Chapter 4 Multiple Feedforward Neural Network

4.1 Error Backpropagation Algorithm

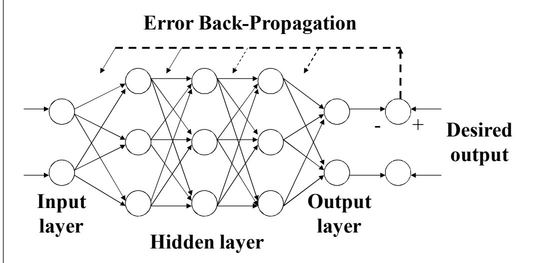
4.2 Improving on BP Algorithm

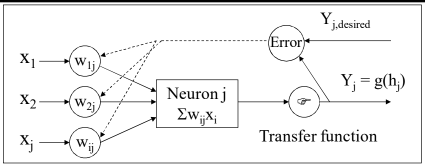
4.3 Practical aspects of neural computing

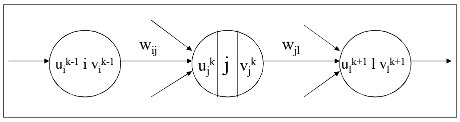
4.4 Introduction to special neural network architectures

4.5 Applications

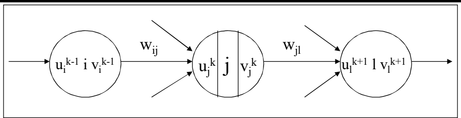
4.1 Error Backpropagation Algorithm







Assumption:
Transfer function : Sigmoid
Input sample (x_k, y_k) , k=1,2,…,N
u_j^k : Total input to j^{th} neuron on the k^{th} layer
v_j^k : Output from j^{th} neuron on the k^{th} layer
w_{ij} : interconnecting weight between the i^{th} node of layer k-1 and the j^{th} node of layer k



$$u_j^k = \sum_i w_{ij} v_i^{k-1}$$

$$v_j^k = f(u_j^k)$$

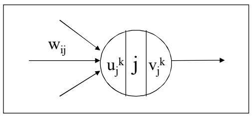
$$E = \frac{1}{2}\sum_j (y_j - v_j^m)^2$$

Where, y_j is the desired output

min E(w, x)

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} \qquad \Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

η is the learning rate



$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial u_j^k}\frac{\partial u_j^k}{\partial w_{ij}} = \frac{\partial E}{\partial u_j^k}\frac{\partial}{\partial w_{ij}}\left(\sum_i w_{ij} v_i^{k-1}\right) = \frac{\partial E}{\partial u_j^k} v_i^{k-1}$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial u_j^k} v_i^{k-1}$$

Assuming:

$$\delta_j^k = -\frac{\partial E}{\partial u_j^k} = -\frac{\partial E}{\partial v_j^k}\frac{\partial v_j^k}{\partial u_j^k} = -\frac{\partial E}{\partial v_j^k} f'(u_j^k)$$

Learning formula

$$\Delta w_{ij} = \eta \delta_j^k v_i^{k-1} = -\eta \frac{\partial E}{\partial v_j^k} v_i^{k-1} f'(u_j^k)$$

Learning formula

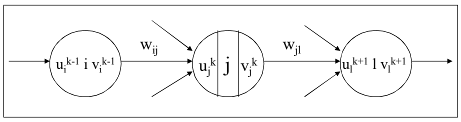
$$\Delta w_{ij} = \eta \delta_j^k v_i^{k-1} = -\eta \frac{\partial E}{\partial v_j^k} v_i^{k-1} f'(u_j^k)$$

Sigmoid function:

$$f(u_j^k) = \frac{1}{1+\exp(-u_j^k)}$$

$$f'(u_j^k) = f(u_j^k)[1 - f(u_j^k)] = v_j^k(1 - v_j^k)$$

$$\Delta w_{ij} = \eta \delta_j^k v_i^{k-1} = -\eta \frac{\partial E}{\partial v_j^k} v_i^{k-1} f'(u_j^k)$$



1. j[th] node on the output layer m

$$E = \frac{1}{2}\sum_j (y_j - v_j^m)^2 \qquad \frac{\partial E}{\partial v_j^k} = -(y_j - v_j^m)$$

$$\delta_j^m = -\frac{\partial E}{\partial v_j^k} f'(u_j^m) = v_j^m(1-v_j^m)(y_j - v_j^m)$$

2. j[th] node on the hidden layer k

$$\frac{\partial E}{\partial v_j^k} = \sum \frac{\partial E}{\partial u_l^{k+1}} \frac{\partial u_l^{k+1}}{\partial v_j^k}$$

$$\frac{\partial E}{\partial u_l^{k+1}} = -\delta_l^{k+1}, \qquad \frac{\partial u_l^{k+1}}{\partial v_j^k} = \sum_l w_{jl}$$

$$\delta_j^k = v_j^k(1-v_j^k)\sum_l w_{jl}\delta_l^{k+1}$$

BP algorithm:

$$\Delta w_{ij} = \eta \delta_j^k v_i^{k-1} = -\eta \frac{\partial E}{\partial v_j^k} v_i^{k-1} f'(u_j^k)$$

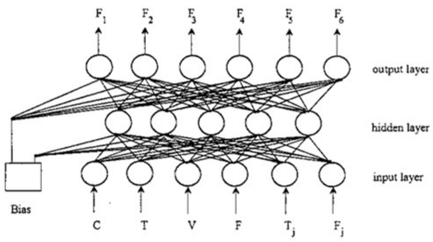
$$\delta_j^m = v_j^m(1-v_j^m)(y_j - v_j^m)$$

$$\delta_j^k = v_j^k(1-v_j^k)\sum_l w_{jl}\delta_l^{k+1}$$

$$\mathbf{w_{ij,new} = w_{ij,old} + \Delta w_{ij}}$$

(New weight factor) = (Old weight factor) + (Learning rate) × (Gradient-descent correction term) × (Input term)

NN with thresholds



$$v_{bias}^{k-1} = 1 \qquad \Delta w_{j,bias}^k = \eta \delta_j^k$$

$$w_{j,new-bias}^k = w_{j,old-bias}^k + \eta \delta_j^k$$

Calculating steps of BP algorithm

1. Randomly assign values for weights w_{ij}

2. Repeat the following procedures until the NN is converged

(1) forward activation flow to calculate v_j^k on each layer
backward error flow to calculate δ_j^k through the network

(2) Adjust weight factors $\mathbf{w_{ij,new} = w_{ij,old} + \Delta w_{ij}}$

An illustrative example: fault diagnosis

Input and output for fault-diagnosis network

Input vector	Output vector
I_1: reactor inlet temperature, ^{0}F	**c_1 : low conversion**
I_2: reactor inlet pressure, psia	**c_2 : low catalyst selectivity**
I_3: feed flow rate, 1b/min	**c_3 : catalyst sintering**

The desired output,d_k from the neural network is Boolean:
0 indicates that no operational fault exist,
1 indicates that a fault does exist.

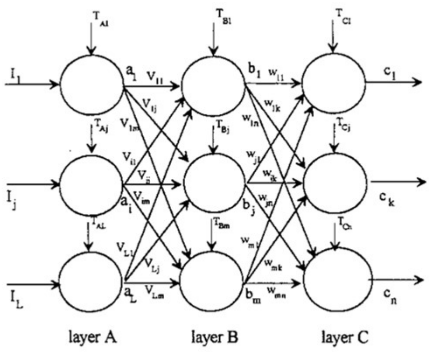
The actual output from the neural network is a numeric value between 0 and 1, and can be viewed almost the "probability" that a given input produce the operational fault.
0 = the fault definitely will not occur,
1 = the fault definitely will occur.

Specific conditions:

Input:	**I_1=300/1000 ^{0}F=0.3**	**Output:**	**d_1=1 (low conversion)**
	I_2=100/1000 psia=0.1		**d_2=0 (no problem)**
	I_3=200/1000 lb/min=0.2		**d_3=0 (no problem)**

In neural network training, we recommend *normalizing* the input and output values to a finite range, such as [0,1] or [-1,1].

A three-layer perceptron neural network



Attempts to properly map given inputs with desired outputs by minimizing an error function.

The total mean-square error function:

$$E = \sum_k \varepsilon_k^2 = \sum_k (d_k - c_k)$$

***here*,**

ε_k is the output error vector from the k^{th} node on the output layer

d_k is the desired output value

c_k is the calculated value

Step 1: Randomly specify numerical values for all weight factors (v_{ij}'s and w_{ik}'s) within the interval [-1,+1]. Likewise, assign internal threshold values (T_{ai}, T_{Bj}, T_{Ck}) for every node, also between –1 and +1.

Example: To use computer, let layer A=layer 1, layer B=layer 2, and layer C=layer 3.

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

For the internal threshold values (T_{1i}, T_{2j}, T_{3k}), we set T_{1i}=0 because usually there is no threshold for every node on the first layer

$$[T_{ij}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.5 & 0.0 & -0.5 \\ 0.0 & 0.5 & -0.5 \end{bmatrix}$$

Step 2: **Introduce the input I_i into the neural network and calculate all outputs from the first layer, using the standard sigmoid function.**

$$x_i = I_i - T_{Ai}$$
$$f(x_i) = \frac{1}{1+e^{-x_i}} = a_i$$

Example: **We introduce the input vector into the neural network and calculate outputs from layer 1:**

$$x_1 = I_1 - T_{11} = 0.3 - 0 = 0.3$$
$$x_2 = I_2 - T_{12} = 0.1 - 0 = 0.1$$
$$x_3 = I_3 - T_{13} = 0.2 - 0 = 0.2$$

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 0.57444$$
$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 0.52498$$
$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 0.54983$$

Note that had we *not* normalized our input values to the ranger [0,1], we would have the following results:

$$x_1 = I_1 - T_{11} = 300 - 0 = 300$$
$$x_2 = I_2 - T_{12} = 100 - 0 = 100$$
$$x_3 = I_3 - T_{13} = 200 - 0 = 200$$

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 1.0$$
$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 1.0$$
$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 1.0$$

If I_1=300 ^{0}F, then a_1=1. If I_1=150 ^{0}F, then a_1=1 again.

That is, the sigmoid transfer function is *insenstitive* to changes in the input when I_1>2.

Step 3: **Given the output layer A, calculate the output from layer B, using the equation:**

$$b_j = f(\sum_{i=1}^{L}(v_{ij}a_i) - T_{Bj})$$ **where *f ( )* is the same sigmoid function.**

Example: **We calculate the output from each node in layer 2:**

$$\begin{aligned} b_1 &= f(v_{11}a_1 + v_{21}a_2 + v_{31}a_3 - T_{21}) \\ &= f(-1.0*0.57444 + 1.0*0.52498 + 0.5*0.54983 - 0.5) \\ &= f(-0.27455) \\ &= 0.43179 \end{aligned}$$
$$b_2 = f(v_{12}a_1 + v_{22}a_2 + v_{32}a_3 - T_{22}) = f(-0.56214) = 0.36305$$
$$b_3 = f(v_{13}a_1 + v_{23}a_2 + v_{33}a_3 - T_{23}) = f(0.79965) = 0.68990$$

Step 4: Given the output from layer B, calculate the output from layer C, using the equation:

$$c_k = f\left(\sum_{j=1}^{m}(w_{jk}b_j) - T_{ck}\right)$$ Where *f ( )* is the same sigmoid function

Example: We calculate the output from each node in layer 3:

$$c_1 = f(w_{11}b_1 + w_{21}b_2 + w_{31}b_3 - T_{31}) = f(0.27621) = 0.56862$$
$$c_2 = f(w_{12}b_1 + w_{22}b_2 + w_{32}b_3 - T_{32}) = f(-1.06085) = 0.25715$$
$$c_3 = f(w_{13}b_1 + w_{23}b_2 + w_{33}b_3 - T_{33}) = f(1.24237) = 0.77598$$

Step 1 to 4 represent the forward activation flow.

Obviously, the actual output values c_k deviate from the desired output values (d_1=1, d_2=0, and d_3=0).

The next few steps of the backpropagation algorithm represent *the backward error flow* in which we *propagate* the error between the desired d_k (k=1 to 3) and the actual output c_k *backward* through the network, and try to find the best set of network parameters (v_{ij}, w_{ij}, and T_{ij}).

Step 5: Now *backpropagate* the error through the network, starting at the output and moving backward toward the input.

Calculate the k^{th} component of the output error, ε_k, for each node in layer C, according to the equation:

$\varepsilon_k = c_k(1-c_k)(d_k-c_k)$

$$\delta_j^m = -\frac{\partial E}{\partial v_j^k} f'(u_j^m) = v_j^m(1-v_j^m)(y_j - v_j^m)$$

Example:

We backpropagate, first calculating the error for each node in layer 3:

$$\varepsilon_1 = c_1(1-c_1)(d_1-c_1) = 0.10581$$
$$\varepsilon_2 = c_2(1-c_2)(d_2-c_2) = -0.04912$$
$$\varepsilon_3 = c_3(1-c_3)(d_3-c_3) = -0.13498$$

Step 6: Continue backpropagation, moving to layer B.

Calculate the j^{th} component of the error vector, e_j, of layer B relative to each ε_k, using the equation:

$$e_j = b_j(1-b_j)\left(\sum_{k=1}^{n}(w_{jk}\varepsilon_k)\right)$$

$$\delta_j^k = v_j^k(1-v_j^k)\sum_l w_{jl}\delta_l^{k+1}$$

Example:

We calculate the errors associated with layer 2, the hidden layer:

$$e_1 = b_1(1-b_1)(w_{11}\varepsilon_1 + w_{12}\varepsilon_2 + w_{13}\varepsilon_3) = -0.03648$$
$$e_2 = b_2(1-b_2)(w_{21}\varepsilon_1 + w_{22}\varepsilon_2 + w_{23}\varepsilon_3) = 0.008872$$
$$e_3 = b_3(1-b_3)(w_{31}\varepsilon_1 + w_{32}\varepsilon_2 + w_{33}\varepsilon_3) = 0.002144$$

Step 7: Adjust weight factors, calculating the new w_{jk}, i.e., $w_{jk,new}$, as:

$$\begin{pmatrix}\text{New weight}\\\text{factor}\end{pmatrix} = \begin{pmatrix}\text{Old weight}\\\text{factor}\end{pmatrix} + \begin{pmatrix}\text{Learning}\\\text{rate}\end{pmatrix} \times \begin{pmatrix}\text{Input}\\\text{term}\end{pmatrix} \times \begin{pmatrix}\text{Gradient-descent}\\\text{Correction term}\end{pmatrix}$$

$$w_{jk,new} = w_{jk,old} + \eta_c b_j \varepsilon_k$$
$$or$$
$$w_{jk,new} = w_{jk,old} + \eta_c b_j c_k(1-c_k)(d_k-c_k)$$

Example: We adjust weight factors for interconnections between layer 2 and 3. For simplicity, we assume that η=0.7 for all values of η.

$$w_{11,new} = w_{11} + \eta b_1 \varepsilon_1 = -0.9680$$
$$w_{12,new} = w_{12} + \eta b_1 \varepsilon_2 = -0.5149$$
$$w_{13,new} = w_{13} + \eta b_1 \varepsilon_3 = 0.4593$$

We continue these adjustments for the rest of the w_{jk}'s.

A comparison of the old and new weight factors shows:

$$[w_{jk,old}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

$$[w_{jk,new}] = \begin{bmatrix} -0.9680 & -0.5149 & 0.4593 \\ 1.0269 & -0.0125 & 0.4657 \\ 0.5511 & -0.5237 & 0.4349 \end{bmatrix}$$

Step 8: Adjust the thresholds T_{ck} (k=1 to n) in layer C, according to the equation:

$$T_{ck,new} = T_{ck} + \eta_c \varepsilon_k$$

Example: We adjust the internal thresholds for layer three:

$$T_{31,new} = T_{31} + \eta\varepsilon_1 = 0.0741$$
$$T_{32,new} = T_{32} + \eta\varepsilon_2 = 0.4656$$
$$T_{33,new} = T_{33} + \eta\varepsilon_3 = -0.5944$$

Thus, new and old threshold values are:

old :	T_{31}=0.0	T_{32}=0.5	T_{33}=-0.5
new:	T_{31}=0.0741	T_{32}=0.4656	T_{33}=-0.5944

Step 9: **Adjust weight factors v_{ij} for interconnections between layer 1 and 2, according to the equation:**

$$v_{ij,new} = v_{ij} + \eta_B a_i e_j$$

Example: **Again, η=0.7.**

$$v_{11,new} = v_{11} + \eta a_1 e_1 = -1.0147$$
$$v_{12,new} = v_{12} + \eta a_1 e_2 = -0.4964$$
$$v_{13,new} = v_{13} + \eta a_1 e_3 = 0.5009$$

We continue these adjustments for the rest of the v_{ij}'s, and the old and new weight factors are:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [v_{ij,new}] = \begin{bmatrix} -1.0147 & -0.4964 & 0.5009 \\ 0.9866 & 0.0033 & -0.4992 \\ 0.4860 & -0.4966 & 0.5008 \end{bmatrix}$$

Step 10: **Adjust the thresholds T_{Bj} (j=1 to m) in layer B, according to the equation:**

$$T_{Bj,new} = T_{Bj} + \eta_B e_j$$

Example: **We adjust the internal thresholds for layer three:**

$$T_{21,new} = T_{21} + \eta e_1 = 0.4745$$
$$T_{22,new} = T_{22} + \eta e_2 = 0.0062$$
$$T_{23,new} = T_{23} + \eta e_3 = -0.4985$$

Thus, new and old internal threshold values are:

Old : T_{21}=0.5	**T_{22}=0.0**	**T_{23}=-0.5**
New : T_{21}=0.4745	**T_{22}=0.0062**	**T_{23}=-0.4985**

Step 11: ***Repeat steps 2-10 until the squared error, E, or the output-error vector, ε, is zero or sufficiently small.***

Example: **This problem requires 3860 time steps to get results less than 2% error on variable d_k.**

Number of time steps: 3860			
Desired values:	d_1=1	d_2=0	d_3=0
Actual values:	c_1=0.9900	c_2=0.0156	c_3=0.0098
Percent error:	e_1=1.00%	e_2=1.56%	e_3=0.98%

4.2 Improving on BP Algorithm

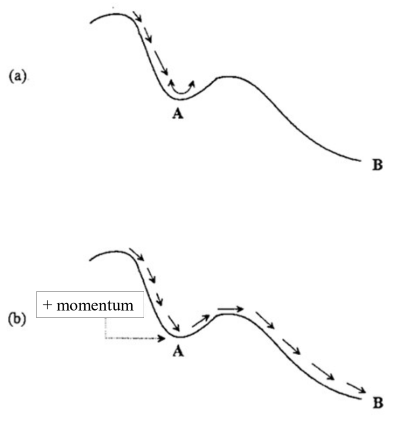
4.2.1 Generalized Delta-Rule Algorithm and Its Application to Fault Diagnosis

A. Adding momentum term

The delta rule uses a technique known as momentum to speed up the training.

Momentum is an extra weight added onto the weight factors when they are adjusted.

By accelerating the change in the weight factors, we improve the training rate.



$$\begin{pmatrix}\text{New weight}\\ \text{factor}\end{pmatrix} = \begin{pmatrix}\text{Old weight}\\ \text{factor}\end{pmatrix} + \begin{pmatrix}\text{Learning}\\ \text{rate}\end{pmatrix} \times \begin{pmatrix}\text{Input}\\ \text{term}\end{pmatrix} \times \begin{pmatrix}\text{Gradient-descent}\\ \text{correction term}\end{pmatrix} + \begin{pmatrix}\text{Momentum}\\ \text{coefficient}\end{pmatrix} \times \begin{pmatrix}\text{Previous}\\ \text{weight change}\end{pmatrix}$$

$$\Delta w_{ij}(n) = \eta \delta_j^k v_i^{k-1} + \alpha \Delta w_{ij}(n-1)$$

$$w_{ij,new} = w_{ij,old} + \Delta w_{ij}$$

Here, the momentum coefficient, α, is restricted such that 0< α<1.

To see the comparison, we will apply the delta rule for the three-layer-feedforward perceptron to the fault-diagnosis problem of section 4.1.

Step 1: Randomly assign values between 0 and 1 to weights v_{ij} and w_{jk}.

All input-layer thresholds must equal zero, i.e. $T_{1i}=0$;

All hidden- and output-layer thresholds must equal one, i.e., $T_{2j}=T_{3j}=1$.

Example:

i到j的权值

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \quad [w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix}$$

$$[T_{ij}] = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 1.0 & 1.0 & 1.0 \\ 1.0 & 1.0 & 1.0 \end{bmatrix}$$

j到k的权值

This T-matrix serves as a bias function.

Step 2: **Introduce the input I_i into the neural network, and calculate all outputs from the first layer according to the equations:.**

$$x_i = I_i + T_{1i} = I_i + 0$$
$$a_i = f(x_i) = \frac{1}{1+e^{-x_i}}$$

Note that f( ) here is again the sigmoid function

Example: We introduce the input vector into the neural network and calculate outputs from layer 1:

$$x_1 = I_1 + T_{11} = 0.3 + 0 = 0.3$$
$$x_2 = I_2 + T_{12} = 0.1 + 0 = 0.1$$
$$x_3 = I_3 + T_{13} = 0.2 + 0 = 0.2$$

$$a_1 = f(x_1) = \frac{1}{1+e^{-x_1}} = 0.57444$$
$$a_2 = f(x_2) = \frac{1}{1+e^{-x_2}} = 0.52498$$
$$a_3 = f(x_3) = \frac{1}{1+e^{-x_3}} = 0.54983$$

Step 3: **Knowing the output from the first layer, calculate outputs from the second layer, using the equation:**

$$b_j = f\left(\sum_{i=1}^{L}(v_{ij}a_i) + T_{2j}\right)$$

Example: We calculate the output from each node in layer 2:

$$b_1 = f(v_{11}a_1 + v_{21}a_2 + v_{31}a_3 + T_{21}) = f(1.22546) = 0.77302$$
$$b_2 = f(v_{12}a_1 + v_{22}a_2 + v_{32}a_3 + T_{22}) = f(0.43786) = 0.60775$$
$$b_3 = f(v_{13}a_1 + v_{23}a_2 + v_{33}a_3 + T_{23}) = f(1.29965) = 0.78578$$

Step 4: **Knowing the output from layer B, calculate the output from layer C, using the equation:**

$$c_k = f\left(\sum_{j=1}^{m}(w_{jk}b_j) + T_{3k}\right)$$

Where *f ( )* is the same sigmoid function

Example: We calculate the output from each node in layer 3:

$$c_1 = f(w_{11}b_1 + w_{21}b_2 + w_{31}b_3 + T_{31}) = f(1.22762) = 0.77340$$
$$c_2 = f(w_{12}b_1 + w_{22}b_2 + w_{32}b_3 + T_{32}) = f(0.22060) = 0.55493$$
$$c_3 = f(w_{13}b_1 + w_{23}b_2 + w_{33}b_3 + T_{33}) = f(2.08327) = 0.88927$$

Step 5: **Continue steps 1-4 for P number of training patterns presented to the input layer. Calculate the mean-squared error, E, according to the following equation:**

$$E = \sum_{p=1}^{P}\sum_{k=1}^{n}(d_k^p - c_k^p)^2$$

Where
P is the number of training patterns presented to the input layer,
n is the number of nodes on the output layer,
d_k^p is the desired output value from the k^{th} node in the p^{th} training pattern,
c_k^p is the actual output value from the k^{th} node in the p^{th} training pattern.

Example: We are training the network with just one input pattern (P=1). With desired output values $d_1=1$, $d_2=0$, and $d_3=0$, our total mean-squared error is:

$$E = \sum_{k=1}^{3}(d_k - c_k)^2 = (d_1 - c_1)^2 + (d_2 - c_2)^2 + (d_3 - c_3)^2 = 1.1501$$

Step 6: **Knowing the p^{th} pattern, calculate δ_{3k}^p, the gradient-descent term for the k^{th} node in the output layer for training pattern p. use the following equation:**

$$\delta_{3k}^p = (d_k^p - c_k^p)\frac{\partial f}{\partial x_k}$$

$$\delta_j^m = -\frac{\partial E}{\partial v_j^k} f'(u_j^m) = v_j^m(1 - v_j^m)(y_j - v_j^m)$$

无法显示该图片。

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

The partial derivative of the sigmoid function is:

$$\frac{\partial f}{\partial x_k} = \frac{e^{-x_k}}{(1+e^{-x_k})^2}$$

Note that x_k is the sum of the weighted inputs to the k^{th} node on the output layer plus the bias function(i.e., for the p^{th} training session):

$$x_k^p = \sum_j w_{jk}^p b_j^p + T_{3k}^p$$

Where for training pattern p, b_j^p is the output value of the j^{th} node in the hidden layer and T_{3k}^p is the threshold value on the output layer.

Example: We calculate δ_{3k}, the gradient-descent term for layer 3. To obtain δ_{3k}, we must first calculate the gradients by setting $x_1=c_1$, $x_2=c_2$, and $x_3=c_3$:

$$\frac{\partial f}{\partial x_1} = \frac{e^{-x_1}}{(1+e^{-x_1})^2} = \frac{e^{-0.77340}}{(1+e^{-0.77340})^2} = 0.27193$$
$$\frac{\partial f}{\partial x_2} = \frac{e^{-x_2}}{(1+e^{-x_2})^2} = \frac{e^{-0.55493}}{(1+e^{-0.55493})^2} = 0.23170$$
$$\frac{\partial f}{\partial x_3} = \frac{e^{-x_3}}{(1+e^{-x_3})^2} = \frac{e^{-0.88927}}{(1+e^{-0.88927})^2} = 0.20643$$

Knowing the gradients, we calculate the δ_{3k}'s:

$$\delta_{31} = (d_1 - c_1)\frac{\partial f}{\partial x_1} = 0.06162$$
$$\delta_{32} = (d_2 - c_2)\frac{\partial f}{\partial x_2} = -0.12858$$
$$\delta_{33} = (d_3 - c_3)\frac{\partial f}{\partial x_3} = -0.18357$$

Step 7: **Again knowing the p^{th} pattern, calculate δ_{2k}^p, the gradient-descent term for the j^{th} node on the *hidden layer* (layer 2). use the equation:**

$$\delta_j^{2j} = (\sum_k \delta_{3k}^p w_{jk}^p)\frac{\partial f}{\partial x_j} \qquad \delta_j^k = v_j^k(1-v_j^k)\sum_l w_{jl}\delta_l^{k+1}$$

Where the subscript k denotes a node in the output layer.
Recall that x_j is defined by:

$$x_j^p = \sum_i v_{ij}^p a_i^p + T_{2j}^p$$

and the partial derivative of the sigmoid function, again, is

$$\frac{\partial f}{\partial x_j} = \frac{e^{-x_j}}{(1+e^{-x_j})^2}$$

Example: We calculate δ_{2j}, the gradient-descent term for layer 2. Again, we must first calculate the gradients by setting $x_1=b_1$, $x_2=b_2$, and $x_3=b_3$:

$$\frac{\partial f}{\partial x_1} = \frac{e^{-x_1}}{(1+e^{-x_1})^2} = \frac{e^{-0.77302}}{(1+e^{-0.77302})^2} = 0.21608$$

$$\frac{\partial f}{\partial x_2} = \frac{e^{-x_2}}{(1+e^{-x_2})^2} = \frac{e^{-0.60775}}{(1+e^{-0.60775})^2} = 0.23512$$

$$\frac{\partial f}{\partial x_3} = \frac{e^{-x_3}}{(1+e^{-x_3})^2} = \frac{e^{-0.78578}}{(1+e^{-0.78578})^2} = 0.21506$$

Then, the δ_{2j}'s are:

$$\delta_{21} = (\delta_{31}w_{11} + \delta_{32}w_{12} + \delta_{33}w_{13})\frac{\partial f}{\partial x_1} = -0.019257$$

$$\delta_{22} = (\delta_{31}w_{21} + \delta_{32}w_{22} + \delta_{33}w_{23})\frac{\partial f}{\partial x_2} = -0.070936$$

$$\delta_{23} = (\delta_{31}w_{31} + \delta_{32}w_{32} + \delta_{33}w_{33})\frac{\partial f}{\partial x_3} = -0.026941$$

Step 8: **Knowing δ_{2j}^p for the hidden layer and δ_{3k}^p for the output layer, calculate the weight changes using the equations:**

$$\Delta v_{ij,new}^p = \eta \delta_{2j}^p a_i^p + \alpha \Delta v_{ij}^{p-1}$$

$$\Delta w_{jk,new}^p = \eta \delta_{3k}^p b_j^p + \alpha \Delta w_{jk}^{p-1}$$

Where η is the *learning rate*, and α is the *momentum coefficient*.

Example: We calculate the changes in weights, $\Delta v_{ij,new}$, and $\Delta w_{jk,new}$. We arbitrarily set the learning rate, η, to 0.9, and the momentum coefficient, α, to 0.7. Thus, for $v_{ij,new}$:

$$\Delta v_{11,new} = \eta\delta_{21}a_1 + \alpha\Delta v_{11} = 0.9*(-0.019257)*0.57444+0.7*0 = -0.009956$$

$$\Delta v_{12,new} = \eta\delta_{22}a_1 + \alpha\Delta v_{12} = 0.9(-0.070936)*0.57444+0.7*0 = -0.36674$$

$$\Delta v_{13,new} = \eta\delta_{23}a_1 + \partial\Delta v_{13} = 0.9*(-0.026941)*0.57444+0.7*0 = -0.013928$$

On this first time step, Δv_{ij} (from the "previous step") is zero, since no previous step exists. We continue this procedure for all $\Delta v_{ij,new}$'s, and end up with:

$$\left[\Delta v_{ij,new}\right] = \begin{bmatrix} -9.96*10^{-3} & -3.67*10^{-2} & -1.39*10^{-2} \\ -9.10*10^{-3} & -3.35*10^{-2} & -1.73*10^{-2} \\ -9.53*10^{-4} & -3.51*10^{-2} & -1.33*10^{-2} \end{bmatrix}$$

We also calculate the weight change $\Delta w_{jk,new}$ using the same learning rate (η=0.9) and momentum coefficient (α=0.7):

$$\Delta w_{11,new} = \eta\delta_{31}b_1 + \alpha\Delta w_{11} = 0.9*0.06162)0.77302+0.7*0 = 0.041856$$

$$\Delta w_{12,new} = \eta\delta_{32}b_1 + \alpha\Delta w_{12} = 0.9*(-0.12858)*0.77302+0.7*0 = -0.089455$$

$$\Delta w_{13,new} = \eta\delta_{33}b_1 + \alpha\Delta w_{13} = 0.9*(-0.18357)*0.77302+0.7*0 = -0.12771$$

Again, on this first time step, Δw_{ij}=0, since no previous time step exists.
We continue this procedure for all $w_{ij,new}$'s to yield:

$$\left[\Delta w_{jk,new}\right] = \begin{bmatrix} 4.19*10^{-2} & -8.95*10^{-2} & -1.28*10^{-1} \\ 3.31*10^{-2} & -7.03*10^{-2} & -1.00*10^{-2} \\ 3.92*10^{-2} & -9.09*10^{-2} & -1.30*10^{-1} \end{bmatrix}$$

Step 9: **Knowing the weight changes, update the weights according to the equations:**

$$w_{jk,new}^p = w_{jk}^{p-1} + \Delta w_{jk,new}^p$$

$$v_{ij,new}^p = v_{ij}^{p-1} + \Delta v_{ij,new}^p$$

Where

v_{ij}^p is the connection weight between the i^{th} element in the input layer and j^{th} element in the hidden layer,

w_{jk}^p is the connection weight between the j^{th} element in the hidden layer and k^{th} element in the output layer,

both for the p^{th} training pattern.

Repeat steps 2-9 for all training patterns until the squared error is zero or sufficiently low.

Example: Knowing the weight changes, we update the weights.

The old and new weights are:

$$[v_{ij}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & -0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \qquad [v_{ij,new}] = \begin{bmatrix} -1.0009 & -0.5367 & 0.4867 \\ 0.9909 & -0.0335 & -0.5173 \\ 0.4990 & -0.5351 & 0.4867 \end{bmatrix}$$

$$[w_{jk}] = \begin{bmatrix} -1.0 & -0.5 & 0.5 \\ 1.0 & 0.0 & 0.5 \\ 0.5 & -0.5 & 0.5 \end{bmatrix} \qquad [w_{jk,new}] = \begin{bmatrix} -0.9581 & -0.5895 & 0.4872 \\ 1.0331 & -0.0703 & 0.4900 \\ 0.5392 & -0.4091 & 0.4870 \end{bmatrix}$$

The results are:

Number of time steps:784			
Desired values:	d_1=1	d_2=0	d_3=0
Actual values:	c_1=0.9900	c_2=0.0099	c_3=0.0099
Percent error:	e_1=1.00%	e_2=0.99%	e_3=0.99%

B. Comparing two results:

Using standard BP algorithm

Number of time steps: 3860			
Desired values:	d_1=1	d_2=0	d_3=0
Actual values:	c_1=0.9900	c_2=0.0156	c_3=0.0098
Percent error:	e_1=1.00%	e_2=1.56%	e_3=0.98%

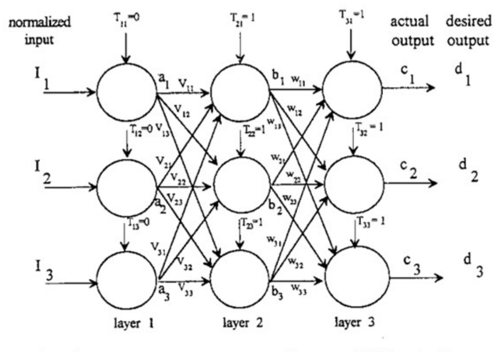
Using modified BP algorithm (with a momentum term)

Number of time steps:784			
Desired values:	d_1=1	d_2=0	d_3=0
Actual values:	c_1=0.9900	c_2=0.0099	c_3=0.0099
Percent error:	e_1=1.00%	e_2=0.99%	e_3=0.99 %

Note that above, we used only one data point (input pattern).

C. Alternative Formulation of Backpropagation Learning

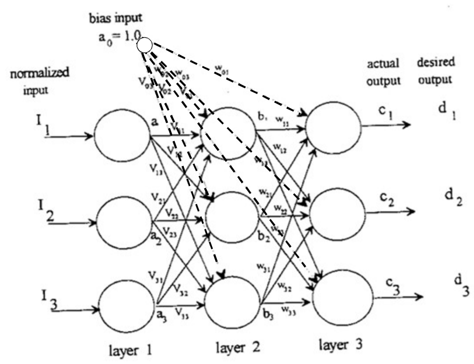
A simplified three-layer feedforward network



- Output from the hidden layer : $b_j = f(\sum_{i=1}^{L}(v_{ij}a_i)+T_{2j})$
- Output from the output layer : $c_k = f(\sum_{j=1}^{m}(w_{jk}b_j)+T_{3k})$

i=1 to 3 (=L)
j=1 to 3 (=m)
k=1 to 3 (=n)

An alternative architecture of the three-layer feedforward network



with a basis input and weight factors v_{oj} and w_{ok} replacing internal thresholds T_{ij}

This bias input a_0 is fed to all nodes in the hidden and output layers with weight factors v_{oj} and w_{ok} (j=1 to 3;k=1 to 3),and the resulting weighted biases replace the internal thresholds T_{2j} and T_{3k} (j=1 to 3;k=1 to 3).

output from the hidden layer:

$$b_j = f(\sum_{i=1}^{L}(v_{ij}a_i)+T_{2j})$$
$$b_j = f(\sum_{i=1}^{L}(v_{ij}a_i)+v_{oj}a_o)$$
$$b_j = f(\sum_{i=0}^{L}(v_{ij}a_i)) \qquad a_o=1.0$$

output from the output layer:

$$c_k = f(\sum_{j=1}^{m}(w_{jk}b_j)+T_{3k})$$
$$c_k = f(\sum_{j=1}^{m}(w_{jk}b_j)+w_{ok}b_o)$$
$$c_k = f(\sum_{j=0}^{m}(w_{jk}b_j)) \qquad b_o=1.0$$

The internal thresholds T_{2j} and T_{3k} in the original neural network become the weight factors between the bias node and the nodes in the hidden and output layers.

大作业：

- 编写**BP**算法程序（不限语言种类）
- 图示计算过程与结果
- 多入单出或多入多出实例验证
- 数据格式：

x1 x2 x3 ……xn y1 y2 ym

………………………………

………………………………

4.2.2 overview of special modified methods

$$\Delta w_{ij}(n) = \eta \delta_j^k v_i^{k-1} + \alpha \Delta w_{ij}(n-1)$$

A. Learning rate

$$\eta = \eta(0) e^{-E(n)}$$

Where η(0) is the initial learning rate, E(n) is the n^{th} mean-squared error.

If E(n) > E(n-1) Then $e^{-E(n)} < e^{-E(n-1)}$, η(n) < η(n-1)

$$\eta = \eta(0) e^{-\frac{E(n)}{T(n)}}$$

T is the simulated annealing temperature

$$\begin{aligned}
&if \quad \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0 \quad Then \\
&\quad \eta_{ij}(n) = \eta_{ij}(n-1) * u \\
&else \\
&\quad \eta_{ij}(n) = \eta_{ij}(n-1) * d \\
&end\ if
\end{aligned}$$

$u \approx \frac{1}{d}$ u=1.1~1.3, d=0.7~0.9

$$\Delta w_{ij}(n) = -\eta_{ij}(n) \times \left(\frac{\partial E}{\partial w_{ij}} + \alpha \times \Delta w_{ij}(n-1) \right)$$

$$\begin{aligned}
&if \quad \frac{\partial E}{\partial w_{ij}}(n) * \frac{\partial E}{\partial w_{ij}}(n-1) \geq 0 \quad Then \\
&\quad \Delta w_{ij}(n) = -\eta_{ij}(n) \times \frac{\partial E}{\partial w_{ij}} + \alpha \times \Delta w_{ij}(n-1) \\
&else \\
&\quad w_{ij}(n+1) = w_{ij}(n-1); \quad \Delta w_{ij}(n) = 0.0 \\
&end\ if
\end{aligned}$$

See page 79~92

New

$\eta(t) = \eta_{max} - \Delta\eta(t) = \eta_{max} - (\eta_{max} - \eta_{min}) * t / t_{max}$

t—number of iteration, $\eta_{max}=0.9$， $\eta_{min}=0.1$

New

$$\eta(t+1) = \begin{cases} (1+\theta_1)\eta(t), & E_{m}(t) < E_{m}(t-1) \\ \eta(t), & E_{m}(t) = E_{m}(t-1) \\ (1-\theta_1)\eta(t), & E_{m}(t) > E_{m}(t-1) \end{cases}$$

$$\mu(t+1) = \begin{cases} (1+\theta_2)\mu(t), & E_{m}(t) < E_{m}(t-1) \\ \mu(t), & E_{m}(t) = E_{m}(t-1) \\ (1-\theta_2)\mu(t), & E_{m}(t) > E_{m}(t-1) \end{cases}$$

$$0 < \theta_1, \theta_2 < 1$$

B. **Activation functions**

1. New activation function
2. Combination of activation function
3. Different node uses different activation function.
4. Each node uses the same “combination” activation function.
5. Choose activation function for each node based on the leaning case.

Example

MSE (structure 2:26:1)

Function \ Iteration	5000	10000
Sin(x)-TanH(x)	0.16	0.11
Sin(x)	0.19	0.16
TanH(x)	0.33	0.23

MSE (structure 2:15:11:1)

Function \ Iteration	5000	10000
TanH(x)-sin(x)	0.23	0.07
Sin(x)-TanH(x)	0.37	0.09
Sin(x)	0.57	0.13
TanH(x)	0.47	0.15

4.3 Practical aspects of neural computing

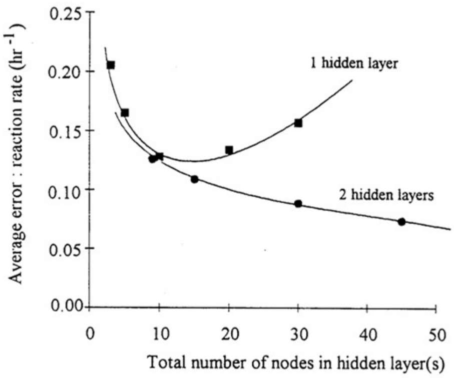
A. Design for input and output layers

B. Selecting the number of hidden layers and theirs nodes

single hidden layer is sufficient for identifying process faults or feature categories;
double hidden layer is sufficient for prediction,
using 30:15 hidden-layer configuration as the initial architecture for most networks.

Example:
3 input
1 output

Number of nodes in specified layer		Average error Reaction rate
Hidden layer 1	Hidden layer 2	
3	0	0.205
5	0	0.165
10	0	0.128
20	0	0.134
30	0	0.157
6	3	0.126
10	5	0.109
20	10	0.089
30	15	0.074



C. Normalizing input and output data sets

Problem 1: data distribution

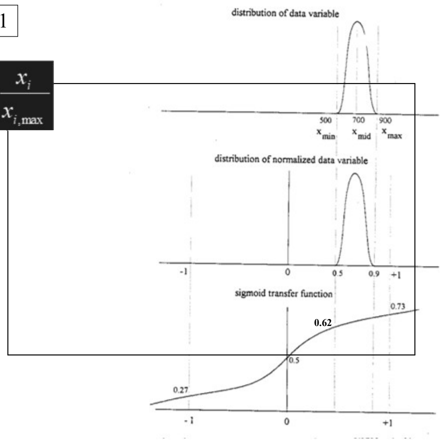
Problem 2: if input variable 1 has a value of 10000 and input variable 2 has a value of 10, the assigned weight for the second variable going into a node of hidden layer 1 must be much greater than that of the first variable to have any significance.

Problem 3: using the sigmoid function,
we find x_i=5, $f(x_i)$=0.993;
x_i=50, $f(x_i)$=1.00;
x_i=500, $f(x_i)$=1.00;

Three main types of normalization procedures will be introduced here.

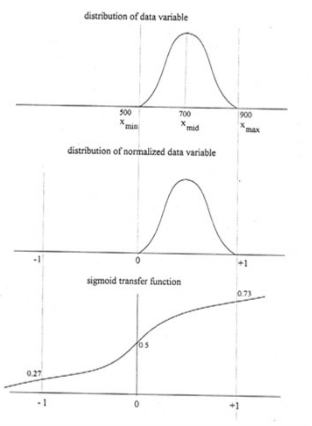
Method 1

$$x_{i,norm} = \frac{x_i}{x_{i,\max}}$$



Method 2

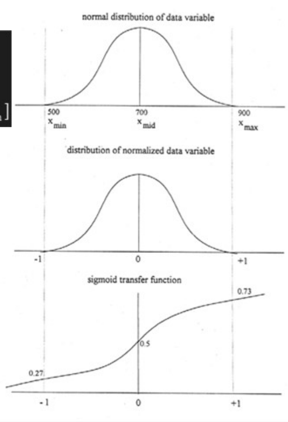
$$x_{i,norm} = \frac{x_i - x_{i,\min}}{x_{i,\max} - x_{i,\min}}$$



Method 3

$$x_{i,norm} = \frac{x_i - x_{i,avg}}{R_{i,\max}}$$

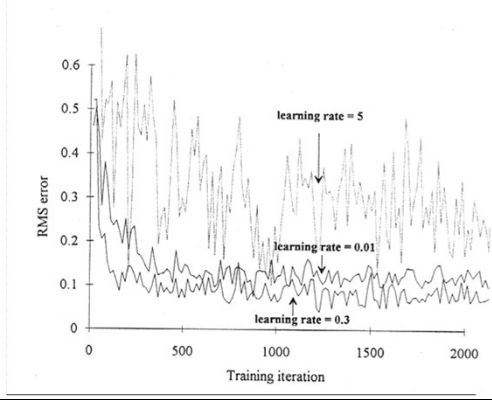
$$R_{i,\max} = \max imum \; [x_{i,\max} - x_{i,avg}, \; x_{i,avg} - x_{i,\min}]$$



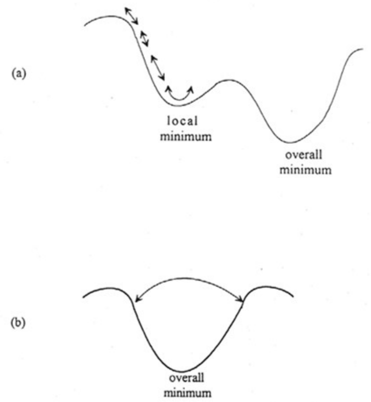
D. Initializing the weight-factor distribution

Weight-factor distribution	PH Average error 1	PH Average error 2
Gaussian : [-0.50 to 0.50]	0.136	0.333
Gaussian : [-0.75 to 0.75]	0.130	0.294
Gaussian : [-1.00 to 1.00]	0.104	0.253
Gaussian : [**-1.25 to 1.25**]	**0.098**	**0.214**
Gaussian : [-1.75 to 1.75]	0.108	0.276
Gaussian : [-2.50 to 2.50]	0.136	0.333

E. Setting the learning rate and momentum coefficient



With a low learning rate, training may be trapped in a locaal minimum



Adding momentum term

Learning rate	Momentum coefficient				
	0.00	0.20	0.40	0.60	0.80
0.01	0.137	0.139	0.139	0.138	0.130
0.10	0.100	0.095	0.082	0.063	0.093
0.30	0.075	0.062	0.054	0.038	>0.3
0.50	0.057	0.045	0.037	0.034	>0.3
0.70	0.043	0.035	0.035	>0.3	>0.3
1.00	0.040	0.026	0.034	>0.3	>0.3
2.00	0.030	0.140	>0.3	>0.3	>0.3
3.00	0.191	>0.3	>0.3	>0.3	>0.3
4.00	>0.3	>0.3	>0.3	>0.3	>0.3

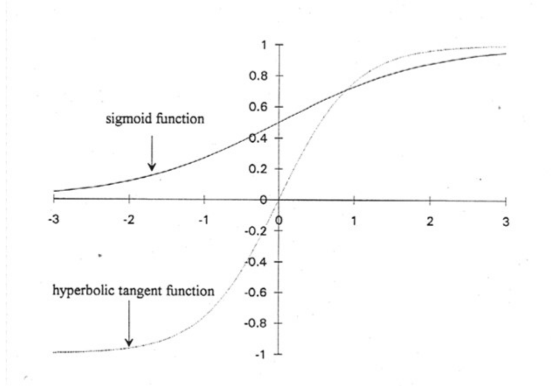
A typical learning schedule for a 3-hidden-layer BP network

Hidden layer 1				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.3	0.15	0.0375	0.0234
Momentum coefficient	0.4	0.2	0.05	0.00312

Hidden layer 2				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.25	0.125	0.03125	0.0195
Momentum coefficient	0.4	0.2	0.05	0.00312

Hidden layer 3				
Training iteration	0 -10,000	10,001- 30,000	30,001- 70,000	70,001-150,000
Learning rate	0.2	0.1	0.025	0.00156
Momentum coefficient	0.4	0.2	0.05	0.00312

F. Selecting the proper transfer function



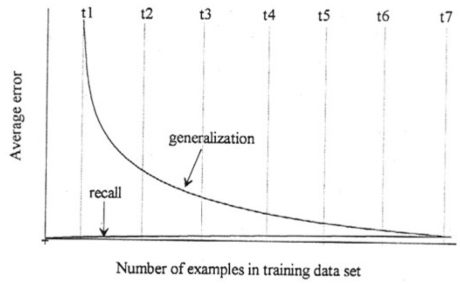
A comparison of the effectiveness of the hyperbolic tangent and sigmoid transfer function in training a prediction network

Number of nodes			Average error : tyrosine mole fraction	
hidden layer 1	hidden layer 2	hidden layer 3	hyperbolic tangent	sigmoid
5	0	0	0.071	0.126
10	0	0	0.064	0.132
20	0	0	0.058	0.126
30	0	0	0.063	0.150
40	0	0	0.061	0.154
6	3	0	0.079	0.094
10	5	0	0.035	0.097
20	10	0	0.035	0.103
30	15	0	$\boxed{0.019}$	0.103
9	6	3	0.060	0.097
15	10	5	0.049	0.096
20	12	7	0.035	0.096

G. Generating a network learning curve

A general procedure for generating a learning curve below:

1. Divide all variable training examples into selected groups(t1,t2,…,t7).
2. Use one group to train the network.
3. Test the network for both recall of the training data set and generalization of the remaining data groups not used in network training.
4. Add one randomly selected generalization data group to the training data set.
5. Retrain the network using the new training data set, reducing the learning rates and momentum coefficients.
(this is done to maintain some of the integrity of the previously trained network.)
6. Repeat steps 2-5 until there are no more groups remaining in the generalization data set.



Hidden-layer configuration 30:15

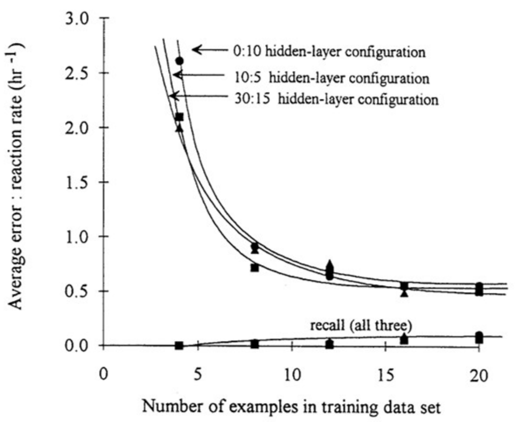
Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate	Avg error reaction rate
1	4	20	<0.001	2.002
1,6	8	16	0.022	0.887
1,3,6	12	12	0.015	0.763
1,3,5,6	16	8	0.059	0.498
1,2,3,5,6	20	4	0.073	0.555

Hidden-layer configuration 10:5

Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate	Avg error reaction rate
1	4	20	<0.001	2.100
1,6	8	16	0.027	0.720
1,3,6	12	12	0.029	0.699

Hidden-layer configuration 0:10

Training			Recall	Generalization
Group numbers in training set	Experiments in training set	Experiments in testing set	Avg error reaction rate	Avg error reaction rate
1	4	20	<0.001	2.613
1,6	8	16	0.013	0.912
1,3,6	12	12	0.050	0.644
1,3,5,6	16	8	0.102	0.552
1,2,3,5,6	20	4	0.109	0.562



H. Discussion

1. MSE：mean square error criterion:

$$E_{ym} = \frac{1}{2p}\sum_{t=1}^{p}\left[y(t) - \hat{y}(t \mid \mathbf{W})\right]^2$$

where p is the training samples

2. stability
3. convergence
4. overtraining
5. overfitting

作业1

编写BP算法程序。要求：

1、选择训练与泛化数据文件，数据格式为：

x1 x2……xn y1 y2……ym

并选择自变量与因变量数目；

2、选择网络结构；

3、选择学习速率、动量因子；

4、选择训练结束判据：训练误差、最大迭代次数；

5、动态画出训练误差曲线；

6、分别给出训练和泛化数据与网络计算结果比较图，并给出各自的相对误差。