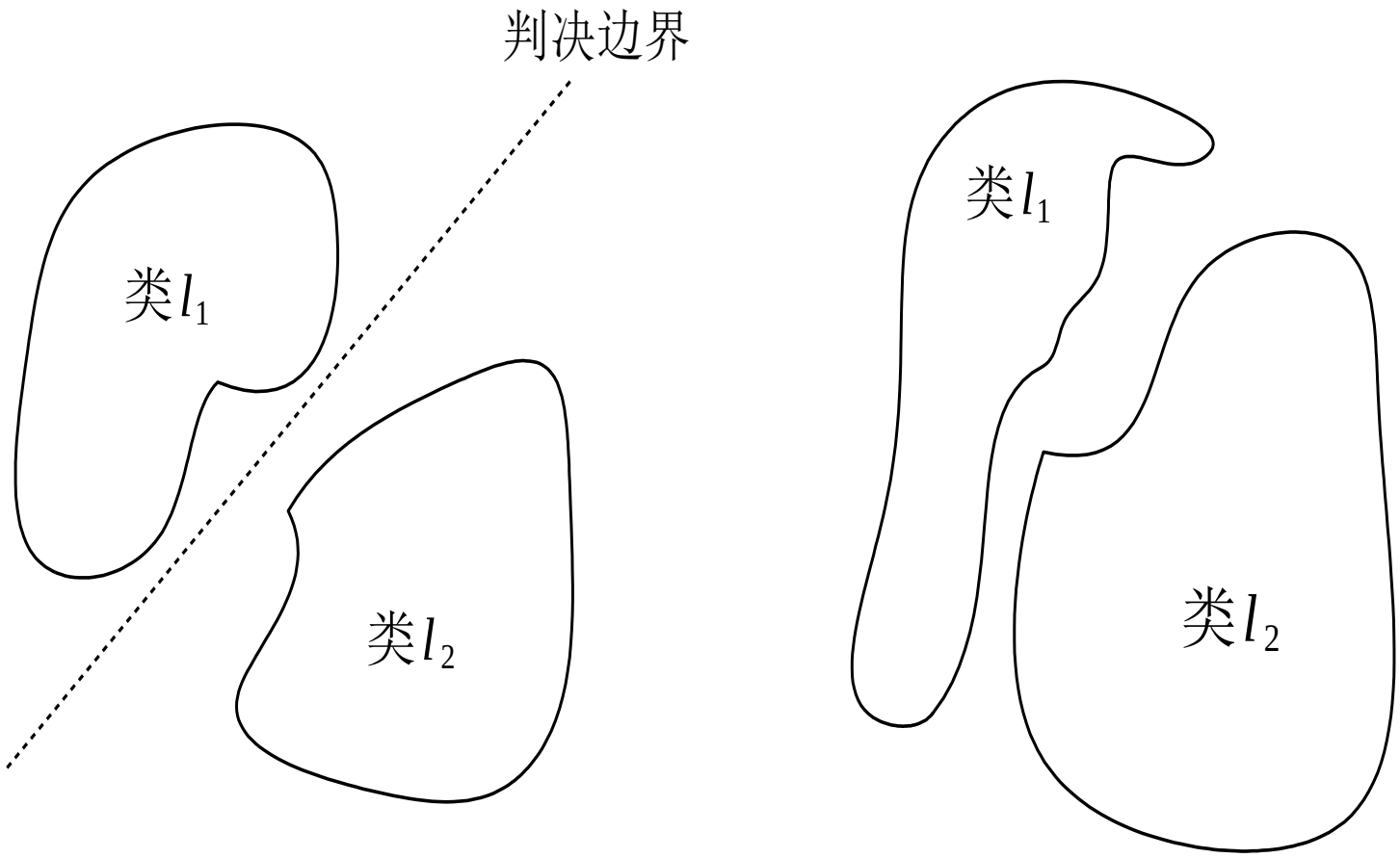
单层感知器的学习算法

- 第五步：调整感知器的权值向量：

$$w(n+1)=w(n)+\eta\left[d(n)-y(n)\right]X(n)$$

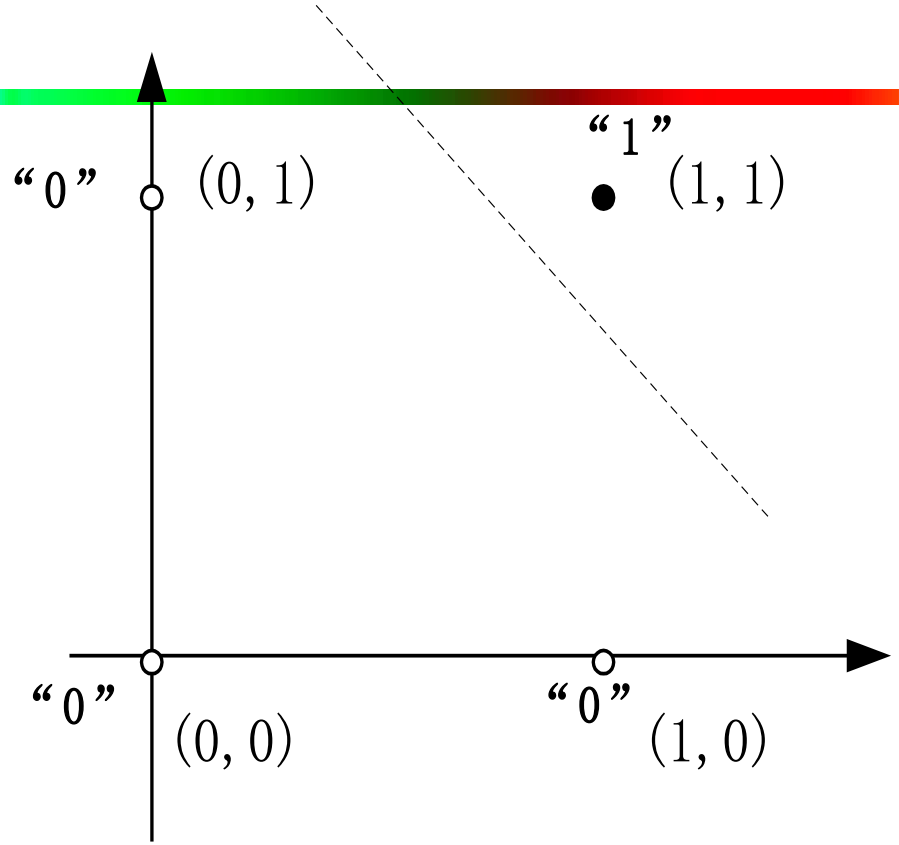
- 第六步：判断是否满足条件，若满足算法结束，若不满足将*n*值增加1，转到第三步重新执行。

对于线性可分的两类模式，单层感知器的学习算法是收敛的。

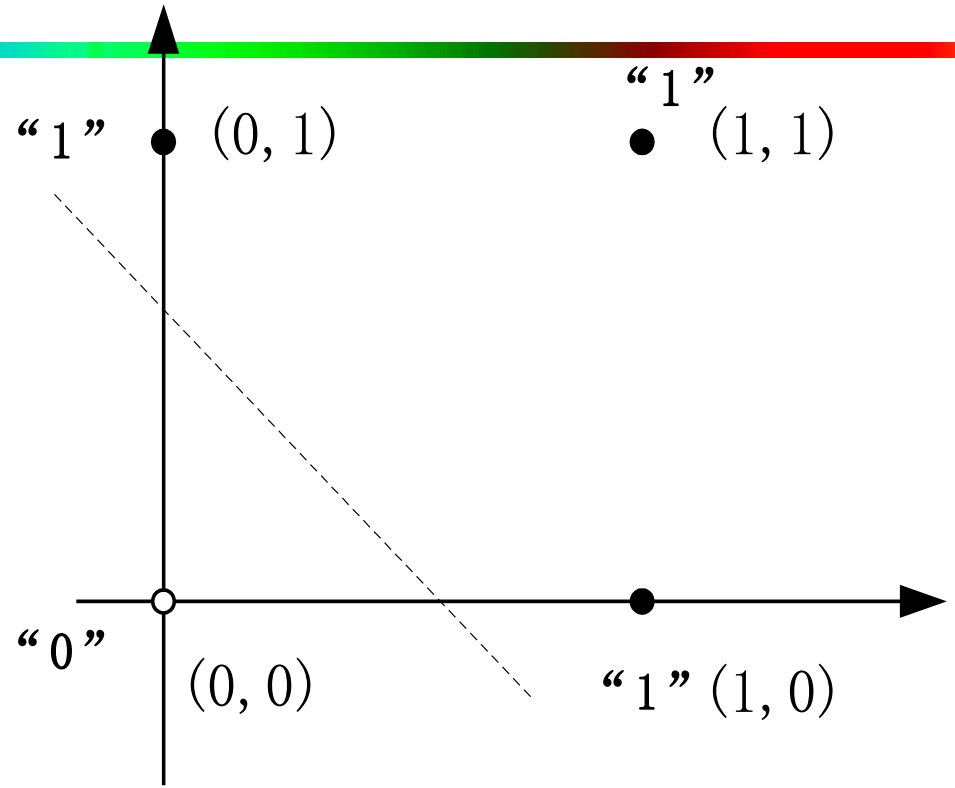


x_1	x_2	$x_1 x_2$	$Y=w_1 x_1+w_2 x_2-b=0$	条件
“与”				
0	0	0	$Y=w_1 \cdot 0+w_2 \cdot 0-b<0$	$b>0$
0	1	0	$Y=w_1 \cdot 0+w_2 \cdot 1-b<0$	$b>w_2$
1	0	0	$Y=w_1 \cdot 1+w_2 \cdot 0-b<0$	$b>w_1$
1	1	1	$Y=w_1 \cdot 1+w_2 \cdot 1-b\geq 0$	$b\leq w_1+w_2$

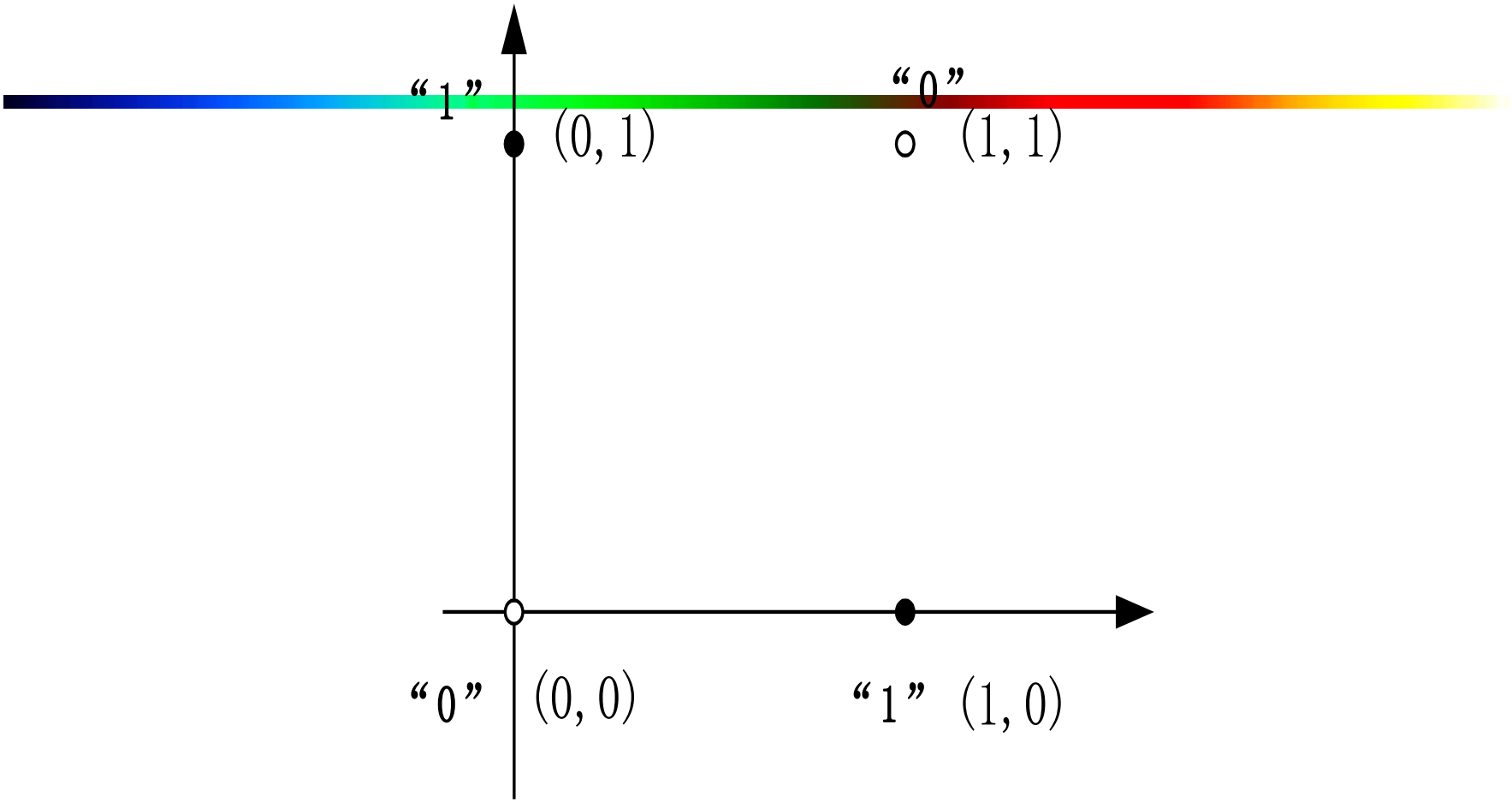
可解。比如取$w_1=1$，$w_2=1$，$b=1.5$。



“与”运算图示，线性可分，可以实现



“或”运算图示，线性可分，可以实现



“异或”运算图示，线性不可分，不可以实现

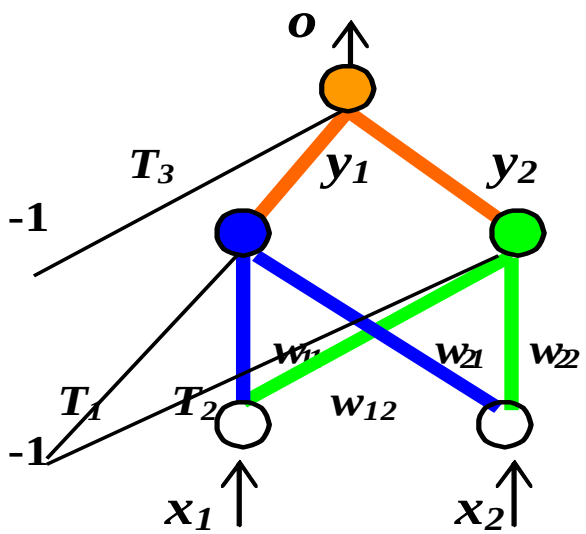
多层感知器

例四 用两计算层感知器解决“异或”问题。

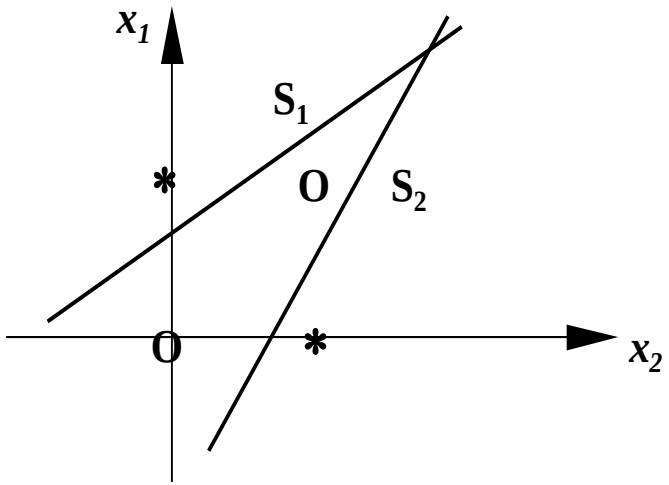
“异或”的真值表

x1	x2	y1	y2	o
0	0	1		
0	1	1		
1	0	0		
1	1	1		

双层感知器



“异或”问题分类



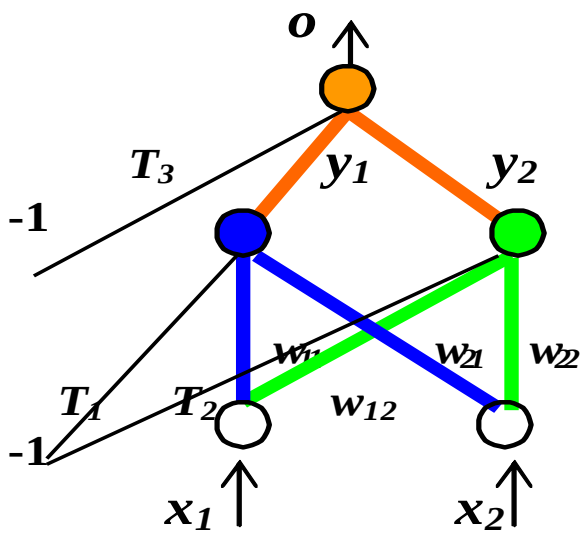
多层感知器

“异或”的真值表

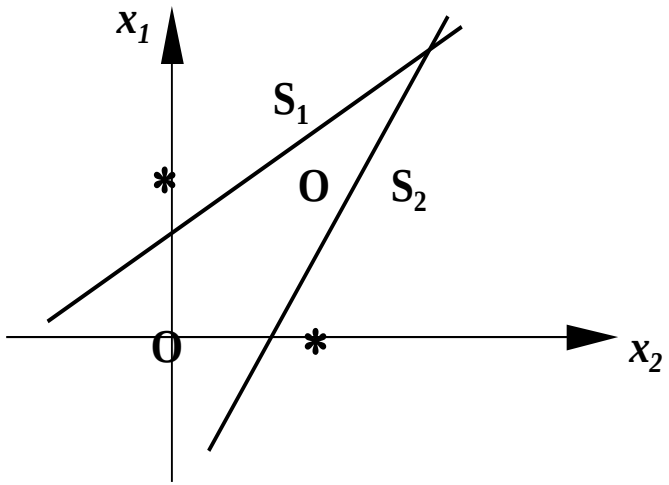
x1	x2	y1	y2	o
0	0		1	
0	1		0	
1	0		1	
1	1		1	

例四 用两计算层感知器解决“异或”问题。

双层感知器



“异或”问题分类



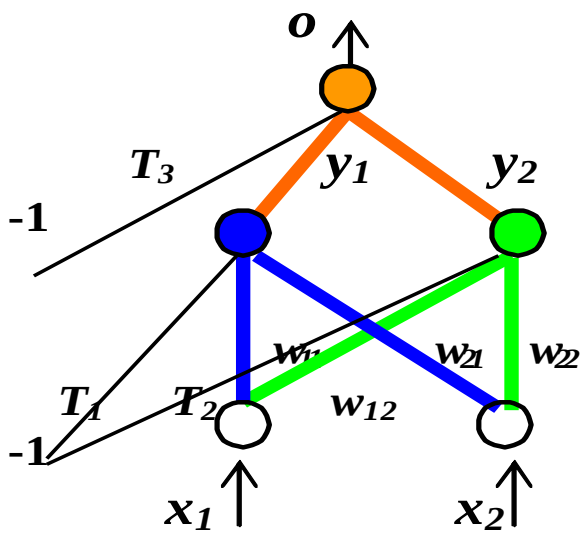
多层感知器

“异或”的真值表

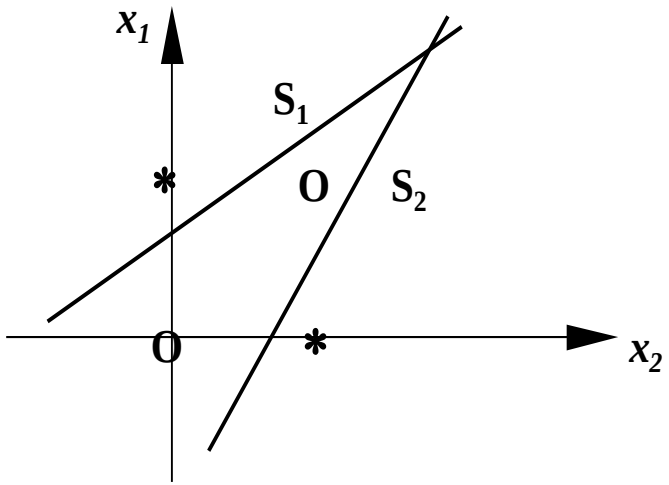
x1	x2	y1	y2	o
0	0	1	1	
0	1	1	0	
1	0	0	1	
1	1	1	1	

例四 用两计算层感知器解决“异或”问题。

双层感知器



“异或”问题分类



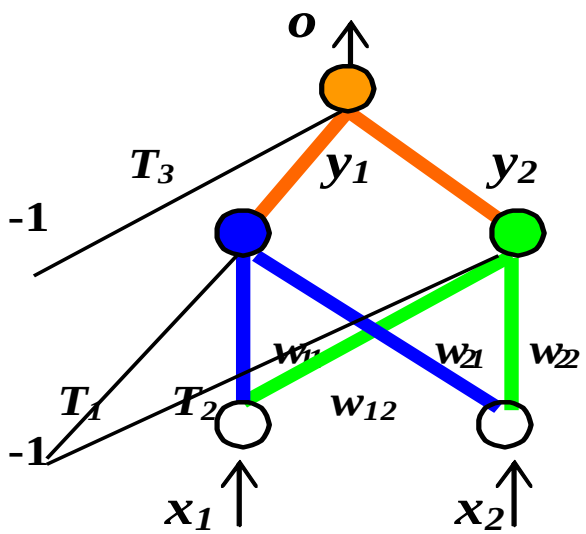
多层感知器

“异或”的真值表

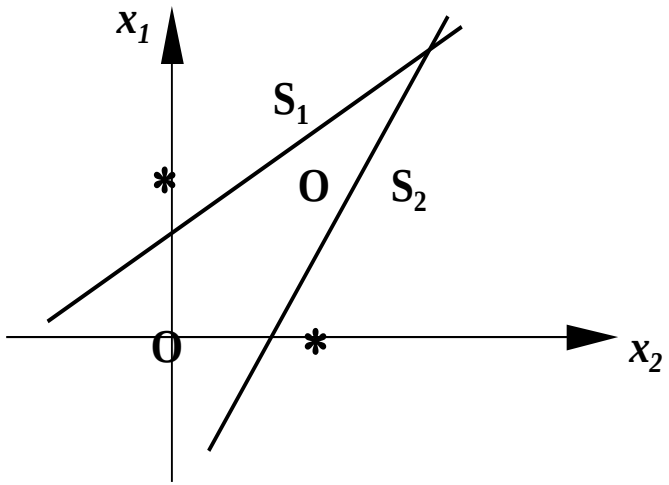
x1	x2		o
0	0		0
0	1		1
1	0		1
1	1		0

例四 用两计算层感知器解决“异或”问题。

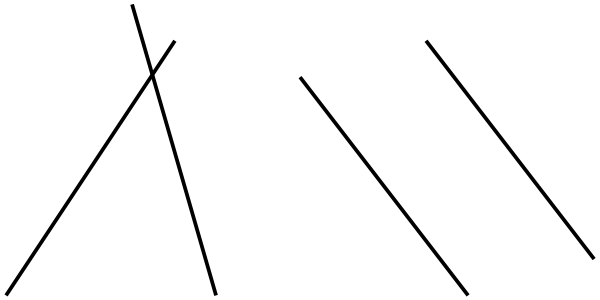
双层感知器



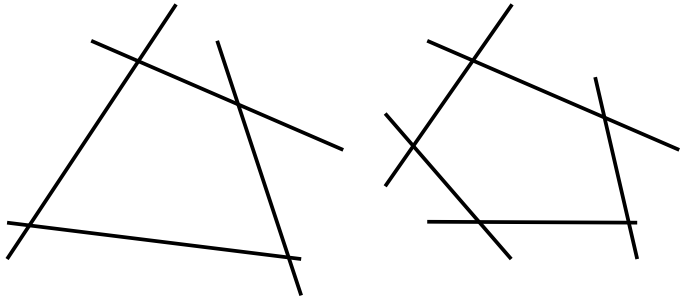
“异或”问题分类



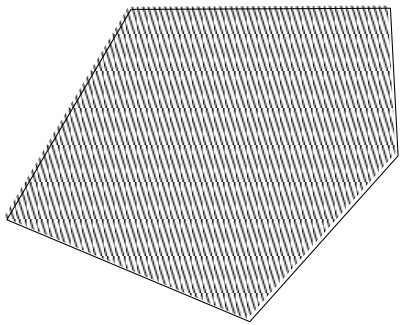
多层感知器

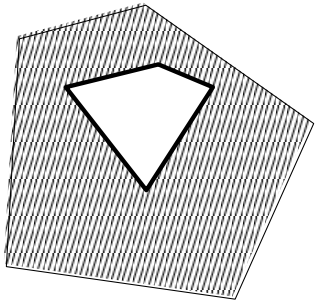


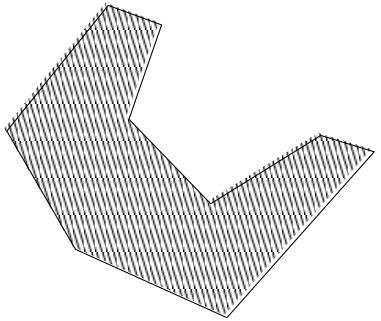
(a)开域



(b)闭域

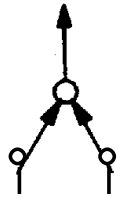
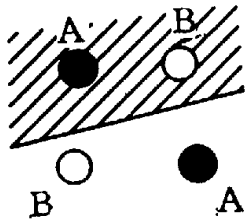
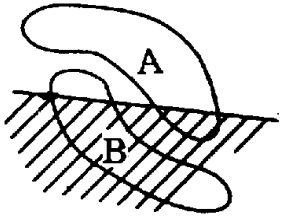
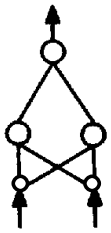
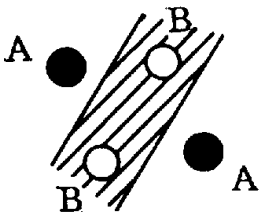
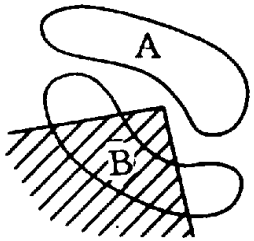
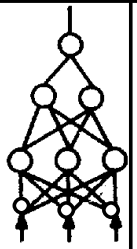
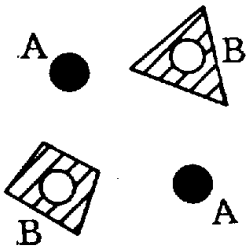
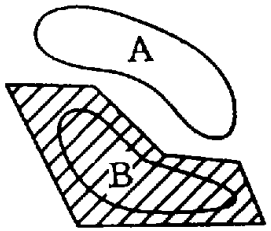
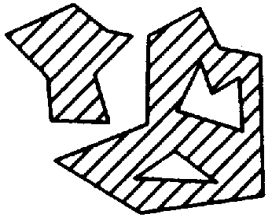




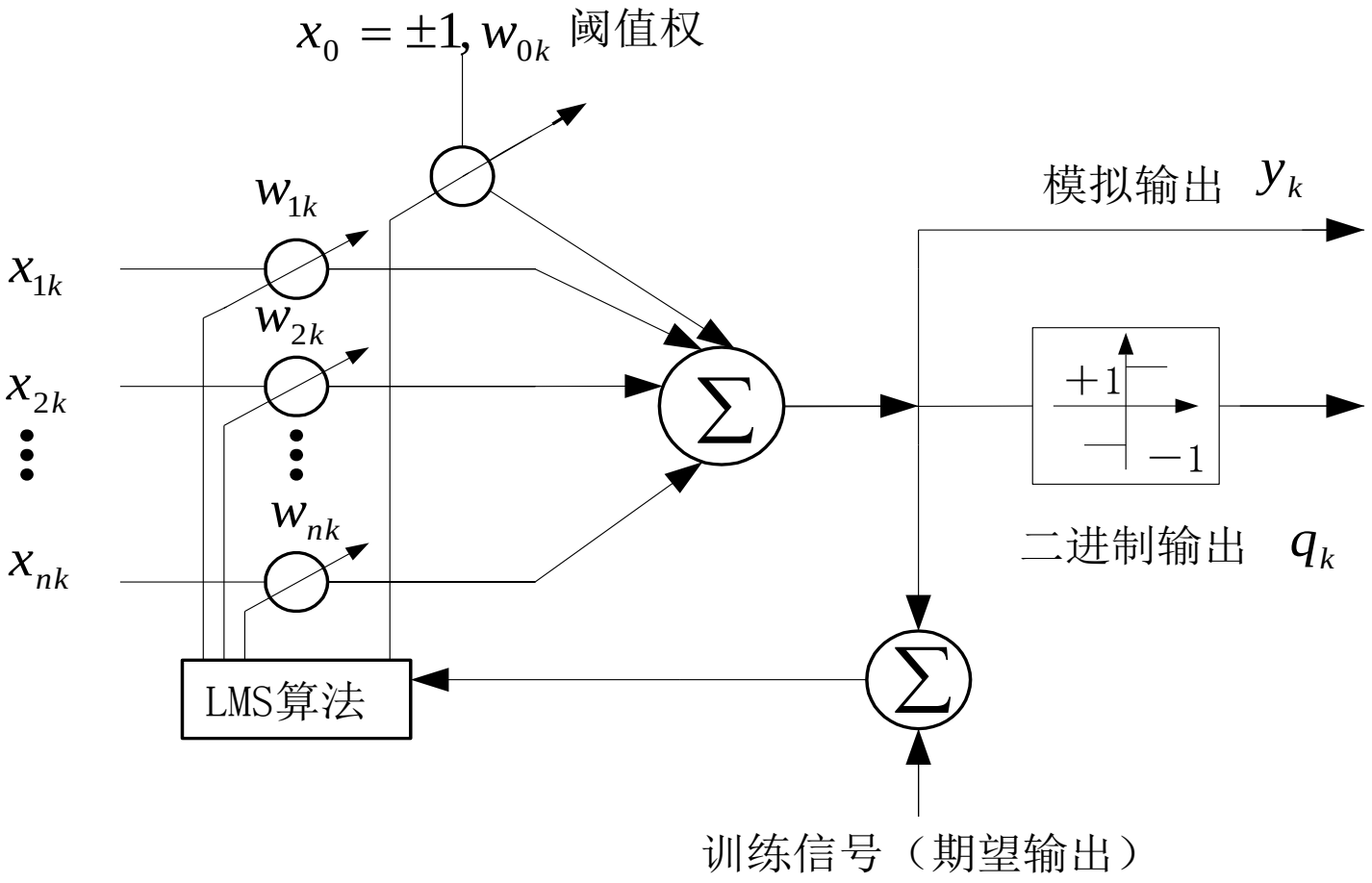


多层感知器

具有不同隐层数的感知器的分类能力对比

感知器结构	异或问题	复杂问题	判决域形状	判决域
无隐层 				半平面
单隐层 				凸　域
双隐层 				任意复杂形状域

2.2 自适应线性元件


$x_0 = \pm 1, w_{0k}$ 阈值权
w_{1k}
x_{1k}
w_{2k}
x_{2k}
w_{nk}
x_{nk}
Σ
模拟输出 y_k
+1
−1
二进制输出 q_k
LMS算法
Σ
训练信号（期望输出）

自适应线性元件

- 神经元*i*的输入信号向量：

$$X_i = [x_{0i}, x_{1i}, \cdots x_{mi}]^T, X_i \text{常取} \pm 1$$

- 突触权值向量：$W_i = [w_{0i}, w_{1i}, \cdots w_{mi}]^T$
- w_{0i}常接有单位输入，用以控制阈值电平。
- 模拟输出：$y_i = X_i^T W = W^T X$
- 二值输出：$q_i = Sgn(y_i)$

自适应线性元件

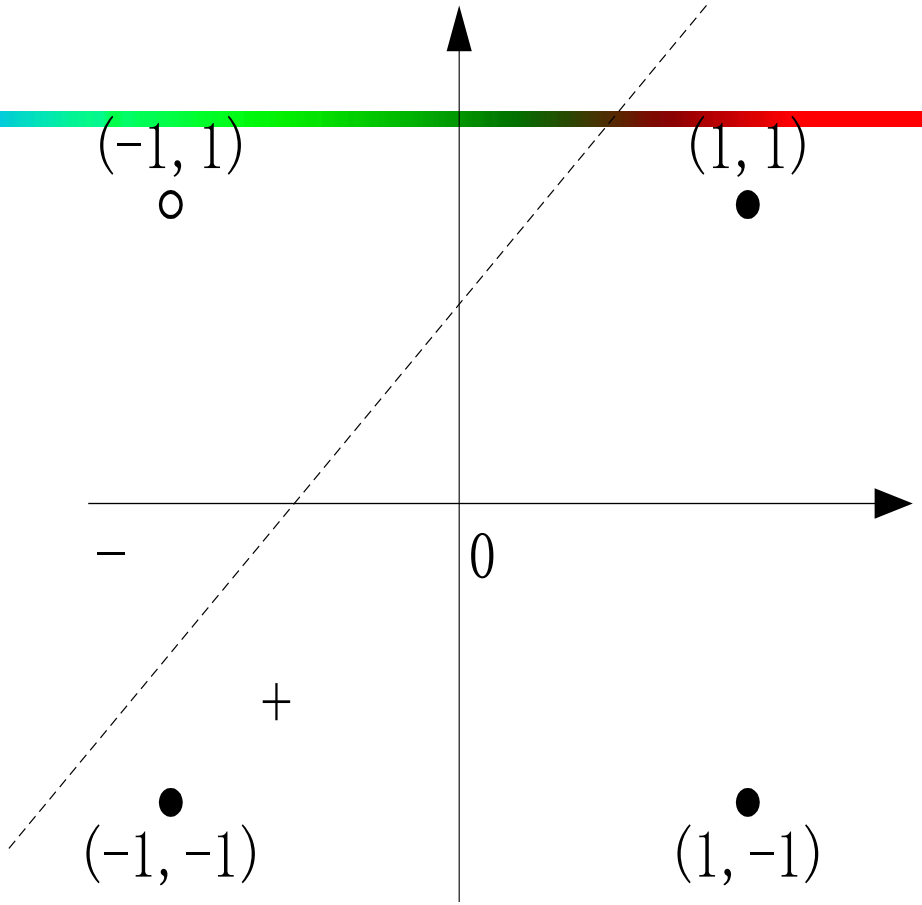
假定只有两个输入x_1和x_2，则自适应线性元件的模拟输出为：

$$y = x_1 w_1 + x_2 w_2 + w_0$$

调整临界阈值条件，可令模拟输出为零，即

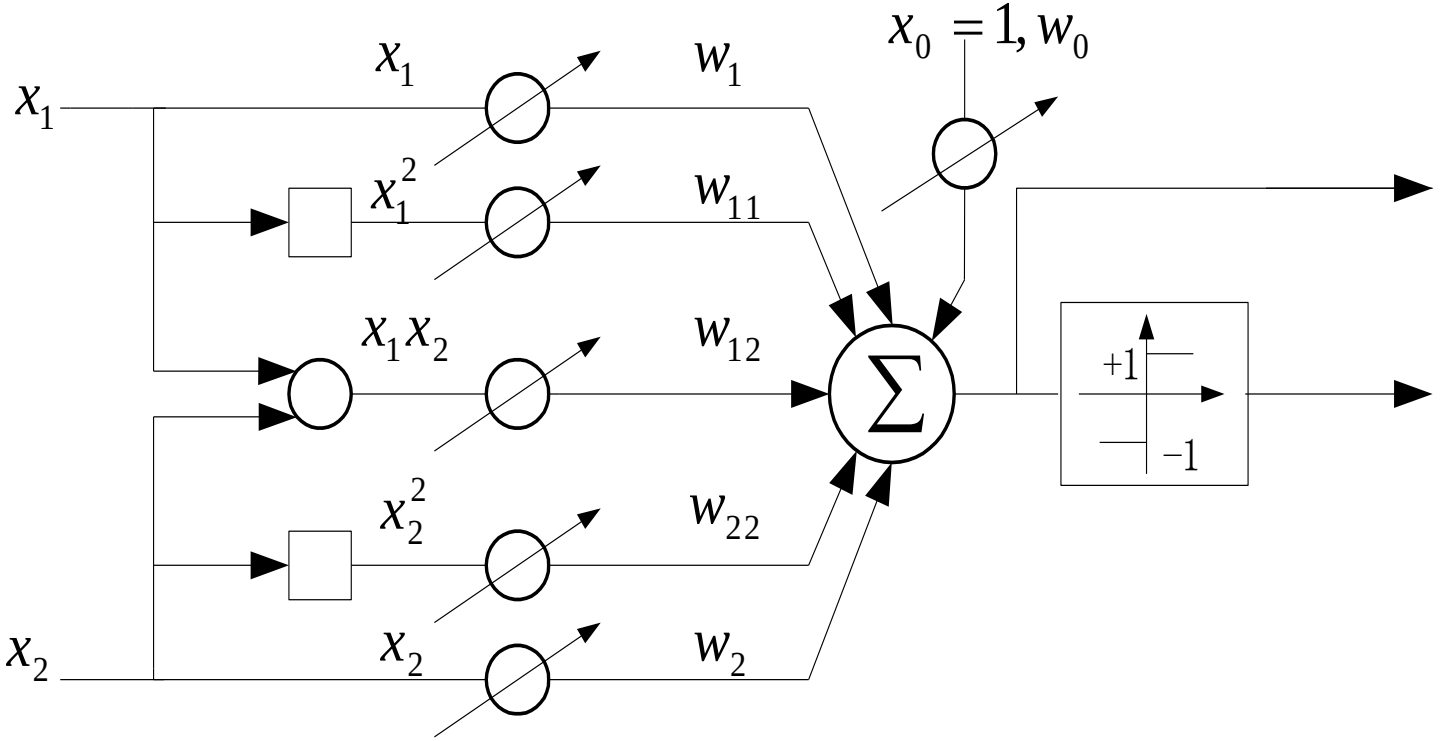
- $$y = x_1 w_1 + x_2 w_2 + w_0 = 0 \quad \Rightarrow \quad x_2 = -\frac{w_0}{w_2} - \frac{w_1}{w_2} x_1$$

该方程为直线方程，即单个自适应线性元件实现线性可分函数。



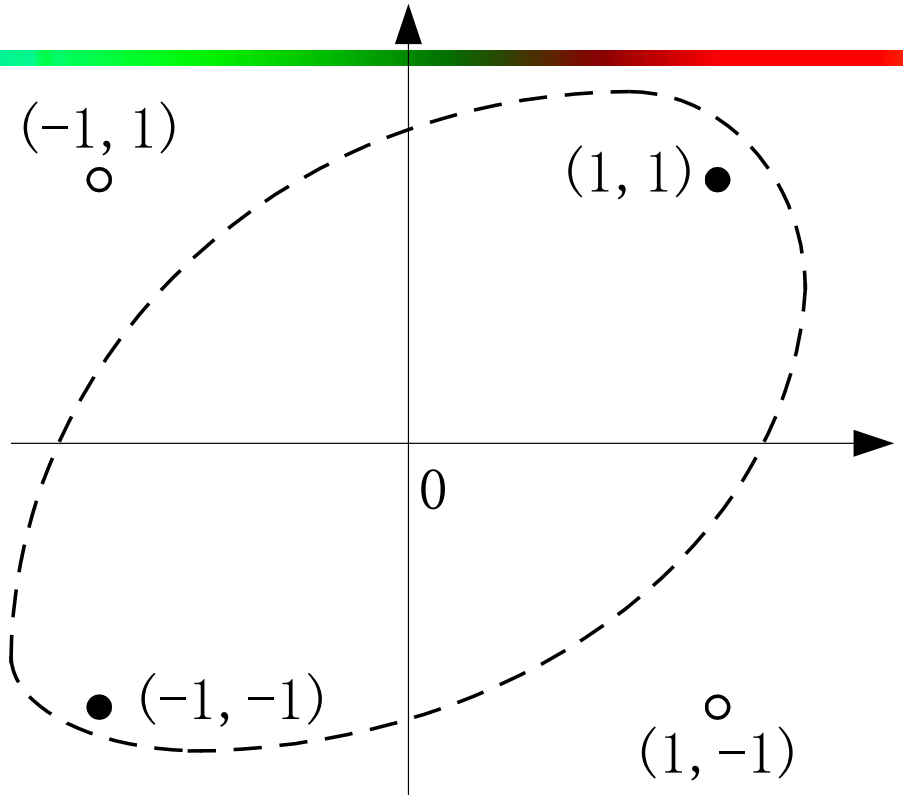
自适应线性元件的线性可分性图示

自适应非线性元件之一



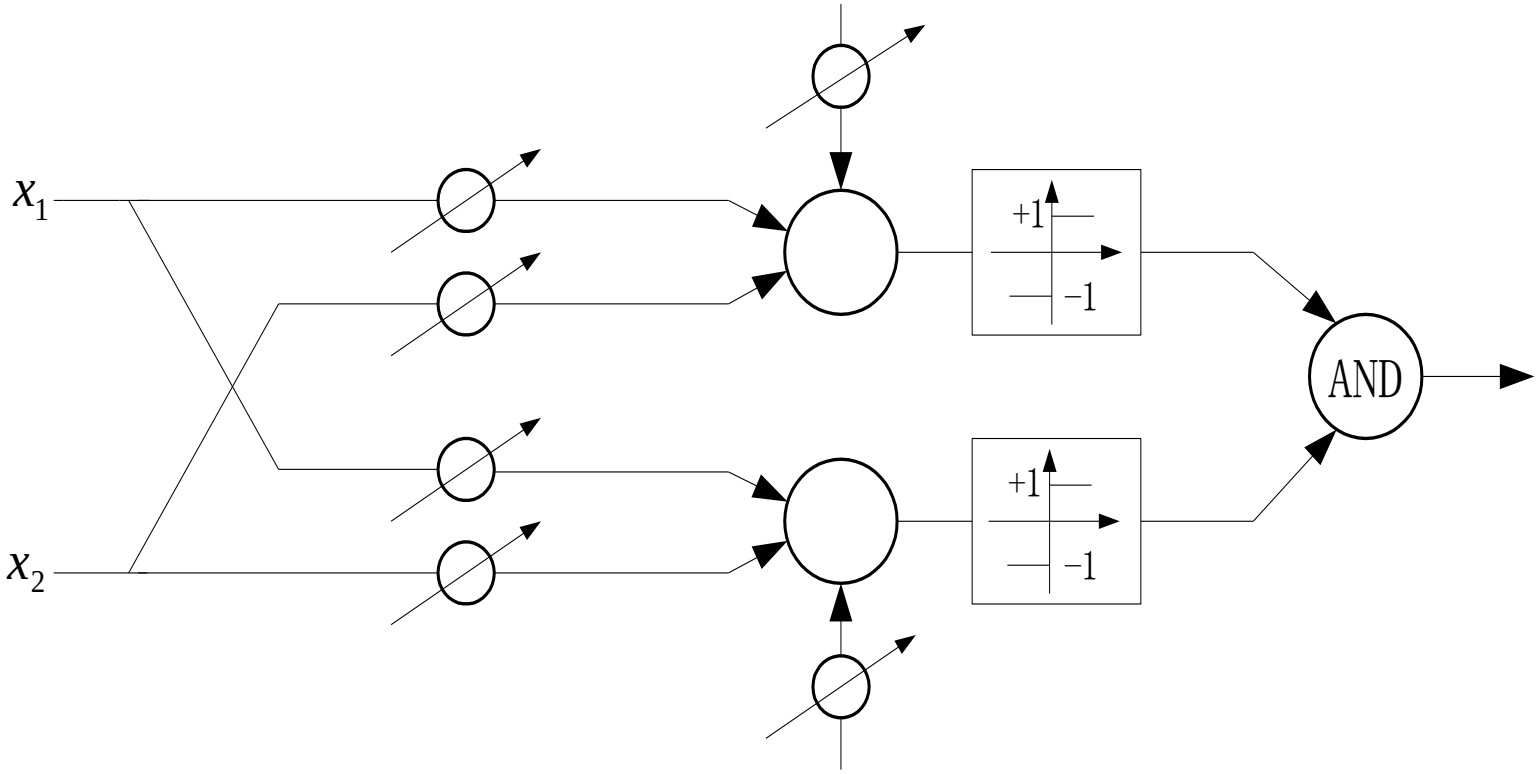
$$y = w_0 + x_1 w_1 + x_1^2 w_{11} + x_1 x_2 w_{12} + x_2^2 w_{22} + x_2 w_2$$

令$y = 0$ 上式为曲线方程，即通过选择W，可实现非线性函数

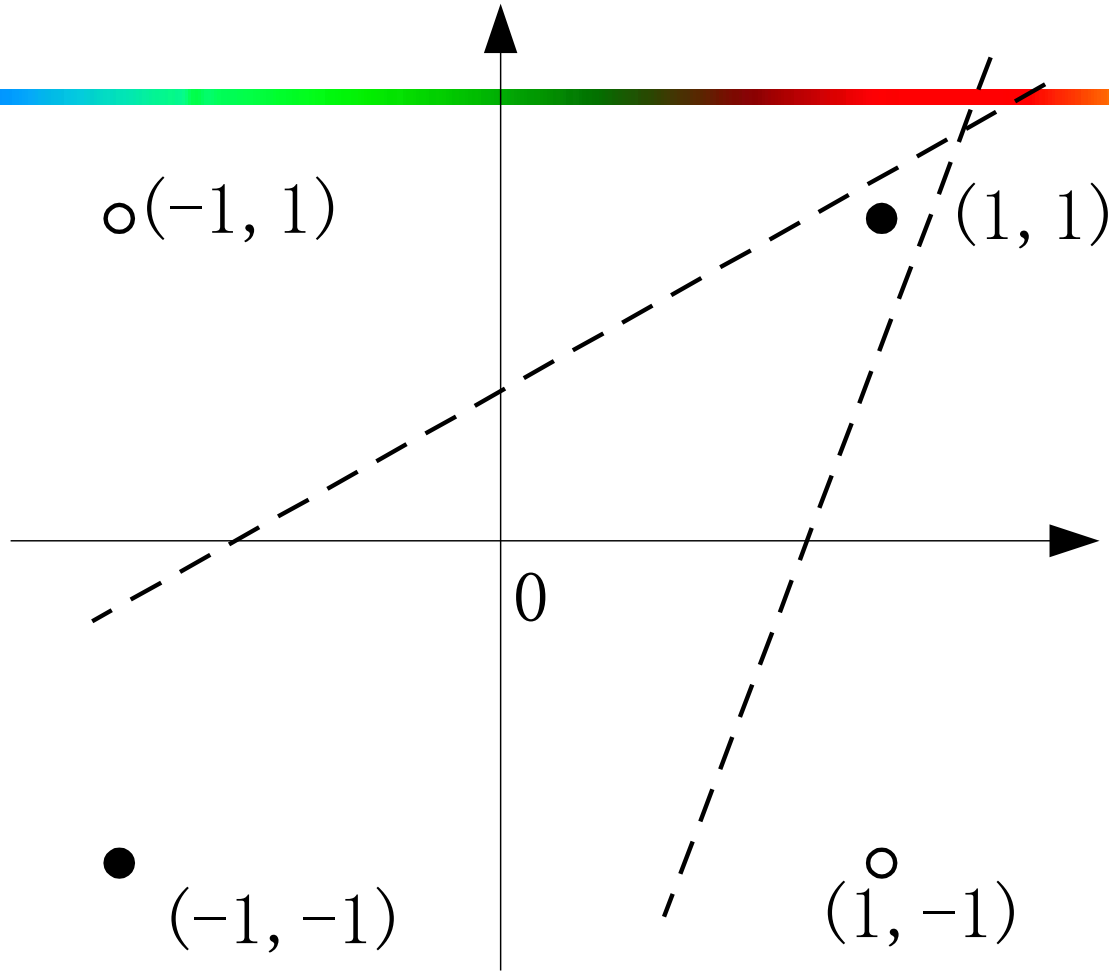


自适应非线性元件的非线性可分性

自适应非线性元件之二



其原理是实现多个线性函数，对线性不可分区域进行划分。


(−1, 1)
(1, 1)
0
(−1, −1)
(1, −1)

2.3 LMS学习算法

- 感知器和自适应线性元件在历史上几乎是同时提出的，并且两者在对权值的调整的算法非常相似，它们都是基于纠错学习规则的学习算法。
- 感知器算法存在如下问题：（1）不能推广到一般的前向网络中去；（2）函数不是线性可分时，得不出任何结果。
- 而由美国斯坦福大学的**Widrow**和**Hoff**在研究自适应理论时提出的**LMS**算法，由于其容易实现而很快得到了广泛的应用，成为自适应滤波的标准算法。

LMS学习算法

- 设$e(n)$为在时刻n时的误差信号，$E(W)$为代价函数：

$$e(n) = d(n) - X^T(n)W(n) \qquad E(W) = \frac{1}{2}e^2(n)$$

- 对上式两边求关于权值向量W的导数可得：

$$\frac{\partial e(n)}{\partial W} = -X(n) \qquad \frac{\partial E(W)}{\partial W} = e(n)\frac{\partial e(n)}{\partial W}$$

$$\frac{\partial E(W)}{\partial W} = -X(n)e(n)$$

为使误差尽快减小，令权值沿着误差函数负梯度方向改变，即：

$$\Delta W = -\eta \frac{\partial E(W)}{\partial W} = \eta X(n)e(n)$$

$$\begin{aligned} W(n+1) &= W(n) + \eta X(n)e(n) \\ &= W(n) + \eta X(n)\left[d(n) - X^T(n)W(n)\right] \\ &= \left[I - \eta X(n)\, X^T(n)\right]W(n) + \eta X(n)d(n) \end{aligned}$$

LMS算法

- 第一步：设置变量和参量：

  $X(n)= [1, x_1(n), x_2(n), \ldots, x_m(n)]$为输入向量，或称训练样本；

  $W(n)= [b(n), w_1(n), w_2(n), \ldots, w_m(n)]$为权值向量；

  $b(n)$ 为偏差；$y(n)$为实际输出；$d(n)$为期望输出；

  η为学习速率；n为迭代次数。

- 第二步：初始化，赋给$W_j(0)$各一个较小的随机非零值， $n = 0$；

LMS算法（续）

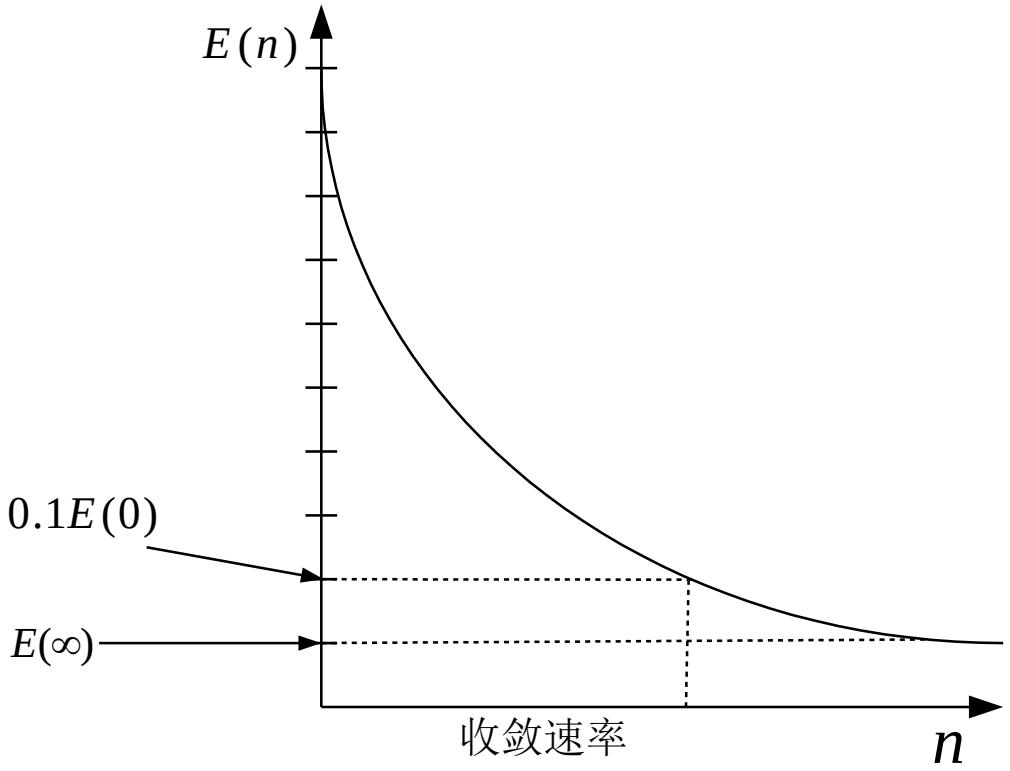
- 第三步：对于一组输入样本$X(n)= [1, x_1(n), x_2(n), \ldots, x_m(n)]$和对应的期望输出，计算：

$$e(n) = d(n) - X^T(n)W(n)$$

$$W(n+1) = W(n) + \eta X(n)e(n)$$

- 第四步：判断是否满足条件，若满足算法结束，若不满足将n值增加1，转到第三步重新执行。

以迭代次数n为横坐标，以误差信号的均方差$E(n)$为纵坐标，画出学习曲线，从学习曲线的单调性来判定LMS算法是否收敛。



标准的LMS算法学习曲线

Thank You

Intelligence Science

http://www.intsci.ac.cn/