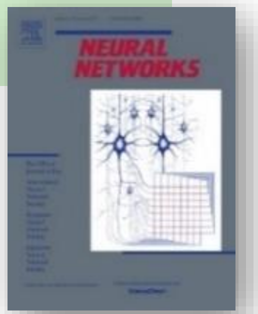
Chapter 6

Multilayer Neural Networks

Artificial Neural Networks (ANN)

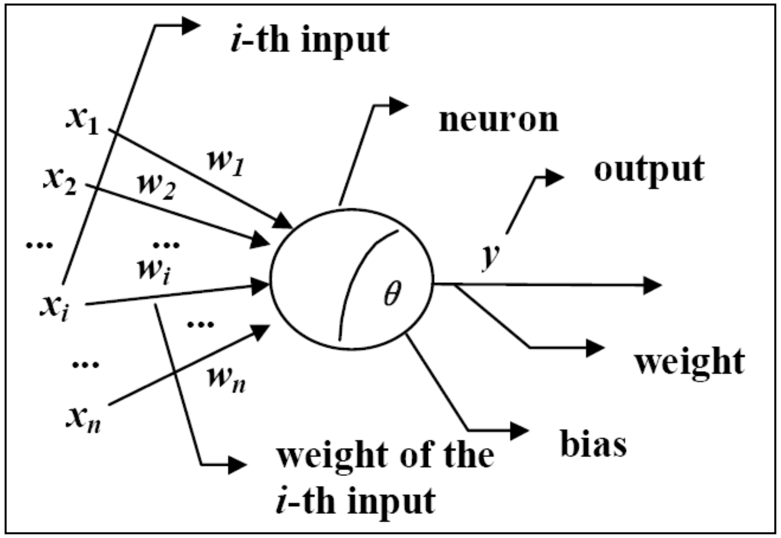
*"Artificial Neural Networks (ANN) are **massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations** which are intended to interact with the objects of the real world in the same way as biological nervous systems do"*

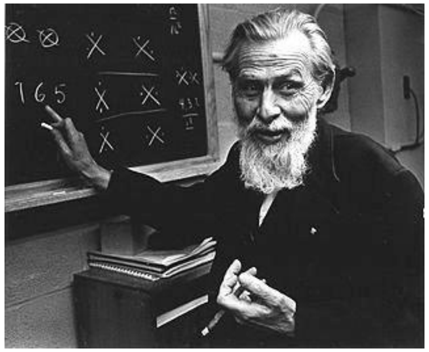


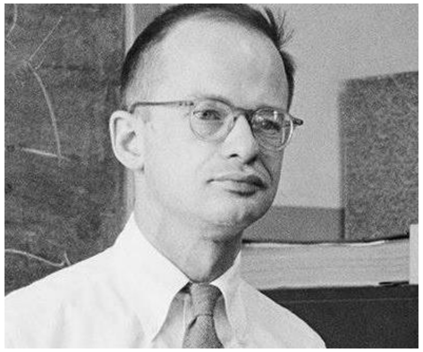
- T. Kohonen. *An introduction to neural computing*. ***Neural Networks**, 1988, 1(1): 3-16.*

人工神经网络是由简单（通常是自适应的）元素及其层次组织组成的大规模并行互连网络，旨在以与生物神经系统相同的方式与现实世界的对象进行交互

The M-P Neuron Model




Warren S. McCulloch
(1898-1969)

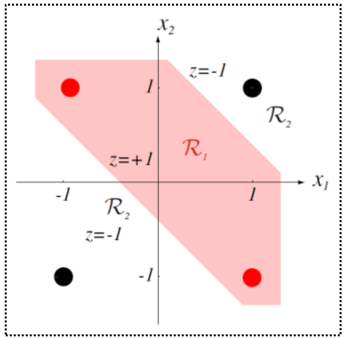

Walter Pitts
(1923-1969)

The M-P neuron model

- **Input**: $x_i \ (1 \le i \le n)$
- **Weight**: $w_i \ (1 \le i \le n)$
- **Bias**: θ
- **Activation function**: $f(\cdot)$
- **Output**: y

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i - \theta\right)$$

The XOR Problem

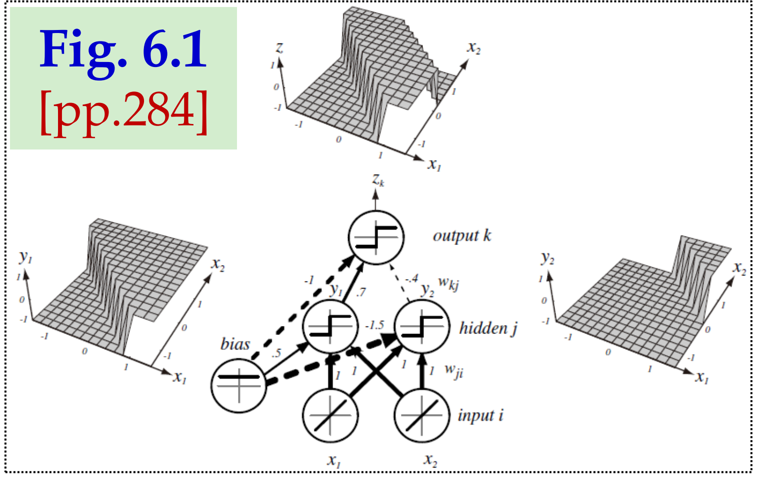


The XOR (“异或”) problem

- **Decide +1 if** $x_1 \cdot x_2 = 1$
- **Decide -1 if** $x_1 \cdot x_2 = -1$

$\Rightarrow$ ***linearly inseparable***

Fig. 6.1
[pp.284]



A 2-2-1 three-layer ANN

$(d = 2;\ n_H = 2)$

$$net_j = \sum_{i=1}^{d} w_{ji} x_i + w_{j0} = \mathbf{w}_j^t \mathbf{x}$$ *input to the hidden unit*

$$y_j = f(net_j)$$ *activation of the hidden unit*

$$f(net) = \text{Sgn}(net) = \begin{cases} 1 \text{ if } net \geq 0 \\ -1 \text{ if } net < 0 \end{cases}$$ *activation function*

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j + w_{k0} = \mathbf{w}_k^t \mathbf{y}$$ *input to the output unit*

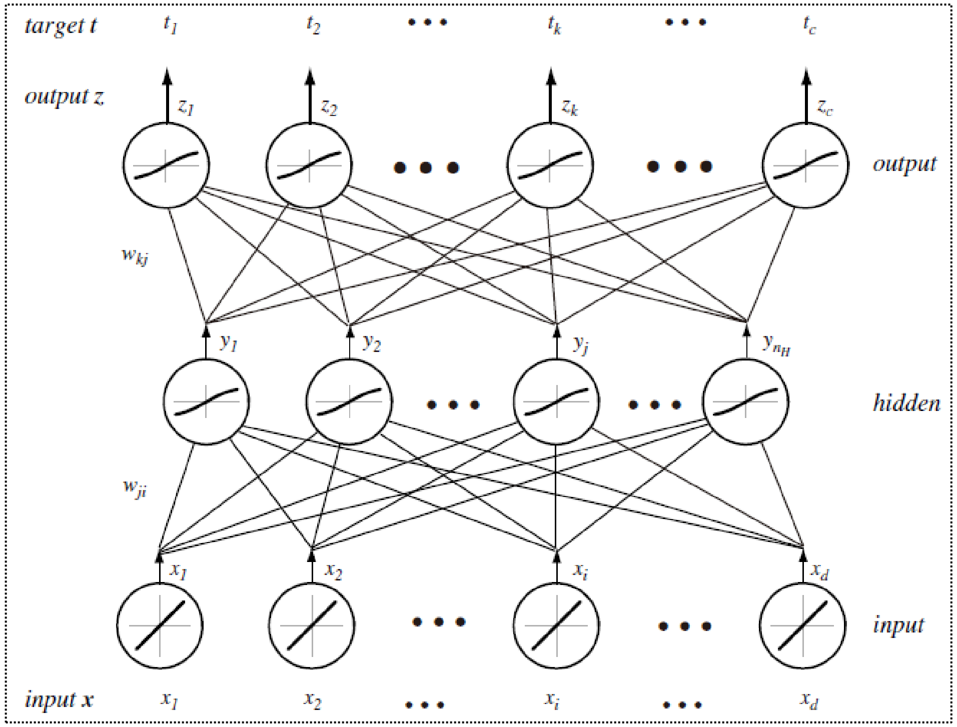
$$z_k = f(net_k)$$ *activation of the output unit*

Feedforward (前馈) Neural Network

Settings A *d*-n_H-*c* fully connected three-layer network

d: # features n_H: # hidden neurons *c*: # output neurons

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Parameters to be learned

w_{ji}: **input-to-hidden** layer weight
(i-th feature to j-th hidden unit)

w_{kj}: **hidden-to-output** layer weight
(j-th hidden to k-th output unit)

$(1 \leq i \leq d;\ 1 \leq j \leq n_H;\ 1 \leq k \leq c)$

$\mathbf{w} = (w_{11}, \ldots, w_{n_H d}, \ldots, w_{c n_H})^t$

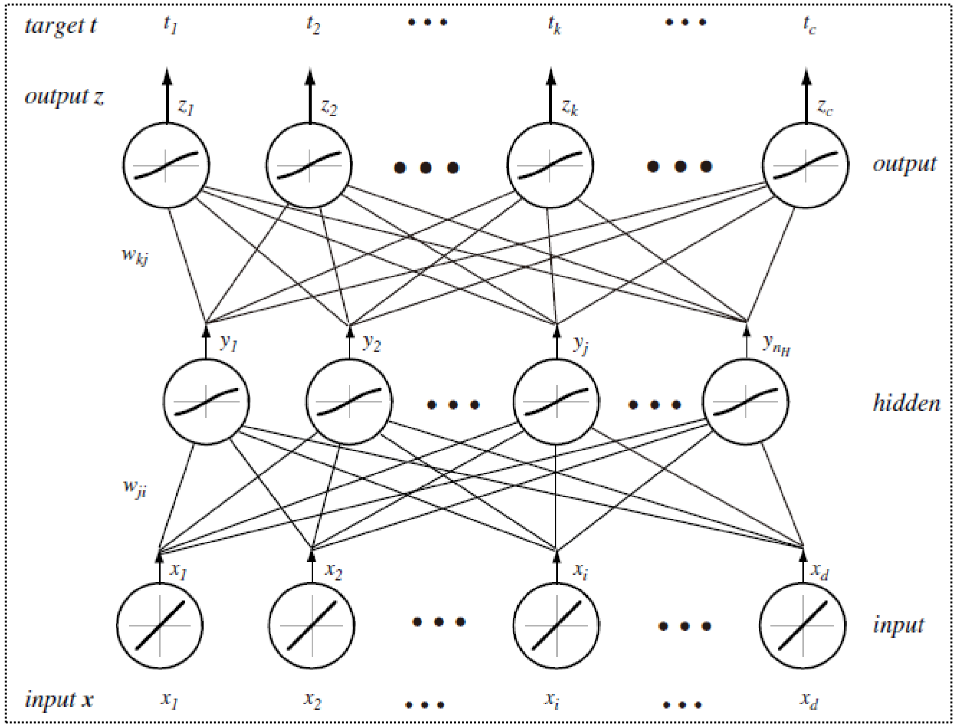
$\#\ parameters\ in\ \mathbf{w} : n_H(d+c)$

Feedforward Neural Network (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Feedforward procedure

$$net_j = \sum_{i=1}^{d} w_{ji} x_i \quad (1 \le j \le n_H)$$

$$y_j = f(net_j) \quad (1 \le j \le n_H)$$

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j \quad (1 \le k \le c)$$

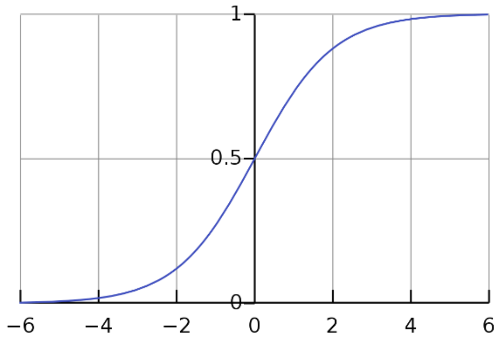
$$z_k = f(net_k) \quad (1 \le k \le c)$$

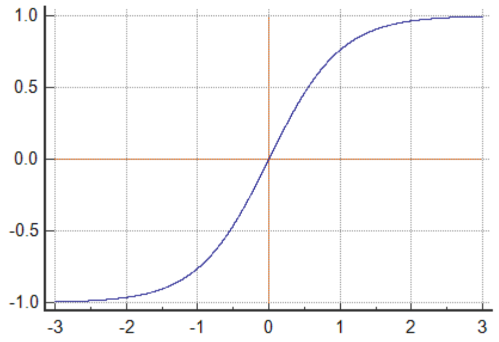
$$g_k(\mathbf{x}) = z_k \quad \textit{(discriminant function)}$$

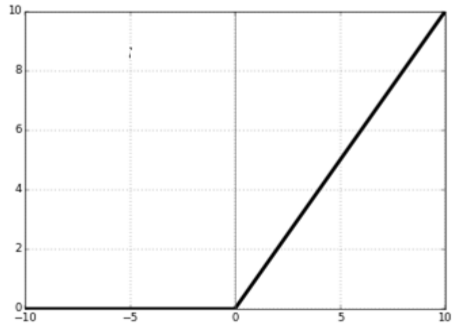
$$= f\left(\sum_{j=1}^{n_H} w_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i\right)\right)$$

Feedforward Neural Network (Cont.)

Activation function







Sigmoid $f(x) = \frac{1}{1+e^{-x}}$

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

ReLU $f(x) = \max(0, x)$

Expressive power of ANN ⇒ ***theoretical ("can", not "how")***

One layer of hidden units with sigmoid activation function is sufficient for approximating any function with finitely many discontinuities to arbitrary precision

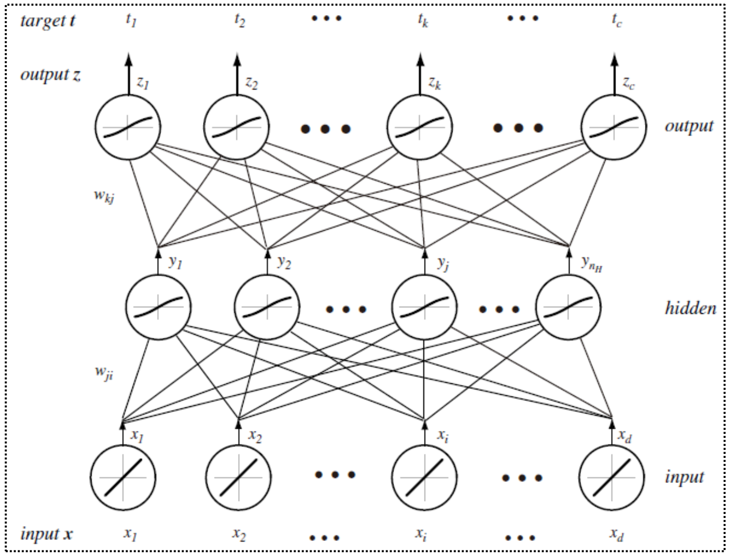
- K. Hornik, M. Stinchcombe, H. L. White. Multilayer feedforward neural networks are universal approximators. ***Neural Networks****, 1989, 2(5): 359-366.*

Backpropagation (反向传播) Algorithm

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output



Criterion function

$$J(\mathbf{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2$$

Gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$$

$$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}} \quad \left(\Delta w_{pq} = -\eta\frac{\partial J}{\partial w_{pq}}\right)$$

P. J. Werbos. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. PhD Thesis, Harvard University, 1974.

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

w_{kj}: **hidden-to-output** layer weight *(j-th hidden to k-th output unit)*

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\sum_{j=1}^{n_H} w_{kj}y_j}{\partial w_{kj}} = -\delta_k y_j$$

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k}\frac{\partial z_k}{\partial net_k}$$

$$= -\frac{\partial\left(\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right)}{\partial z_k}\frac{\partial f(net_k)}{\partial net_k} = (t_k - z_k)f'(net_k)$$

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

Sigmoid

$f' = f(1-f)$

tanh

$f' = 1 - f^2$

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

w_{ji}: **input-to-hidden** layer weight *(i-th feature to j-th hidden unit)*

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial\left(\sum_{i=1}^{d} w_{ji}x_i\right)}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} f'(net_j)x_i = -\delta_j x_i$$

$$\frac{\partial J}{\partial y_j} = \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} = -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)f'(net_k)w_{kj} = -\sum_{k=1}^{c} w_{kj}\delta_k$$

$$\Delta w_{ji} = -\eta\frac{\partial J}{\partial w_{ji}} = \eta\delta_j x_i = -\eta\frac{\partial J}{\partial y_j}f'(net_j)x_i = \eta\left[\sum_{k=1}^{c} w_{kj}\delta_k\right]f'(net_j)x_i$$

Backpropagation Algorithm (Cont.)

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathbf{x} = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t} = (t_1, t_2, \ldots, t_c)^t$: desired output

Forward procedure → **Backpropagation procedure**

$$net_j = \sum_{i=1}^{d} w_{ji} x_i \quad (1 \leq j \leq n_H)$$

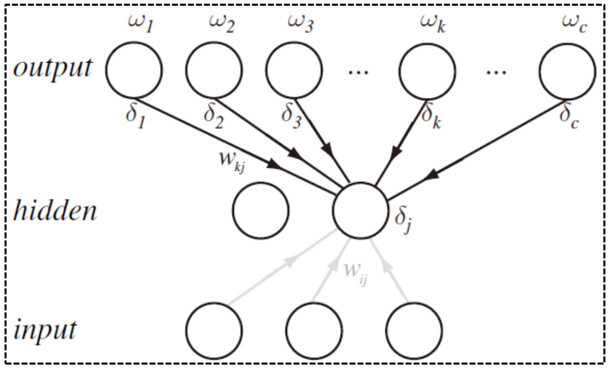
$$y_j = f(net_j) \quad (1 \leq j \leq n_H)$$

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j \quad (1 \leq k \leq c)$$

$$z_k = f(net_k) \quad (1 \leq k \leq c)$$

$$\delta_k = (t_k - z_k) f'(net_k) \quad (1 \leq k \leq c)$$

$$\delta_j = f'(net_j) \left[\sum_{k=1}^{c} w_{kj} \delta_k\right] \quad (1 \leq j \leq n_H)$$



δ_k, δ_j: *neuron unit's* ***sensitivity***

Backpropagation Algorithm (Cont.)

Stochastic training

One pattern is randomly selected from the training set, and the weights are updated by presenting the chosen pattern to the network

```
1. begin initialize $n_H$, $\mathbf{w}$, criterion $\theta$, $\eta$, $m \leftarrow 0$
2.      do $m \leftarrow m + 1$
3.         $\mathbf{x}^m \leftarrow$ randomly chosen training pattern
4.         Invoke the forward and backpropagation procedures on $\mathbf{x}^m$
           to obtain $\delta_k$ $(1 \leq k \leq c)$, $y_j$ and $\delta_j$ $(1 \leq j \leq n_H)$
5.         $w_{ji} \leftarrow w_{ji} + \eta\delta_j x_i$;   $w_{kj} \leftarrow w_{kj} + \eta\delta_k y_j$
6.      until $\|\nabla J(\mathbf{w})\| \leq \theta$
7.    return $\mathbf{w}$
8. end
```

Stochastic backpropagation

Backpropagation Algorithm (Cont.)

Batch training

All patterns in the training set are presented to the network at once, and the weights are updated in ***one epoch***

Settings A d-n_H-c fully connected three-layer network

d: # features n_H: # hidden neurons c: # output neurons **w: weights**

$\mathcal{D} = \{(\mathbf{x}^m, \mathbf{t}^m) \mid 1 \leq m \leq n\}$: training set consisting of n patterns

$\mathbf{x}^m = (x_1, x_2, \ldots, x_d)^t$: training pattern $\mathbf{t}^m = (t_1, t_2, \ldots, t_c)^t$: desired output

(WLOG, the superscript m is ignored for elements of $\mathbf{x}^m$ *and* $\mathbf{t}^m$*)*

$$J(\mathbf{w}) = \frac{1}{2}\|\mathbf{t} - \mathbf{z}\|^2 \Longrightarrow J(\mathbf{w}) = \frac{1}{2}\sum_{m=1}^{n} \|\mathbf{t}^m - \mathbf{z}^m\|^2$$

Backpropagation Algorithm (Cont.)

```
1. begin initialize $n_H$, $\mathbf{w}$, criterion $\theta$, $\eta$, $r \leftarrow 0$
2.     do $r \leftarrow r + 1$ (increment epoch)
3.         $m \leftarrow 0$; $\Delta w_{ji} \leftarrow 0$; $\Delta w_{kj} \leftarrow 0$
4.         do $m \leftarrow m + 1$
5.             $\mathbf{x}^m \leftarrow$ the $m$-th pattern in the training set
6.             Invoke the forward and backpropagation procedures
               on $\mathbf{x}^m$ to obtain $\delta_k$ $(1 \le k \le c)$, $y_j$ and $\delta_j$ $(1 \le j \le n_H)$
7.             $\Delta w_{ji} \leftarrow \Delta w_{ji} + \eta\delta_j x_i$;  $\Delta w_{kj} \leftarrow \Delta w_{kj} + \eta\delta_k y_j$
8.         until $m = n$
9.         $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$; $w_{kj} \leftarrow w_{kj} + \Delta w_{kj}$
10.    until $\|\nabla J(\mathbf{w})\| \le \theta$
11.    return w
12. end
```

Batch backpropagation

Summary

- Artificial neural networks
  - The M-P neuron model
  - Feedforward neural network
  - Expressive power of ANN
- Backpropagation algorithm
  - Criterion function, activation function
  - Feedforward procedure
  - Backpropagation procedure
  - Stochastic/Batch mode