Ch 08. Multilayer Neural Network

What is Neural Network?

- **Neural Network(NN,神经网络)**
  as known as **artificial neural network** (人工神经网络)

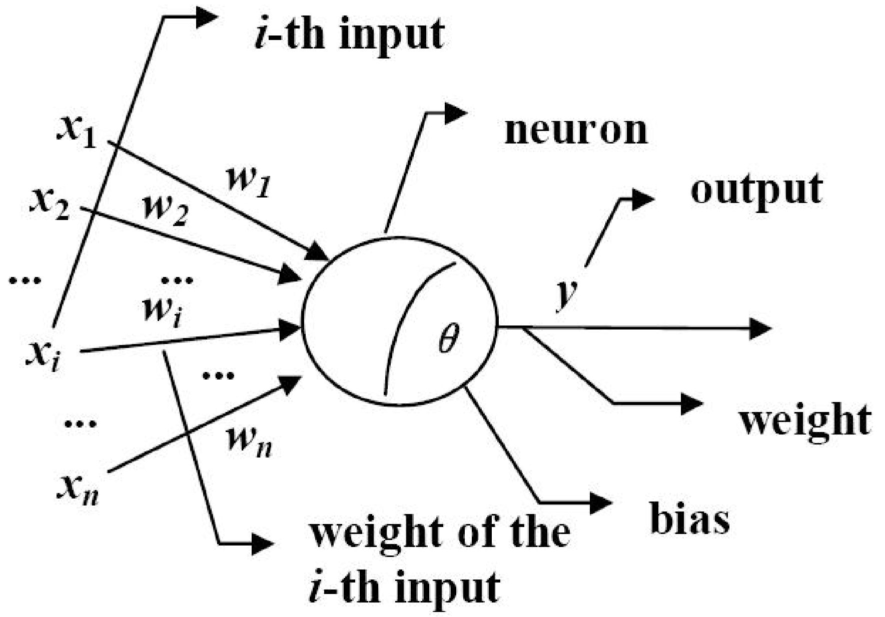
  “neural networks are massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do”

  [T. Kohonen, NN88]

What is Neural Network?

- The basic components of a neural network are the **neuron** (神经元) and the **weight** (权值)
  - **The neurons M-P model**
    - Input: x_i
    - Weight: w_i
    - Bias (权值): θ
    - Activation function (激活函数): $f(\cdot)$
    - Output: y

$$y = f\Big(\sum_i x_i w_i - \theta\Big)$$

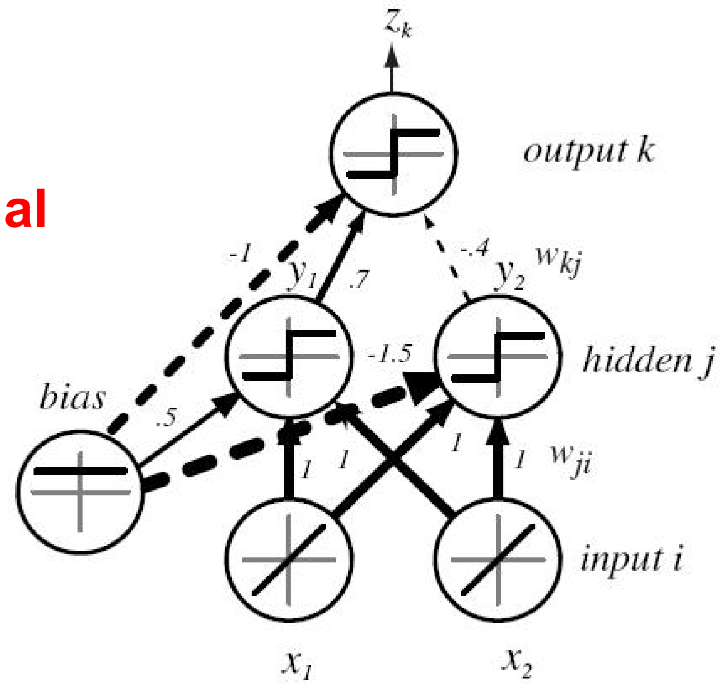


Neuron is sometimes called “unit“ (单元)

What is Neural Network?

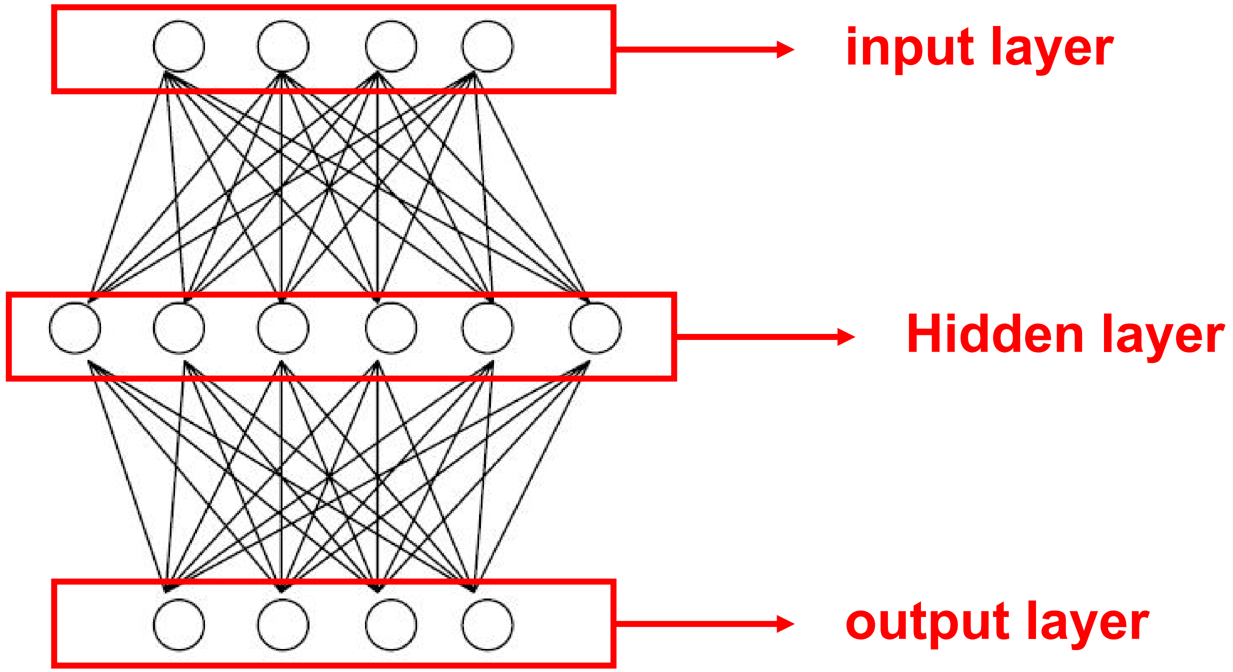
- The basic components of a neural network are the **neuron** (神经元) and the **weight** (权值)
  - **Neurons are connected by weights**
  - The knowledge acquired by neural network learning is encoded by weights and biases in neurons, that is, the neural network training process determines all weights and biases through training data

The bias is sometimes represented by a special neuron: the bias unit



Feedforward Neural Network

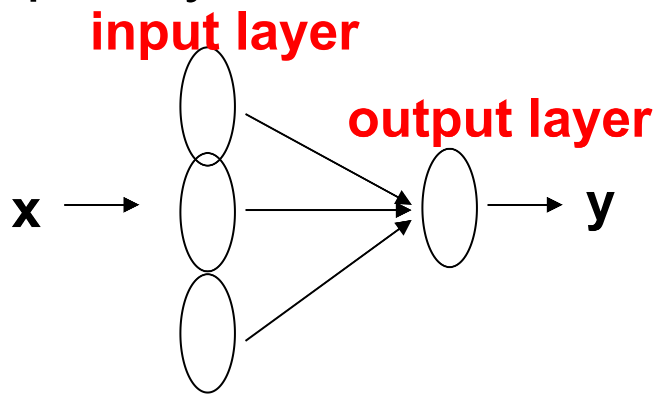
- Feedforward neural network (前馈神经网络)
- Generally, feedforward neural network refers to the neural network that each neuron is only connected to the next layer of neurons



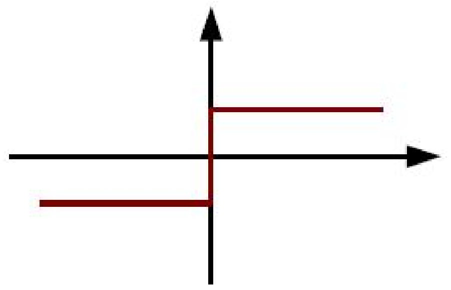
three-layer feedforward neural network

Simple Perceptron

- **Simple perceptron** (简单感知器) refers to a two-layer feedforward neural network consisting only of input layer and output layer
- **Suppose:**
  - The activation function of the output unit is y=x
  - The activation function of the output unit is a symbolic function



$$f(net) = \mathrm{sgn}(net) = \begin{cases} 1 & \text{if } net \geq 0 \\ -1 & \text{if } net < 0 \end{cases}$$



$$y = \mathrm{sgn}\left(\sum_{i=1}^{d} w_i x_i + w_0\right) = \mathrm{sgn}(\mathbf{w}^t\mathbf{x} + w_0)$$

Simple Perceptron

- Training sample with category label

$$\left(\mathbf{x}^{(p)}, \mathrm{t}^{(p)}\right) \qquad 1 \le p \le n$$

$\mathrm{t}^{(p)}$ is the output (+1 or -1) that you want when the input is $\mathbf{x}^{(p)}$

- **Augmented** eigenvectors and weight spaces
  - Suppose in original d-dimensional space

    $$\mathbf{x} = (x_1, \ldots, x_d)^{\,t} \qquad \mathbf{w} = (w_1, \ldots, w_d)^{\,t}$$

    Define x_0 as constant 1, then $\sum_{i=1}^{d} w_i x_i + w_0$ can be written as $\sum_{i=0}^{d} w_i x_i$
  - $(1, x_1, \ldots, x_d)^{\,t}$ and $(w_0, w_1, \ldots, w_d)^{\,t}$ are vectors of d+1 dimensional augmented space
  - This is equivalent to adding a special bias unit

Simple Perceptron

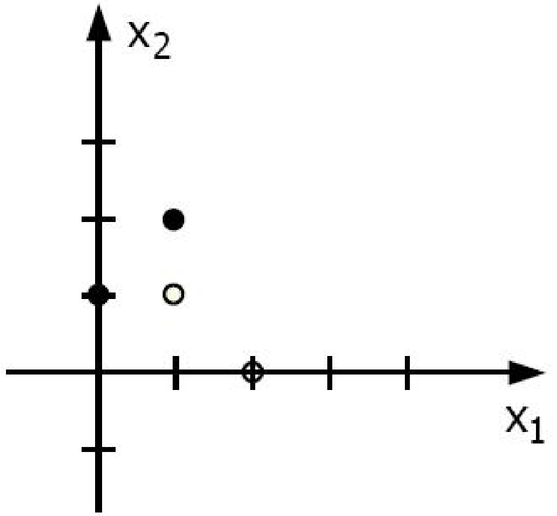
- Perceptron learning process

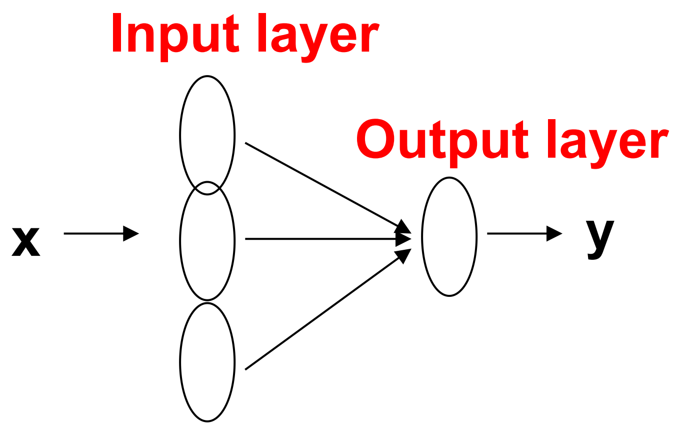
```
Begin
    Initialize network with arbitrary weights;  (WLOG, initialize them to 0)
    Loop
        If $y^{(p)} = t^{(p)}$ for all p, then exit loop;
        For every p, for every i (*in order* or *at random*):
            $\Delta w_i = \begin{cases} x_i^{(p)} t^{(p)} & y^{(p)} \neq t^{(p)} \\ 0 & y^{(p)} = t^{(p)} \end{cases}$
            $w_i = w_i + \Delta w_i$
    End Loop
End
```

Simple Perceptron

- Example
  - training dataset

$x_0^{(p)}$	$x_1^{(p)}$	$x_2^{(p)}$	$t^{(p)}$
1	1	1	1
1	2	0	1
1	0	1	-1
1	1	2	-1





$$y = \begin{cases} +1 & \text{sum} \geq 0 \\ -1 & \text{sum} < 0 \end{cases}$$

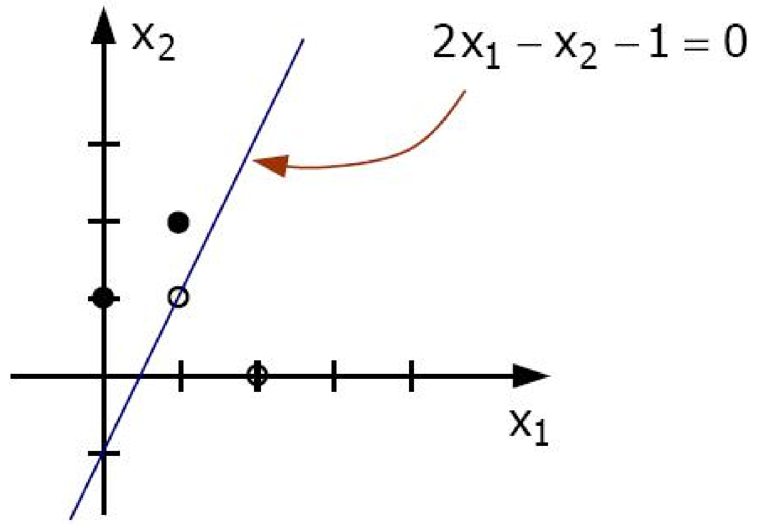
Simple Perceptron

- Example
  - The initial value of w is $w_0 = w_1 = w_2 = 0$

	$\mathbf{w}$			$\mathbf{x}^{(p)}$		$t^{(p)}$	$y^{(p)}$		$\Delta\mathbf{w}$	
0	0	0	1	1	1	1	1	0	0	0
0	0	0	1	2	0	1	1	0	0	0
0	0	0	1	0	1	-1	1	-1	0	-1
-1	0	-1	1	1	2	-1	-1	0	0	0
-1	0	-1	1	1	1	1	-1	1	1	1
0	1	0	1	2	0	1	1	0	0	0
0	1	0	1	0	1	-1	1	-1	0	-1
-1	1	-1	1	1	2	-1	-1	0	0	0
-1	1	-1	1	1	1	1	-1	1	1	1
0	2	0	1	2	0	1	1	0	0	0
0	2	0	1	0	1	-1	1	-1	0	-1
-1	2	-1	1	1	2	-1	-1	0	0	0
-1	2	-1	1	1	1	1	1	0	0	0
-1	2	-1	1	2	0	1	1	0	0	0
-1	2	-1	1	0	1	-1	-1	0	0	0

Simple Perceptron

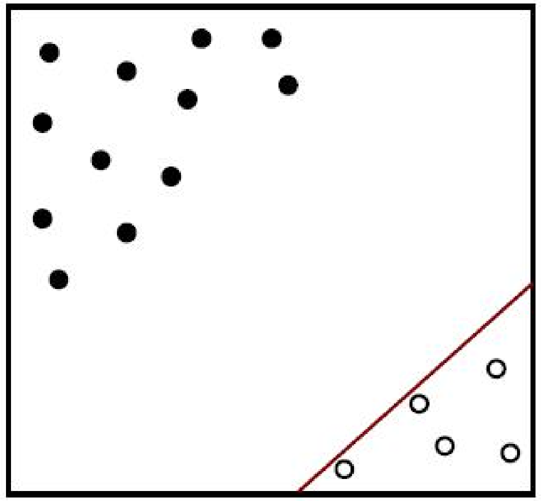
- Example
  - The convergent solution of w is $w_0 = -1,\ w_1 = 2, w_2 = -1$



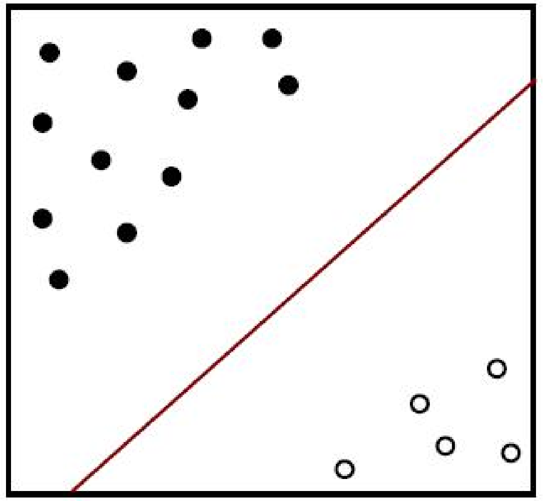
Simple Perceptron

- The shortage of perceptron
  - Can only deal with linearly separable case
  - Even in the case of linear separability, the optimal solution may not be given

perceptron solution

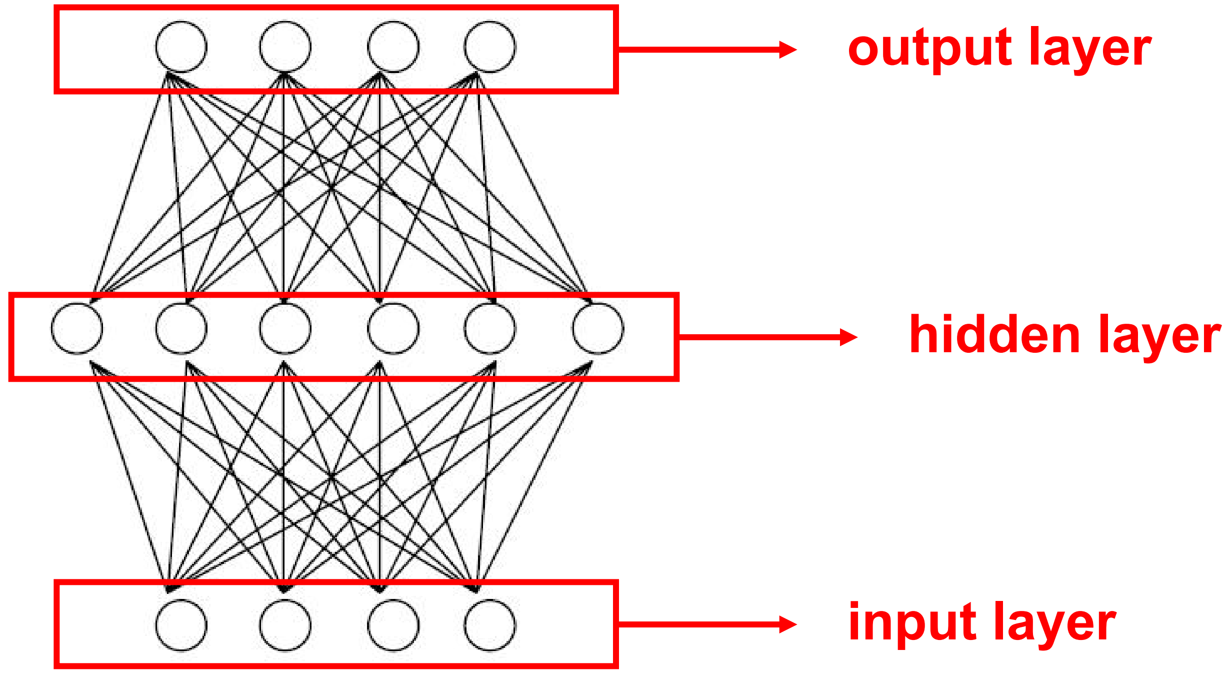


optimal solution



Multilayer Feedforward Neural Network

- The feedforward neural network containing at least one hidden layer



three-layer feedforward neural network

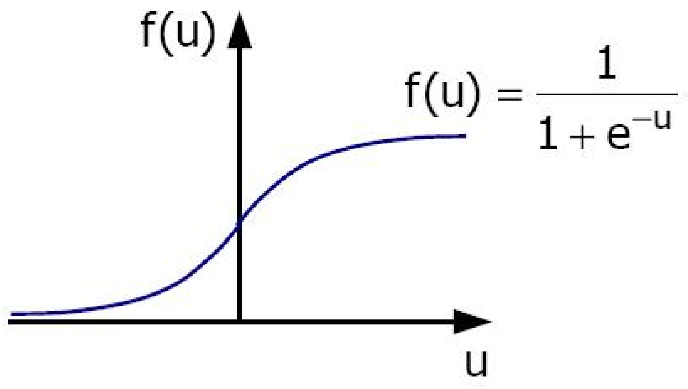
General Feedforward Computation

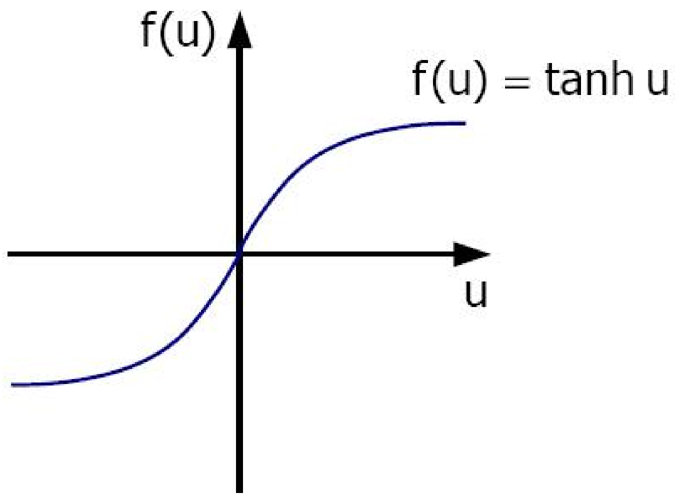
- d input units (generally corresponding to feature dimension d)
- n_H hidden units (n_H is determined by the designer according to the specific problem)
- c output units (generally corresponding to c categories)
- The signal generated by each output unit is the discriminant function of the corresponding category

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

The Representation Ability of Multilayer Networks

- sigmoid function (S形函数)





- Universal approximation (一般逼近) theory

[Cybenko 1989; Hornik et al. 1989]

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

The Representation Ability of Multilayer Networks

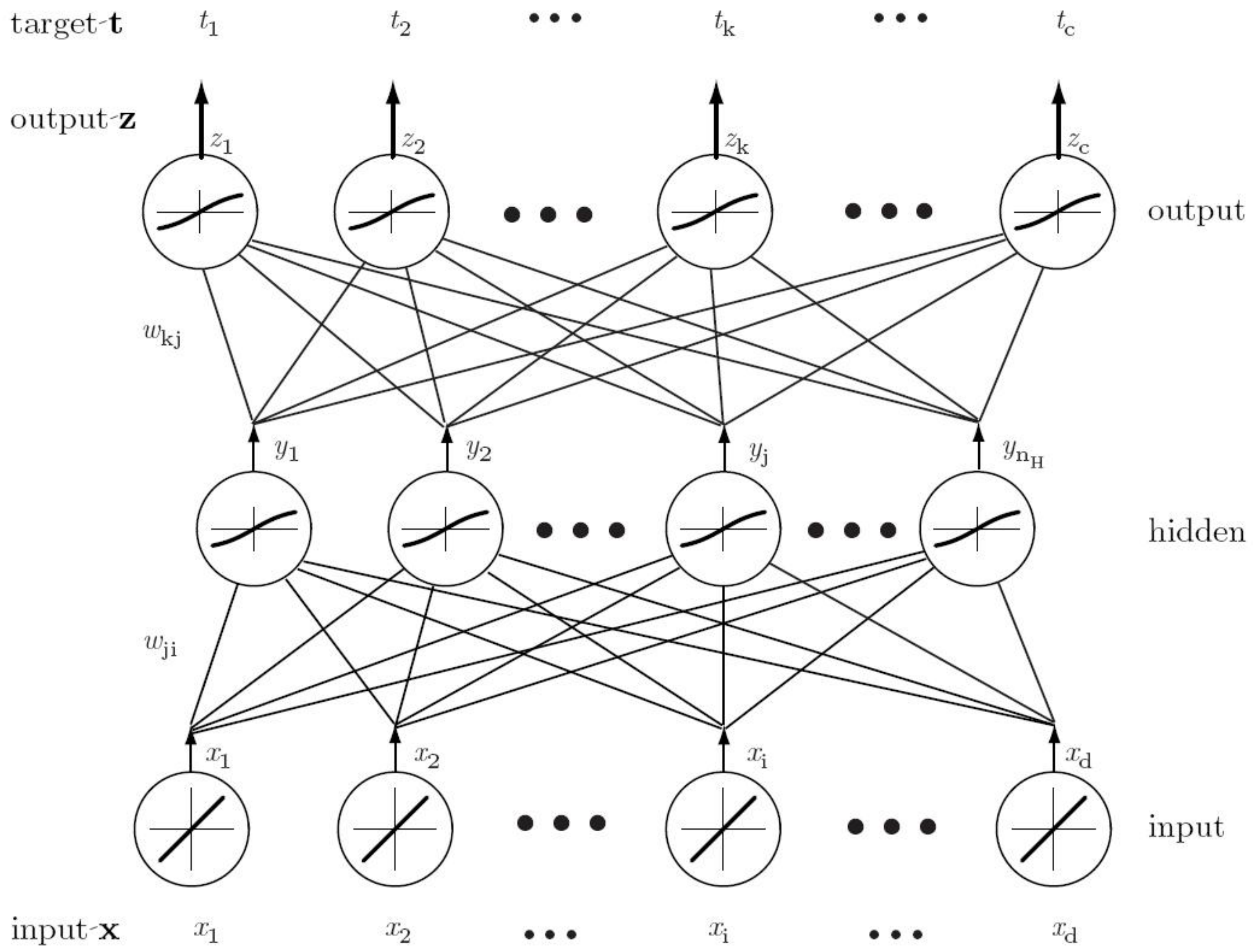
- Any discriminant function can be implemented by a three-layer neural network !
- But……
  - It only makes theoretical sense that a multilayer network can achieve arbitrary decisions
  - The theory tells us "Can" but not "How“
  - The structure design and training methods of the network require specific analysis of specific problems

Neural Network Learning

- Neural network learning algorithm
  - **backpropagation algorithm** (反向传播算法) **[Rumelhart et al. 1985]**
  - Similar solutions have been proposed independently by others **[Werbos 1974; Parker 1985; Le Cun 1985]**
  - Complex decision surfaces can be learned
  - The optimal solution cannot be guaranteed, but a better solution can be obtained

Neural Network Learning

- Three-layer neural network



Backpropagation Algorithm

- Criterion function: the squares sum of the difference between the output value z and the true value t

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

- Gradient descent algorithm

$$\Delta\mathbf{w} =- \eta \frac{\partial J}{\partial \mathbf{w}} \qquad \Delta w_{mn} =- \eta \frac{\partial J}{\partial w_{mn}}$$

$$\mathbf{w}(m+1) = \mathbf{w}(m) + \Delta\mathbf{w}(m)$$

Backpropagation Algorithm

- Weight update from hidden layer to output layer

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = -\delta_k \frac{\partial net_k}{\partial w_{kj}}$$

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

sensitivity (敏感度)

$$\frac{\partial net_k}{\partial w_{kj}} = y_j$$

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

Backpropagation Algorithm

- Weight update from input layer to hidden layer

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

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

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j}$$

$$= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k}\frac{\partial net_k}{\partial y_j}$$

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

δ_k

Backpropagation Algorithm

- Weight update from input layer to hidden layer

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

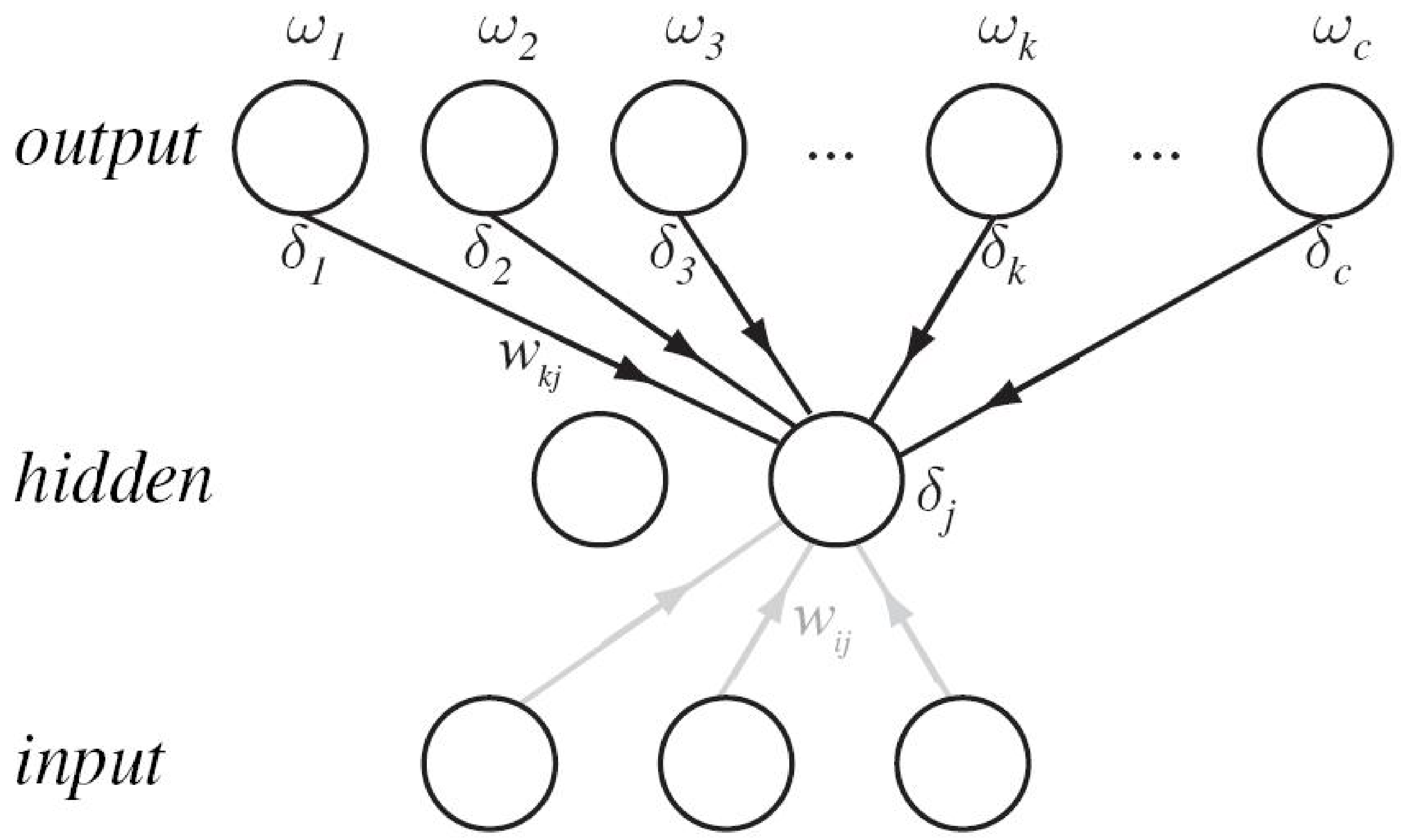
$$\delta_j \equiv f'(net_j)\sum_{k=1}^{c} w_{kj}\delta_k$$

The sensitivity of the output unit is propagated back to the hidden unit

$$\boxed{\Delta w_{ji} = \eta x_i \delta_j = \eta x_i f'(net_j)\sum_{k=1}^{c} w_{kj}\,\delta_k}$$

Backpropagation Algorithm

- **Backpropagation of sensitivity**



Backpropagation Algorithm

- **Training protocol**

  How to adjust weights based on training samples during neural network training

  - **Stochastic training (随机训练)**

    The pattern is randomly selected from the training set, and the weight is updated once each pattern is entered

  - **Batch training (成批训练)**

    All patterns are sent to the network all at once, and then the weight is updated once

  - **Online training (在线训练)**

    Each pattern is provided only once, each pattern is provided, and the weight is updated once

Backpropagation Algorithm

Stochastic backpropagation

```
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

■ Batch backpropagation

```
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 $\mathbf{w}$
12. end
```

epoch (回合): All patterns are provided once called one epoch

Batch backpropagation

Backpropagation Algorithm

- **Online backpropagation**

```
1 begin initialize network topology (# hidden units), $\mathbf{w}$, criterion $\theta$, $\eta$, $m \leftarrow 0$
2     do $m \leftarrow m + 1$
3         $\mathbf{x}^m \leftarrow$ sequential selection of training patterns
4         $w_{ij} \leftarrow w_{ij} + \eta \delta_j x_i$; $w_{jk} \leftarrow w_{jk} + \eta \delta_k y_j$
5     until $||\nabla J(\mathbf{w})|| \leqslant \theta$
6 return $\mathbf{w}$
7 end
```

Online backpropagation