Lecture Series on

Intelligent Control

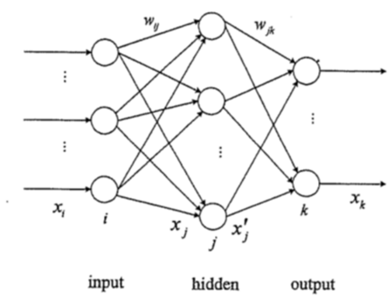
Lecture 14

Neural Networks in
Control System Applications - Examples

Kwang Y. Lee
Professor of Electrical & Computer Engineering
Baylor University
Waco, TX 76706, USA
Kwang_Y_Lee@baylor.edu

1

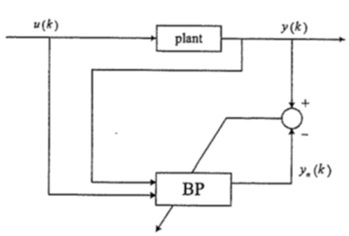
Feedforward NN



2

2

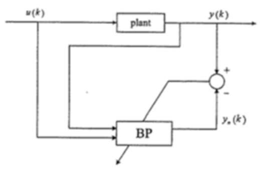
Modeling with NN

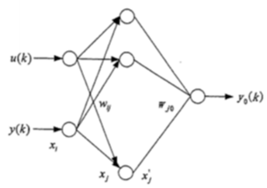


3

3

Modeling with NN

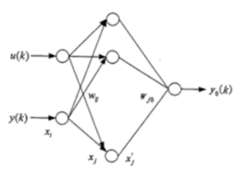




4

4

Backpropagation



(1) Feed-forward calculation

Input of hidden layer is

$$x_j = \sum_i w_{ij} x_i \tag{7.7}$$

Output of hidden layer is

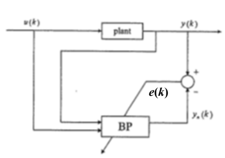
$$x_j' = f(x_j) = \frac{1}{1+e^{-x_j}} \tag{7.8}$$

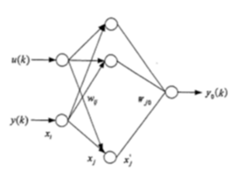
then

$$\frac{\partial x_j'}{\partial x_j} = x_j'(1-x_j')$$

5

5





Output of output layer is

$$y_0(k) = \sum_j w_{jo} x_j' \tag{7.9}$$

Then, the approximation error is

$$e(k) = y(k) - y_n(k)$$

Error index function is designed as

$$E = \frac{1}{2} e(k)^2 \tag{7.10}$$

6

6

Backpropagation

Output of output layer is

$$y_o(k) = \sum_j w_{jo} x_j' \qquad (7.9)$$

Then, the approximation error is

$$e(k) = y(k) - y_n(k)$$

Error index function is designed as

$$E = \frac{1}{2} e(k)^2 \qquad (7.10)$$

(2) Learning algorithm of BP

According to the steepest descent (gradient) method, the learning of weight value w_{jo} is

$$\Delta w_{jo} = -\eta \frac{\partial E}{\partial w_{jo}} = \eta \cdot e(k) \cdot \frac{\partial y_o}{\partial w_{jo}} = \eta \cdot e(k) \cdot x_j'$$

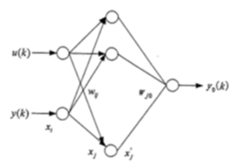
The weight value at time $k+1$ is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo}$$

7

7

Backpropagation



Output of output layer is

$$y_o(k) = \sum_j w_{jo} x_j'$$

Then, the approximation error is

$$e(k) = y(k) - y_n(k)$$

Error index function is designed as

$$E = \frac{1}{2} e(k)^2$$

Output of hidden layer is

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

then

$$\frac{\partial x_j'}{\partial x_j} = x_j'(1 - x_j')$$

The learning of weight value w_{ij} is

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta \cdot e(k) \cdot \frac{\partial y_o}{\partial w_{ij}}$$

where the chain rule is used, $\frac{\partial y_o}{\partial w_{ij}} = \frac{\partial y_o}{\partial x_j'} \cdot \frac{\partial x_j'}{\partial x_j} \cdot \frac{\partial x_j}{\partial w_{ij}} = w_{jo} \cdot \frac{\partial x_j'}{\partial x_j} \cdot x_i = w_{jo} \cdot x_j'(1 - x_j') \cdot x_i$.

The weight value at time $k+1$ is

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}$$

8

8

Backpropagation

The weight value at time $k+1$ is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo}$$

$$w_{ij}(k+1) = w_{ij}(k) + \Delta w_{ij}$$

Considering the effect of previous weight value change, the algorithm of weight value is

$$w_{jo}(k+1) = w_{jo}(k) + \Delta w_{jo} + \alpha\big(w_{jo}(k) - w_{jo}(k-1)\big) \qquad (7.11)$$

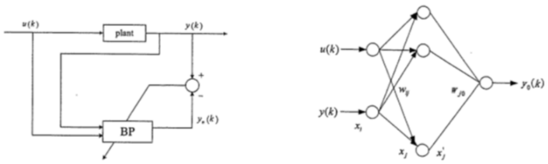
$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} + \alpha(w_{ij}(t) - w_{ij}(t-1)) \qquad (7.12)$$

where η is learning rate, α is momentum factor, $\eta \in [0,1]$, $\alpha \in [0,1]$.

9

9

Modeling with NN



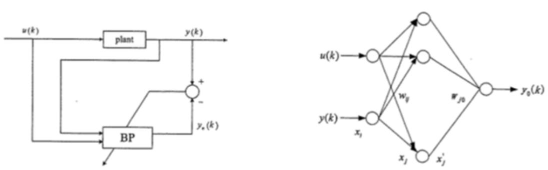
By using BP neural network approximation, Jacobian value can be calculated as follows:

$$\frac{\partial y(k)}{\partial u(k)} \approx \frac{\partial y_o(k)}{\partial u(k)} = \frac{\partial y_o(k)}{\partial x_j'} \times \frac{\partial x_j'}{\partial x_j} \times \frac{\partial x_j}{\partial x(1)} = \sum_j w_{jo} x_j' \left(1 - x_j'\right) w_{1j} \qquad (7.13)$$

10

10

Simulation Example



The plant is as follows

$$y(k) = u(k)^3 + \frac{y(k-1)}{1 + y(k-1)^2}$$

Input signal is chosen as $u(k) = 0.5\sin(6\pi t)$, let the neural network input vector as $\boldsymbol{x} = [u(k)\ y(k)]$, NN structure is chosen as 2-6-1, the initial value of $\boldsymbol{W}_{jo}$, $\boldsymbol{W}_{ij}$ is chosen as random value in $[-1 \quad +1]$, $\eta = 0.50$, $\alpha = 0.05$.

11

11

Simulation Example

```
%BP approximation
clear all;
close all;

xite=0.50;
alfa=0.05;

wjo=rands(6,1);
wjo_1=wjo;wjo_2=wjo_1;

wij=rands(2,6);
wij_1=wij;wij_2=wij;

dwij=0*wij;

x=[0,0]';

u_1=0;
y_1=0;

I=[0,0,0,0,0,0]';
Iout=[0,0,0,0,0,0]';
FI=[0,0,0,0,0,0]';

ts=0.001;
for k=1:1:1000

time(k)=k*ts;
u(k)=0.50*sin(3*2*pi*k*ts);
y(k)=u_1^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y(k);

for j=1:1:6
   I(j)=x'*wij(:,j);
   Iout(j)=1/(1+exp(-I(j)));
end
```

```
yo(k)=wjo'*Iout;      % Output of NNI networks

e(k)=y(k)-yo(k);  % Error calculation

wjo=wjo_1+(xite*e(k))*Iout+alfa*(wjo_1-wjo_2);

for j=1:1:6
  FI(j)=exp(-I(j))/(1+exp(-I(j)))^2;
end

for i=1:1:2
  for j=1:1:6
    dwij(i,j)=e(k)*xite*FI(j)*wjo(j)*x(i);
  end
End
wij=wij_1+dwij+alfa*(wij_1-wij_2);

%%%%%%%%%%%%%%Jacobian%%%%%%%%%%%%%%%%
yu=0;
for j=1:1:6
  yu=yu+wjo(j)*wij(1,j)*FI(j);
end
dyu(k)=yu;

wij_2=wij_1;wij_1=wij;
wjo_2=wjo_1;wjo_1=wjo;
u_1=u(k);
y_1=y(k);
end
figure(1);
plot(time,y,'r',time,yo,'b');
xlabel('times');ylabel('y and yo');
figure(2);
plot(time,y-yo,'r');
xlabel('times');ylabel('error');
figure(3);
plot(time,dyu);
xlabel('times');ylabel('dyu');
```

12

12

Simulation Example

Fig. 7.8 BP approximation

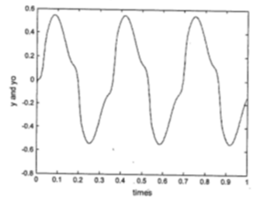
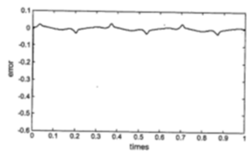


Fig. 7.9 BP approximation error

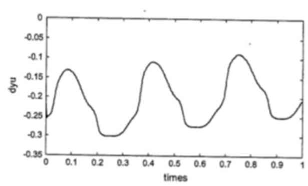


13

13

Simulation Example

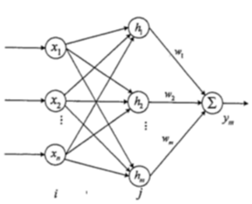
Fig. 7.10 Jacobian value identification



14

14

Radial Basis Function NN



In RBF neural network, $x = [x_i]^T$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $h = [h_j]^T$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_j = \exp\left(-\frac{\|x - c_j\|^2}{2b_j^2}\right) \qquad (7.14)$$

15

15

Radial Basis Function NN

In RBF neural network, $\boldsymbol{x} = [x_i]^{\mathrm{T}}$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $\boldsymbol{h} = [h_j]^{\mathrm{T}}$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_j = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2}\right) \quad (7.14)$$

where $\boldsymbol{c} = [c_{ij}] = \begin{bmatrix} c_{11} & \cdots & c_{1m} \\ \vdots & \cdots & \vdots \\ c_{n1} & \cdots & c_{nm} \end{bmatrix}$ represents the coordinate value of center point of the Gaussian function of neural net j for the ith input, $i = 1, 2, \ldots, n$, $j = 1, 2, \ldots, m$. For the vector $\boldsymbol{b} = [b_1, \ldots, b_m]^{\mathrm{T}}$, b_j represents the width value of Gaussian function for neural net j.

The weight value of RBF is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^{\mathrm{T}} \quad (7.15)$$

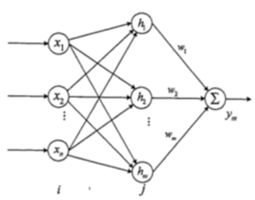
The output of RBF neural network is

$$y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \quad (7.16)$$

16

16

Radial Basis Function NN



The weight value of RBF is

$$\boldsymbol{w} = [w_1, \ldots, w_m]^{\mathrm{T}} \quad (7.15)$$

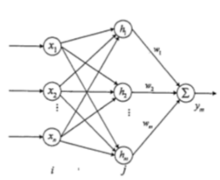
The output of RBF neural network is

$$y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \quad (7.16)$$

17

17

RBF Neural Network Simulation



In RBF neural network, $\boldsymbol{x} = [x_i]^{\mathrm{T}}$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $\boldsymbol{h} = [h_j]^{\mathrm{T}}$, h_j is Gaussian function value for neural net j in hidden layer, and

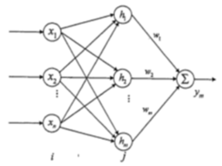
$$h_j = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2}\right) \quad (7.14)$$

Consider a structure 1-5-1 RBF neural network, we have one input as $x = x_1$, and $\boldsymbol{b} = [b_1 \ b_2 \ b_3 \ b_4 \ b_5]^{\mathrm{T}}$, $\boldsymbol{c} = [c_{11} \ c_{12} \ c_{13} \ c_{14} \ c_{15}]$, $\boldsymbol{h} = [h_1 \ h_2 \ h_3 \ h_4 \ h_5]^{\mathrm{T}}$, $\boldsymbol{w} = [w_1 \ w_2 \ w_3 \ w_4 \ w_5]$, and $y(t) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

18

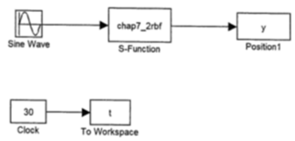
18

RBF Neural Network Simulation



Simulation programs:

(1) Simulink main program: chap7_2sim.mdl



19

19

RBF Neural Network Simulation

(2) S function of RBF: chap7_2rbf.m

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 7;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x=u(1);  %Input Layer
%i=1
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5];  %cij
b=[0.2 0.2 0.2 0.2 0.2]';  %bj

W=ones(5,1);  %Wj
h=zeros(5,1);  %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));  %Hidden Layer
end
y=W'*h;   %Output Layer

sys(1)=y;
sys(2)=x;
sys(3)=h(1);
sys(4)=h(2);
sys(5)=h(3);
sys(6)=h(4);
sys(7)=h(5);
```

20

20

RBF Neural Network Simulation

(3) Plot program: chap7_2plot.m

```
close all;

% y=y(:,1);
% x=y(:,2);
% h1=y(:,3);
% h2=y(:,4);
% h3=y(:,5);
% h4=y(:,6);
% h5=y(:,7);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,3),'k','linewidth',2);
xlabel('x');ylabel('hj');
hold on;
plot(y(:,2),y(:,4),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
```

21

21

RBF Neural Network Simulation

Fig. 7.12 Output of RBF

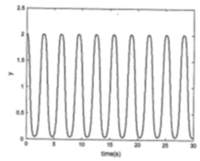
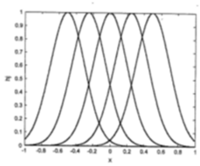


Fig. 7.13 Output of hidden neural net



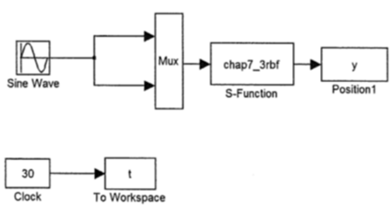
22

22

RBF Neural Network Simulation

Consider a structure 2-5-1 RBF neural network, we have $\mathbf{x} = [x_1, x_2]^{\mathrm{T}}$, $\mathbf{b} = [b_1 \quad b_2 \quad b_3 \quad b_4 \quad b_5]^{\mathrm{T}}$, $\mathbf{c} = \begin{bmatrix} c_{11} & c_{12} & c_{13} & c_{14} & c_{15} \\ c_{21} & c_{22} & c_{23} & c_{24} & c_{25} \end{bmatrix}$, $\mathbf{h} = [h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5]^{\mathrm{T}}$, $\mathbf{w} = [w_1 \quad w_2 \quad w_3 \quad w_4 \quad w_5]^{\mathrm{T}}$, and $y(t) = \mathbf{w}^{\mathrm{T}}\boldsymbol{h} = w_1h_1 + w_2h_2 + w_3h_3 + w_4h_4 + w_5h_5$.

(1) **Simulink main program: chap7_3sim.mdl**



23

23

RBF Neural Network Simulation

(2) **S function of RBF: chap7_3rbf.m**

```
function [sys,x0,str,ts] = spacemodel(t,x,u,flag)
switch flag,
case 0,
  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2,4,9}
  sys=[];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 8;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys = simsizes(sizes);
x0  = [];
str = [];
ts  = [];
function sys=mdlOutputs(t,x,u)
x1=u(1);  %Input Layer
x2=u(2);
x=[x1 x2]';
%i=2
%j=1,2,3,4,5
%k=1
c=[-0.5 -0.25 0 0.25 0.5;
  -0.5 -0.25 0 0.25 0.5];  %cij
b=[0.2 0.2 0.2 0.2 0.2]';   %bj

W=ones(5,1);  %Wj
h=zeros(5,1); %hj
for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j))); %Hidden
end
yout=W'*h;   %Output Layer

sys(1)=yout;
sys(2)=x1;
sys(3)=x2;
sys(4)=h(1);
sys(5)=h(2);
sys(6)=h(3);
sys(7)=h(4);
sys(8)=h(5);
```

24

24

RBF Neural Network Simulation

(3) **Plot program: chap7_3plot.m**

```
close all;
% y=y(:,1);
% x1=y(:,2);
% x2=y(:,3);
% h1=y(:,4);
% h2=y(:,5);
% h3=y(:,6);
% h4=y(:,7);
% h5=y(:,8);

figure(1);
plot(t,y(:,1),'k','linewidth',2);
xlabel('time(s)');ylabel('y');

figure(2);
plot(y(:,2),y(:,4),'k','linewidth',2);
xlabel('x1');ylabel('hj');
hold on;
plot(y(:,2),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,2),y(:,8),'k','linewidth',2);

figure(3);
plot(y(:,3),y(:,4),'k','linewidth',2);
xlabel('x2');ylabel('hj');
hold on;
plot(y(:,3),y(:,5),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,6),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,7),'k','linewidth',2);
hold on;
plot(y(:,3),y(:,8),'k','linewidth',2);
```

25

25

RBF Neural Network Simulation

Fig. 7.14 Output of RBF

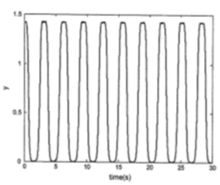
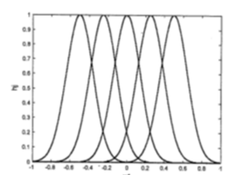


Fig. 7.15 Output of hidden neural net for first input

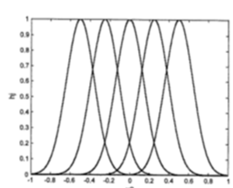


26

26

RBF Neural Network Simulation

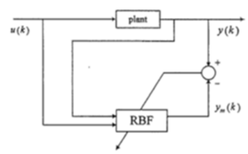
Fig. 7.16 Output of hidden neural net for second input

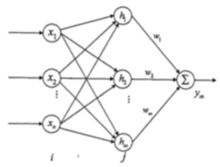


27

27

Modeling with RBF Neural Network





The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \quad (7.19)$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2 \quad (7.20)$$

28

28

Modeling with RBF Neural Network

The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \quad (7.19)$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2 \quad (7.20)$$

According to gradient descent method, the parameters can be updated as follows:

$$\Delta w_j(t) = -\eta \frac{\partial E}{\partial w_j} = \eta (y(t) - y_m(t)) h_j$$

$$w_j(t) = w_j(t-1) + \Delta w_j(t) + \alpha \left(w_j(t-1) - w_j(t-2) \right) \quad (7.21)$$

29

29

Modeling with RBF Neural Network

In RBF neural network, $\boldsymbol{x} = [x_i]^{\mathrm{T}}$ is input vector. Assuming there are mth neural nets, and radial basis function vector in hidden layer of RBF is $\boldsymbol{h} = [h_j]^{\mathrm{T}}$, h_j is Gaussian function value for neural net j in hidden layer, and

$$h_j = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{2b_j^2} \right) \quad (7.14)$$

The output of RBF is

$$y_m(t) = w_1 h_1 + w_2 h_2 + \cdots + w_m h_m \quad (7.19)$$

The performance index function of RBF is

$$E(t) = \frac{1}{2}(y(t) - y_m(t))^2 \quad (7.20)$$

$$\Delta b_j = -\eta \frac{\partial E}{\partial b_j} = \eta (y(t) - y_m(t)) w_j h_j \frac{\|\boldsymbol{x} - \boldsymbol{c}_j\|^2}{b_j^3} \quad (7.22)$$

$$b_j(t) = b_j(t-1) + \Delta b_j + \alpha \left(b_j(t-1) - b_j(t-2) \right) \quad (7.23)$$

$$\Delta c_{ji} = -\eta \frac{\partial E}{\partial c_{ji}} = \eta (y(t) - y_m(t)) w_j \frac{x_j - c_{ji}}{b_j^2} \quad (7.24)$$

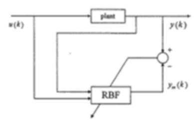
$$c_{ji}(t) = c_{ji}(t-1) + \Delta c_{ji} + \alpha \left(c_{ji}(t-1) - c_{ji}(t-2) \right) \quad (7.25)$$

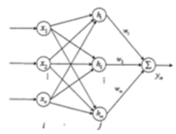
where $\eta \in (0,1)$ is the learning rate, $\alpha \in (0,1)$ is momentum factor.

30

30

Simulation Example





First example: only update *w*

Using RBF neural network to approximate the following discrete plant

$$G(s) = \frac{133}{s^2 + 25s}$$

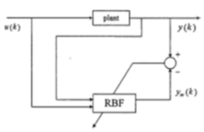
Consider a structure 2-5-1 RBF neural network, we choose inputs as $x(1) = u(t)$, $x(2) = y(t)$, and set $\alpha = 0.05$, $\eta = 0.5$. The initial weight value is chosen as random value between 0 and 1.

Choose the input as $u(\mathrm{t}) = \sin t$, consider the range of the first input $x(1)$ is $[0, 1]$, the range of the second input $x(2)$ is about $[0, 10]$, we choose the initial parameters of Gaussian function as $\boldsymbol{c}_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -10 & -5 & 0 & 5 & 10 \end{bmatrix}^{\mathrm{T}}$, $\boldsymbol{b}_j = 1.5$, $j = 1, 2, 3, 4, 5$.

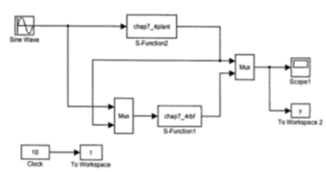
31

31

Simulation Example



(1) **Simulink main program: chap7_4sim.mdl**



32

32

Simulation Example

(2) **S function of RBF: chap7_4rbf.m**

```
function [sys,x0,str,ts]=s_function(t,x,u,flag)
switch flag,

  [sys,x0,str,ts]=mdlInitializeSizes;
case 3,
  sys=mdlOutputs(t,x,u);
case {2, 4, 9 }
  sys = [];
otherwise
  error(['Unhandled flag = ',num2str(flag)]);
end

function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 0;
sys=simsizes(sizes);
x0=[];
str=[];
ts=[];
function sys=mdlOutputs(t,x,u)
persistent w w_1 w_2 b ci
alfa=0.05;
xite=0.5;
if t==0
  b=1.5;
  ci=[-1 -0.5 0 0.5 1;
     -10 -5 0 5 10];
  w=rands(5,1);
  w_1=w;w_2=w_1;
end
ut=u(1);
yout=u(2);
xi=[ut yout]';
for j=1:1:5
   h(j)=exp(-norm(xi-ci(:,j))^2/(2*b^2));
end
ymout=w'*h';

d_w=0*w;
for j=1:1:5  %Only weight value update
  d_w(j)=xite*(yout-ymout)*h(j);
end
w=w_1+d_w+alfa*(w_1-w_2);

w_2=w_1;w_1=w;
sys(1)=ymout;
```

(3) **Plot program: chap7_4plot.m**

```
close all;

close all;
figure(1);
plot(t,y(:,1),'r',t,y(:,2),'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
```

33

33

Simulation Example

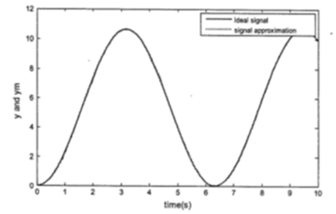
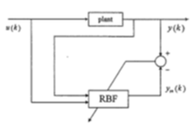
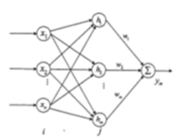


Fig. 7.18 RBF neural network approximation

Simulation Example





Second example: update w, c_j, b by gradient descent method

Using RBF neural network to approximate the following discrete plant

$$y(k) = u(k)^3 + \frac{y(k-1)}{1 + y(k-1)^2}$$

Consider a structure 2-5-1 RBF neural network, and we choose $x(1) = u(k)$, $x(2) = y(k)$, and $\alpha = 0.05$, $\eta = 0.15$. The initial weight value is chosen as random value between 0 and 1. Choose the input as $u(k) = \sin t$, $t = k \times T$, $T = 0.001$, we set the initial parameters of Gaussian function as $c_j = \begin{bmatrix} -1 & -0.5 & 0 & 0.5 & 1 \\ -1 & -0.5 & 0 & 0.5 & 1 \end{bmatrix}^T$, $b_j = 3.0$, $j = 1, 2, 3, 4, 5$.

Simulation Example

Simulation program: chap7_5.m

```
%RBF approximation
clear all;
close all;

alfa=0.05;
xite=0.15;
x=[0,1]';
b=3*ones(5,1);
c=[-1 -0.5 0 0.5 1;
  -1 -0.5 0 0.5 1];
w=rands(5,1);

w_1=w;w_2=w_1;
c_1=c;c_2=c_1;
b_1=b;b_2=b_1;
d_w=0*w;
d_b=0*b;
y_1=0;

ts=0.001;
for k=1:1:10000

time(k)=k*ts;
u(k)=sin(k*ts);

y(k)=u(k)^3+y_1/(1+y_1^2);

x(1)=u(k);
x(2)=y_1;

for j=1:1:5
  h(j)=exp(-norm(x-c(:,j))^2/(2*b(j)*b(j)));
end
ym(k)=w'*h';
em(k)=y(k)-ym(k);

M=1;
if M==1  %Only weight value update
  d_w(j)=xite*em(k)*h(j);
elseif M==2  %Update w,b,c
   for j=1:1:5
     d_w(j)=xite*em(k)*h(j);
     d_b(j)=xite*em(k)*w(j)*h(j)*(b(j)^-3)*norm(x-c(:,j))^2;
    for i=1:1:2
     d_c(i,j)=xite*em(k)*w(j)*h(j)*(x(i)-c(i,j))*(b(j)^-2);
    end
   end
   b=b_1+d_b+alfa*(b_1-b_2);
   c=c_1+d_c+alfa*(c_1-c_2);
end
w=w_1+d_w+alfa*(w_1-w_2);

y_1=y(k);

w_2=w_1;
w_1=w;

c_2=c_1;
c_1=c;

b_2=b_1;
b_1=b;
end
figure(1);
subplot(211);
plot(time,y,'r',time,ym,'k:','linewidth',2);
xlabel('time(s)');ylabel('y and ym');
legend('ideal signal','signal approximation');
subplot(212);
plot(time,y-ym,'k','linewidth',2);
xlabel('time(s)');ylabel('error');
```

Simulation Example

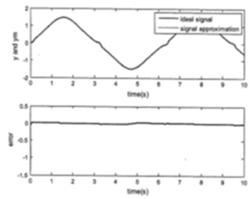


Fig. 7.19 RBF neural network approximation by only updating w (M = 1)

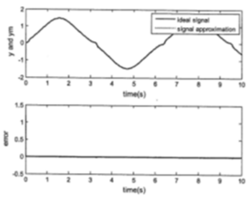


Fig. 7.20 RBF neural network approximation by updating w, b, c (M = 2)

37

37