Assignment 3

Experiment 6 and 7

Title: Learning Rules for single layer single continuous perceptron

Name of Student: Sangeet Agrawal PRN No. 21070122140

DoP6: 26 Aug

DoP7: 2 Sep DoS: 10 Sep

Experiment 6

Title: Implement Delta learning rule for single layer single continuous perceptron

Aim: To design and simulate delta learning rule for single layer single continuous perceptron using MATLAB. To simulate testing (Recall) delta learning rule for single layer single continuous perceptron

* Theory

→ Delta Learning Rule:

It is used to train single-layer continuous perceptrons by adjusting weights to minimize the error betʷ the desired and actual outputs. The weight update formula is

$$W = W + c \times (D - Y) \times X,$$

where c is the learning rate, D is the desired output, and Y is the ~~act~~ actual output. The rule aims to minimize error iteratively until convergence is achieved.

* Procedure:

1) Initialize input X, desired output D, weights W and learning rate c.

2) Add bias to X.

3) Train using the Delta Rule: update weights iteratively until error is minimized.

4) Test using final weights to predict outputs and calculate errors.

5) Plot desired vs predicted outputs.

Input:

X=[10 2; 2 -5; 5 5]

D = [1 -1 -1 ; -1 1 -1 ; -1 -1 1]

W=[1 -2 0; 0 -1 2; 1 3 -1] , c = 0.1

Code:

```
X = [10 2; 2 -5; 5 5]; % Inputs
D = [1 -1 -1; -1 1 -1; -1 -1 1]; % Desired outputs
W = [1 -2 0; 0 -1 2; 1 3 -1]; % Initial weights
c = 0.1;  % Learning rate (η)
% Add bias input to X
X = [X, -1*ones(size(X,1), 1)];
epochs = 38;
% Training phase using Delta Learning Rule
for i = 1:epochs
   % Calculate the net input and apply the activation function
(sign function)
   Y = sign(W * X');

   % Update weights using the delta rule
   W = W + c * (D - Y) * X;

   % Calculate the total error to check for convergence
   total_error = sum(sum(abs(D - Y)));

   % Stop training if error reaches zero
   if total_error == 0
       fprintf('Training converged with zero error at epoch %d\n',
i);
       break;
   end
end
disp('Final weights after training:');
disp(W);
% Testing phase (Recall)
Y_test = sign(W * X');
error = D - Y_test;
disp('Desired Output (D):');
disp(D);
disp('Output after testing:');
disp(Y_test);
disp('Error after testing:');
disp(error);
% Plotting the results
figure;
hold on;
```

```
colors = {'r', 'g', 'b'};
for i = 1:size(X, 1)
   for j = 1:size(Y_test, 1)
       if Y_test(i, j) == 1
           plot(X(i, 1), X(i, 2), '+', 'Color', colors{j},
'MarkerSize', 10, 'LineWidth', 1); % '+' for predicted output
       else
           plot(X(i, 1), X(i, 2), '*', 'Color', colors{j},
'MarkerSize', 10, 'LineWidth', 1); % '*' for predicted output
       end
   end
end
xlabel('Input X1');
ylabel('Input X2');
title('Desired vs. Predicted Outputs after Perceptron Training');
legend('Desired Class 1', 'Desired Class 2', 'Desired Class 3',
...
      'Predicted Class 1', 'Predicted Class 2', 'Predicted Class
3');
hold off;
grid on;
```

Output:

Command Window

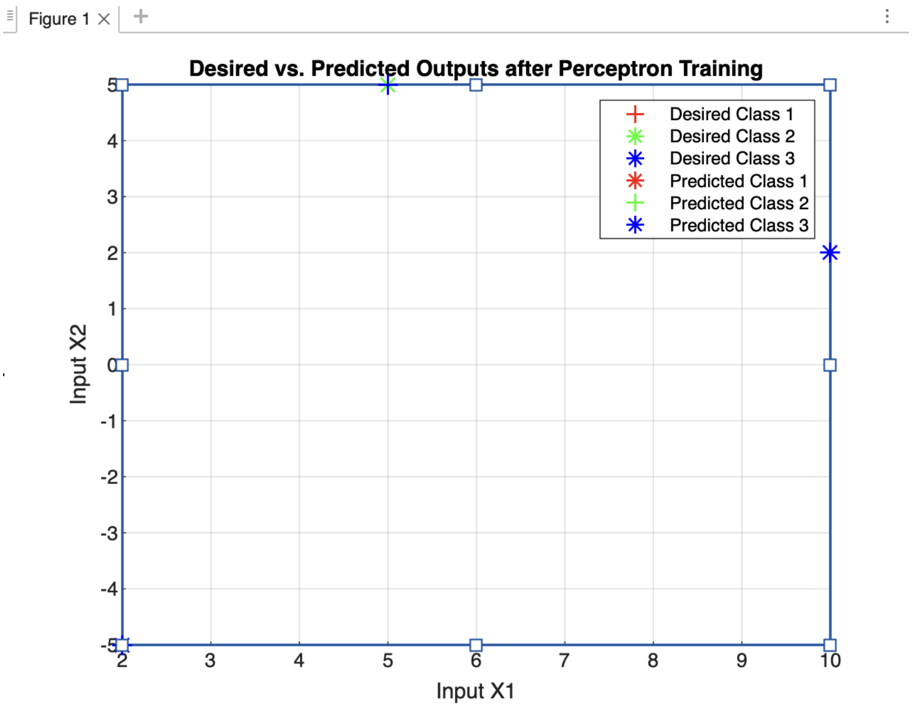
```
>> Experiment_6
Training converged with zero error at epoch 38
Final weights after training:
    1.1000   -0.3000    4.5000
         0   -1.0000    2.0000
   -1.0000    2.6000   -0.8000

Desired Output (D):
     1    -1    -1
    -1     1    -1
    -1    -1     1

Output after testing:
     1    -1    -1
    -1     1    -1
    -1    -1     1

Error after testing:
     0     0     0
     0     0     0
     0     0     0

>> |
```



* Conclusion:

The Delta learning rule successfully minimized the error and converged to an optimal weight configuration for the single layer perceptron.

Experiment 7

Title: Perceptron learning rule for single layer single continuous perceptron

Aim: To design and simulate perceptron learning rule for single layer single continuous perceptron using MATLAB. To simulate testing (Recall) delta learning rule for single layer single continuous perceptron

* Theory

→ Perceptron Learning Rule:

It trains single-layer perceptrons by adjusting weights to correct misclassifications. Weights are updated using

$$W = W + c \times e \times X,$$

where e is the error, c is the learning rate and X is the input. The training continues until all examples are correctly classified, ensuring convergence for linearly separable data.

* Procedure:

1) Initialize input X, desired output D, weights W, and learning rate c

2) Add bias to X.

3) Train using the Perceptron rule: update weights for each misclassification until ~~all~~ convergence.

4) Test using final weights to ~~produce~~ predict outputs and compute errors.

5) Plot desired vs predicted outputs.

Input:

X=[10 2; 2 -5; 5 5]

D = [1 -1 -1 ; -1 1 -1 ; -1 -1 1]

W=[1 -2 0; 0 -1 2; 1 3 -1] , c = 0.1

Code:

```
X = [10 2; 2 -5; 5 5]; % Inputs
D = [1 -1 -1; -1 1 -1; -1 -1 1]; % Desired outputs
W = [1 -2 0; 0 -1 2; 1 3 -1]; % Initial weights
c = 0.1;  % Learning rate
% Add bias input to X
X = [X, -1*ones(size(X,1), 1)];
epochs = 20;
% Training phase using Perceptron Learning Rule
for i = 1:epochs
   total_error = 0;  % Initialize total error for the epoch

   % Iterate over each input vector
   for j = 1:size(X, 1)
       % Calculate the net input and apply the activation function (sign
function)
       Y = sign(W * X(j, :)');

       % Calculate the error between desired and predicted output
       e = D(:, j) - Y;

       % Update weights if there is any error (misclassification)
       if any(e)  % If there is a misclassification
           W = W + c * e * X(j, :);  % Update weights
           total_error = total_error + sum(abs(e));  % Update total
error
       end
   end

   % Stop training if total error is zero (i.e., no misclassification)
   if total_error == 0
       fprintf('Training converged with zero error at epoch %d\n', i);
       break;
   end
end
disp('Final weights after training:');
disp(W);
```

```
% Testing phase (Recall)
Y_test = sign(W * X');  % Testing with the same input data
error = D - Y_test;
disp('Desired Output (D):');
disp(D);
disp('Output after testing:');
disp(Y_test);
disp('Error after testing:');
disp(error);
% Plotting the results
figure;
hold on;
colors = {'r', 'g', 'b'};
for i = 1:size(X, 1)
   for j = 1:size(Y_test, 1)
       if Y_test(j, i) == 1
           plot(X(i, 1), X(i, 2), '+', 'Color', colors{j}, 'MarkerSize',
10, 'LineWidth', 1); % '+' for predicted output
       else
           plot(X(i, 1), X(i, 2), '*', 'Color', colors{j}, 'MarkerSize',
10, 'LineWidth', 1); % '*' for predicted output
       end
   end
end
xlabel('Input X1');
ylabel('Input X2');
title('Desired vs. Predicted Outputs after Perceptron Training');
legend('Desired Class 1', 'Desired Class 2', 'Desired Class 3', ...
      'Predicted Class 1', 'Predicted Class 2', 'Predicted Class 3');
hold off;
grid on;
```

Output:

Command Window

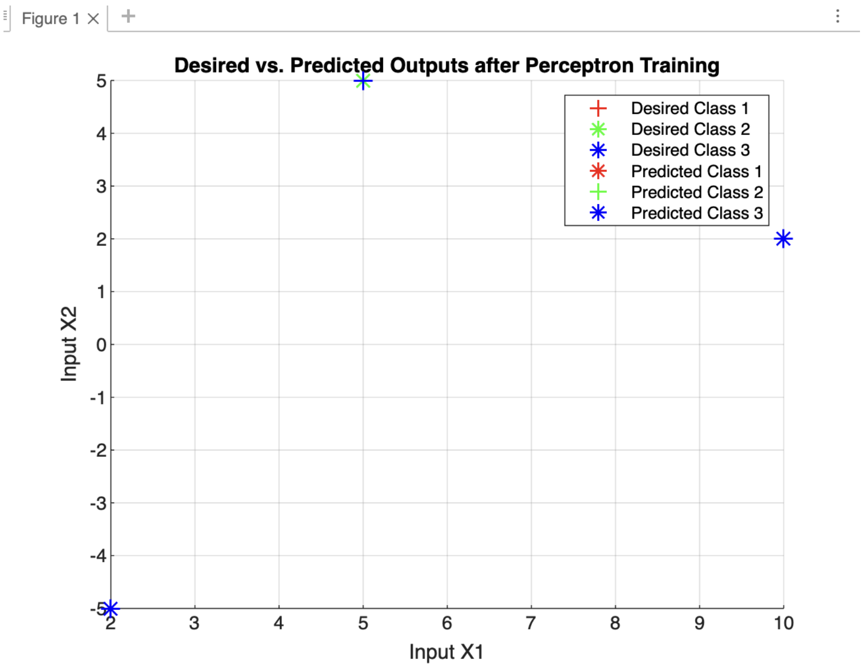
```
>> Experiment_7
Training converged with zero error at epoch 20
Final weights after training:
    1.0000   -0.2000    4.2000
         0   -1.0000    2.0000
   -1.0000    2.6000   -0.8000

Desired Output (D):
     1    -1    -1
    -1     1    -1
    -1    -1     1

Output after testing:
     1    -1    -1
    -1     1    -1
    -1    -1     1

Error after testing:
     0     0     0
     0     0     0
     0     0     0

>> |
```



* ~~Conclu~~ Conclusion:

The Perceptron learning rule effectively adjusted the weights to correctly classify all inputs, achieving convergence for linearly separable data.