

# 第 8 章 人工神经网络及其应用

---

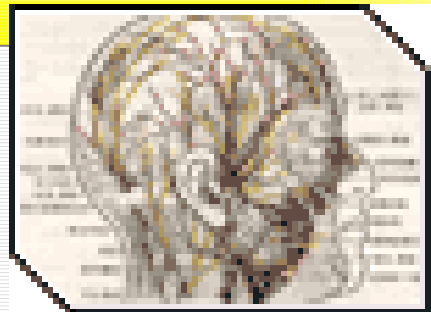


# 第 8 章 人工神经网络及其应用

## 神经网络（**neural networks**，**NN**）

- **生物神经网络**（**natural neural network, NNN**）：由中枢神经系统（脑和脊髓）及周围神经系统（感觉神经、运动神经等）所构成的错综复杂的神经网络，其中最重要的是**脑神经系统**。
- **人工神经网络**（**artificial neural networks, ANN**）：模拟**人脑神经系统**的结构和功能，运用大量简单处理单元经广泛连接而组成的人工网络系统。

神经网络方法：**隐式**的  
知识表示方法






# 第 8 章 人工神经网络及其应用

- 8.1 神经元与神经网络
- 8.2 **BP** 神经网络及其学习算法
- 8.3 **BP** 神经网络的应用
- 8.4 **Hopfield** 神经网络及其改进
- 8.5 **Hopfield** 神经网络的应用
- 8.6 **Hopfield** 神经网络优化方法求解 **JSP**
- 8.7 卷积神经网络及其应用

# 第 8 章 人工神经网络及其应用

- ✓ 8.1 神经元与神经网络
- 8.2 **BP** 神经网络及其学习算法
- 8.3 **BP** 神经网络的应用
- 8.4 **Hopfield** 神经网络及其改进
- 8.5 **Hopfield** 神经网络的应用
- 8.6 **Hopfield** 神经网络优化方法求解 **JSP**
- 8.7 卷积神经网络 及其应用

# 8.1 神经元与神经网络

-  **8.1.1** 生物神经元的结构
-  **8.1.2** 神经元数学模型
-  **8.1.3** 神经网络结构与工作方式

## 8.1.1 生物神经元的结构

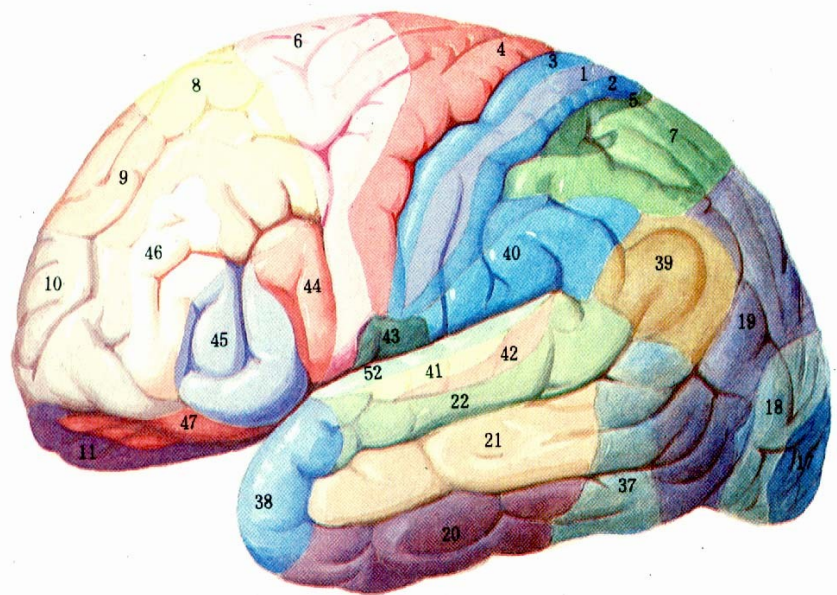
- 人脑由一千多亿（**1011** 亿—**1014** 亿）个神经细胞（神经元）交织在一起的网状结构组成，其中大脑皮层约 **140** 亿个神经元，小脑皮层约 **1000** 亿个神经元。



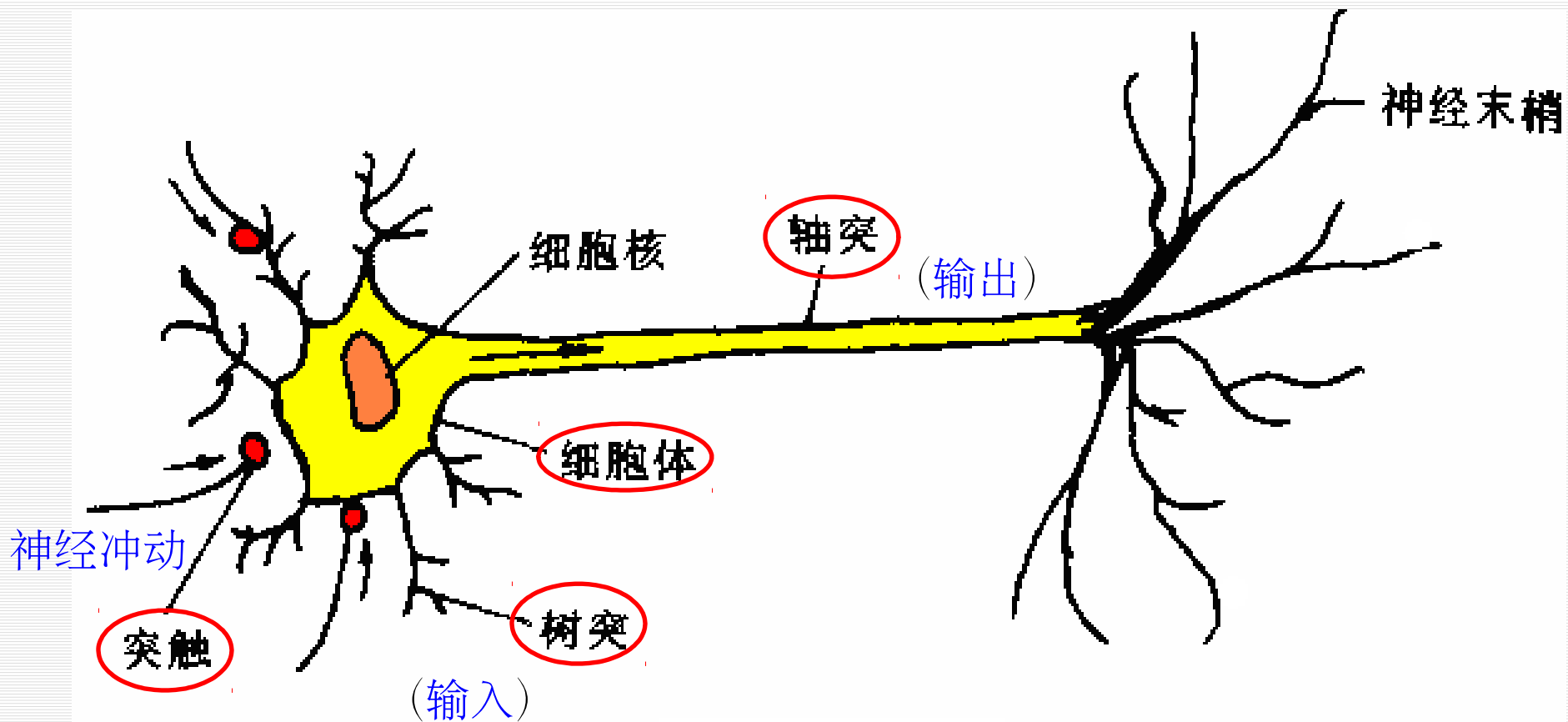
- 神经元约有 **1000** 种类型，每个神经元大约与  **$10^3$  —  $10^4$**  个其他神经元相连接，形成极为错综复杂而又灵活多变的神经网络。
- 人的智能行为就是由如此高度复杂的组织产生的。浩瀚的宇宙中，也许只有包含数千亿颗星球的银河系的复杂性能够与大脑相比。

# 大脑神经系统

- 神经系统是多层次的。这些不同层次的研究互相启示，互相推动。在低层次（细胞、分子水平）上的工作为较高层次的观察提供分析的基础，而较高层次的观察又有助于引导低层次工作的方向和体现其功能意义。



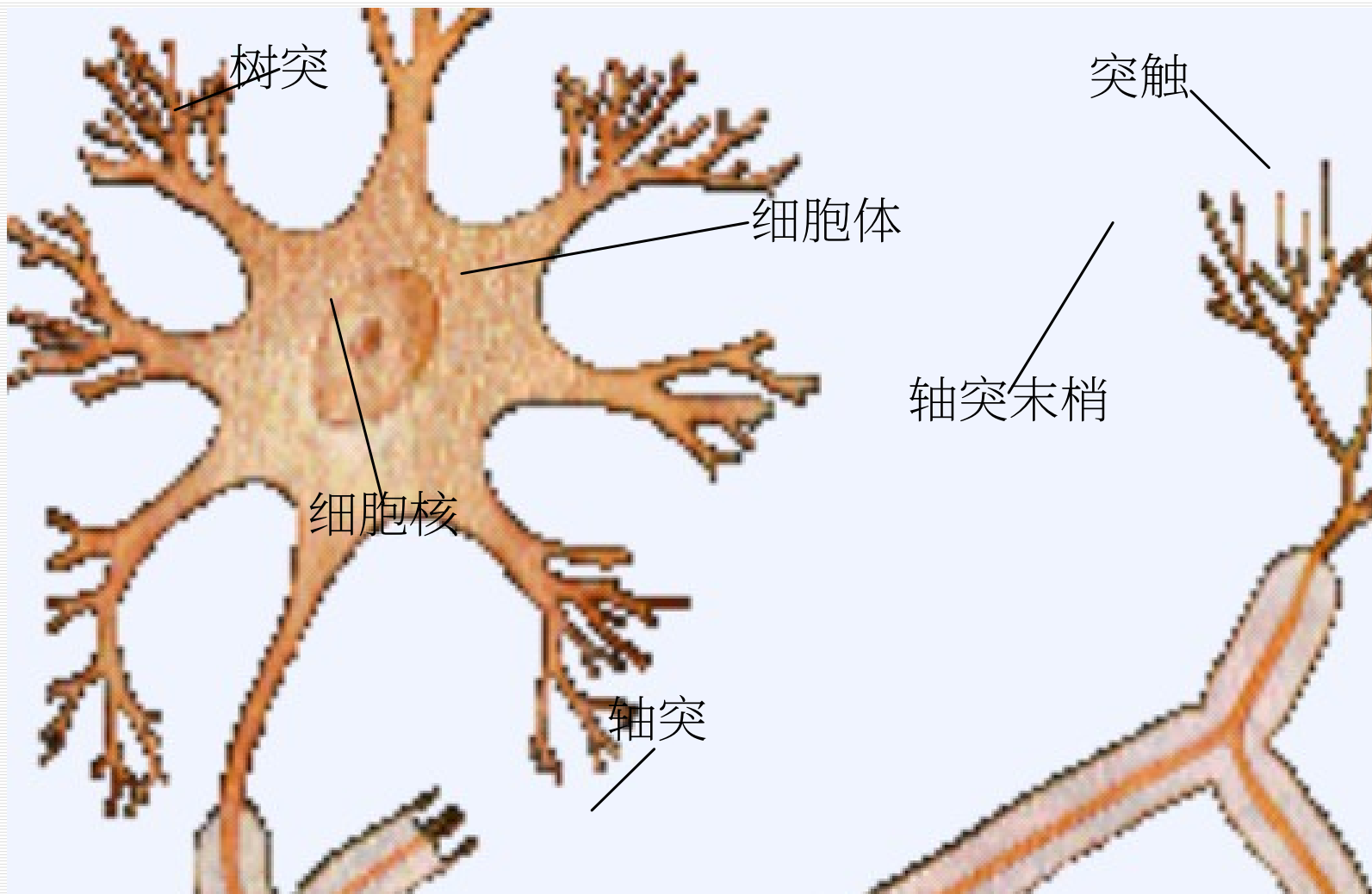
## 8.1.1 生物神经元的结构

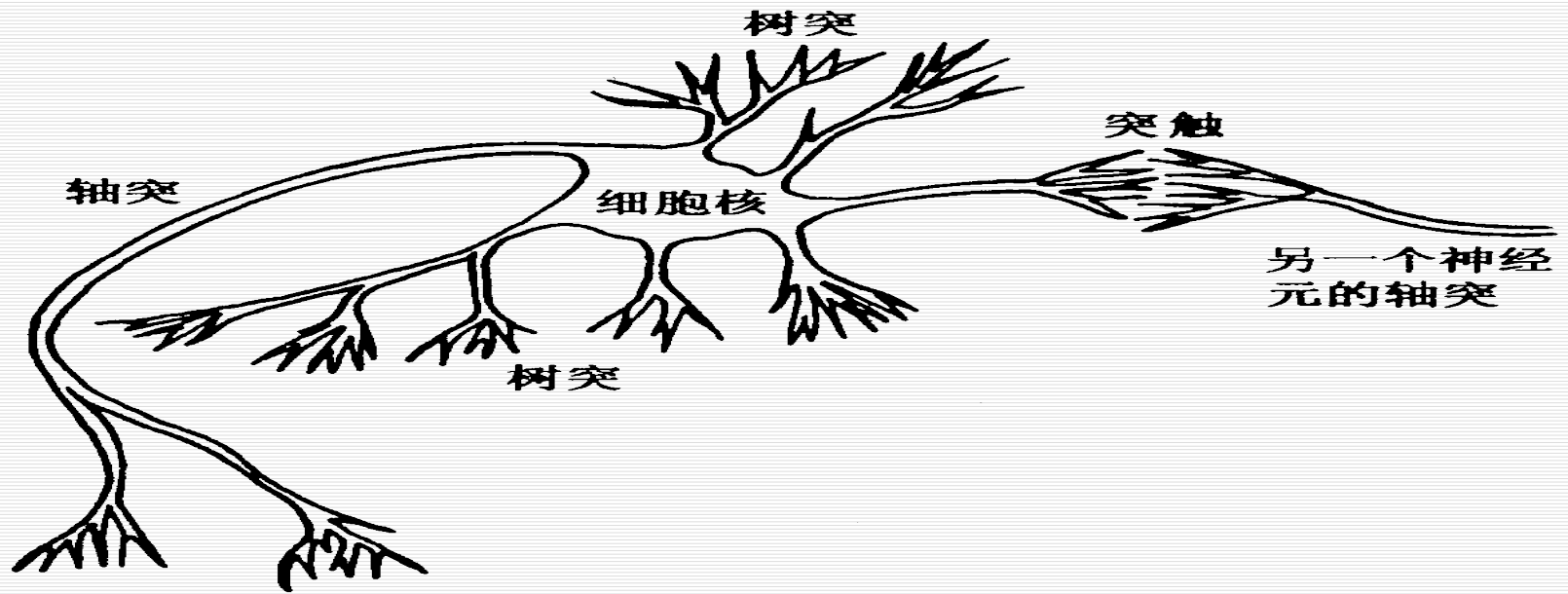


生物神经元结构

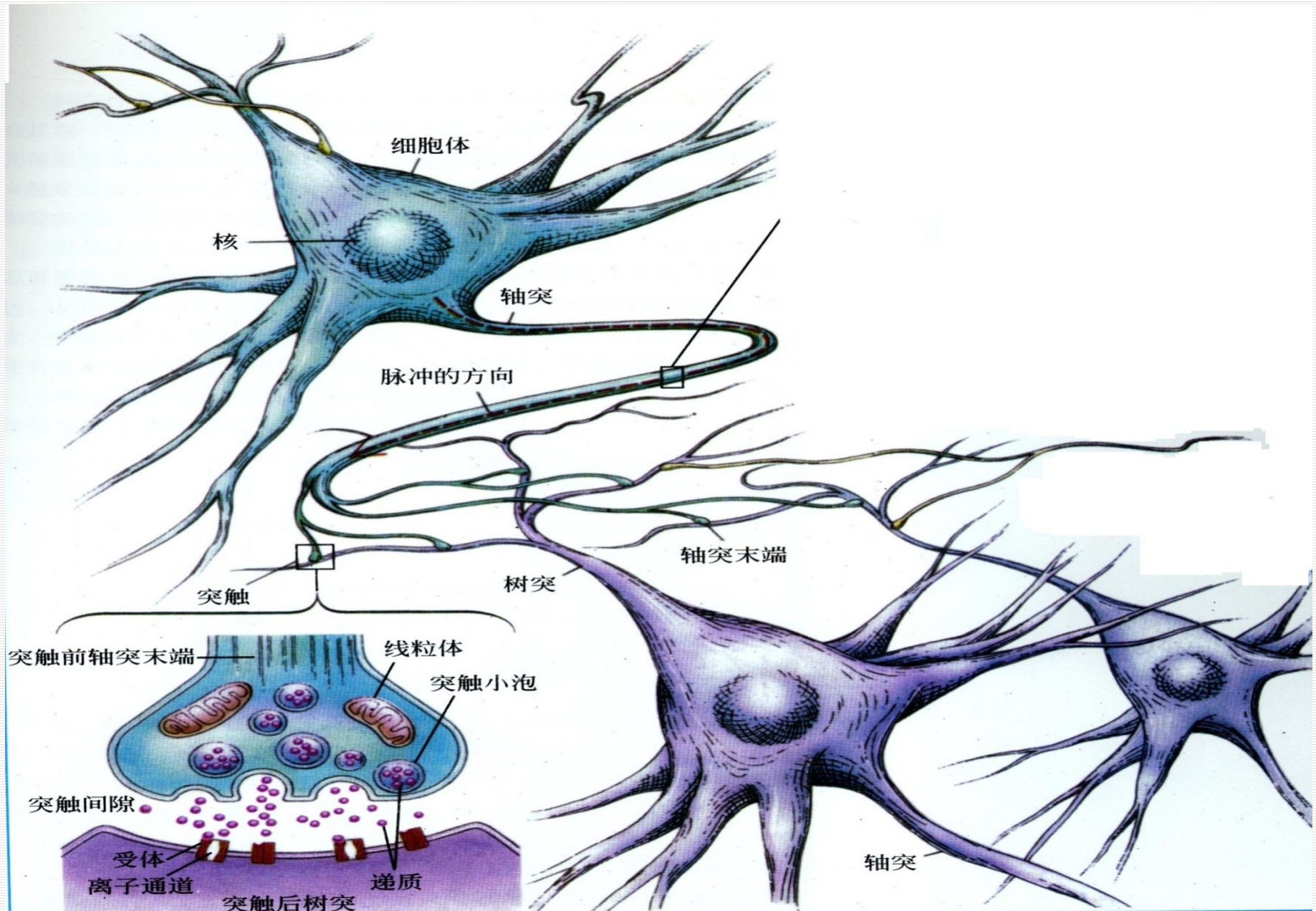


# 生物神经元基本结构示意图





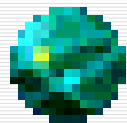
生物神经元在结构上由：  
细胞体 (Cell body)、  
树突 (Dendrite)、  
轴突 (Axon)、  
突触 (Synapse)  
四部分组成。用来完成神经元  
间信息的接收、传递和处理。



## 8.1.1 生物神经元的结构

- 工作状态:
- 兴奋状态: 细胞膜电位  $>$  动作电位的阈值  $\rightarrow$  神经冲动
- 抑制状态: 细胞膜电位  $<$  动作电位的阈值
- 学习与遗忘: 由于神经元结构的可塑性, 突触的传递作用可增强和减弱。

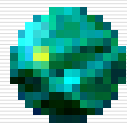
# 8.1 神经元与神经网络



## 8.1.1 生物神经元的结构



## 8.1.2 神经元数学模型



## 8.1.3 神经网络的结构与工作方式

## 8.1.2 神经元数学模型

麦克洛奇和皮兹提出 **M — P** 模型。

1943，神经生理学家 McCulloch 和数学家 Pitts 基于早期神经元学说，归纳总结了生物神经元的基本特性，建立了具有逻辑演算功能的神经元模型以及这些人工神经元互联形成的人工神经网络，即所谓的 McCulloch-Pitts 模型。

McCulloch-Pitts 模型（MP模型）是世界上第一个神经计算模型，即人工神经系统。



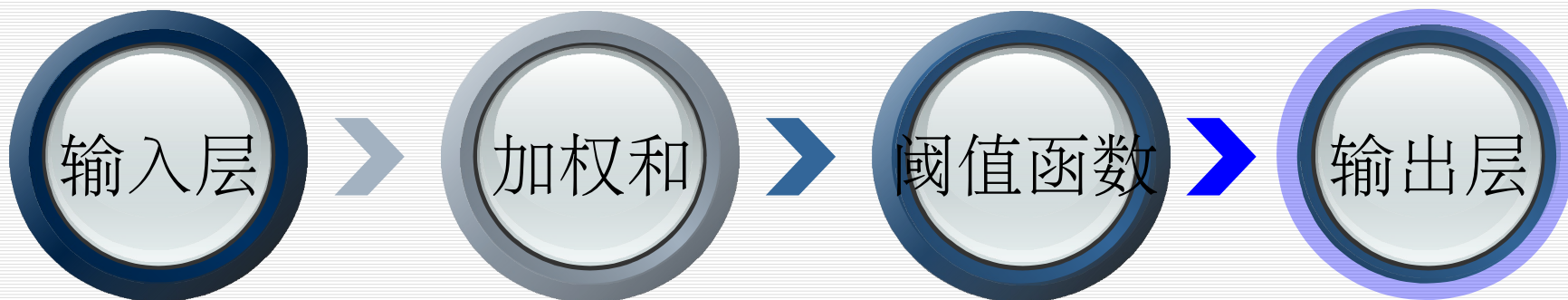
## 生物神经元

- 树突
- 细胞体
- 轴突
- 突触

## 类比关系

## 人工神经元

- 输入层
- 加权和
- 阈值函数
- 输出层



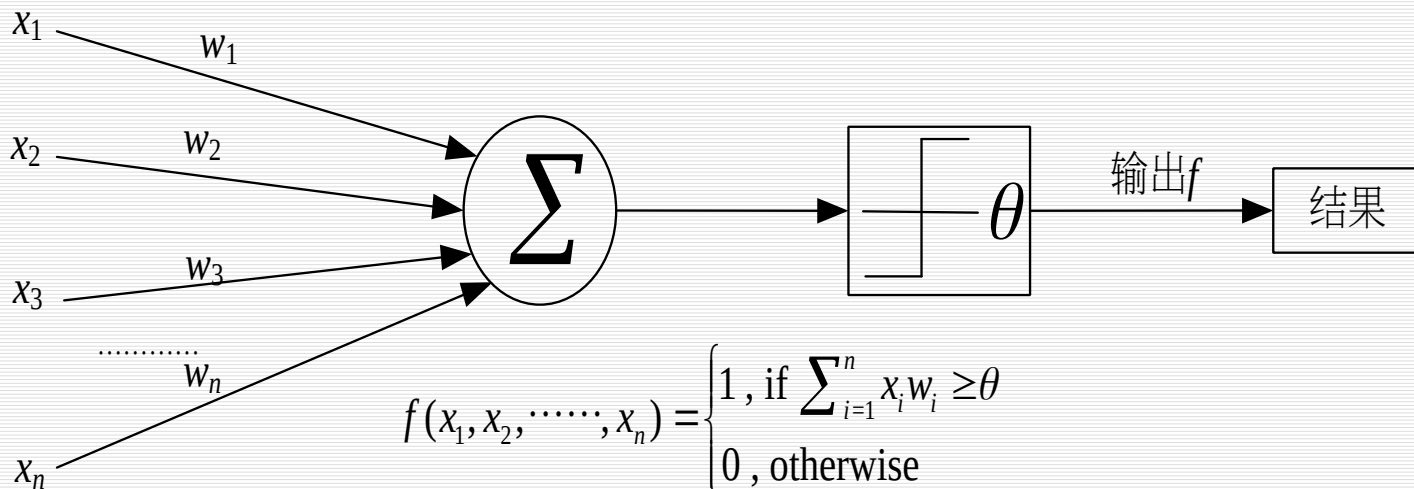
模拟神经元的树突  
接收输入信号

模拟神经元的细胞体  
加工和处理信号

模拟神经元的轴突  
控制信号的输出

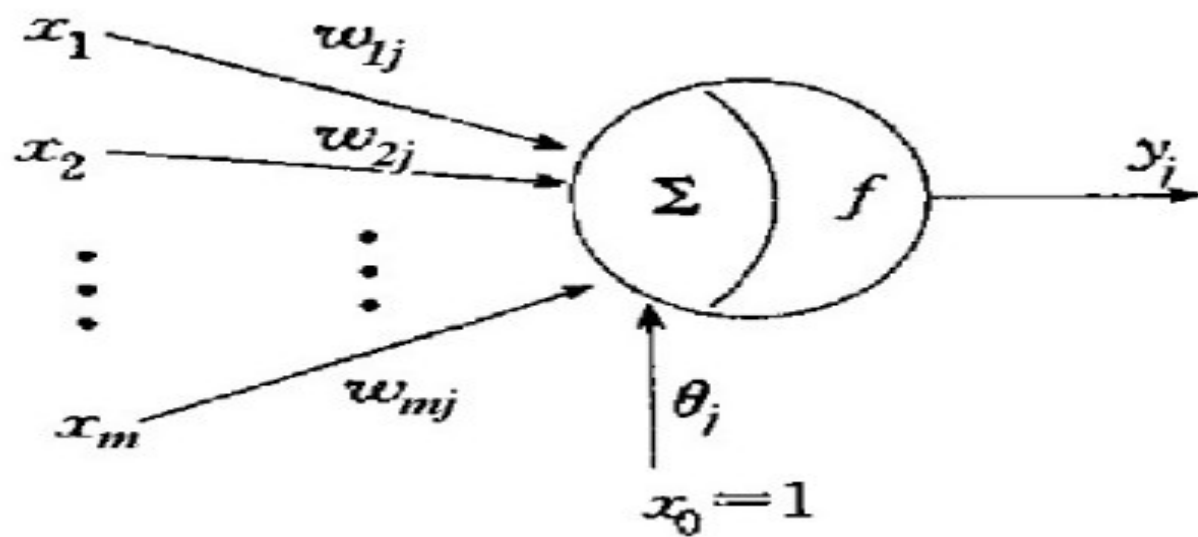
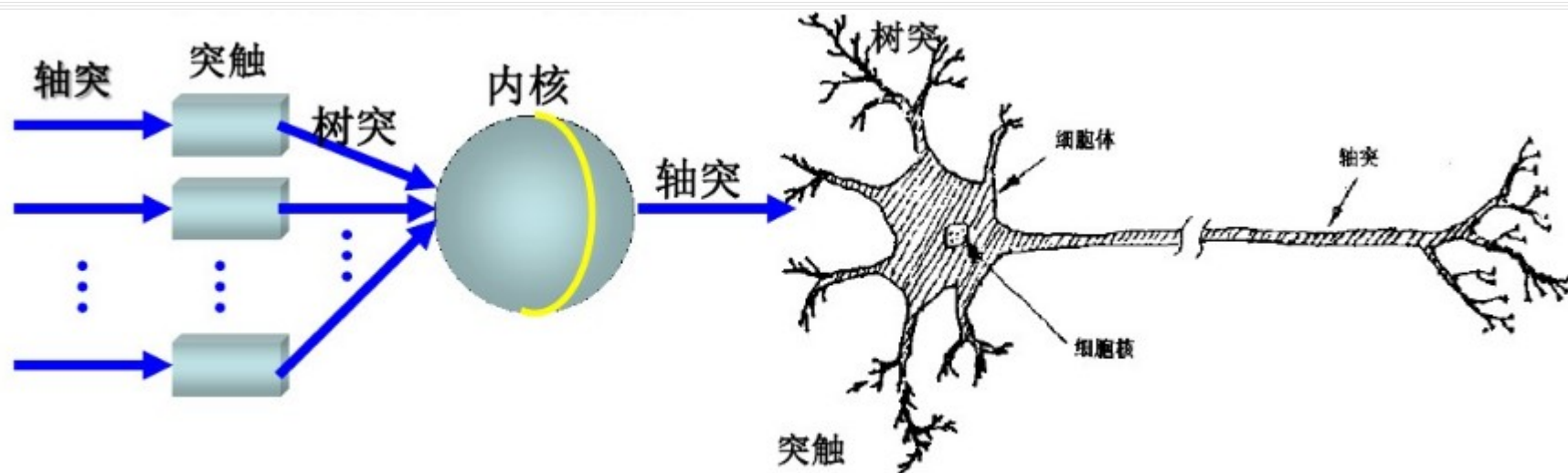
模拟神经元的突触  
对结果进行输出

## 人工神 经元结 构功能 示意图



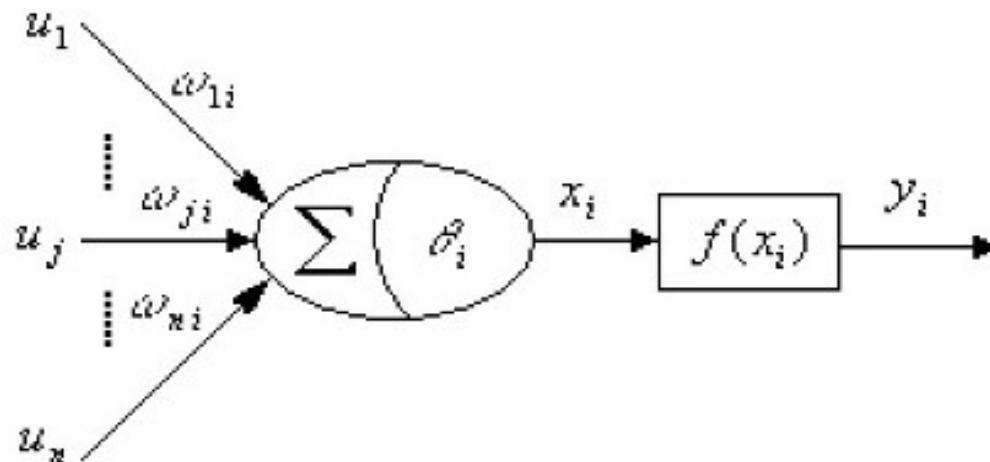


## 8.1.2 神经元数学模型



## 8.1.2 神经元数学模型

### ■ MP模型



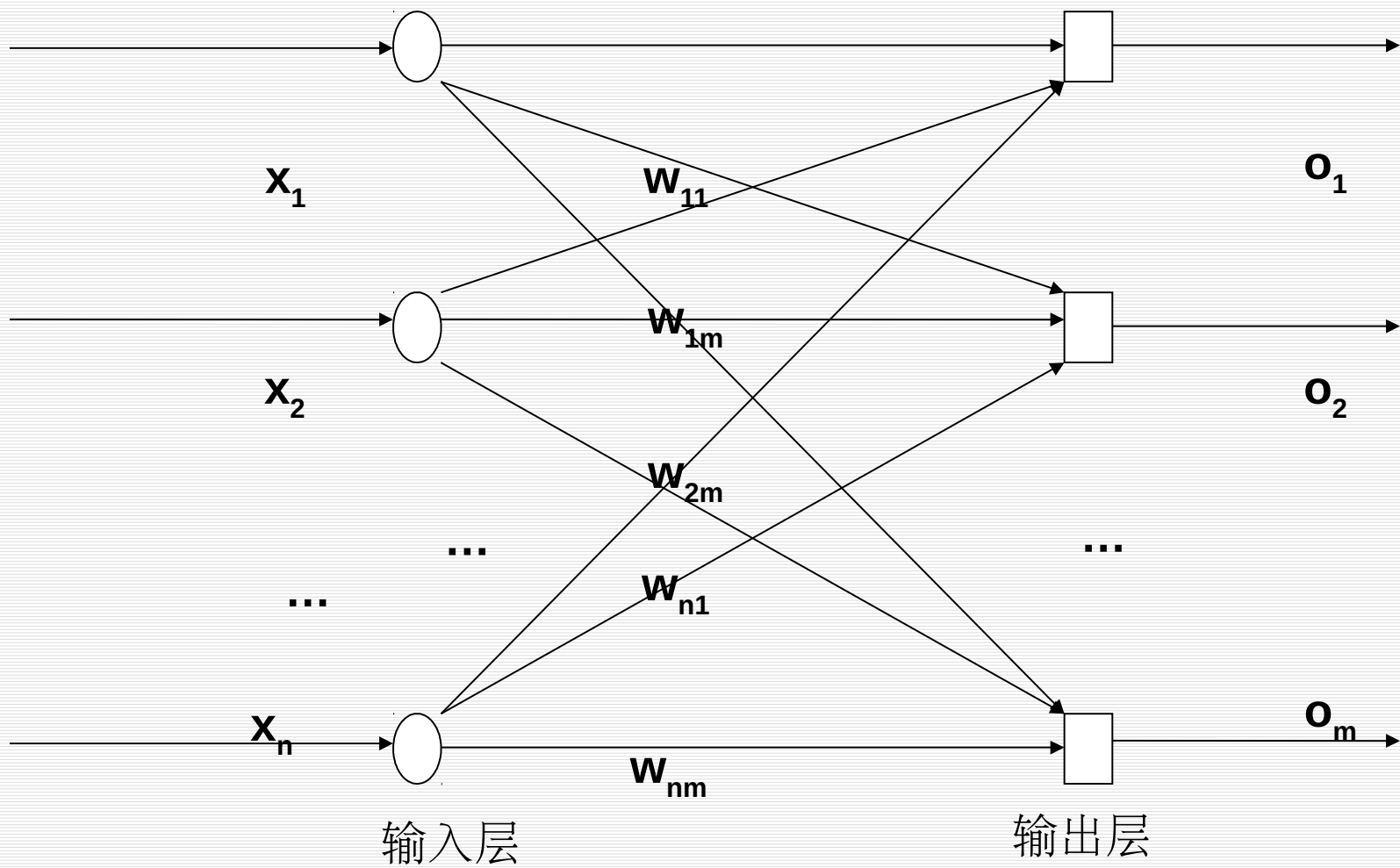
### ■ 求和操作

$$x_i = \sum_{j=1}^n w_{ji} u_j - \theta_i$$

### ■ 作用函数

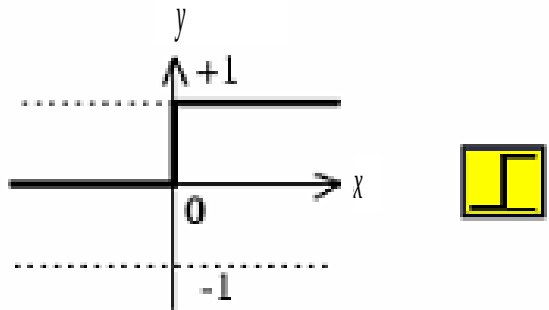
$$y_i = f(x_i) = f\left(\sum_{j=1}^n w_{ji} u_j - \theta_i\right)$$

# 单层感知器



## 8.1.2 神经元数学模型

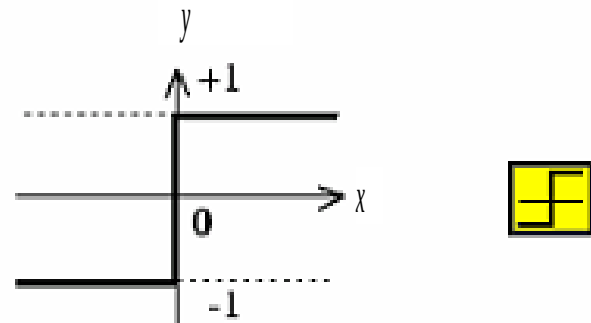
- 非线性激励函数（作用函数、传输函数、输出变换函数）



$$y = \text{hardlim}(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases}$$

Hard-Limit Transfer Function

(硬极限函数或阶跃函数)



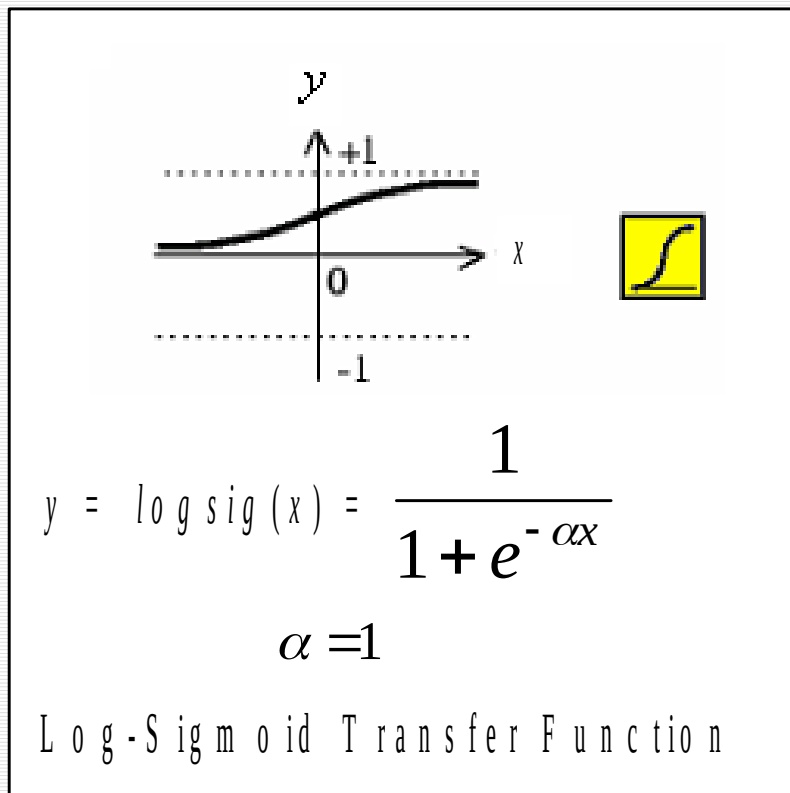
$$y = \text{hardlims}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

Symmetric Hard-Limit Trans. Funct.

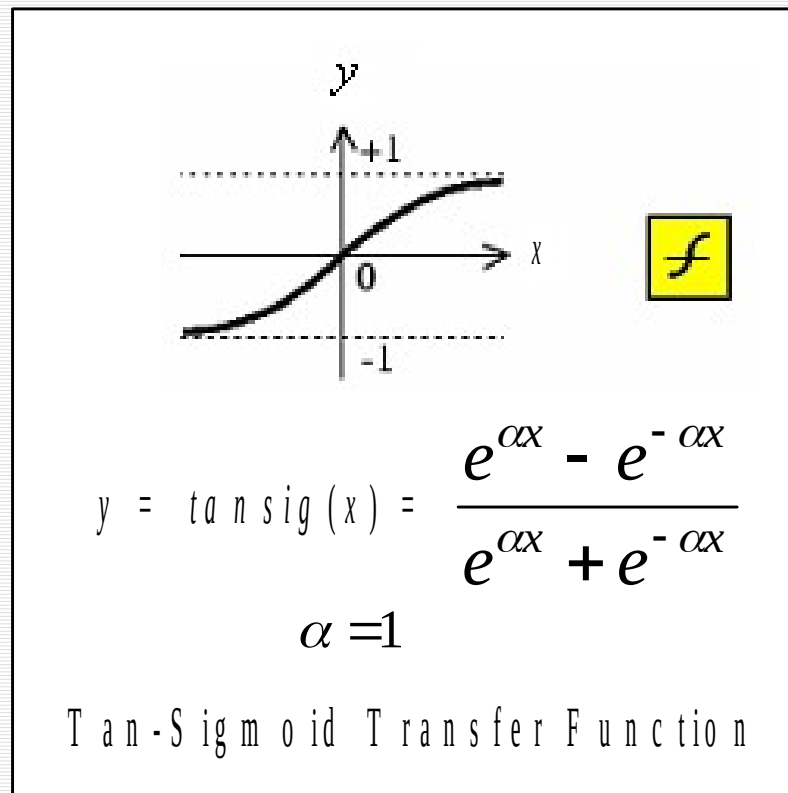
(对称硬极限函数)

## 8.1.2 神经元数学模型

- 非线性激励函数（传输函数、输出变换函数）



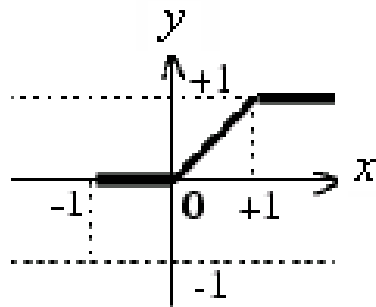
（对数 - S 形函数或 S 型函数）



（双曲正切 S 形函数）

## 8.1.2 神经元数学模型

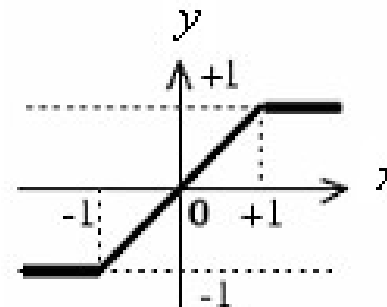
- 非线性激励函数（传输函数、输出变换函数）



$$y = \text{satlin}(x) = \begin{cases} 1 & x \geq 1 \\ x & 0 < x < 1 \\ 0 & x \leq 0 \end{cases}$$

Satlin Transfer Function

(饱和线性函数)



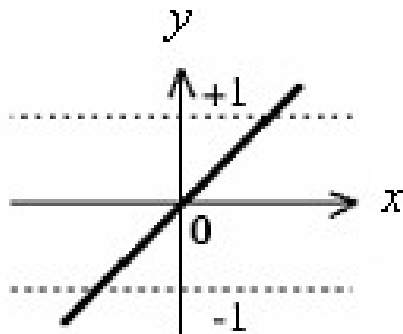
$$y = \text{satlins}(x) = \begin{cases} 1 & x \geq 1 \\ x & -1 < x < 1 \\ -1 & x \leq -1 \end{cases}$$

Symmetric Satlins Trans. Func.

(对称饱和线性函数)

## 8.1.2 神经元数学模型

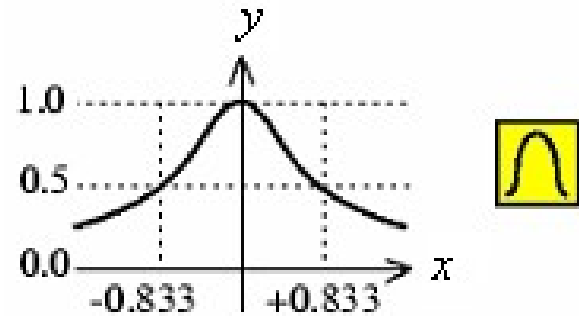
- 非线性激励函数（传输函数、输出变换函数）



$$y = \text{purelin}(x) = x$$

Linear Transfer Function

（线性函数）



$$y = \text{radbas}(x) = e^{-\left(\frac{x}{\sigma}\right)^2}$$

Radial Basis Function

（高斯或径向基函数）

## 8.1.2 神经元数学模型

### ■ 工作过程：

- 从各输入端接收输入信号  $x_j (j = 1, 2, \dots, n)$
- 根据连接权值求出所有输入的加权和

$$u_i = \sum_{j=1}^n x_j w_{ij} + \theta_i$$




$$u_i = \sum_{j=1}^n x_j w_{ij} + \theta_i$$

- 用非线性激励函数进行转换，得到输出

$$y_i = f(u_i) = f\left(\sum_{j=1}^n x_j w_{ij} + \theta_i\right)$$



# 8.1 神经元与神经网络

-  **8.1.1** 生物神经元的结构
-  **8.1.2** 神经元的数学模型
-  **8.1.3** 神经网络的结构与工作方式

## 8.1.3 神经网络的结构与工作方式

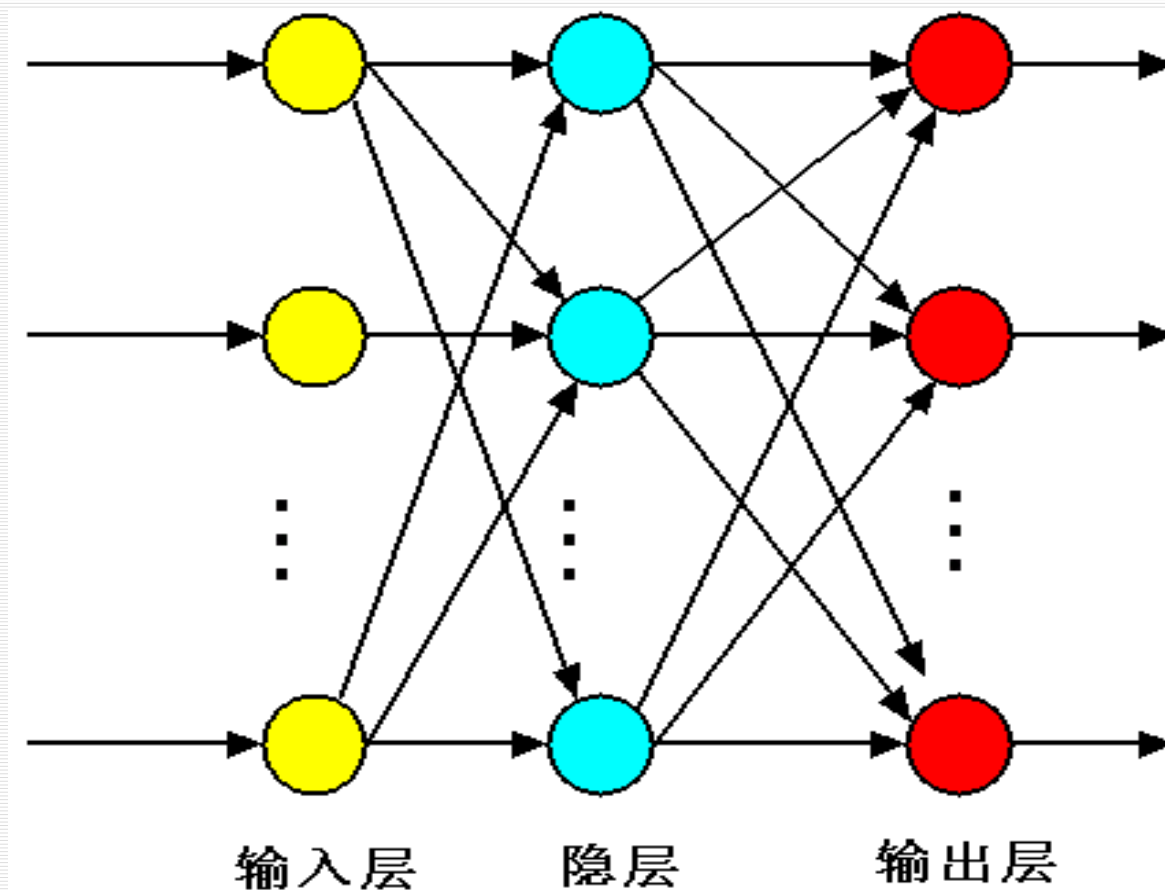
□ 决定人工神经网络性能的三大要素：

- 神经元的特性。
- 神经元之间相互连接的形式——拓扑结构。
- 为适应环境而改善性能的学习规则。

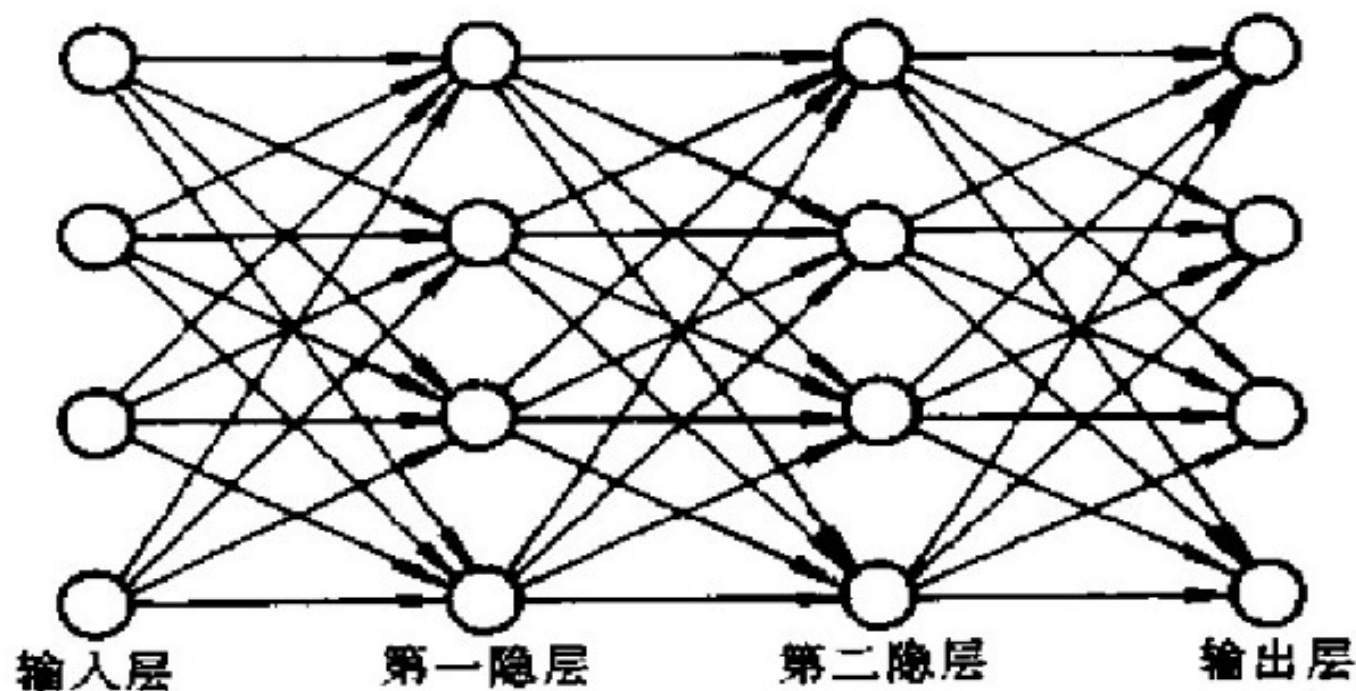
## 8.1.3 神经网络的结构与工作方式

### 1. 神经网络的结构

#### (1) 前馈型（前向型）

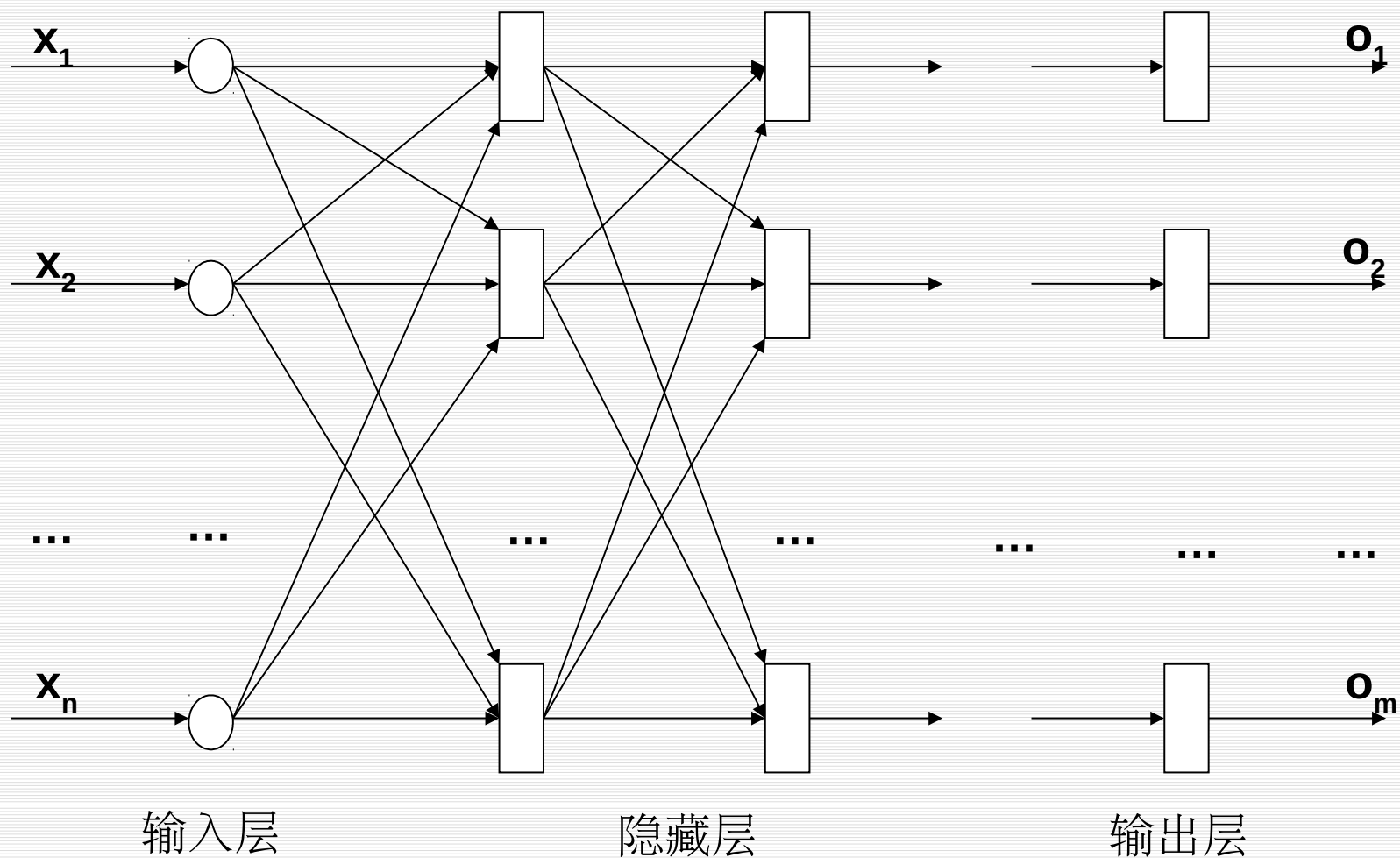


只要在输入层与输出层之间增加一层或多层隐层，就可得到多层感知器。



三层感知器

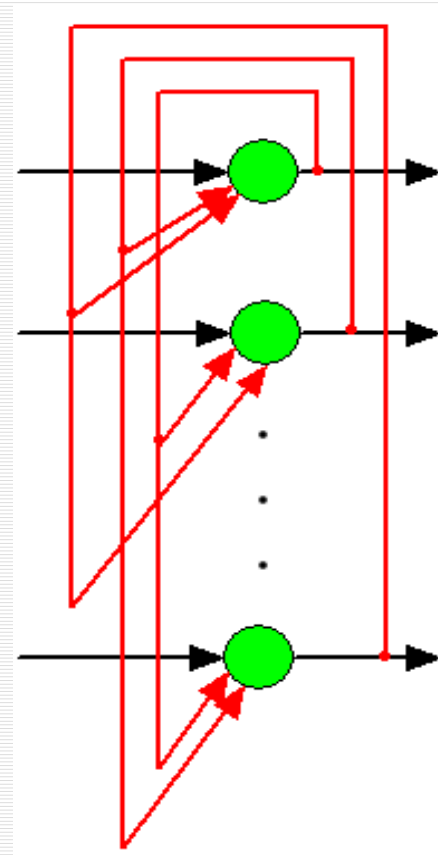
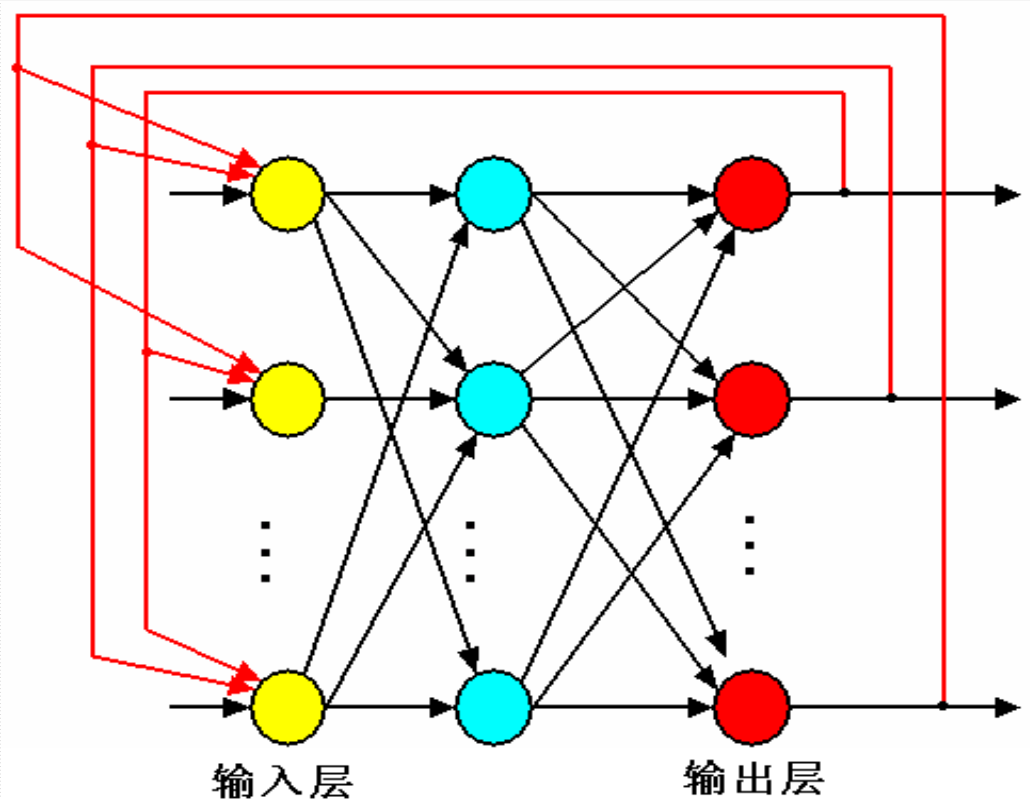
# 多层感知器网（多级网，多层网）



## 8.1.3 神经网络的结构与工作方式

### 1. 神经网络的结构

#### (2) 反馈型



( Hopfield 神经网络)

## 8.1.3 神经网络的结构与工作方式

### □ 2. 神经网络的工作方式

- **同步**（并行）方式：任一时刻神经网络中所有神经元同时调整状态。
- **异步**（串行）方式：任一时刻只有一个神经元调整状态，而其它神经元的状态保持不变。

## 8.1.4 神经网络的发展概况

□ 探索时期（开始于 20 世纪 40 年代）：

- 1943 年，麦克劳（**W. S. McCulloch**）和匹茨（**W. A. Pitts**）首次提出一个神经网络模型——**M - P** 模型。
- 1949 年，赫布（**D. O. Hebb**）提出改变神经元连接强度的 **Hebb** 学习规则。



## 8.1.4 神经网络的发展概况

- ❑ 第一次热潮时期： 20 世纪 50 年代末— 20 世纪 60 年代初
  - 1958 年，罗森布拉特（ **F. Rosenblatt** ）提出感知器模型（ **perceptron** ）。
  - 1959 年，威德罗（ **B. Widrow** ）等提出自适应线性元件（ **adaline** ）网络，通过训练后可用于抵消通信中的回波和噪声。 1960 年， 他和 **M. Hoff** 提出 **LMS**（ **Least Mean Square** 最小方差）算法的学习规则。

## 8.1.4 神经网络的发展概况

- ❑ 低潮时期： 20 世纪 60 年代末— 20 世纪 70 年代
  - 1969 年，明斯基（ **M. Minsky** ）等在《 **Perceptron** 》中对感知器功能得出悲观结论。
  - 1972 年， **T. Kohonen** 和 **J. Anderson** 分别提出能完成记忆的新型神经网络。
  - 1976 年， **S. Grossberg** 在自组织神经网络方面的研究十分活跃。

## 8.1.4 神经网络的发展概况

□ 第二次热潮时期： 20 世纪 80 年代至今

- 1982 年 — 1986 年，霍普菲尔德（ **J. J. Hopfield** ）陆续提出离散的和连续的全互连神经网络模型，并成功求解旅行商问题（ **TSP** ）。
- 1986 年，鲁姆尔哈特（ **Rumelhart** ）和麦克劳（ **McCellan** ）等在《 **Parallel Distributed Processing** 》中提出反向传播学习算法（ **B — P** 算法）。
- 1987 年 6 月，首届国际神经网络学术会议在美国圣地亚哥召开，成立了国际神经网络学会（ **INNS** ）。

## 8.1.4 神经网络的发展概况

### □ 神经网络控制的研究领域

- 基于神经网络的系统辨识
- 神经网络控制器
- 神经网络与其他算法（模糊逻辑、专家系统、遗传算法等）相结合
- 优化计算

# 第 8 章 人工神经网络及其应用

- 8.1 神经元与神经网络
- ✓ 8.2 **BP** 神经网络及其学习算法
- 8.3 **BP** 神经网络的应用
- 8.4 **Hopfield** 神经网络及其改进
- 8.5 **Hopfield** 神经网络的应用
- 8.6 **Hopfield** 神经网络优化方法求解 **JSP**
- 8.7 卷积神经网络及其应用

## 8.2 BP 神经网络及其学习算法

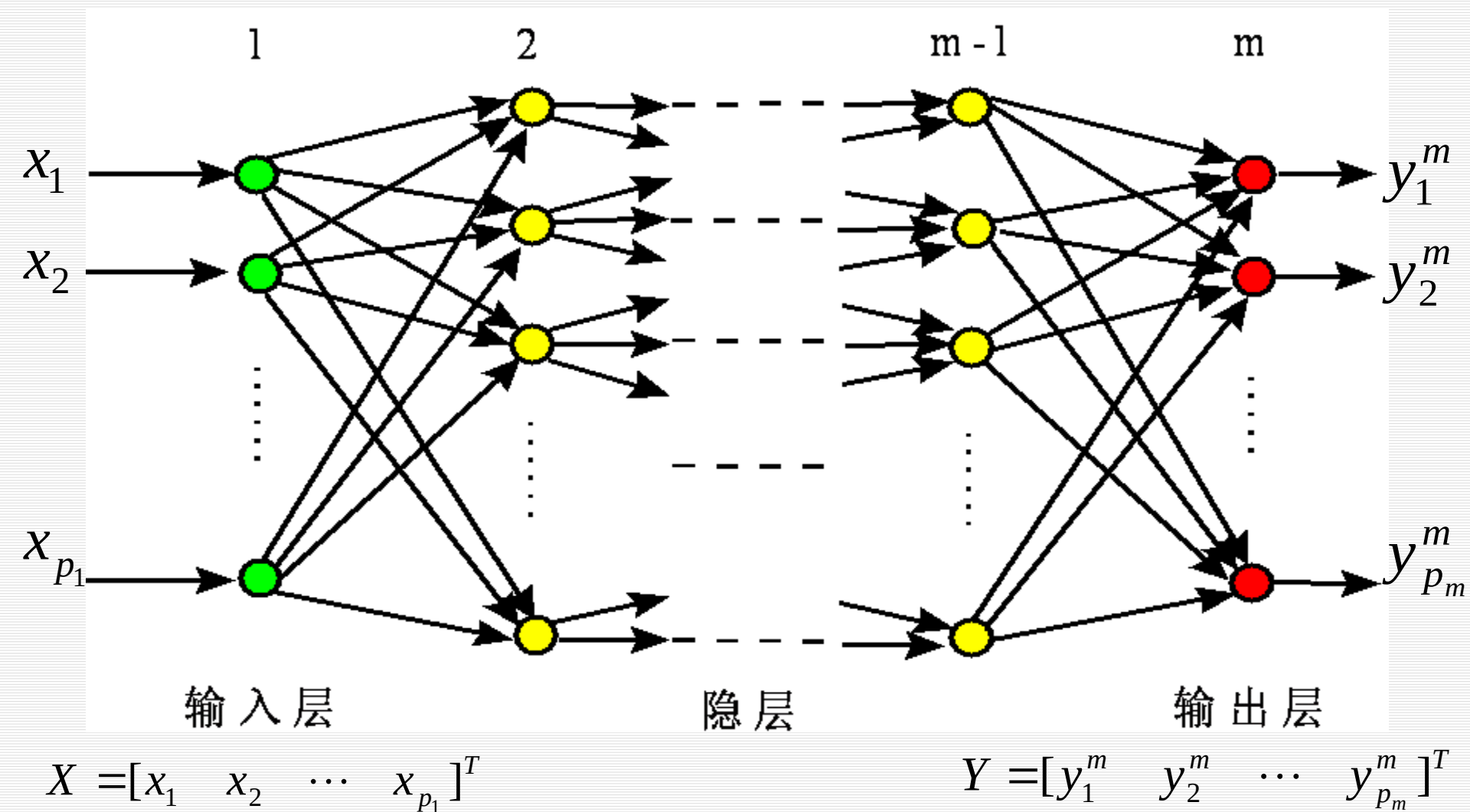
- 8.2.1 BP 神经网络（**back-propagation neural network**）的结构
- 8.2.2 BP 学习算法
- 8.2.3 BP 算法的实现

## 8.2 BP 神经网络及其学习算法

- 8.2.1 BP 神经网络（**back-propagation neural network**）的结构
- 8.2.2 BP 学习算法
- 8.2.3 BP 算法的实现

# 8.2.1 BP 神经网络的结构

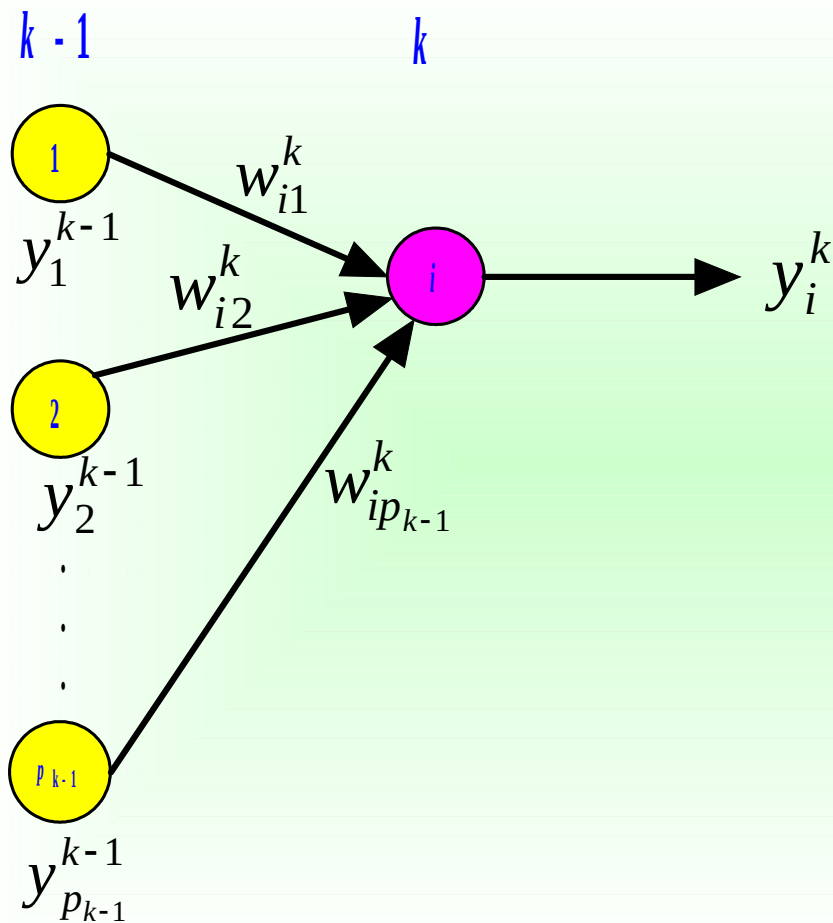
## 1. BP 网络结构





## 8.2.1 BP 神经网络的结构

### 2. 输入输出变换关系



$$u_i^k = \sum_{j=1}^{p_{k-1}} w_{ij}^k y_j^{k-1} - \theta_i = \sum_{j=0}^{p_{k-1}} w_{ij}^k y_j^{k-1}$$

$$(y_0^{k-1} = \theta_i, w_{i0}^k = -1)$$

$$y_i^k = f(u_i^k) = \frac{1}{1 + e^{-s_i^k}}$$

$$i = 1, 2, \dots, p_k$$

$$k = 1, 2, \dots, m$$

## 8.2.1 BP 神经网络的结构

### 3. 工作过程

- 第一阶段或网络训练阶段:

  $N$  组输入输出样本:  $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{ip1}]^T$

$$\mathbf{d}_i = [d_{i1}, d_{i2}, \dots, d_{ipm}]^T$$

$$i=1, 2, \dots, N$$

 对网络的连接权进行学习和调整, 以使该网络实现给定样本的输入输出映射关系。

- 第二阶段或称工作阶段: 把实验数据或实际数据输入到网络, 网络在误差范围内预测计算出结果。

## 8.2 BP 神经网络及其学习算法

 **8.2.1 BP 神经网络的结构**

 **8.2.2 BP 学习算法**

 **8.2.3 BP 算法的实现**

## 8.2.2 BP 学习算法

### ■ 两个问题:

- (1) 是否存在一个 **BP** 神经网络能够逼近给定的样本或者函数。

**Kolmogorov 定理:** 给定任意  $\varepsilon > 0$ , 对于任意的  $L_2$  型连续函数  $f: [0,1]^n \rightarrow R^m$ , 存在一个三层 BP 神经网络, 其输入层有  $n$  个神经元, 中间层有  $2n+1$  个神经元, 输出层有  $m$  个神经元, 它可以在任意  $\varepsilon$  平方误差精度内逼近  $f$ 。

- (2) 如何调整 **BP** 神经网络的连接权, 使网络的输入与输出与给定的样本相同。
- ◆ 1986 年, 鲁梅尔哈特 ( **D. Rumelhart** ) 等提出 **BP** 学习算法。

## 8.2.2 BP 学习算法

### 1. 基本思想

- 目标函数: 
$$J = \frac{1}{2} \sum_{j=1}^{p_m} (y_j^m - d_j)^2$$

- 约束条件: 
$$u_i^k = \sum_j w_{ij}^{k-1} y_j^{k-1} \quad i = 1, 2, \dots, p_k$$

$$y_i^k = f_k(u_i^k) \quad k = 1, 2, \dots, m$$

- 连接权值的修正量:

$$\Delta w_{ij}^{k-1} = -\varepsilon \frac{\partial J}{\partial w_{ij}^{k-1}} \quad j = 1, 2, \dots, p_{k-1}$$

## 8.2.2 BP 学习算法

先求 
$$\frac{\partial J}{\partial w_{ij}^{k-1}} = \frac{\partial J}{\partial u_i^k} \frac{\partial u_i^k}{\partial w_{ij}^{k-1}} = \frac{\partial J}{\partial u_i^k} \frac{\partial}{\partial w_{ij}^{k-1}} \left( \sum_j w_{ij}^{k-1} y_j^{k-1} \right) = \frac{\partial J}{\partial u_i^k} y_j^{k-1}$$

记 
$$d_i^k = \frac{\partial J}{\partial u_i^k} = \frac{\partial J}{\partial y_i^k} \frac{\partial y_i^k}{\partial u_i^k} = \frac{\partial J}{\partial y_i^k} f'_k(u_i^k)$$

(1) 对输出层的神经元 
$$\frac{\partial J}{\partial y_i^k} = \frac{\partial J}{\partial y_i^m} = y_i^m - y_{si}$$

$$d_i^m = (y_i^m - y_{si}) f'_m(u_i^m)$$

(2) 对隐单元层, 则有 
$$\frac{\partial J}{\partial y_i^k} = \sum_l \frac{\partial J}{\partial u_l^{k+1}} \frac{\partial u_l^{k+1}}{\partial y_i^k} = \sum_l d_l^{k+1} w_{li}^k$$

$$d_i^k = f'_k(u_i^k) \sum_l d_l^{k+1} w_{li}^k$$

$$\Delta w_{ij}^{k-1} = -\varepsilon d_i^k y_j^{k-1}$$

## 8.2.2 BP 学习算法

### ■ 2. 学习算法

$$\Delta w_{ij}^{k-1} = -\varepsilon d_i^k y_j^{k-1}$$

$$d_i^m = (y_i^m - y_{si}) f'_m(u_i^m)$$

——输出层连接权调整公式

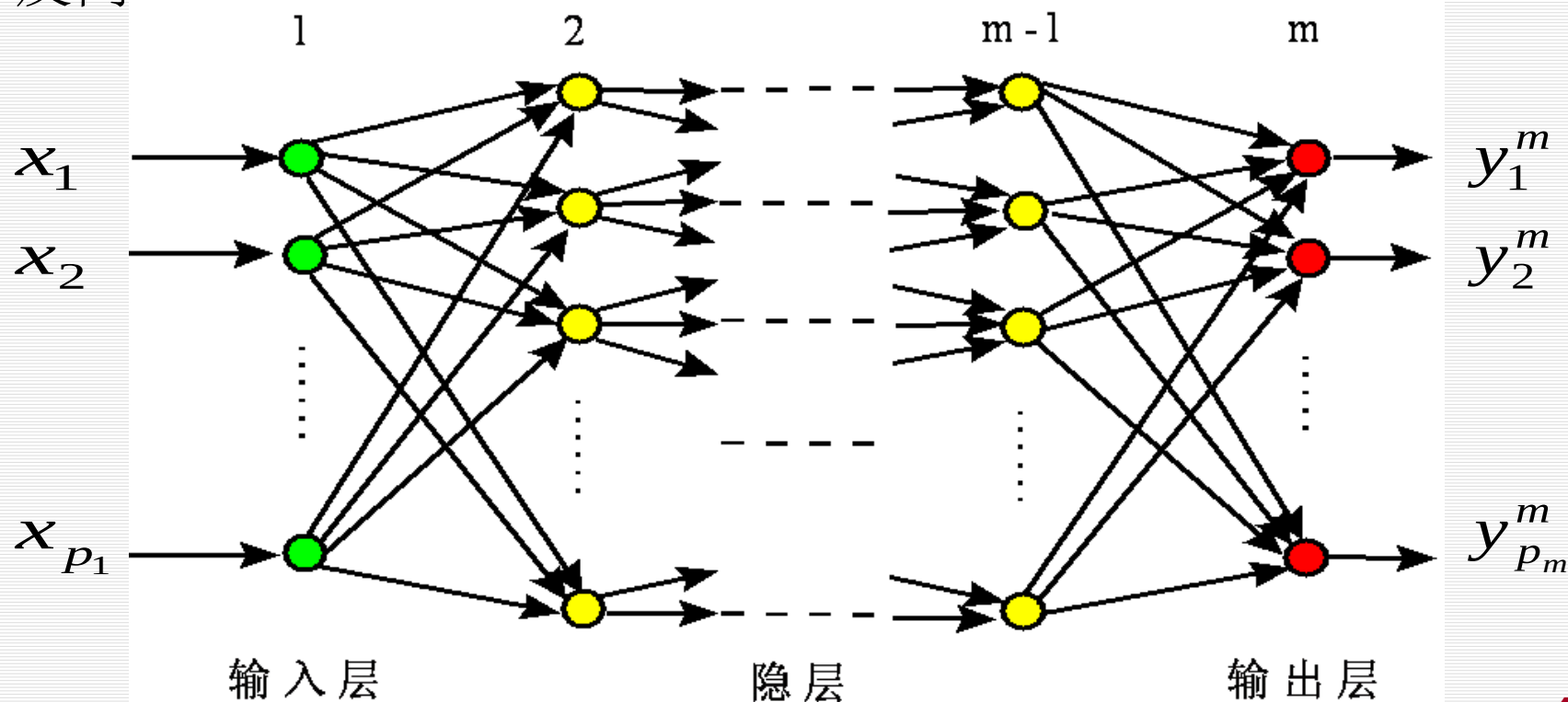
$$d_i^k = f'_k(u_i^k) \sum_l d_l^{k+1} w_{li}^k$$

——隐层连接权调整公式

## 8.2.2 BP 学习算法

### 2. 学习算法

- 正向传播：输入信息由输入层传至隐层，最终在输出层输出。
- 反向传播：修改各层神经元的权值，使误差信号最小。





## 8.2.2 BP 学习算法

### ■ 2. 学习算法

$$\text{当 } y_i^k = \frac{1}{1 + e^{-u_i^k}} \text{ 时}$$

$$\Delta w_{ij}^{k-1} = -\varepsilon d_i^k y_j^{k-1}$$

$$d_i^m = y_i^m (1 - y_i^m) (y_i^m - y_i)$$

——输出层连接权调整公式

$$d_i^k = y_i^k (1 - y_i^k) \sum_{l=1}^{p_{k+1}} w_{li}^{k+1} d_l^{k+1}$$

——隐层连接权调整公式

## 8.2 BP 神经网络及其学习算法

- 8.2.1 BP 神经网络的结构
- 8.2.2 BP 学习算法
- 8.2.3 BP 算法的实现

## 8.2.3 BP 算法的实现

### 1. BP 算法的设计

- （1）隐层数及隐层神经元数的确定：目前尚无理论指导。
- （2）初始权值的设置：一般以一个均值为 **0** 的随机分布设置网络的初始权值。
- （3）训练数据预处理：线性的特征比例变换，将所有特征变换到 **[0, 1]** 或者 **[-1, 1]** 区间内，使得在每个训练集上，每个特征的均值为 **0**，并且具有相同的方差。
- （4）后处理过程：当应用神经网络进行分类操作时，通常将输出值编码成所谓的名义变量，具体的值对应类别标号。

## 8.2.3 BP 算法的实现

### 2. BP 算法的计算机实现流程

- (1) 初始化: 对所有连接权和阈值赋以随机任意小值;  
 $w_{ij}^k(t), \theta_i^k(t), (k=1, \dots, m; i=1, \dots, p_k; j=1, \dots, p_{k-1}; t=0)$
- (2) 从  $N$  组输入输出样本中取一组样本:  $\mathbf{x}=[x_1, x_2, \dots, x_{p_1}]^T$ ,  
 $\mathbf{d}=[d_1, d_2, \dots, d_{p_m}]^T$ , 把输入信息  $\mathbf{x}=[x_1, x_2, \dots, x_{p_1}]^T$  输入到 BP 网络中
- (3) 正向传播: 计算各层节点的输出 ( $k=1, \dots, m$ )
- (4) 计算网络的实际输出与期望输出的误差 ( $i=1, \dots, p_m$ )

## 8.2.3 BP 算法的实现

### 2. BP 算法的计算机实现流程

（5）反向传播：从输出层方向计算到第一个隐层，按连接权值修正公式向减小误差方向调整网络的各个连接权值。

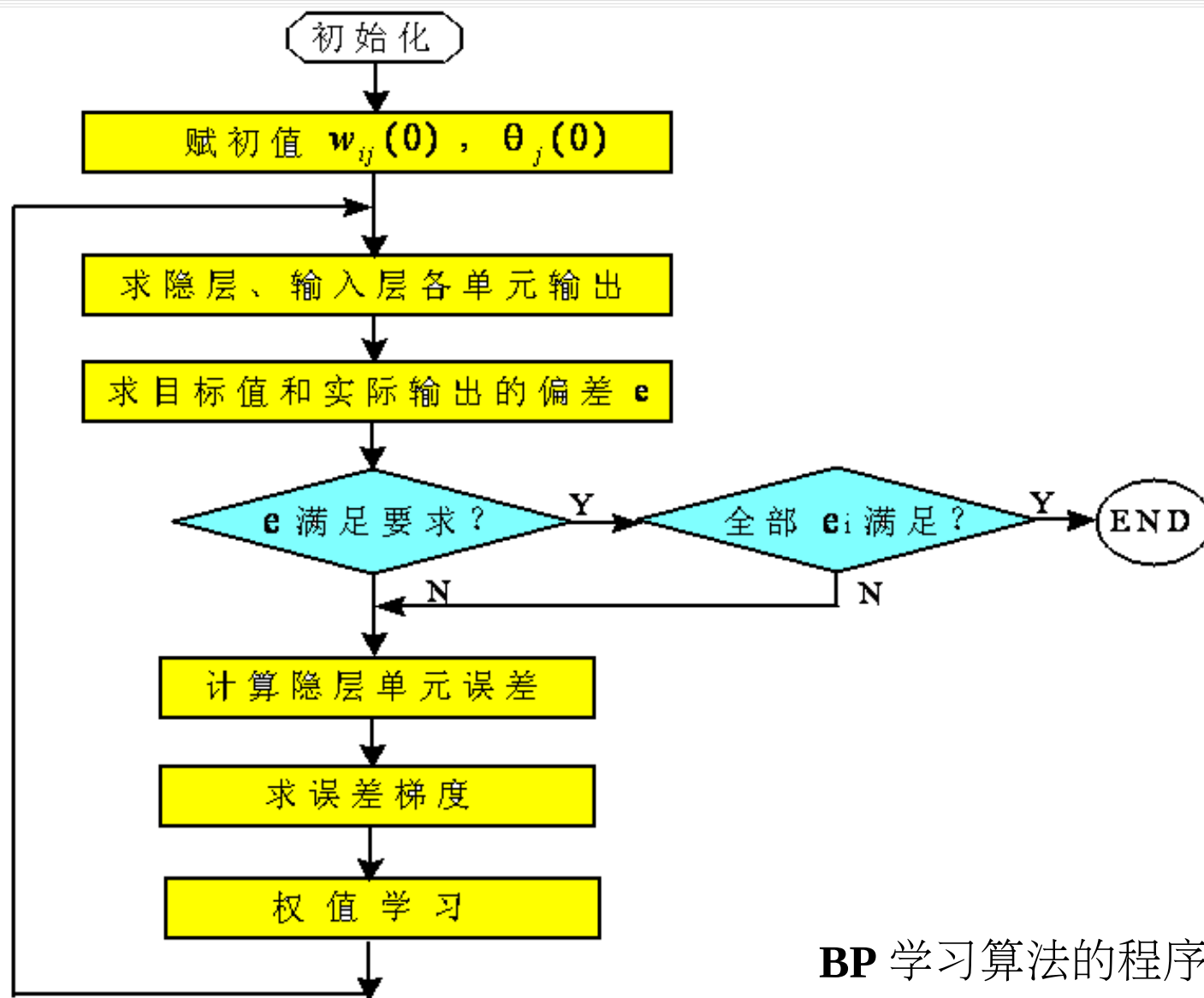
$$\Delta w_{ij} = -\alpha d_i^k y_j^{k-1}$$

$$d_i^m = y_i^m (1 - y_i^m)(y_i^m - y_i) \quad \text{—— 输出层连接权调整公式}$$

$$d_i^k = y_i^k (1 - y_i^k) \sum_{l=1}^{p_{k+1}} w_{li}^{k+1} d_l^{k+1} \quad \text{—— 隐层连接权调整公式}$$

（6）让  $t+1 \rightarrow t$ ，取出另一组样本重复（2）—（5），直到  $N$  组输入输出样本的误差达到要求时为止。

## 8.2.3 BP 算法的实现



BP 学习算法的程序框图

## 8.2.4 BP 算法的特点分析

### □ 1. 特点

- **BP 网络**：多层前向网络（输入层、隐层、输出层）。
- 连接权值：通过 **Delta** 学习算法进行修正。
- 神经元传输函数：**S** 形函数。
- 学习算法：正向传播、反向传播。
- 层与层的连接是单向的，信息的传播是双向的。

## 8.2.4 BP 算法的特点分析

### □ 2. BP 网络的主要优缺点

#### ■ 优点

- 很好的逼近特性。
- 具有较强的泛化能力。
- 具有较好的容错性。

#### ■ 缺点

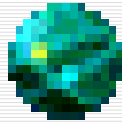
- 收敛速度慢。
- 局部极值。
- 难以确定隐层和隐层结点的数目。



## 8.3 BP 神经网络的应用



### 8.3.1 BP 神经网络在模式识别中的应用



### 8.3.2 BP 神经网络在软测量中的应用

## 8.3.1 BP 神经网络在模式识别中的应用

模式识别研究用计算机模拟生物、人的感知，对模式信息，如图像、文字、语音等，进行识别和分类。

传统人工智能的研究部分地显示了人脑的归纳、推理等智能。但是，对于人类底层的智能，如视觉、听觉、触觉等方面，现代计算机系统的信息处理能力还不如一个幼儿园的孩子。

神经网络模型模拟了人脑神经系统的特点：处理单元的广泛连接；并行分布式信息储存、处理；自适应学习能力等。

神经网络模式识别方法具有较强的容错能力、自适应学习能力、并行信息处理能力。

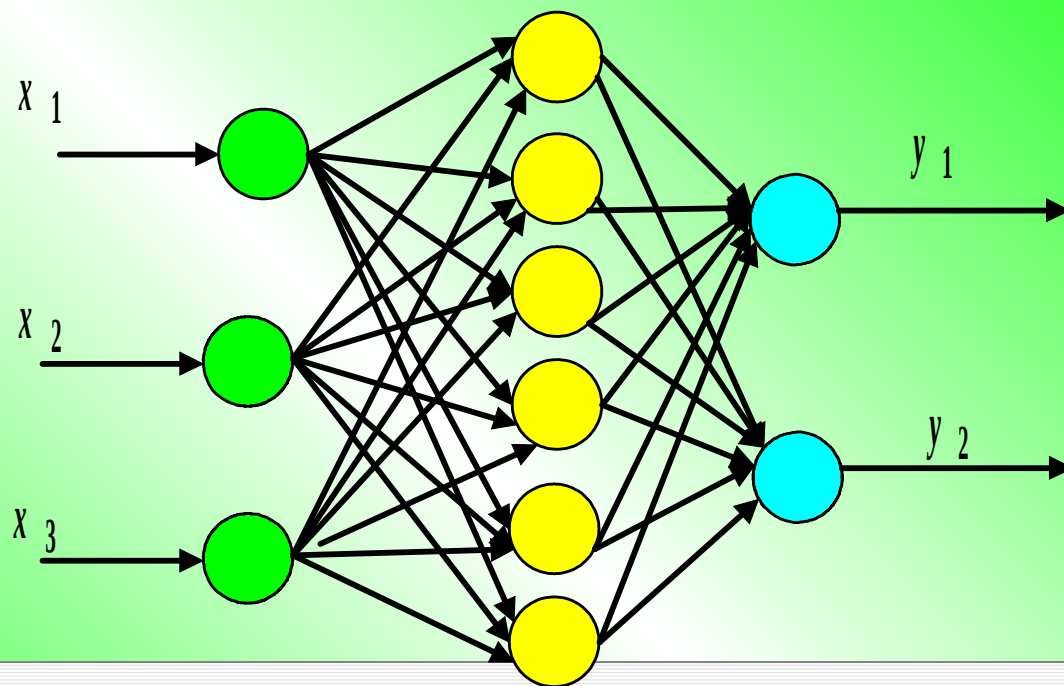
## 8.3.1 BP 神经网络在模式识别中的应用

- 例 输入输出样本:

输入			输出	
1	0	0	1	0
0	1	0	0	0.5
0	0	1	0	1

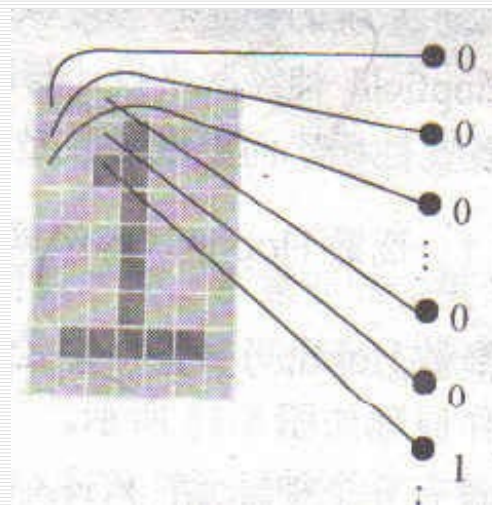
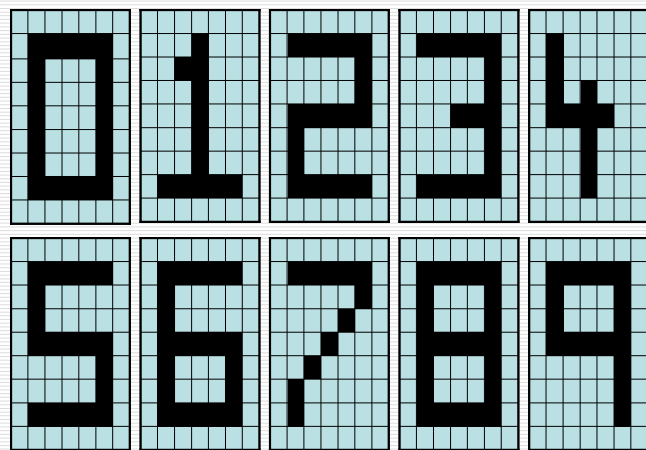
- 测试数据:

输入		
0.97	0.001	0.001
0	0.98	0
0.002	0	1.04
0.5	0.5	0.5
1	0	0
0	1	0
0	0	1



## 8.3.1 BP 神经网络在模式识别中的应用

例 设计一个三层 BP 网络对数字 0 至 9 进行分类。



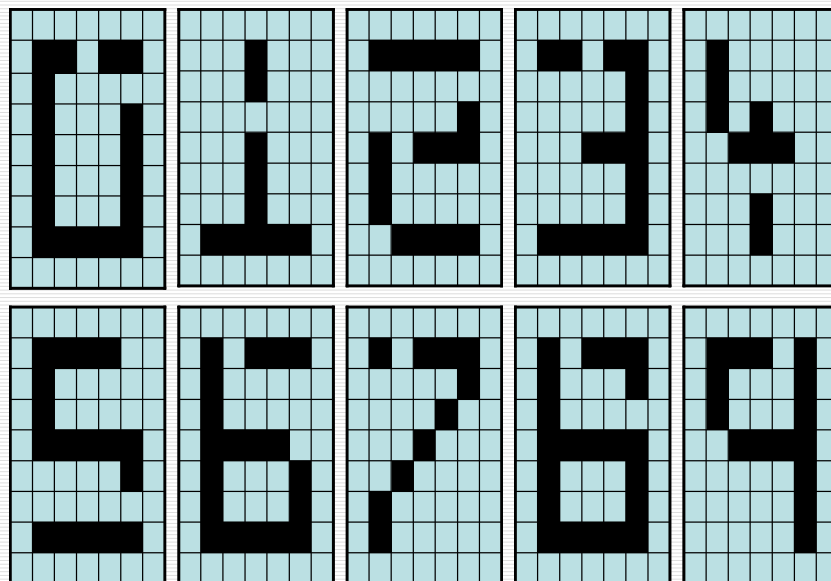
每个数字用  $9 \times 7$  的网格表示，灰色像素代表 0，黑色像素代表 1。将每个网格表示为 0, 1 的长位串。位映射由左上角开始向下直到网格的整个一列，然后重复其他列。

选择 BP 网络结构为 63-6-9。97 个输入结点，对应上述网格的映射。9 个输出结点对应 10 种分类。

使用的学习步长为 0.3。训练 600 个周期，如果输出结点的值大于 0.9，则取为 ON，如果输出结点的值小于 0.1，则取为 OFF。

## 8.3.1 BP 神经网络在模式识别中的应用

当训练成功后，对如图所示测试数据进行测试。测试数据都有一个或者多个位丢失。

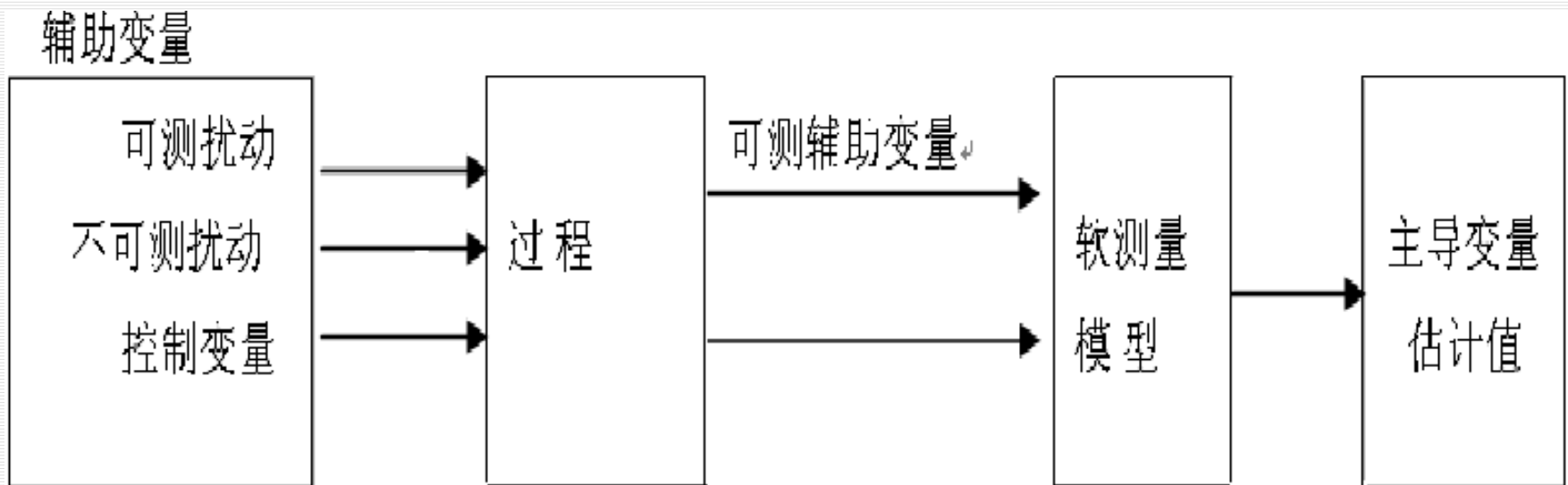


测试结果表明：除了 8 以外，所有被测的数字都能够被正确地识别。

对于数字 8，神经网络的第 6 个结点的输出值为 0.53，第 8 个结点的输出值为 0.41，表明第 8 个样本是模糊的，可能是数字 6，也可能是数字 8，但也不完全确信是两者之一。

## 8.3.2 BP 神经网络在软测量中的应用

### □ 软测量技术



□ 主导变量：被估计的变量。

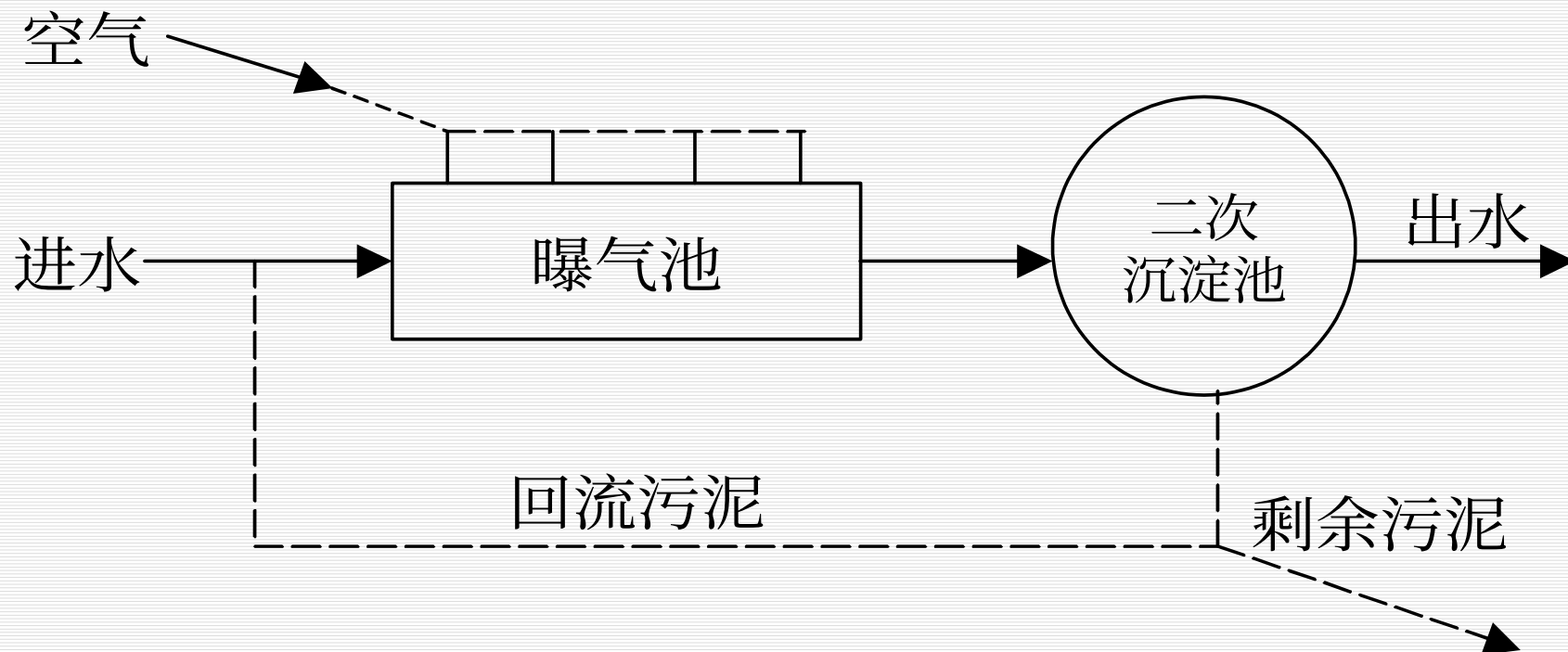
□ 辅助变量：与被估计变量相关的一组可测变量。

## 8.3.2 BP 神经网络在软测量中的应用

- 软测量系统的设计：
  - 辅助变量的选择：变量类型、变量数量和检测点位置的选择。
  - 数据采集与处理。
  - 软测量模型的建立：通过辅助变量来获得对主导变量的最佳估计。

## 8.3.2 BP 神经网络在软测量中的应用

### ■ 序批式活性污泥法（**SBR**）



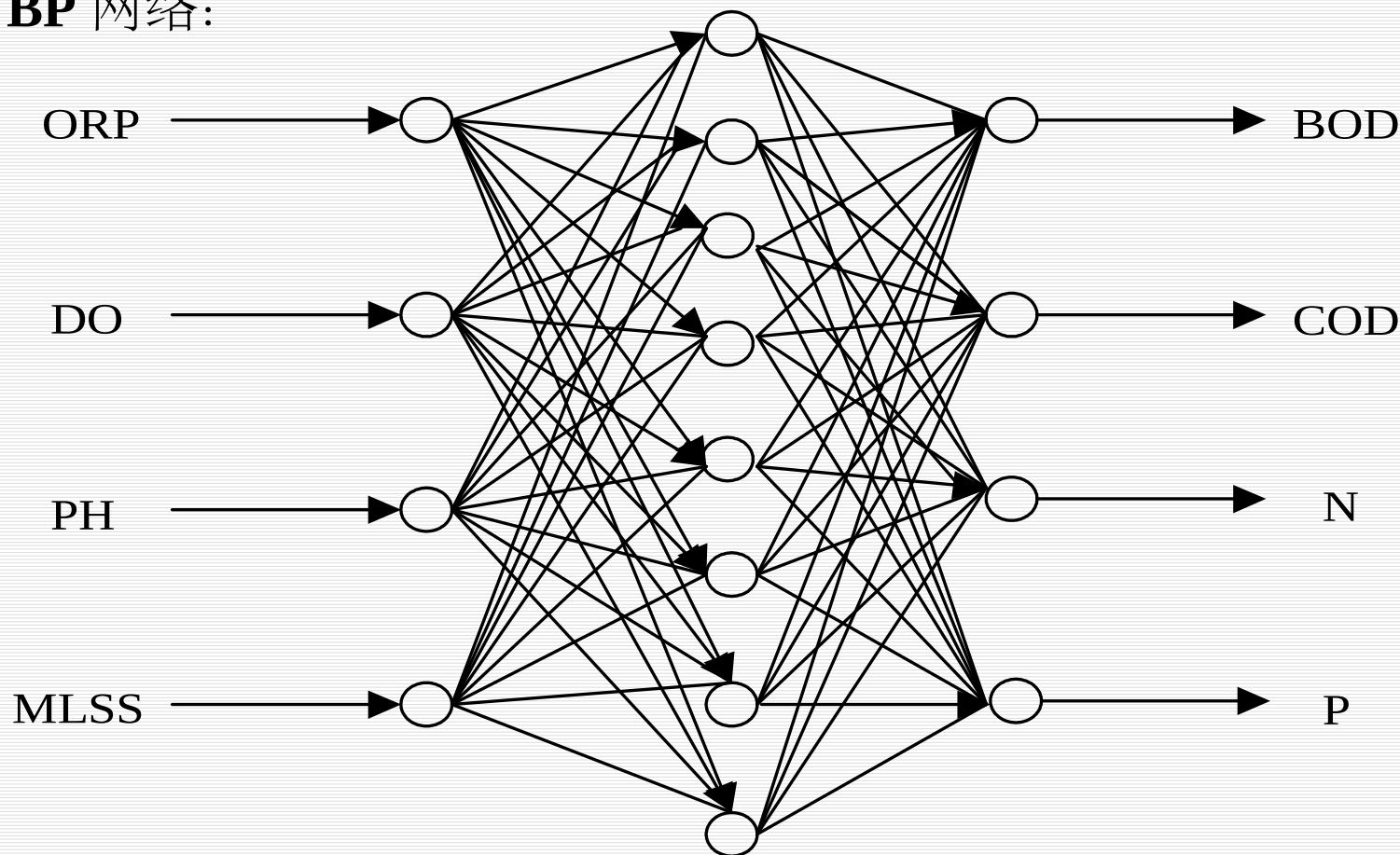


## 8.3.2 BP 神经网络在软测量中的应用

**BOD**、**COD**、**N** 和 **P**：为软测量模型的主导变量。

**ORP**、**DO**、**PH** 和 **MLSS**：辅助变量。

三层 **BP** 网络：



# 第 8 章 人工神经网络及其应用

- 8.1 神经元与神经网络
- 8.2 BP 神经网络及其学习算法
- 8.3 BP 神经网络的应用
- ✓ 8.4 Hopfield 神经网络及其改进
- 8.5 Hopfield 神经网络的应用
- 8.6 Hopfield 神经网络优化方法求解 JSP
- 8.7 卷积神经网络 及其应用

## 8.4 Hopfield 神经网络及其改进

- 8.4.1 离散型 **Hopfield** 神经网络
- 8.4.2 连续型 **Hopfield** 神经网络及其 **VLSI** 实现
- 8.4.3 随机神经网络
- 8.4.4 混沌神经网络

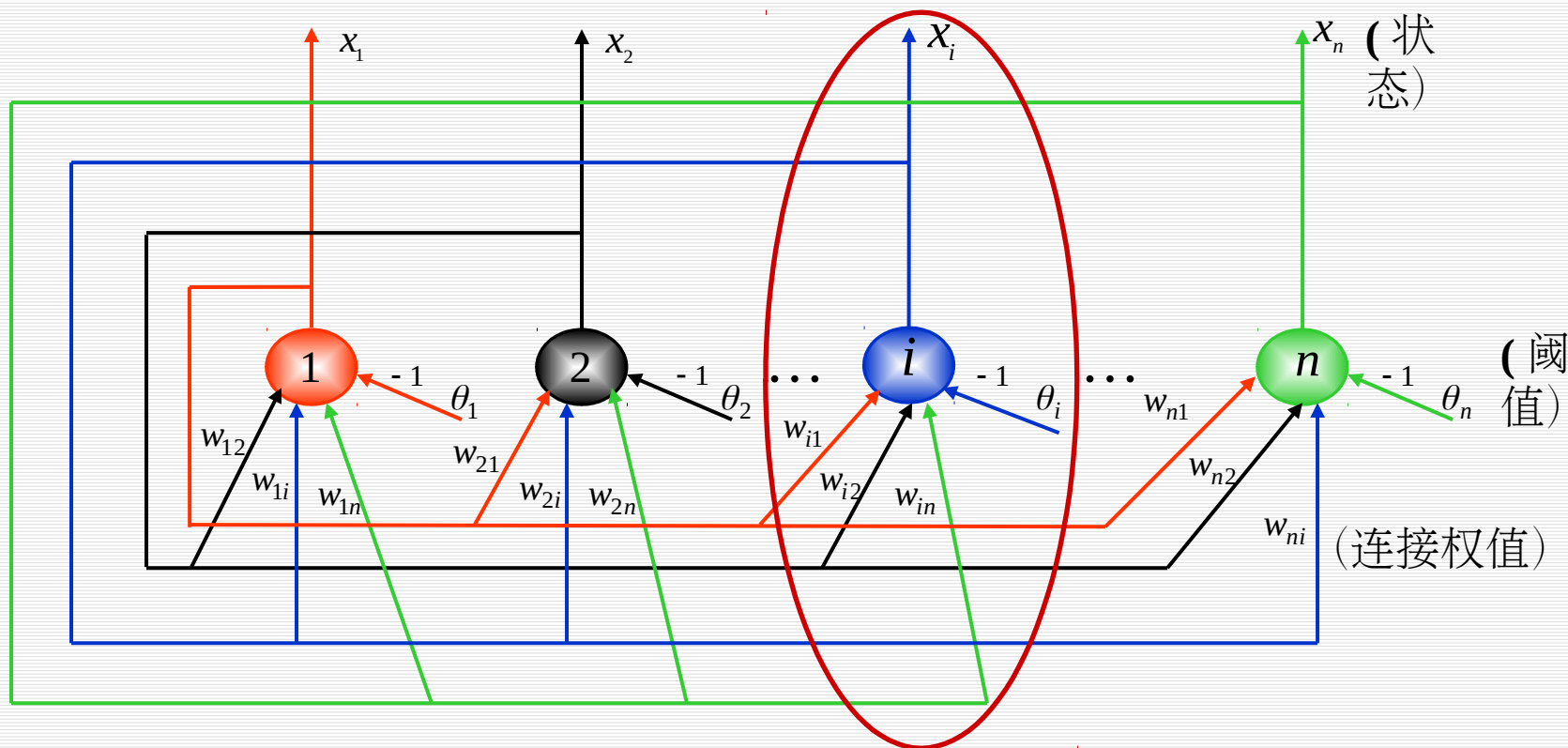
## 8.4 Hopfield 神经网络及其改进

- 8.4.1 离散型 **Hopfield** 神经网络
- 8.4.2 连续型 **Hopfield** 神经网络及其 **VLSI** 实现
- 8.4.3 随机神经网络
- 8.4.4 混沌神经网络

## 8.4.1 离散 Hopfield 神经网络

### 1. 离散 Hopfield 神经网络模型

#### ■ 网络结构:

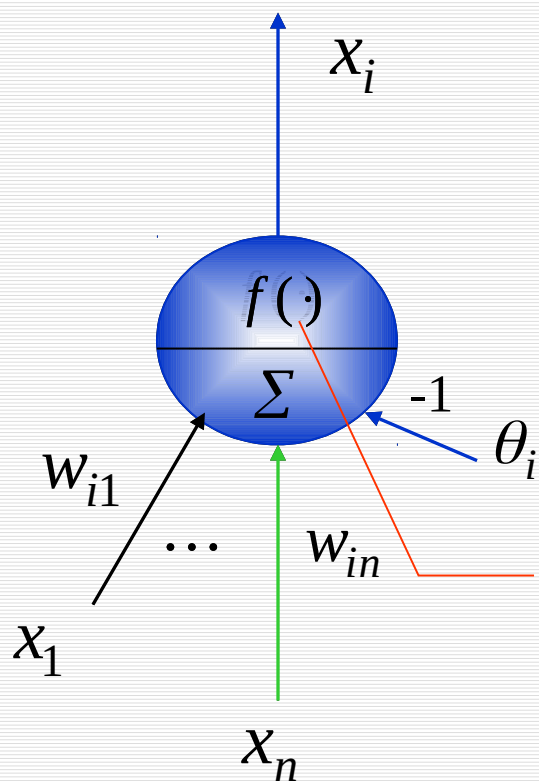


离散 Hopfield 神经网络结构图

## 8.4.1 离散 Hopfield 神经网络

### 1. 离散 Hopfield 神经网络

■ 输入输出关系:



$$x(k+1) = f(W_x(k) - \theta), \forall i$$

$$\mathbf{x} = [x_1 \ x_2 \ \cdots \ x_n]^T \quad \boldsymbol{\theta} = [\theta_1 \ \theta_2 \ \cdots \ \theta_n]^T$$

$$\mathbf{W} = [w_{ij}]_{n \times n} \quad \mathbf{f}(s) = [f(s_1) \ f(s_2) \ \cdots \ f(s_n)]^T$$

$$x_i(k+1) = f\left(\sum_{j=1}^n w_{ij} x_j(k) - \theta_i\right)$$

注:  $w_{ii} = 0$

$$f(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases} \quad \text{或} \quad f(s) = \begin{cases} 1 & s \geq 0 \\ 0 & s < 0 \end{cases}$$

## 8.4.1 离散 Hopfield 神经网络

### 1. 离散 Hopfield 神经网络模型

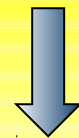
■ 工作方式:

异步（串行）方式：

$$\begin{cases} x_i(k+1) = f\left(\sum_{j=1}^n w_{ij} x_j(k) - \theta_i\right) \\ x_j(k+1) = x_j(k), \quad j \neq i \end{cases}$$

同步（并行）方式：

$$x_i(k+1) = f\left(\sum_{j=1}^n w_{ij} x_j(k) - \theta_i\right), \quad \forall i$$



$$x(k+1) = f(W_x(k) - \theta), \quad \forall i$$

## 8.4.1 离散 Hopfield 神经网络

### 1. 离散 Hopfield 神经网络模型

- 工作过程:

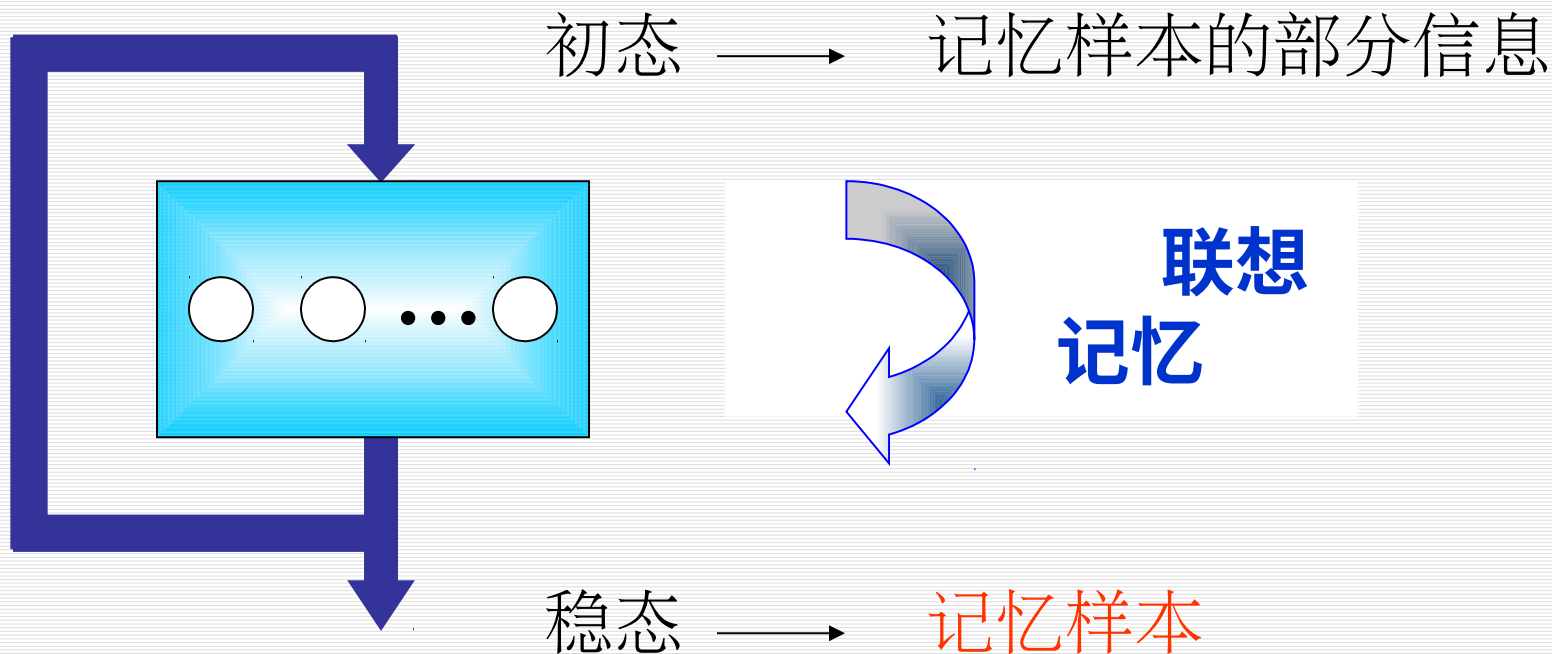




## 8.4.1 离散 Hopfield 神经网络

### 1. 离散 Hopfield 神经网络模型

- 工作过程:



## 8.4.1 离散 Hopfield 神经网络

### 2. 网络的稳定性

- 稳定性定义:

- 若从某一时刻开始，网络中所有神经元的状态不再改变，即  $x = f(w_x - \theta)$ ，则称该网络是稳定的，为网络的稳定点或吸引子。
- **Hopfield** 神经网络是高维非线性系统，可能有许多稳定状态。从任何初始状态。这些稳定状态  $x(k) = f(W_x(k) - \theta) = x(k+1)$  某个稳定状态。

## 8.4.1 离散 Hopfield 神经网络

### 2. 网络的稳定性

- 稳定性定理证明：1983 年，科恩（**Cohen**）、葛劳斯伯格（**S. Grossberg**）。
- 稳定性定理（**Hopfield**）
  - 串行稳定性 ——  $W$ ：对称阵
  - 并行稳定性 ——  $W$ ：非负定对称阵

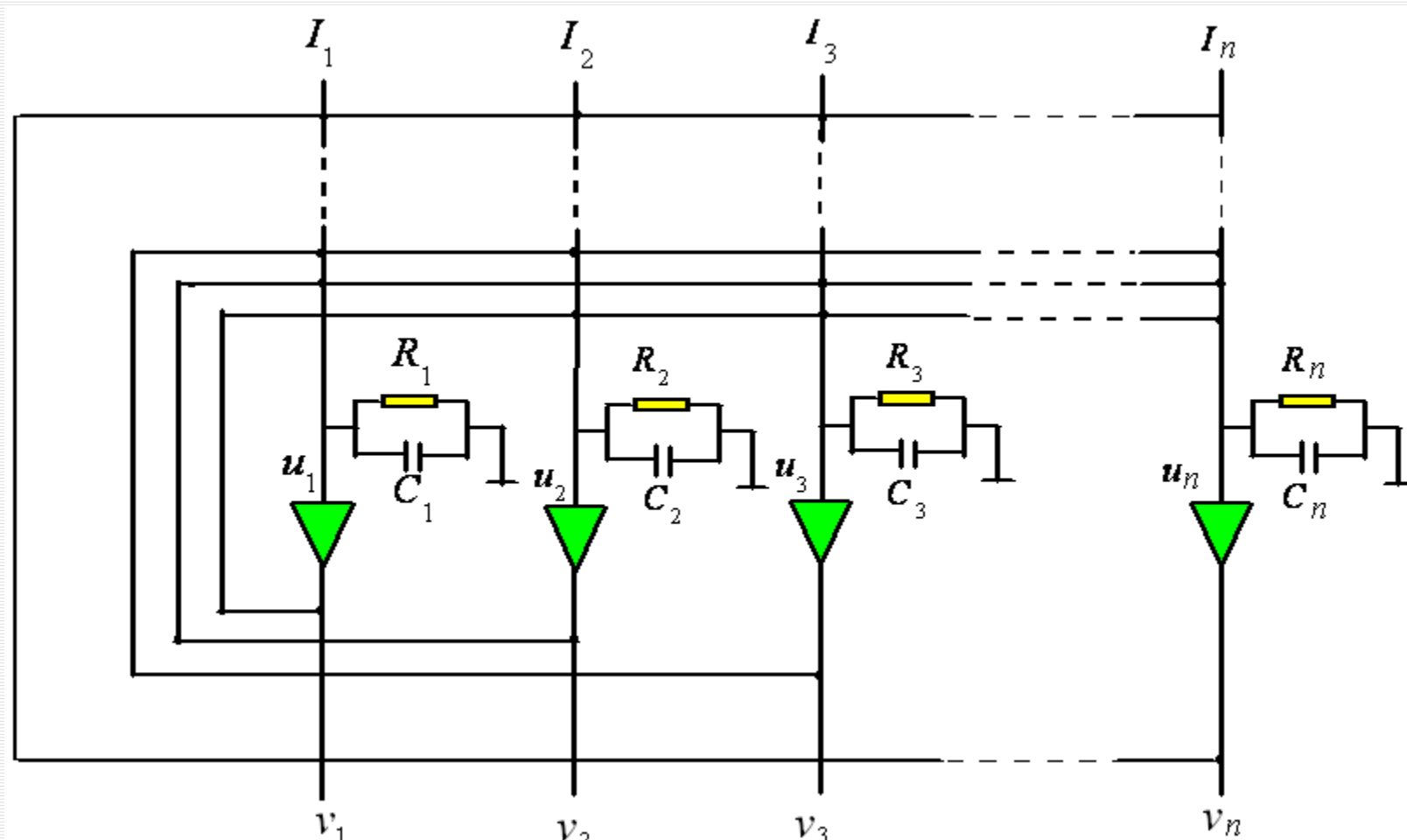
## 8.4 Hopfield 神经网络及其改进

 **8.4.1** 离散型 **Hopfield** 神经网络

 **8.4.2** 连续型 **Hopfield** 神经网络及其 **VLSI** 实现

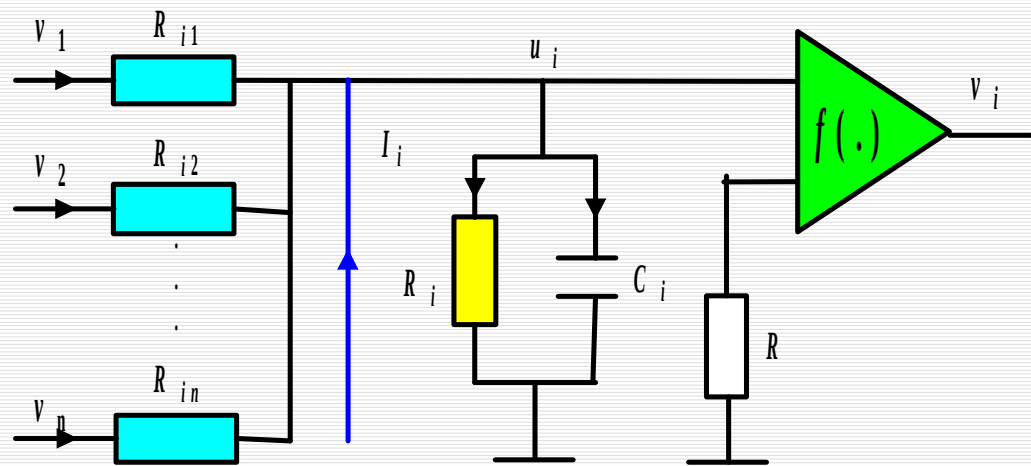
- 连续 **Hopfield** 神经网络模型
- 网络的稳定性
- 应用举例

## 1. 连续 Hopfield 神经网络模型



# 3.1.2 连续型 Hopfield 神经网络及其 VLSI 实现

## 1. 连续 Hopfield 神经网络模型



$$\begin{cases} \frac{u_i}{R_i} + C_i \frac{du_i}{dt} = \sum_{j=1}^n \frac{v_j - u_i}{R_{ij}} + I_i \\ v_i = f(u_i) \end{cases}$$

$$\frac{1}{R'_i} = \frac{1}{R_i} + \sum_{j=1}^n \frac{1}{R_{ij}}$$

$$w_{ij} = \frac{1}{R_{ij}}$$

$$C_i \frac{du_i}{dt} = -\frac{u_i}{R'_i} + \sum_{j=1}^n w_{ij} v_j + I_i$$

$$v_i = f(u_i) = \frac{1}{1 + e^{-\frac{2u_i}{u_0}}} = \frac{1}{2} [1 + \tanh(\frac{u_i}{u_0})]$$

## 2. 网络的稳定性

- 计算能量函数：

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n w_{ij} v_i v_j - \sum_{i=1}^n v_i I_i + \sum_{i=1}^n \frac{1}{R'_i} \int_0^{v_i} f^{-1}(v) dv$$

- 定理：对于连续型 **Hopfield** 神经网络， $f^{-1}(\cdot)$  为单调递增的连续函数， $C_i > 0$ ， $w_{ij} = \frac{dE}{dv_j} \leq 0$  ；  
当且仅当  $\frac{dv_i}{dt} = 0$ ， $(1 \leq i \leq n)$  时， $\frac{dE}{dt} = 0$



## 8.4.3 随机神经网络

- **Hopfield** 神经网络中，神经元状态为 **1** 是根据其输入是否大于阈值确定的，是确定性的。
- 随机神经网络中，神经元状态为 **1** 是随机的，服从一定的概率分布。例如，服从玻尔兹曼 (**Boltzmann**)、高斯 (**Gaussian**)、柯西 (**Cauchy**) 分布等，从而构成玻尔兹曼机、高斯机、柯西机等随机机。

## 8.4.3 随机神经网络

### 1. Boltzmann 机

- 1985 年，加拿大多伦多大学教授欣顿 (**Hinton**) 等人借助统计物理学的概念和方法，提出了 **Boltzmann** 机神经网络模型。
- **Boltzmann** 机是离散 **Hopfield** 神经网络的一种变型，通过对离散 **Hopfield** 神经网络加以扰动，使其以概率的形式表达，而网络的模型方程不变，只是输出值类似于 **Boltzmann** 分布以概率分布取值。
- **Boltzmann** 机是按 **Boltzmann** 概率分布动作的神经网络。

## 8.4.3 随机神经网络

### 1. Boltzmann 机 (续)

□ 离散 Hopfield 神经网络的输出:

$$v_i(t+1) = \text{sgn}\left(\sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j(t) - \theta_i\right)$$

□ Boltzman 机的内部状态:

$$I_i = \sum_{\substack{j=1 \\ j \neq i}}^n w_{ij} v_j(t) - \theta_i$$

□ 神经元 $i$  输出值为 0 和 1 时的概率:

$$p_i(1) = \frac{1}{1 + e^{-I_i/T}} \quad p_i(0) = 1 - p_i(1)$$

## 8.4.3 随机神经网络

### 1. Boltzmann 机 (续)

□ Boltzmann 的能量函数:

$$E = -\frac{1}{2} \sum_i \sum_{j \neq i} w_{ij} v_i v_j - \sum_i v_i \theta_i$$

- 神经元  $i$  状态转换时网络能量的变化:

$$\Delta E_i = \sum_j w_{ij} v_j + \theta_i$$

- 神经元  $i$  改变为状态“1”的概率:

$$p_i = \frac{1}{1 + \exp(-\frac{\Delta E_i}{T})}$$

## 8.4.3 随机神经网络

### 2. 高斯机

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^n w_{ji} \eta_j$$

$\eta_j$  均值为 0 的高斯随机变量（白噪声），其方差为  $\sigma = cT$

### 3. 柯西机

$$\frac{du_i}{dt} = -u_i + \sum_{j=1}^n w_{ji} \eta_j$$

$\eta_j$  柯西随机变量（有色噪声）

## 8.4.4 混沌神经网络

### 1. 混沌

- 混沌：自然界中一种较为普遍的非线性现象，其行为看似混乱复杂且类似随机，却存在精致的内在规律性。
- 混沌的性质：
  - （1）随机性：类似随机变量的杂乱表现。
  - （2）遍历性：不重复地历经一定范围内的所有状态。
  - （3）规律性：由确定性的迭代式产生。

## 8.4.4 混沌神经网络

### 1. 混沌（续）

- 混沌学的研究热潮开始于 20 世纪 70 年代初期。
- 1963 年，Lorenz 在分析气候数据时发现：初值十分接近的两条曲线的最终结果会相差很大，从而获得了混沌的第一个例子。
- 1975 年，Li-Yorke 的论文《周期 3 意味着混沌》使“混沌”一词首先出现在科技文献中。混沌的发现，对科学的发展具有深远的影响。

## 8.4.4 混沌神经网络

### 2. 混沌神经元

□ 混沌神经元（**1987** 年，**Freeman**）：构造混沌神经网络的基本单位。

■ 混沌神经元模型：

$$y(t+1) = ky(t) - F(x(t)) + I(t) - \theta$$

$$x(t+1) = G(y(t+1))$$



## 8.4.4 混沌神经网络

### 3. 混沌神经网络

- 1990 年， **Aihara** 等提出了第一个混沌神经网络模型 (**chaotic neural network** , **CNN**) 。
- 1991 年， **Inoue** 等利用两个混沌振荡子耦合成一个神经元的方法，构造出一个混沌神经计算机。
- 1992 年， **Nozawa** 基于欧拉离散化的 **Hopfield** 神经网络，通过增加一个大的自反馈项，得到了一个与 **Aihara** 等提出的类似的 **CNN** 模型。

## 8.4.4 混沌神经网络

### 3. 混沌神经网络

#### (1) 基于模拟退火策略的自抑制混沌神经网络

1995 年, **Chen** 等提出的暂态混沌神经网络 (**transient chaotic neural network**, **TCNN**) :

$$v_i(t) = f(u_i(t)) = \frac{1}{1 + e^{-u_i(t)/\varepsilon}}$$

$$u_i(t+1) = ku_i(t) + \alpha \left[ \sum_j^n w_{ij} v_j(t) + I_i \right] - z_i(t) [v_i(t) - I_0]$$

$$z_i(t+1) = (1 - \beta) z_i(t)$$

## 8.4.4 混沌神经网络

### 3. 混沌神经网络

#### (1) 基于模拟退火策略的自抑制混沌神经网络

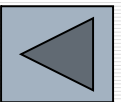
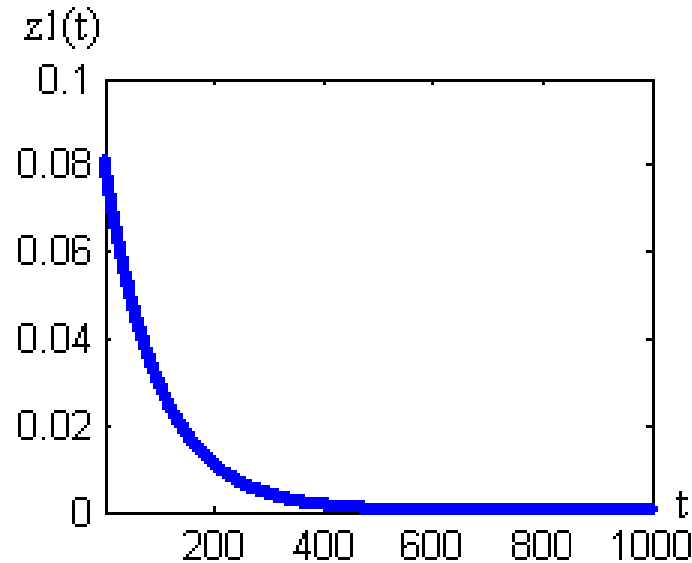
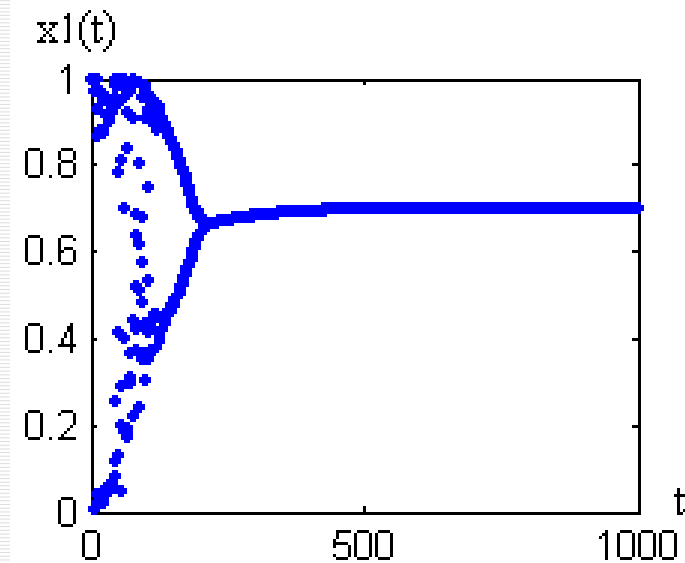
- ① 具有暂态混沌特性。
- ② 能演化到一个稳定状态。
- ③ 搜索区域为一分形结构。
- ④ 具有混沌退火机制。
- ⑤ 一种广义的混沌神经网络。
- ⑥ 可求解 **0-1** 问题，也可求解连续非线性优化问题。

## 8.4.4 混沌神经网络

□ 非线性函数:

$$E(x_1, x_2) = (x_1 - 0.7)^2 [(x_2 + 0.6)^2 + 0.1] + (x_2 - 0.5)^2 [(x_1 + 0.4)^2 + 0.15]$$

$$\varepsilon = 150, \quad k = 1.0, \quad \alpha = 0.015, \quad \beta = 0.01, \quad I_0 = 0.5, \quad z(0) = [-0.082, 0.082],$$



## 8.4.4 混沌神经网络

### 3. 混沌神经网络

#### (2) 基于加大时间步长的混沌神经网络

□ CHNN 的欧拉离散化:

$$u_i(t + \Delta t) = (1 - \frac{\Delta t}{\tau})u_i(t) + \Delta t[\sum_j^n w_{ij}v_j(t) + I_i]$$

- 1998 年, **Wang** 和 **Smith** 采用加大时间步长产生混沌:

$$\Delta t(t+1) = (1 - \beta)\Delta t(t) \quad 0 < \beta < 1$$

## 8.4.4 混沌神经网络

### 3. 混沌神经网络

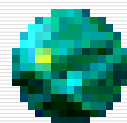
#### (3) 引入噪声的混沌神经网络

□ 1995 年, **Hayakawa** 等的混沌神经网络:

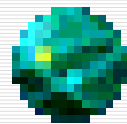
$$u_i(t + \Delta t) = (1 - \frac{\Delta t}{\tau})u_i(t) + \Delta t[\sum_j^n w_{ij}v_j(t) + I_i]$$

$$v_i(t) = f(u_i(t) + A\eta_i(t))$$

## 8.5 Hopfield 神经网络的应用

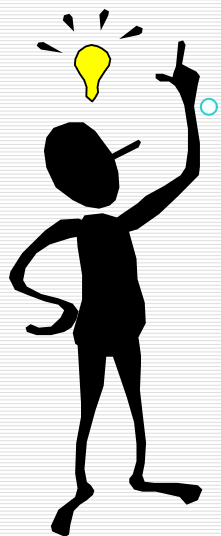


### 8.5.1 Hopfield 神经网络在联想记忆中的应用



### 8.5.2 Hopfield 神经网络优化方法

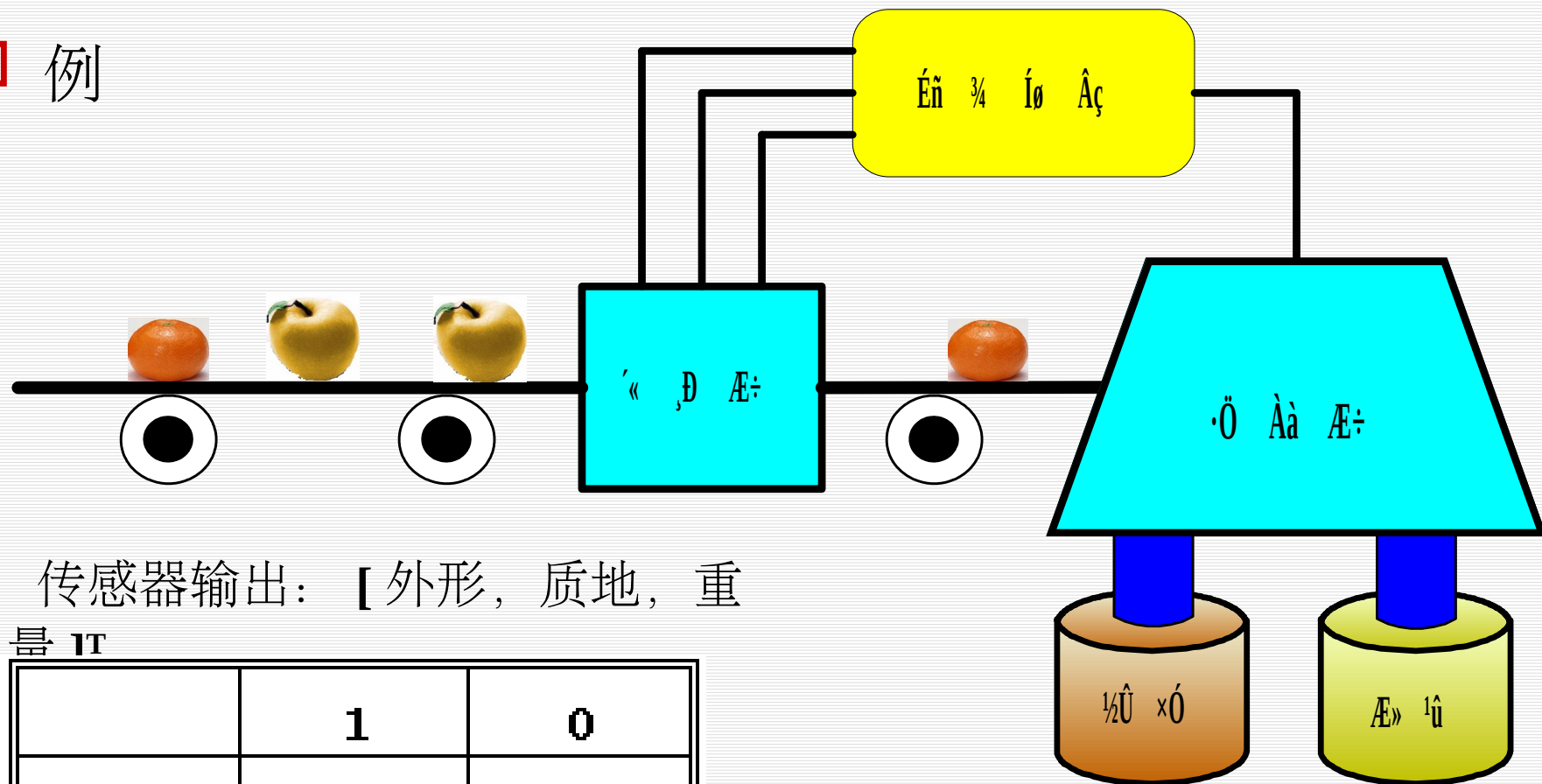
如何实现 HNN 的联想记忆功能？



- 网络能够通过联想来输出和输入模式最为相似的样本模式。



## □ 例



- 传感器输出：【外形，质地，重量】

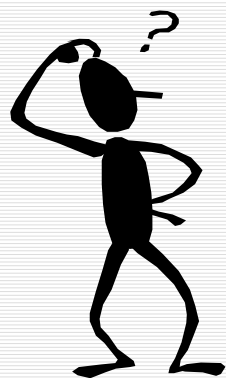
	1	0
外形	圆形	椭圆形
质地	光滑	粗糙
重量	< 1 磅	> 1 磅

标准桔子：  $x^{(1)} = [1 \ 0 \ 1]^T$

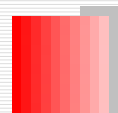
标准苹果：  $x^{(2)} = [0 \ 1 \ 0]^T$

- 例
  - 传感器输出：【外形，质地，重量】<sup>T</sup>
  - 样本： $x^{(1)} = [1, 0, 1]^T$  (桔子)       $x^{(2)} = [0, 1, 0]^T$  (苹果)

具体怎样实现联想记忆？

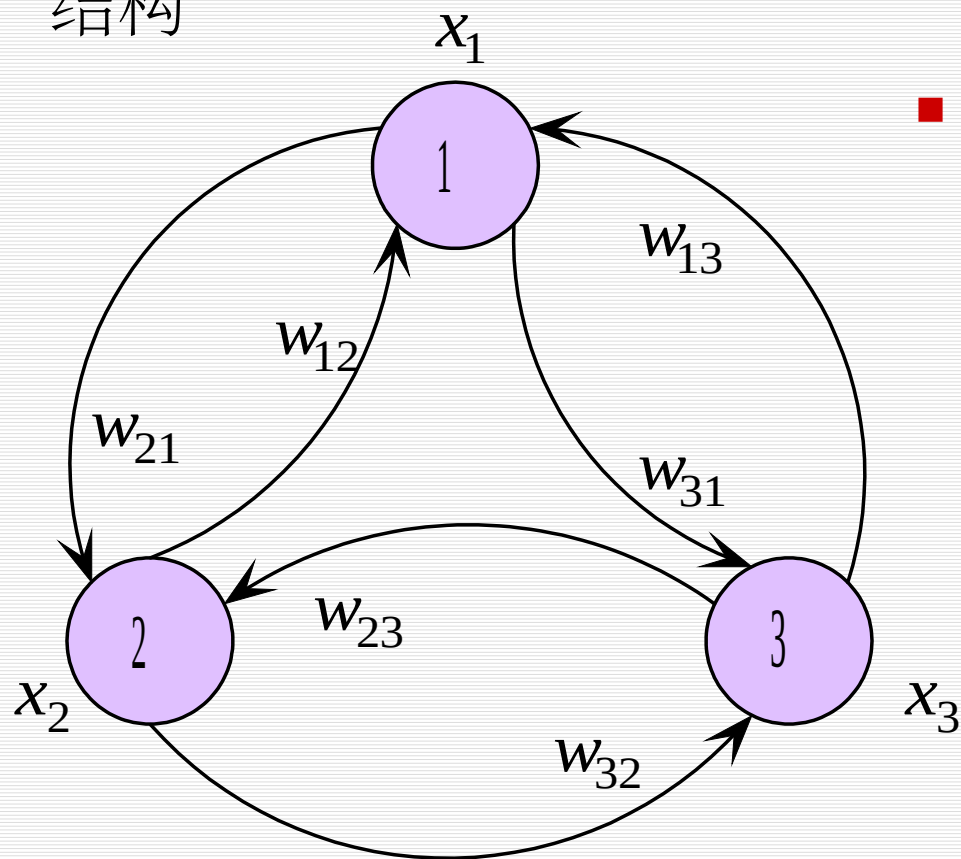


- 步骤：
  - (1) 设计 **DHNN** 结构
  - (2) 设计连接权矩阵
  - (3) 测试



## (1) 设计 DHNN

结构



3 神经元的 DHNN 结构图

- 样本： $x^{(1)} = [1, 0, 1]^T$  (桔子)  
 $x^{(2)} = [0, 1, 0]^T$  (苹果)

注： $\theta_i = 0 (i = 1, 2, 3)$

## ■ (2) 设计连接权矩阵

■ 样本  $x^{(1)} = [1, 0, 1]^T$  (桔子)  $x^{(2)} = [0, 1, 0]^T$  (苹果)

■ 连接权:  $w_{ij} = \begin{cases} \sum_{l=1}^2 (2x_i^{(l)} - 1)(2x_j^{(l)} - 1) & i \neq j \\ 0 & i = j \end{cases}$

$$w_{ji} = w_{ij} \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, n)$$

$$= (2 \times 1 - 1) \times (2 \times 0 - 1) + (2 \times 0 - 1) \times (2 \times 1 - 1)$$

$$= -1 - 1 = -2$$

$$w_{21} = w_{12} = -2$$

## ■ (2) 设计连接权矩阵

■ 样本： $x^{(1)} = [1, 0, 1]^T$        $x^{(2)} = [0, 1, 0]^T$

■ 连接权： $w_{ij} = \begin{cases} \sum_{l=1}^2 (2x_i^{(l)} - 1)(2x_j^{(l)} - 1) & i \neq j \\ 0 & i = j \end{cases}$

$$w_{ji} = w_{ij} \quad (i = 1, 2, \dots, n; j = 1, 2, \dots, n)$$

~~$$w_{32} = w_{23} = (2 \times 1 - 1) \times (2 \times 0 - 1) + (2 \times 0 - 1) \times (2 \times 1 - 1)$$~~

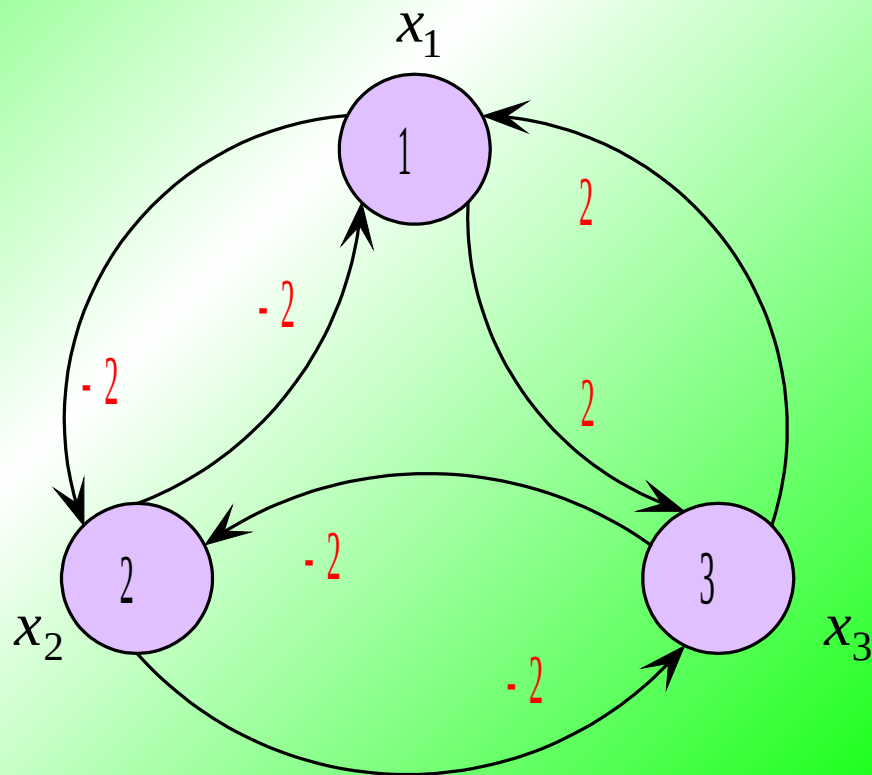
$$= (2 \times 0 - 1) \times (2 \times 1 - 1) + (2 \times 1 - 1) \times (2 \times 0 - 1)$$

$$= -1 + (-1) = -2$$

$$w_{32} = w_{23} = -2$$

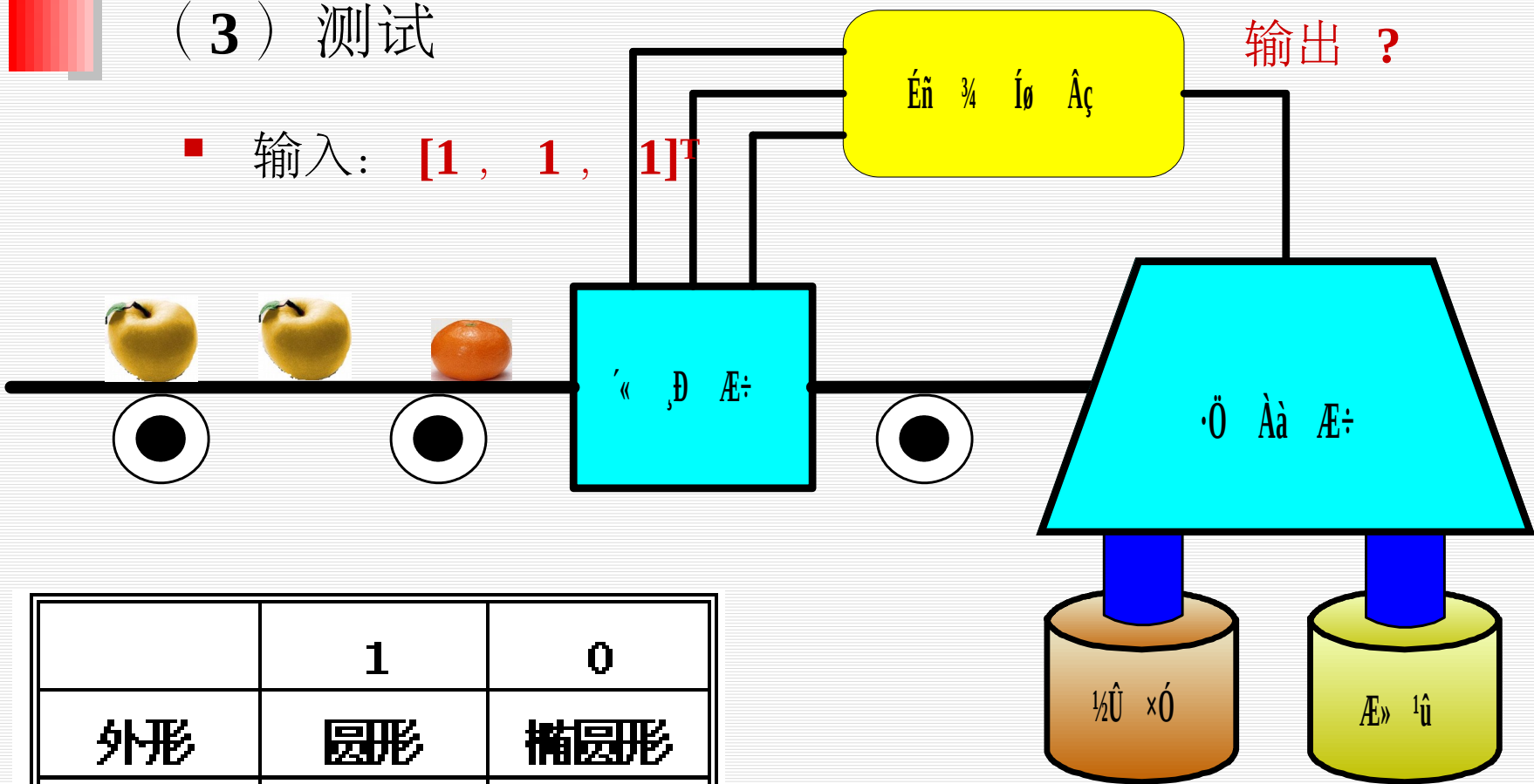
## （2）设计连接权矩阵

$$W = \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$



## (3) 测试

输入:  $[1, 1, 1]^T$



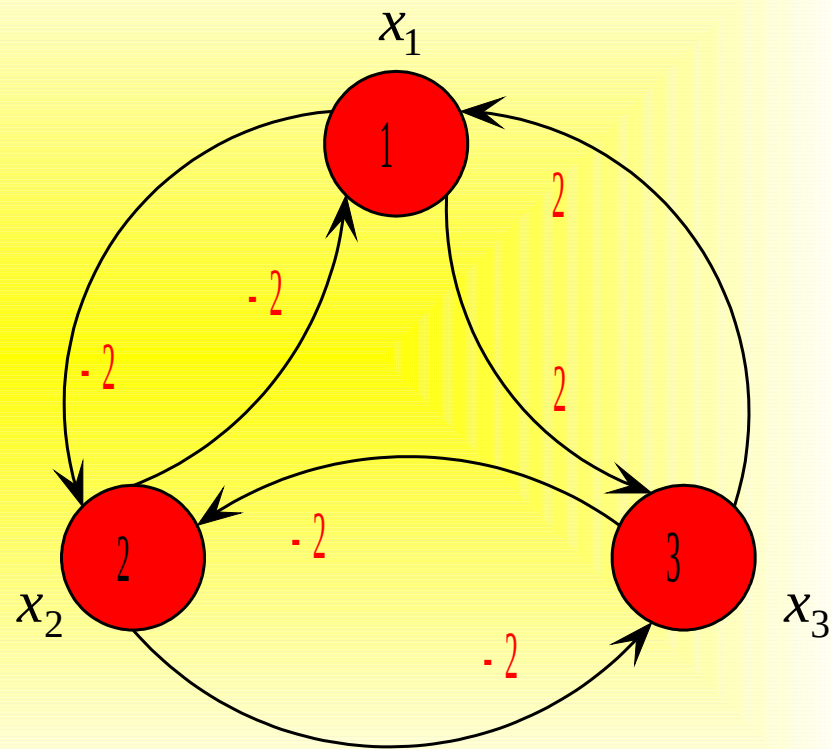
	1	0
外形	圆形	椭圆形
质地	光滑	粗糙
重量	< 1 磅	> 1 磅

标准桔子:  $x^{(1)} = [1, 0, 1]^T$

标准苹果:  $x^{(2)} = [0, 1, 0]^T$

## ■ (3) 测试

- 样本:  $x^{(1)} = [1, 0, 1]^T$  (桔子)       $x^{(2)} = [0, 1, 0]^T$  (苹果)
- 测试用  $[1, 1, 1]^T$   
例:
- 初始状态:  $\begin{cases} x_1(0) = 1 \\ x_2(0) = 1 \\ x_3(0) = 1 \end{cases}$
- 调整次序:  $2 \rightarrow 1 \rightarrow 3$   
序:



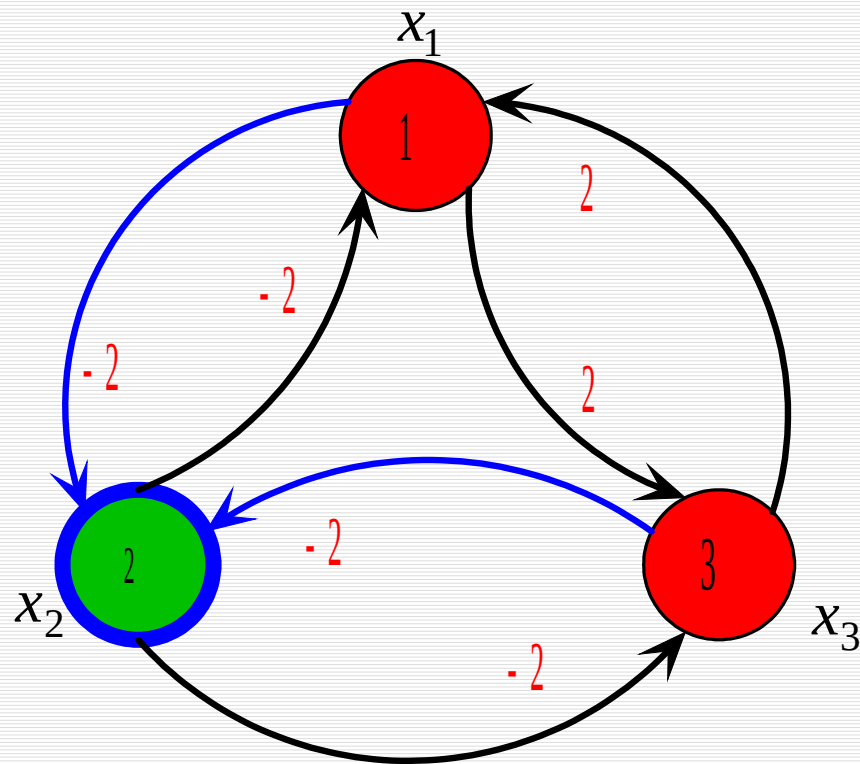


- 调整次序:

**2** → **1** → **3**

**$k = 0$**

$$\begin{cases} x_1(0) = 1 \\ x_2(0) = 1 \\ x_3(0) = 1 \end{cases}$$



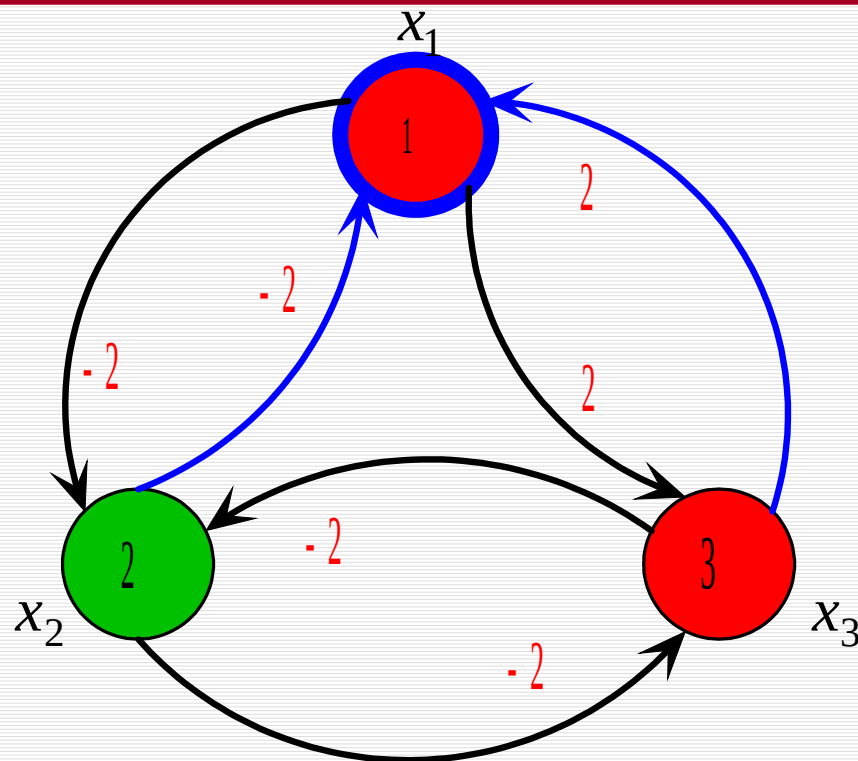
$$x_2(1) = f\left(\sum_{j=1}^3 w_{2j}x_j(0)\right) = f((-2) \times 1 + 0 \times 1 + (-2) \times 1) = f(-4) = 0$$

$$x_1(1) = x_1(0) = 1, \quad x_3(1) = x_3(0) = 1$$

- 2** → **1** → **3**

$$k = 1$$

$$\begin{cases} x_1(1) = 1 \\ x_2(1) = 0 \\ x_3(1) = 1 \end{cases}$$



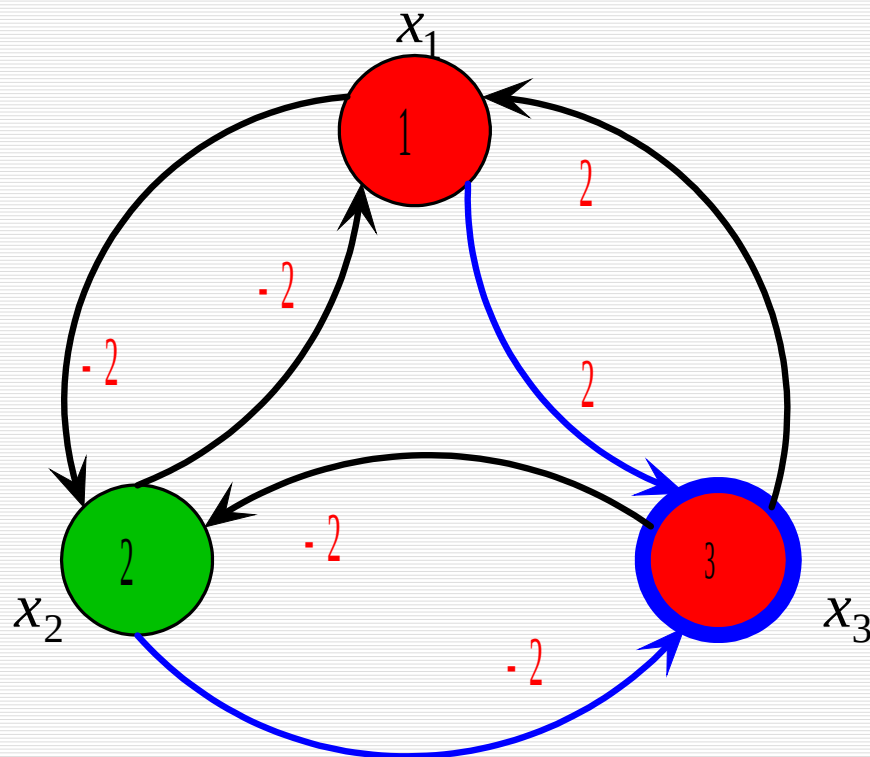
$$x_2(2) = x_2(1) = 0, \quad x_3(2) = x_3(1) = 1$$

- 调整次序:

$2 \rightarrow 1 \rightarrow 3$

$k = 2$

$$\begin{cases} x_1(2) = 1 \\ x_2(2) = 0 \\ x_3(2) = 1 \end{cases}$$

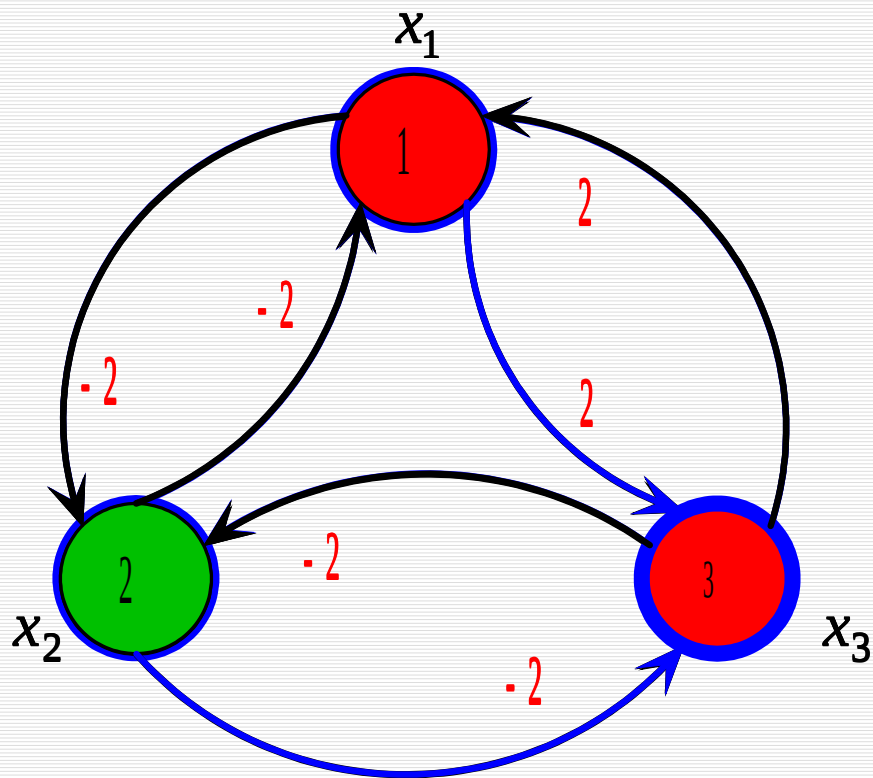


$x_3$

$$x_1(3) = x_1(2) = 1, \quad x_2(3) = x_2(2) = 0$$

- 调整次 序:  $2 \rightarrow 1 \rightarrow 3$
- 样本:  $x^{(1)} = [1, 0, 1]^T$  (桔子)

$x^{(2)} = [0, 1, 0]^T$  (苹果)



$k = 0$

$$\begin{cases} x_1(0) = 1 \\ x_2(0) = 1 \\ x_3(0) = 1 \end{cases}$$

$k = 1$

$$\begin{cases} x_1(1) = 1 \\ x_2(1) = 0 \\ x_3(1) = 1 \end{cases}$$

$k = 2$

$$\begin{cases} x_1(2) = 1 \\ x_2(2) = 0 \\ x_3(2) = 1 \end{cases}$$

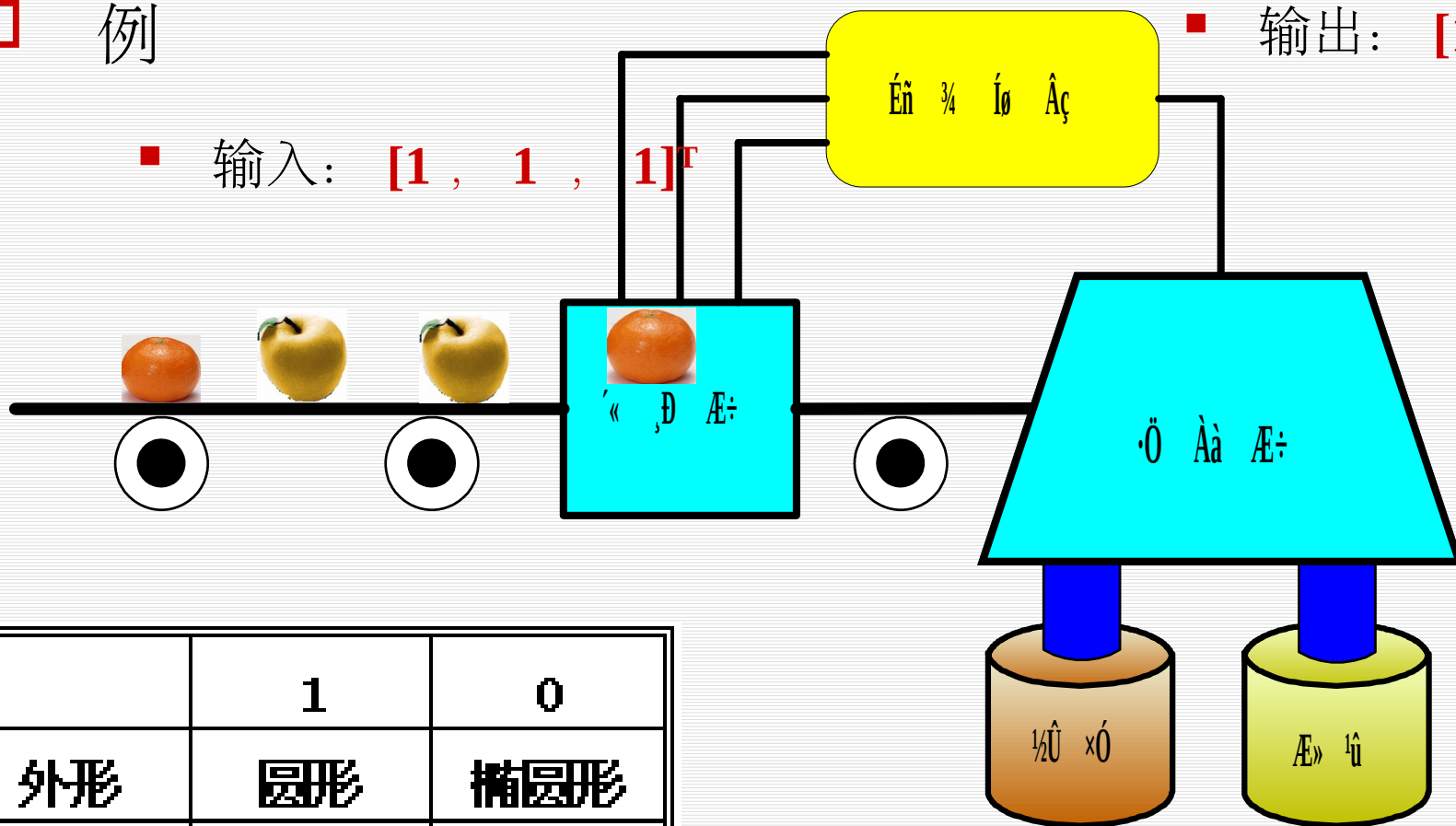
$k = 3$

$$\begin{cases} x_1(3) = 1 \\ x_2(3) = 0 \\ x_3(3) = 1 \end{cases}$$

□ 例

■ 输入:  $[1, 1, 1]^T$

■ 输出:  $[1, 0, 0]^T$



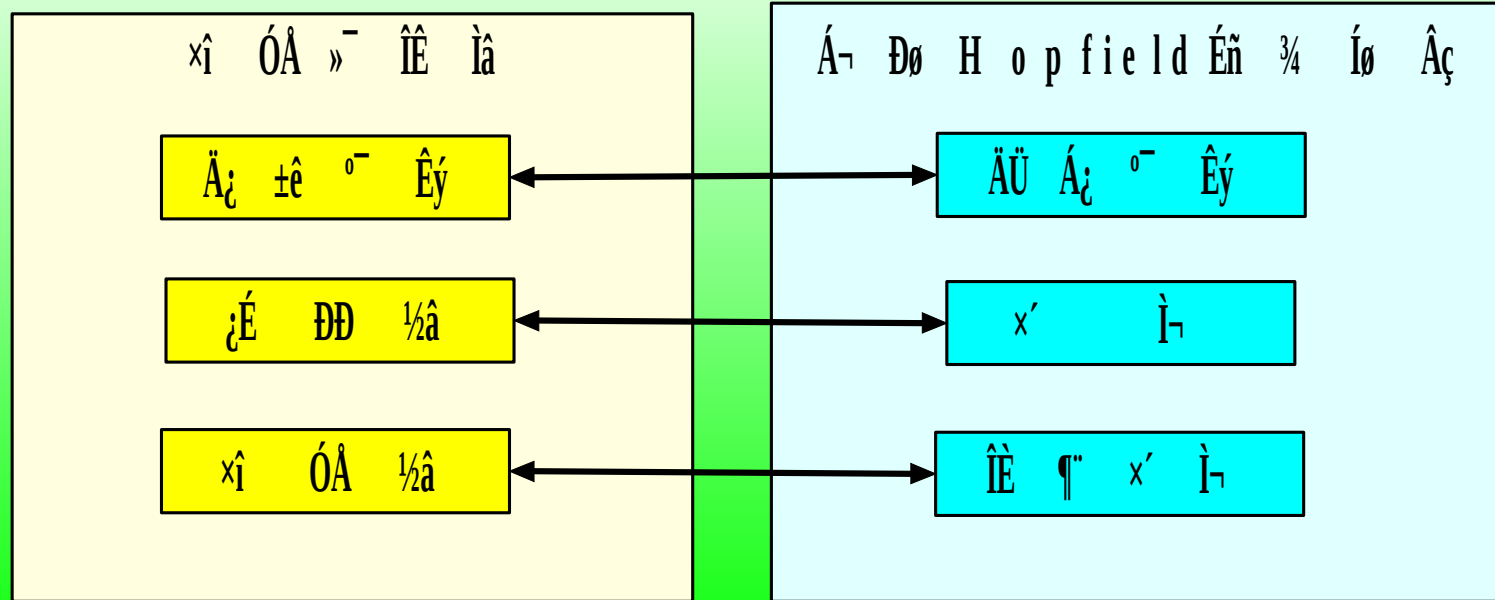
	1	0
外形	圆形	椭圆形
质地	光滑	粗糙
重量	< 1 磅	> 1 磅

标准桔子:  $x^{(1)} = [1, 0, 1]^T$

标准苹果:  $x^{(2)} = [0, 1, 0]^T$

## 8.5.2 Hopfield 神经网络优化方法

- 1985 年，霍普菲尔德和塔克（**D. W. Tank**）应用连续 **Hopfield** 神经网络求解旅行商问题（**traveling salesman problem**，**TSP**）获得成功。
- 连续 **Hopfield** 神经网络求解约束优化问题的基本思路：



## 8.5.2 Hopfield 神经网络优化方法

□ 用神经网络方法求解优化问题的一般步骤:

- (1) 将优化问题的每一个可行解用换位矩阵表示。
- (2) 将换位矩阵与由  $n$  个神经元构成的神经网络相对应: 每一个可行解的换位矩阵的各元素与相应的神经元稳态输出相对应。
- (3) 构造能量函数, 使其最小值对应于优化问题的最优解, 并满足约束条件。
- (4) 用罚函数法构造目标函数, 与 **Hopfield** 神经网络的计算能量函数表达式相等, 确定各连接权和偏置参数。
- (5) 给定网络初始状态和网络参数等, 使网络按动态方程运行, 直到稳定状态, 并将它解释为优化问题的解。

## 8.5.2 Hopfield 神经网络优化方法

□ 应用举例: **Hopfield** 神经网络优化方法求解 **TS**

**D**。

- 1985 年, 霍普菲尔德和塔克 ( **D. W. Tank** ) 应用连续 **Hopfield** 神经网络求解旅行商问题获得成功。

- 旅行商问题 ( **traveling salesman problem** , **TSP** ) : 有  $n$  个城市, 城市间的距离或旅行成本已知, 求合理的路线使每个城市都访问一次, 且总路径 (或者总成本) 为最短。



## 8.5.2 Hopfield 神经网络优化方法

### □ 应用举例：Hopfield 神经网络优化方法求解 TSP

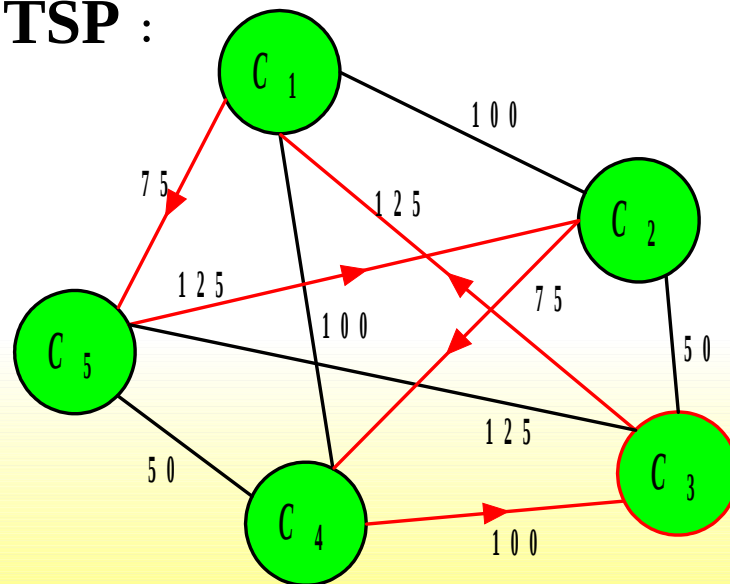
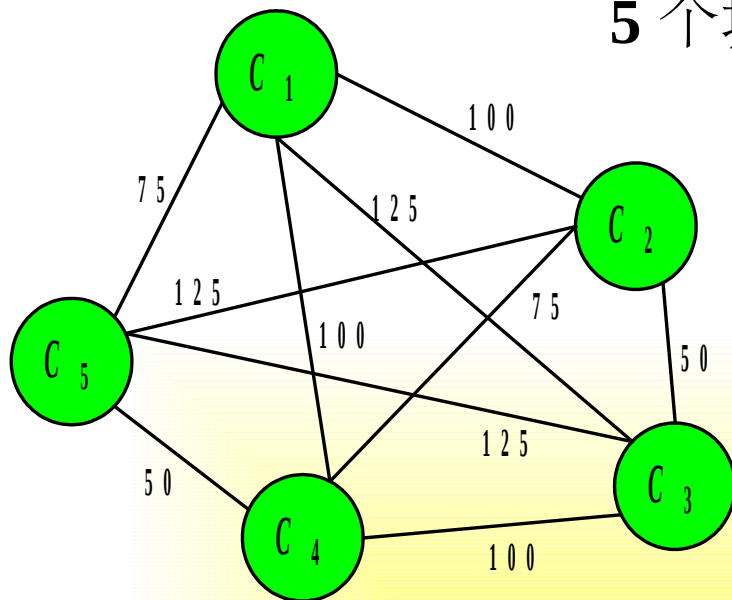
- 旅行商问题（**TSP**）：典型的组合优化问题
- 用穷举法，Cray 计算机的计算速度： $10^8$  次 / 秒。

城市数 ( $n$ )	7	15	20	50	100	200
搜索时间	$2.5 \times 10^{-5}$ 秒	1.8 小时	350 年	$5 \times 10^{48}$ 年	$10^{142}$ 年	$10^{358}$ 年

- 1985 年，Hopfield 和 Tank 用 Hopfield 网络求解  $n = 30$  的 TSP 问题，0.2 s 就得到次优解。

## 8.5.2 Hopfield 神经网络优化方法

5 个城市的 TSP :



	1	2	3	4	5
$C_1$	0	1	0	0	0
$C_2$	0	0	0	1	0
$C_3$	1	0	0	0	0
$C_4$	0	0	0	0	1
$C_5$	0	0	1	0	0

神经元数目:  
**25**

## 8.5.2 Hopfield 神经网络优化方法

### ■ TSP 的描述:

$$\min l = \frac{1}{2} \sum_x \sum_{y \neq x} \sum_i d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1})$$

$$\text{st.} \quad \sum_x v_{xi} = 1 \quad \forall i \quad (\text{每次只能访问一个城市})$$

$$\sum_i v_{xi} = 1 \quad \forall x \quad (\text{每个城市只能访问一次})$$

### ■ 用罚函数法，写出优化问题的目标函数:

$$J = \frac{A}{2} \sum_x \sum_i \sum_{j \neq i} v_{xi} v_{xj} + \frac{B}{2} \sum_i \sum_x \sum_{y \neq x} v_{xj} v_{yi} + \frac{C}{2} \left( \sum_x \sum_i v_{xi} - n \right)^2 + \frac{D}{2} \sum_x \sum_{y \neq x} \sum_i d_{xy} v_{xi} (v_{y,i+1} + v_{y,i-1})$$

## 8.5.2 Hopfield 神经网络优化方法

- **Hopfield** 神经网络能量函数:

$$\begin{aligned} E &= -\frac{1}{2} \sum_{x=1}^n \sum_{i=1}^n \sum_{y=1}^n \sum_{j=1}^n w_{xi,yj} v_{xi} v_{yj} - \sum_{x=1}^n \sum_{i=1}^n v_{xi} I_{xi} + \sum_{x=1}^n \sum_{i=1}^n \frac{1}{R'_{xi}} \int_0^{v_{xi}} f^{-1}(v) dv \\ &= E_1 + \sum_{x=1}^n \sum_{i=1}^n \frac{1}{R'_{xi}} \int_0^{v_{xi}} f^{-1}(v) dv \end{aligned}$$

- 令  $E_1$  与目标函数  $J$  相等, 确定神经网络的连接权值和偏置电流:  
$$w_{xi,yj} = A\delta_{xy}(1 - \delta_{ij}) - B\delta_{ij}(1 - \delta_{xy}) - C - D\delta_{xy}(\delta_{j,i+1} + \delta_{j,i-1})$$

$$I_{xi} = -Cn$$

## 8.5.2 Hopfield 神经网络优化方法

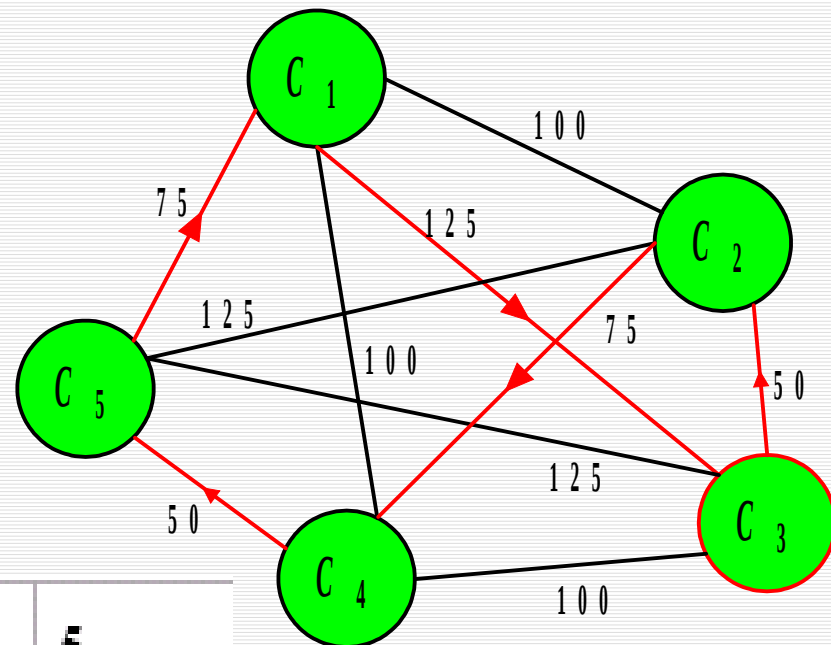
■ 神经网络的动态方程:

$$\begin{aligned}
 C_{xi} \frac{du_{xj}}{dt} &= - \frac{\partial E}{\partial v_{xi}} = - \frac{\partial (E_1 + \sum_{x=1}^n \sum_{i=1}^n \frac{1}{R'_{xi}} \int_0^{v_{xi}} f^{-1}(v) dv)}{\partial v_{xi}} \\
 &= - \frac{\partial (J + \sum_{x=1}^n \sum_{i=1}^n \frac{1}{R'_{xi}} \int_0^{v_{xi}} f^{-1}(v) dv)}{\partial v_{xi}} \\
 &= - \frac{u_{xi}}{R'_{xi}} - A \sum_{j \neq i} v_{xj} - B \sum_{y \neq x} v_{yi} - C (\sum_x \sum_i v_{xi} - n) - D \sum_y d_{xy} (v_{y,i+1} + v_{y,i-1}) \\
 v_{xi} &= f(u_{xi}) = \frac{1}{2} [1 + \tanh(\frac{u_{xi}}{u_0})]
 \end{aligned}$$

## 8.5.2 Hopfield 神经网络优化方法

- 选择合适的  $A$ 、 $B$ 、 $C$ 、 $D$  和网络的初始状态，按网络动态方程演化直到收敛。

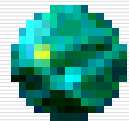
	1	2	3	4	5
$C_1$	0	0	0	0	1
$C_2$	0	1	0	0	0
$C_3$	1	0	0	0	0
$C_4$	0	0	1	0	0
$C_5$	0	0	0	1	0



## 8.5.2 Hopfield 神经网络优化方法

- 神经网络优化计算目前存在的问题：
  - （1）解的不稳定性。
  - （2）参数难以确定。
  - （3）能量函数存在大量局部极小值，难以保证最优解。

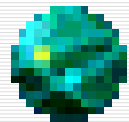
## 8.6 Hopfield 神经网络优化方法求解 JSP



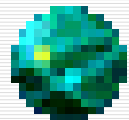
### 8.6.1 作业车间调度问题



### 8.6.2 JSP 的 Hopfield 神经网络及其求解



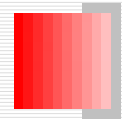
### 8.6.3 作业车间生产调度举例



### 8.6.4 基于随机神经网络的生产调度方法



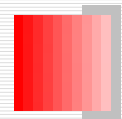
## 8.6.1 作业车间调度问题



作业车间调度问题（**job-shop scheduling**

**Problem** ,

**JSP** ) : 一类满足任务配置和顺序约束要求的资源分配问题。



问题描述：给定一个作业（工件）的集合和一个机器的集合，每个作业包括多道工序，每道工序需要在一台给定的机器上非间断地加工一段时间；每台机器一次最多只能加工一道工序，调度就是把工序分配给机器上某个时间段，使加工完成时间最短。

## 8.6.1 作业车间调度问题

- **Conway** 等（**1967**），《生产调度理论》：“一般作业车间调度问题是一个迷人的挑战性问题。尽管问题本身描述非常容易，但是朝着问题求解的方向作任何的推进都是极端困难的”。
- 对于单台机器加工问题，如果有  $n$  个作业而每个作业只考虑加工时间以及与操作序列有关的安装时间，则这个问题就和  $n$  个城市的 **TSP** 等价。
- **Foo S. Y.** 和 **Y. Takefuji** 在 **1988** 年最早提出用 **Hopfield** 神经网络求解 **JSP**。

## 8.6.2 JSP 的 Hopfield 神经网络及其求解

### 1. JSP 的换位矩阵表示

“工序  $(i,j,k)$  不依赖于任何别的工序”的命题。

(1,2,2)：作业 1 的工序  
2 在机器 2 上执行。

2—作业 2—机器 JSP						
		0	1,1,1	1,2,2	2,2,1	2,1,2
1,1,1		1	0	0	0	0
1,2,2		0	0	0	0	1
2,2,1		0	0	1	0	0
2,1,2		1	0	0	0	0

“工序 (2,2,1) 依赖于另一工序 (1,2,2)”的命题成立。

## 8.6.2 JSP 的 Hopfield 神经网络及其求解

$n$  作业  $m$  机器 JSP 的工序约束条件:

- (1) 各工序应服从优先顺序关系。任一工序可以依赖于另一个工序,也可以不依赖于任何工序(如在 0 时刻启动的工序)。
- (2) 所有工序不允许自依赖和互依赖。
- (3) 允许在 0 时刻启动的工序数不超过  $n$ 。即在 0 时刻启动的工序数应为  $n$ 。
- (4) 在同一时刻启动的同一作业的工序不多于一个。
- (5) 在同一时刻同一机器上启动的工序不多于一个。

# 8.6.2 JSP 的 Hopfield 神经网络及其求解

## 2. JSP 计算能量函数

$$\begin{aligned}
 E = & \frac{A}{2} \sum_{x=1}^{mn} \sum_{i=1}^{mn+1} \sum_{\substack{j=1 \\ j \neq i}}^{mn+1} v_{xi} v_{xj} + \frac{B}{2} \left( \sum_{x=1}^{mn} \sum_{i=1}^{mn+1} v_{xi} - mn \right)^2 + \frac{C}{2} \sum_{i \geq 2}^{mn+1} \sum_{x=1}^{mn} v_{xi} v_{i-1, x+1} \\
 & + \frac{D_1}{2} \left( \sum_{x=1}^{mn} v_{x1} \right)^2 \quad \text{行约束} \quad \sum_{k_1=1}^n \sum_{\substack{k_2=1 \\ k_3 \neq k_2}}^m v_{k_1+(k_2-1)m} v_{k_1+(k_3-1)m} \quad \text{全局约束} \quad \text{非对称约束} \\
 & + \frac{D_3}{2} \sum_{k_1=1}^m \sum_{k_2=1}^n \sum_{\substack{k_3=1 \\ k_3 \neq k_2}}^n v_{k_1+(k_2-1)m} v_{k_1+(k_3-1)m} \quad \text{列约束}
 \end{aligned}$$

$v_{xi}$  : 与矩阵中  $(x, i)$  位置相对应的神经元的输出状态。

## 8.6.2 JSP 的 Hopfield 神经网络及其求解

### 3. Hopfield 神经网络的参数

- 连续型 Hopfield 神经网络的计算能量函数：

$$E = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N w_{ij} v_i v_j - \sum_{i=1}^N v_i I_i + \sum_{i=1}^N \frac{1}{R_i} \int_0^{v_i} f^{-1}(v) dv$$

- 神经元  $(x, i)$  与神经元  $(y, j)$  之间的连接权

$$w_{xi,yj} = -A\delta_{xy}(1-\delta_{ij}) - B - C\delta_{y(i-1)}\delta_{j(x+1)}(1-\delta_{i1})(1-\delta_{xy})(1-\delta_{ij})$$

$$- D_1\delta_{i1}\delta_{j1} - \sum_{k_1=1}^n \sum_{k_2=1}^m \sum_{\substack{k_3=1 \\ k_3 \neq k_2}}^m D_2\delta_{x[k_2+(k_1-1)m]}\delta_{y[k_3+(k_1-1)m]}$$

$$- \sum_{k_1=1}^m \sum_{k_2=1}^n \sum_{\substack{k_3=1 \\ k_3 \neq k_2}}^n D_3\delta_{x[k_1+(k_2-1)m]}\delta_{y[k_3+(k_2-1)m]}$$

$$\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

- 神经元  $(x, i)$  的偏置电流  $I_{xi} = Bmn + D_1m\delta_{i1}$

## 8.6.2 JSP 的 Hopfield 神经网络及其求解

### 4. Hopfield 神经网络的运动方程

$$\begin{aligned} C_{xi} \frac{du_{xi}}{dt} = & - \frac{u_{xi}}{r_{xi}} - A \sum_{\substack{j=1 \\ j \neq i}}^{mn+1} v_{xj} - B \sum_{y=1}^{mn} \sum_{j=1}^{mn+1} v_{yj} \\ & - C v_{(i-1)(x+1)} (1 - \delta_{i1}) (1 - \delta_{x(i-1)}) (1 - \delta_{i(x+1)}) \\ & - D_1 \sum_{y=1}^{mn} v_{y1} \delta_{i1} - D_2 \sum_{k_1=1}^n \sum_{k_2=1}^m \sum_{\substack{k_3=1 \\ k_3 \neq k_2}}^m v_{[k_3+(k_1-1)m]i} \delta_{x[k_2+(k_1-1)m]} \\ & - D_3 \sum_{k_1=1}^m \sum_{k_2=1}^n \sum_{\substack{k_3=1 \\ k_3 \neq k_2}}^n v_{[k_1+(k_3-1)]i} \delta_{x[k_1+(k_2-1)m]} + I_{xi} \end{aligned}$$

## 8.6.2 JSP 的 Hopfield 神经网络及其求解

### 5. 成本树

**step1** : 根据换位矩阵, 构造成本树。

**step2** : 计算成本树上各操作  $(i, j, k)$  的开始时间  $t_{ijk}$  和结束时间  $E_{ijk}$ 。

**step3** : 判断是否出现死锁调度。

**step4** : 调整死锁调度。



## 8.6.2 JSP 的 Hopfield 神经网络及其求解

### 6. 甘特图

**step1** : 根据换位矩阵, 计算成本树上各操作的开始时间和结束时间, 并给出相应的甘特图。

**step2** : 判断甘特图中每台机器上各作业的开始时间是否发生重叠。

**step 3** : 判断同一作业的各操作的开始时间是否发生重叠。

**step4** : 重复 **step2** 和 **step3** , 直至甘特图中同一机器上各作业的开始时间和同一作业的各操作的开始时间都不发生重叠为止。

## 8.6.3 作业车间生产调度举例

### □ 2 作业 3 机器的 JSP 例子

机器分配

作业	工序		
	1	2	3
1	1	2	3
2	3	1	2

加工时间分配

作业	工序		
	1	2	3
1	5	8	2
2	7	3	9

所有的操作:

111 , 122 , 133  
, 213 , 221 , 2  
32 。

## 8.6.3 作业车间生产调度举例

### □ 换位矩阵

	0	111	122	133	221	232	213
111		0	0	0			
122	0		0	0			
133	0			0			
221	0				0	0	
232	0					0	
213					0	0	0

**Hopfield** 神经网络: 6 行 7 列的神经元阵列

## 8.6.3 作业车间生产调度举例

### □ 神经网络偏置电流矩阵

	0	111	122	133	221	232	213
111	-1600	0.1	0.1	0.1	-600	-600	-600
122	0.1	-600	0.1	0.1	-600	-600	-600
133	0.1	-600	-600	0.1	-600	-600	-600
221	0.1	-600	-600	-600	0.1	0.1	-600
232	0.1	-600	-600	-600	-600	0.1	-600
213	-1600	-600	-600	-600	0.1	0.1	0.1

$$A = 500, B = 100, C = 200, D_1 = 500, D_2 = 300, D_3 = 300$$

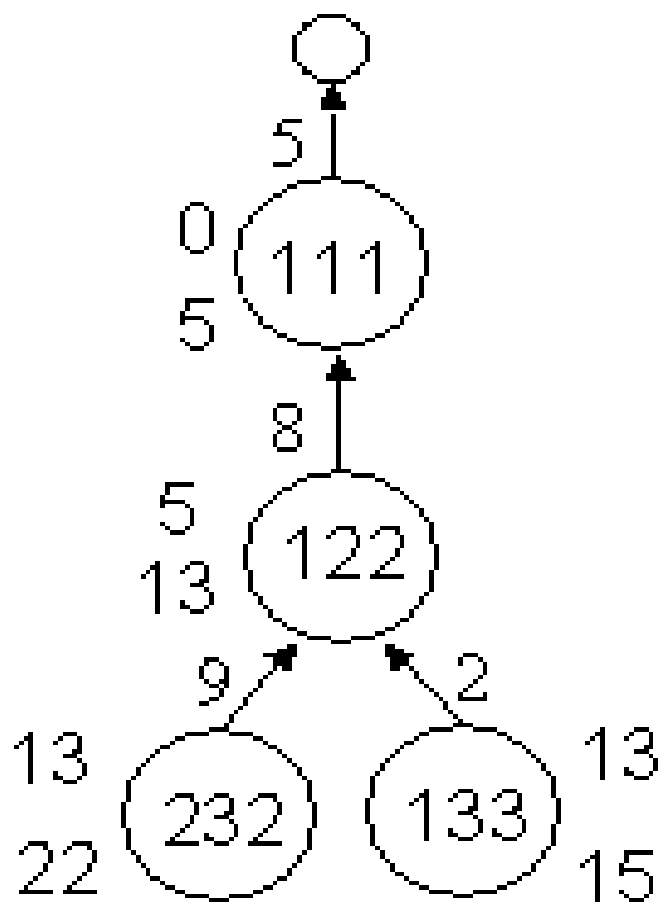
## 8.6.3 作业车间生产调度举例

□ 计算能量函数为 **0** 的换位矩阵

	<b>0</b>	<b>111</b>	<b>122</b>	<b>133</b>	<b>221</b>	<b>232</b>	<b>213</b>
<b>111</b>	1	0	0	0	0	0	0
<b>122</b>	0	1	0	0	0	0	0
<b>133</b>	0	0	1	0	0	0	0
<b>221</b>	0	0	0	0	0	0	1
<b>232</b>	0	0	1	0	0	0	0
<b>213</b>	1	0	0	0	0	0	0

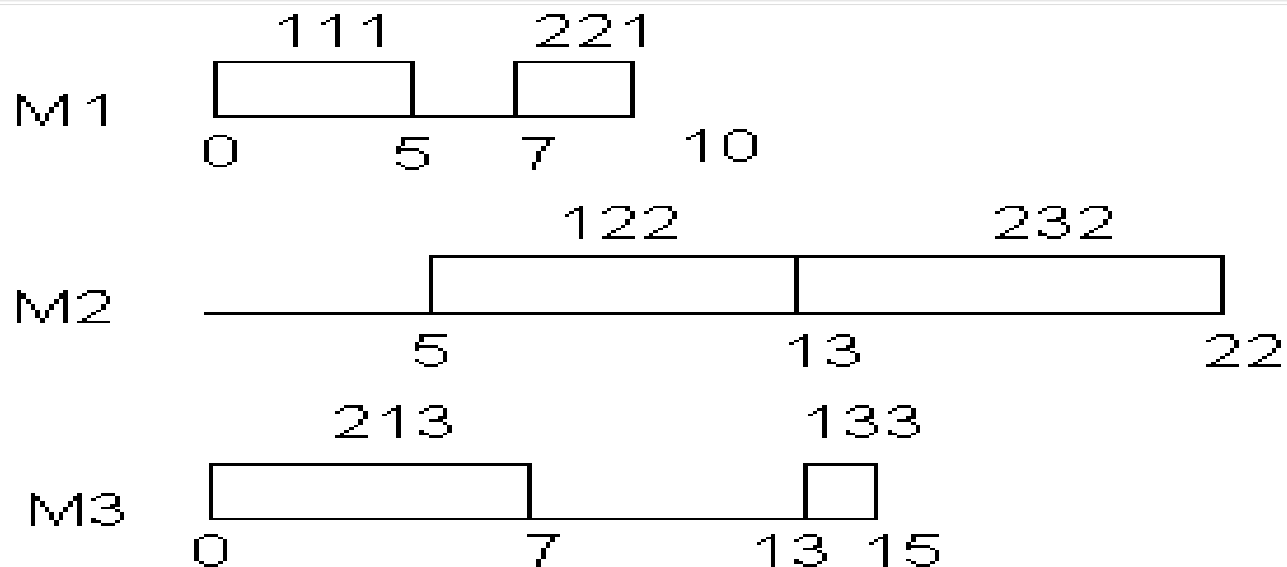
## 8.6.3 作业车间生产调度举例

### □ 成本树



## 8.6.3 作业车间生产调度举例

### □ 甘特图



## 8.6.4 基于随机神经网络的生产调度方法

### □ 基本思想:

在系统寻优过程中，利用神经元状态更新的随机性，允许向较差方向搜索，以跳出局部极小。经多次寻查后，最终使系统稳定于能量最低状态，使神经网络收敛到计算能量函数的最小值 **0**，从而使神经网络输出是一个可行调度解。



## 8.6.4 基于随机神经网络的生产调度方法

□ 根据改进 **Metropolis** 方法，求解 **JSP** 的基于模拟退火的神经网络算法：

（1）初始化：

设置初始温度  $\leftarrow T_{\max}$ ，合适的输入偏置电流，凝结温度，温度下降速率，在每个温度点的循环处理次数。

（2）随机爬山：

对每个神经元，由求解网络方程计算输出电压。由网络稳定状态集组成成本树；求出最大成本变化量  $\text{cost}_i$ 。

## 8.6.4 基于随机神经网络的生产调度方法

- 若  $\Delta \text{cost}_i < 0$  , 则转去 ( 3 ) ; 否则计算能量变化量

$$\Delta E_i = \sum_j w_{ij} v_j + \theta_i$$

- 若  $\Delta E_i < 0$  , 则令  $\Delta E_i \leftarrow \Delta E_i + \Delta \text{cost}_i$

否则, 令  $\Delta E_i \leftarrow \Delta E_i - \Delta \text{cost}_i$

- 计算概率

$$p_i = \frac{1}{1 + \exp(-\frac{\Delta E_i}{T})}$$

# 第 8 章 人工神经网络及其应用

- 8.1 神经元与神经网络
- 8.2 **BP** 神经网络及其学习算法
- 8.3 **BP** 神经网络的应用
- 8.4 **Hopfield** 神经网络及其改进
- 8.5 **Hopfield** 神经网络的应用
- 8.6 **Hopfield** 神经网络优化方法求解 **JSP**
- 8.7 卷积神经网络及其应用

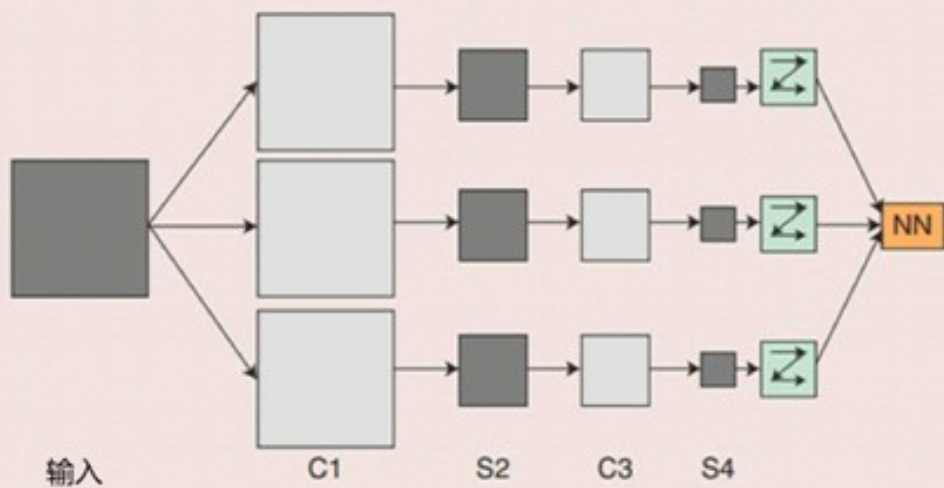
## 8.7 卷积神经网络

### □ 卷积神经网络（**Convolutional Neural Networks**，**CNN**）

- **1962** 年 **Hubel** 和 **Wiesel** 通过对猫视觉皮层细胞的研究，提出了感受野 (**receptive field**) 的概念。视觉皮层的神经元就是局部接受信息的，只受某些特定区域刺激响应，而不是对全局图像进行感知。
- **1984** 年日本学者 **Fukushima** 基于感受野概念提出神经认知机 (**neocognitron**)。
- **CNN** 可看作是神经认知机的推广形式。

# 8.7.1 卷积神经网络的结构

## 卷积神经网络的结构



- **CNN** 是一个多层的神经网络，每层由多个二维平面组成，而每个平面由多个独立神经元组成。
- **C** 层为特征提取层（卷积层）
- **S** 层是特征映射层（下采样层）。
- **CNN** 中的每一个 **C** 层都紧跟着一个 **S** 层。

**概念示范：**输入图像通过与 **m** 个可训练的滤波器和可加偏置进行卷积，在 **C1** 层产生 **m** 个特征映射图，然后特征映射图中每组的 **n** 个像素再求和，加权值，加偏置，通过 **Sigmoid** 函数得到 **m** 个 **S2** 层的特征映射图。这些映射图再经过滤波得到 **C3** 层。这个层级结构再和 **S2** 一样产生 **S4**。最终，这些像素值被光栅化，并连接成一个向量输入到传统神经网络，得到输出。

## 8.7.1 卷积神经网络的结构

□ 特征提取层（卷积层）—— C 层（**Convolution layer**）

1 <sub>x1</sub>	1 <sub>x0</sub>	1 <sub>x1</sub>	0	0
0 <sub>x0</sub>	1 <sub>x1</sub>	1 <sub>x0</sub>	1	0
0 <sub>x1</sub>	0 <sub>x0</sub>	1 <sub>x1</sub>	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

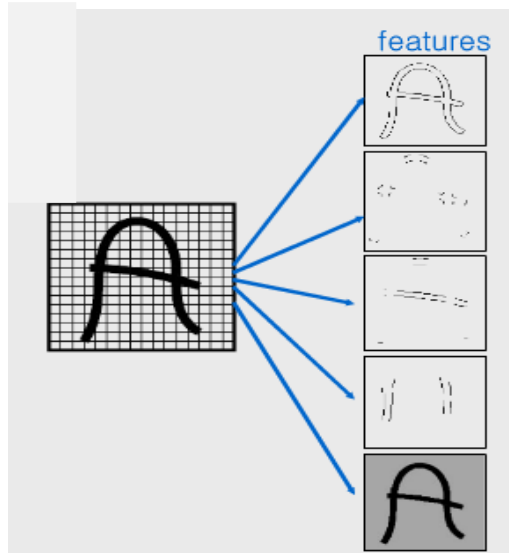
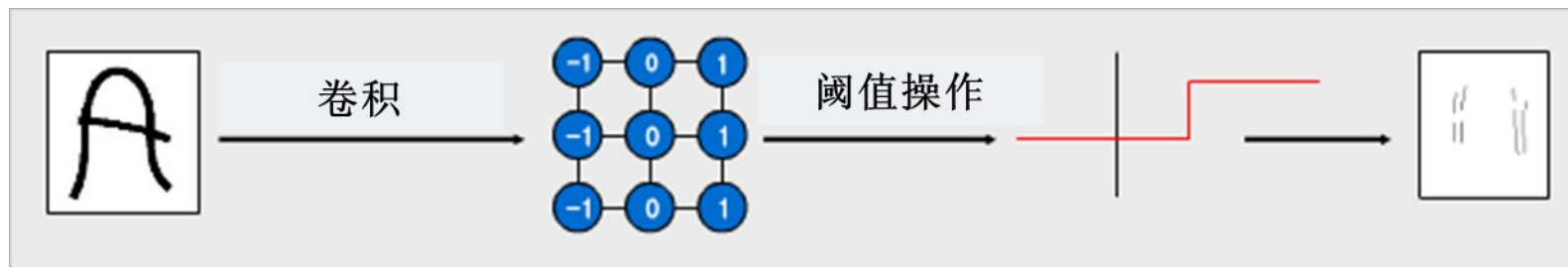
Convolved  
Feature

- 左图展示了一个 **3\*3** 的卷积核在 **5\*5** 的图像上做卷积的过程。
- 卷积实际上提供了一个权重模板。
- 卷积运算是一种用邻域点按一定权重去重新定义该点值的运算。
- 对图像用一个卷积核进行卷积运算，实际上是一个滤波的过程。每个卷积核都是一种特征提取方式，就像是一个筛子，将图像中符合条件的部分筛选出来。



## 8.7.1 卷积神经网络的结构

□ 特征提取层（卷积层）—— C 层（**C**onvolution layer）

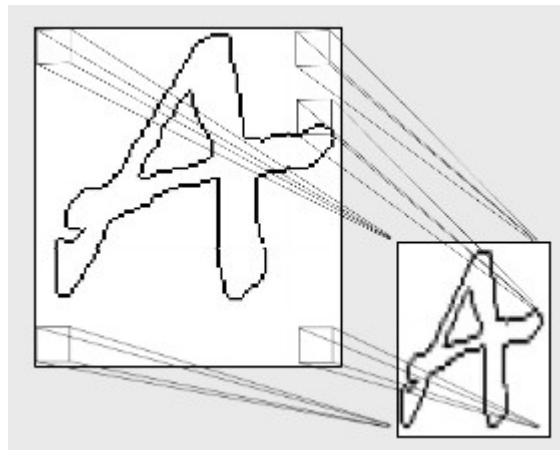
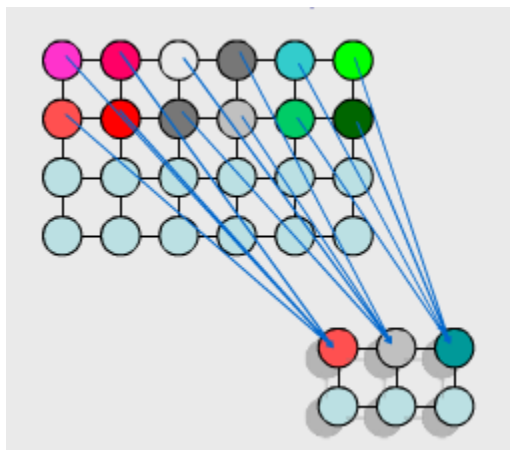


- 大部分的特征提取都依赖于卷积运算
- 利用卷积算子对图像进行滤波，可以得到显著的边缘特征。

## 8.7.1 卷积神经网络的结构

□ 特征映射层（下采样层）—— **S** 层（ **Subsampling layer** ）

卷积层的作用是探测上一层特征的局部连接，然而下采样层的作用是在语义上把相似的特征合并起来。

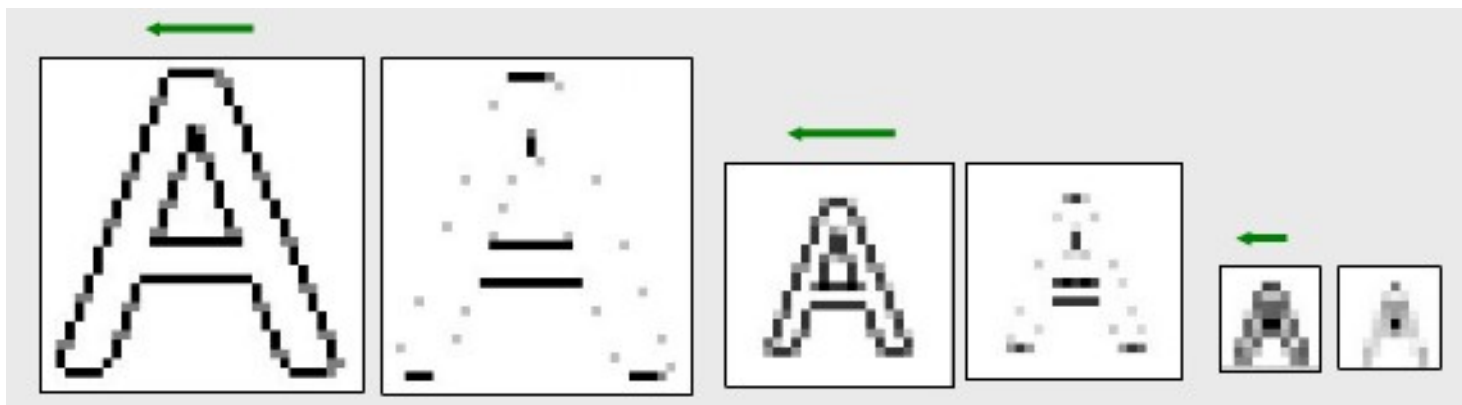




## 8.7.1 卷积神经网络的结构

□ 特征映射层（下采样层）—— **S** 层（ **S**ubsampling layer）

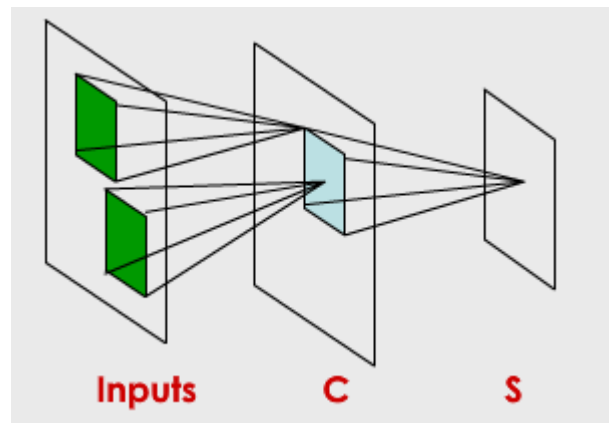
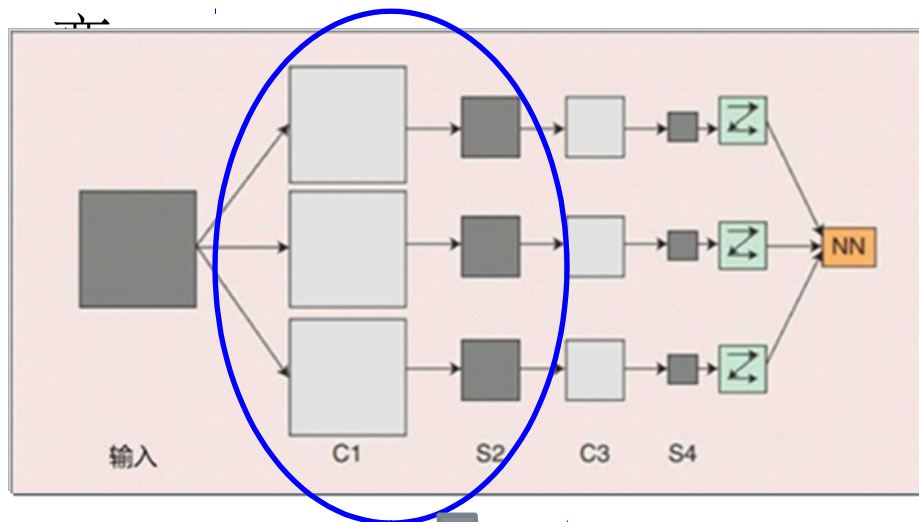
► 下采样层降低了每个特征图的空间分辨率。



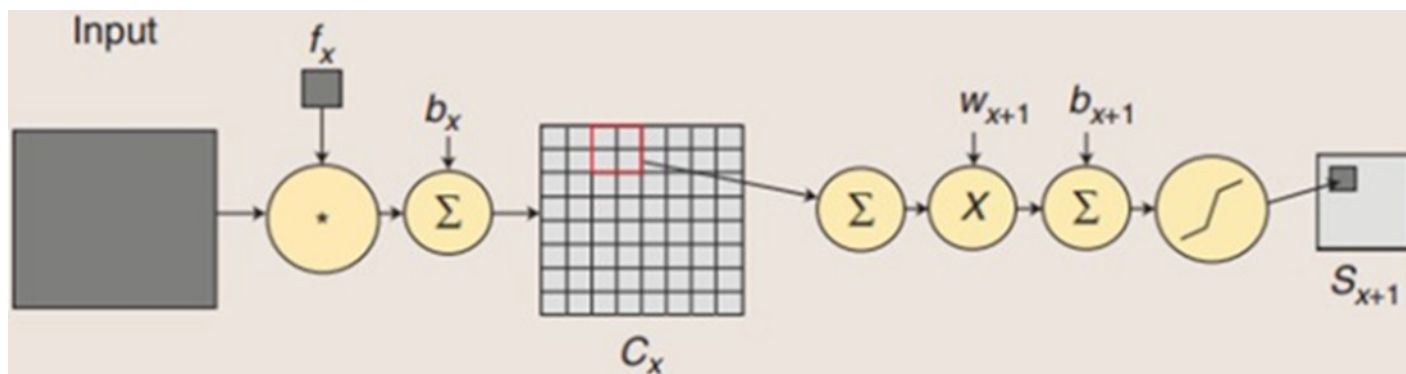
## 8.7.1 卷积神经网络的结构

□ 特征映射层（下采样层）—— **S** 层（ **Subsampling layer** ）

➤ **CNN** 中的每一个特征提取层（**C**）都紧跟着一个用来求局部平均与二次提取的计算层（**S**）。这种特有的两次特征提取结构能够容许识别过程中输入样本有较严重的畸

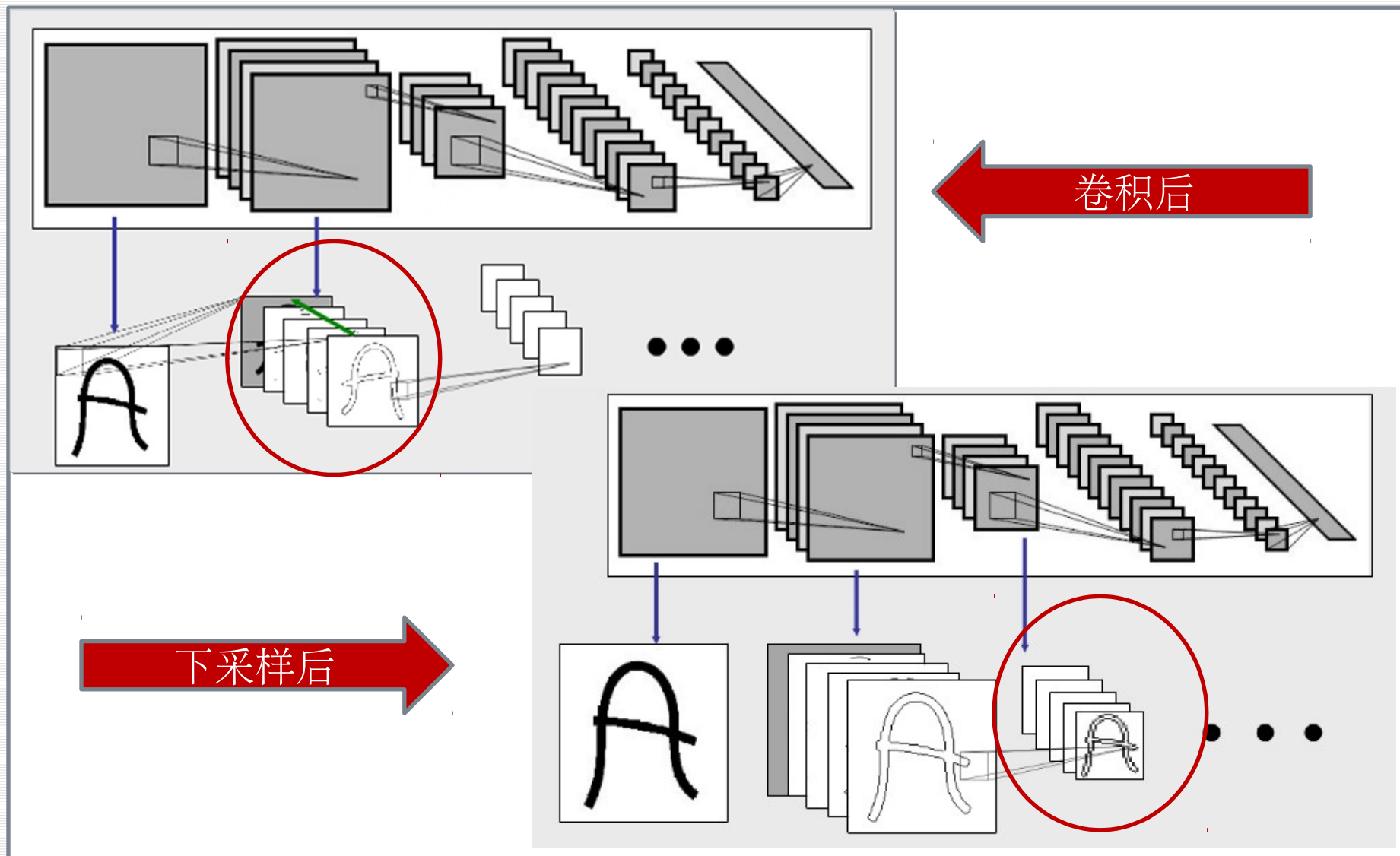


## 8.7.1 卷积神经网络的结构



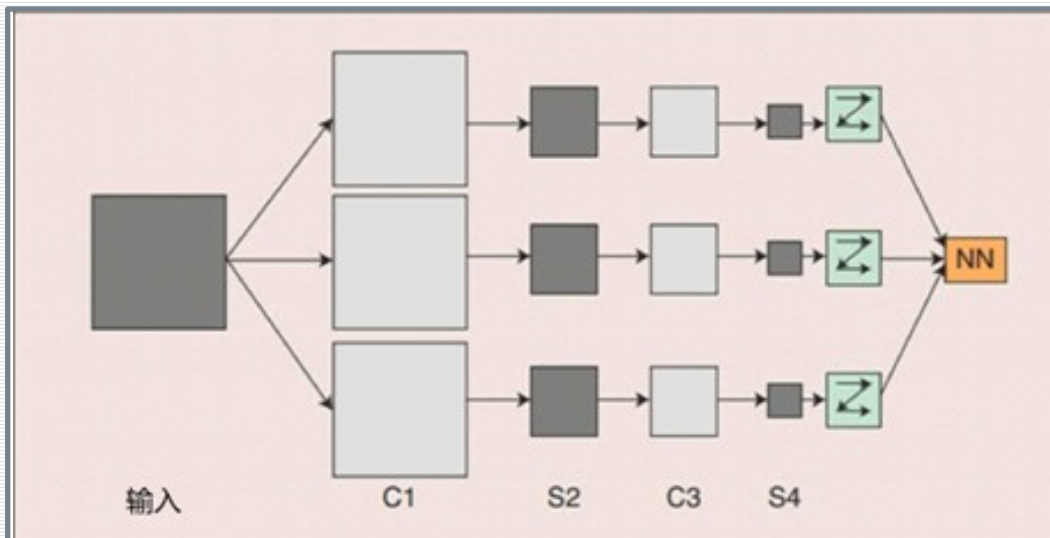
- **卷积过程**：用一个可训练的滤波器  $f_x$  去卷积一个输入的图像（第一阶段是输入的图像，后面的阶段就是 **Feature Map** 了），然后加一个偏置  $b_x$ ，得到卷积层  $C_x$ 。
- **下采样过程**：邻域  $n$  个像素通过池化（**pooling**）步骤变为一个像素，然后通过标量  $w_{x+1}$  加权，再增加偏置  $b_{x+1}$ ，然后通过一个 **sigmoid** 激活函数，产生一个大概缩小  $n$  倍的特征映射图  $S_{x+1}$ 。

## 8.7.1 卷积神经网络的结构



# 8.7.1 卷积神经网络的结构

## 卷积神经网络的结构



- **CNN** 是一个多层的神经网络，每层由多个二维平面组成，而每个平面由多个独立神经元组成。
- **C** 层为特征提取层（卷积层）
- **S** 层是特征映射层（下采样层）。
- **CNN** 中的每一个 **C** 层都紧跟着一个 **S** 层。

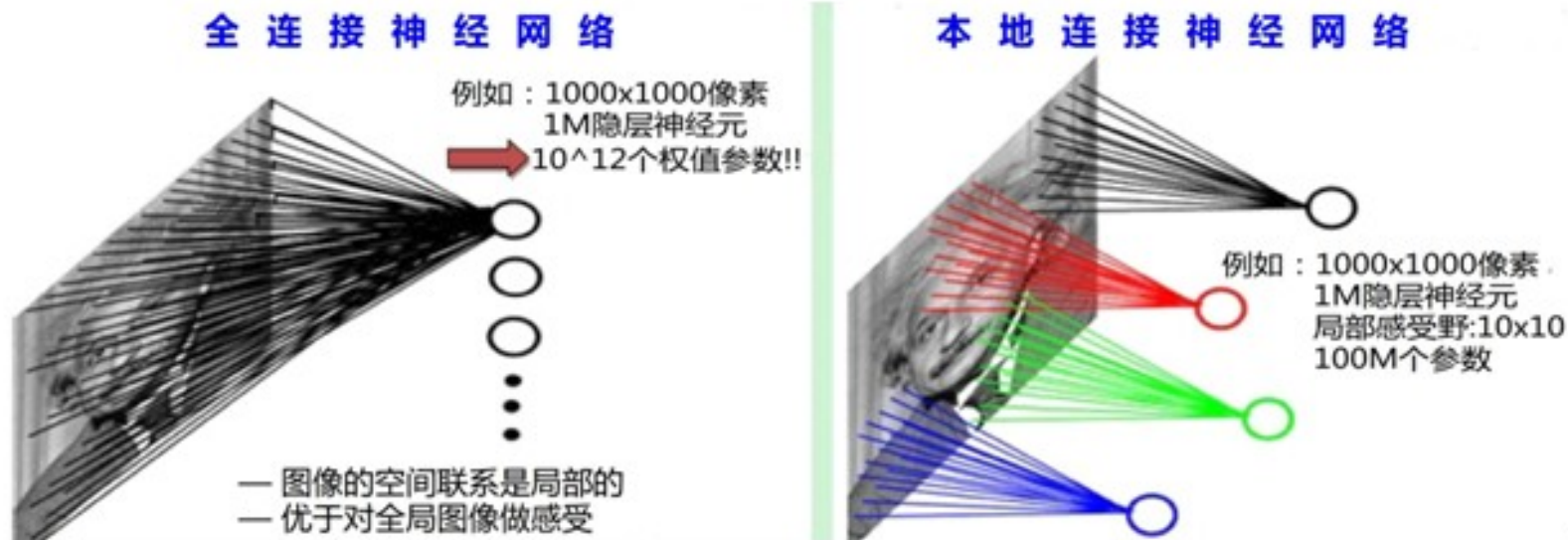
**概念示范：**输入图像通过与  $m$  个可训练的滤波器和可加偏置进行卷积，在 **C1** 层产生  $m$  个特征映射图，然后特征映射图中每组的  $n$  个像素再进行求和，加权重，加偏置，通过一个 **Sigmoid** 函数得到  $m$  个 **S2** 层的特征映射图。这些映射图再经过滤波得到 **C3** 层。这个层级结构再和 **S2** 一样产生 **S4**。最终，这些像素值被光栅化，并连接成一个向量输入到传统的神经网络，得到输出。

# 8.7.1 卷积神经网络的结构

□ 卷积神经网络的 4 个关键技术：

- 局部连接（**8.7.2**）
- 权值共享（**8.7.3**）
- 多卷积核（**8.7.4**）
- 池化（**8.7.5**）

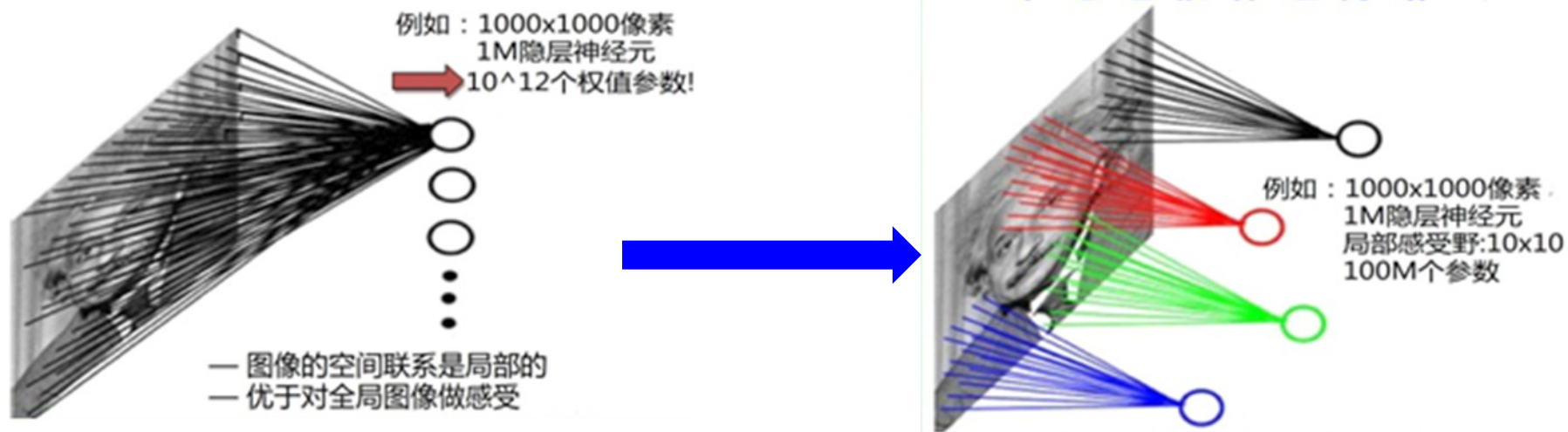
## 8.7.2 卷积神经网络的局部连接



- 视觉皮层中每个神经元不是对全局图像进行感知，而只对局部进行感知，然后在更高层将局部的信息综合起来得到全局信息。



## 8.7.3 卷积神经网络的权值共享



- 隐含层的每一个神经元如果只和 **10x10** 个像素连接，也就是说每一个神经元存在 **10x10=100** 个连接权值参数。如果将每个神经元的参数设置成相同，那么，不管隐层的神经元个数有多少，两层间的连接都只有 **100** 个参数，这就是卷积神经网络的**权值共享**。



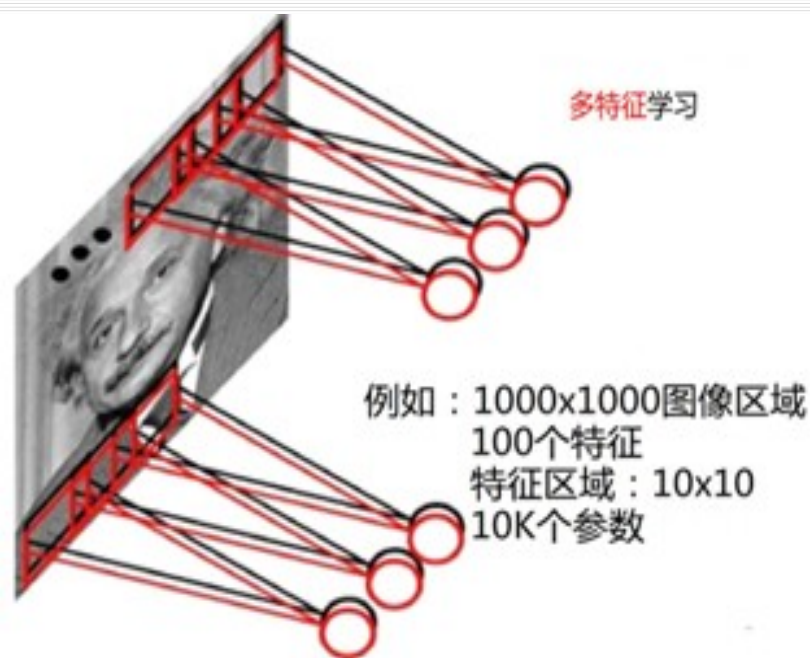
## 8.7.3 卷积神经网络的权值共享

□ 减少参数的方法:

- **局部连接**: 每个神经元无需对全局图像进行感知, 而只需对局部进行感知, 然后在更高层将局部的信息综合起来得到全局信息。
- **权值共享**: 每个神经元参数设为相同, 即权值共享, 也即每个神经元用同一个卷积核去卷积图像。

## 8.7.4 卷积神经网络的多卷积核

- 下图中不同的颜色表示不同的卷积核，每个卷积核都会将图像生成为另一幅特征映射图（即：一个卷积核提取一种特征）。
- 为了使特征提取更充分，我们可以添加多个卷积核（滤波器）以提取不同的特征。



✓ 每层隐层神经元的个数按卷积核的数量翻倍。

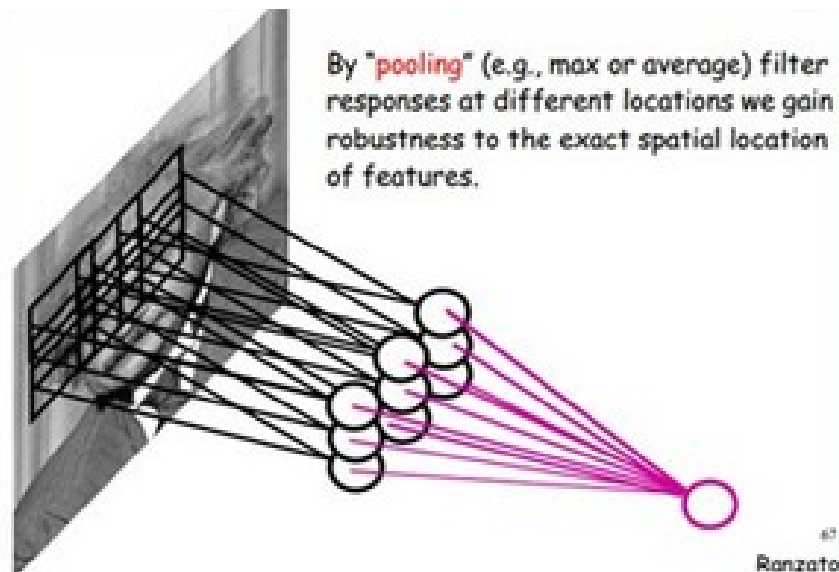
✓ 每层隐层参数个数仅与特征区域大小、卷积核的多少有关。

例如：隐含层的每个神经元都连接 **10x10** 像素图像区域，同时有 **100** 种卷积核（滤波器）。则参数总个数为：

$$(10 \times 10 + 1) \times 100 = 10100 \text{ 个}$$

## 8.7.5 卷积神经网络的池化

- 计算图像一个区域上的某个特定特征的平均值（或最大值），这种聚合操作就叫做池化 (pooling)，有时采用平均池化或者最大池化方法。



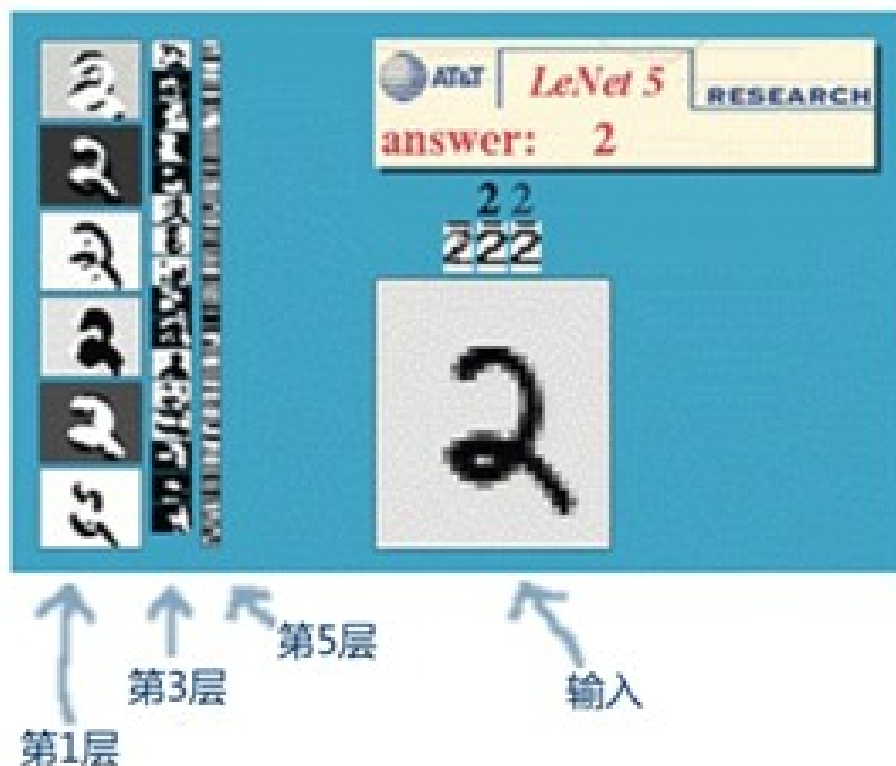
- 这些概要统计特征不仅具有低得多的维度（相比使用所有提取得到的特征），同时还会改善结果（不容易过拟

## 8.7.6 卷积神经网络的实现与应用

- 目前 **CNN** 架构有 **10-20** 层采用 **ReLU** 激活函数、上百万个权值以及几十亿个连接。硬件、软件以及算法并行的进步，使得训练时间大大压缩。
- **CNN** 容易在芯片或者现场可编程门阵列（**FPGA**）中实现，许多公司如 **NVIDIA**、**Mobileye**、**Intel**、**Qualcomm** 以及 **Samsung**，都在开发 **CNN** 芯片，以使智能机、相机、机器人以及自动驾驶汽车中的实时视觉系统成为可能。
- **20** 世纪 **90** 年代末，这个系统用于美国超过 **10%** 的支票阅读上。
- **21** 世纪开始，**CNN** 就被成功的大量用于检测、分割、物体识别以及图像的各个领域。近年来，卷积神经网络的一个重大成功应用是人脸识别。
- 图像可以在像素级进行打标签，可以应用在自动电话接听机器人、汽车自动驾驶等技术中。
- **CNN** 用于自然语言的理解以及语音识别中。

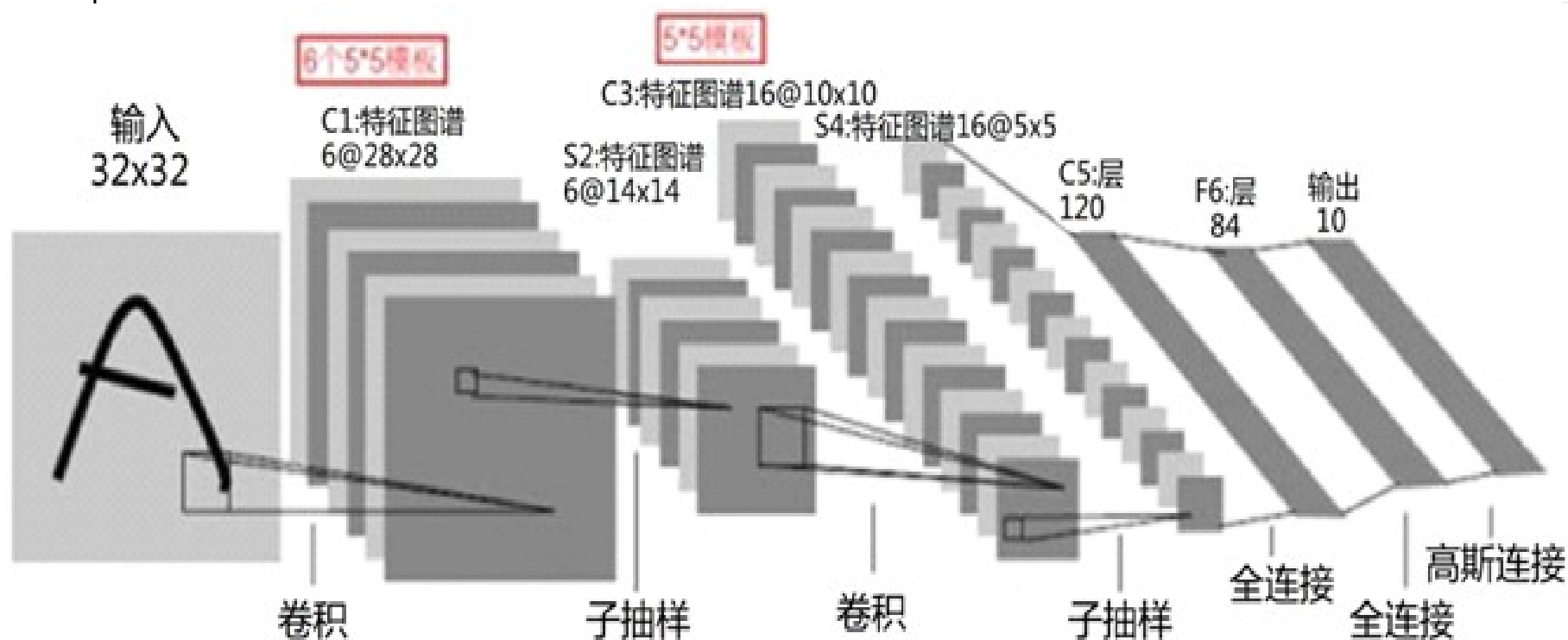
## 8.7.7 CNN 在手写数字识别中的应用

- 一种典型的用来识别数字的卷积网络是 [LeNet-5](#)。美国大多数银行当年用它识别支票上面的手写数字，达到了商用地步，说明该算法具有很高的准确性。



## 8.7.7 CNN 在手写数字识别中的应用

- **LeNet-5** 是一个数字手写系统，不包含输入层，共有 **7** 层，每层都包含可训练参数（连接权重）。输入图像为 **32\*32** 大



## 8.7 卷积神经网络

### □ CNN 的优点:

- 隐式地从训练数据中进行学习，避免了显式的特征抽取；
- 同一特征映射面上的神经元共享权值，网络可以并行学习，降低了网络学习的复杂性；
- 采用时间或者空间的下采样结构，可以获得某种程度的位移、尺度、形变的鲁棒性；
- 输入信息和网络拓扑结构能很好地吻合，在语音识别和图像处理方面有着独特优势。

## 8.7 卷积神经网络

### □ CNN 的缺点:

- **CNN** 的结构参数，无论是卷积层、下采样层还是分类层，都有太多的随意性或试凑性，且不能保证拓扑结构参数收敛。
- 重点放在由细尺度特征到大尺度特征的层层提取，只有前馈没有反馈。已有的认知不能帮助当前视觉感知和认知，没有体现选择性注意。
- 要求海量训练样本，样本的均等性没有反映认知的积累性。



**THE END**

**THE END**