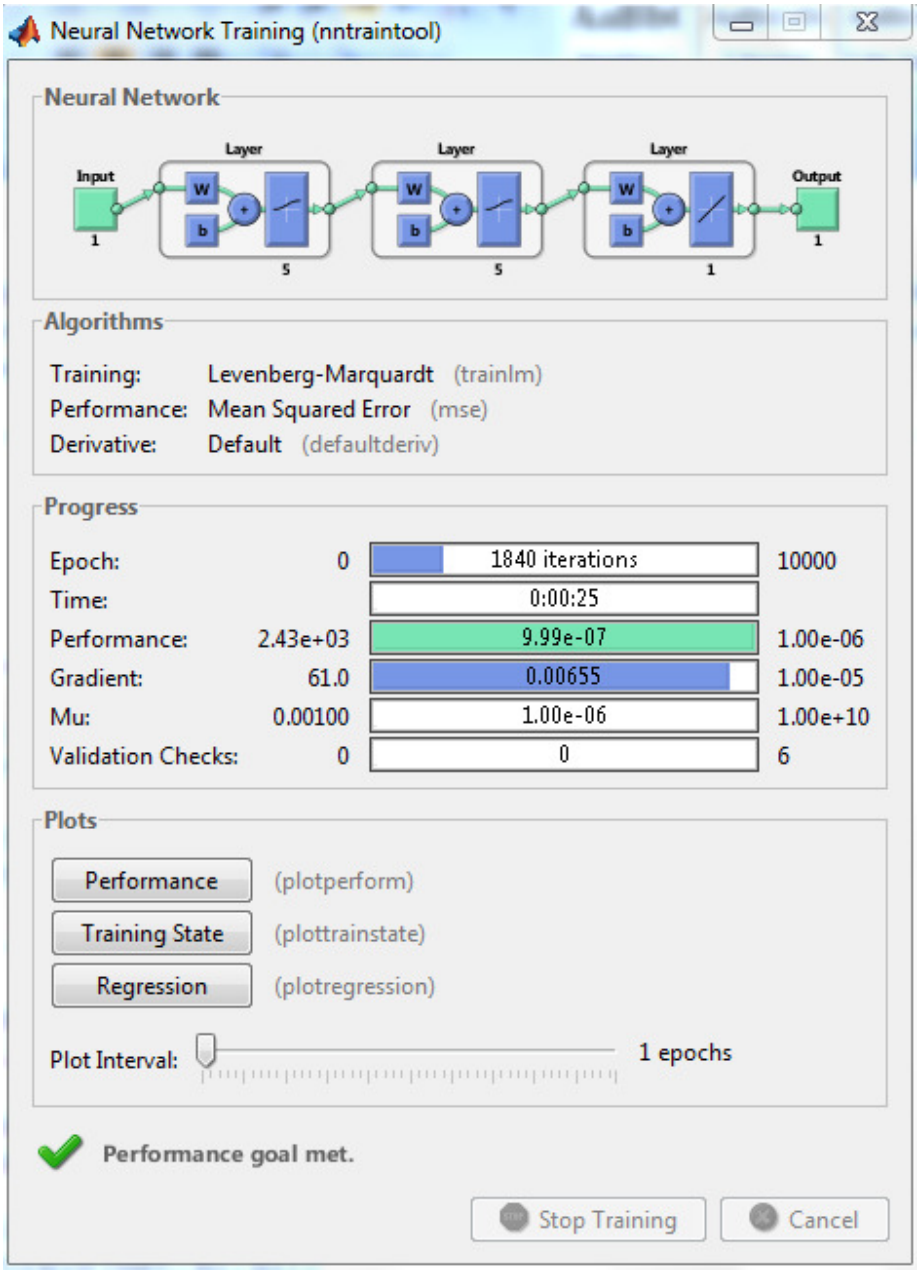
Chapter 4

Back propagation learning algorithm (1)

```
clear all

x=[-15 -10 -5 0 5 10 15];
y=0.05*x.^3-0.2*x.^2-3*x+20;

x1=[-15:1:15];
y1=0.05*x1.^3-0.2*x1.^2-3*x1+20;

%-------------------------------------------------

net = newff([-15 15],[5 5 1],{'logsig' 'logsig'
'purelin'});

net.trainParam.epochs = 10000;
net.trainParam.goal = 1e-6;

net = init(net);
net = train(net,x1,y1);

Test=[-15:0.1:15];
Output=sim(net,Test);

%-------------------------------------------------

figure(1);
plot(x,y,'b*',x1,y1,'b')
title('Training Vectors');
xlabel('Input Vector x');
ylabel('Target Vector y for trainning');
legend({'Input Point','Desired Output y=0.05*x^3-0.2*x^2-
3*x+20'})

%-------------------------------------------------

figure(2);
plot(x,y,'b*',x1,y1,'b',Test,Output,'r')
```

```
title('Testing MLP Network');
xlabel('Input Vector');
ylabel('Output');
legend({'Input point','Desired output','Test output'})
```



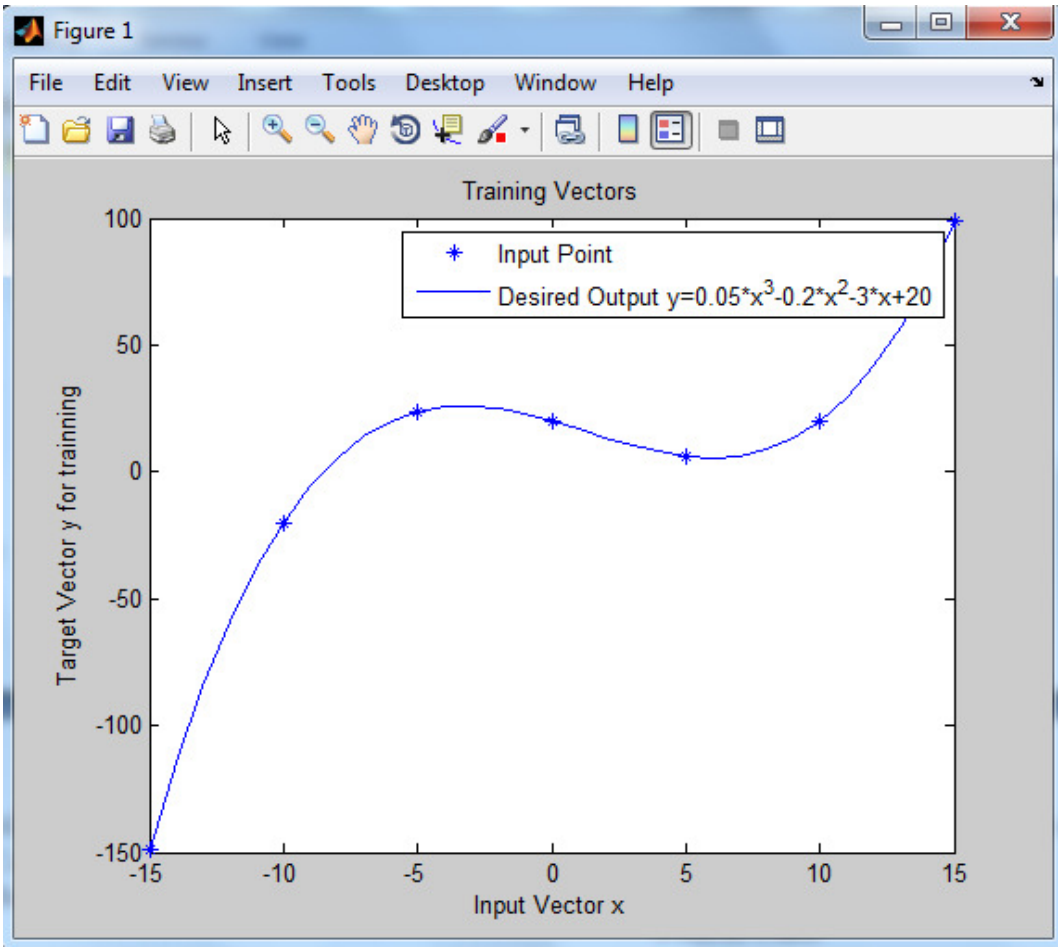


Figure 1
File Edit View Insert Tools Desktop Window Help
Training Vectors
Input Point
Desired Output y=0.05*x^3-0.2*x^2-3*x+20
Target Vector y for trainning
Input Vector x
100
50
0
-50
-100
-150
-15
-10
-5
0
5
10
15

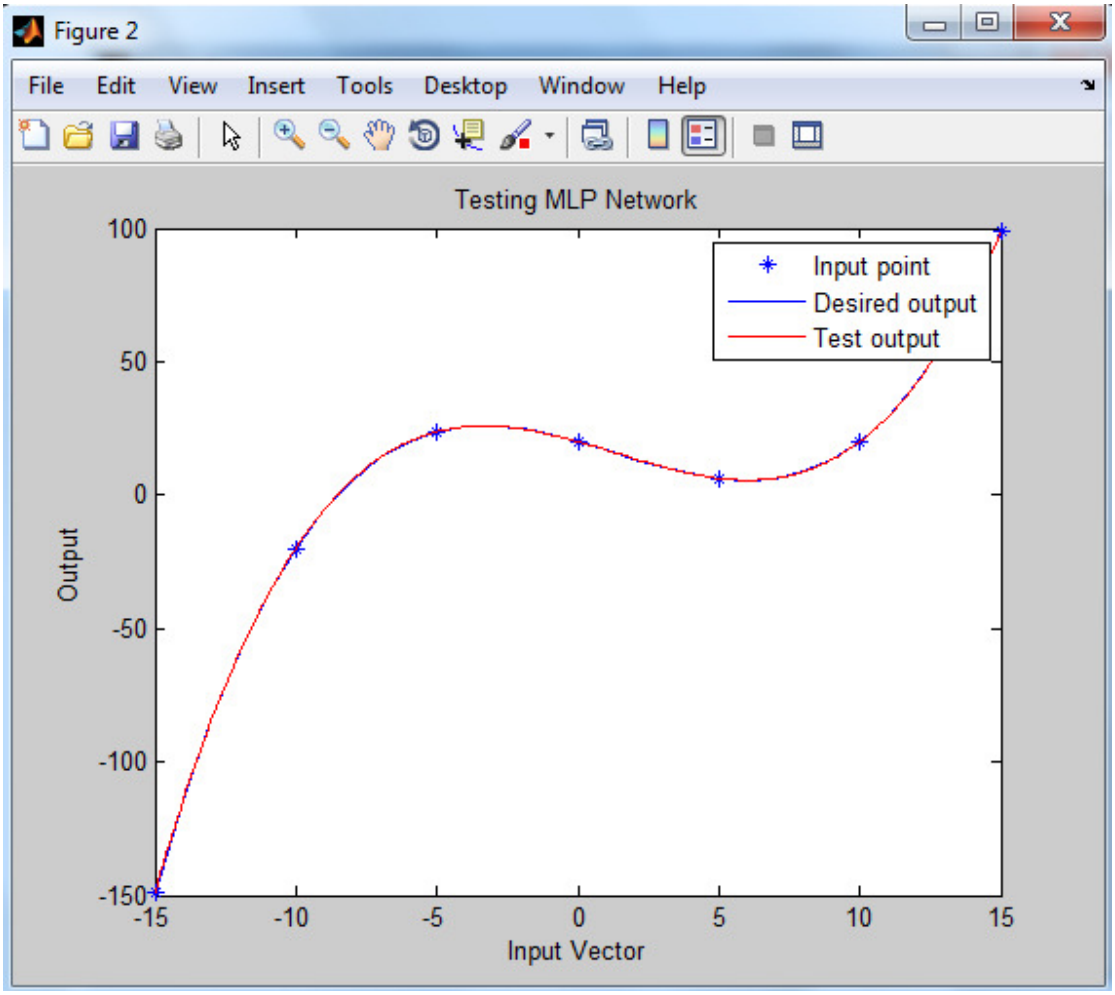


Figure 2
File Edit View Insert Tools Desktop Window Help
Testing MLP Network
Input point
Desired output
Test output
Output
Input Vector
100
50
0
-50
-100
-150
-15
-10
-5
0
5
10
15

Back propagation learning algorithm (2)

```
%Matlab Program for MLP Error Back-propagation Learning:
clear;
 %Network structure and learning parameter definition
 no_input=2;
no_output=2;
no_hidden=8;
eita=0.5;
alpha=0.5;
max_epoch=100;
min_err=0.05;

%Training data generation
%The training data consist of data points which can be
divided
%into two classes in terms of their distances from the
origin.

no_samples=200;
radius=3;
range=8;

rand('seed',0);
P=rand(no_samples,2)*range-range/2;

figure
hold on
for i=1:no_samples
    if P(i,1)*P(i,1)+P(i,2)*P(i,2)<radius*radius
        P(i,1)=P(i,1)*0.9;          %x coordinate
        P(i,2)=P(i,2)*0.9;          %y coordinate
        P(i,3)=0.95;                %desired output
component 1
        P(i,4)=0.05;                %desired output
component 2
        plot(P(i,1),P(i,2),'o');    %plot points belonging
to class 1
    else
        P(i,1)=P(i,1)*1.1;
        P(i,2)=P(i,2)*1.1;
```

```
        P(i,3)=0.05;
        P(i,4)=0.95;
        plot(P(i,1),P(i,2),'+');    %plot points belonging
to class 2
    end
end
title('Training Data');
print -deps input_space.eps;
hold off
disp('The training data is now shown in the figure
window.')
disp('Press any key to continue ......')
pause;

%MLP training by error back-propagation

%Initialisation

W_hidden=rand(no_hidden,no_input+1)-0.5;
W_output=rand(no_output,no_hidden+1)-0.5;
dW_hidden_old=W_hidden*0;
dW_output_old=W_output*0;
epoch=1;
stop_flag=0;

%Start a learning epoch

while epoch<max_epoch+1 & stop_flag==0
   no_misclassification(epoch)=0;
   for i=1:no_samples

%Forward calculation

      input=[1 P(i,1:no_input)]';
      desired_output=P(i,no_input+1:no_input+no_output)';
      sum_hidden=W_hidden*input;
      out_hidden=1./(1+exp(-sum_hidden));
      sum_output=W_output*[1; out_hidden];
      actual_output=1./(1+exp(-sum_output));

%Error calculation

      output_error=desired_output-actual_output;
      err(i)=output_error'*output_error; %err will be a row
vector
```

```
      if max(abs(output_error))>0.45

no_misclassification(epoch)=no_misclassification(epoch)+1;
      end

%Backward calculation and weight updating

      delta_output=output_error.*actual_output.*(1-
actual_output);
      dW_output=eita*delta_output*[1
out_hidden']+alpha*dW_output_old;
      W_output=W_output+dW_output;
      dW_output_old=dW_output;


delta_hidden=(W_output(:,2:no_hidden+1)'*delta_output).*out
_hidden.*(1-out_hidden);

dW_hidden=eita*delta_hidden*input'+alpha*dW_hidden_old;
      W_hidden=W_hidden+dW_hidden;
      dW_hidden_old=dW_hidden;
   end %for i

   average_err(epoch)=sqrt(sum(err)/no_samples);
   fprintf('epoch=%4d, no_misclassified=%4d,
root_mean_squared_error=%8.4f\n', epoch,
no_misclassification(epoch), average_err(epoch));
   if average_err(epoch)<min_err &
no_misclassification(epoch)==0
      stop_flag=1;
   end
    epoch=epoch+1;
end %while

%Learning curve ploting
figure
subplot(211);
plot(average_err);
xlabel('Epoch');
ylabel('Average error');
title('Learning Curve');
subplot(212);
plot(no_misclassification);
xlabel('Epoch');
ylabel('No. of misclassified patterns');
```

```
print -deps learning_curve.eps;
disp('The learning curve is now shown in the figure
window.')
disp('Press any key to continue ......')
pause;

%Decision boundary calculation

step=0.1;
x=-5:step:5;
y=-5:step:5;
for i=1:length(x)
   for j=1:length(y)
      input=[1;x(i);y(j)];
      sum_hidden=W_hidden*input;
      out_hidden=1./(1+exp(-sum_hidden));
      sum_output=W_output*[1; out_hidden];
      out_output=1./(1+exp(-sum_output));
      if out_output(1)>out_output(2)
         A(j,i)=-1;
      else
         A(j,i)=1;
      end
   end
end
figure
subplot(111);
contour(x,y,A,1);
hold on
for i=1:no_samples
   if P(i,1)*P(i,1)+P(i,2)*P(i,2)<radius*radius
       plot(P(i,1),P(i,2),'o'); %plot points belonging to
class 1
   else
       plot(P(i,1),P(i,2),'+'); %plot points belonging to
class 2
    end
end
title('Decision Boundary');
print -deps decision_boundary.eps;
hold off
disp('The decision boundary is now shown in the figure
window.')
```