CS5486 Intelligent Systems – Assignment 1

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.

Perceptron – OR:

Program

```
clear;
eta = 0.05; %learning rate
x = [0,0,1; 0,1,1; 1,0,1; 1,1,1]; %data
z = [-1; 1; 1; 1]; %ground truth
[p,n] = size(x);
X = [-3 -2 -1 0 1 2 3]; %x values for graph
w = [1 0.5 0]; %weight
E = 1; %error
E_threshold = 1e-4;
it = 0; iter = 100; %iteration
%alpha = 1;
%-----------------------------------------------
while (E > E_threshold) && (it < iter)
    for i = 1:p
        if (z(i) == 1)
            scatter(x(i,1),x(i,2), '+'); %plot data points
            hold on
        else
            scatter(x(i,1),x(i,2));
            hold on
        end
    end
%-----------------------------------------------
    it = it + 1;
%-----------------------------------------------
    for i=1:p
        u = x(i,:)*w';
        if u > 0    %transfer function
            y(i) = +1;
        else
            y(i) = -1;
        end
        w = w + eta*(z(i)-y(i))*x(i,:);
    end
%-----------------------------------------------
    mse(it) = 1/p * ((y-z')*(y-z')'); %mean squared error
    sse(it) = ((y-z')*(y-z')'); %sum squared error
    E = mse(it);
%-----------------------------------------------
    Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
    plot(X,Y);
    hold off;
end
```

Figure Mean squared error of perceptron convergence algorithm with a bipolar activation function for the classification of OR function

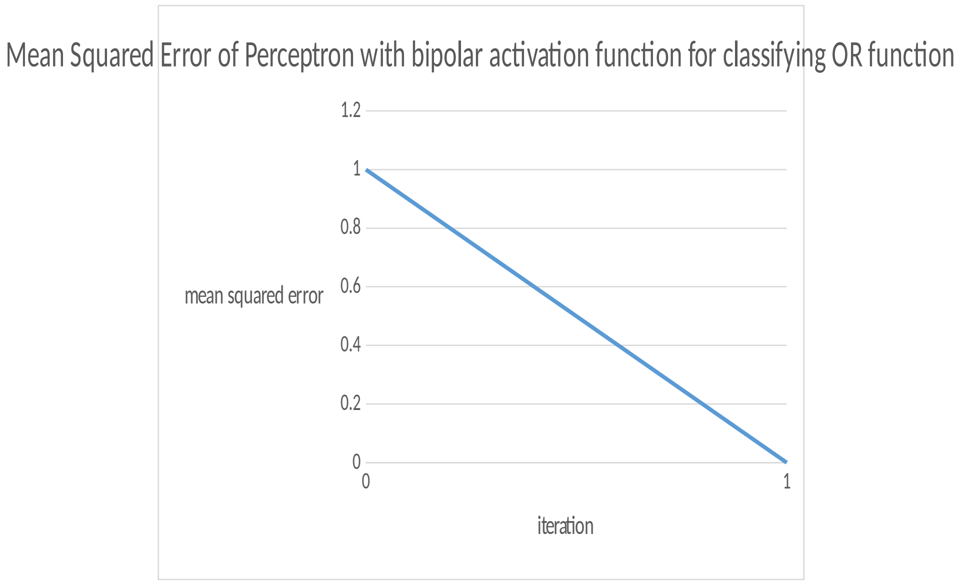


Figure Sum squared error of perceptron convergence algorithm with a bipolar activation function for the classification of OR function

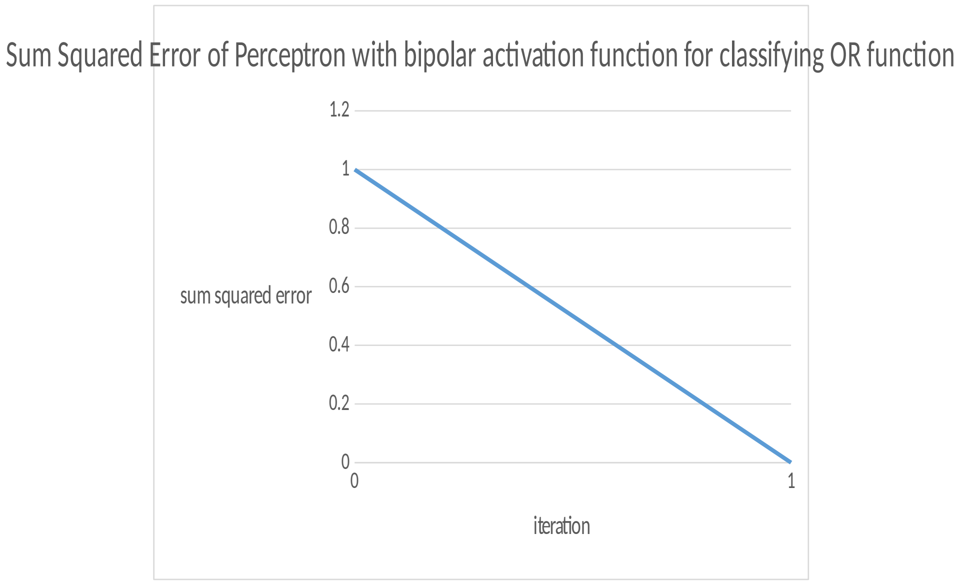


Figure plot generated for perceptron convergence algorithm with bipolar activation functions for the classification of OR function in the last iteration

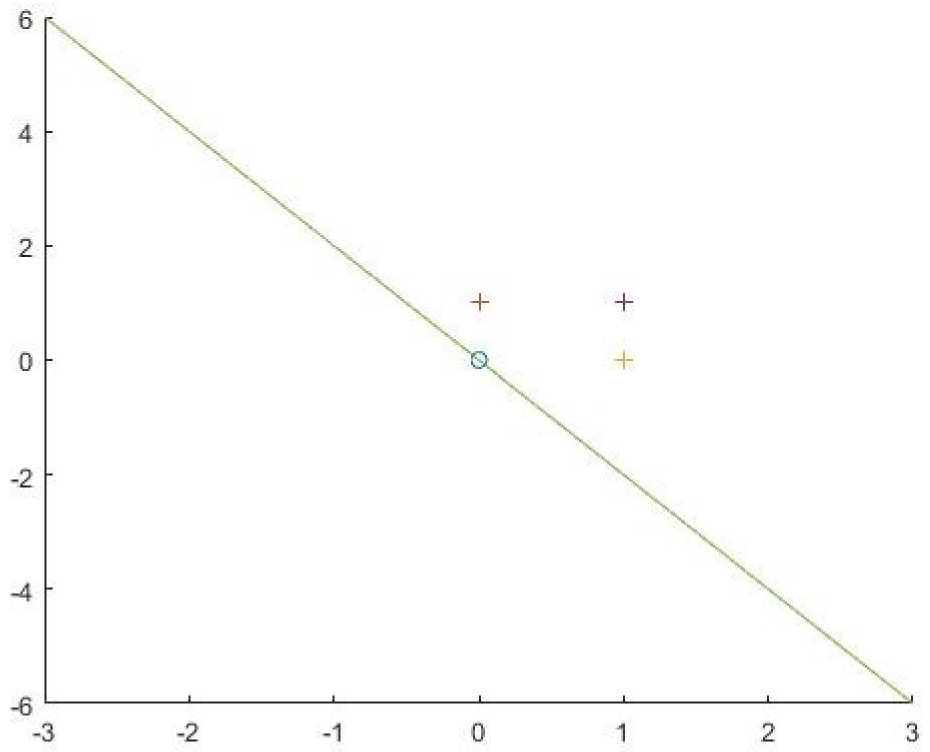


Table target values with their corresponding predicted values generated by the perceptron convergence algorithm for the classification of OR function in the last iteration

x1	x2	y	ŷ	error
0	0	-1	-1	0
0	1	1	1	0
1	0	1	1	0
1	1	1	1	0

To conclude, it is difficult to say whether the perceptron convergence algorithm is able to linearly separate the data points belonging to two different groups for classifying the OR function. The reason is the algorithm with the bipolar function is able to predict the values correctly in just one iteration. The weights of the network never have to change because (z^p – y^p) would be zero for all patterns when the program begins. Hence, the equation of separating line is determined by the weights that it is first assigned, which is [1 0.5 0]. After all, the data points of the OR function could be linearly separated.

<u>Perceptron – XOR:</u>

Program

```
clear;
eta = 0.05; %learning rate
x = [0,0,1; 0,1,1; 1,0,1; 1,1,1]; %data
z = [-1; 1; 1; -1]; %ground truth
[p,n] = size(x);
X = [-3 -2 -1 0 1 2 3]; %x values for graph
w = [1 0.5 0]; %weight
E = 1; %error
E_threshold = 1e-4;
it = 0; iter = 100; %iteration
```

```
%alpha = 1;
%-----------------------------------------------
while (E > E_threshold) && (it < iter)
    for i = 1:p
        if (z(i) == 1)
            scatter(x(i,1),x(i,2), '+'); %plot data points
            hold on
        else
            scatter(x(i,1),x(i,2));
            hold on
        end
    end
%-----------------------------------------------
    it = it + 1;
%-----------------------------------------------
    for i=1:p
        u = x(i,:)*w';
        if u > 0
            y(i) = +1;
        else
            y(i) = -1;
        end
        w = w + eta*(z(i)-y(i))*x(i,:);
    end
%-----------------------------------------------
    mse(it) = 1/p * ((y-z')*(y-z')'); %mean squared error
    sse(it) = ((y-z')*(y-z')'); %sum squared error
    E = mse(it);
%-----------------------------------------------
    Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
    plot(X,Y);
    hold off;
end
```

Figure Mean squared error of perceptron convergence algorithm with bipolar activation functions for the classification of XOR function

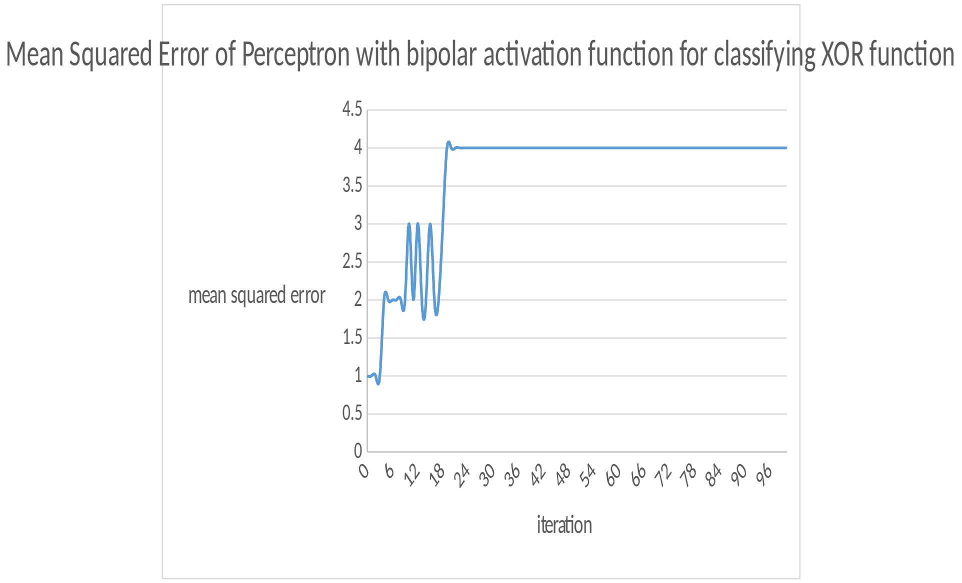


Figure Sum squared error of perceptron convergence algorithm with a bipolar activation function for the classification of XOR function

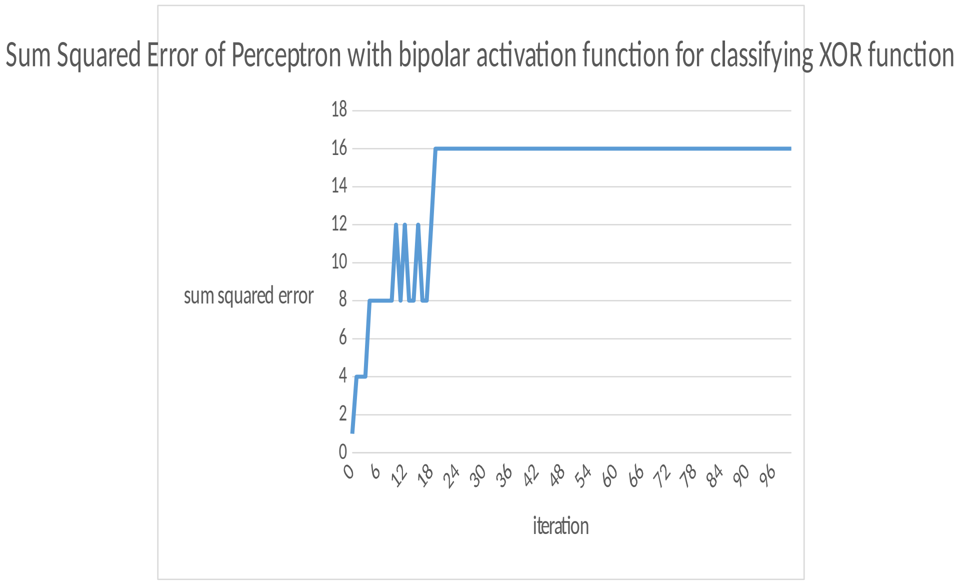


Figure plot generated for perceptron convergence algorithm with a bipolar activation function for the classification of XOR function in the last iteration

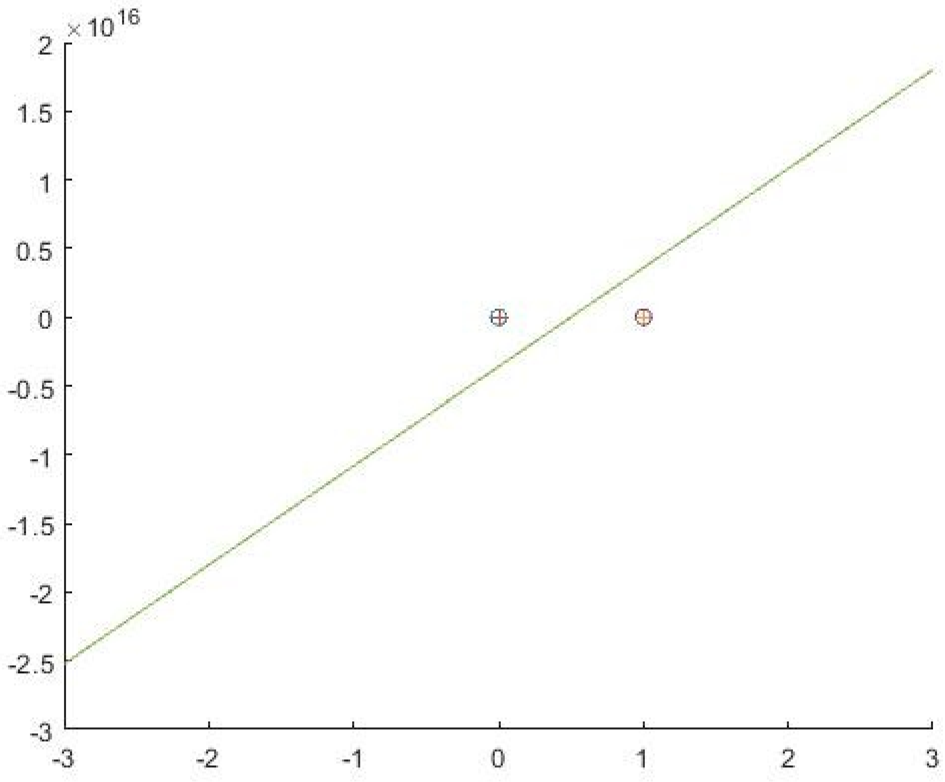


Table target values with their corresponding predicted values generated by the perceptron convergence algorithm for the classification of XOR function in the last iteration

x1	x2	y	ŷ	Error
0	0	-1	1	-2
0	1	1	-1	2
1	0	1	-1	2

1	1	-1	1	-2

Figure Mean squared error of ADALINE for the classification of OR function

To conclude, the perceptron convergence algorithm is not able to linearly separate the data points of the XOR function as it is shown on the plot. The separating line divides the data points into two sides in a way that there are two different classes of data points on both sides. It clearly shows data points of the XOR function cannot be linearly separated. The mean squared error fluctuates from iteration 8 to 17 and stay at the mse value of 4 or sse value of 16, which is not optimal.

<u>ADALINE – OR:</u>

Program

```
clear;
eta = 0.05; %learning rate
x = [0,0,1; 0,1,1; 1,0,1; 1,1,1]; %data
z = [-1; 1; 1; 1]; %ground truth
[p,n] = size(x);
X = [-3 -2 -1 0 1 2 3]; %x values for graph
w = [1 0.5 0]; %weight
E = 1; %error
E_threshold = 1e-4;
it = 0; iter = 100; %iteration
alpha = 1;
%-----------------------------------------------
while (E > E_threshold) && (it < iter)
   for i = 1:p
      if (z(i) == 1)
         scatter(x(i,1),x(i,2), '+'); %plot data points
         hold on
      else
         scatter(x(i,1),x(i,2));
         hold on
      end
   end
%-----------------------------------------------
   it = it + 1;
%-----------------------------------------------
   for i=1:p
      for j=1:p
         y(j) = x(j,:)*w'*alpha;
      end
      for k=1:p
         w = w + eta*(z(k)-y(k))*x(k,:);
      end
   end
%-----------------------------------------------
   mse(it) = 1/p * ((y-z')*(y-z')'); %mean squared error
   sse(it) = ((y-z')*(y-z')'); %sum squared error
   E = mse(it);
%-----------------------------------------------
   Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
   plot(X,Y);
   hold off;
end
```

Figure Mean squared error of ADALINE for the classification of OR function

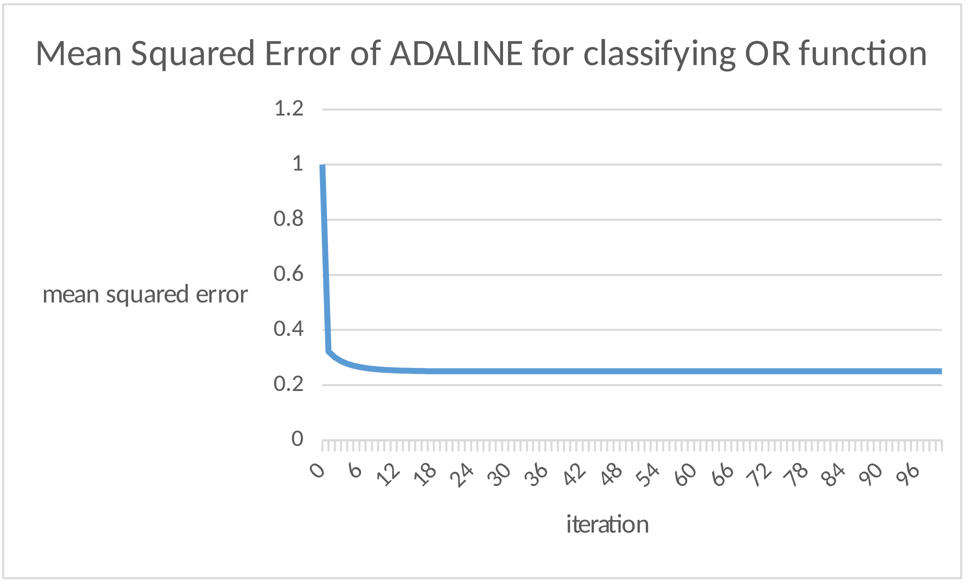


Figure Sum squared error of ADALINE for the classification of OR function

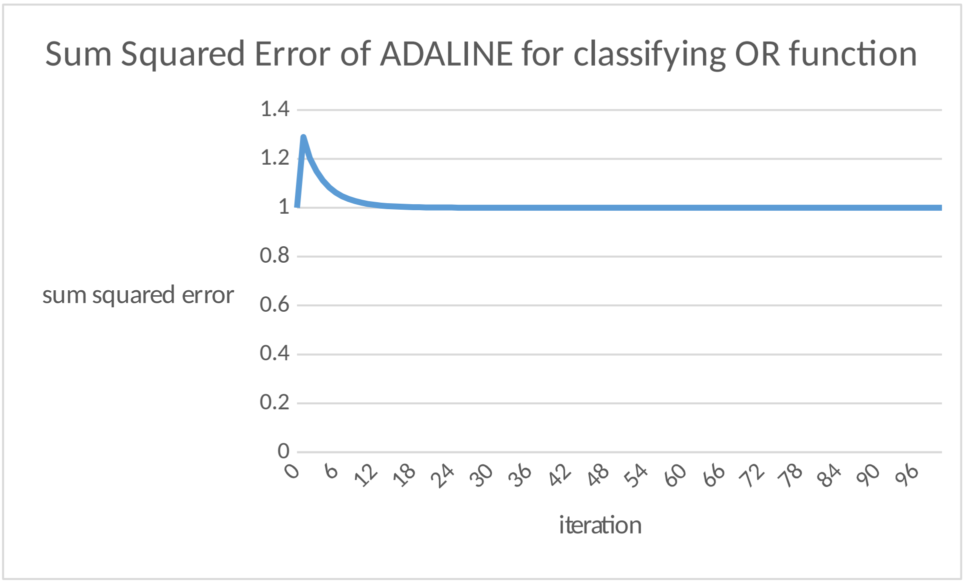


Figure plot generated for ADALINE for the classification of OR function in the last iteration

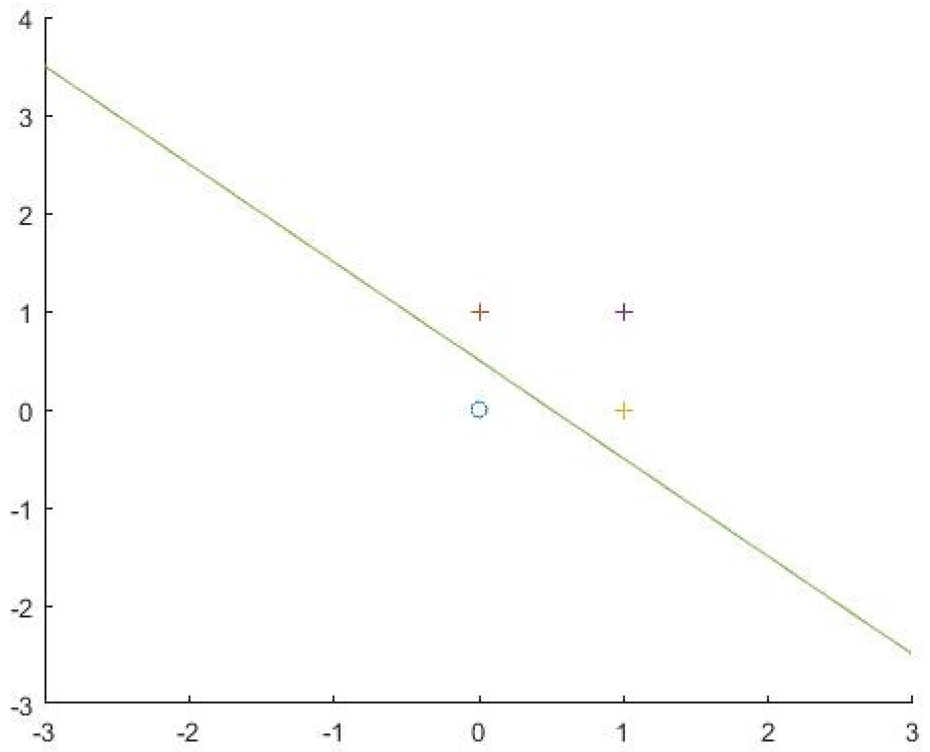


Table target values with their corresponding predicted values generated by ADALINE for the classification of OR function in the last iteration

x1	x2	y	ŷ	error
0	0	-1	-0.5	-0.5
0	1	1	0.5	0.5
1	0	1	0.5	0.5
1	1	1	1.5	-0.5

To conclude, adaline is able to linearly separate the data points of the OR function as it is shown in the plot even though the mse maintained at 0.25 or sse at 1, which never meets the goal (i.e. error_threshold). Without doing the experiment, we are still able to know that the data points of the OR function cannot be linearly separated. If the network parameters are adjusted more optimally, the mse might be able to reach the goal.

ADALINE – XOR:

Program

```
clear;
eta = 0.05; %learning rate
x = [0,0,1; 0,1,1; 1,0,1; 1,1,1]; %data
z = [-1; 1; 1; -1]; %ground truth
[p,n] = size(x);
X = [-3 -2 -1 0 1 2 3]; %x values for graph
w = [1 0.5 0]; %weight
E = 1; %error
E_threshold = 1e-4;
it = 0; iter = 100; %iteration
```

```
alpha = 1;
%-----------------------------------------------
while (E > E_threshold) && (it < iter)
    for i = 1:p
        if (z(i) == 1)
            scatter(x(i,1),x(i,2), '+'); %plot data points
            hold on
        else
            scatter(x(i,1),x(i,2));
            hold on
        end
    end
%-----------------------------------------------
    it = it + 1;
%-----------------------------------------------
    for i=1:p
        for j=1:p
            y(j) = x(j,:)*w'*alpha;
        end
        for k=1:p
            w = w + eta*(z(k)-y(k))*x(k,:);
        end
    end
%-----------------------------------------------
    mse(it) = 1/p * ((y-z')*(y-z')'); %mean squared error
    sse(it) = ((y-z')*(y-z')'); %sum squared error
    E = mse(it);
%-----------------------------------------------
    Y = -(w(1,1)/w(1,2))*X-(w(1,3)/w(1,2)); %equation for graph
    plot(X,Y);
    hold off;
end
```

Figure Mean squared error of ADALINE for the classification of XOR function

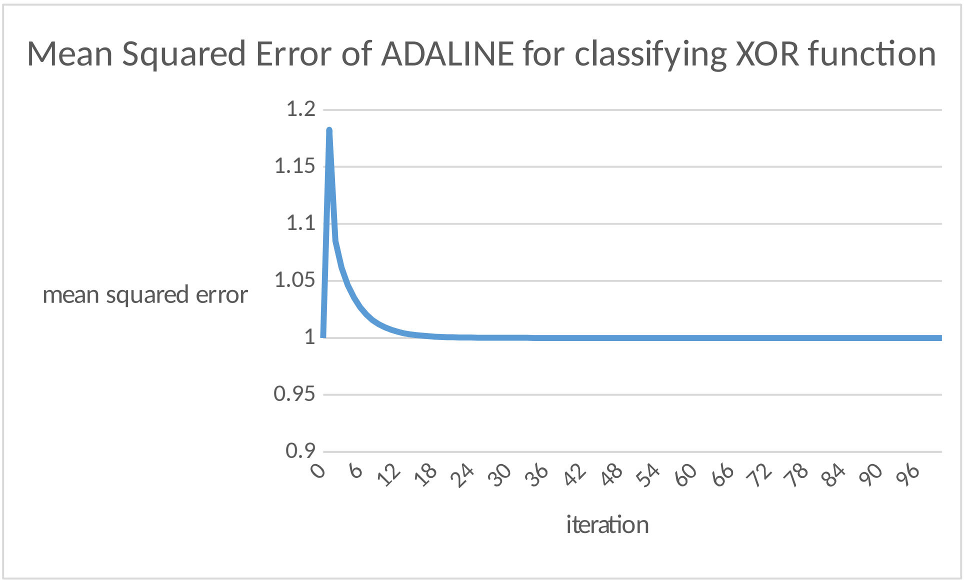


Figure Sum squared error of ADALINE for the classification of XOR function

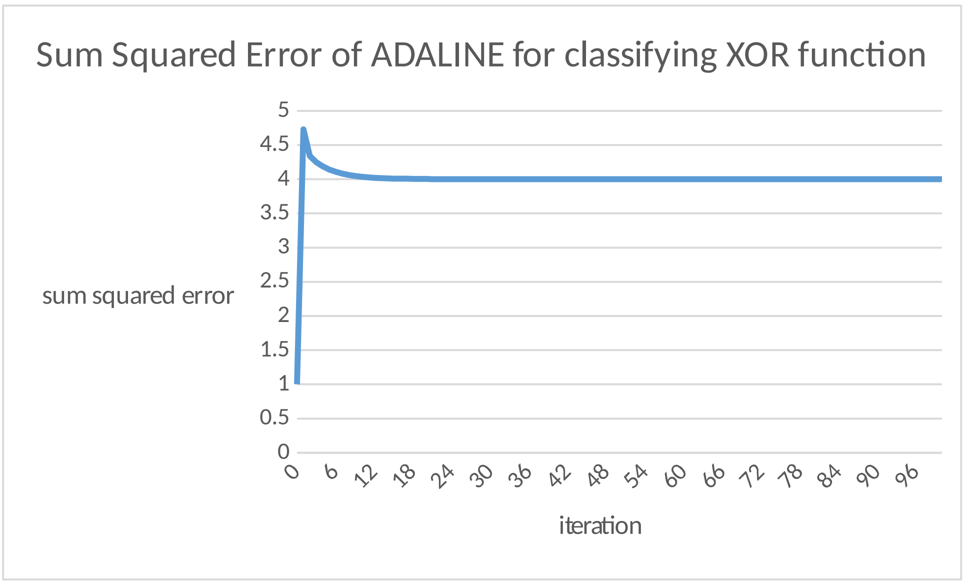


Figure plot generated for ADALINE for the classification of XOR function in the last iteration

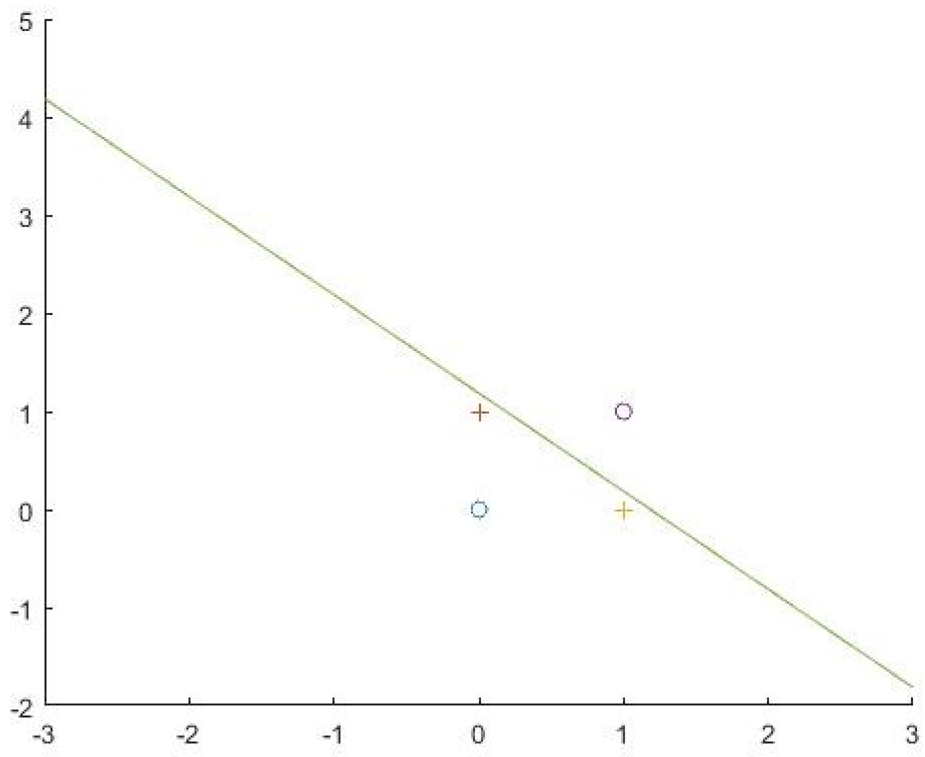


Table target values with their corresponding predicted values generated by ADALINE for the classification of XOR function in the last iteration

x1	x2	y	ŷ	Error
0	0	-1	-1.556e-06	-1
0	1	1	-2.445e-07	1
1	0	1	-2.439e-07	1
1	1	-1	1.068e-06	-1

To conclude, adaline is not able to linearly separate the data points of the XOR function. The mean squared error increases rapidly in the first few iterations and the value drops to 1 and maintains at this value until the last iteration as it never meets the goal. The mse or sse values are higher than that of the OR function. This is because the data points of the XOR function is not linearly separable.

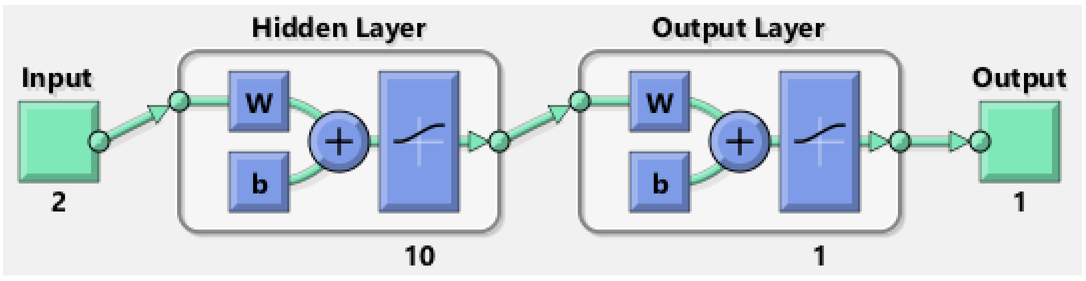
2. Using MATLAB Neural Networks toolbox or your own program to simulate a multilayer Perceptron with unipolar sigmoid activation functions and a radial-basis function network with Gaussian function for classifying XOR data of two variables. Show your detailed results (i.e., convergence behaviors and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

A multilayer Perceptron with unipolar sigmoid activation functions:

Code

```
clear
d = [0 0 0,
0 1 1,
1 0 1,
1 1 0];
t = d(:,3)';
i = d(:,1:2)';
net = newff([0 1; 0 1],[10 1], {'logsig' 'logsig'}, 'trainlm', 'learngdm', 'mse');
net.trainParam.epochs = 50;
net.trainParam.goal = 0;
net = train(net,i,t);
Y = sim(net, i);
view(net);
```

Diagram a multiplayer perceptron neural network with unipolar sigmoid activation functions



Network Properties:

Network Type: Feed-Forward Backpropagation

Training function: Levenberg-Marquardt (TRAINLM)

Adaption learning function: LEARNGDM

Performance function: Mean Squared Error (MSE)

Row cell array of output processing functions: Default is {‘remconstantrows’,’mapminmax’}

Data division function: dividerand (default)

Calculations: MEX

Number of layers: 2

Properties for Layer 1:

Number of neurons: 10

Transfer Function: LOGSIG

Properties for Layer 2:

Transfer Function: LOGSIG

Stopping conditions:

Epochs: 50

Goal: 0

Time: Infinite

The result in the end of the iteration:

It goes for 8 iterations.

Output Data: [1.2095e-04 0.9999 1.0000 1.4359e-04]

Error Data: [-1.21e-04 7.76e-05 4.64e-05 -1.44e-04]

Mean Squared Error: 4.34e-08 ≈ 0

Table target values with their corresponding predicted values generated by a multilayer Perceptron with unipolar sigmoid activation functions for the classification of XOR function

x1	x2	y	ŷ	error
0	0	0	1.2095e-04	-1.21e-04
0	1	1	0.9999	7.76e-05
1	0	1	1.0000	4.64e-05
1	1	0	1.4359e-04	-1.44e-04

Figure performance plot generated by a multilayer Perceptron with unipolar sigmoid activation functions for the classification of XOR function

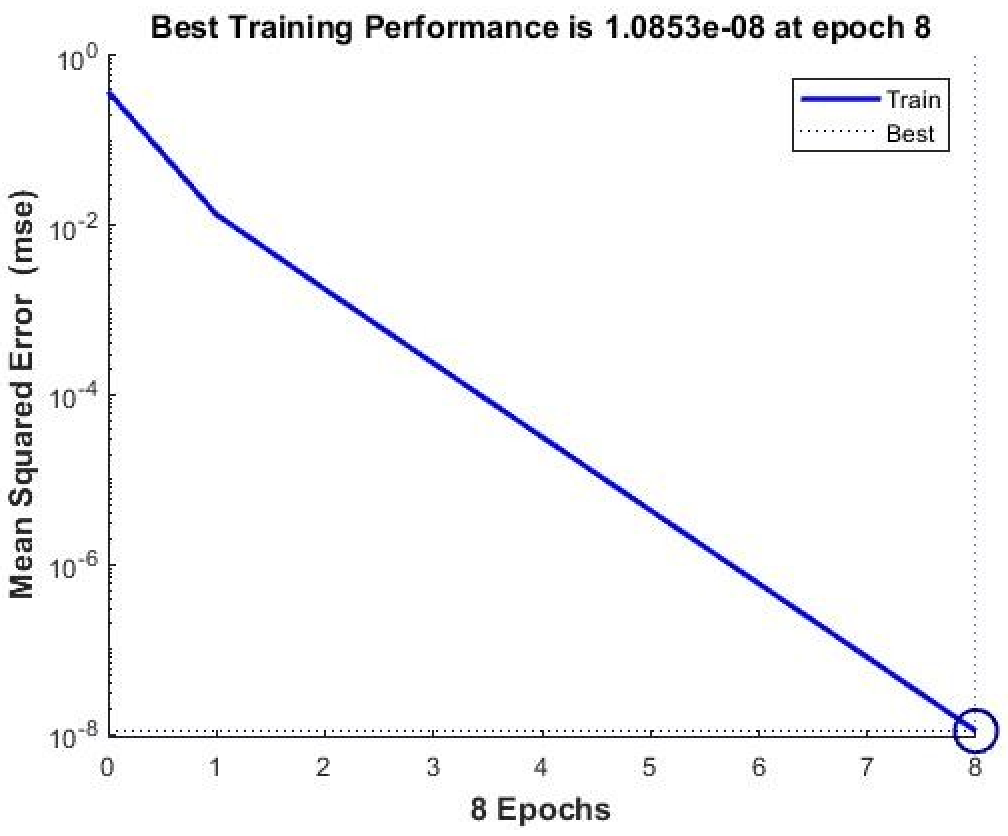


Figure training state plot generated by a multilayer Perceptron with unipolar sigmoid activation functions

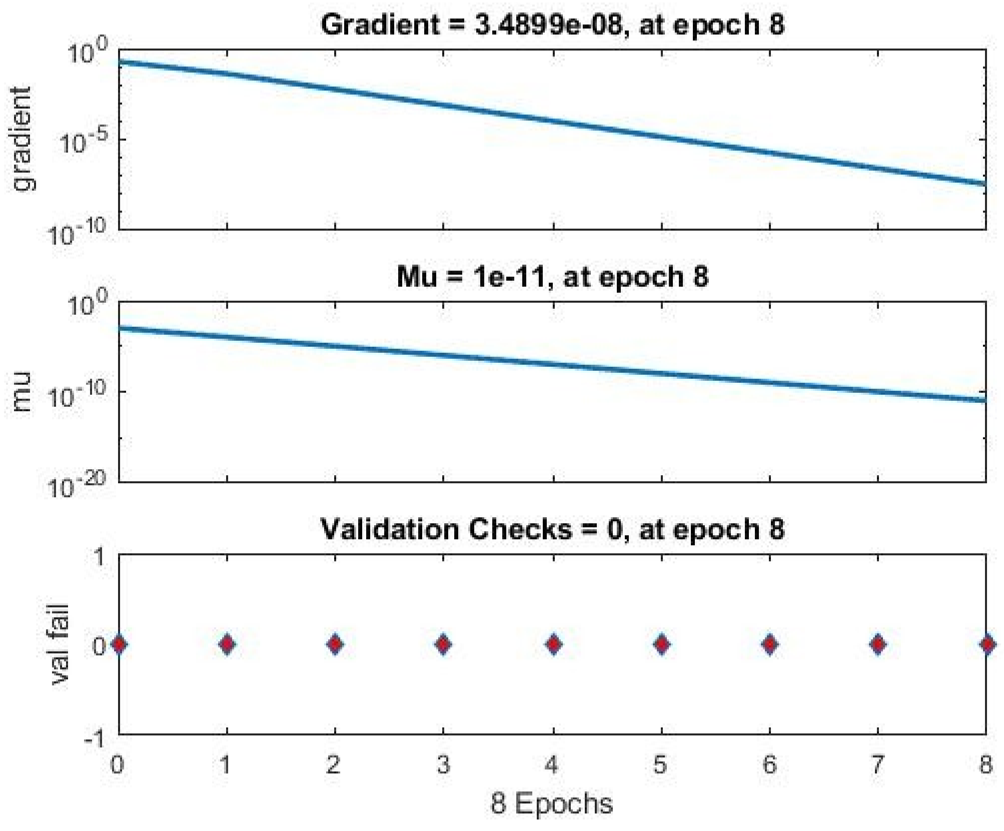
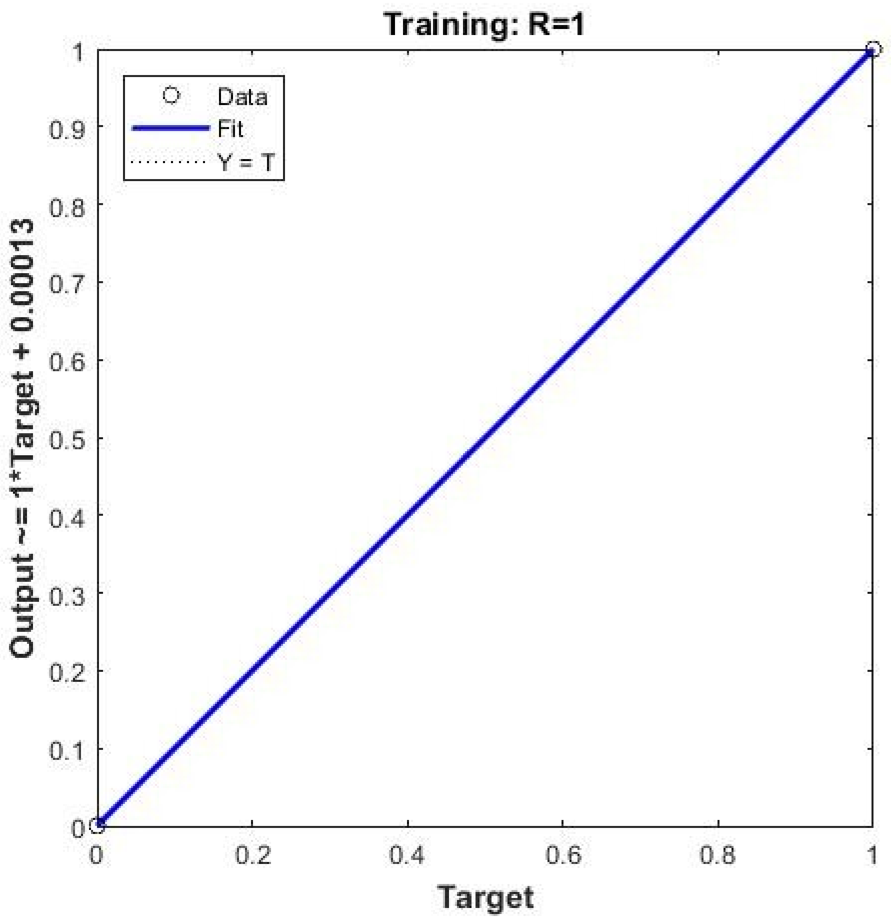


Figure regression plot generated by a multilayer Perceptron with unipolar sigmoid activation functions



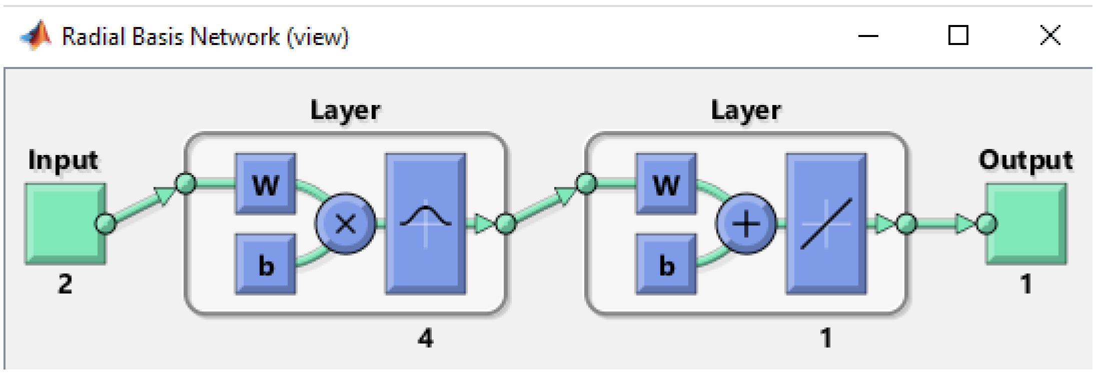
To conclude, the mean squared error (4.34e-08) is so low that it can be considered as zero. The multiplayer perceptron neural network with unipolar sigmoid activation functions successfully predict the target values of the XOR function with four sample inputs as it is shown by its performance.

A radial-basis function network with Gaussian function:

Code

```
clear
d = [0 0 0,
0 1 1,
1 0 1,
1 1 0];
t = d(:,3)';
i = d(:,1:2)';
net = newrb(i, t);
Y = sim(net, i);
view(net);
```

Diagram a radial-basis function neural network with Gaussian function



Network Properties:

Mean squared error goal: 0.0 (default)

Spread constant: 1.0 (default)

Maximum number of neurons: Q = 4 as the number of vectors (default)

Number of neurons to add between displays: 25 (default)

Results:

No. of radbas neurons = 0

MSE = 0.25

Figure target values with their corresponding predicted values generated by the radial-basis function network with Gaussian function for the classification of XOR function

x1	x2	y	ŷ	Error
0	0	0	0	0
0	1	1	1	0
1	0	1	1	0
1	1	0	0	0

To conclude, since the input vector has no error in the beginning, not a single radbas neuron is added with weights equal to that vector according to the algorithm. The purelin layer weights never have to be redesigned to minimize any error. This is because the four vectors provided are too simple for the radial-basis neural network to predict.