CS5486 Intelligent Systems Assignment 1

Name: WANG Yue
Student ID: 56359462
EID: ywang2554

---

1. Program using MATLAB or other codes to simulate the Perceptron with bipolar activation functions and ADALINE to classify OR and XOR data of two variables. Show the convergence behaviors graphically and your programs.

(1) Perceptron - OR

- **Code:**

```
%{
Name   :  perceptron - OR
Author :  Wang Yue
Date   :  2020.10.18
%}
clear;
data = [0,0,1,-1;  %dataset(x1, x2, -threshold, y)
        0,1,1,1;
        1,0,1,1;
        1,1,1,1];
[d,n] = size(data); %d=datasize  n=#of x
% w = rand(1,3);     %weight
w = [0 0 0];        %weight
lr = 0.005;         %learning rate
E = 1;              %Error
E_threshold = 0.0001;
t=0;                %Cumulative number of iterations
iteration = 100;    %Maximum iterations
mse = zeros(1,iteration);
sse = zeros(1,iteration);
while (E > E_threshold) && (t < iteration)    %------------------------------
    t= t+1;
    for i=1:d
        u = data(i,1:3)*w';
        if u > 0    %transfer function
            y(i) = +1;
        else
            y(i) = -1;
        end
```

```
        w = w + lr*(data(i,4)-y(i))*data(i,1:3);
    end
    mse(t) = 1/d * ((y-data(:,4)')*(y-data(:,4)')'); %mean squared error
    sse(t) = ((y-data(:,4)')*(y-data(:,4)')'); %sum squared error
    E = mse(t);
end  %----------------
figure(1);%----------------------------------------
%---line
X = -3:3; %x values for graph
Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
plot(X,Y); hold on;
%---spot
for i = 1:d
    if ( data(i,4)==1 )
        scatter(data(i,1),data(i,2),'b+');
        hold on;
    else
        scatter(data(i,1),data(i,2),'ro');
        hold on;
    end
end
title('The results of classification');
xlabel('x1')
ylabel('x2')
hold off;
figure(2);%-----------------------------------------
a = 1:iteration;
b = mse;
c = sse;
subplot(2,1,1);
plot(a,b, 'b*-');
title('mean squared error');
xlabel('t') ;
ylabel('mse') ;
hold on;
subplot(2,1,2);
plot(a,c, 'b*-');
title('sum squared error');
xlabel('t') ;
ylabel('sse') ;
hold off;
```

◆ **Figure:**

Figure1 - The result of classification:

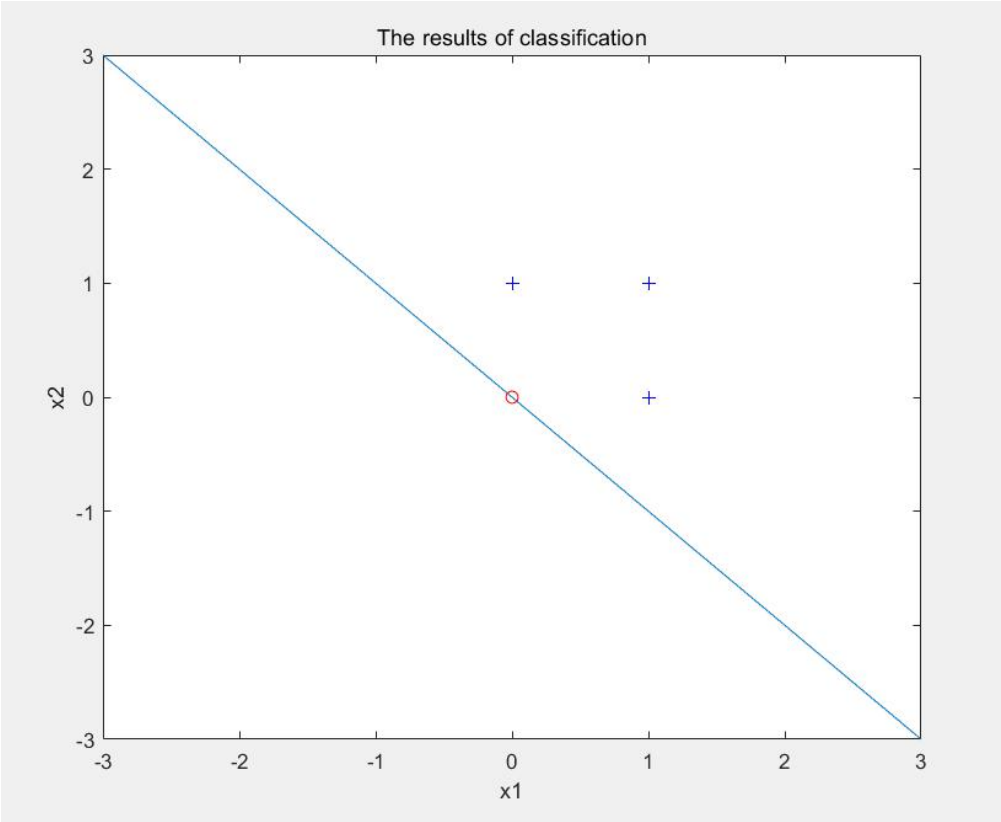
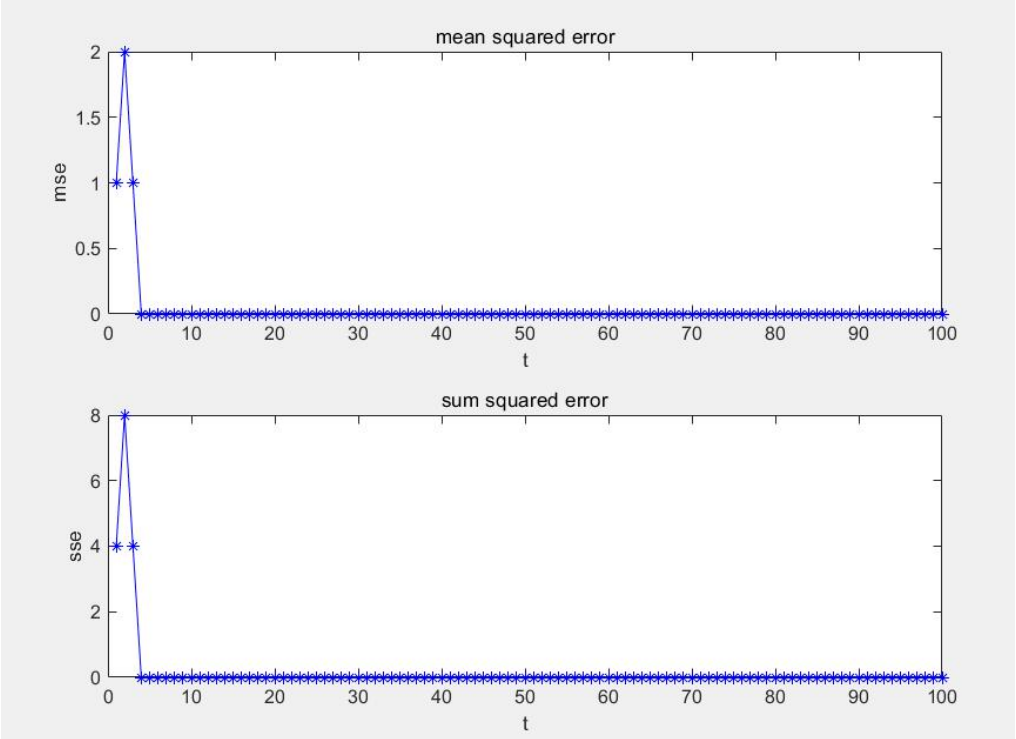


Figure2 - The result of mse and sse:



PS: We can see that OR function could be linearly separated by Single-layer perceptron successfully.

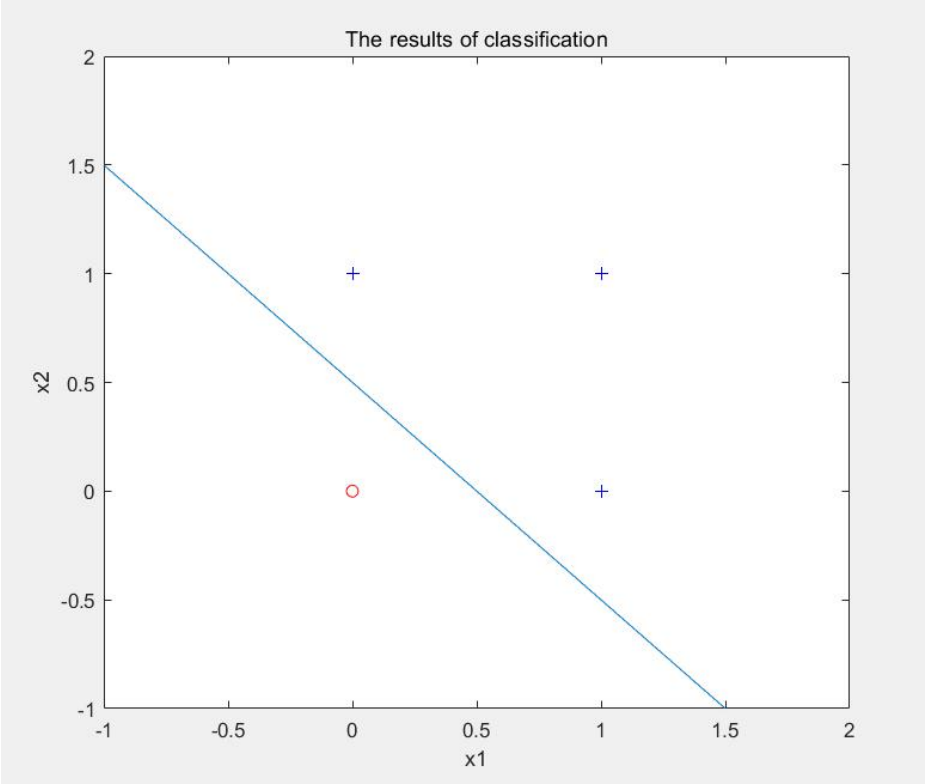
(2) ADAINE - OR

- **Code:**

```
%{
Name   :  adaline - OR
Author :  Wang Yue
Date   :  2020.10.18
%}
clear;
data = [0,0,1,-1;  %dataset(x1, x2, -threshold, y)
        0,1,1,1;
        1,0,1,1;
        1,1,1,1];
[d,n] = size(data); %d=datasize  n=#of x
x = data(:,1:3);    %data
z = data(:,4);      %ground truth
% w = rand(1,3);    %weight+ th
w = [0 0 1];        %weight
lr = 0.05;          %learning rate
E = 1;              %Error
E_threshold = 0.00001;
t = 0;
iteration = 100;    %iteration
alpha = 1;
while (E > E_threshold) && (t < iteration)
% for t=1:100
    t = t + 1;
    %--------------
    for i=1:d
        for j=1:d
            y(j) = x(j,:)*w'*alpha; %Identity activition function
        end
        for k=1:d
            w = w + lr*(z(k)-y(k))*x(k,:);
        end
    end
    mse(t) = 1/d * ((y-z')*(y-z')');
    sse(t) = ((y-z')*(y-z')');
    E = mse(t);
end
figure(1);%---------------------------------
%-----------line
X = -3:3; %x values for graph
Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
plot(X,Y); axis([-1 2 -1 2]);  hold on;
```

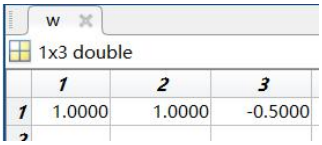
```
%-------------spot
for i = 1:d
    if ( data(i,4)==1 )
        scatter(data(i,1),data(i,2),'b+');
        hold on;
    else
        scatter(data(i,1),data(i,2),'ro');
        hold on;
    end
end
title('The results of classification');
xlabel('x1')     ylabel('x2')
hold off;
figure(2);%---------------------------------------
a = 1:iteration;
b = mse;
c = sse;
subplot(2,1,1);plot(a,b, 'b*-');
title('mean squared error');
xlabel('t') ;  ylabel('mse') ;     hold on;
subplot(2,1,2);plot(a,c, 'b*-');
title('sum squared error');
xlabel('t') ;  ylabel('sse') ;     hold off;
```

◆ **Figure:** The result of classification



◆ **The result of weights and threshold:**

W1	W2	threshold
1	1	-0.5



(3) Perceptron - XOR

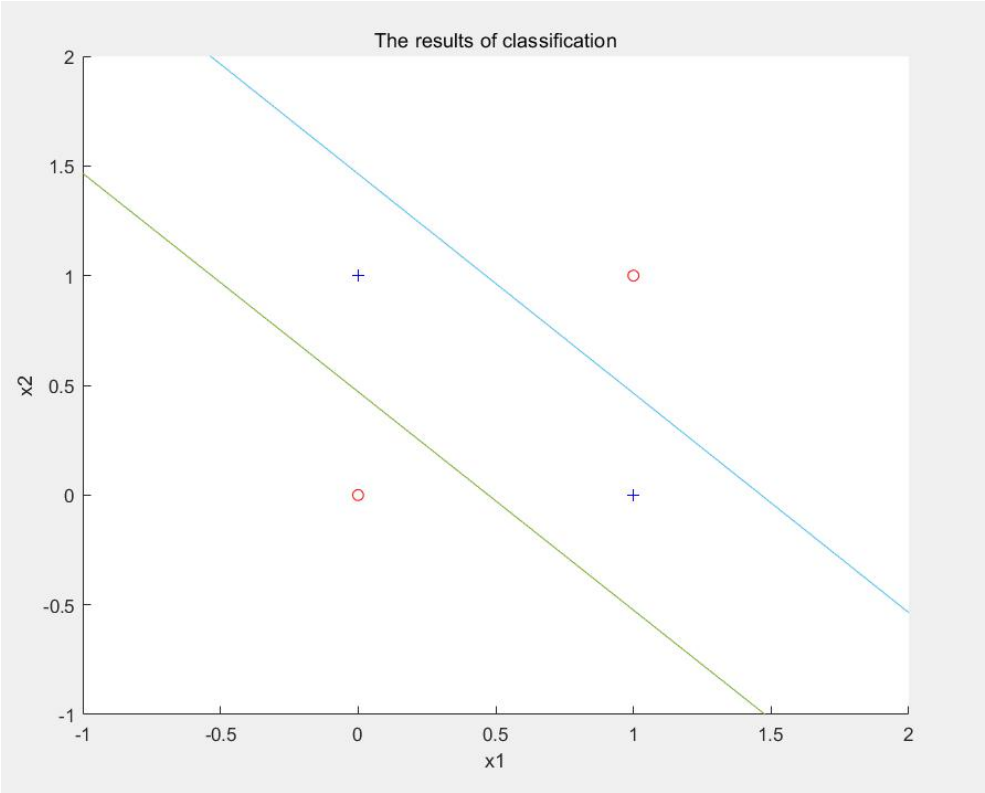
◆ **Code:**

```
%{
Name   :  perceptron - XOR
Author :  Wang Yue
Date   :  2020.10.19
%}
clear;
x =[1 1;
    1 0;
    0 1;
    0 0];
t = [0 1 1 0]';
lr = 0.5; %learning rate
feature_number = size(x,2);
w_hidden_node_number = 2;
w_output_node_number= 1; % regression problem
w_hidden = rand(feature_number, w_hidden_node_number);
w_hidden_th = rand(1, w_hidden_node_number);
% w_hidden = [0.5 0.4; 0.3 0.1];
% w_hidden_th = [0.1 0.2];
w_output = rand(w_hidden_node_number, w_output_node_number);
w_output_th = rand(1, w_output_node_number);
% w_output = [0.1 ; -0.2];
y = zeros(4,1000);
for k = 1:1000
    for i = 1:size(t, 1)
        y_hidden = tanh(x(i,:) * w_hidden + w_hidden_th);
        y_output = logsig(y_hidden * w_output + w_output_th);
        y(i,k) = y_output;
        e = t(i) - y_output;
        d_output = e.* y_output .* (1 - y_output);
        d_hidden = (1-y_hidden.^2) .* (d_output * w_output');
        w_output = w_output + lr * y_hidden' * d_output;
        w_output_th = w_output_th + lr * d_output;
        w_hidden = w_hidden + lr * x (i, :)' * d_hidden;
        w_hidden_th = w_hidden_th +  lr * d_hidden;
    end
```

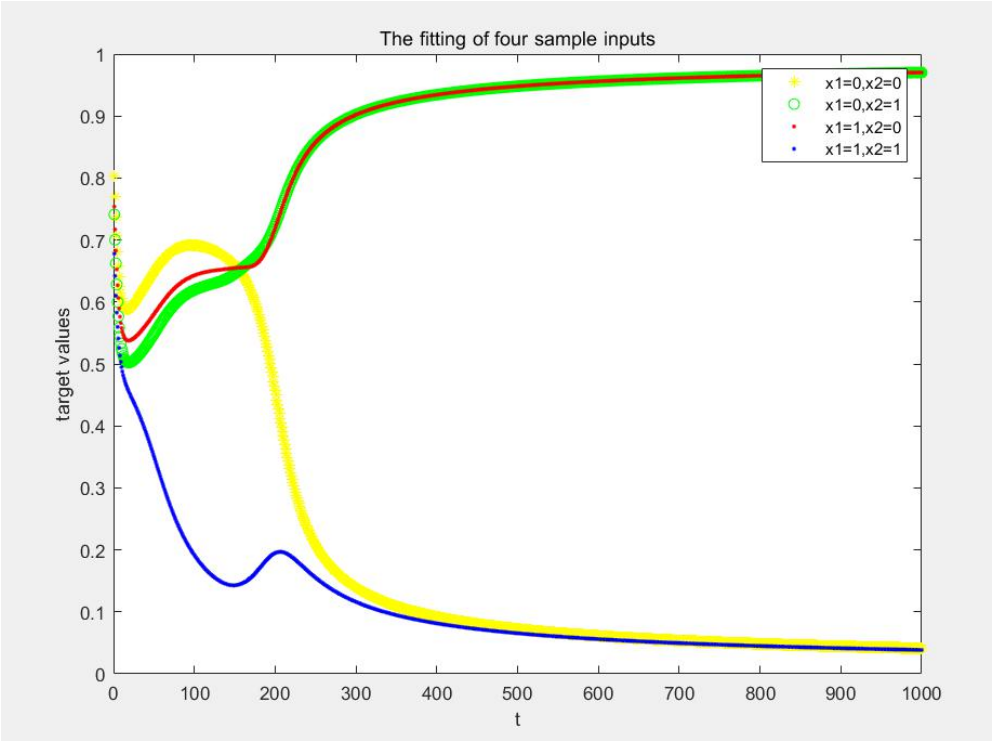
```
    [X1, X2] = meshgrid(-0.5:1.5);
    Y1 = w_hidden_th(1) + X1*w_hidden(1, 1) + X2 * w_hidden(2, 1);
    Y2 = w_hidden_th(2) + X1*w_hidden(1, 2) + X2 * w_hidden(2, 2);
    %Dynamic figure-start-------------
    for i = 1:4
        if ( t(i)==1 )
            scatter(x(i,1),x(i,2),'b+');
            hold on
        else
            scatter(x(i,1),x(i,2),'ro');
            hold on
        end
    end
    hold on;
    contour(X1, X2, Y1, [0,0], 'k');
    contour(X1, X2, Y2, [0,0], 'k');
    title(['Iteration: ' num2str(k)]);
    hold off
    drawnow;
    %Dynamic figure-end-------------
End

% %Static figure--------------
% for i = 1:4
%     if ( t(i)==1 )
%         scatter(x(i,1),x(i,2),'b+');
%         hold on
%     else
%         scatter(x(i,1),x(i,2),'ro');
%         hold on
%     end
% end
X1 = -2:2;
X2 = -2:2;
Y1 = -(w_hidden(1, 1)/w_hidden(2, 1)) * X1 - (w_hidden_th(1,1)/w_hidden(2, 1));
hold on;
Y2 = -(w_hidden(2, 1)/w_hidden(2, 1)) * X2 - (w_hidden_th(1,2)/w_hidden(2, 2));
hold on;
plot(X1,Y1);
plot(X2,Y2);
axis([-1 2 -1 2]);
title('The results of classification');
xlabel('x1')
ylabel('x2')
```

```
hold off;
%plot
figure(2);
a = 1:1000;
b = y(1,:);
c = y(2,:);
d = y(3,:);
e = y(4,:);
plot(a, b, 'y*'); hold on;
plot(a, c, 'go'); hold on;
plot(a, d, 'r.'); hold on;
plot(a, e, 'b.'); hold on;
title('The fitting of four sample inputs');
legend('x1=0,x2=0','x1=0,x2=1','x1=1,x2=0','x1=1,x2=1');
xlabel('t')
ylabel('target values')
hold off;
```

◆ **Figure: The result of classification**



◆ **Figure: The fitting of four sample inputs**



(4) Adaline - XOR

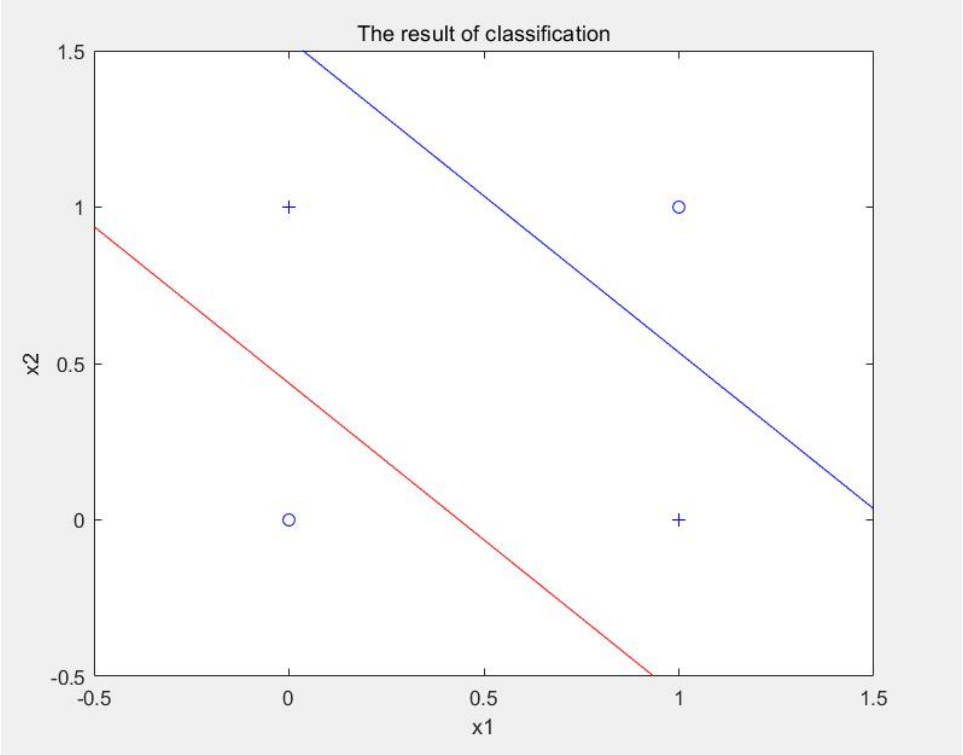
◆ **Code:**

```
%{
Name   :  adaline - XOR
Author :  Wang Yue
Date   :  2020.10.22
%}
clear
% Network initialization--------
dataset = [0 1 0 1;
           1 0 0 1;
           1 1 0 0];
X=dataset(1:2,:);
Y=dataset(3,:);
[l,c]=size(X);
disp('The initial weights are randomly generated as follows: ');
tt=0;
% presion=input('Please enter the training error accuracy: ');  %0.00001
% speed1=input('Please enter your learning rate: ');  %0.1
presion=0.00001;
lr1=0.5;  lr2=lr1;
w1=rands(2,2);     %Implicit layer weights initialization
w2=rands(1,2);     %Output layer weights initialization
b1=rands(2,1);
```

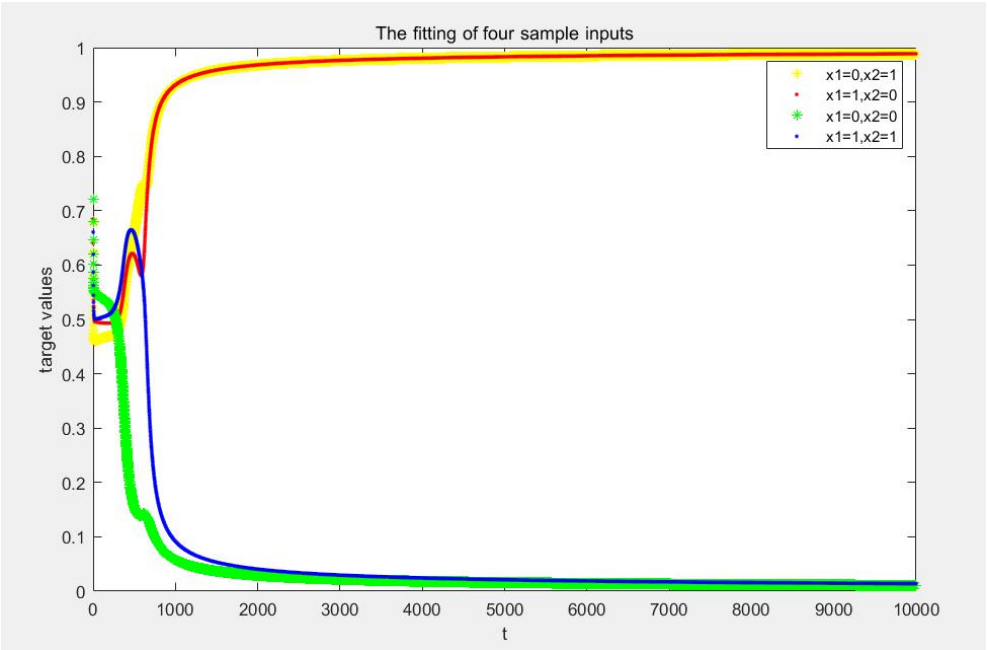
```
b2=rands(1);
iteration=10000;   %Maximum iteration number
t=1;               %Initialize the number of iterations
e=1;               %Initialization error
%-------------------------------------------------------------------
while(e>presion && t<iteration)   %Less than the error accuracy and the maximum number
of iterations
    e=0;
    for i=c*(t-1)+1:t*c
        % Feedforward
        % The first layer
        x0=X(:,i-c*t+c);
        n1=w1*x0+b1;
        y1=logsig(n1);
        % The second layer
        n2=w2*y1+b2;
        y2(i)=logsig(n2);
        % Feedback algorithm
        e=e+(dataset(3,i-c*t+c)-y2(i))^2;
        deta2=-2*dlogsig(n2,y2(i))*(dataset(3,i-c*t+c)-y2(i)); %Calculate deta2 for
the output layer
        temp=zeros(size(y1,1));
        for j=1:size(y1,1)
            temp(j,j)=(1-y1(j))*y1(j);
        end
        deta1=temp*w2'*deta2; %Calculate the deta1 for the input layer
        %A weight iteration
        w1=w1-lr1*deta1*x0';
        w2=w2-lr2*deta2*y1';
        b1=b1-lr1*deta1;
        b2=b2-lr2*deta2;
    end
    E(t)=0.5*e;
    t=t+1;
end
% Results output
 for n=1:1:t-1
     p0(n)=y2(c*n-3);
     p1(n)=y2(c*n-2);
     p2(n)=y2(c*n-1);
     p3(n)=y2(c*n);
 end
 if t<35000
     tt=tt+1;
```

```
 end
disp('The ideal output is: 1 1 0 0')
fprintf('The actual output is: %f,%f,%f,%f\n',p0(n),p1(n),p2(n),p3(n))
fprintf('The final iteration error is: %f\n',e)
fprintf('Number of iterations is: %d\n',t)
%plot
figure(1);
plotpv(X,Y);      hold on %point
plotpc(w1,b1);    hold on %line
title('The result of classification');
xlabel('x1');    ylabel('x2');
figure(2);
plot(p0,'y*'); hold on;
plot(p1,'r.'); hold on;
plot(p2,'g*'); hold on;
plot(p3,'b.'); hold on;
legend('x1=0,x2=1','x1=1,x2=0','x1=0,x2=0','x1=1,x2=1');
title('The fitting of four sample inputs');
xlabel('t');    ylabel('target values')
hold off;
```

◆ **Figure: The result of classification**



◆ **Figure: The fitting of four sample inputs**



◆ **Figure: The result**

```
命令行窗口
  The initial weights are randomly generated as follows:
  The ideal output is: 1 1 0 0
  The actual output is: 0.989198, 0.986803, 0.011503, 0.010302
  The final iteration error is: 0.000529
  Number of iterations is: 10000
fx >>
```

2.Using MATLAB Neural Networks toolbox or your own program to simulate a multilayer Perceptron with unipolar sigmoid activation functions and a radial-basis function network with Gaussian function for classifying XOR data of two variables. Show your detailed results (i.e., convergence behaviors and output values for all four training samples).

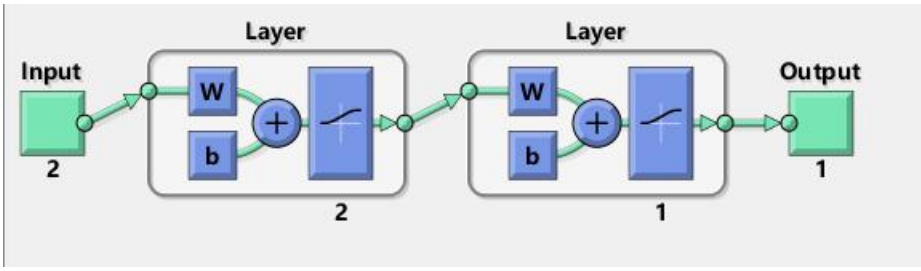
(1) A multilayer Perceptron with unipolar sigmoid activation functions

◆ **Code:**

```
%{
Name   : A multilayer Perceptron with unipolar sigmoid activation functions
Author : Wang Yue
Date   : 2020.10.20
%}
clear;
dataset = [0 0 0;
```

```
            0 1 1;
            1 0 1;
            1 1 0];
x = dataset(:,1:2)';
z = dataset(:,3)';
net = newff([0 1; 0 1],[2 1], {'logsig' 'logsig'}, 'trainlm', 'learngdm', 'mse');
net.trainParam.epochs = 100;
net.trainParam.goal = 0;
net.trainParam.lr=0.01;
% net.trainParam.mc=0.9;
% net.trainParam.show=25;
net = train(net,x,z);
Y = sim(net, x);
view(net);
```

- **Diagram:**



- **Properties :**

Network Type	FNN - BP
Training function(BTF)	Levenberg-Marquardt (LM)
Adaption learning function(BLF)	LEARNGDM
Performance function(BTF)	Mean Squared Error (MSE)
Number of layers	2
Properties for Layer 1:	Number of neurons: 2 Transfer Function: logsig
Properties for Layer 2:	Transfer Function: logsig
Stopping conditions:	Epochs: 100 Goal: 0 Time: Infinite

- **The result output values for all four training samples in the end of the iteration :**

target values with their corresponding predicted values:

x1	x2	z	Y(output data)
0	0	0	2.1057e-05
0	1	1	1.0000
1	0	1	1.0000
1	1	0	1.2511e-04

Y

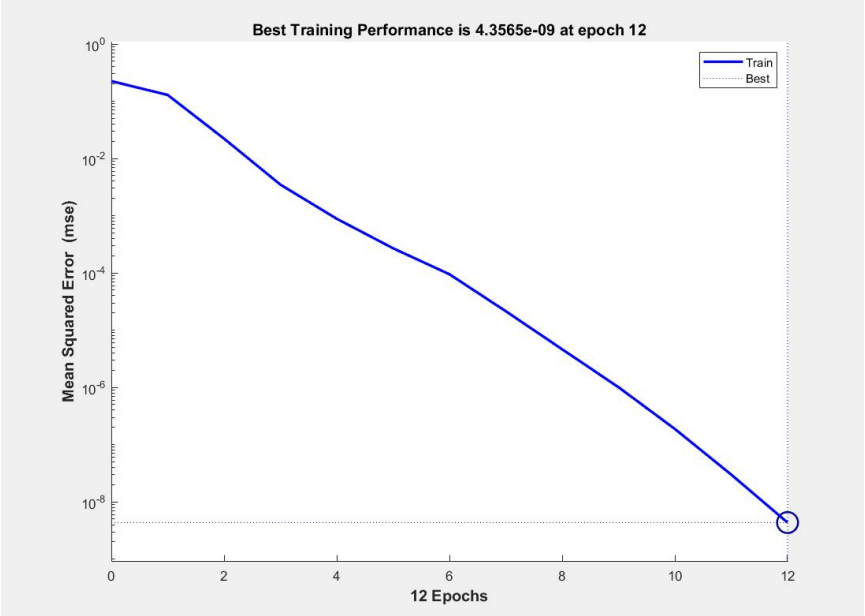
1x4 double

	1	2	3	4
1	2.1057e-05	1.0000	1.0000	1.2511e-04

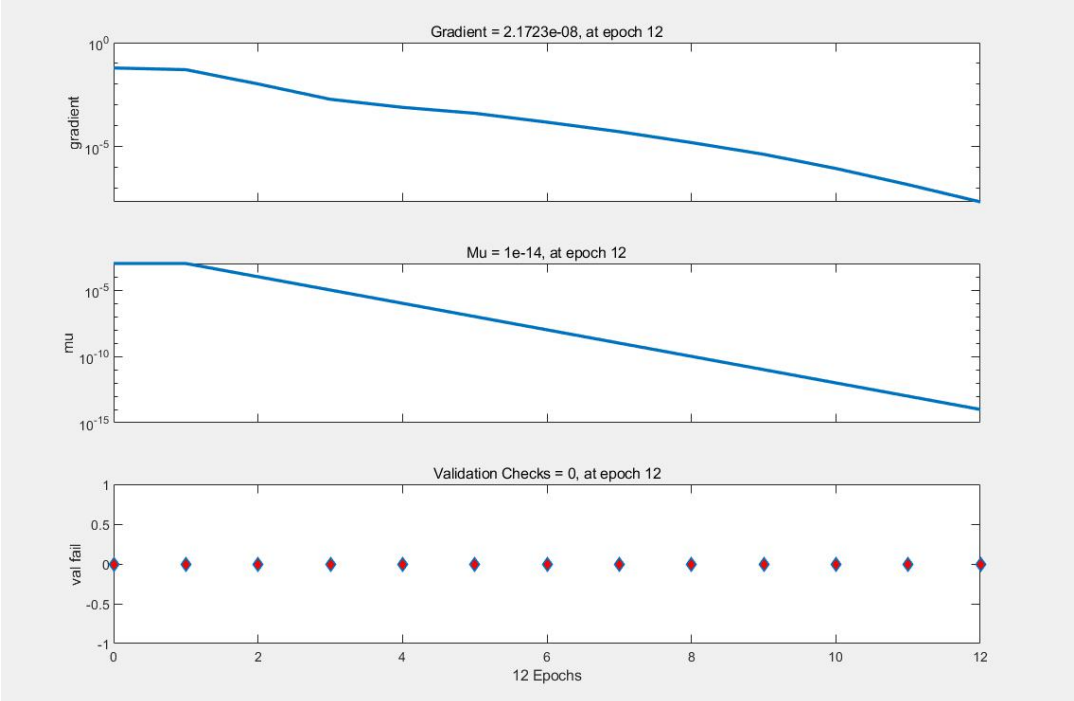
Progress

Epoch:	0	12 iterations	100
Time:		0:00:00	
Performance:	0.225	4.36e-09	0.00
Gradient:	0.0605	2.17e-08	1.00e-07
Mu:	0.00100	1.00e-14	1.00e+10

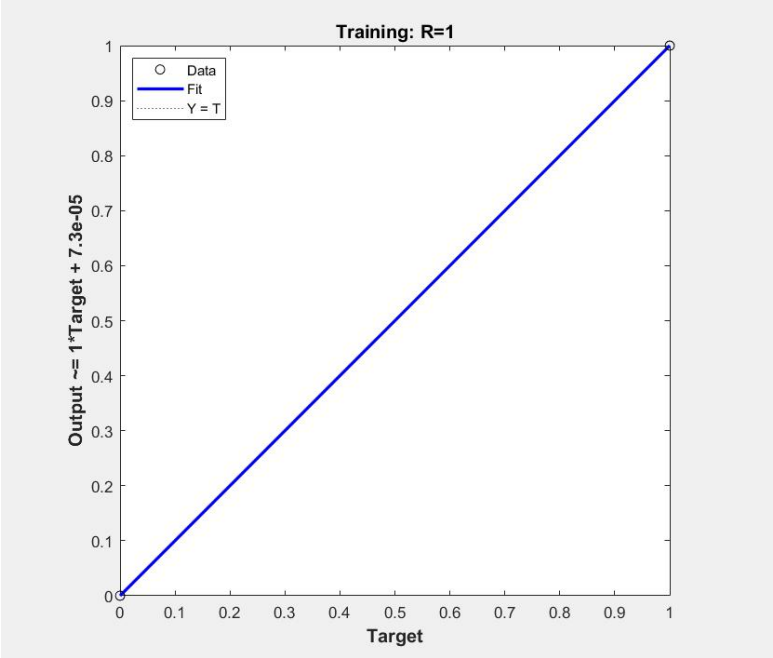
- **Performance plot:**



- **Training state plot:**



- **Regression plot:**



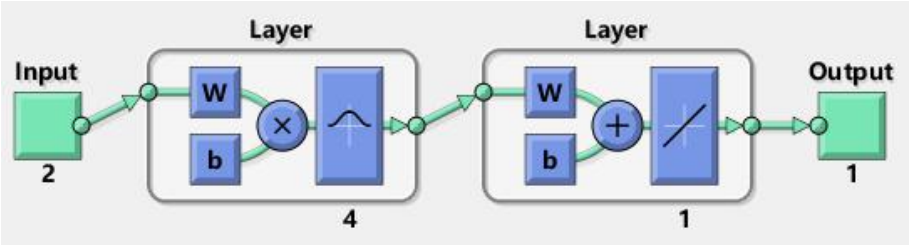
(2) A radial-basis function network with Gaussian function

[1] method 1: (using MATLAB Neural Networks toolbox)

◆ **Code:**

```
%{
Name   : A radial-basis function network with Gaussian function (newrb)
Author : Wang Yue
Date   : 2020.10.20
%}
clear;
dataset = [0 0 0;
           0 1 1;
           1 0 1;
           1 1 0];
x = dataset(:,1:2)';
z = dataset(:,3)';
net = newrb(x, z);
Y = sim(net, x);
view(net);
```

◆ **Diagram:**



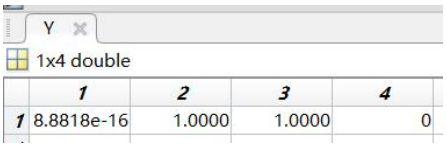
◆ **Properties :**

Mean squared error goal (GOAL):	0.0 (default)
Spread constant (SPRED):	1.0 (default)
Maximum number of neurons(MN):	Q = 4 as the number of vectors (default)
Number of neurons to add between displays (DF):	25 (default)

◆ **The result output values for all four training samples in the end of the iteration :**

*target values with their corresponding predicted values:

x1	x2	z	Y(output data)
0	0	0	0
0	1	1	1
1	0	1	1
1	1	0	0



No. of radbas neurons	= 0
MSE	= 0.25

```
>> final_RBF_XOR
NEWRB, neurons = 0, MSE = 0.25
```

[2] method 2: (using my own program)

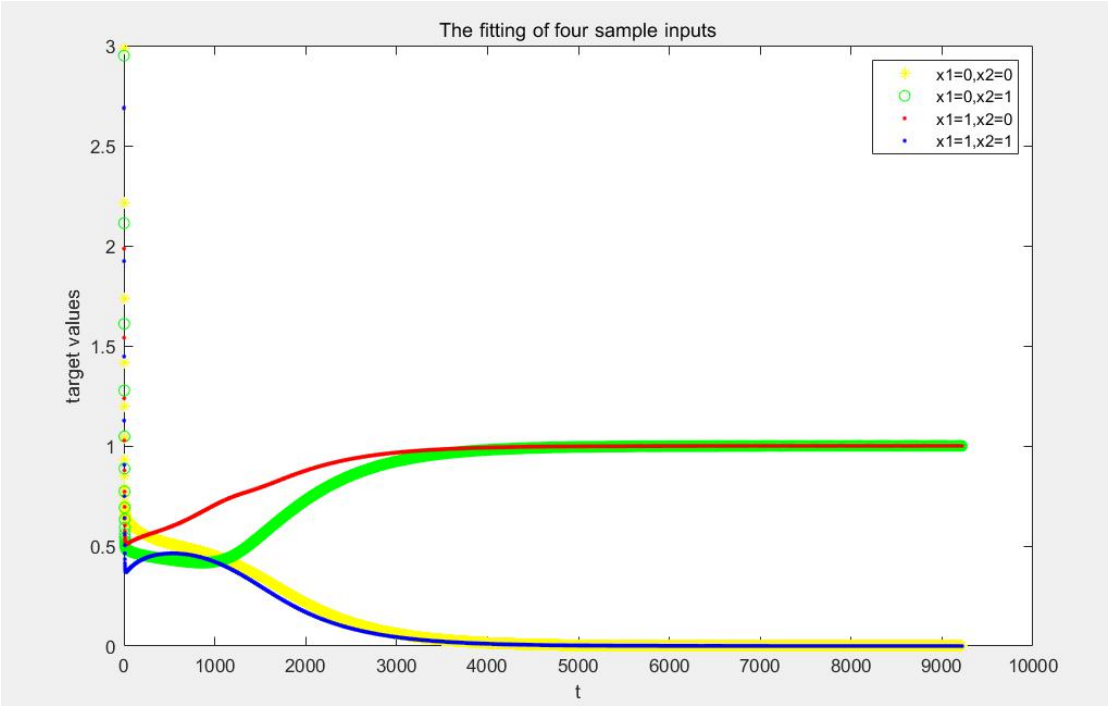
◆ **Code:**

```
%{
Name   : A radial-basis function network with Gaussian function (method2)
Author : Wang Yue
Date   : 2020.10.21
%}
clear
dataset = [0 0 0;
           0 1 1;
           1 0 1;
           1 1 0];
x = dataset(:,1:2);
z = dataset(:,3);
hideNum=8;            %Number of hidden layer neurons
rho=rand(4,hideNum);   %The value of the radial basis function
y=rand(4,1);          %output
w=rand(1,hideNum);     %The weight of the ith neuron in the hidden layer and the output
neuron
sf=rand(1,hideNum);    %The scaling factor of the distance between the sample and the
center of the ith neuron
c=rand(hideNum,2);     %The center of the ith neuron in the hidden layer
t=0;                  %Cumulative number of iterations
sn=0;                 %Same cumulative number of error values
E_pre=0;              %The cumulative error of the previous iteration
```

```
lr=0.05;              %learning rate

while(1)
    t=t+1;
    E=0;
    %Calculate the value of the radial basis function for each sample----------
    for i=1:4
        for j=1:hideNum
            p(i,j)=exp(-sf(j)*(x(i,:)-c(j,:))*(x(i,:)-c(j,:))');
        end
        y(i,t)=w*p(i,:)';
    end
    %Calculate cumulative error-----------------------------------------------
    for i=1:4
        E=E+((y(i,t)-z(i))^2); %Calculate the mean square error
    end
    E=E/2; %accumulated error

    %delta w、sf
    w_d=zeros(1,hideNum);
    sf_d=zeros(1,hideNum);
    for i=1:4
        for j=1:hideNum
            w_d(j)=w_d(j)+(y(i,t)-z(i))*p(i,j);
            sf_d(j)=
sf_d(j)-(y(i,t)-z(i))*w(j)*(x(i,:)-c(j,:))*(x(i,:)-c(j,:))'*p(i,j);
        end
    end
    %update w、sf-------------------------------------------------------------
    w=w-lr*w_d/4;
    sf=sf-lr*sf_d/4;
    %Conditions for iteration termination-------------------------------------
    if(abs(E_pre-E)<1e-10)
        sn=sn+1;
        if(sn==100)
            break;
        end
    else
        E_pre=E;
        sn=0;
    end
end
%plot
a = 1:t;
```

```
b = y(1,:);
c = y(2,:);
d = y(3,:);
e = y(4,:);
plot(a, b, 'y*'); hold on;
plot(a, c, 'go'); hold on;
plot(a, d, 'r.'); hold on;
plot(a, e, 'b.'); hold on;
title('The fitting of four sample inputs');
legend('x1=0,x2=0','x1=0,x2=1','x1=1,x2=0','x1=1,x2=1');
xlabel('t')
ylabel('target values')
hold off;
```

◆ **The result output values for all four training samples :**



◆ **The result output values for all four training samples in the end of the iteration :**

x1	x2	z	Y(output data)
0	0	0	1.2173e-05
0	1	1	1.0000
1	0	1	1.0000
1	1	0	8.4085e-06

9221
1.2173e-05
1.0000
1.0000
8.4085e-06