

模式识别与深度学习 (13)

多层感知器算法

左旺孟

综合楼309

视觉感知与认知组

哈尔滨工业大学计算机学院

cswmzuo@gmail.com

13134506692

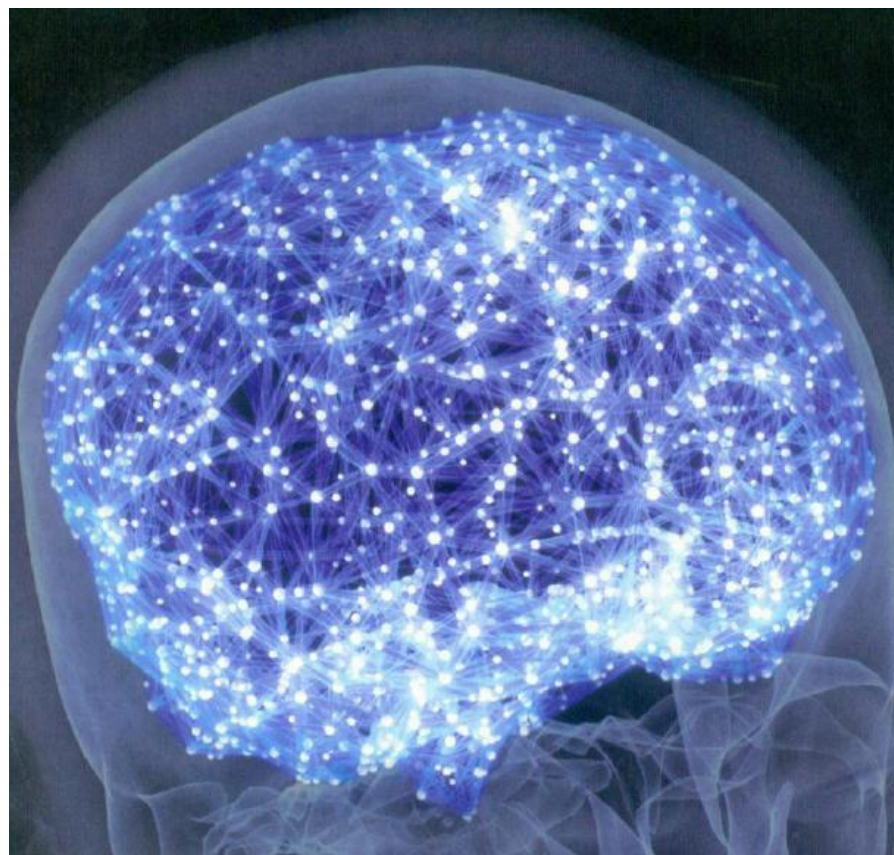
参考教材

- 深度学习, Ian GoodFellow, Yoshua Bengio, Aaron Courville著, 张志华 译, 人民邮电出版社, 2017.8

神经网络发展的三个阶段

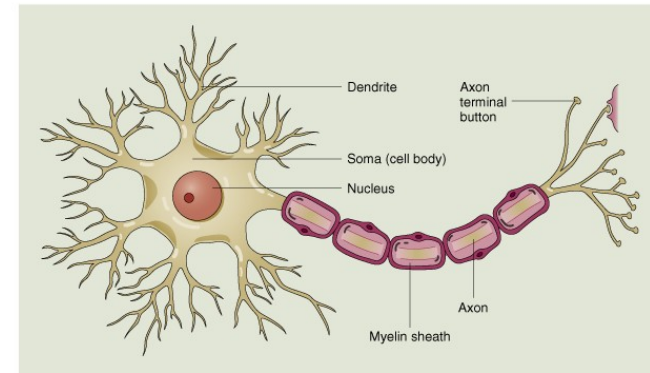
- 一、感知器
 - **McCulloch & Pitts / Hebb / Rosenblatt /**
 - **Minsky**
- 二、多层感知器感知器
 - **Hinton / LeCun**
 - **Minsky**
- 三、深度学习（**Hinton、LeCun、Bengio**）

人脑：神经元及其连接

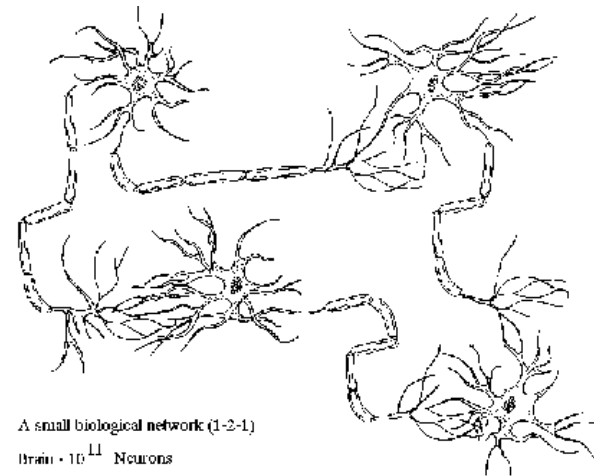
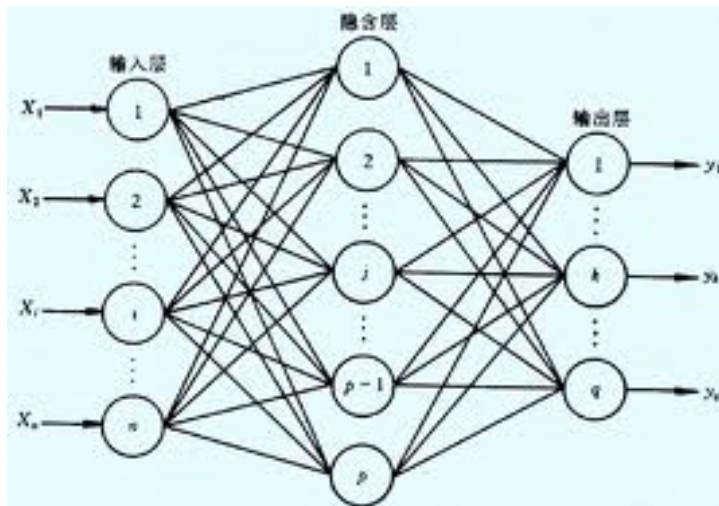


神经网络、感知器、人工神经网络

- 神经元、突触
- 神经网络



© 2000 John Wiley & Sons, Inc.



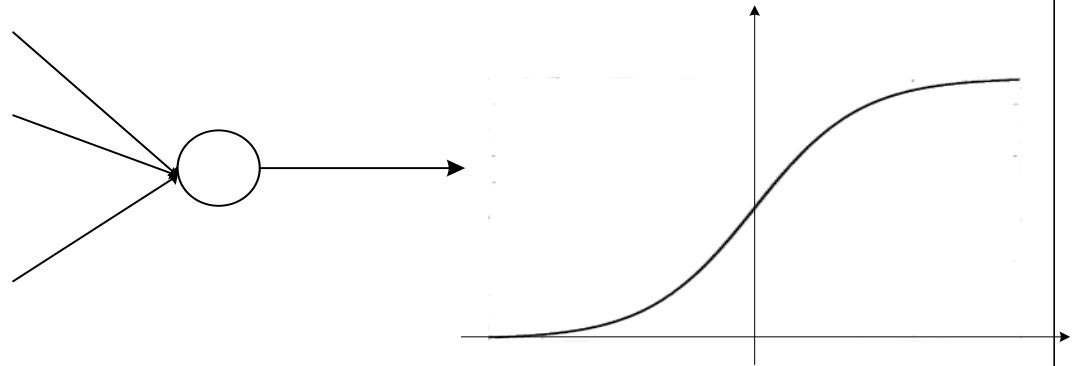
A small biological network (1-2-1)

Brain - 10^{11} Neurons

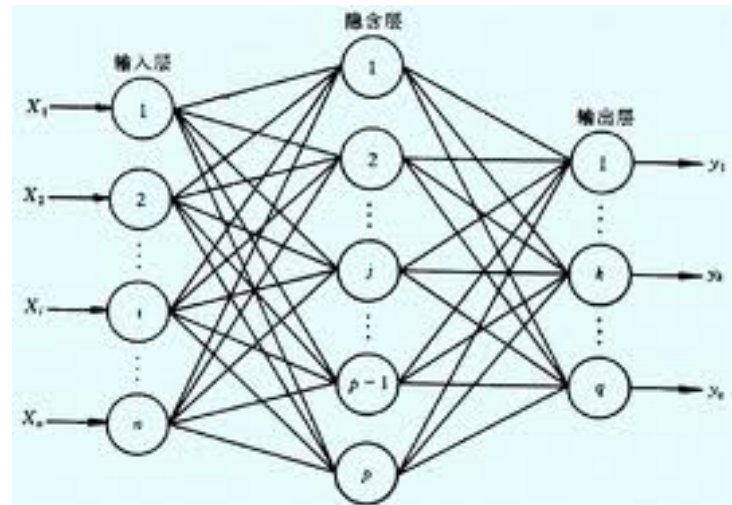
Nervous system - 10^{14} Synapses

人工神经网络

- 出发点
 - 模拟人类智能
- 建模方法
 - 神经元-突触
 - 网络结构
 - 模型参数?



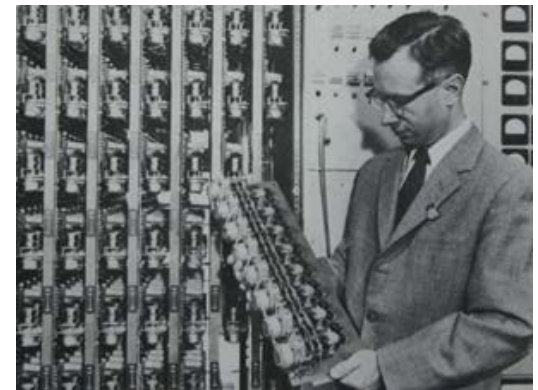
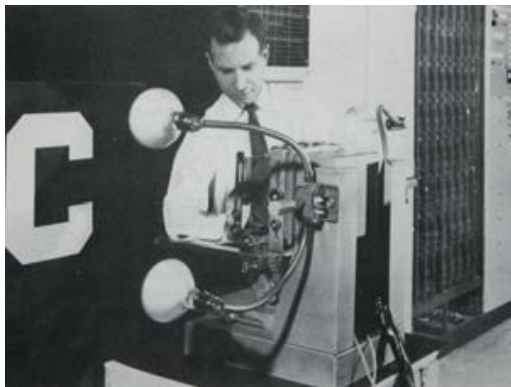
- 学习方法
 - 参数学习 (BP, 反向传播)
 - 结构学习 (NAS, 网络结构搜索)



约·冯·诺依曼, 《计算机与人脑》, 商务出版社 (1958, 1965, 2011)

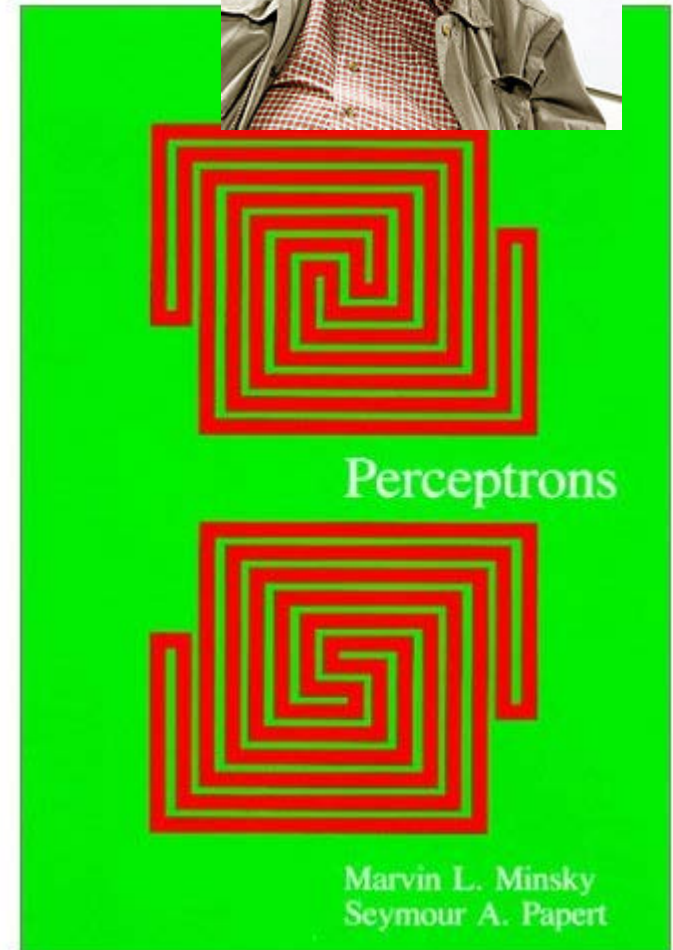
感知器算法: 历史

- 1957年, Rosenblatt提出感知器算法
- 线性 vs. 非线性

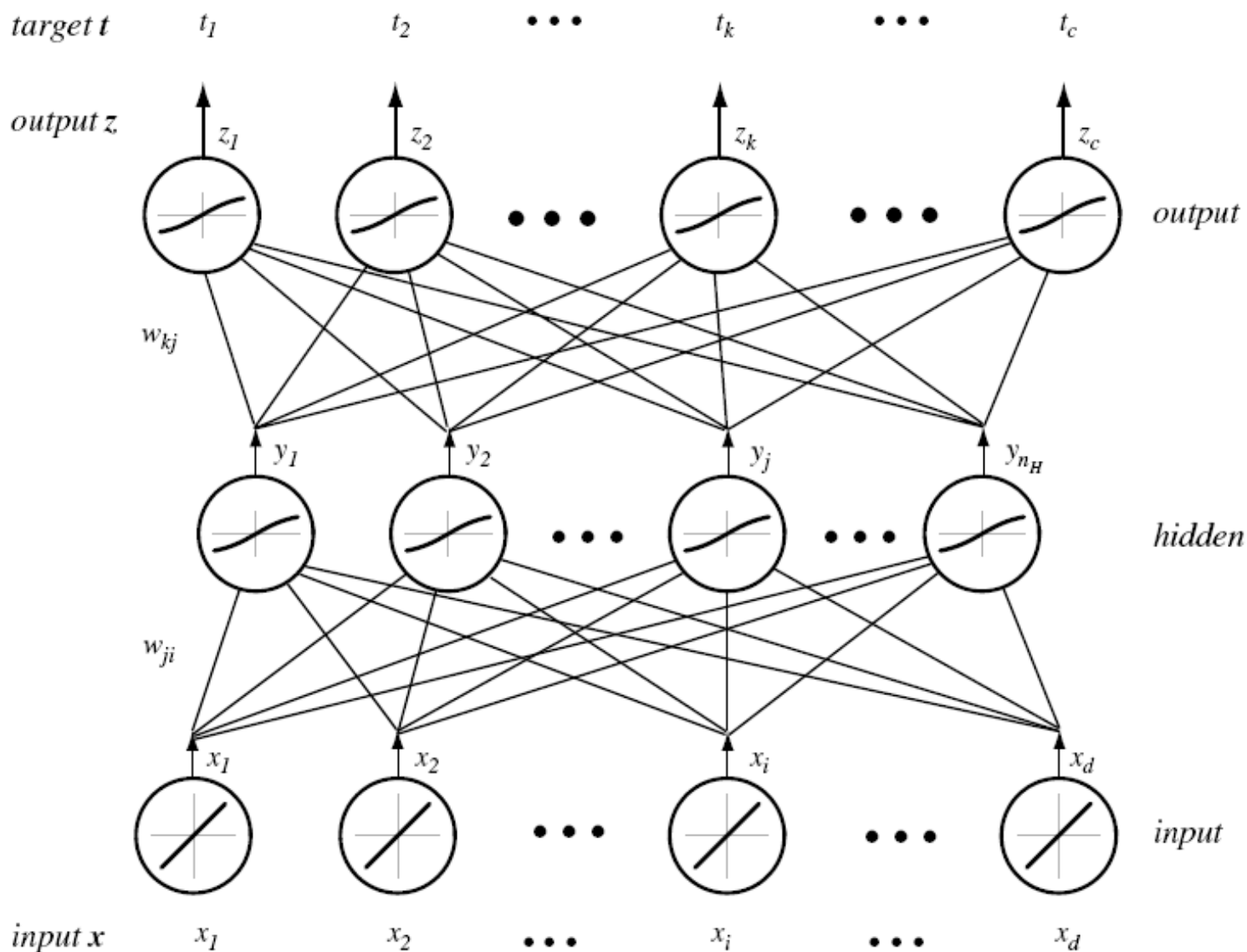


Perceptrons: An Introduction to Computational Geometry

- 1969, 1987
- AI winter
- *Turing* Award, the *Japan* Prize, the *IJCAI* Award for Research Excellence, and the *Benjamin Franklin Medal, Fellow of the Computer History Museum*.
- An adviser on the movie *2001: A Space Odyssey* and is referred to in the movie.

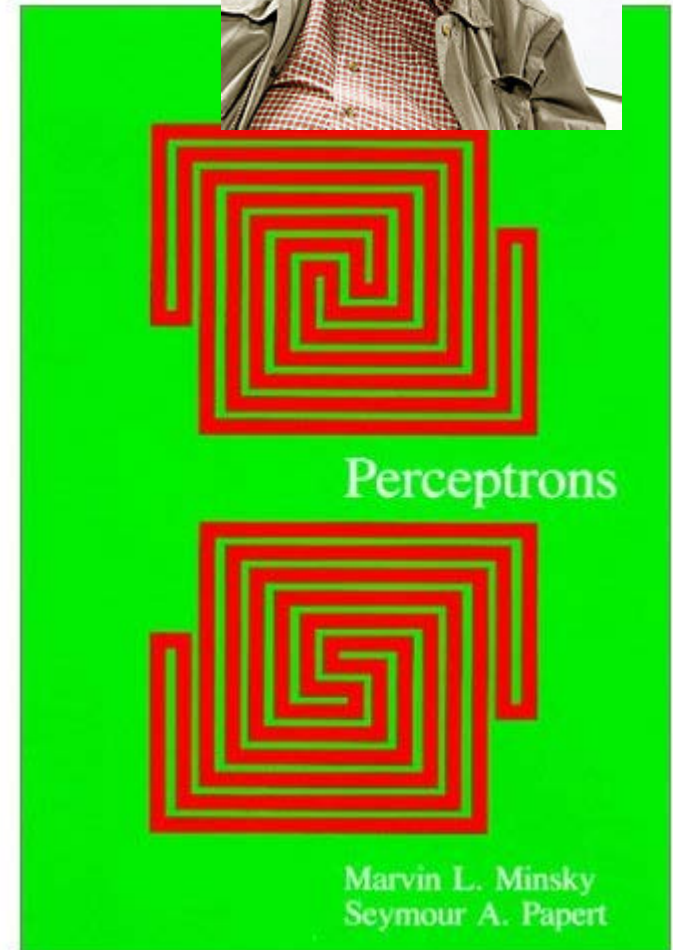


多层感知器



Perceptrons: An Introduction to Computational Geometry

- 1969, 1987
- AI winter
- *Turing* Award, the *Japan* Prize, the *IJCAI* Award for Research Excellence, and the *Benjamin Franklin Medal, Fellow of the Computer History Museum*.
- An adviser on the movie *2001: A Space Odyssey* and is referred to in the movie.



Deep Learning

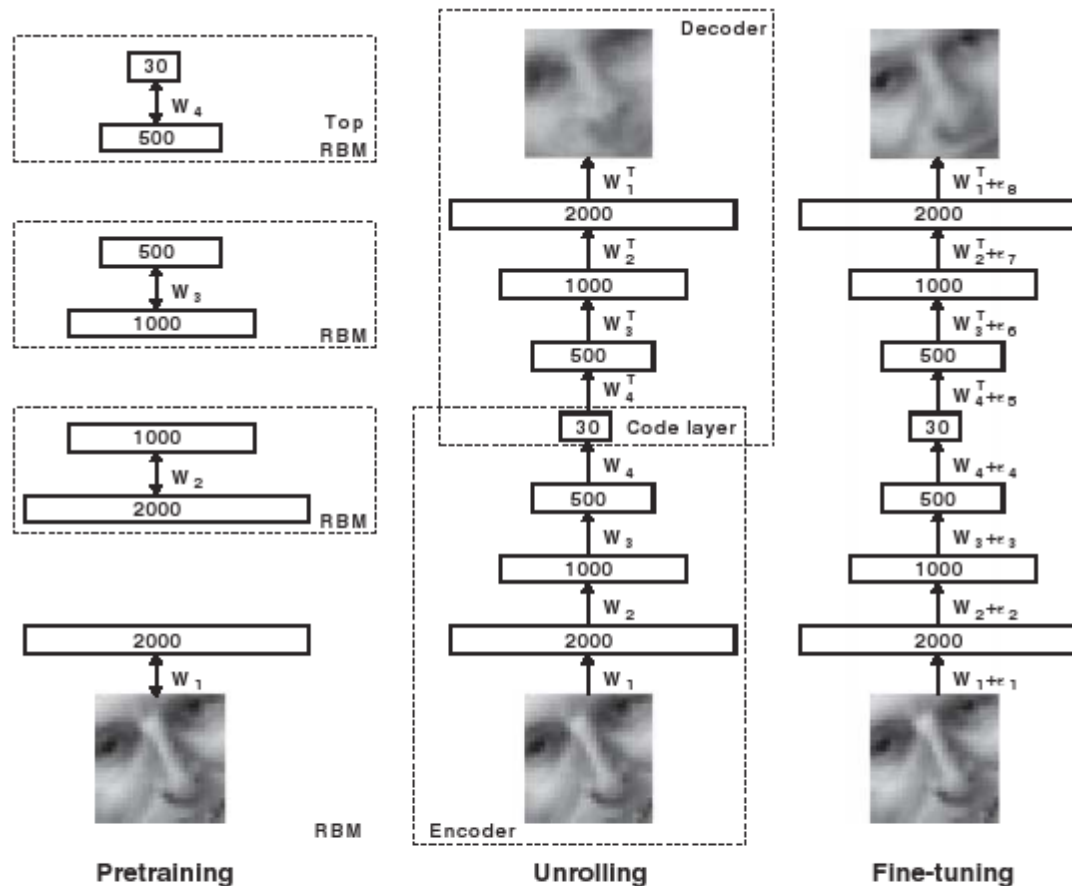
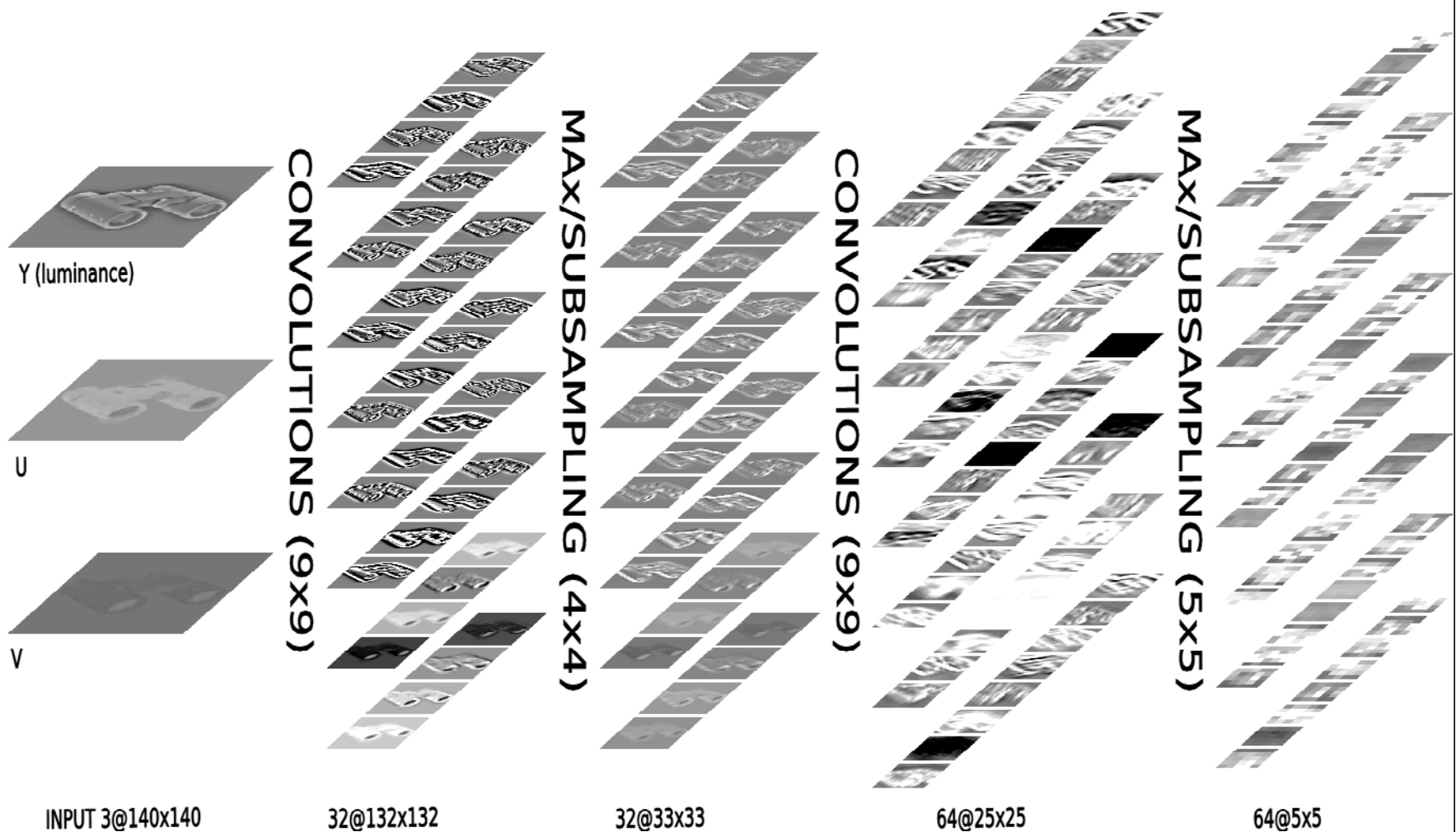


Fig. 1. Pretraining consists of learning a stack of restricted Boltzmann machines (RBMs), each having only one layer of feature detectors. The learned feature activations of one RBM are used as the “data” for training the next RBM in the stack. After the pretraining, the RBMs are “unrolled” to create a deep autoencoder, which is then fine-tuned using backpropagation of error derivatives.

Convolutional Net



2. 多层感知器网络

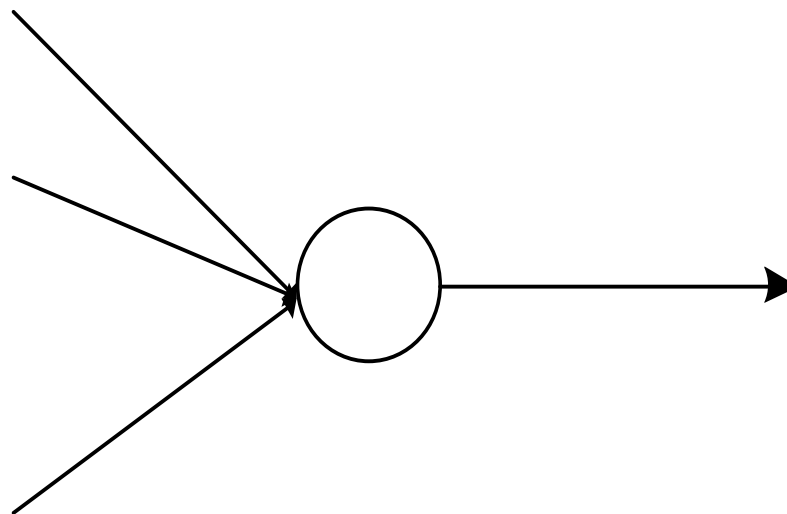
- 多层感知器网络（MLP, Multilayer Perceptron）
 - 多层神经网络
 - BP神经网络
- 主要任务
 - P1: 网络设计
 - P2: 训练

数据的表示

- 单变量: x, y, z, x_1
- 向量: $\mathbf{x}, \mathbf{y}, \mathbf{x}_1$ $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{x}_1$
 - 向量中的组元 $\mathbf{x} = [x_1, \dots, x_d]^t$
- 矩阵 $\mathbf{A}, \mathbf{X}, \mathbf{M}$ $\boldsymbol{A}, \boldsymbol{X}, \boldsymbol{M}$

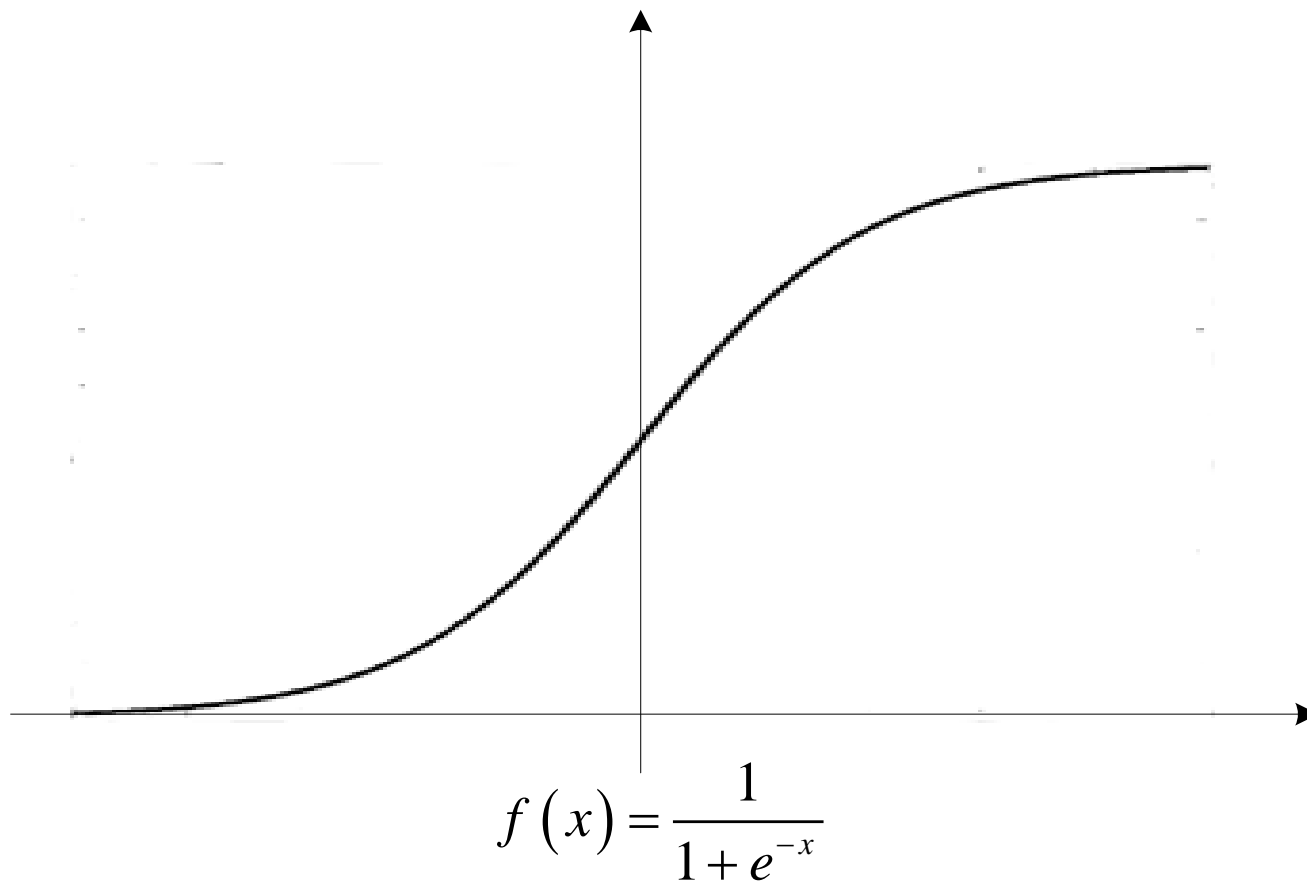
2. 多层感知器网络 (MLP, Multilayer Perceptron)

- 神经元模型

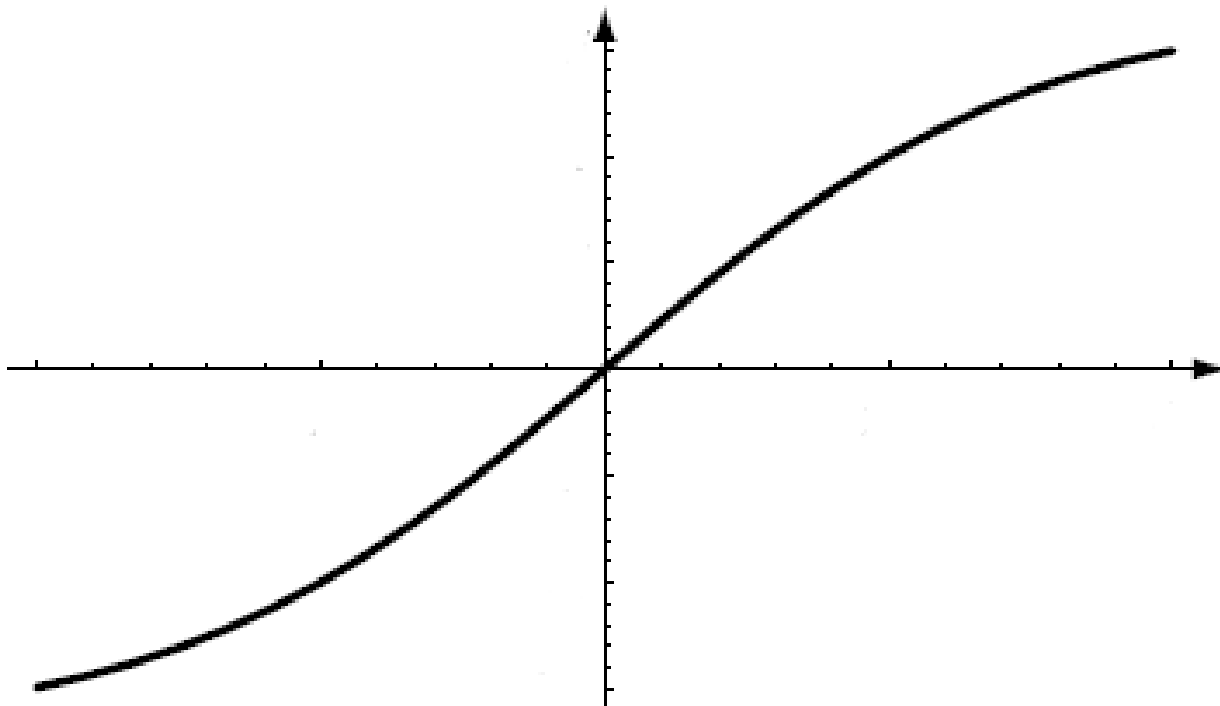


$$y = f(\mathbf{w}^t \mathbf{x}) = f\left(\sum_{i=1}^d w_i x_i\right), \quad \mathbf{f} \text{称为激活函数}$$

激活函数——对数Sigmoid函数



激活函数—双曲正切Sigmoid函数

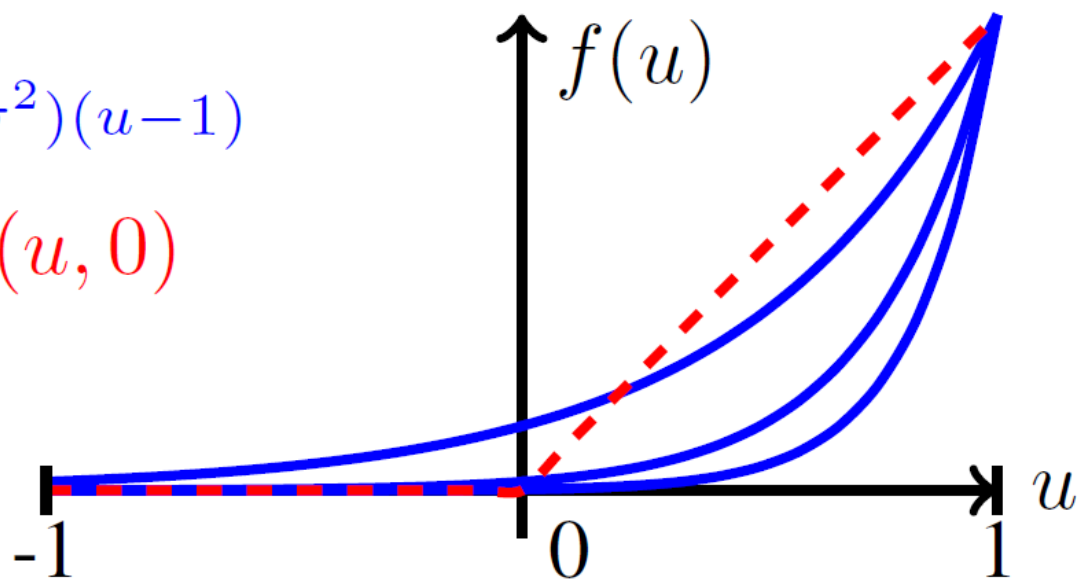


$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

激活函数—Rectified linear unit函数

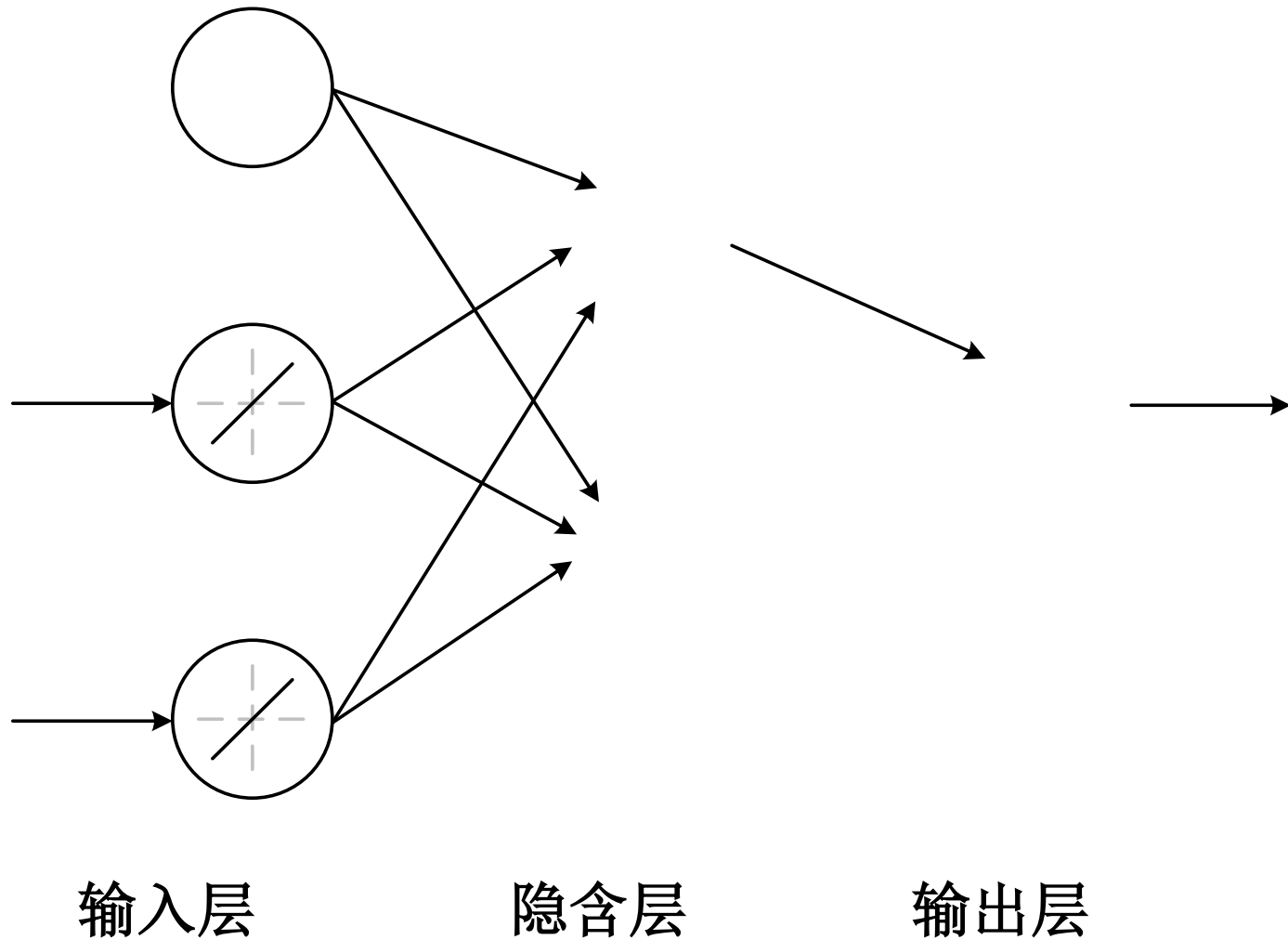
$$f(u) = e^{(2/\sigma^2)(u-1)}$$

$$f(u) = \max(u, 0)$$



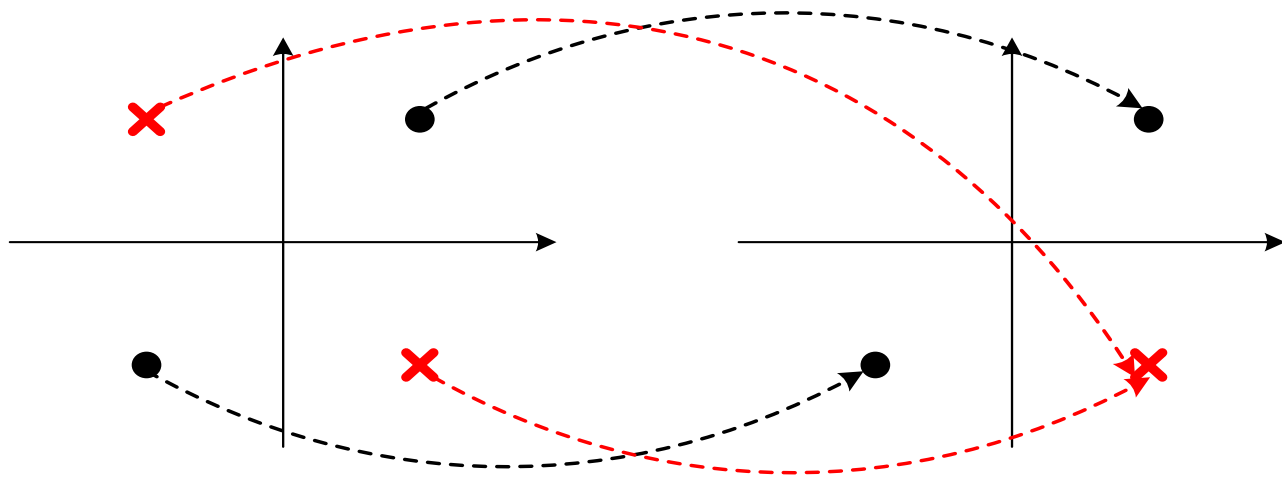
Vinod Nair, Geoffrey E. Hinton, Rectified Linear Units Improve Restricted Boltzmann Machines, ICML 2010.

解决异或 (± 1) 问题的多层感知器



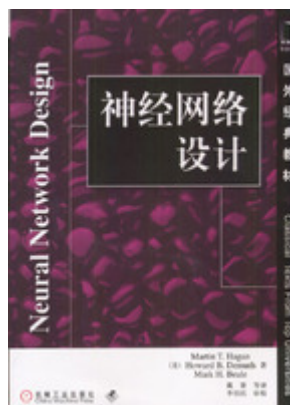
多层感知器的分类原理

- 隐含层实现对输入空间的非线性映射，输出层实现线性分类；
- 非线性映射方式和线性判别函数可以同时学习。

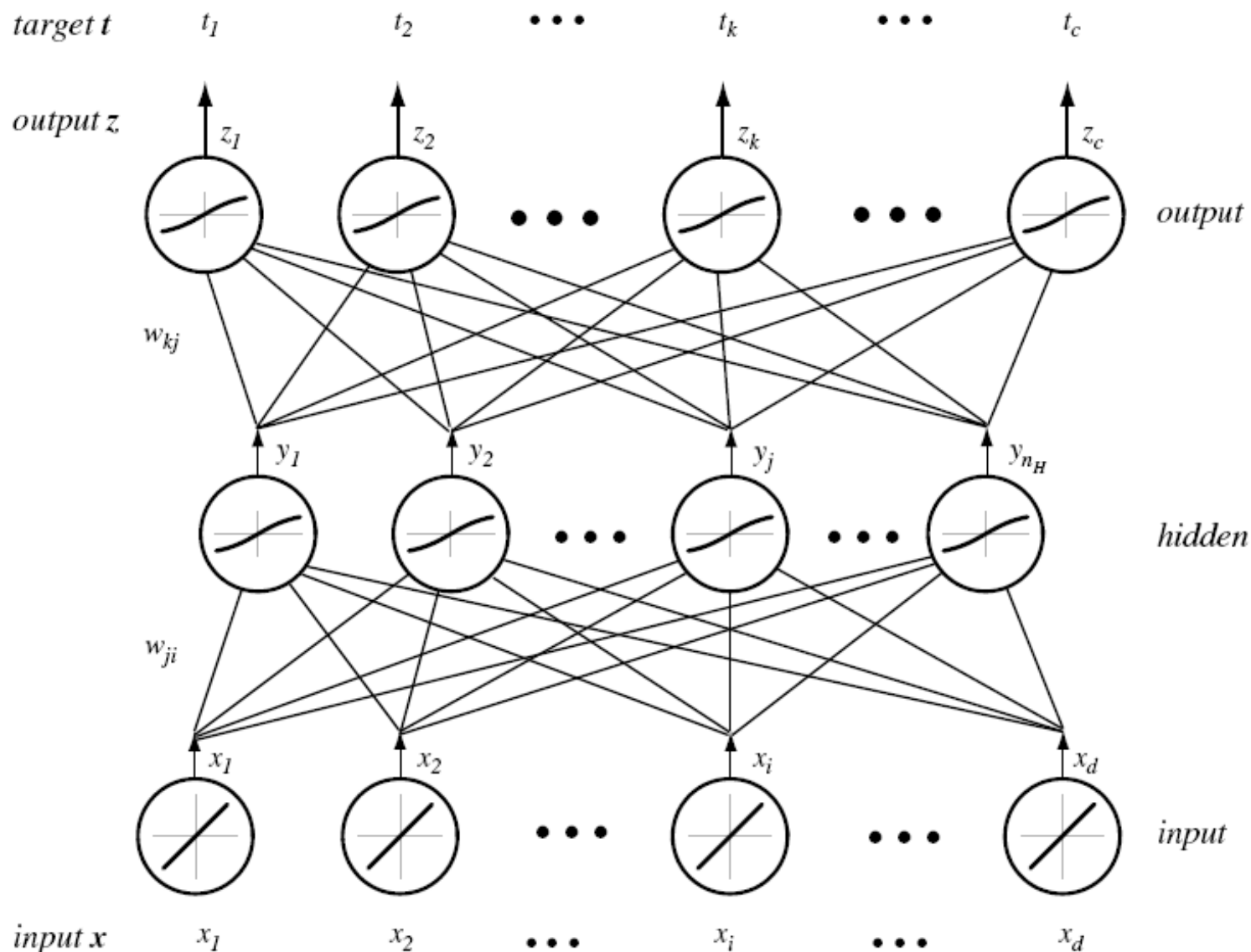


参考书

- 《神经网络设计》，[美]哈根等 著，戴葵等 译，机械工业出版社，2002-9-1



标准的三层感知器网络



多层感知器网络

- 主要任务
 - P1: 网络设计
 - P2: 训练

多层感知器网络的设计

- **选定层数**：三层网络 \rightarrow 深层网络
 - 增加网络层数并不能提高网络的分类能力？
- **输入层**：输入层节点数为输入特征的维数 d ，映射函数采用线性函数；
- **隐含层**：隐含层节点数需要设定，一般来说，隐含层节点数越多，网络的分类能力越强，映射函数一般采用Sigmoid函数；
- **输出层**：输出层节点数可以等于类别数 c ，也可以采用编码输出的方式，少于类别数 c ，输出函数可以采用线性函数或Sigmoid函数。

三层网络的判别函数形式

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

第 k 个输出层神经元的输出，其中 d 为特征维数， n_H 为隐层节点数。

2.2 学习机制

- 大脑神经网络的学习过程
 - 环境
 - 网络结构/参数调整
 - 响应、反馈
- 神经网络学习机制
 - 错误校正
 - 记忆学习
 - Hebbian学习
 - 竞争学习

Who Invented Backpropagation

- <http://people.idsia.ch/~juergen/who-invented-backpropagation.html>
- The first NN-specific application of efficient BP as above was described in 1981 (Werbos, 1981).
- Related work was published several years later (Parker, 1985; LeCun, 1985).
- A paper of 1986 significantly contributed to the popularisation of BP for NNs (Rumelhart et al., 1986).

2.2 MLP的训练--误差反向传播算法

(BP, Backpropagation algorithm)

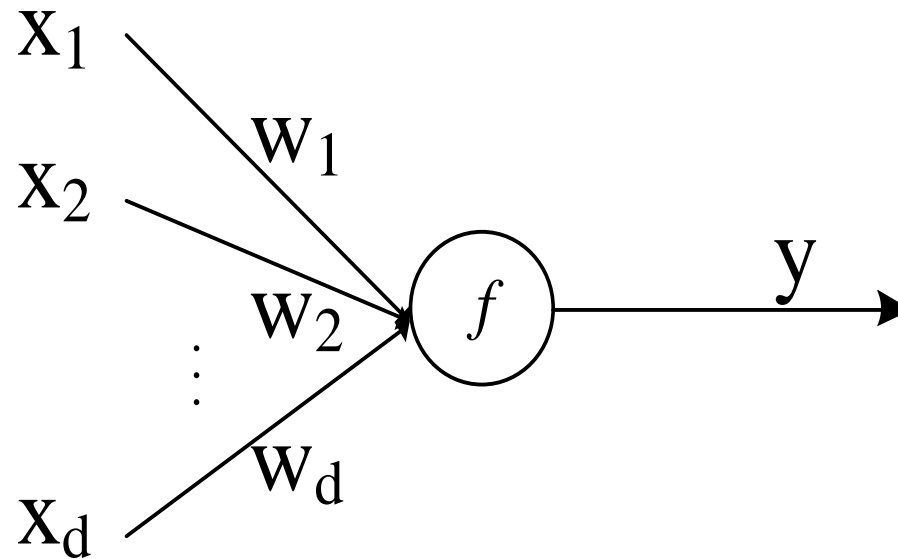
$$z_k = f_2 \left(\sum_{j=1}^{n_H} w_{kj} f_1 \left(\sum_{i=1}^d w_{ji} x_i + w_{j0} \right) + w_{k0} \right)$$

- BP算法的实质是一个均方误差最小算法 (LMS)
- 符号定义：训练样本 \mathbf{x} ，期望输出 $\mathbf{t}=(t_1, \dots, t_c)$ ，网络实际输出 $\mathbf{z}=(z_1, \dots, z_c)$ ，隐层输出 $\mathbf{y}=(y_1, \dots, y_{n_H})$ ，第 k 个神经元的净输出 net_k 。

- 目标函数：
$$J(\mathbf{w}) = \frac{1}{2} \|\mathbf{t} - \mathbf{z}\|^2 = \frac{1}{2} \sum_{i=1}^c (t_i - z_i)^2$$

迭代公式：
$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta \mathbf{w}(m) = \mathbf{w}(m) - \eta \frac{\partial J}{\partial \mathbf{w}}$$

神经元

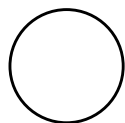
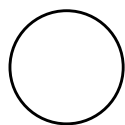
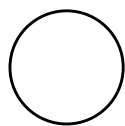


$$y = f(\mathbf{w}^t \mathbf{x}) = f\left(\sum_{i=1}^d w_i x_i\right), \quad \mathbf{f} \text{称为激活函数}$$

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

The diagram illustrates the mapping of terms in the equation to neural network components:

- A blue line connects the outer summation $\sum_{j=1}^{n_H} w_{kj} f_1 \left(\sum_{i=1}^d w_{ji} x_i + w_{j0} \right) + w_{k0}$ to the label net_k .
- A red line connects the inner summation $\sum_{i=1}^d w_{ji} x_i + w_{j0}$ to the label net_j .
- A green line connects the input variable x_i to the label y_j .



输出层

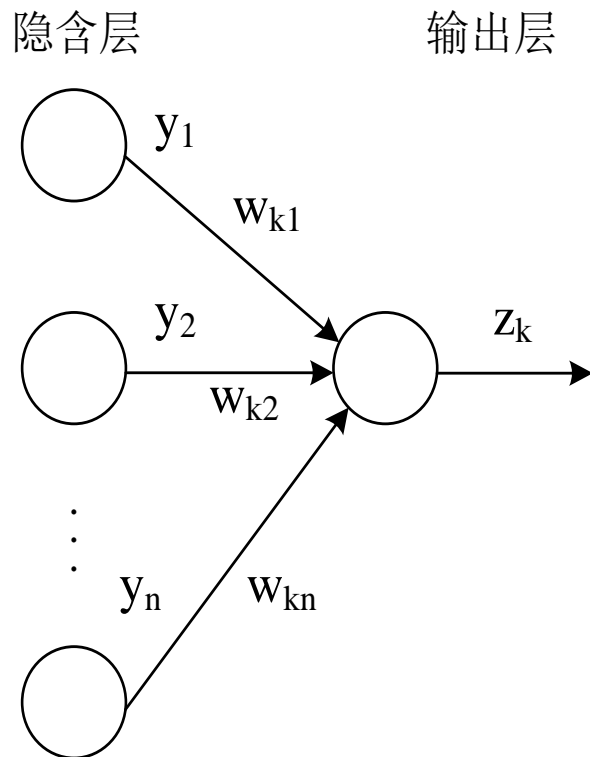
$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}}$$

$$net_k = \sum_{j=0}^{n_H} w_{kj} y_j, \quad \frac{\partial net_k}{\partial w_{kj}} = y_j$$

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

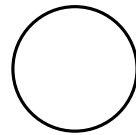
$$\frac{\partial J}{\partial w_{kj}} = -(t_k - z_k) f'(net_k) y_j = -\delta_k y_j$$

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

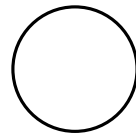


隐含层

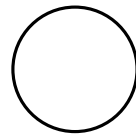
$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}}$$

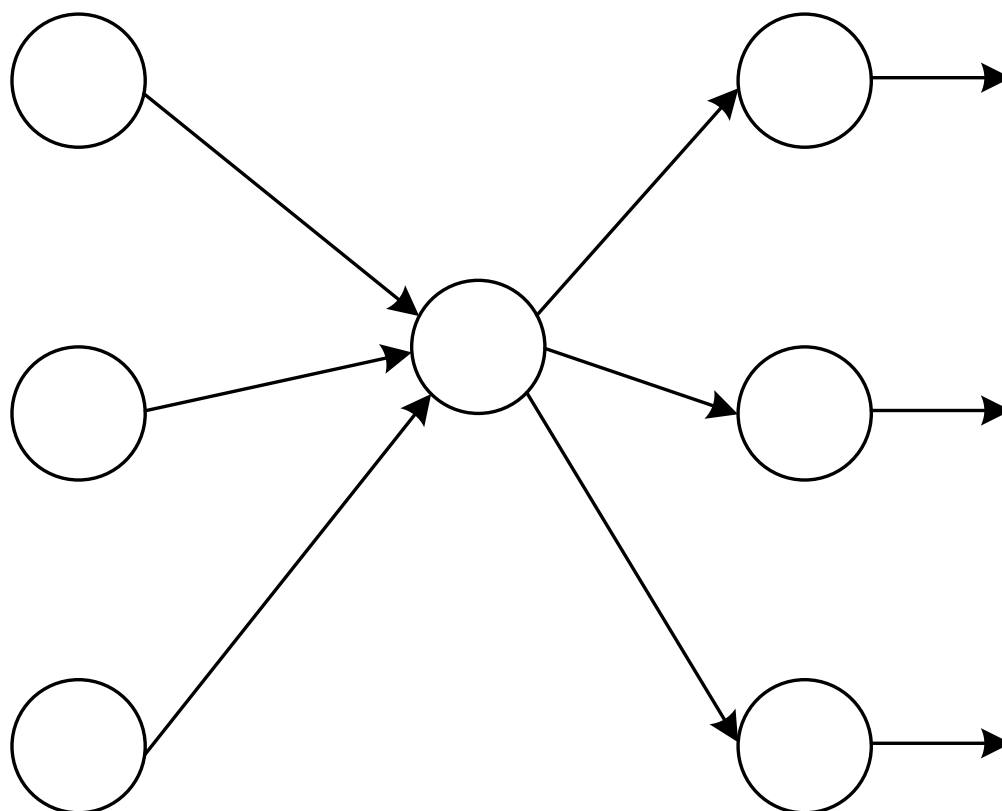


$$\frac{\partial y_j}{\partial net_j} = f'(net_j)$$



$$\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \left(\sum_{m=0}^d w_{jm} x_m \right) = x_i$$





隐含层

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

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

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

迭代公式

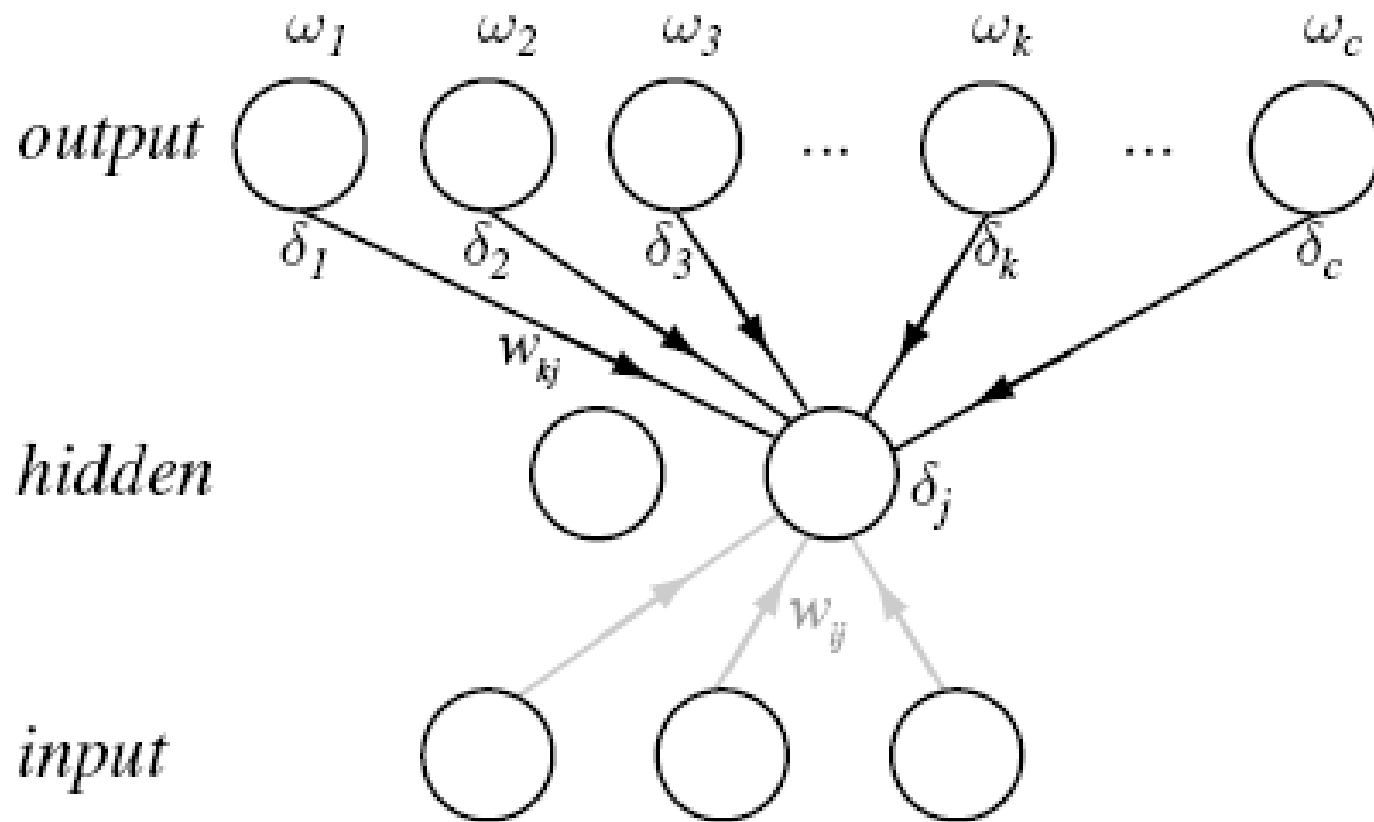
- 输出层:

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

- 隐含层:

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

误差反向传播



P2: BP算法—批量修改

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

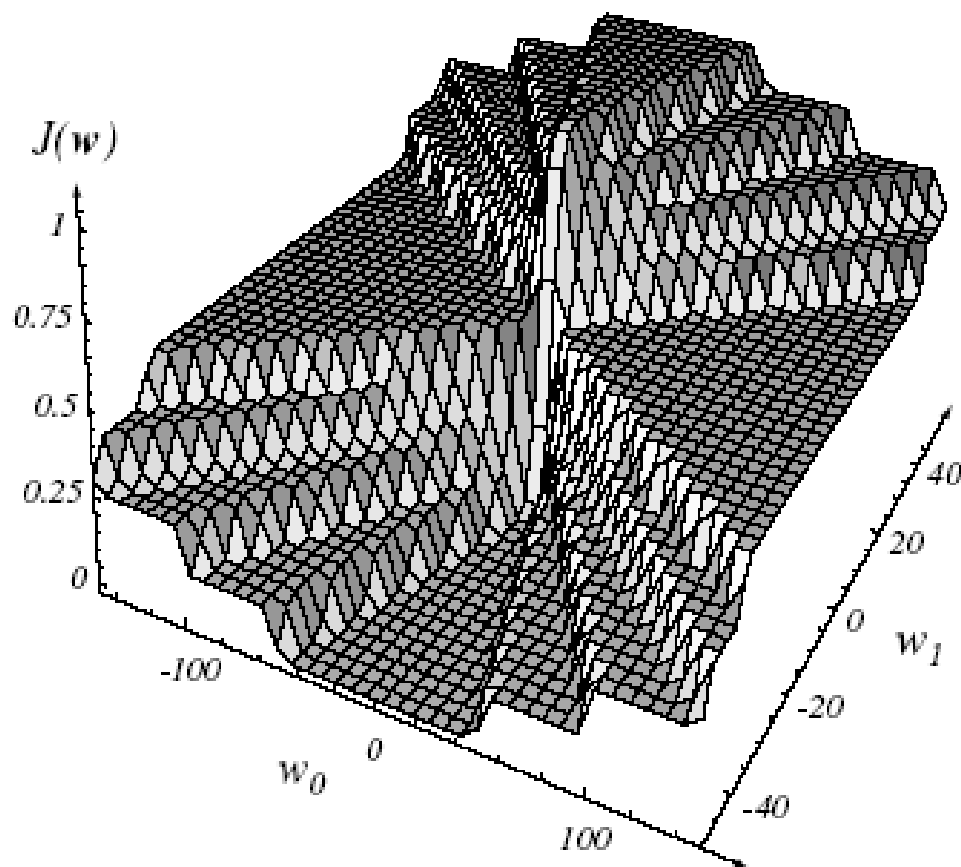
BP算法的一些实用技术

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

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

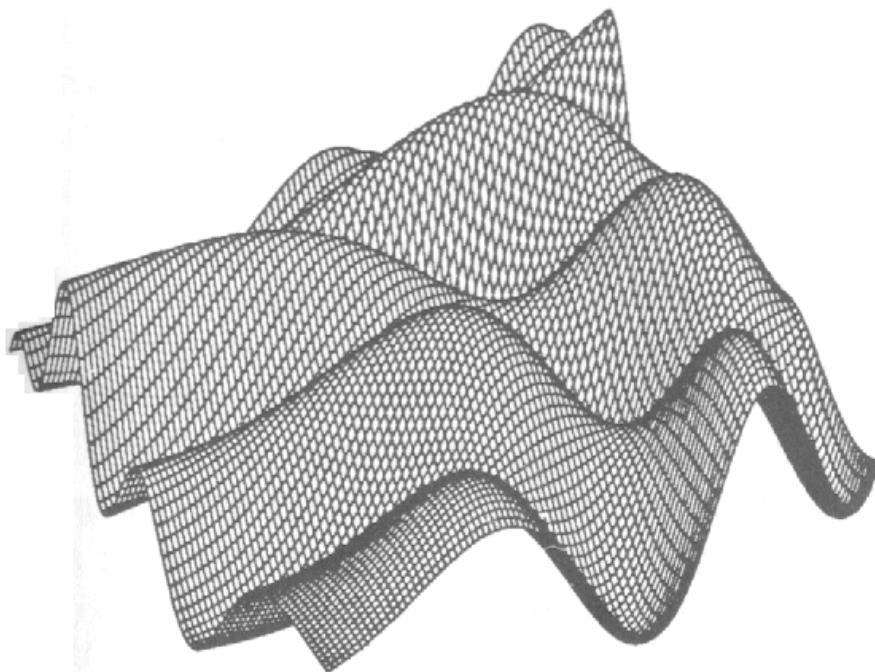
2.3 多层感知器网络存在的问题

1. BP算法的收敛速度一般来说比较慢;



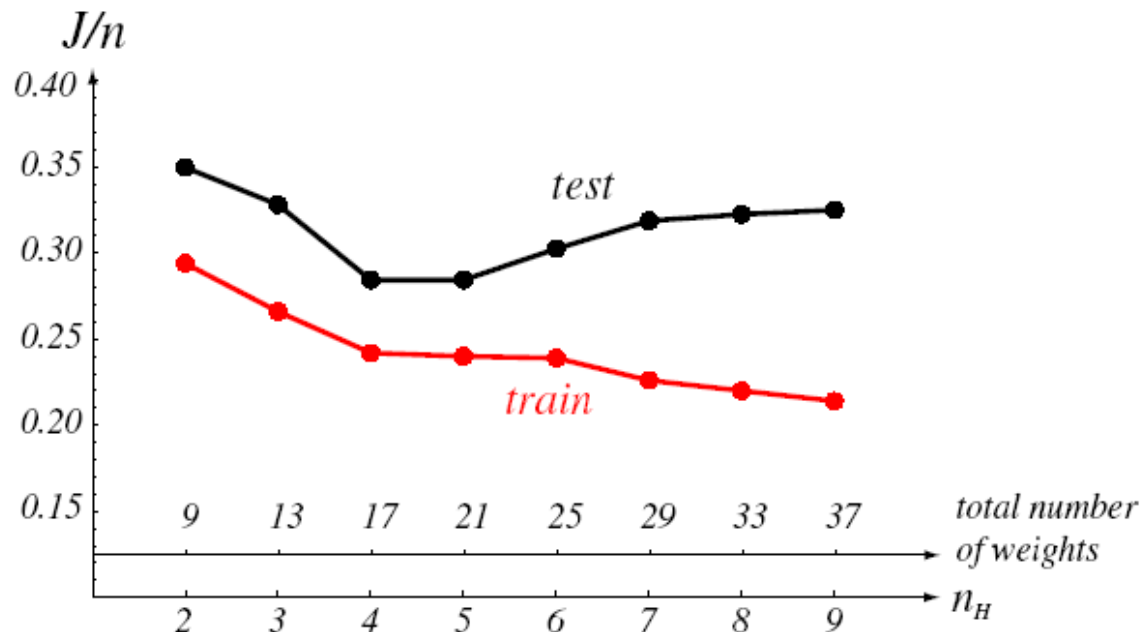
多层感知器网络存在的问题

2. BP算法只能收敛于**局部最优解**, 不能保证收敛于全局最优解;



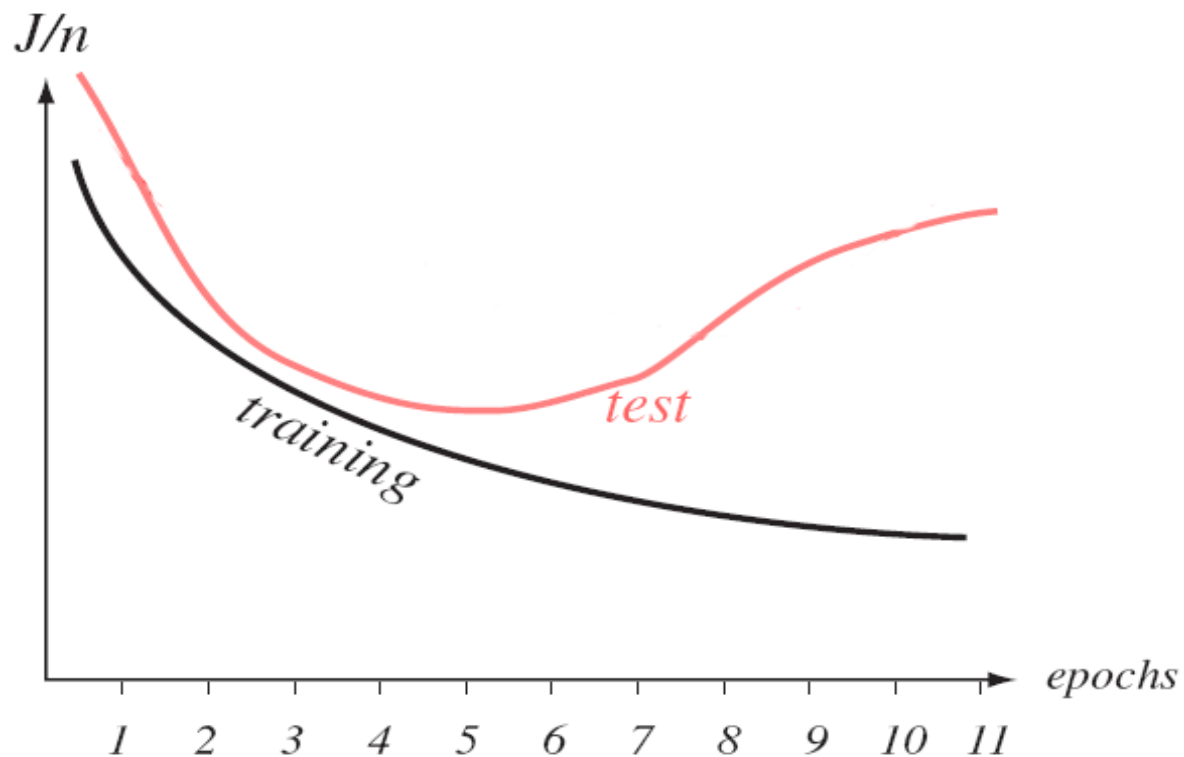
多层感知器网络存在的问题

3. 当隐层元的数量足够多时，网络对训练样本的识别率很高，但对测试样本的识别率有可能很差，即网络的**推广能力**有可能较差。
(Overfitting)



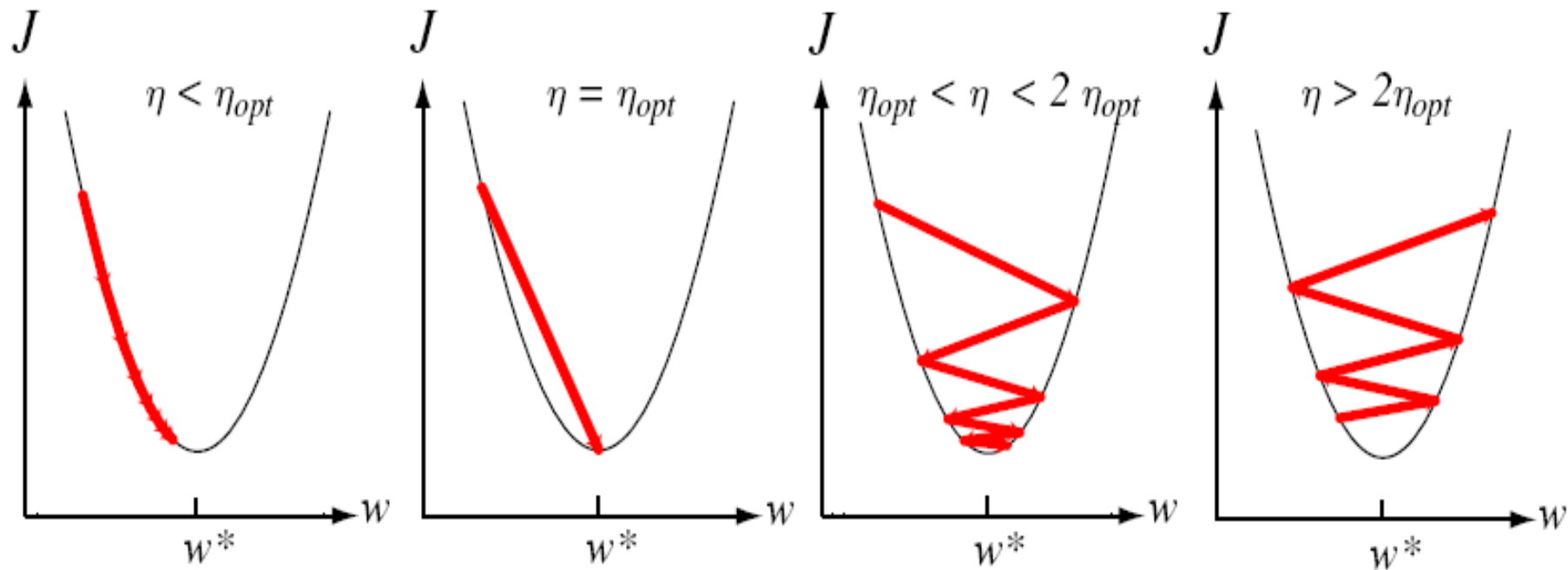
多层感知器网络存在的问题

4. 随着训练周期的增加，在训练样本集上目标函数不断下降，而在测试样本集上反而会上升。
(**Bias-Variance** 理论)



2.4 提高收敛速度的方法

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



梯度下降法

- 目标函数的一阶泰勒级数展开：

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

目标函数增量：

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

使目标函数下降最大：

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

牛顿法

- 目标函数的二阶泰勒级数展开：

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

H是Hessian矩阵，求取目标函数增量的极大值：

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

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

重新思考多层神经网络

- 网络模型：3层 -> 多层
 - 激活函数：ReLU / PReLU, ...
 - 网络参数：Dropout
 - 元学习（Meta Learning）：网络结构搜索
- 学习算法（Mini-batch）
 - AdaGrad
 - RMSProp
 - Adam
- 泛化能力
 - Mini-batch
 - Local minimum
 - Over-fitting