

人工智能技术

Artificial Intelligence

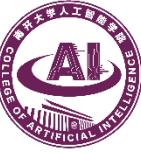
——人工智能:经典智能+智能计算+机器学习

AI: Classical Intelligence + Computing Intelligence + Machine Learning

王鸿鹏、杜月、王润花、许丽

南开大学人工智能学院





Section III

第三部分 计算智能 Computational Intelligence

——第八章：概述、人工神经网络
Chapter 8: Introduction & ANN



目录 Contents

- ① Introduction 概述
- ② Neural Computation 神经计算
 - ANN 人工神经网络
- ③ Fuzzy Computation 模糊计算
- ④ Evolutionary Algorithms, EAs 进化计算
 - Genetic Algorithm, GA 遗传算法
- ⑤ Artificial Life, Alife 人工生命
 - 人工脑、计算机病毒、计算机进程、细胞自动机、人工核苷酸
- ⑥ Swarm Intelligence 群智能
 - Particle swarm optimization, PSO 粒群优化算法
 - Ant colony optimization, ACO 蚁群优化算法

概述

什么是计算智能，它与传统的人工智能有何区别？

第一个对计算智能定义是由贝兹德克(J. C. Bezdek)于1992年提出的。

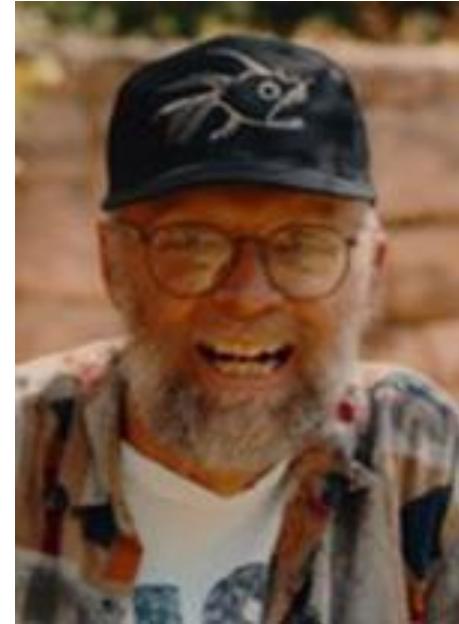
due to J.C. Bezdek who states that:

"...(strictly) computational systems depend on numerical data supplied by manufactured sensors and do not rely upon knowledge".

Bezdek对一些相关术语给予一定的符号和简要说明或定义。给出了有趣的ABC：

- Ⓐ A- Artificial/Symbolic,
- Ⓑ B- Biological/Organic,
- Ⓒ C- Computational/Numeric System,

表示人工的(非生物的)，即人造的
表示物理的+化学的+(?)=生物的
表示数学+计算机

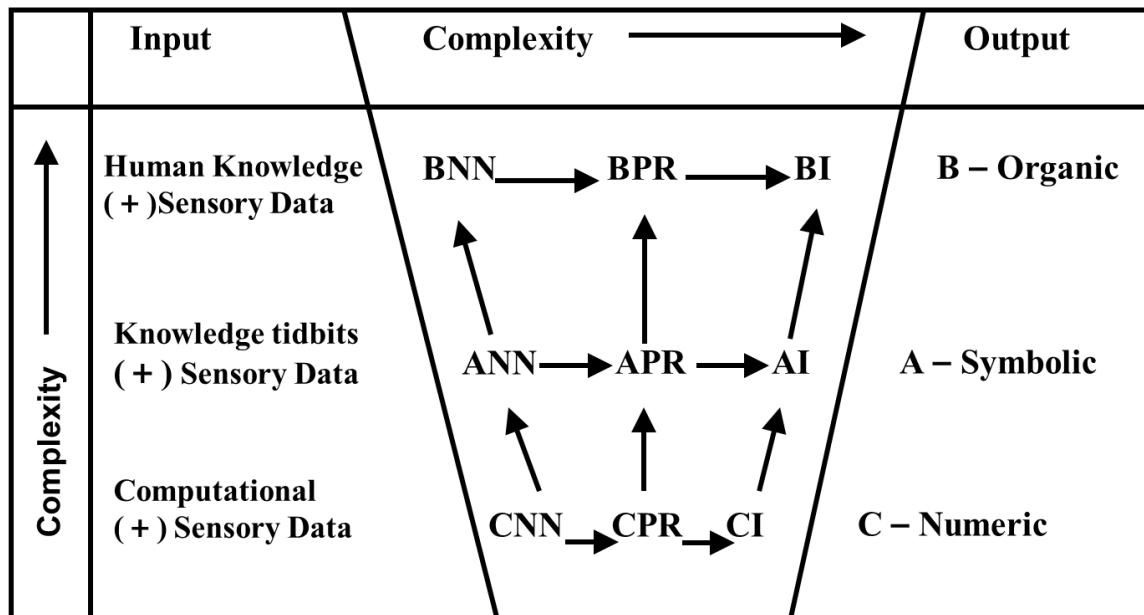


贝兹德克
(J.C. Bezdek)

概述

ABC 及其与神经网络(NN), 模式识别(PR)和智能(I)之间的关系

Later, in 1994, Bezdek offers that CI is low-level computation in the style of the mind", whereas AI is mid-level computation in the style of the mind".



Relationships among components of intelligent system (after Bezdek 1994)



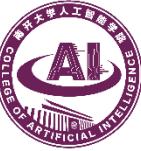
概述

表 ABC及相关领域的定义

BNN	人类智能硬件: 大脑	人的传感输入的处理
ANN	中层模型: CNN+知识精品	以大脑方式中的中层处理
CNN	低层, 生物激励模型	以大脑方式的传感数据处理
BPR	对人的传感数据结构的搜索	对人的感知环境中结构的识别
APR	中层模型: CPR+知识精品	中层数值和语法处理
CPR	对传感数据结构的搜索	所有CNN+模糊、统计和确定性模型
BI	人类智能软件: 智力	人类的认知、记忆和作用
AI	中层模型: CI+知识精品	以大脑方式的中层认知
CI	计算推理的低层算法	以大脑方式的低层认知

"...a system is computational intelligence when it deal only with the numerical (low-level) data, has a pattern recognition component, and does not use knowledge in the AI sense, and additionally when it exhibit:

- ① Computational adaptivity
- ② Computational fault tolerance
- ③ Speed approaching human-like turnaround
- ④ Error rates that approximate human performance



概述

➤ Some other opinions:

➤ Conference: Computational Intelligence

➤ Defining "Computational Intelligence"

impossible, to accommodate in a single definition all the well-established individualities such as fuzzy logic, neural networks, evolutionary computation, machine learning, Bayesian reasoning, etc.

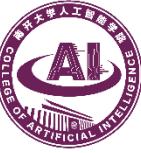
➤ Book: “Computational Intelligence: An Introduction”, Andries P. Engelbrecht, Wiley 2002

➤ Computational intelligence is the study of adaptive mechanisms to enable or facilitate

intelligent behavior in complex and changing environments. As such, computational intelligence works on neural networks, evolutionary computing, swarm

要定义“计算智能”并不简单。要在一个正式的定义中容纳诸如模糊集合、神经网络、进化计算、机器学习、贝叶斯推理等各自具有其既定特性的不同领域，即便不是不可能，也是非常困难的。

计算智能是关于在复杂和变化的环境中如何实现智能行为的自适应机制研究。因此，计算智能综合了人工神经网络、进化计算、群智能和模糊系统。



人工神经网络

Artificial Neural Networks

神经计算是以神经网络为基础的计算



人工神经网络(ANN)

Introduction of ANN

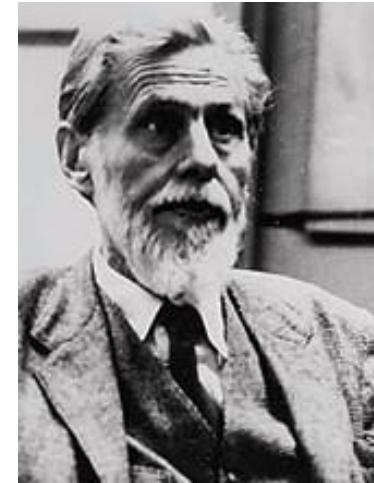
- ① 神经网络是由多个非常简单的处理单元彼此按某种方式相互连接而形成的计算系统，该系统是靠其状态对外部输入的动态响应处理信息的。
- ② 人工神经网络是一个许多简单的并行工作的处理单元组成的系统，其功能取决于网络的结构、连接强度以及各单元的处理方式。
- ③ 人工神经网络是一种旨在模仿人脑结构及其功能的信息处理系统。

ANN研究历史

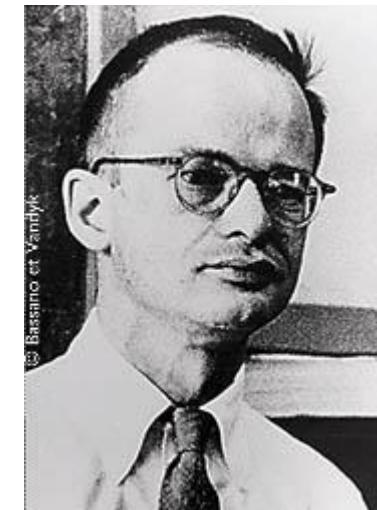
- ◎ McCulloch & Pitts (1943) 被公认为第一个人工神经网络的设计者 (MP model)

他们使用阈值以及用多个简单单元结合在一起以提高计算能力的思想到今天仍被广泛使用

- ◎ 试图通过用计算模型模拟生物神经系统的方法来理解生物神经系统的工作模式
- ◎ 高度的并行性使得其计算效率非常高
- ◎ 有助于理解神经表示的“分布式”特征



麦克洛奇(McCulloch)



皮茨 (Pitts)

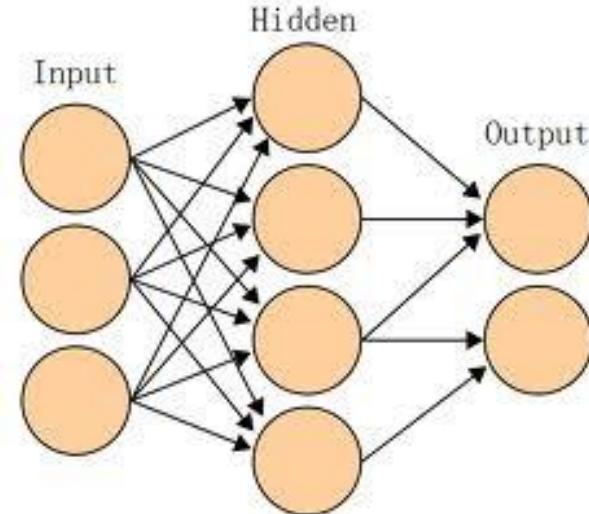


ANN的研究历史

- ◎1949年Hebb提出了第一个神经网络的学习规则
- ◎1958年Rosenblatt提出感知器 (perceptron)模型，1950' s 和1960' s, 许多研究者致力于研究该模型
- ◎1969年 Minsky和Papert展示了感知器模型的局限性，从此神经网络的研究陷入低潮，沉寂了大概15年
- ◎1980' s, 神经网络重新成为研究热点
 - ◎1982 Hopfield: 递归神经网络(recurrent network model, RNN)
 - ◎1982 Kohonen:自组织神经网络(self-organizing maps, SOFM)
 - ◎1986 Rumelhart: 反向传播网络(backpropagation)

ANN的特性

- 并行分布处理
- 非线性映射
- 通过训练进行学习
- 适应与集成
- 硬件实现



这些特性使得人工神经网络具有应用于各种智能系统的巨大潜力

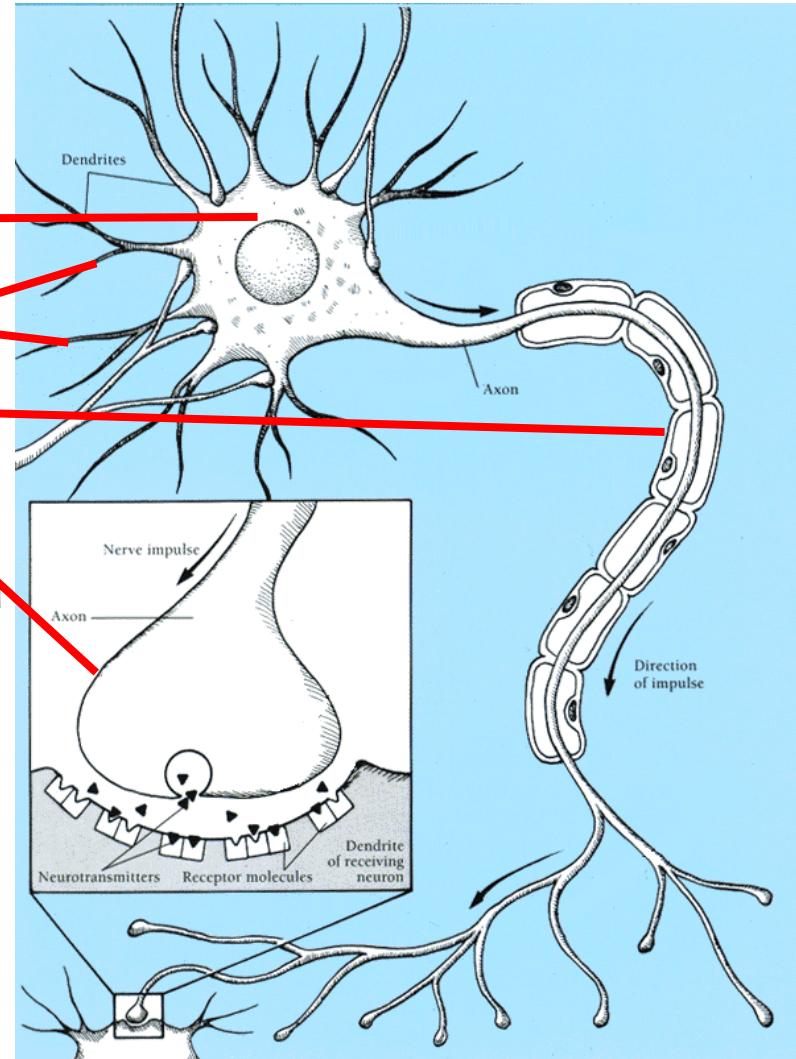
生物神经元模型

Cell structures

- Cell body
- Dendrites(树突)
- Axon(轴突)
- Synapse(突触)

● $10^{11} - 10^{12}$ neurons in human brain

● Each neuron connected to 10^4 others on average



生物神经元结构

- 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。



生物神经元模型

生物神经元特征：

- ① 神经元及其联接；
- ② 神经元之间的联接强度决定信号传递的强弱；
- ③ 神经元之间的联接强度是可以随训练改变的；
- ④ 信号可以起刺激作用，也可以起抑制作用；
- ⑤ 一个神经元接受的信号的累计效果决定该神经元的状态；
- ⑥ 每个神经元可以有一个“阈值”；

ANN的结构

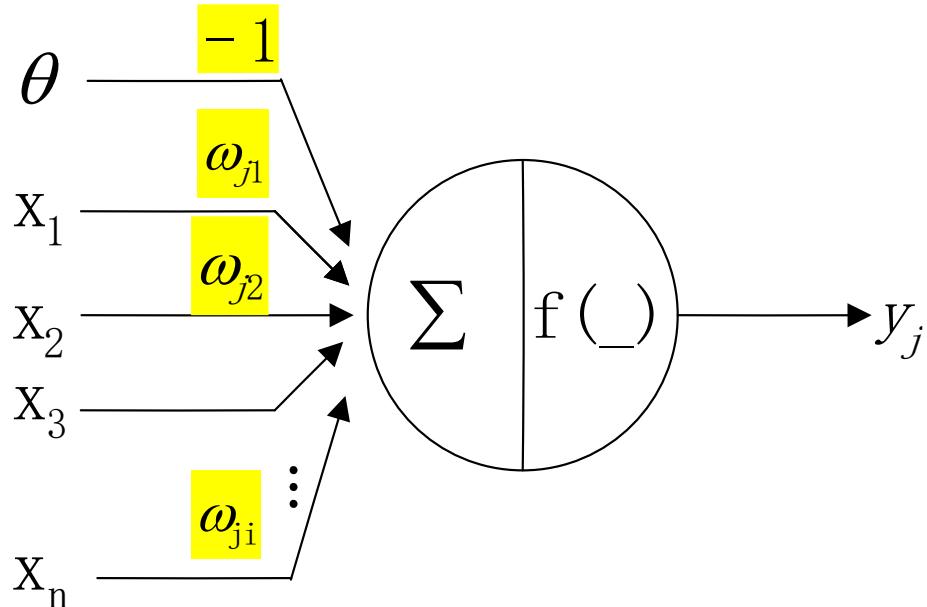
神经元及其特性(MP模型): 非线性阈值元件模型

In (Net)

$$u_j(t) = \sum_{i=1}^n \omega_{ji} x_i - \theta_j$$

Response

$$y_j(t) = f\left(\sum_{i=1}^n \omega_{ji} x_i - \theta_j\right)$$



θ_j

神经元单位的偏置(bias)或阈值(threshold)

ω_{ji}

连接强度/连接权系数(对于激发状态, ω_{ji} 取正值, 对于抑制状态, ω_{ji} 取负值)

n

输入信号数目

y_j

神经元输出

t

时间

$f(\cdot)$

输出变换函数(transfer function), 或激励函数(activation function)

ANN的结构

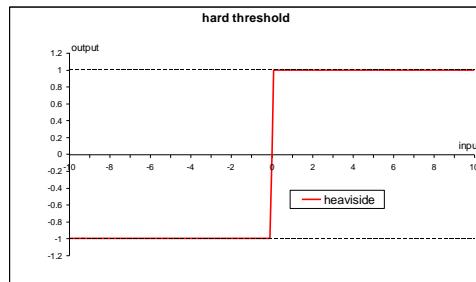
输出变换函数(激励函数)

输出变换函数往往采用0和1这种二值函数或S形函数

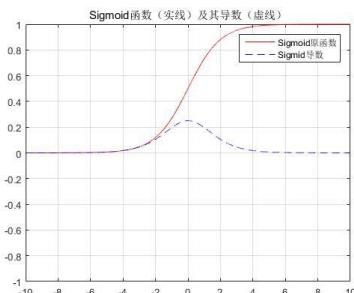
$$f(x) = \begin{cases} 1, & x \geq x_0 \\ 0, & x < x_0 \end{cases}$$

$$f(x) = \frac{1}{1 + e^{-ax}}, 0 < f(x) < 1$$

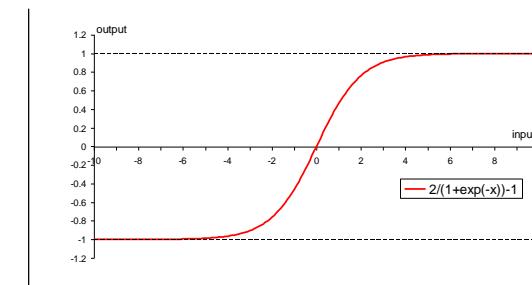
$$f(x) = \frac{1 - e^{-ax}}{1 + e^{-ax}}, -1 < f(x) < 1$$



(a) 二值函数



(b) S形函数



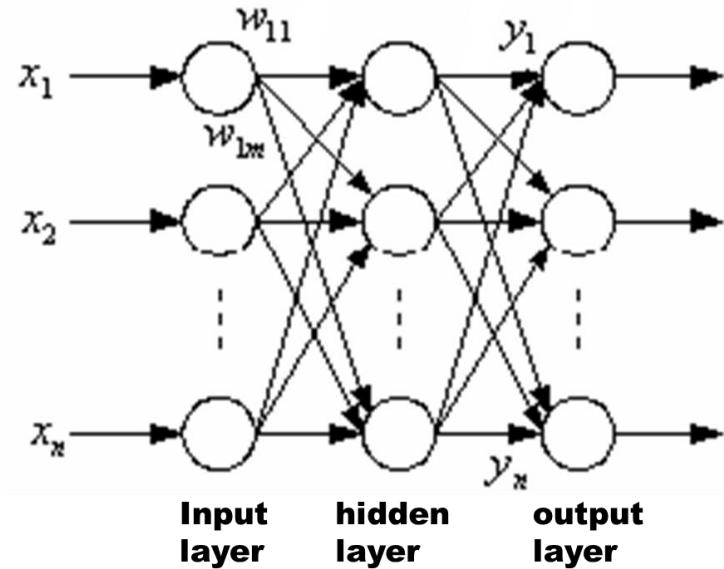
(c) 双曲正切函数

图 神经元中的某些变换(激励)函数

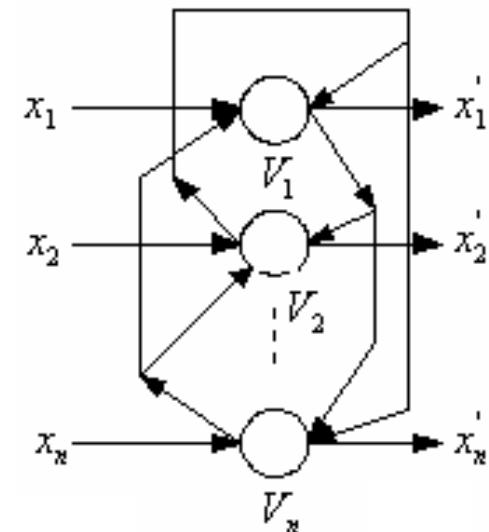
ANN的结构

基本分类

人工神经网络的结构基本上分为两类，即递归(反馈)网络和前馈网络：



(a)前馈网络



(b)递归(反馈)网络



ANN的结构

人工神经网络由神经元模型构成，这种由许多神经元组成的信息处理网络具有并行分布结构。每个神经元具有单一输出，并且能够与其他神经元连接；存在许多(多重输出连接方式)，每种连接方法对应于一个连接权系数。

严格地说，人工神经网络是一种具有下列特征的有向图：

- ① 对于每个节点*i*存在一个状态变量 x_i ；
- ② 从节点*i*至节点*j*，存在一个连接权系数 ω_{ij} ；
- ③ 对于每个节点*i*，存在一个阈值 θ_i ；
- ④ 对于每个节点*i*，定义一个变换函数 $f_i(x_j, \omega_{ij}, \theta_i)$ ， $i \neq j$ ；对于最一般的情况，此函数取 $f_i(\sum_i \omega_{ij} x_j - \theta_i)$ 形式。



ANN工作原理

网络学习的准则：

如果网络作出错误的判决，则通过网络的学习，应使得网络减少下次犯同样错误的可能性。将模式分布地记忆在网络的各个连接权值上。当网络再次遇到其中任何一个模式时，能够作出迅速、准确的判断和识别。

神经网络的工作过程主要由两个阶段组成：

- 一个阶段是**工作期**，此时各连接权值固定，计算单元的状态变化，以求达到稳定状态，
- 另一阶段是**学习期**(自适应期或设计期)，此时各计算单元状态不变，各连接权值可修改(通过学习样本或其他方法)。



ANN学习规则

神经网络的学习过程：

首先设定初始权值，如果无先验知识，初始权值可设定为随机值。接着输入样本数据进行学习，参照评价标准进行评判。如果达到要求，就停止学习，否则按照给定的学习法则调整权值，继续进行学习，直到取得满意的结果为止。

神经网络的学习规则(以Hebb规则为基础)，主要包括**误差传播式学习**、联想学习、竞争学习和基于知识的学习。

ANN的主要学习算法

人工神经网络主要通过两种学习算法进行训练，即指导式(有师)学习算法和非指导式(无师)学习算法。此外，还存在第三种学习算法，即强化学习算法，可把它看做是有师学习的一种特例。

- ① 有师学习、监督学习(Supervised Learning)

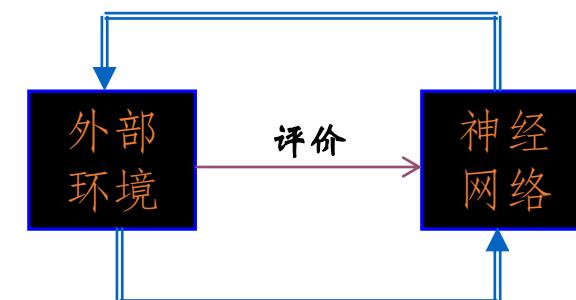
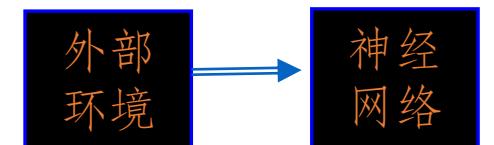
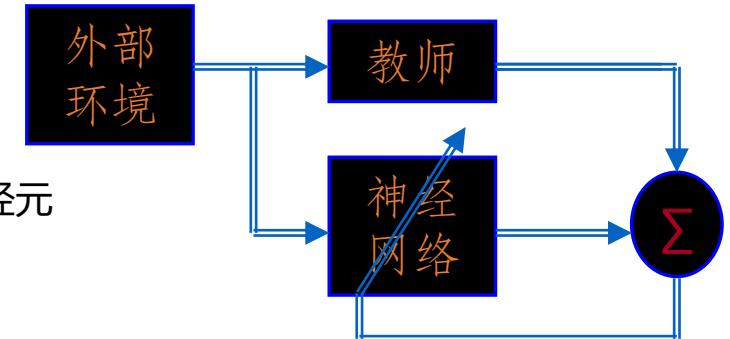
有师学习算法能够根据期望和实际的网络输出（对应于给定输入）之间的差来调整神经元间连接的强度或权。因此，有师学习需要老师或导师来提供期望目标或目标输出信号。

- ② 无师学习、非监督学习(Unsupervised Learning)

无师学习算法不需要知道期望输出。在训练过程中，只要向神经网络提供输入模式，神经网络就能自动地适应连接权，以便按相似特征把输入模式分组聚集

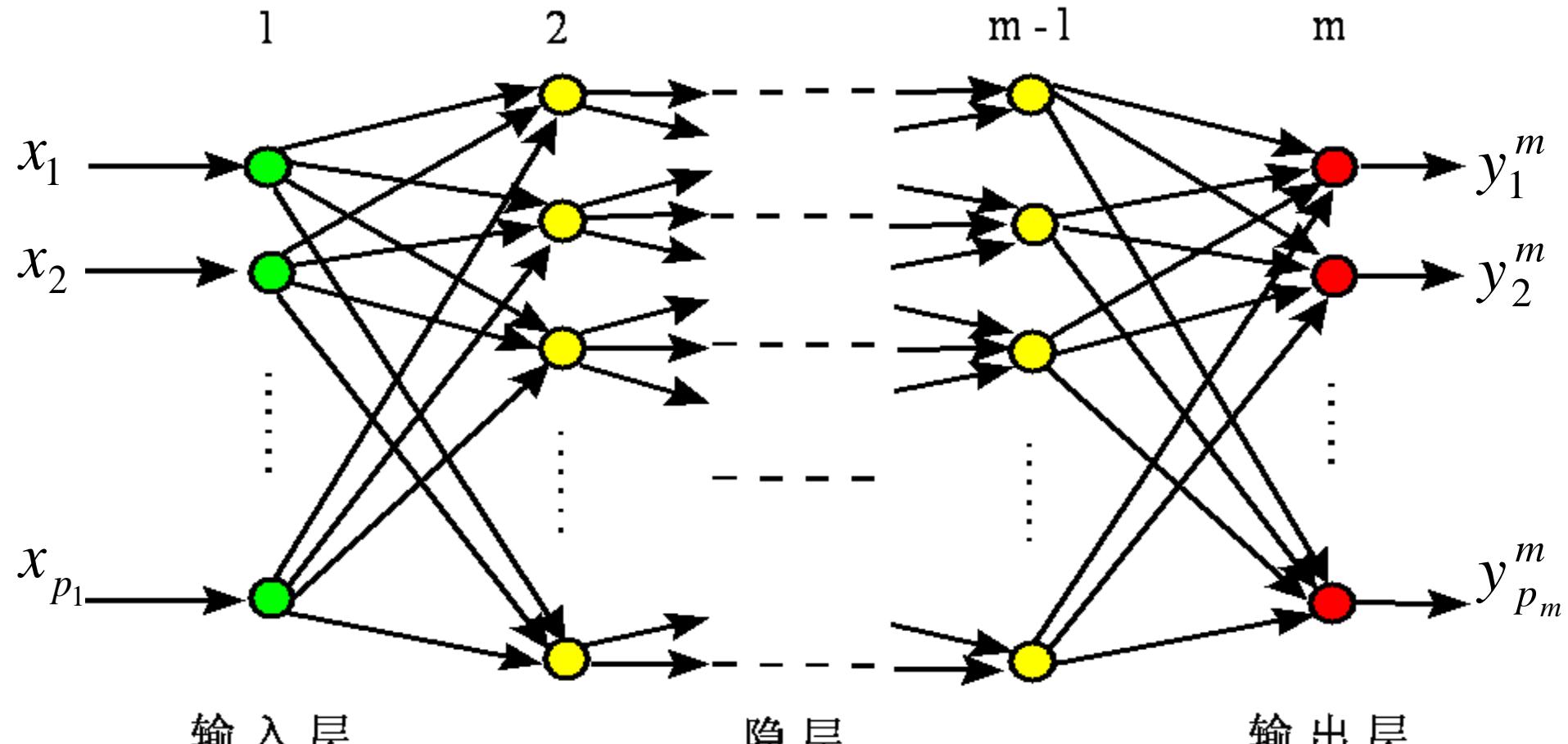
- ③ 再激励学习、增强学习、强化学习(Reinforcement Learning)

增强（强化）学习算法是有师学习的特例，它不需要老师给出目标输出。增强学习算法采用要给“评论员”来评价与给定输入相对应的神经网络输出的优度(质量因数)。增强学习算法的一个例子是遗传算法(GA)



BP神经网络的结构

1. BP网络结构



$$X = [x_1 \quad x_2 \quad \cdots \quad x_{p_1}]^T$$

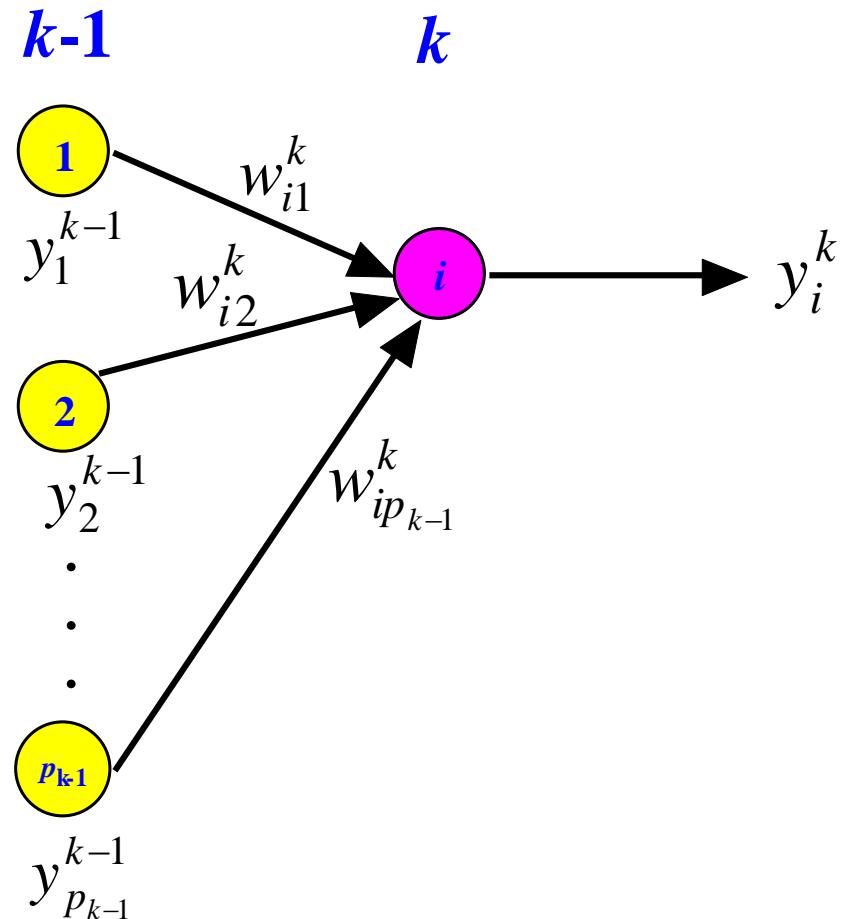
输入层
隐层

$$Y = [y_1^m \quad y_2^m \quad \cdots \quad y_{p_m}^m]^T$$

输出层

BP神经网络的结构

2. 输入输出变换关系



$$u_i^k = \sum_{j=1}^{p_{k-1}} w_{ij}^k y_j^{k-1} - \theta_i = \sum_{j=0}^{p_{k-1}} w_{ij}^k y_j^{k-1}$$
$$(y_0^{k-1} = \theta_i, w_{i0}^k = -1)$$

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

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

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

3. 工作过程

- 第一阶段或网络训练阶段：

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

$$d_i = [d_{i1}, d_{i2}, \dots, d_{ipm}]^T$$

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

- 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。

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 ε 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。

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:

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

BP学习算法

先求 $\frac{\partial J}{\partial w_{ij}^{k-1}} = \frac{\partial J}{\partial \mathbf{u}_i^k} \frac{\partial \mathbf{u}_i^k}{\partial w_{ij}^{k-1}} = \frac{\partial J}{\partial \mathbf{u}_i^k} \frac{\partial}{\partial w_{ij}^{k-1}} (\sum_j w_{ij}^{k-1} y_j^{k-1}) = \frac{\partial J}{\partial \mathbf{u}_i^k} y_j^{k-1}$

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

(1) 对输出层的神经元

$$\frac{\partial J}{\partial y_i^k} = \frac{\partial J}{\partial y_i^m} = y_i^m - y_{si}$$

$$d_i^m = (y_i^m - y_{si}) f'_m(\mathbf{u}_i^m)$$

(2) 对隐单元层，则有

$$\frac{\partial J}{\partial y_i^k} = \sum_l \frac{\partial J}{\partial \mathbf{u}_l^{k+1}} \frac{\partial \mathbf{u}_l^{k+1}}{\partial y_i^k} = \sum_l d_l^{k+1} w_{li}^k$$

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

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

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

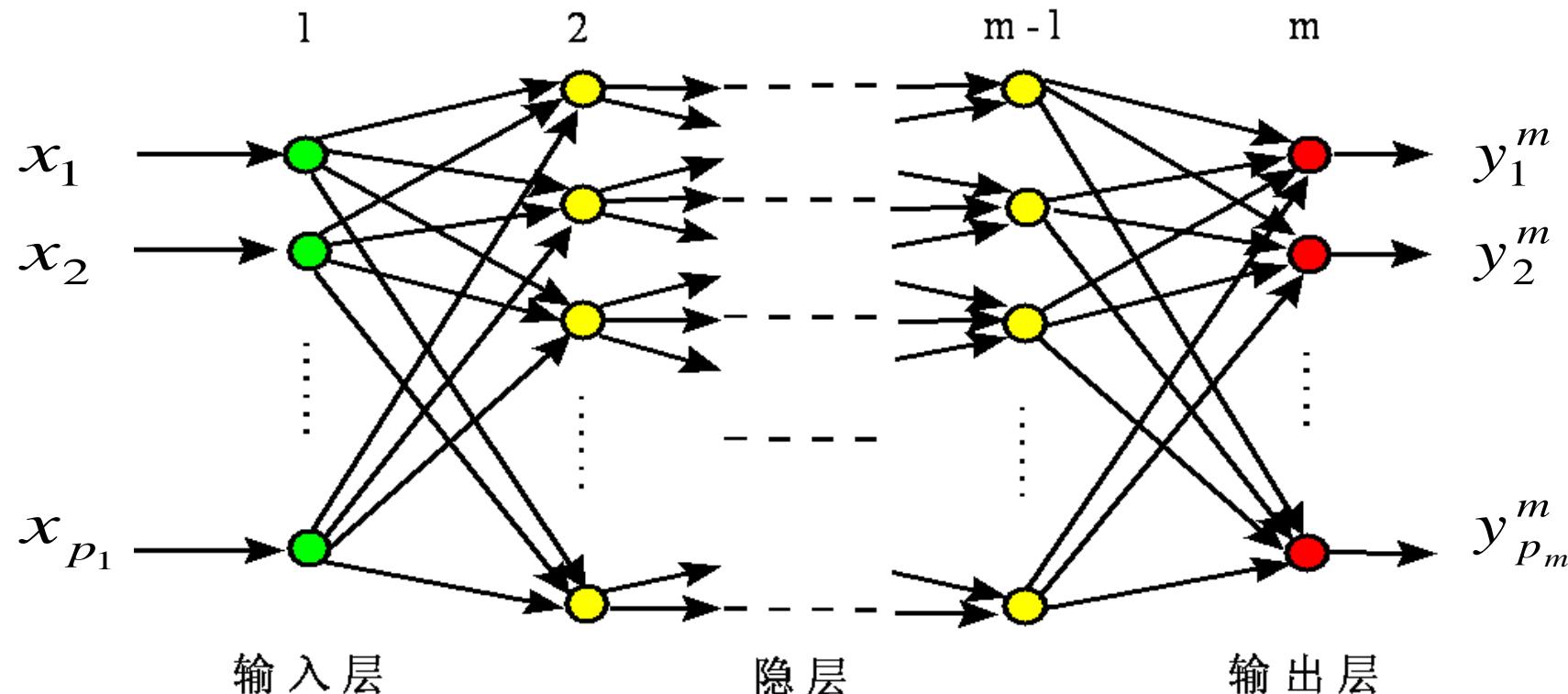
$$d_i^k = f'_k(u_i^k) \sum_l d_l^{k+1} w_{li}^k$$

——隐层连接权调整公式

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。



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

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，将所有的特征变换到[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。

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： $x = [x_1, x_2, \dots, x_{p_1}]^\top$,
 $d = [d_1, d_2, \dots, d_{p_m}]^\top$, 把输入信息 $x = [x_1, x_2, \dots, x_{p_1}]^\top$ 输入到BP网络中

(3) 正向传播：计算各层节点的输出：

$$y_i^k \quad (i = 1, \dots, p_k; k = 1, \dots, m)$$

(4) 计算网络的实际输出与期望输出的误差：

$$e_i = y_i - y_i^m \quad (i = 1, \dots, p_m)$$

BP算法的实现

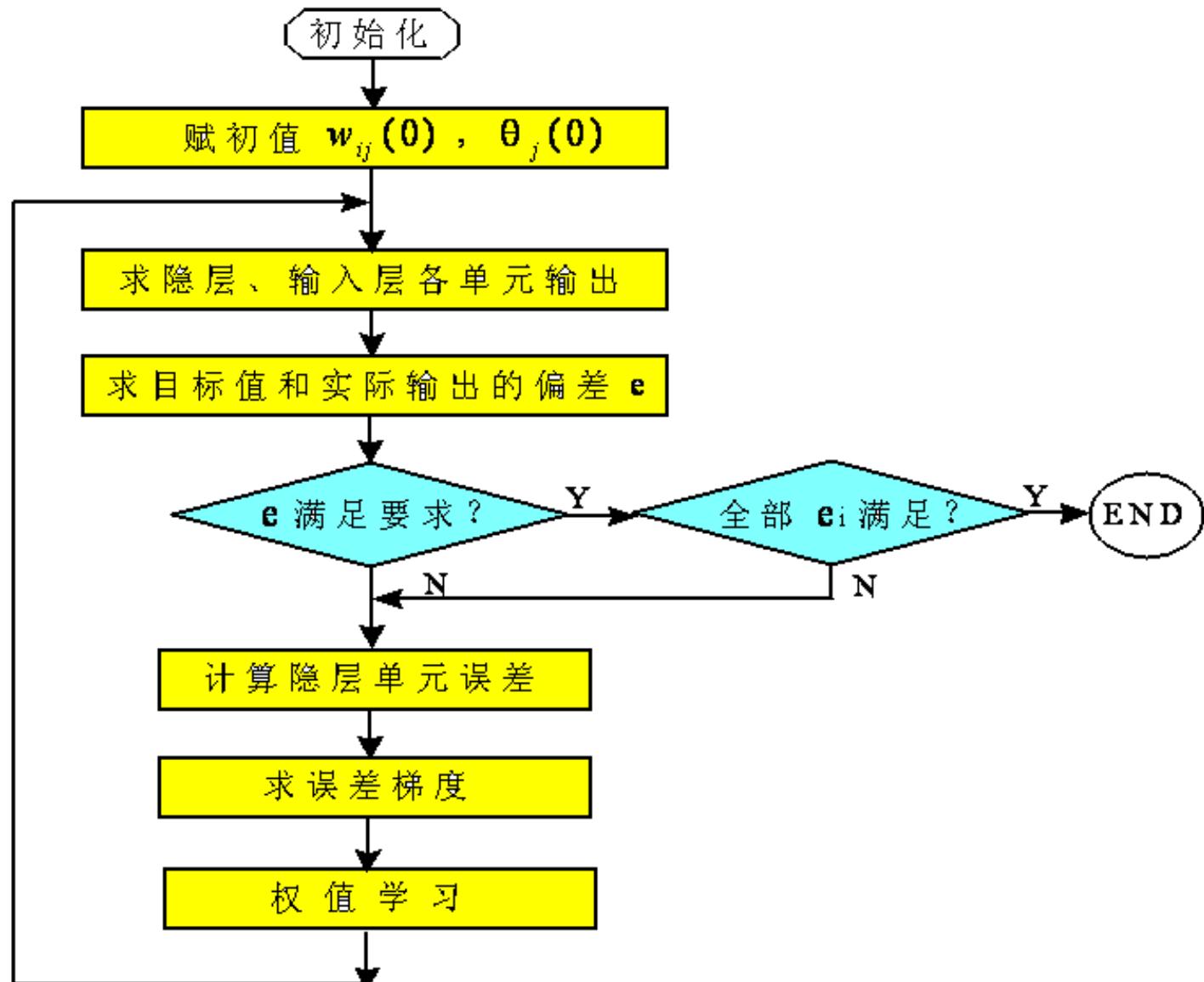
(5) 反向传播：从输出层方向计算到第一个隐层，按连接权值修正公式向减小误差方向调整网络的各个连接权值。

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

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

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

(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。



□ 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。

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。

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。

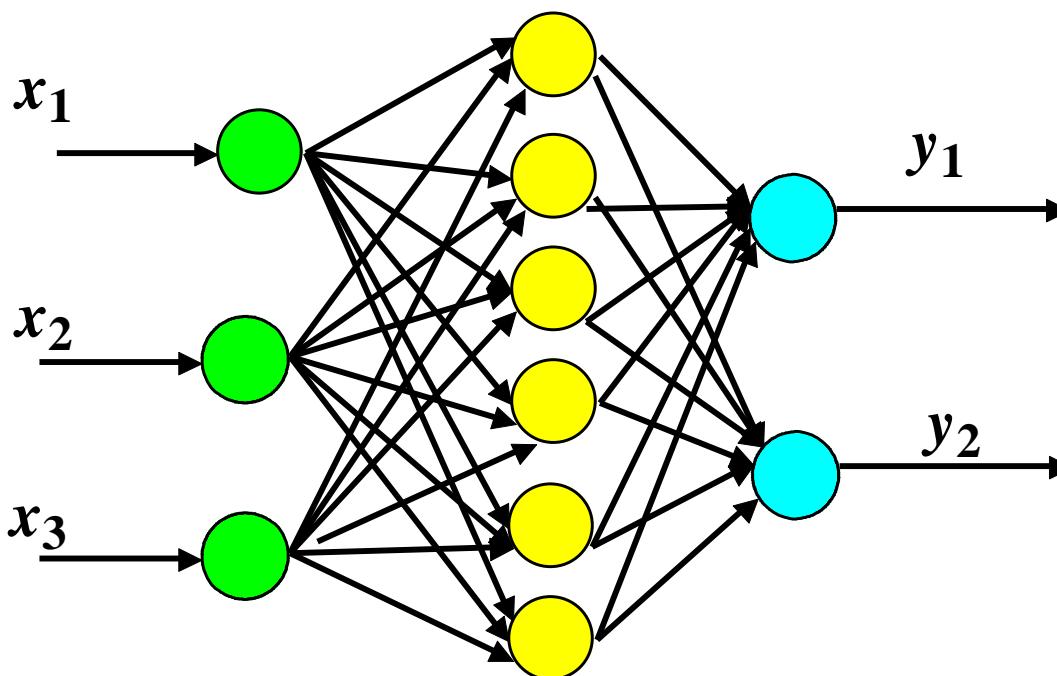
BP神经网络在模式识别中的应用

- 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

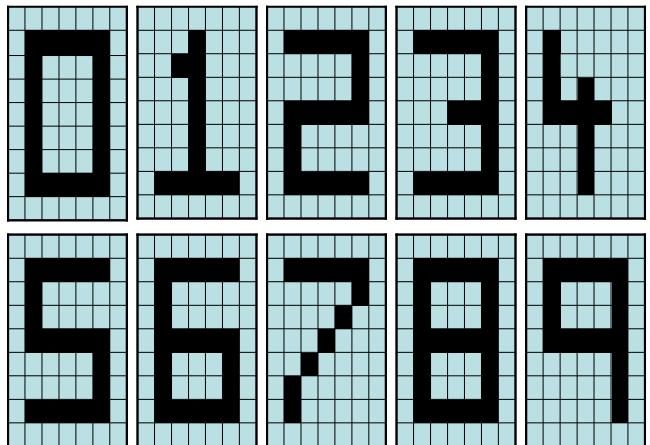
- 测试数据：

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



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。



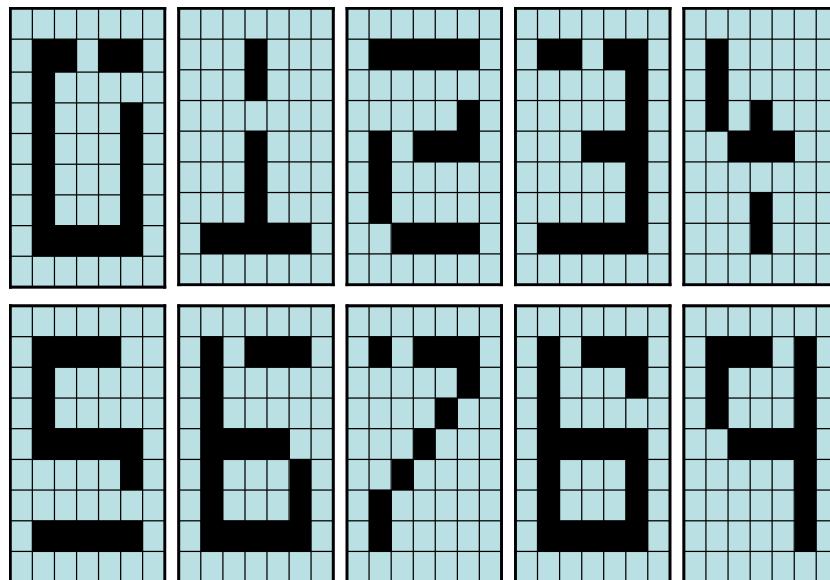
每个数字用 9×7 的网格表示，灰色像素代表0，黑色像素代表1。将每个网格表示为0, 1的长位串。位映射由左上角开始向下直到网格的整个一列，然后重复其他列。

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。

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。

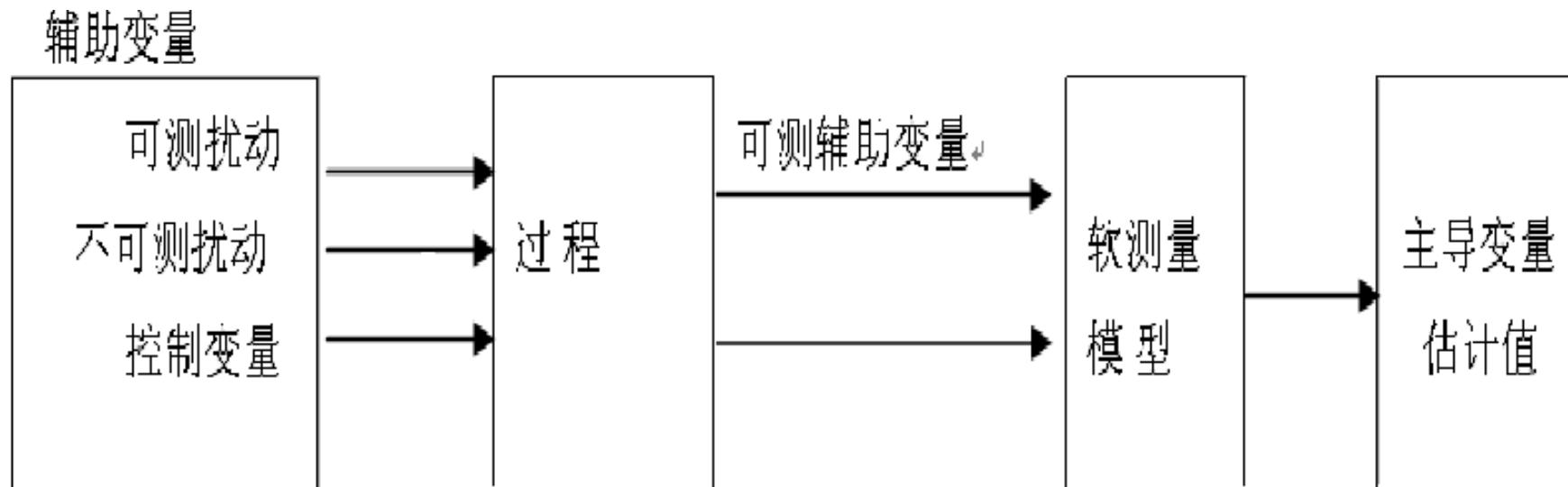


测试结果表明：除了8以外，所有被测的数字都能够被正确地识别。

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。

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。

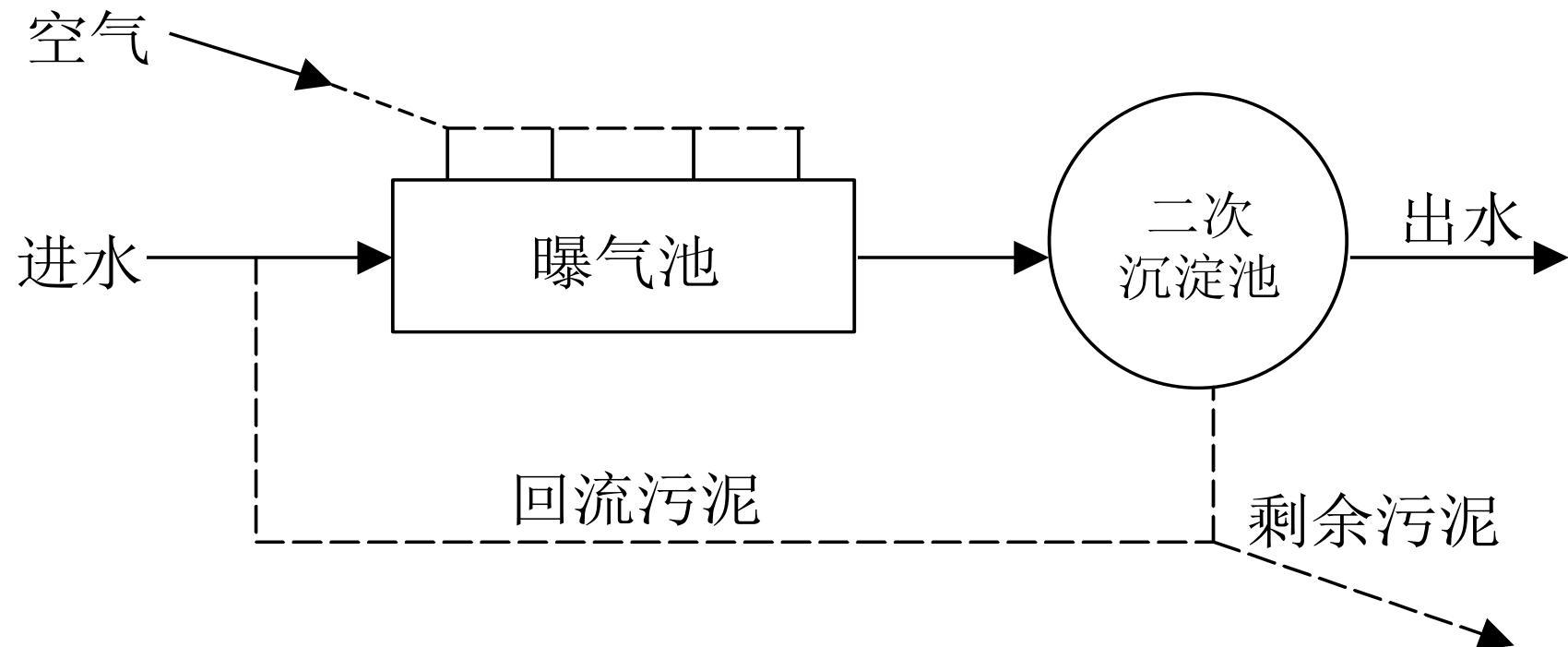
BP神经网络在软测量中的应用

□ 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。

BP神经网络在软测量中的应用

- 序批式活性污泥法（SBR）

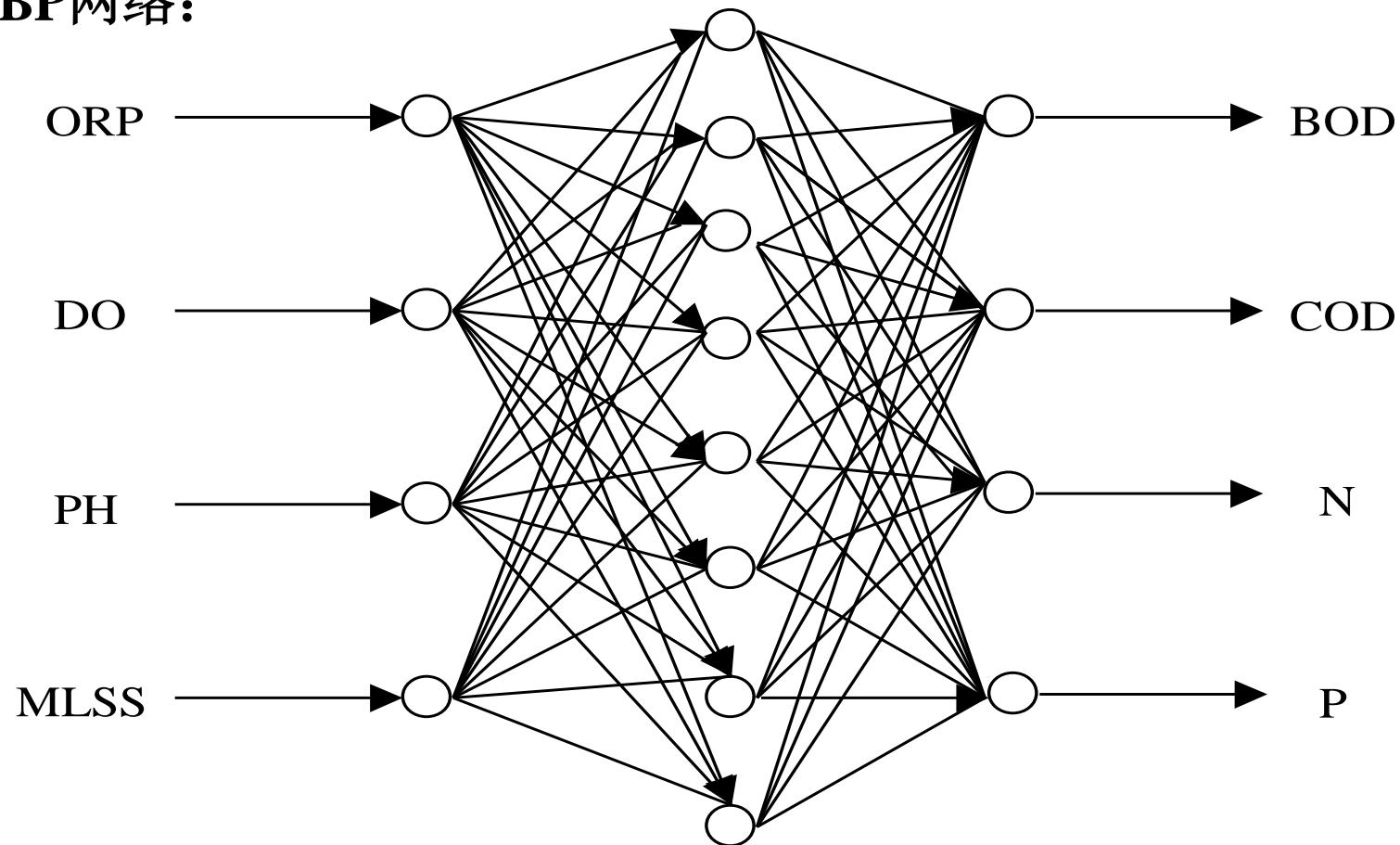


BP神经网络在软测量中的应用

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：



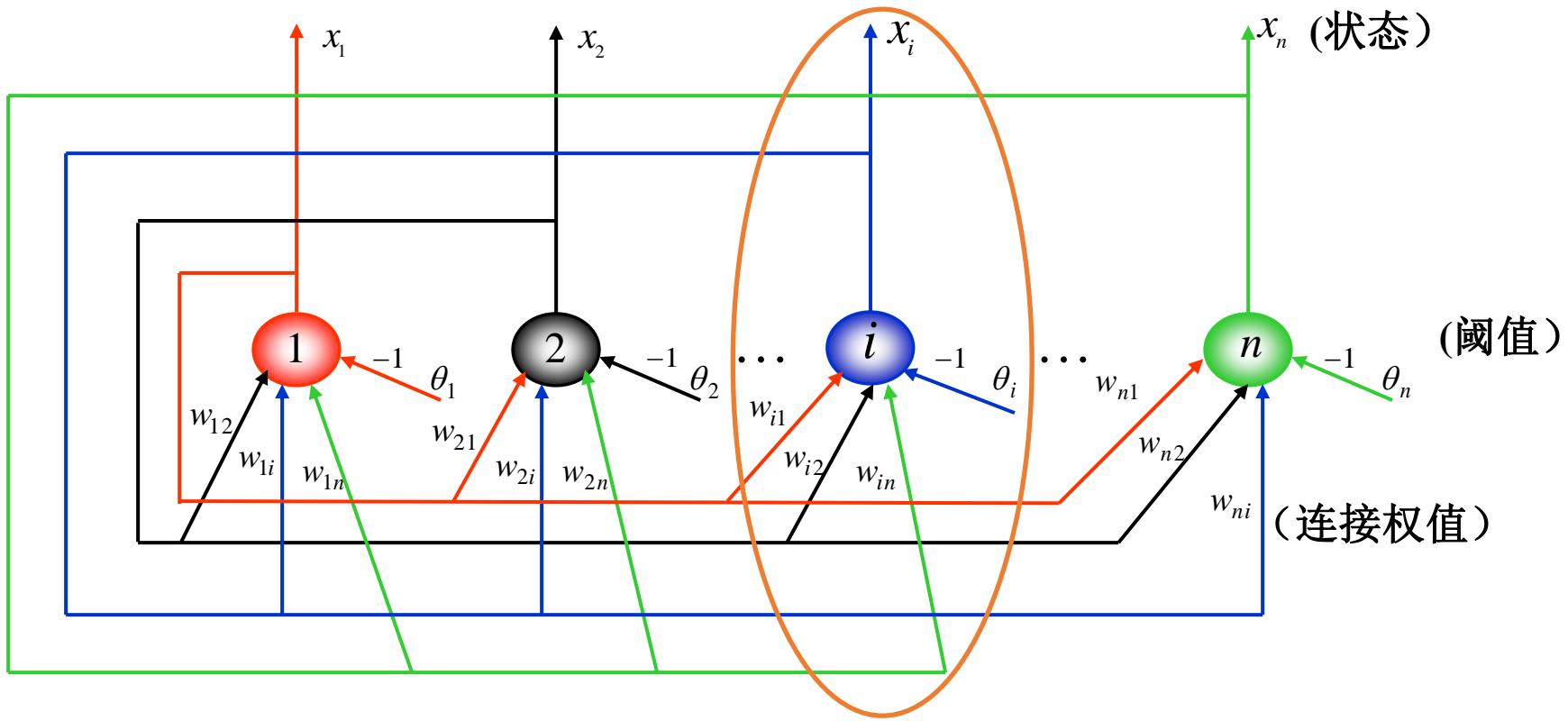
Hopfield 神经网络及其改进

- 离散型Hopfield神经网络
- 连续型Hopfield神经网络及其VLSI实现
- 随机神经网络
- 混沌神经网络

离散Hopfield神经网络

1. 离散Hopfield神经网络模型

- 网络结构：

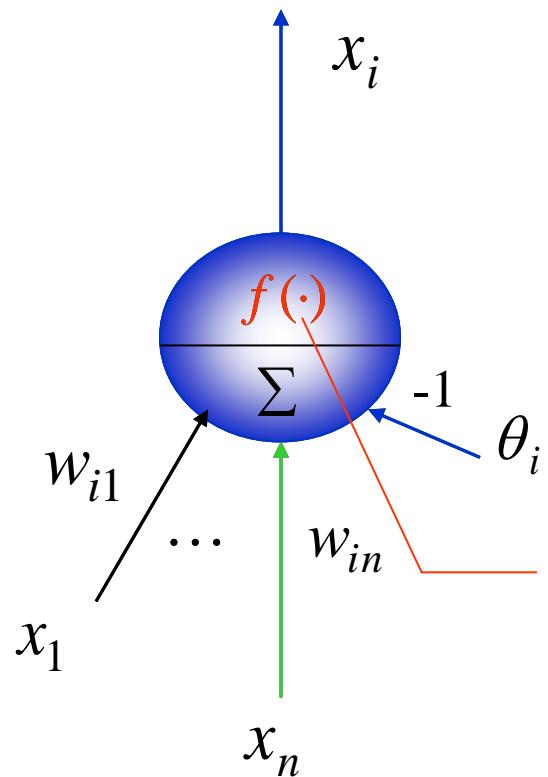


离散Hopfield神经网络结构图

离散Hopfield神经网络

1. 离散Hopfield神经网络模型

■ 输入输出关系：



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

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

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

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

注： $w_{ii}=0$

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

离散Hopfield神经网络

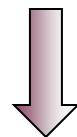
1. 离散Hopfield神经网络模型

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

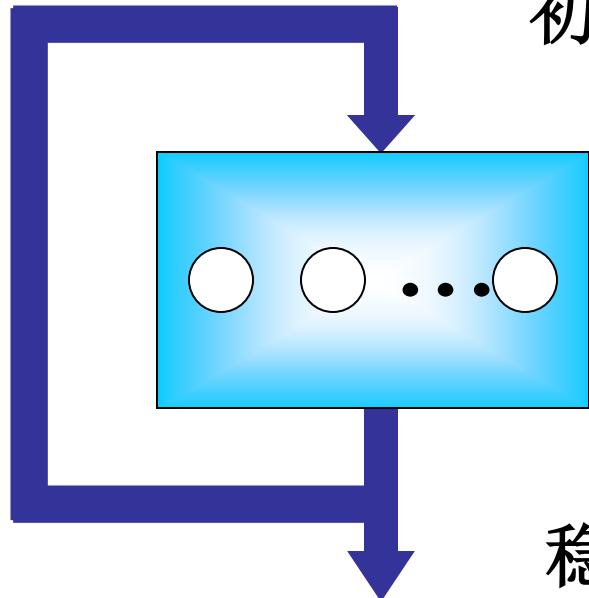


$$x(k+1) = f(W_x(k) - \theta), \forall i$$

离散Hopfield神经网络

1. 离散Hopfield神经网络模型

- 工作过程:



初态: $x(0) = [x_1(0), x_2(0), \dots, x_n(0)]^T$

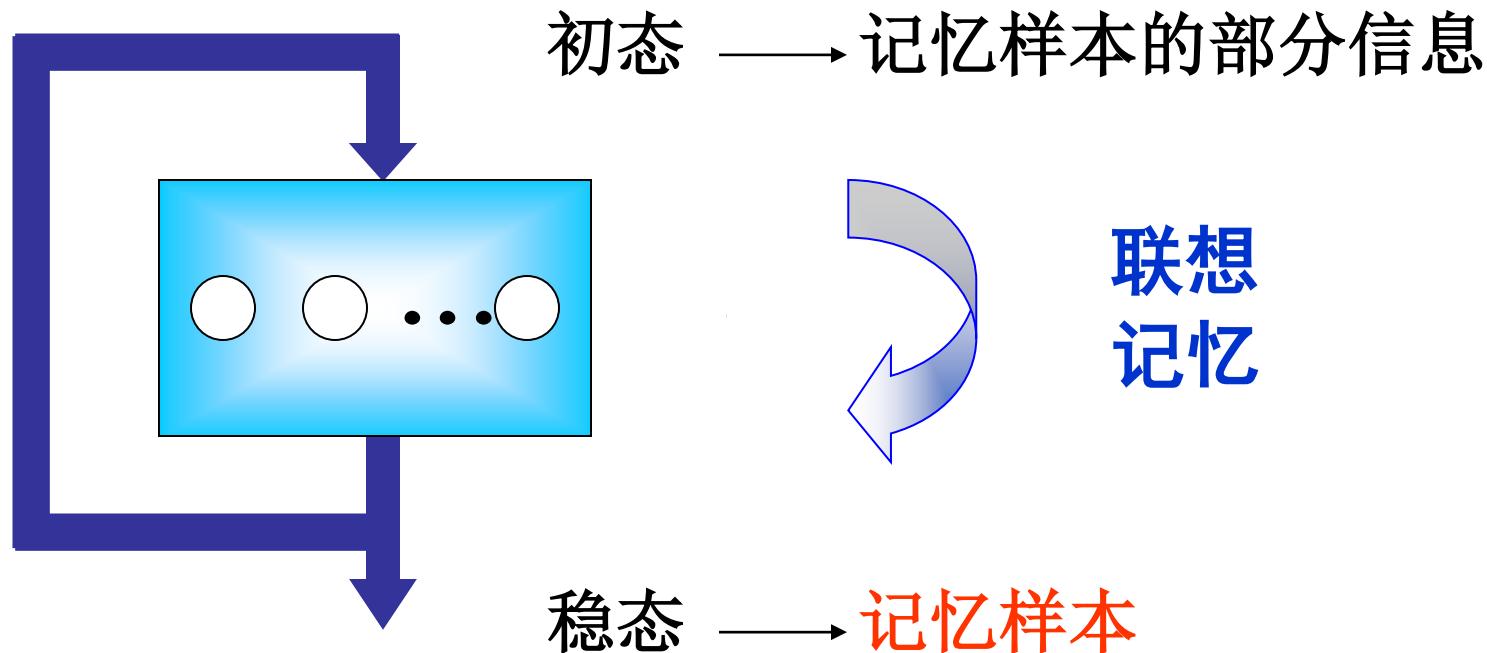
(异步或同步方式)

稳态: $\lim_{k \rightarrow \infty} x(k)$

离散Hopfield神经网络

1. 离散Hopfield神经网络模型

- 工作过程：



2. 网络的稳定性

■ 稳定性定义：

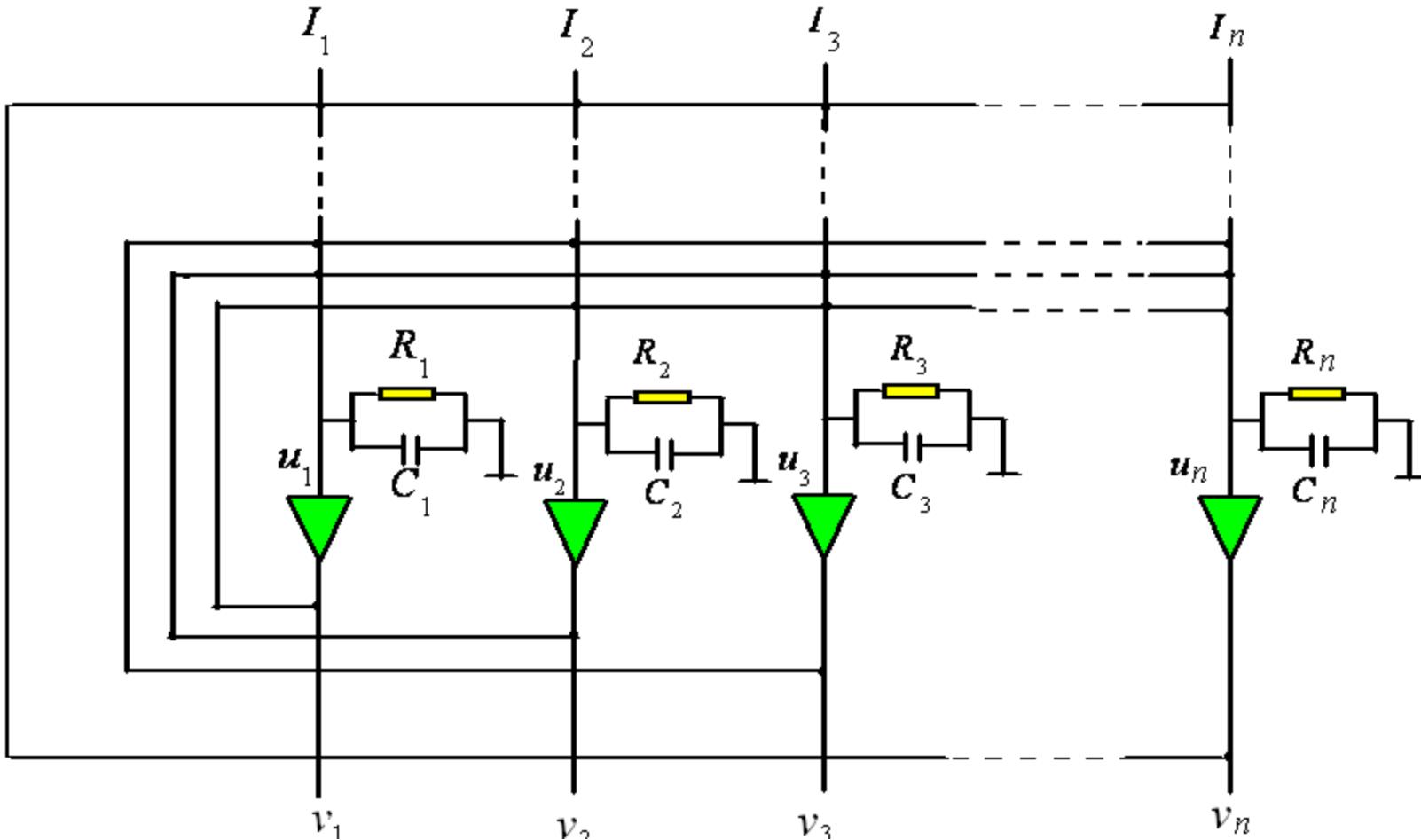
- 若从某一时刻开始，网络中所有神经元的状态不再改变，即 $x = f(w_x - \theta)$ ，则称该网络是稳定的， x 为网络的稳定点或**吸引子**。
$$x(k) = f(W_x(k) - \theta) = x(k+1)$$
- Hopfield神经网络是高维非线性系统，可能有许多稳定优态。从任何初始状态开始运动，总可以到某个稳定状态。这些稳定状态可以通过改变网络参数得到。

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

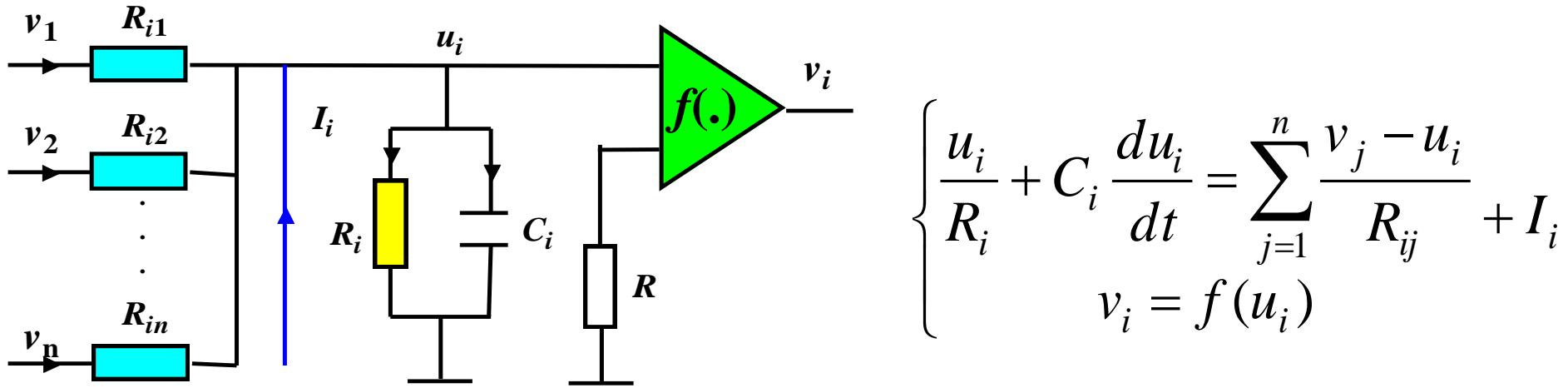
连续型Hopfield神经网络及其VLSI实现

1. 连续Hopfield神经网络模型



连续型Hopfield神经网络及其VLSI实现

1. 连续Hopfield神经网络模型



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

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

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

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

连续型Hopfield神经网络及其VLSI实现

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，若 $f^{-1}(\cdot)$ 为单调递增的连续函数， $C_i > 0$, $w_{ij} = w_{ji}$ ，则 $\frac{dE}{dt} \leq 0$ ；当且仅当 $\frac{dv_i}{dt} = 0$, ($1 \leq i \leq n$)时， $\frac{dE}{dt} = 0$

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。

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。

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：

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

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:

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

2. 高斯机 $\frac{du_i}{dt} = \sum_{j=1}^n w_{ij} v_j + I_i + \eta$

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

3. 柯西机

$$\frac{du_i}{dt} = \sum_{j=1}^n w_{ij} v_j + I_i + \eta$$

η : 柯西随机变量（有色噪声）

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。

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。

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：

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

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

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。

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)：

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

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

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

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

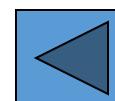
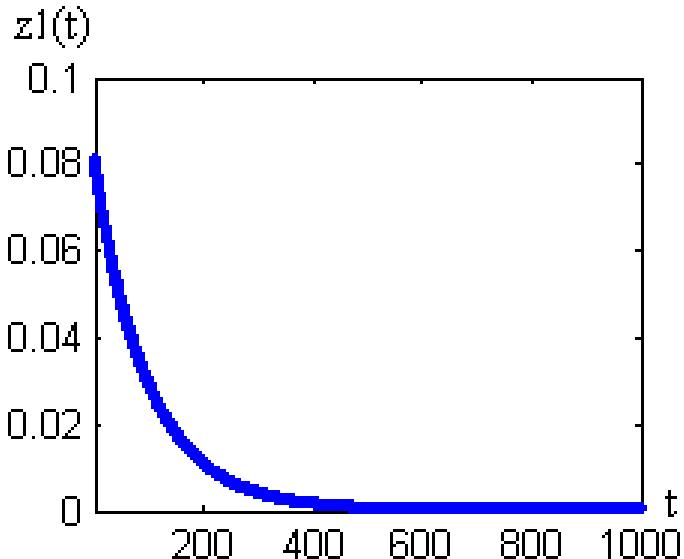
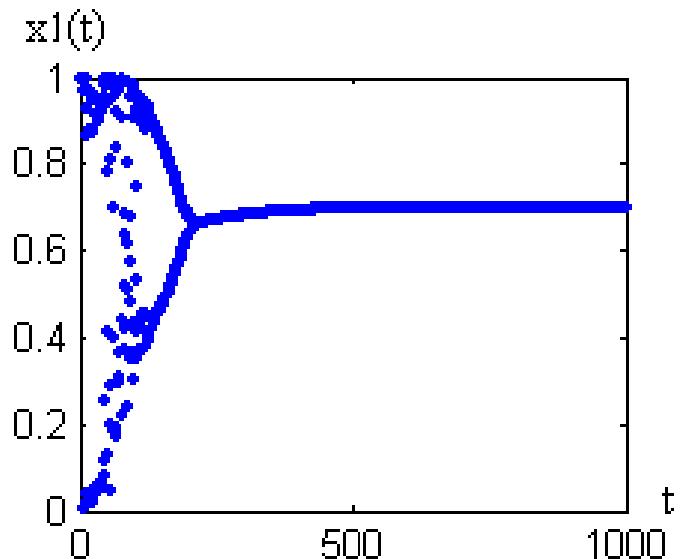
- ① 具有暂态混沌特性。
- ② 能演化到一个稳定状态。
- ③ 搜索区域为一分形结构。
- ④ 具有混沌退火机制。
- ⑤ 一种广义的混沌神经网络。
- ⑥ 可求解0-1问题，也可求解连续非线性优化问题。

混沌神经网络

□ 非线性函数:

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

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



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：

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

3. 混沌神经网络

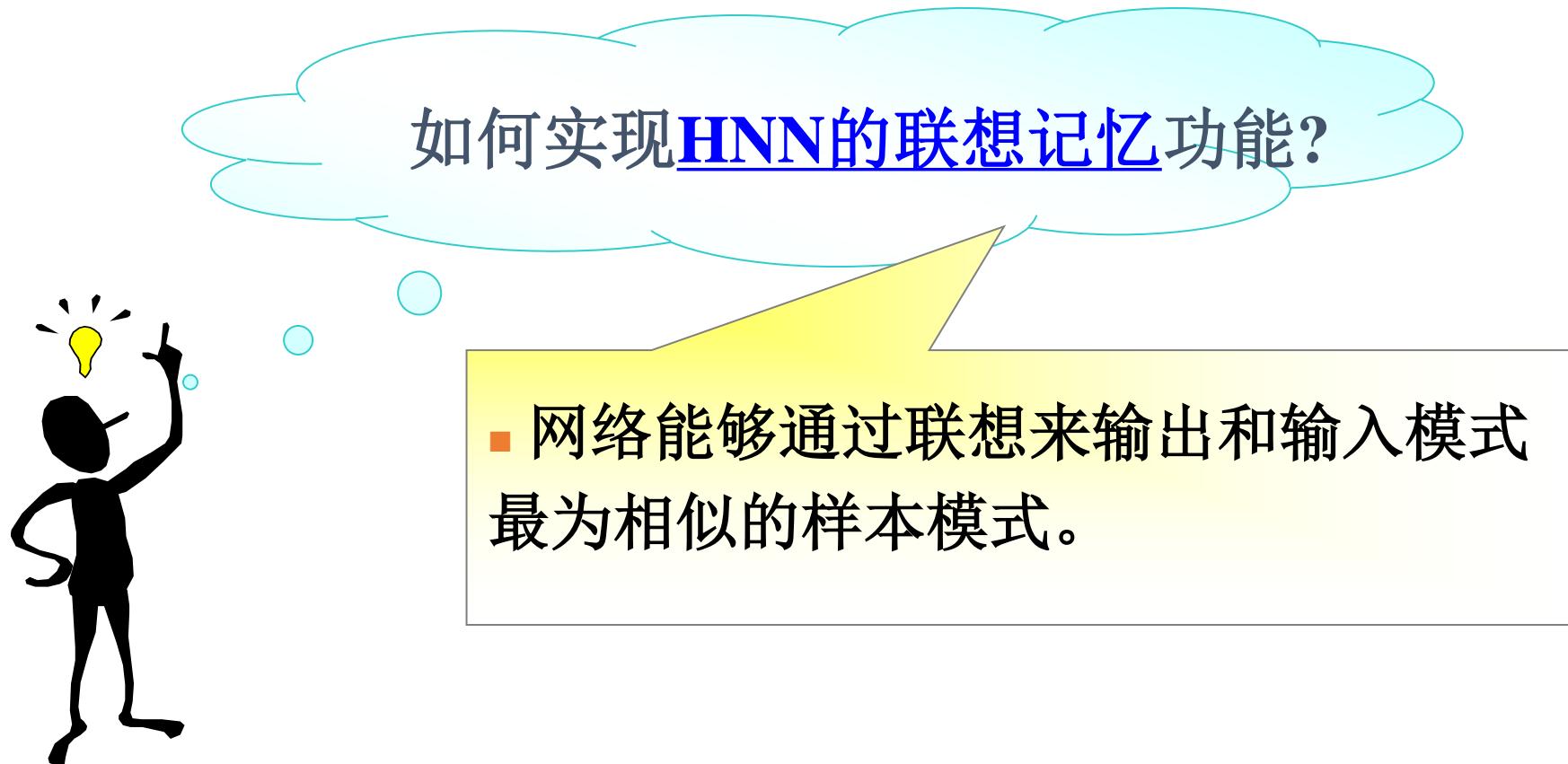
(3) 引入噪声的混沌神经网络

- 1995年，Hayakawa等的混沌神经网络：

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

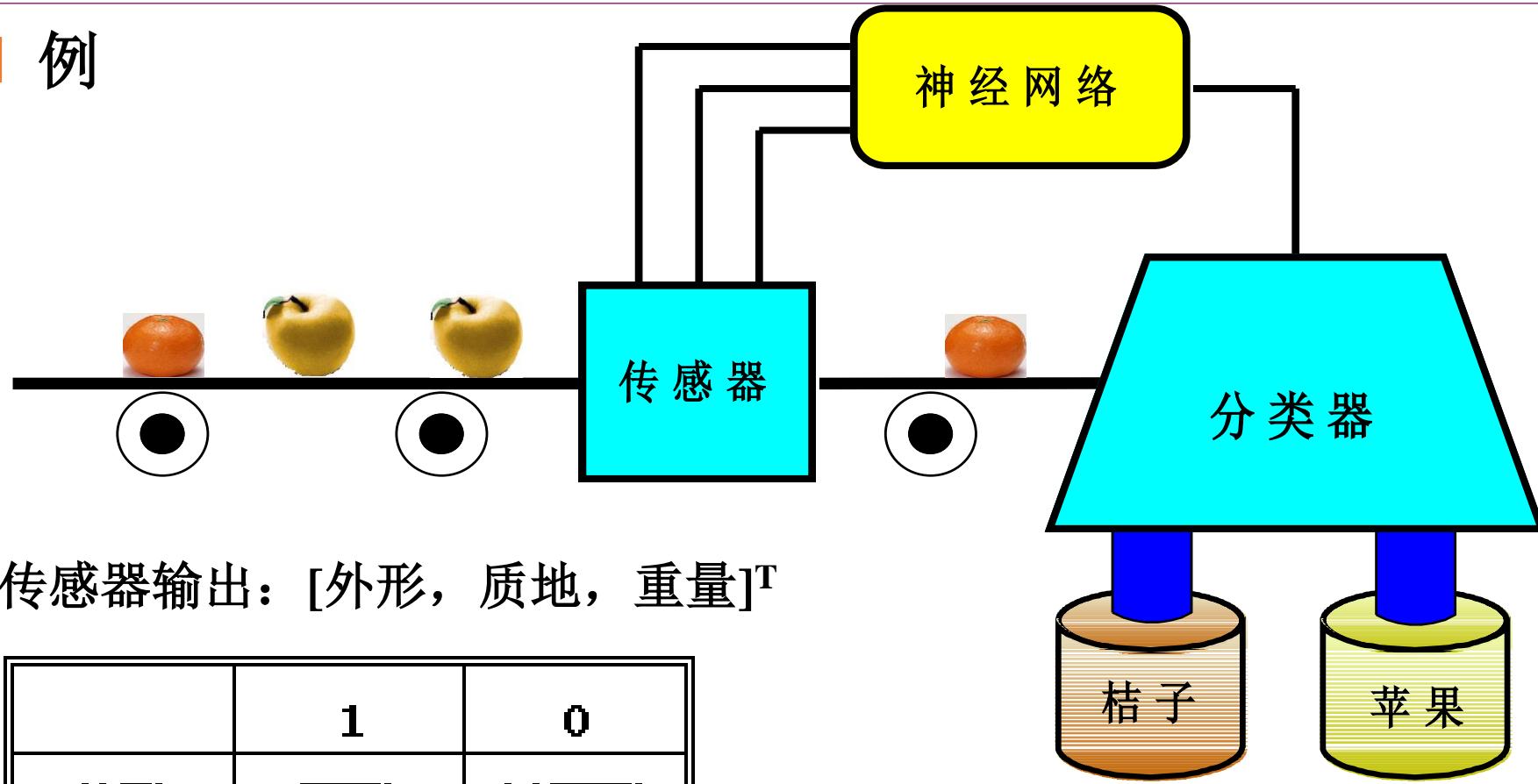
$$v_i(t) = f(u_i(t) + A \eta_i(t))$$

Hopfield神经网络在联想记忆中的应用



Hopfield神经网络在联想记忆中的应用

□ 例



■ 传感器输出: [外形, 质地, 重量]^T

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

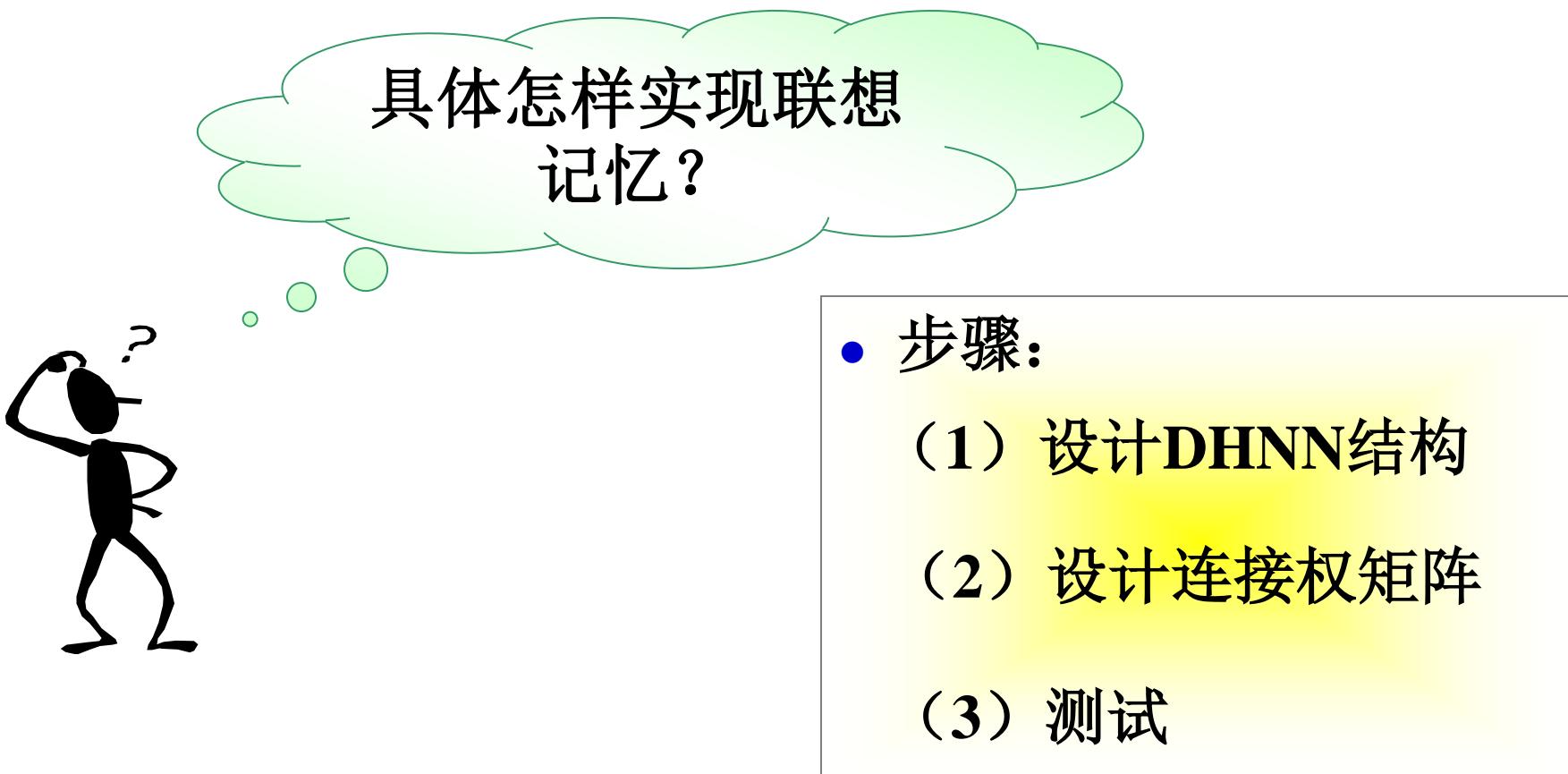
$$\text{标准桔子: } x^{(1)} = [1 \quad 0 \quad 1]^T$$

$$\text{标准苹果: } x^{(2)} = [0 \quad 1 \quad 0]^T$$

Hopfield神经网络在联想记忆中的应用

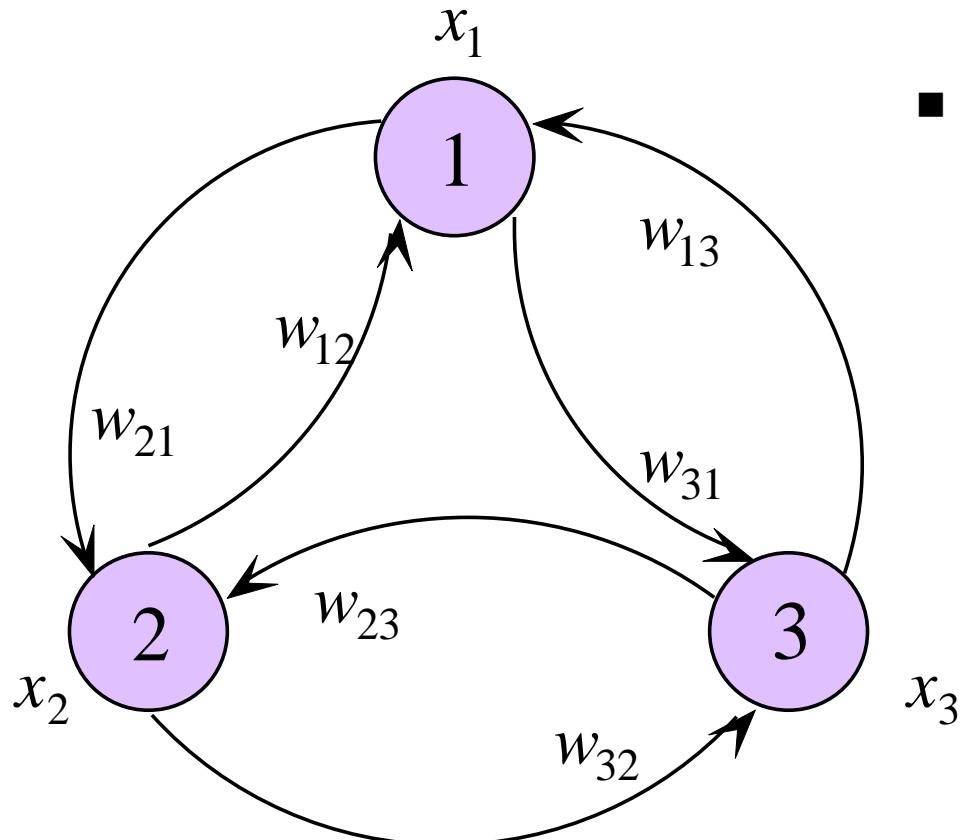
- 例 ■ 传感器输出: [外形, 质地, 重量]^T

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



Hopfield神经网络在联想记忆中的应用

■ (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: $\theta_i = 0 (i = 1, 2, 3)$

Hopfield神经网络在联想记忆中的应用

■ (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: $w_{ij} = \begin{cases} \sum_{l=1}^2 (2x_i^{(l)} - 1)(2x_j^{(l)} - 1) & i \neq j \\ 0 & i = j \end{cases}$
 $w_{ji} = w_{ij} (i = 1, 2, \dots, n; j = 1, 2, \dots, n)$

$$w_{12} = (2x_1^{(1)} - 1)(2x_2^{(1)} - 1) + (2x_1^{(2)} - 1)(2x_2^{(2)} - 1)$$

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

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

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

Hopfield神经网络在联想记忆中的应用

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

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

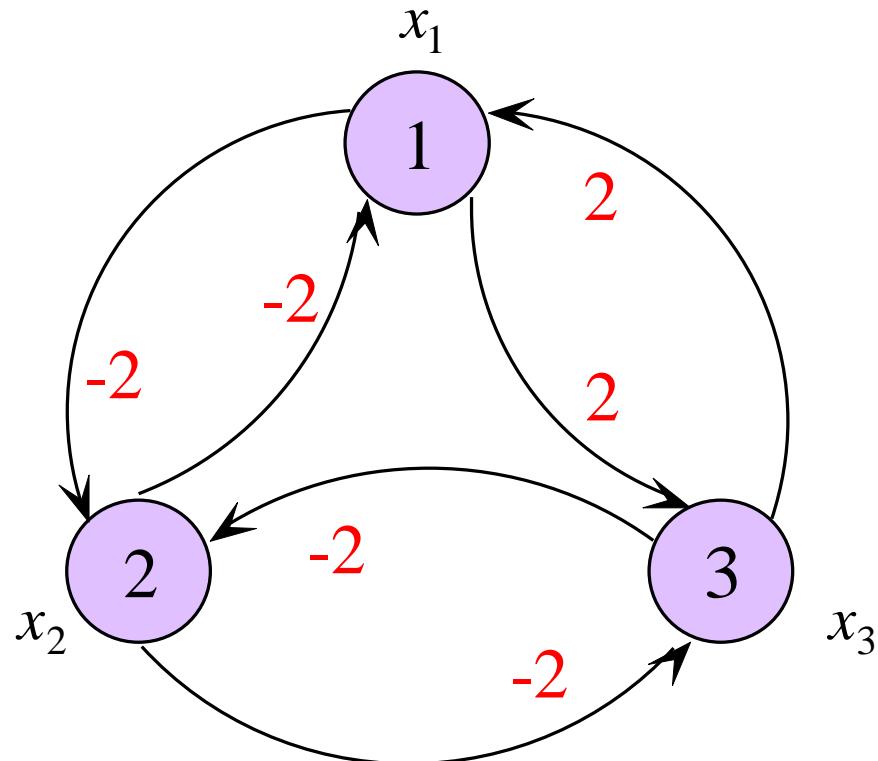
$$\begin{aligned} w_{23} &= (2x_2^{(1)} - 1)(2x_3^{(1)} - 1) + (2x_2^{(2)} - 1)(2x_3^{(2)} - 1) \\ &= (2 \times 0 - 1) \times (2 \times 1 - 1) + (2 \times 1 - 1) \times (2 \times 0 - 1) \\ &= -1 + (-1) = -2 \end{aligned}$$

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

Hopfield神经网络在联想记忆中的应用

■ (2) 设计连接权矩阵

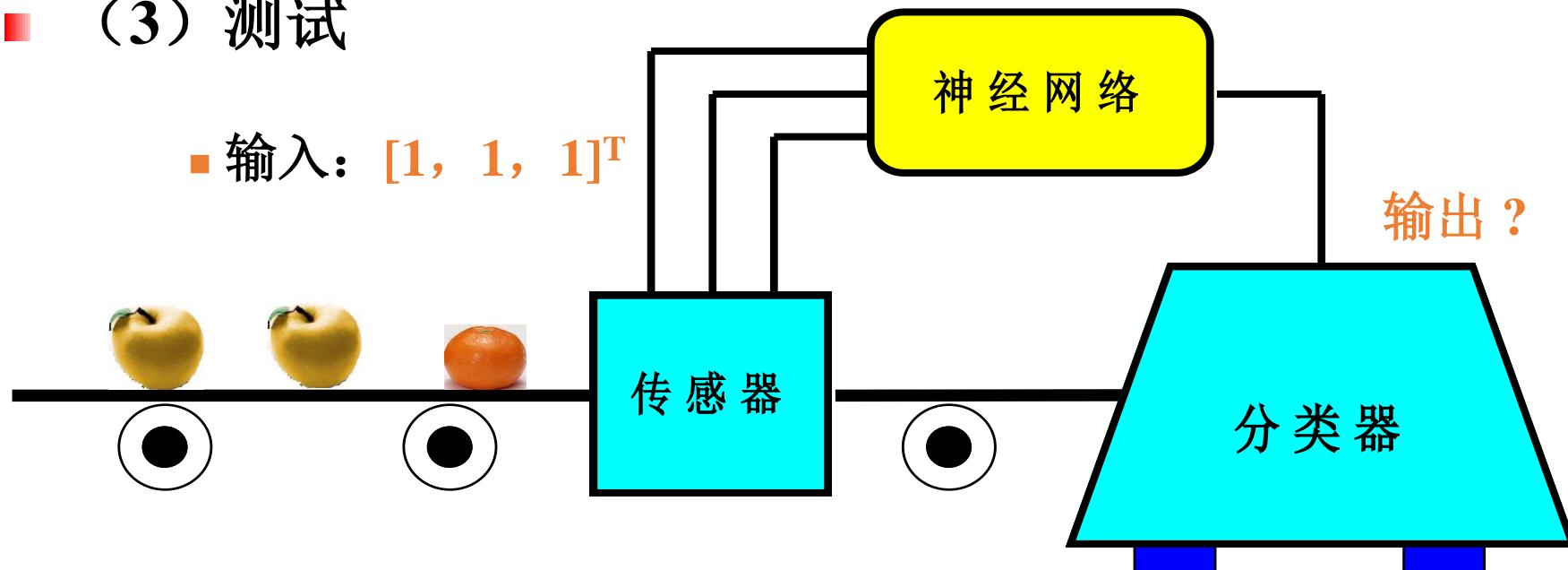
$$W = \begin{bmatrix} 0 & -2 & 2 \\ -2 & 0 & -2 \\ 2 & -2 & 0 \end{bmatrix}$$



Hopfield神经网络在联想记忆中的应用

■ (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: $x^{(2)} = [0, 1, 0]^T$

Hopfield神经网络在联想记忆中的应用

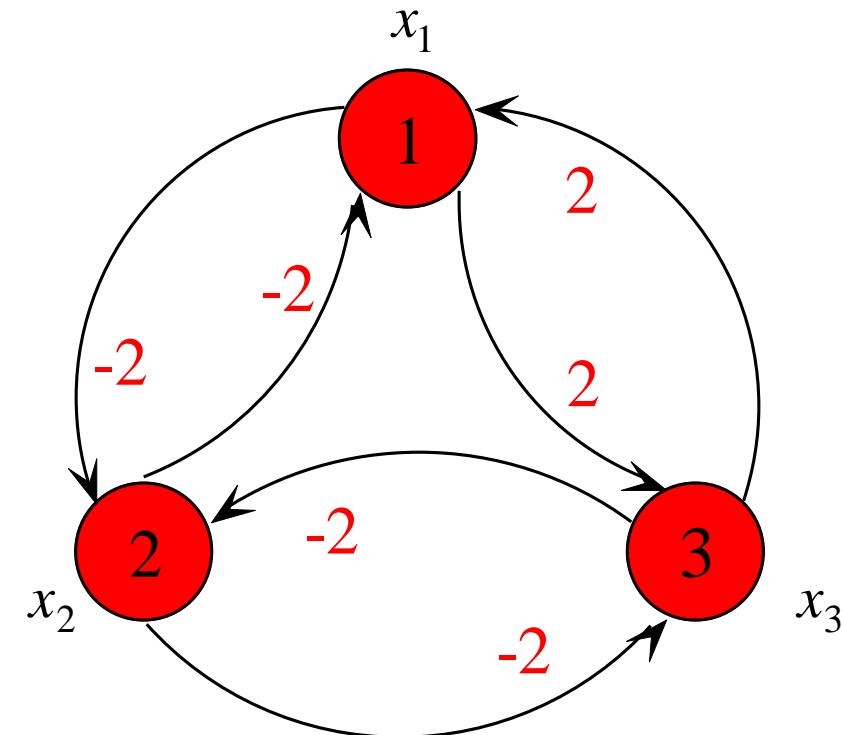
■ (3) 测试

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

- 测试用例: $[1, 1, 1]^T$

- 初始状态:
$$\begin{cases} x_1(0) = 1 \\ x_2(0) = 1 \\ x_3(0) = 1 \end{cases}$$

- 调整次序: $2 \rightarrow 1 \rightarrow 3$

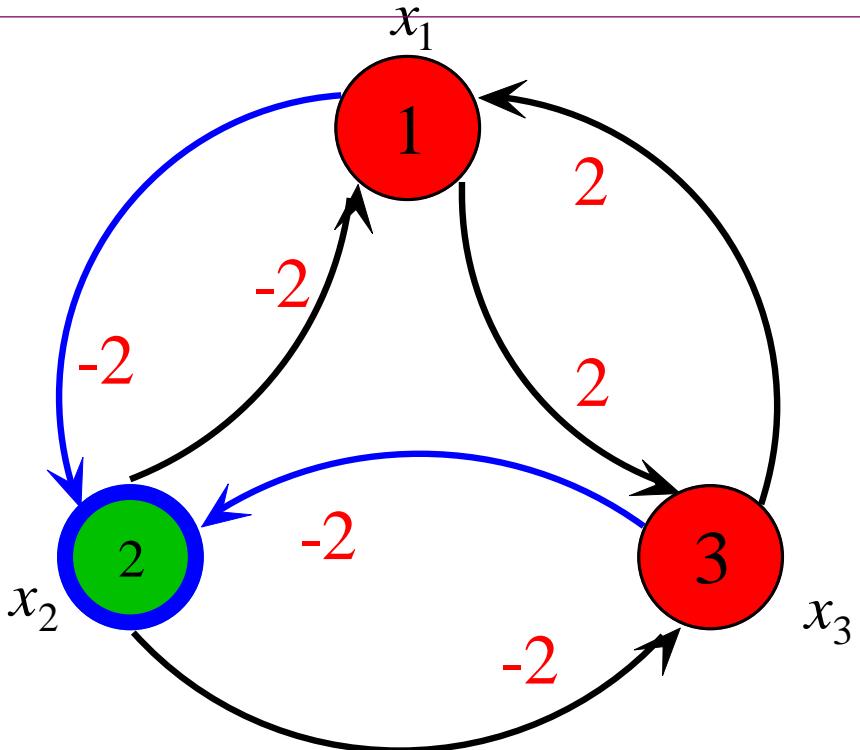


Hopfield神经网络在联想记忆中的应用

- 调整次序: $2 \rightarrow 1 \rightarrow 3$

$$k = 0$$

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



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

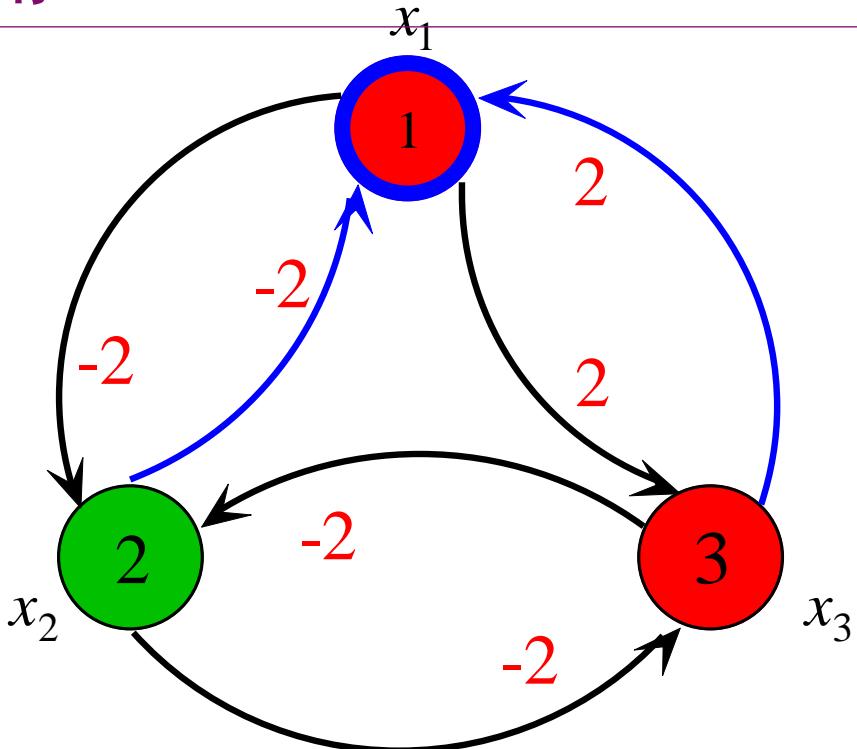
$$x_1(1) = x_1(0) = 1, \quad x_3(1) = x_3(0) = 1$$

Hopfield神经网络在联想记忆中的应用

- 调整次序: $2 \rightarrow 1 \rightarrow 3$

$k = 1$

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



$$x_1(2) = f\left(\sum_{j=1}^3 w_{1j} x_j(1)\right) = f(0 \times 1 + (-2) \times 0 + 2 \times 1) = f(2) = 1$$

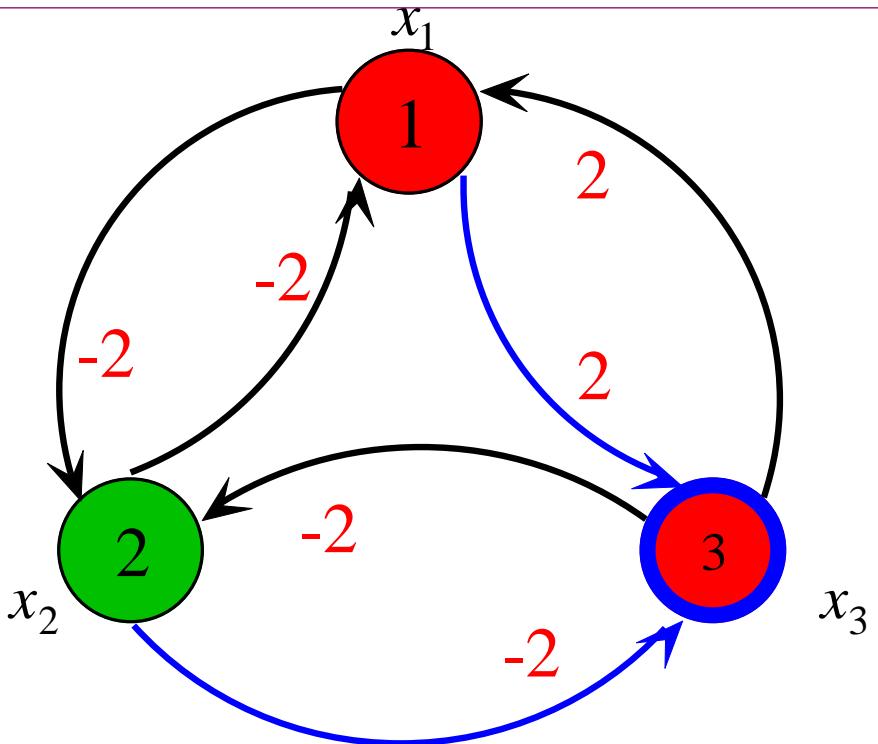
$$x_2(2) = x_2(1) = 0, \quad x_3(2) = x_3(1) = 1$$

Hopfield神经网络在联想记忆中的应用

- 调整次序: $2 \rightarrow 1 \rightarrow 3$

$$k = 2$$

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



$$x_3(3) = f\left(\sum_{j=1}^3 w_{3j} x_j(1)\right) = f(2 \times 1 + (-2) \times 0 + 0 \times 1) = f(2) = 1$$

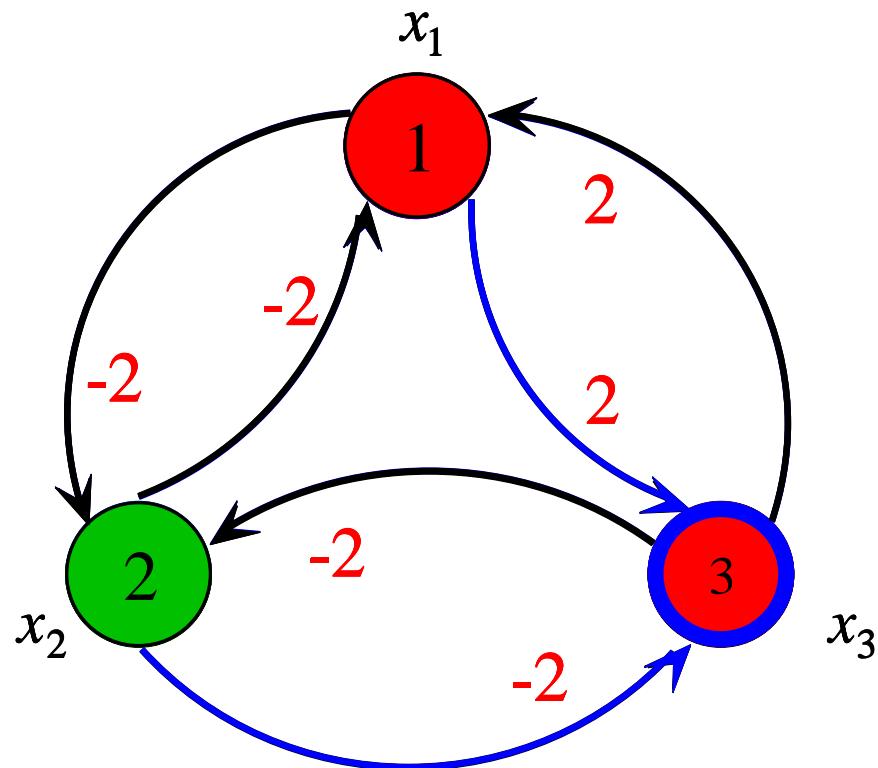
$$x_1(3) = x_1(2) = 1, \quad x_2(3) = x_2(2) = 0$$

Hopfield神经网络在联想记忆中的应用

- 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)



$$k = 0$$

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

$$k = 1$$

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

$$k = 2$$

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

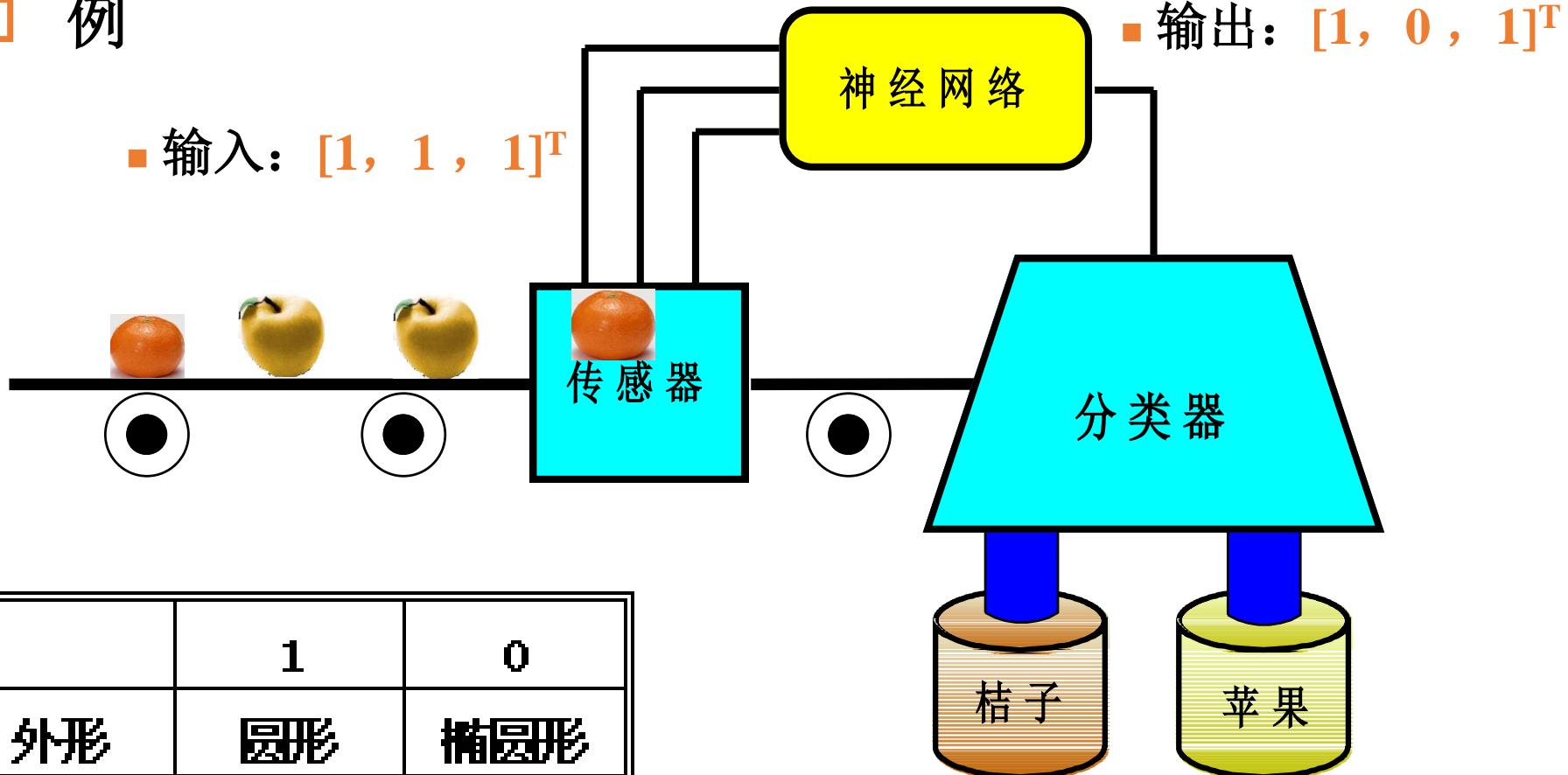
$$k = 3$$

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

Hopfield神经网络在联想记忆中的应用

□ 例

■ 输入: $[1, 1, 1]^T$



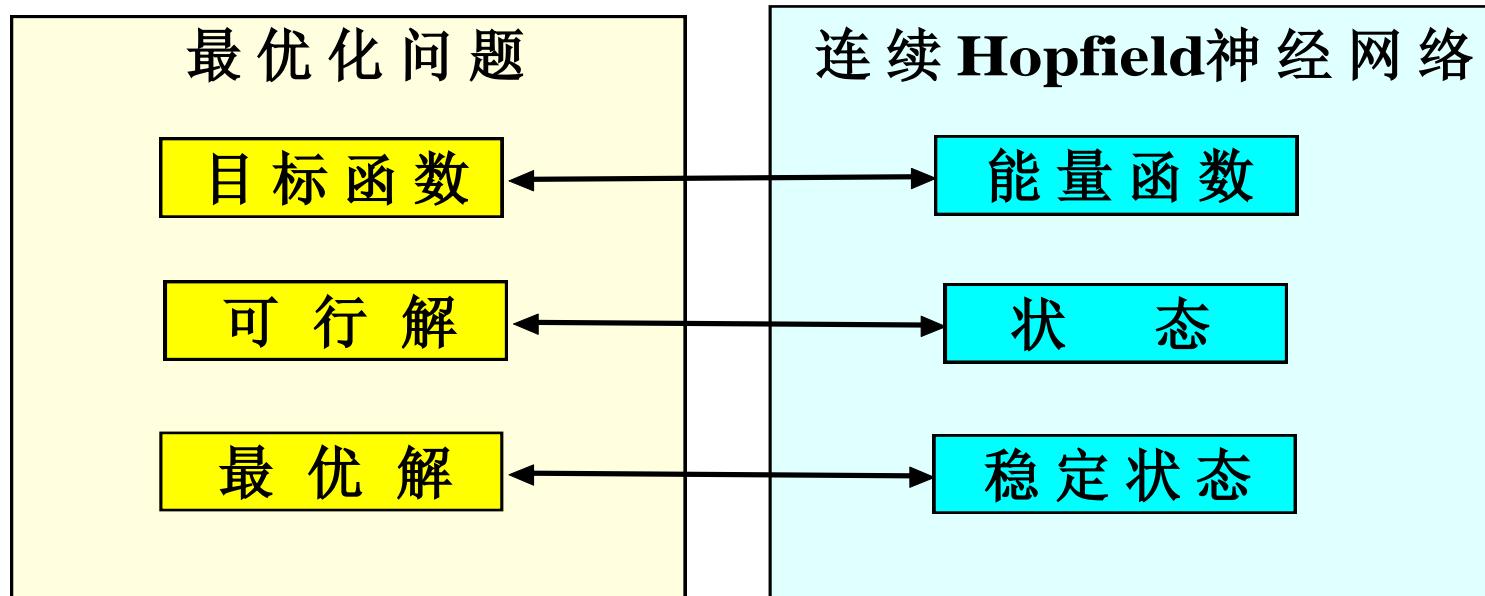
	1	0
外形	圆形	椭圆形
质地	光滑	粗糙
重量	< 1 磅	> 1 磅

标准桔子: $x^{(1)} = [1, 0, 1]^T$

标准苹果: $x^{(2)} = [0, 1, 0]^T$

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：



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。

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。
 - 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。

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

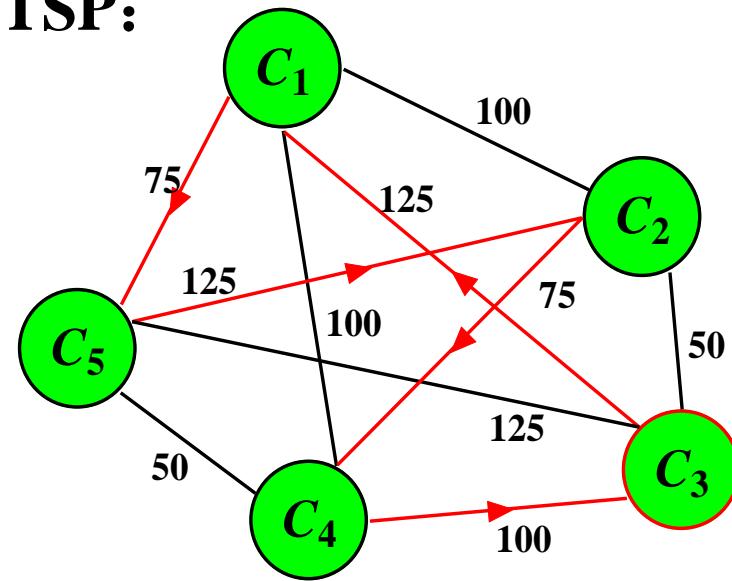
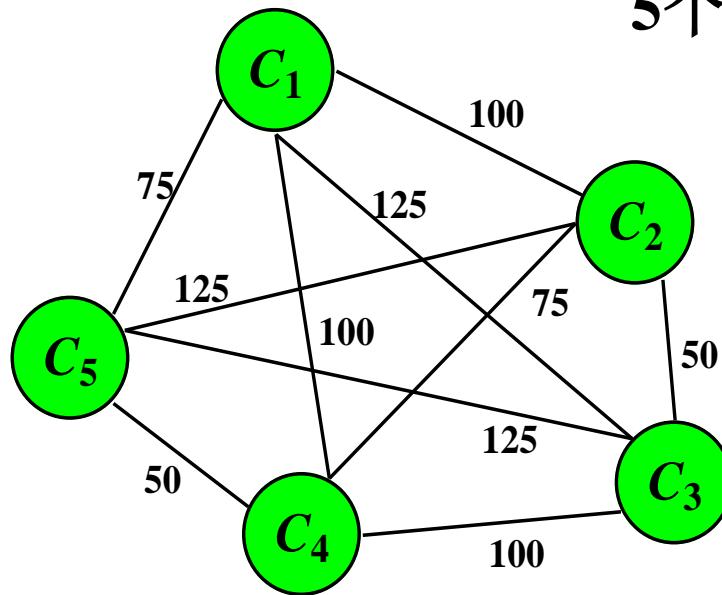
- 用穷举法，Cray计算机的计算速度： 10^8 次/秒。

城市数 (n)	7	15	20	50	100	200
搜索时间	2.5×10^{-5} 秒	1.8小时	350年	5×10^{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。

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

Hopfield神经网络优化方法

- TSP的描述：

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

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

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

- 用罚函数法，写出优化问题的目标函数：

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

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

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：

$$W_{xi,yj} = A\delta_{xy}(1-\delta_{ij}) - B\delta_{ij}(1-\delta_{xy}) - C - D\delta_{xy}(\delta_{j,i+1} + \delta_{j,i-1})$$

$$I_{xi} = -Cn$$

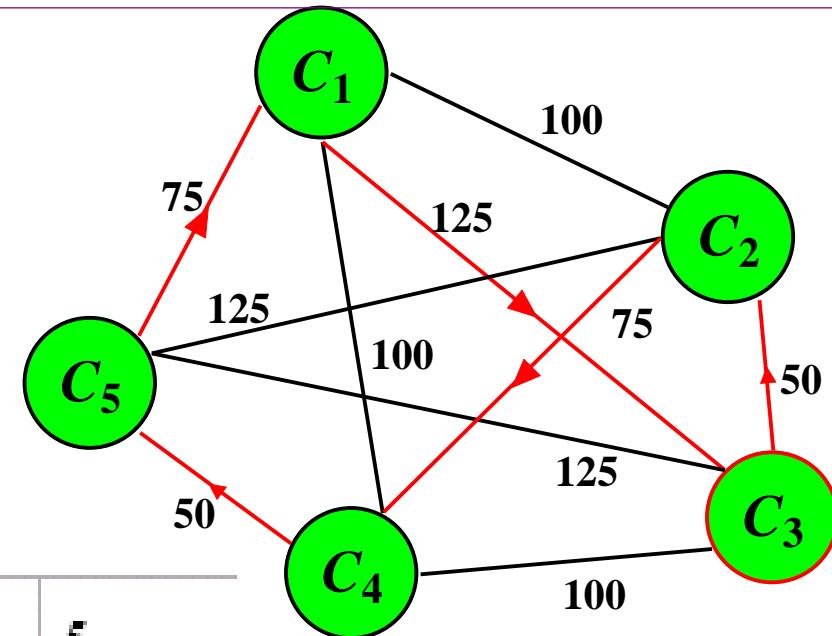
Hopfield神经网络优化方法

■神经网络的动态方程：

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

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

- 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。

- 作业车间调度问题
- JSP的Hopfield神经网络及其求解
- 作业车间生产调度举例
- 基于随机神经网络的生产调度方法

- 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。

Hopfield神经网络优化方法

- 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。

JSP的Hopfield神经网络及其求解

1. JSP的换位矩阵表示

(1,2,2): 作业 1 的工序 2
在机器 2 上执行。

“工序 (i, j, k) 不依赖于任何别的工序”的命题。

		0	1,1,1	1,2,2	2,2,1	2,1,2
1,1,1	1	0	0	0	0	0
1,2,2	0	0	0	0	0	1
2,2,1	0	0	1	0	0	0
2,1,2	1	0	0	0	0	0

“工序 $(2,2,1)$ 依赖于另一工序 $(1,2,2)$ ”的命题成立。

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 m 。即在 $t=0$ 时，在**0**时刻启动的工序数应为 n 。 $n \leq m$
- (4) 在同一时刻启动的同 m 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。

JSP的Hopfield神经网络及其求解

2. JSP计算能量函数

$$E = \frac{A}{2} \sum_{x=1}^{mn} \sum_{\substack{i=1 \\ j \neq i}}^{mn+1} v_{xi} v_{xj} + \frac{B}{2} \left(\sum_{x=1}^{mn} \sum_{i=1}^{mn+1} v_{xi} - mn \right)^2 + \frac{C}{2} \sum_{i=2}^{mn+1} \sum_{x=1}^{mn} v_{xi} v_{i-1,x+1}$$
$$+ \frac{D_1}{2} \left(\sum_{x=1}^{mn} v_{xi} \right) \boxed{\text{行约束}}$$
$$+ \frac{D_2}{2} \sum_{k_1=1}^n \sum_{\substack{k_2=1 \\ k_3=1 \\ k_3 \neq k_2}}^m v_{k_1+k_2 m, i} v_{k_1+(k_3-1)m, i} \boxed{\text{全局约束}}$$
$$+ \frac{D_3}{2} \sum_{k_1=1}^m \sum_{k_2=1}^n \sum_{k_3=1}^n v_{k_1+(k_2-1)m, i} v_{k_1+(k_3-1)m, i} \boxed{\text{列约束}}$$

v_{xi} : 与矩阵中 (x, i) 位置相对应的神经元的输出状态。

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 $w_{xi,yj}$

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

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

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

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

- 神经元 (x, i) 的偏置电流 I_{xi} : $I_{xi} = Bmn + D_1 m \delta_{i1}$

JSP的Hopfield神经网络及其求解

4. Hopfield神经网络的运动方程

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

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 S_{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。

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。

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

所有的操作：

111, 122, 133,
213, 221, 232。

加工时间分配

作业	工序		
	1	2	3
1	5	8	2
2	7	3	9

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

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

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

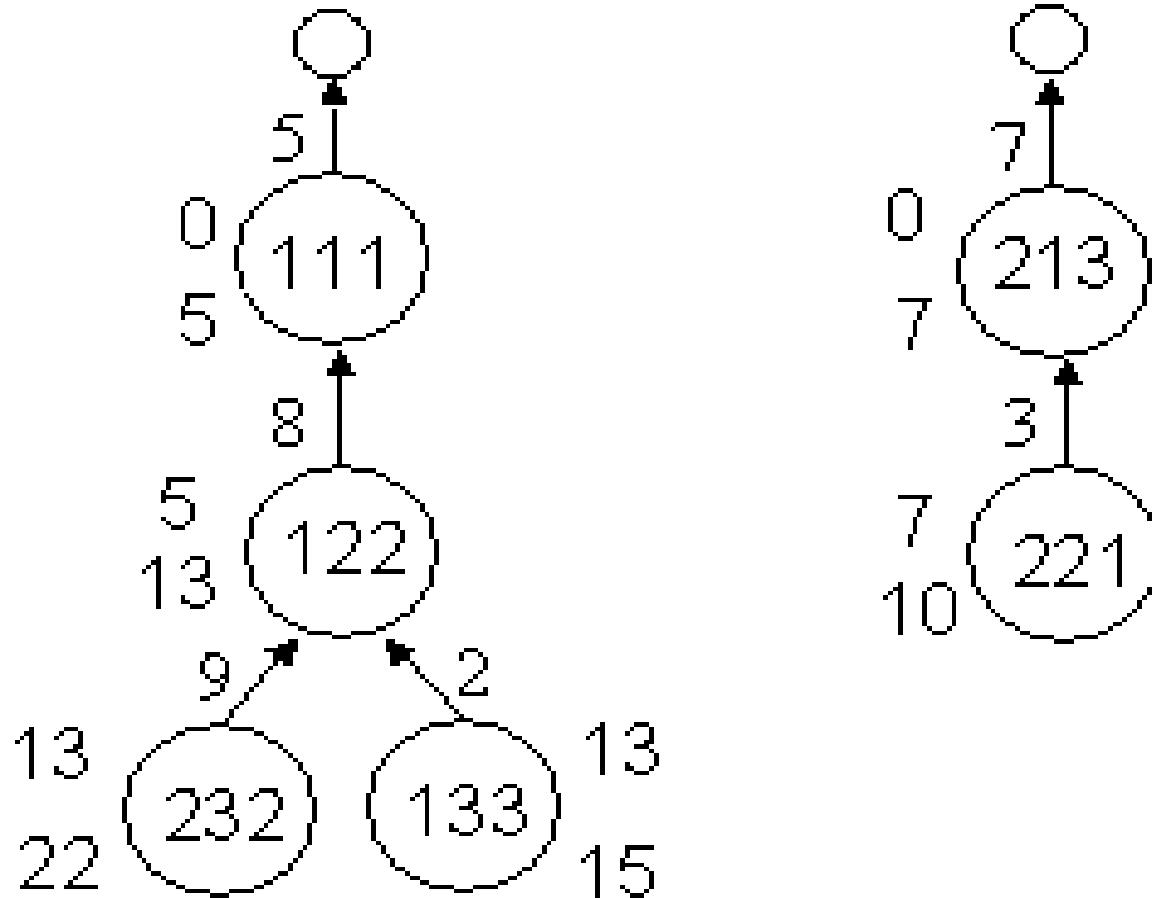
作业车间生产调度举例

□ 计算能量函数为0的换位矩阵

	0	111	122	133	221	232	213
111	1	0	0	0	0	0	0
122	0	1	0	0	0	0	0
133	0	0	1	0	0	0	0
221	0	0	0	0	0	0	1
232	0	0	1	0	0	0	0
213	1	0	0	0	0	0	0

作业车间生产调度举例

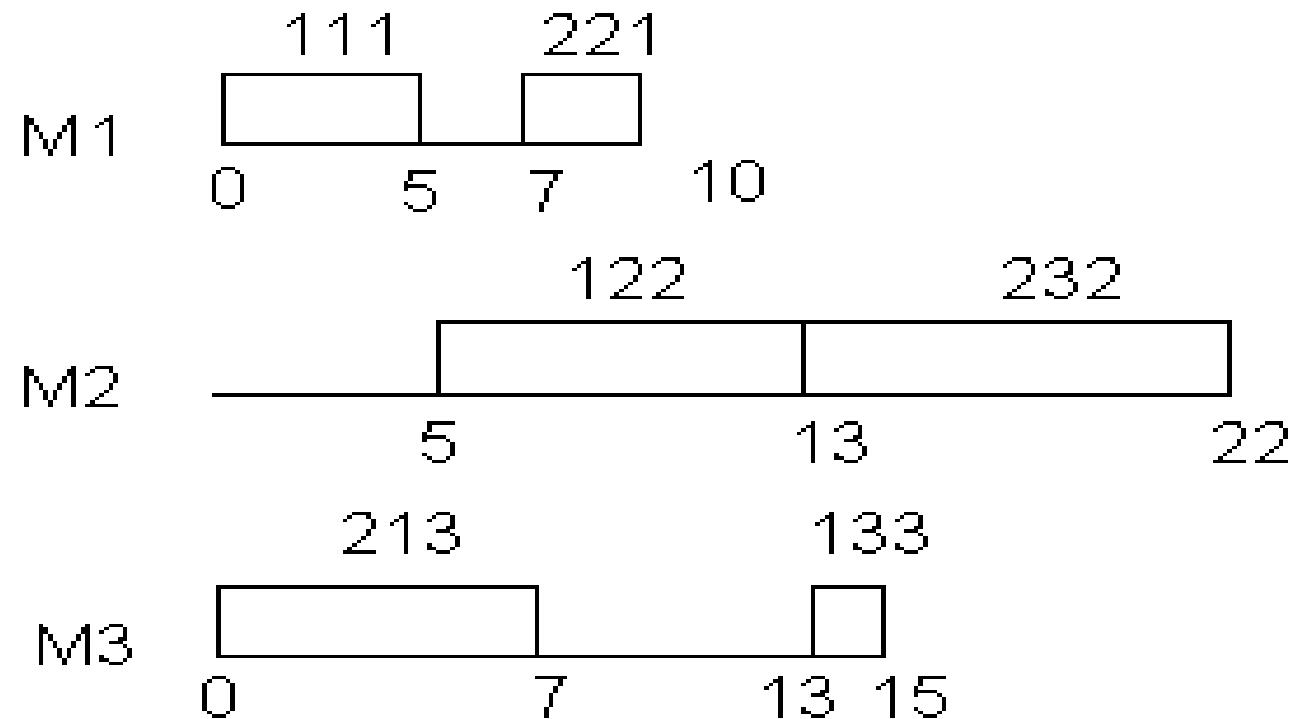
□ 成本树



返回

作业车间生产调度举例

□ 甘特图



返回

□ 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。

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 $T_0 \leftarrow T_{\max}$ ，合适的输入偏置电流，凝结温度 T_{\min} ，温度下降速率 k ，在每个温度点的循环处理次数 r 。

(2) 随机爬山：

对每个神经元 i ，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 Δcost_i 。

基于随机神经网络的生产调度方法

- 若 $\Delta \cos t_i < 0$ ，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 $\Delta E_i \leftarrow \Delta E_i + \Delta \cos t_i$

否则，令 $\Delta E_i \leftarrow \Delta E_i - \Delta \cos t_i$

- 计算概率

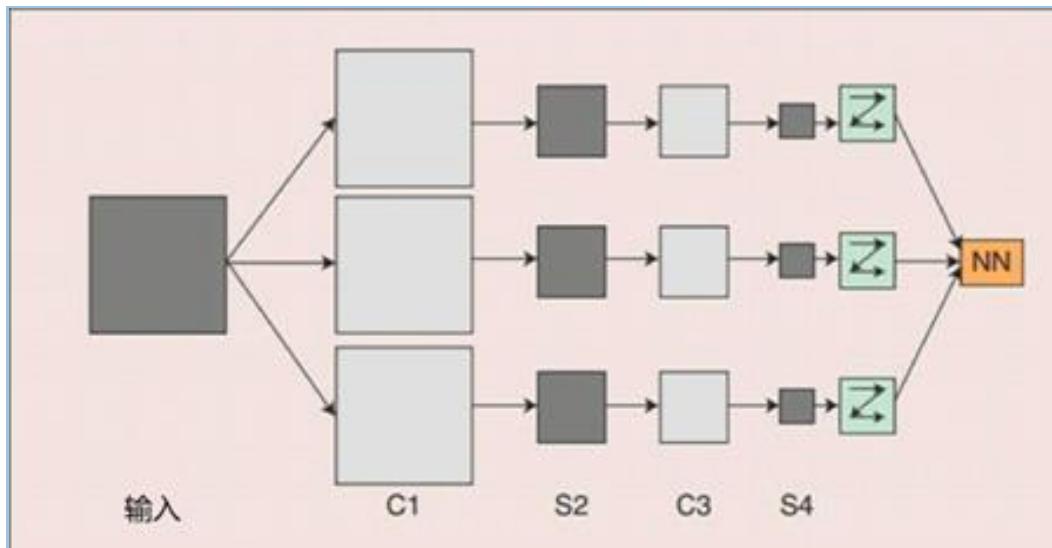
$$p_i = \frac{1}{1 + \exp(-\frac{\Delta E_i}{T})}$$

□ 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的响应，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。

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。这些映射图再进过滤波得到**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。

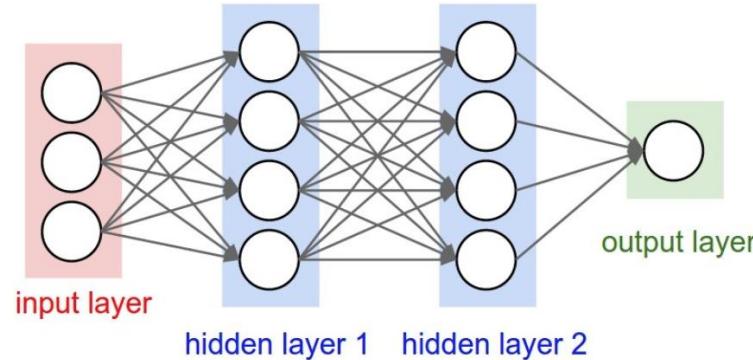
卷积神经网络的结构

CNN与传统神经网络对比

- 1、神经网络的输入是一个向量
- 2、神经元一维排列
- 3、由输入层、隐层、输出层组成
- 4、同层神经元之间不连接，每个神经元与前一层中所有神经元全连接。

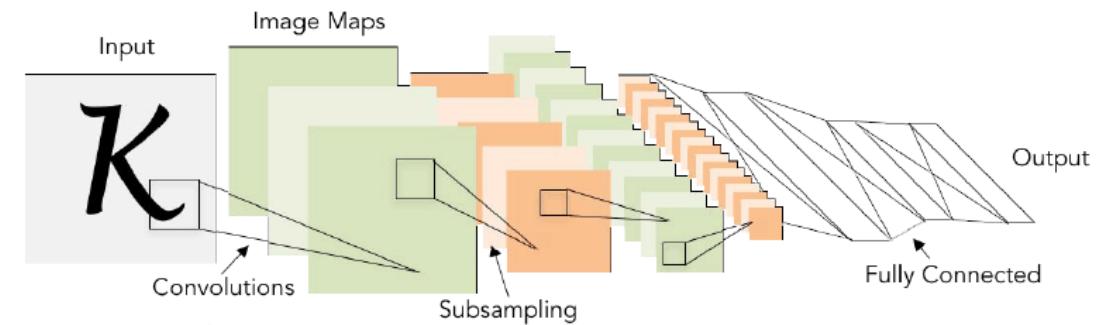
不同

- 1、CNN的输入是图像
- 2、神经元三维排列
- 3、由输入层、卷积层、池化层、激活层、全连接层、输出层组成
- 4、卷积层的每个神经元与前一层以感受野的形式连接，用卷积核大小的窗口依次划过前一层的图层。



相同

- 1、都由神经元组成，神经元中有具有学习能力的权重和偏差
- 2、每个神经元都得到一些输入数据，进行内积运算和激活函数运算
- 3、整个网络是可导的评分函数，前向传播和优化算法同理。





卷积神经网络的结构

卷积层

① 卷积层的特性

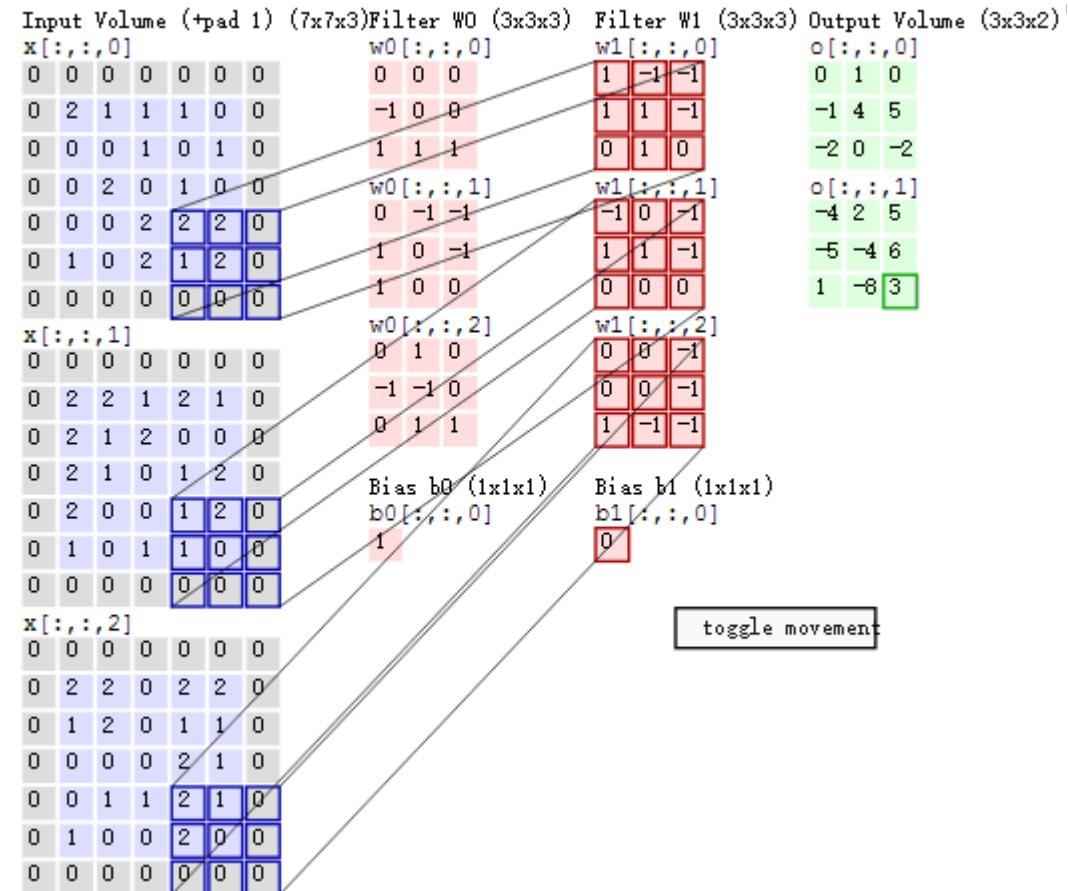
- 局部连接（区别于全连接）
- 参数共享
- 优点：**减少参数量，节约计算成本**

② 滤波器的3个超参数控制输出数据体的尺寸

- ① **输出数据体的深度：**与使用的滤波器数量一致。直观上理解为深度维度上不同神经元将可能被不同方向的边界或颜色斑点激活，因此输入图像的尺寸越大或包含特征种类越多时需设置更多数量的滤波器。
- ② **步长：**滑动滤波器时每一步跨越的像素个数。此操作使得输出数据体相对输入数据尺寸变小。
- ③ **零填充：**将输入数据体用0在边缘处进行填充。一、当卷积核尺寸与步长确定情况下，不能整数步滑过输入数据时，使用零填充；二、保持输入输出数据体尺寸一致

卷积神经网络的结构

例2 卷积层的运行演示



蓝色是输入数据体
红色是权重数据体
绿色是输出数据体

卷积神经网络的结构

池化层

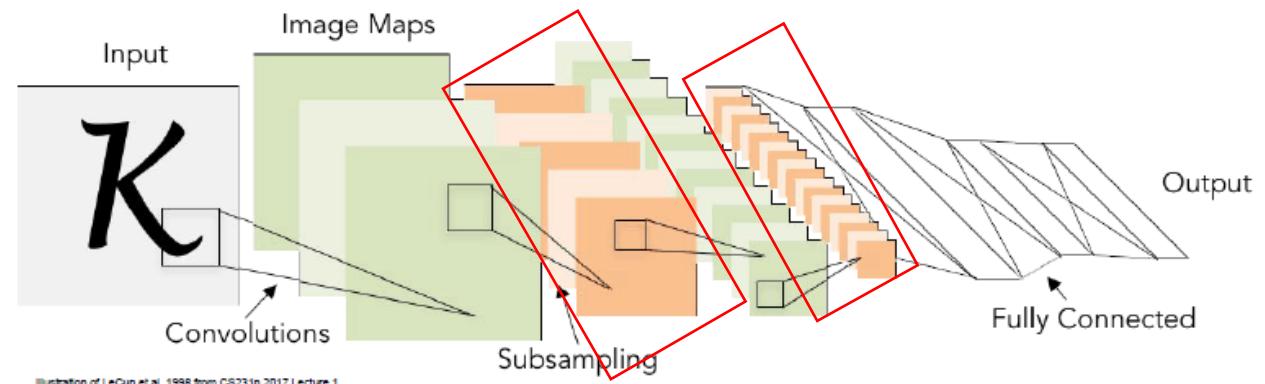
- ① 作用：减少数据量、减轻过拟合
- ② 特点：无参数运算，一旦选定池化滤波器尺寸、步长及最大/平均池化，这一层输入数据发生的改变就固定。
- ③ 涉及的尺寸计算：

- 输入数据体尺寸 W_1 、 H_1 、 D_1
- 两个超参数 滤波器大小 F 、步长 S
- 输出数据体尺寸 W_2 、 H_2 、 D_2

其中 $W_2 = (W_1 - F) / S + 1$

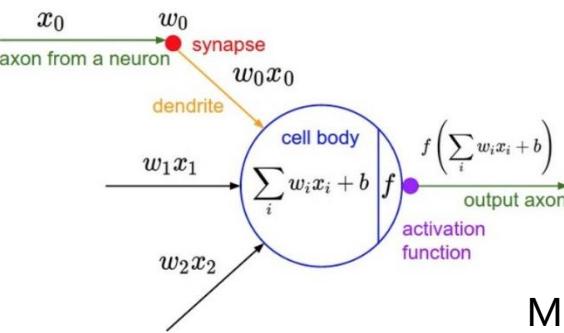
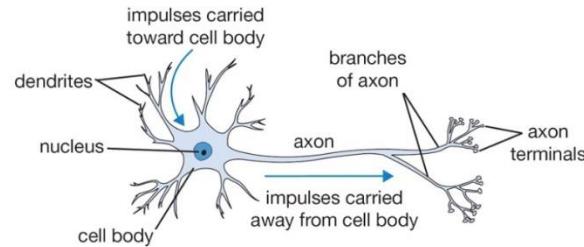
$H_2 = (H_1 - F) / S + 1$

$D_2 = D_1$



卷积神经网络的结构

激活层



M-P神经元模型

① 非线性拟合作用，使网络模型拟合出的特征更复杂

② 常用的激活函数：

f : 激活函数

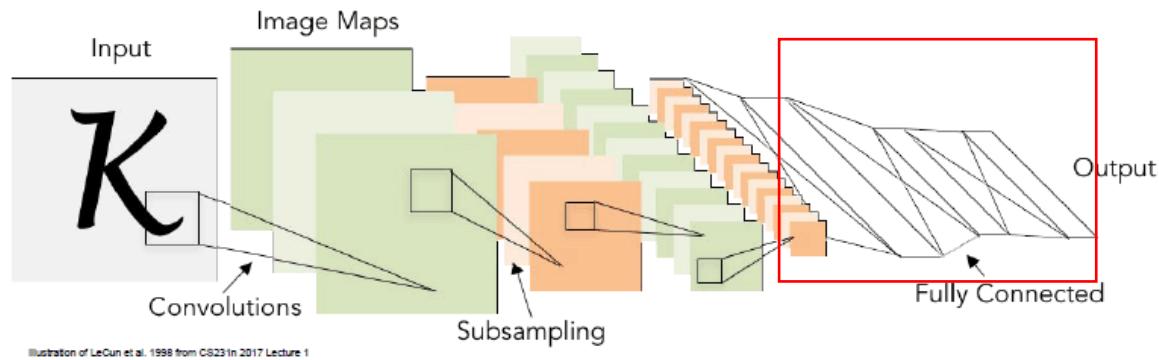
$$y_j = f\left(\sum_{i=1}^n x_i w_{ij} - \theta\right)$$

Name	Functions	Derivatives	Figure
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))^2$	
tanh	$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	
ReLU	$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Leaky ReLU	$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0. \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0. \end{cases}$	
Softmax	$f(x) = \frac{e^x}{\sum_i e^x}$	$f'(x) = \frac{e^x}{\sum_i e^x} - \frac{(e^x)^2}{(\sum_i e^x)^2}$	

Table 1: Non-linear activation function

卷积神经网络的结构

全连接层



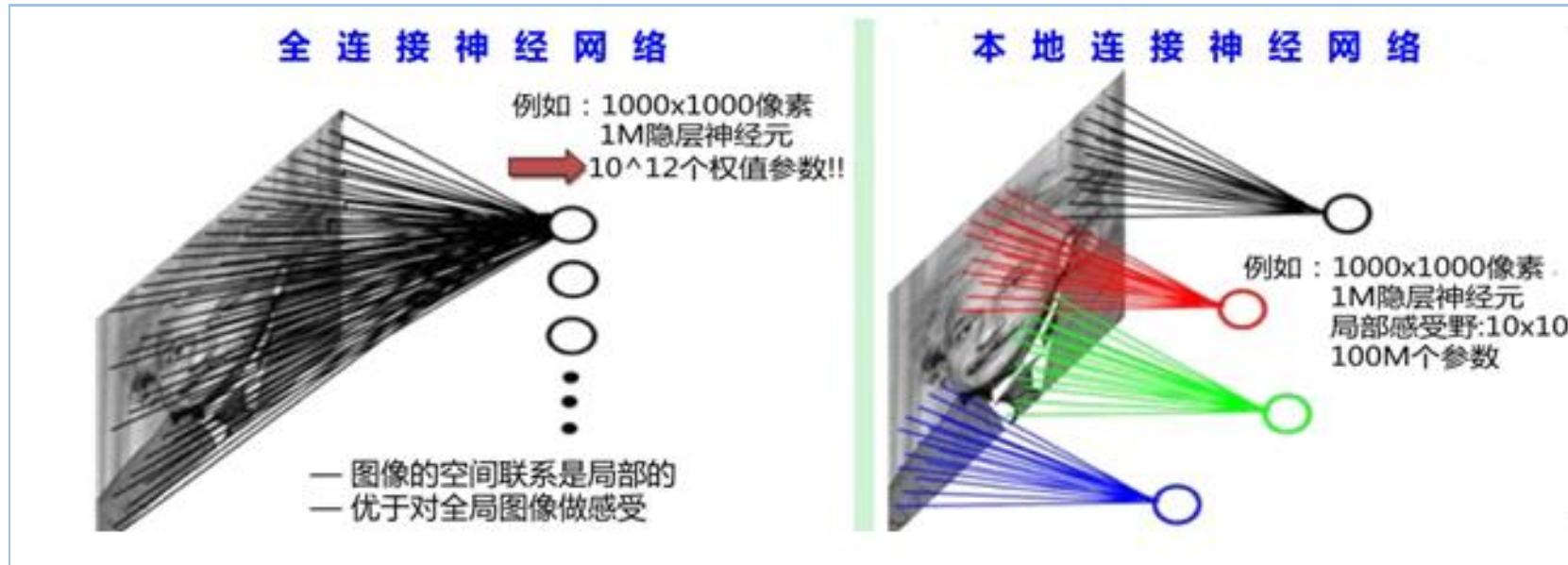
- ④ 连接方式：该层神经元对前一层中所有数据是全部连接的。
- ④ 最后一层全连接中，在神经元的输出端使用合适的损失函数，就能让单个神经元变成一个线性分类器。
即最后一层全连接层的作用为分类器

卷积神经网络的结构

□ 卷积神经网络的**4个关键技术**:

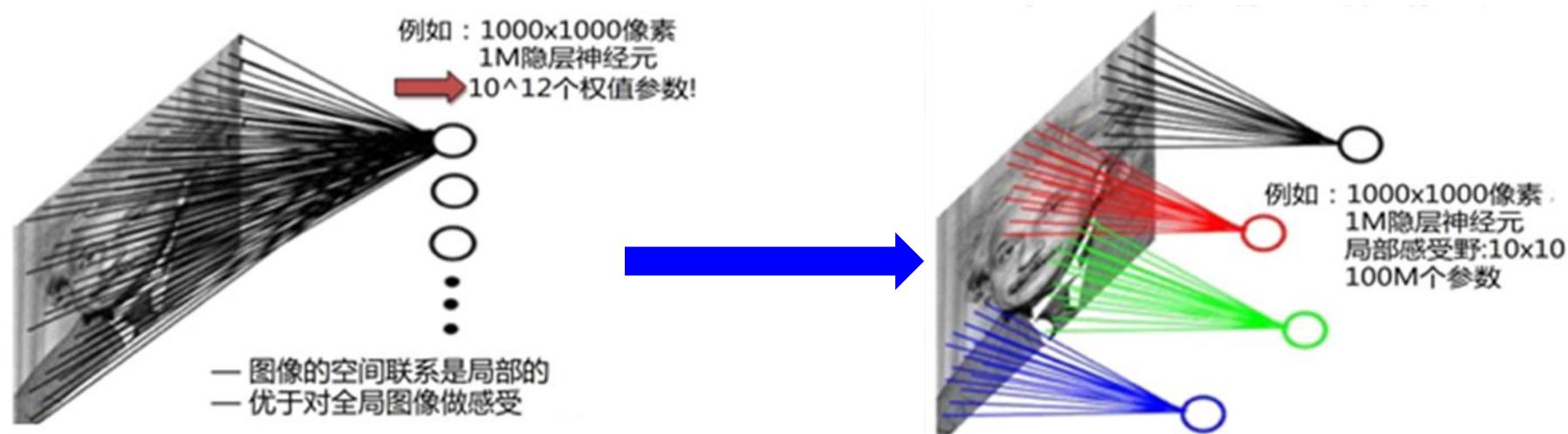
- 局部连接 (**8.7.2**)
- 权值共享 (**8.7.3**)
- 多卷积核 (**8.7.4**)
- 池化 (**8.7.5**)

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。

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 10×10 个像素连接，也就是说每一个神经元存在 $10 \times 10 = 100$ 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。

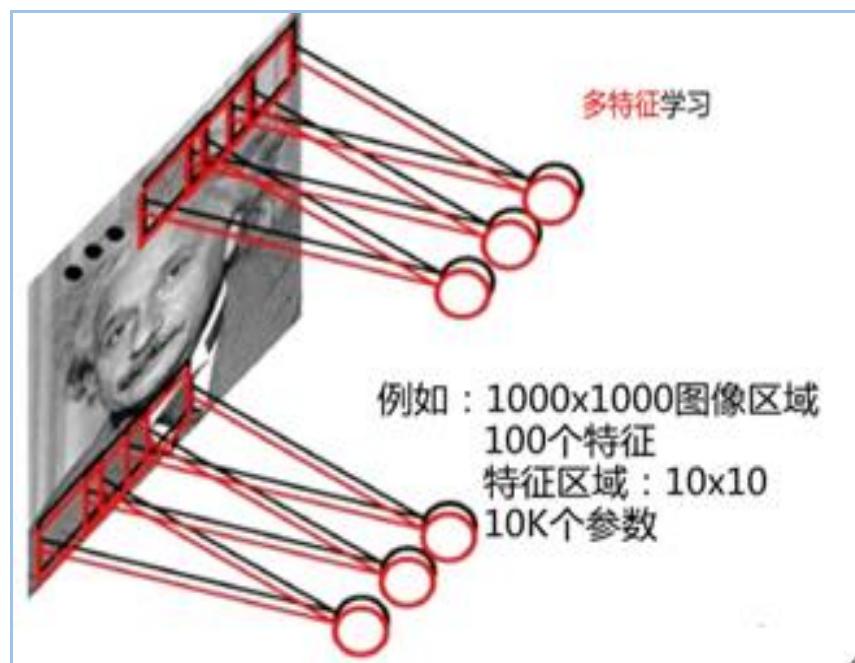
卷积神经网络的权值共享

□ 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。

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。



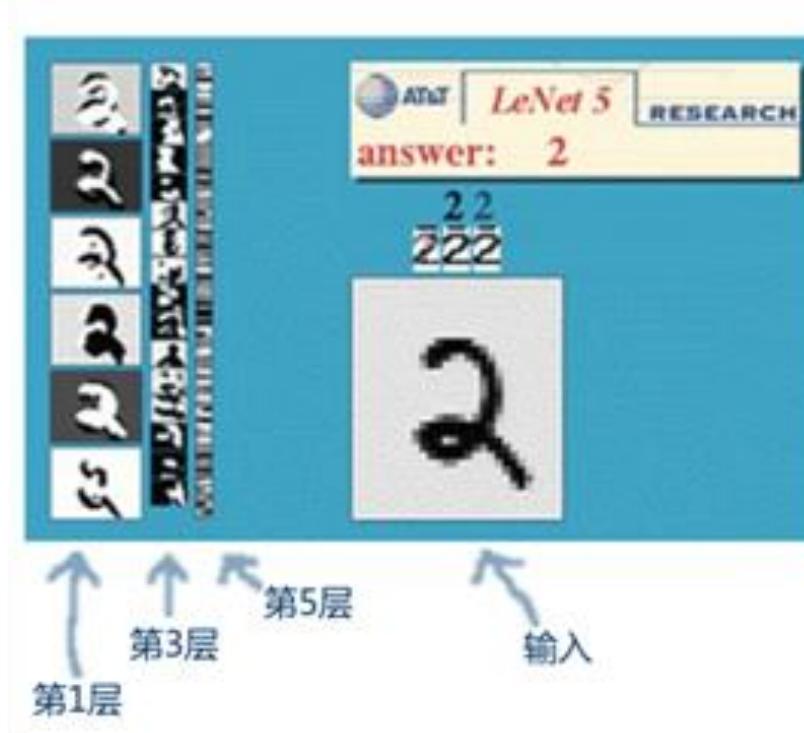
- ✓ 每层隐层神经元的个数按卷积核的数量翻倍。
- ✓ 每层隐层参数个数仅与特征区域大小、卷积核的多少有关。
例如：隐含层的每个神经元都连接**10x10**像素图像区域，同时有**100**种卷积核（滤波器）。则参数总个数为：**(10x10+1) x 100=10100**个

卷积神经网络的实现应用

- 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。

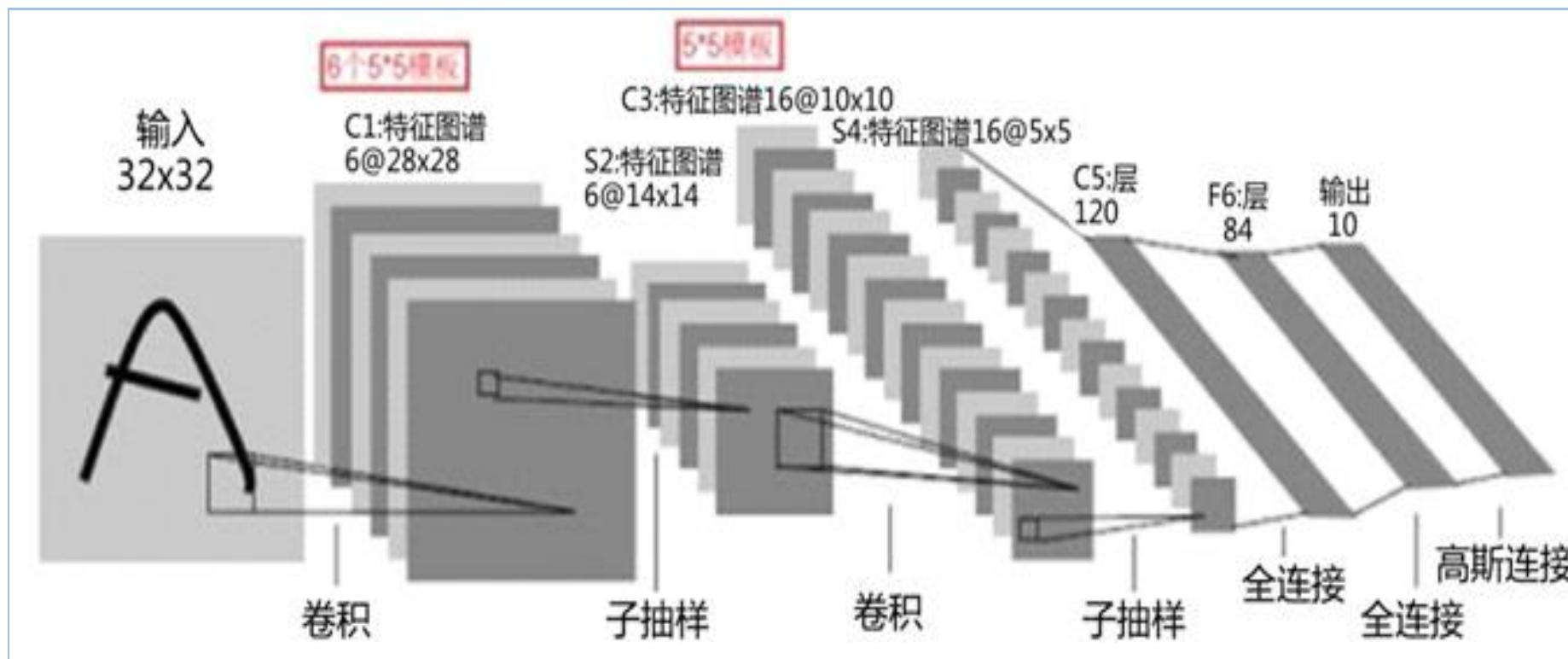
CNN在手写数字识别中的应用

- 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。



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 大小。



□ 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。

□ 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**。

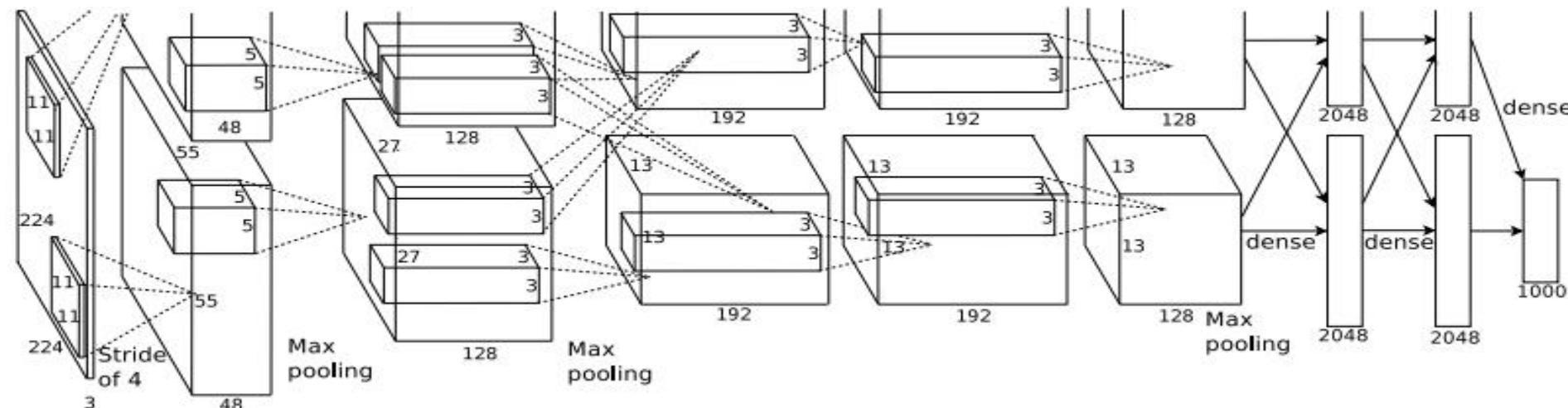
深度学习卷积神经网络

层的排列规律

- 卷积神经网络最常见的形式就是将一些卷积层和激活层放在一起，其后紧跟池化层，重复如此直到图像在空间上被缩小到一个足够小的尺寸，然后过渡到全连接层。最后的全连接层得到输出，比如分类评分。

案例学习

AlexNet: 具有历史意义的一个网络结构,2012年ImageNet图像分类竞赛中, top-5错误率比上一年的冠军下降了十个百分点,而且远远超过当年的第二名,也正因此,深度学习沉寂多年后重回历史舞台。

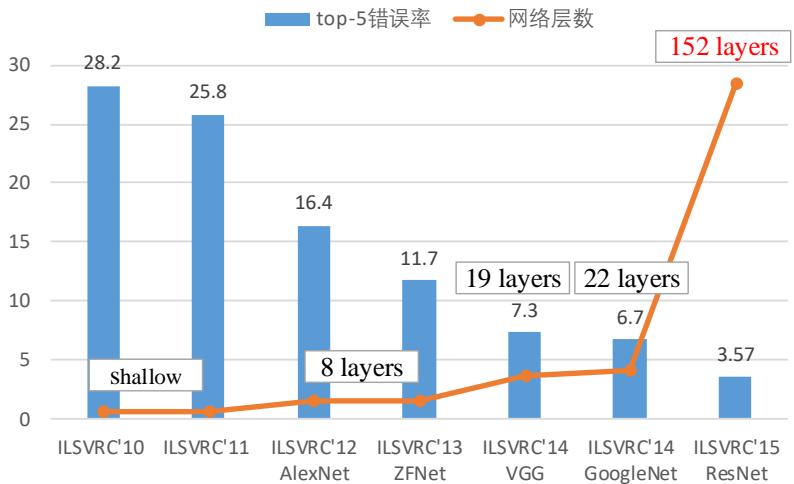


两个GPU并行计算
(卷积+池化)*2+卷积*3+池化*1+全连接*3

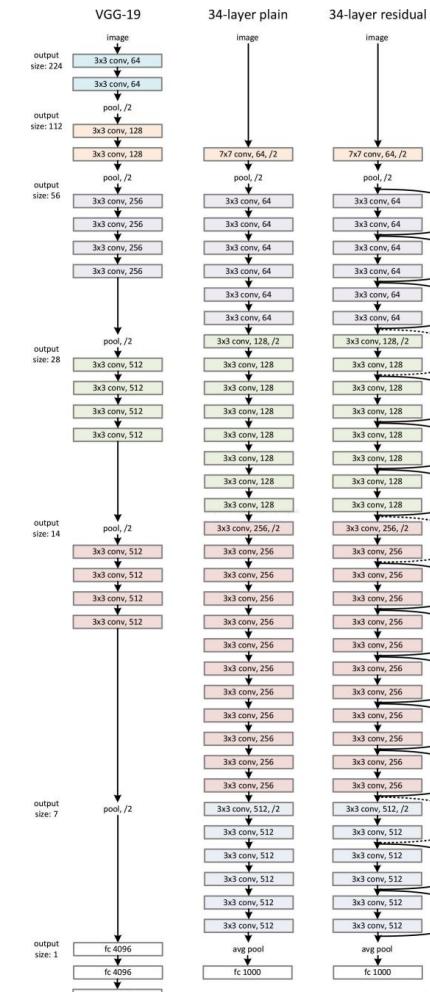
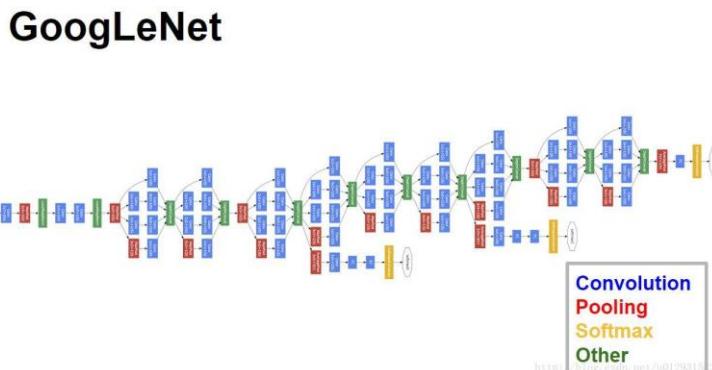
深度学习卷积神经网络

层的排列规律

历届ILSVRC的优秀网络结构



152 layers



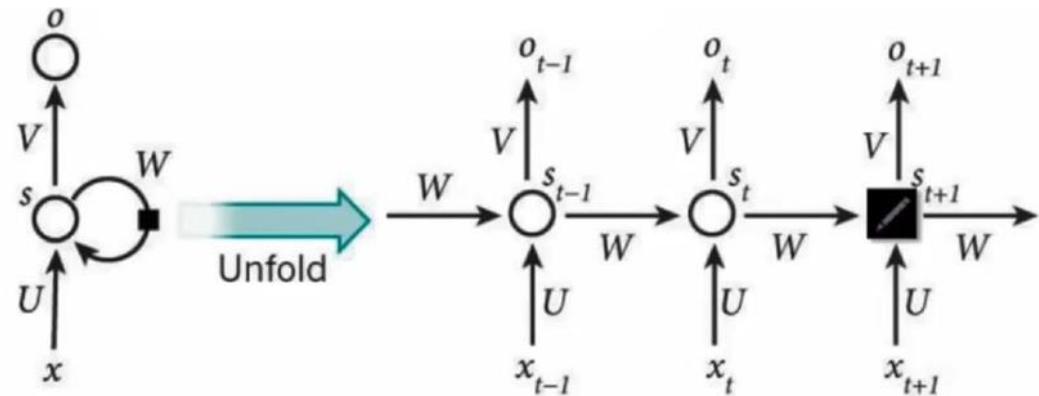
RNN

RNN是一类以在网络中出现环为特点的网络模型，并且能存储着神经元这一时刻的输入与前一时刻输出的依赖关系。

信号从一个神经元到另一个神经元。并不会马上消失，而是继续存活。

核心思想：

样本间存在顺序关系，每个样本和它之前的样本存在关联。
通过神经网络在时序上的展开，能偶找到样本之间的序列相关性。



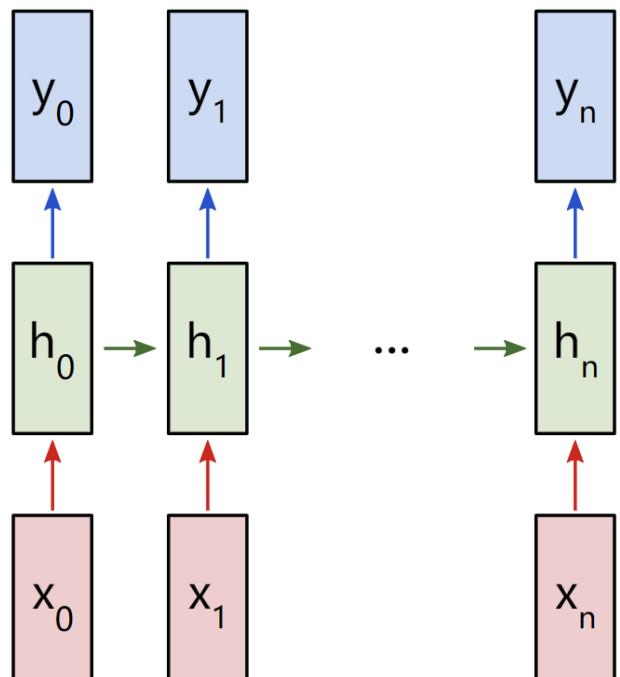
x_t 是时间t处的输入；

s_t 是时间t处的“记忆”， $s_t = f(Ux_t + Ws_{t-1})$ ；

o_t 是时间t处的输出，比如是预测下个词的话，可能是softmax输出的属于每个候选词的概率， $o_t = softmax(Vs_t)$ ；

可以把隐状态 s_t 视作“记忆体”，捕捉了之前时间点的信息；
输出 o_t 由当前时间及之前所有的“记忆”共同计算得到。

RNN



A many to many RNN

- W_{xh} , used for all $x_t \rightarrow h_t$ links.
- W_{hh} , used for all $h_{t-1} \rightarrow h_t$ links.
- W_{hy} , used for all $h_t \rightarrow y_t$ links.

We'll also use two biases for our RNN:

- b_h , added when calculating h_t .
- b_y , added when calculating y_t .

$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = W_{hy}h_t + b_y$$

RNN

Backward Phase

$$\frac{\partial L}{\partial W_{xh}} = \frac{\partial L}{\partial y} \sum_t \frac{\partial y}{\partial h_t} * \frac{\partial h_t}{\partial W_{xh}}$$
$$h_t = \tanh(W_{xh}x_t + W_{hh}h_{t-1} + b_h)$$
$$\frac{\partial h_t}{\partial W_{xh}} = (1 - h_t^2)x_t$$
$$\begin{aligned}\frac{\partial y}{\partial h_t} &= \frac{\partial y}{\partial h_{t+1}} * \frac{\partial h_{t+1}}{\partial h_t} \\ &= \frac{\partial y}{\partial h_{t+1}} (1 - h_t^2)W_{hh}\end{aligned}$$

$$\frac{d \tanh(x)}{dx} = 1 - \tanh^2(x)$$

$$L = -\ln(p_c) = -\ln(\text{softmax}(y_c))$$

$$\frac{\partial L}{\partial y_i} = \begin{cases} p_i & \text{if } i \neq c \\ p_i - 1 & \text{if } i = c \end{cases}$$

For example, if we have $p = [0.2, 0.2, 0.6]$ and the correct class is $c = 0$, then we'd get $\frac{\partial L}{\partial y} = [-0.8, 0.2, 0.6]$. This is also quite easy to turn into code:

RNN的应用

- Language Modeling and Generating Text
- Machine Translation
- Speech recognition
- Generating Image Description

RNN应用

Language modeling and generating text

- You can train a RNN-LM on any kind of text, then generate text in that style.
- RNN-LM trained on *Harry Potter*:



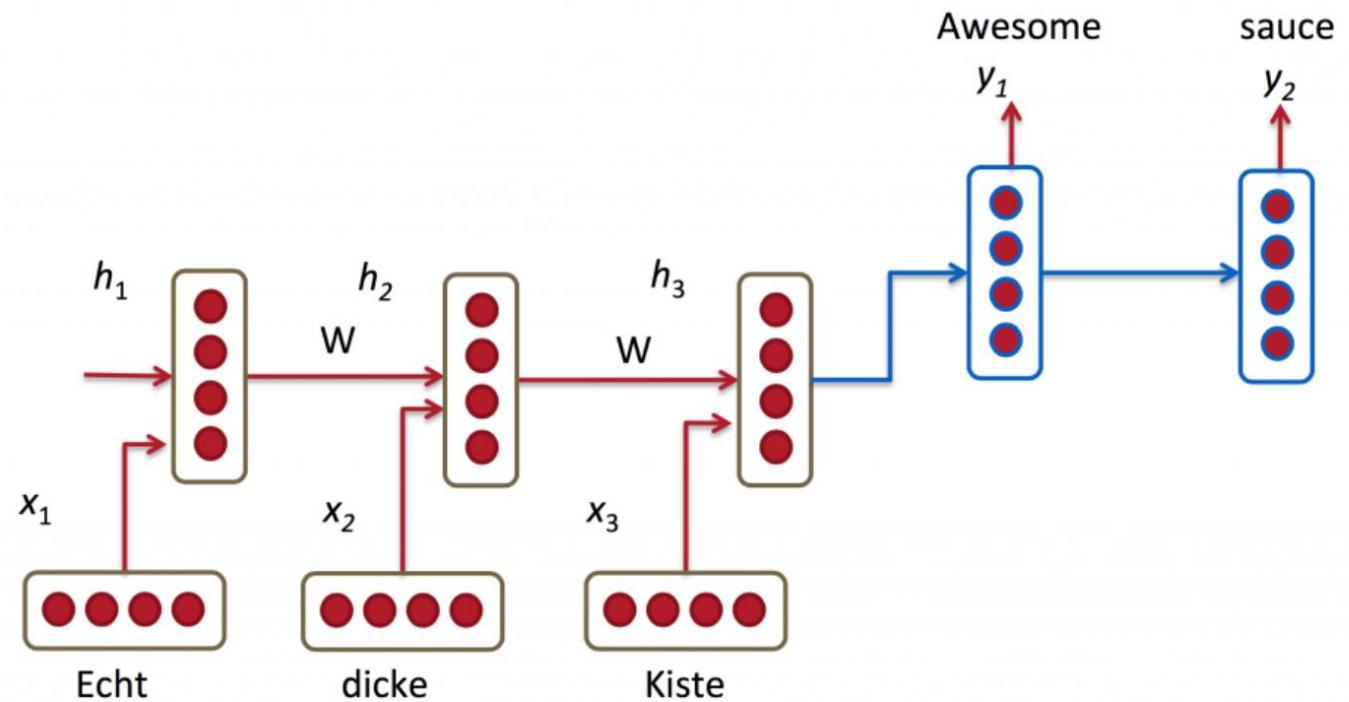
“Sorry,” Harry shouted, panicking—“I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

Source: <https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6>

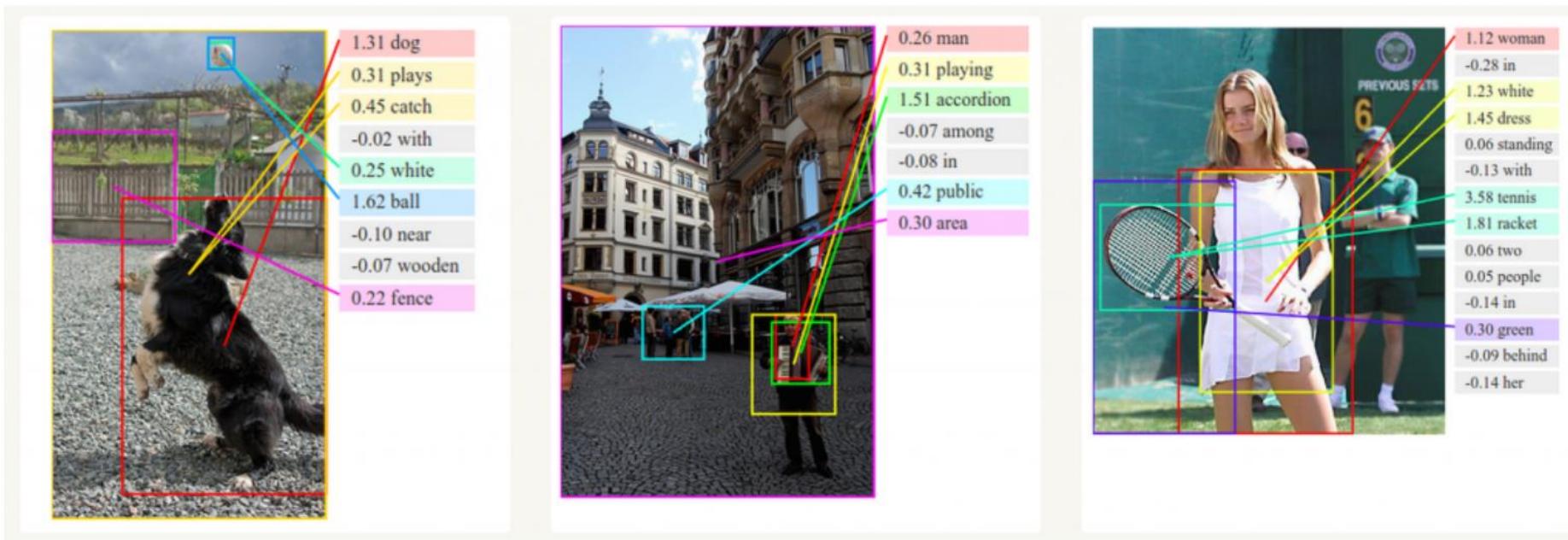
RNN应用

Machine Translation



RNN应用

Generating Image Description



Deep Visual-Semantic Alignments for Generating Image Descriptions. Source:

<http://cs.stanford.edu/people/karpathy/deepimagesent/>



RNN存在的问题

Gradient vanishing and exploding

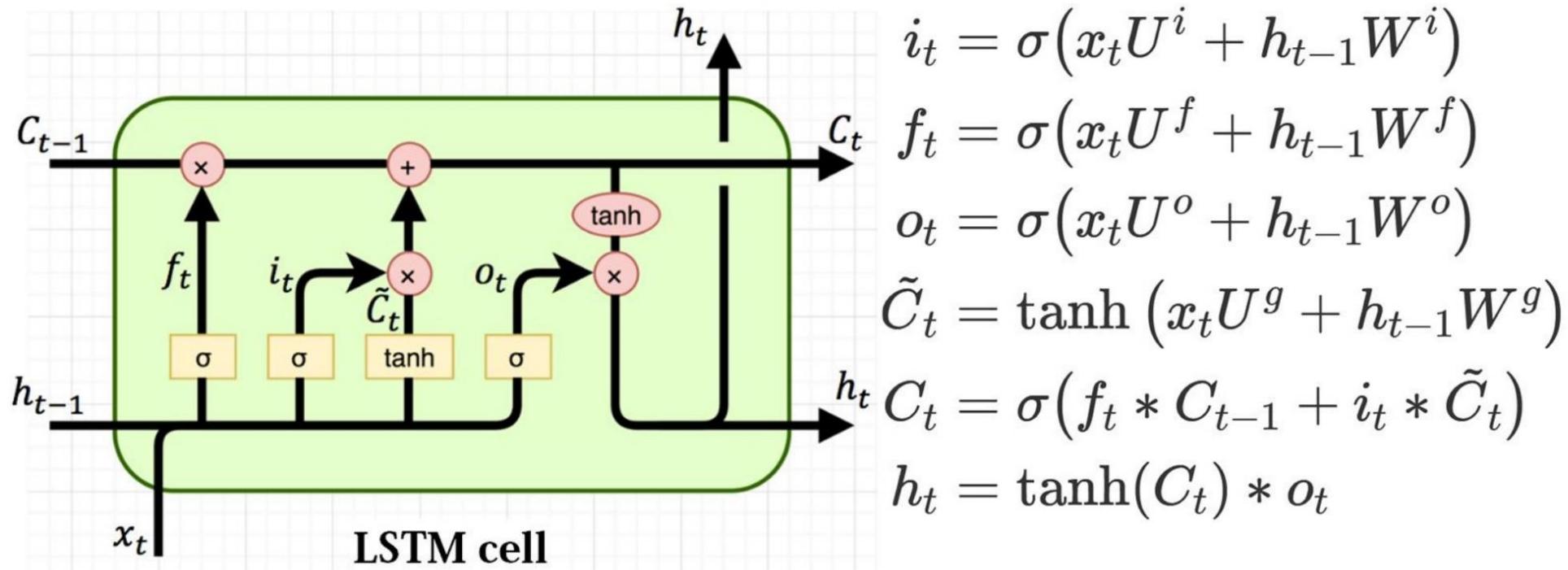
$$S_j = \tanh(W_x X_j + W_s S_{j-1} + b_1)$$

$$\prod_{j=k+1}^t \frac{\delta S_j}{\delta S_{j-1}} = \prod_{j=k+1}^t \tanh' W_s$$

Long Short Term Memory networks

- A special kind of RNN, capable of learning long-term dependencies, “I grew up in France… I speak fluent French.”
- The forget gate controls what information in the cell state to forget, given new information than entered the network
- The input gate controls what new information will be encoded into the cell state, given the new input information
- The output gate controls what information encoded in the cell state is sent to the network as input in the following time step, this is done via the output vector $h(t)$.

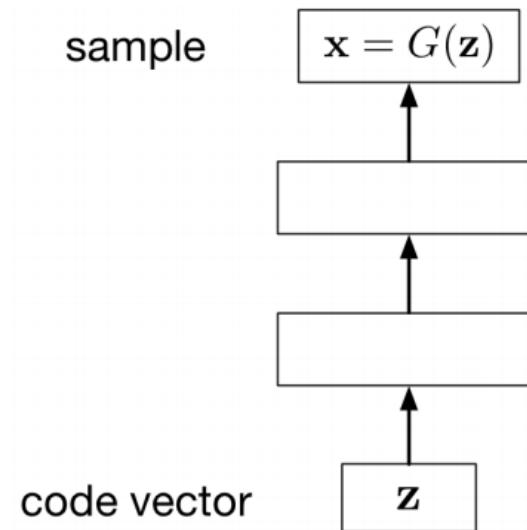
Long Short Term Memory networks



Generative Adversarial Networks

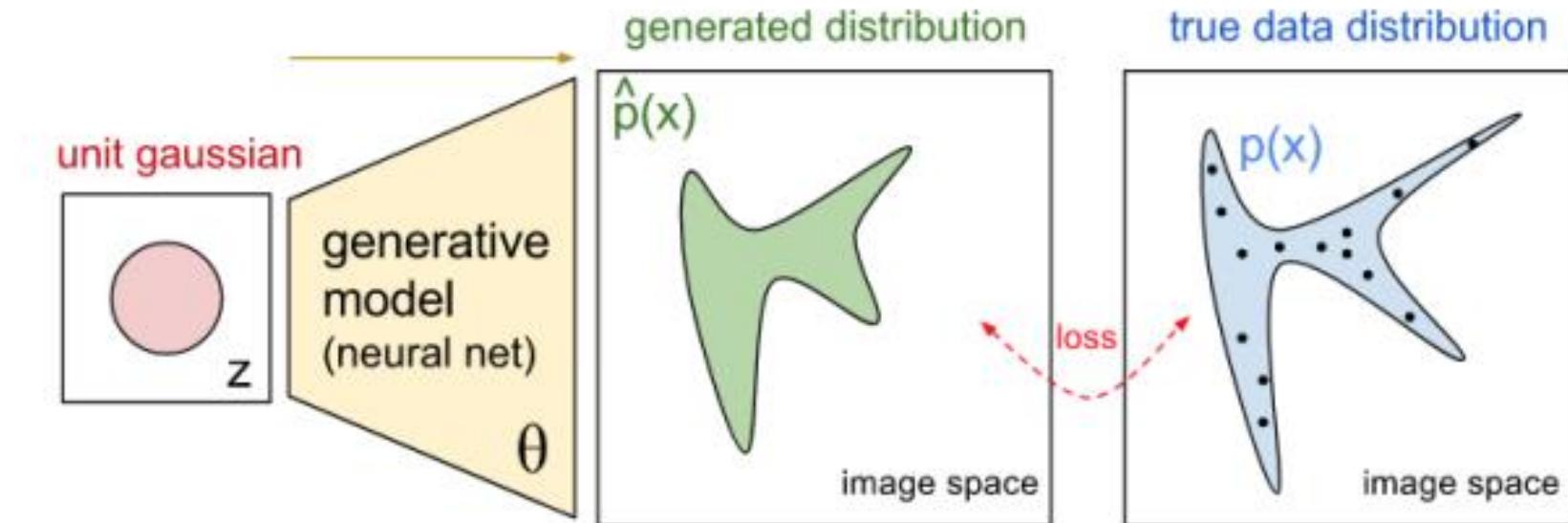
Implicit Generative Models

- Implicit generative models implicitly define a probability distribution
- Start by sampling the code vector \mathbf{z} from a fixed, simple distribution (e.g. spherical Gaussian)
- The generator network computes a differentiable function G mapping \mathbf{z} to an \mathbf{x} in data space



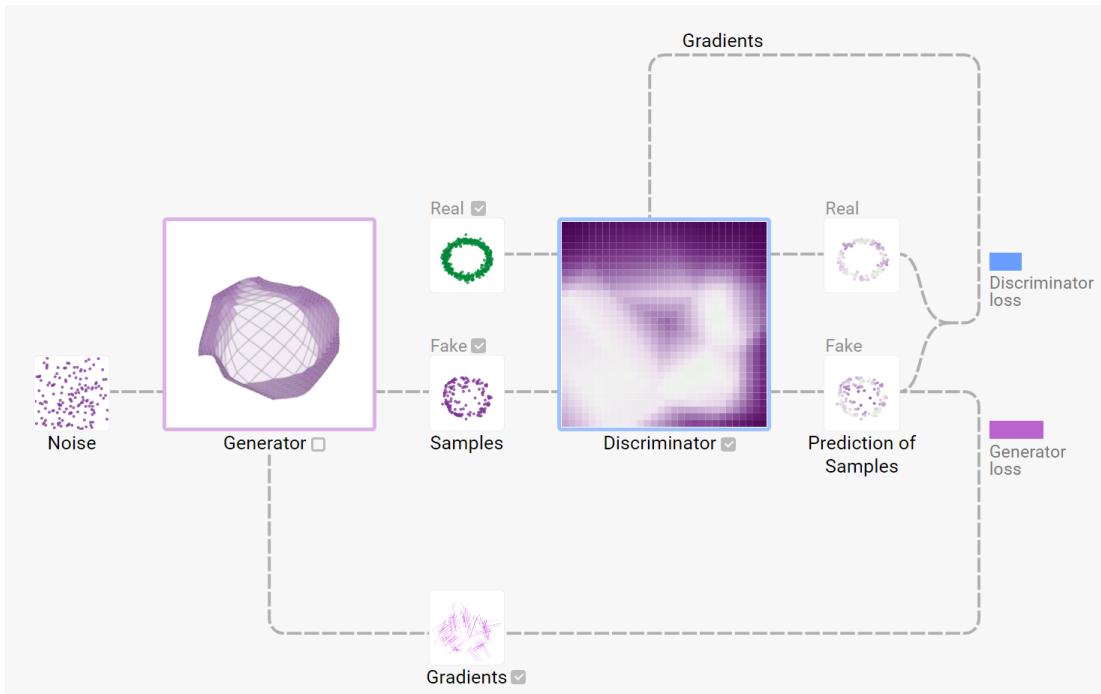
Generative Adversarial Networks

Implicit Generative Models

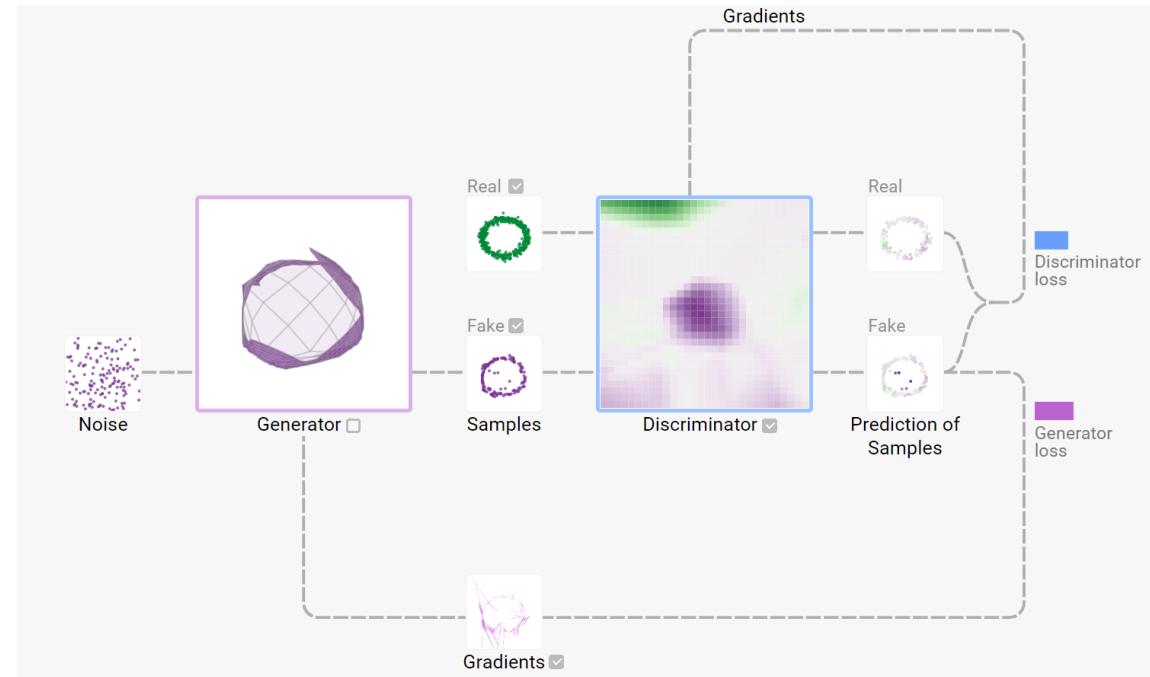


<https://openai.com/blog/generative-models/>

Generative Adversarial Networks



Epoch: 2746

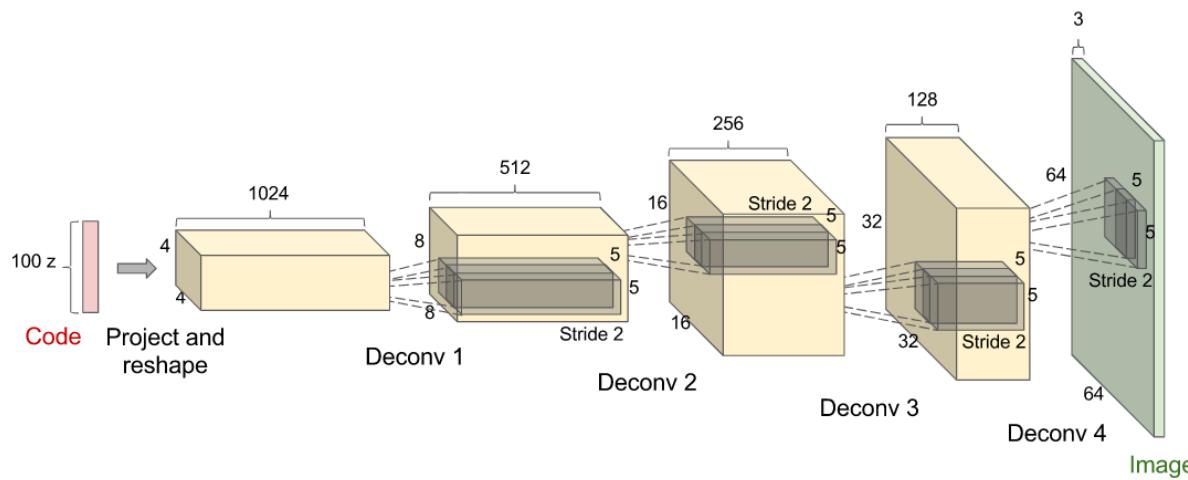
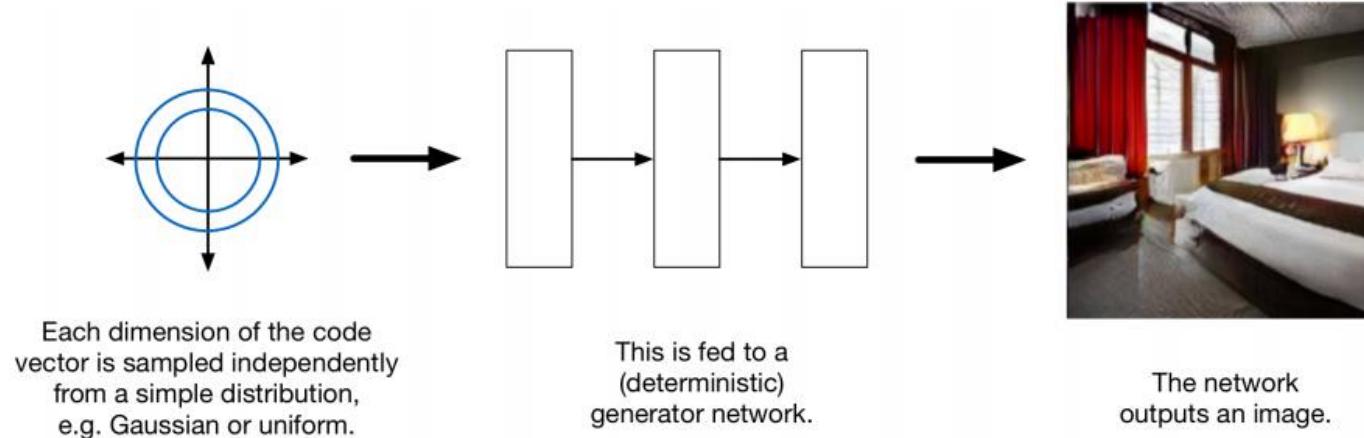


Epoch: 15000

<https://poloclub.github.io/ganlab/>

Generative Adversarial Networks

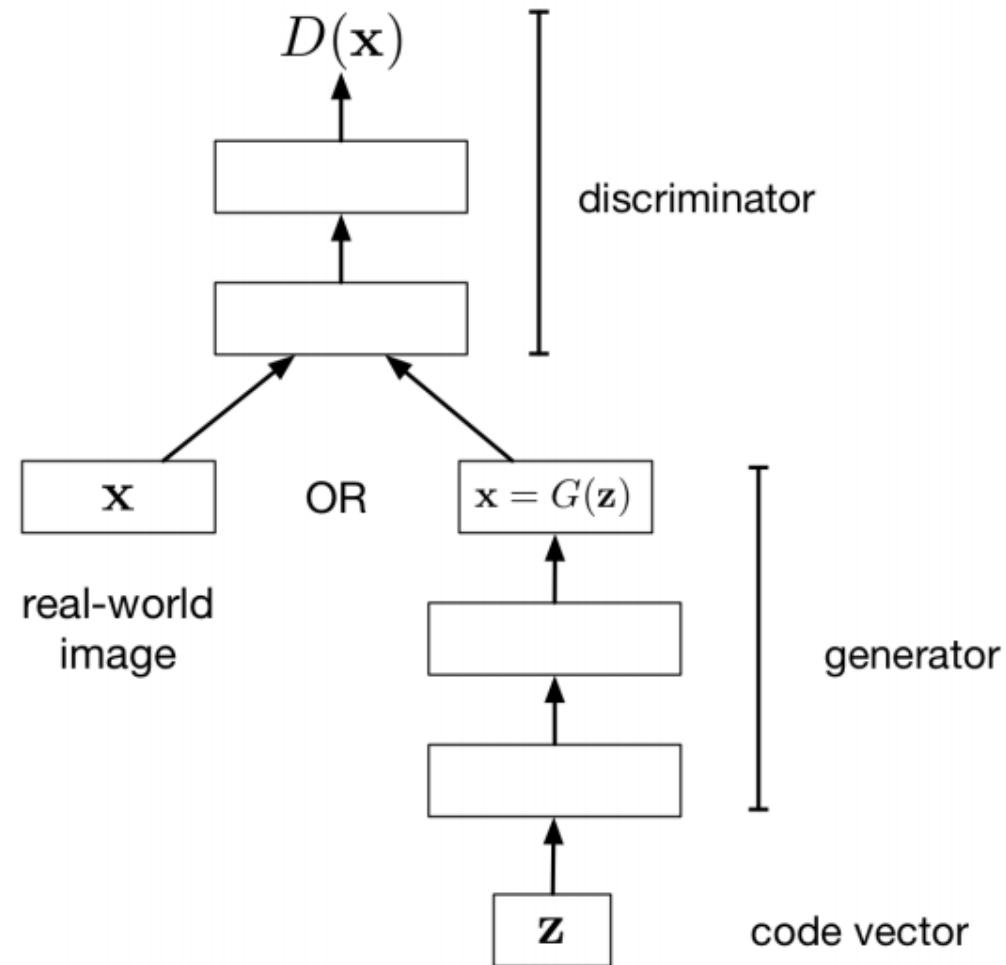
Implicit Generative Models



Generative Adversarial Networks

- The advantage of implicit generative models: if you have some criterion for evaluating the quality of samples, then you can compute its gradient with respect to the network parameters, and update the network's parameters to make the sample a little better.
- The idea behind Generative Adversarial Networks (GANs): train two different networks
 - The generator network tries to produce realistic-looking samples
 - The discriminator network tries to figure out whether an image came from the training set or the generator network
- The generator network tries to fool the discriminator network

Generative Adversarial Networks



Generative Adversarial Networks

- It is formulated as a **minimax game**, where:
 - The Discriminator is trying to maximize its reward $V(D, G)$
 - The Generator is trying to minimize Discriminator's reward (or maximize its loss)

$$V(D, G) = \boxed{\mathbb{E}_{x \sim p(x)}[\log D(x)]} + \boxed{\mathbb{E}_{z \sim q(z)}[\log(1 - D(G(z)))]}$$

- The Nash equilibrium of this particular game is achieved at:
 - $P_{data}(x) = P_{gen}(x) \quad \forall x$
 - $D(x) = \frac{1}{2} \quad \forall x$

Generative Adversarial Networks

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

Discriminator
updates

Generator
updates

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(\mathbf{x}^{(i)}) + \log (1 - D(G(\mathbf{z}^{(i)}))) \right].$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by descending its stochastic gradient:

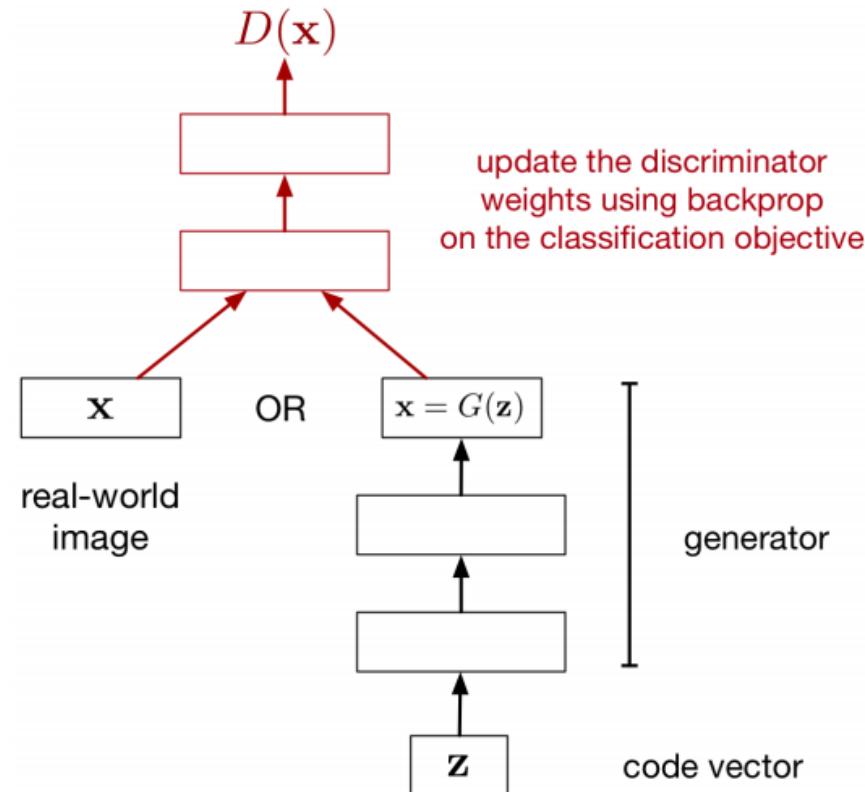
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(\mathbf{z}^{(i)}))).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

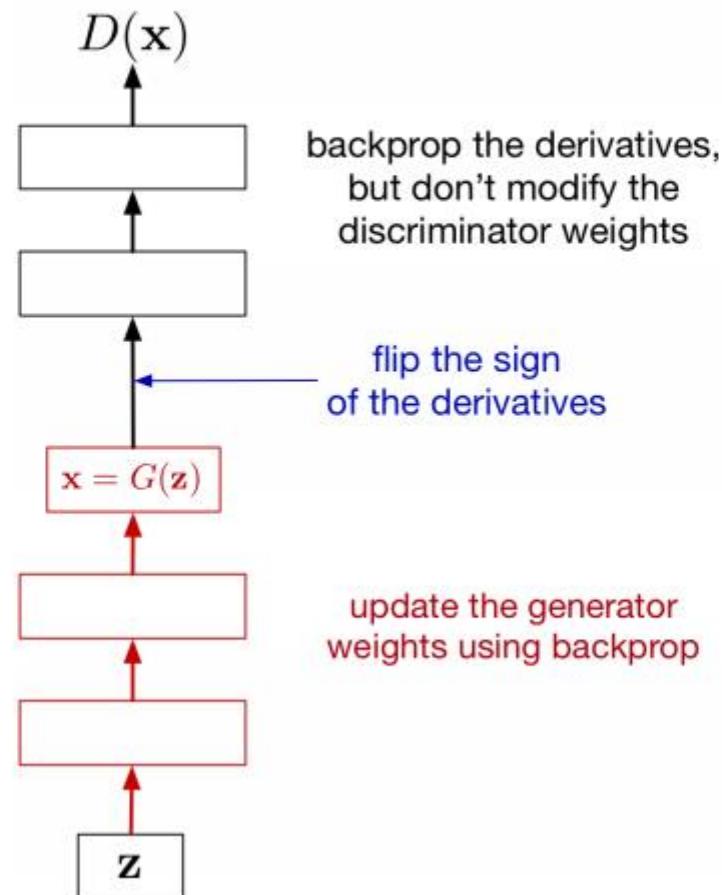
Generative Adversarial Networks

Updating the discriminator:



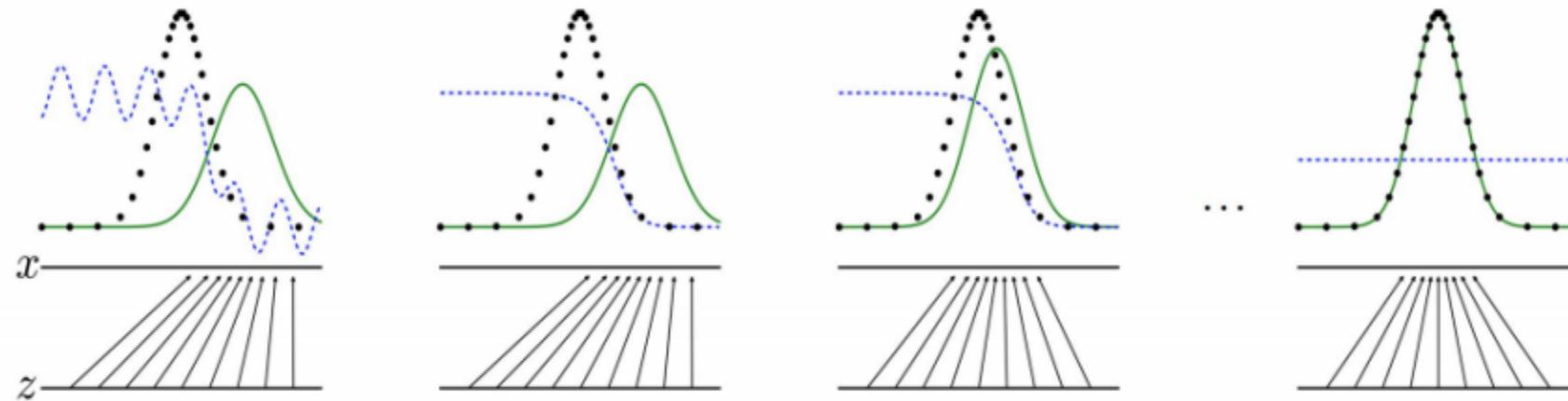
Generative Adversarial Networks

Updating the generator:



Generative Adversarial Networks

Alternating training of the generator and discriminator:



Generative Adversarial Networks

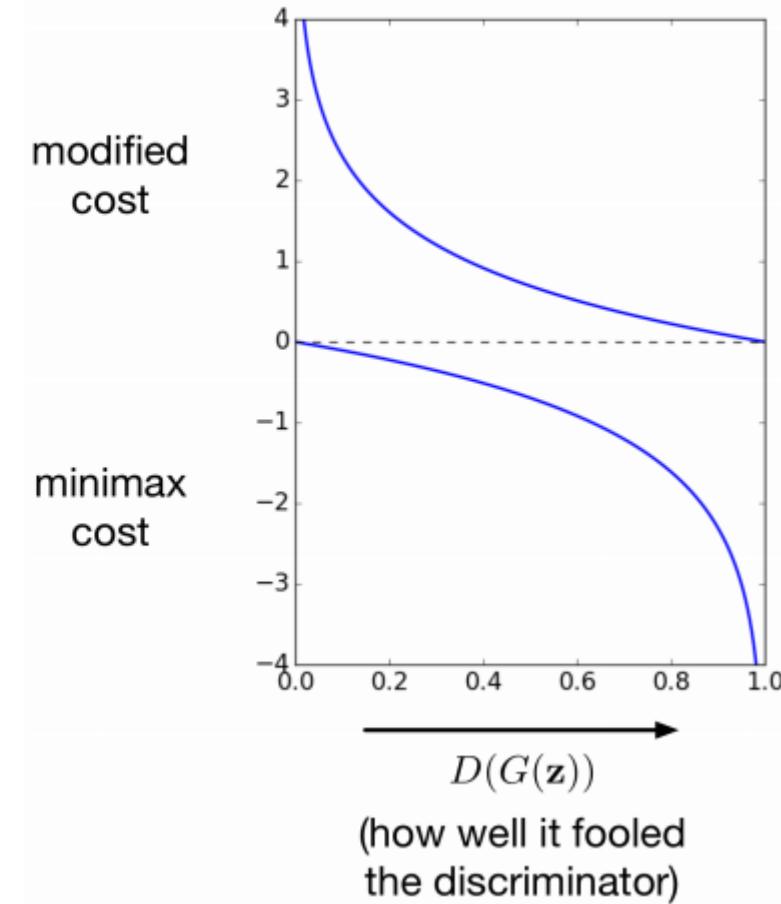
- Original minimax cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

- Modified generator cost:

$$\mathcal{J}_G = \mathbb{E}_{\mathbf{z}}[-\log D(G(\mathbf{z}))]$$

- This fixes the saturation problem.



Generative Adversarial Networks

Training problems

Non-Convergence

$$\min_x \max_y V(x, y)$$

Let $V(x, y) = xy$

- State 1:

x > 0	y > 0	V > 0
-------	-------	-------

Increase y	Decrease x
------------	------------

- State 2:

x < 0	y > 0	V < 0
-------	-------	-------

Decrease y	Decrease x
------------	------------

- State 3:

x < 0	y < 0	V > 0
-------	-------	-------

Decrease y	Increase x
------------	------------

- State 4 :

x > 0	y < 0	V < 0
-------	-------	-------

Increase y	Increase x
------------	------------

- State 5:

x > 0	y > 0	V > 0
-------	-------	-------

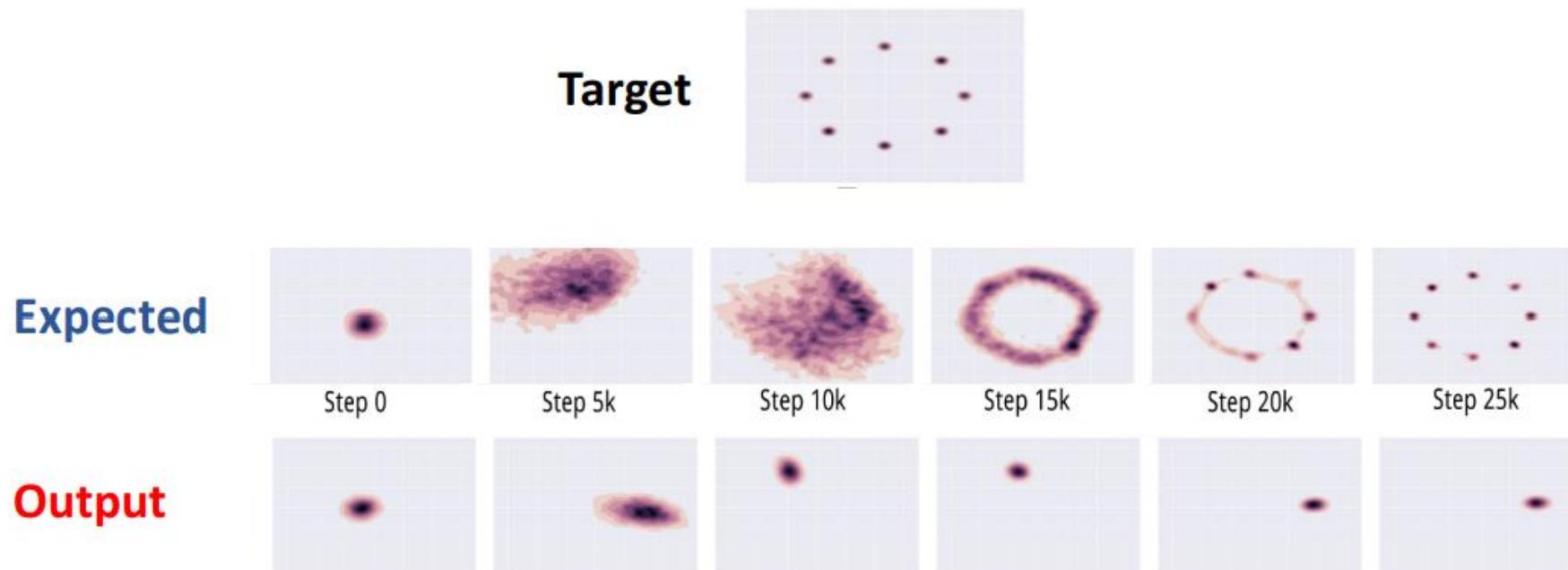
== State 1

Increase y	Decrease x
------------	------------

Generative Adversarial Networks

Mode-Collapse

- Generator fails to output diverse samples



Generative Adversarial Networks

GAN Samples

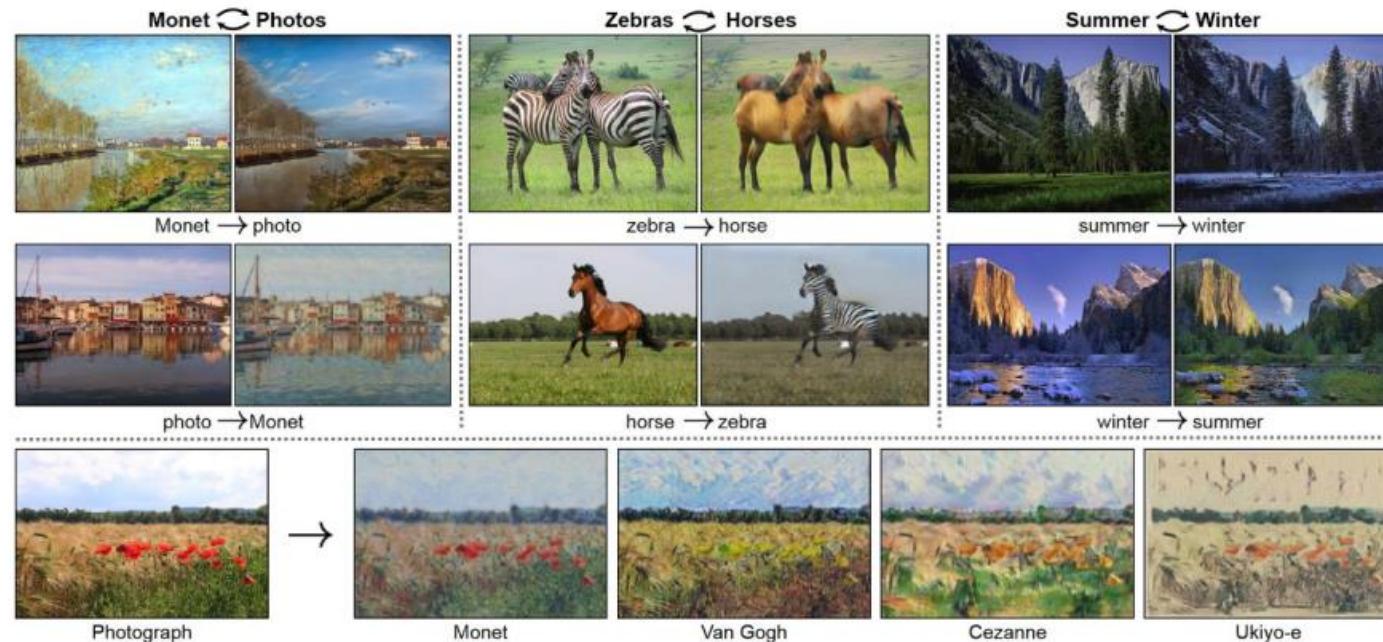


Karras et al., 2017. Progressive growing of GANs for improved quality, stability, and variation

Generative Adversarial Networks

CycleGAN

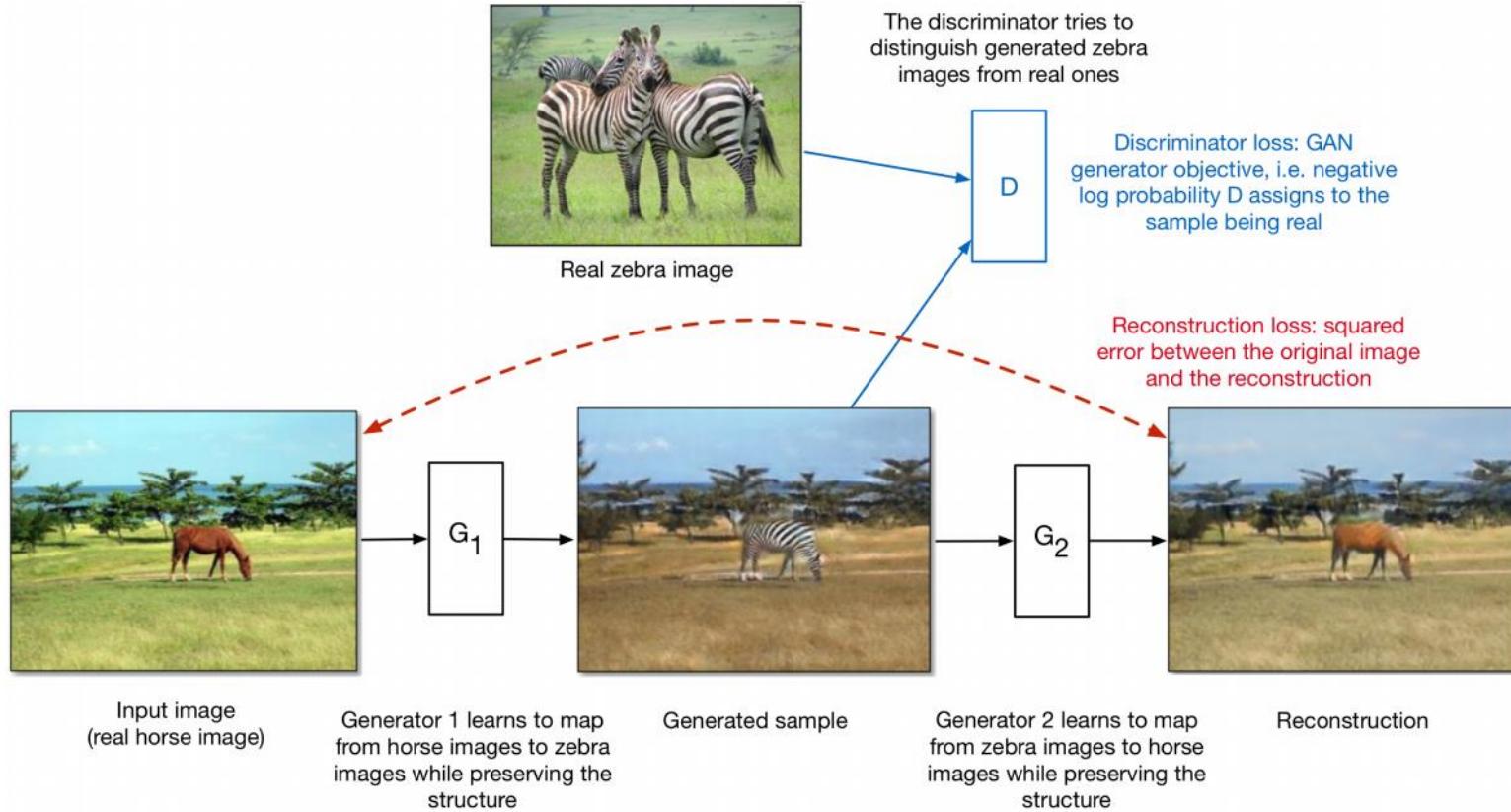
Style transfer problem: change the style of an image while preserving the content.



Data: Two unrelated collections of images, one for each style

Generative Adversarial Networks

CycleGAN



$$\text{Total loss} = \text{discriminator loss} + \text{reconstruction loss}$$

Generative Adversarial Networks

CycleGAN

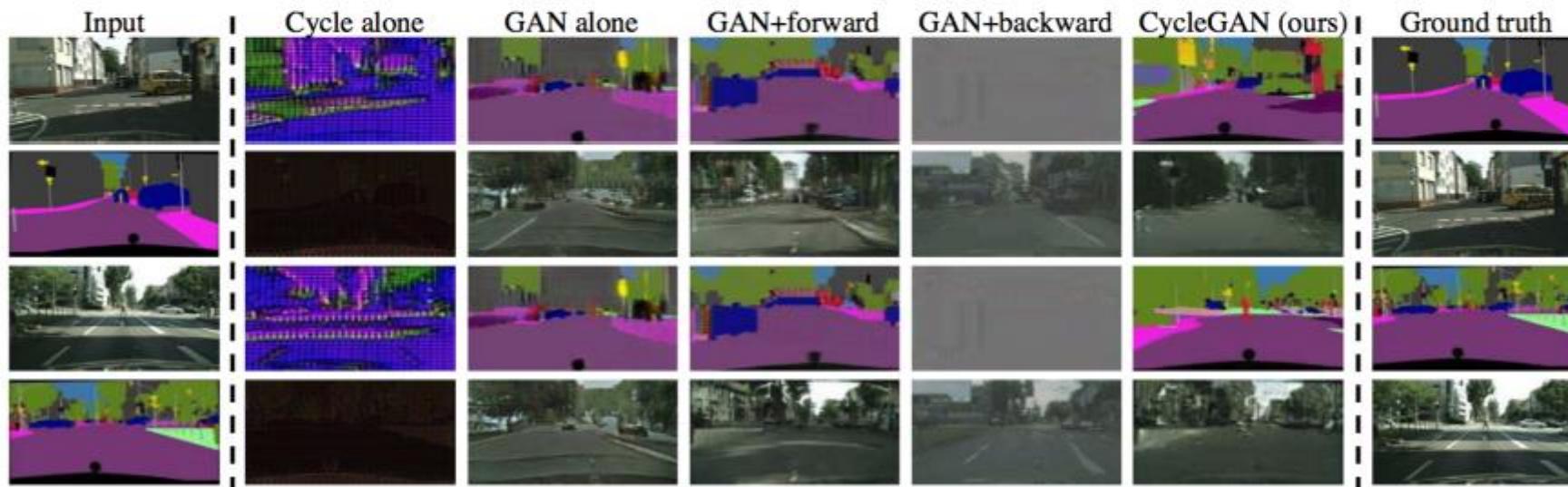
Style transfer between aerial photos and maps:



Generative Adversarial Networks

CycleGAN

Style transfer between road scenes and semantic segmentations (labels of every pixel in an image by object category):



Q&A

THANKS!

