## Table of Contents

```
%FML_LAB_1
% Author: ARISH
% Hebbian Learning for Threshold Neuron

clear; clc;

% Input Vectors
X = [ 1.0  -2.0   1.5   0.0;
      1.0  -0.5  -2.0  -1.5;
      0.0   1.0  -1.0   1.5 ];  % Each row is x1, x2, x3

% Initial Weight
w = [1.0  -1.0   0.0   0.5];     % Initial weight vector

% Hebbian Learning Rule
for i = 1:size(X,1)
    w = w + X(i,:);             % Update weights: w_new = w_old + x_i
end

% Output
fprintf('\nQ1)');
fprintf('\nFinal weight vector after Hebbian learning:\n');
disp(w);
```

```
Q1)
Final weight vector after Hebbian learning:
    3.0000   -2.5000   -1.5000    0.5000
```

## Q2) Continuous Hebbian Learning with nonlinear activation

```
%clear; clc;

% Input Vectors
X = [ 1.0  -2.0   1.5   0.0;
      1.0  -0.5  -2.0  -1.5;
      0.0   1.0  -1.0   1.5 ];  % Each row is x1, x2, x3

% Initial Weight
W = [1.0  -1.0   0.0   0.5];     % Initial weight vector
```

```
% Activation Function
f = @(x) (2 ./ (1 + exp(-x))) - 1;

%Hebbian Learning
for i = 1:size(X,1)
    x_i = X(i,:);
    y_i = f(dot(W, x_i));        % Continuous output
    W = W + y_i * x_i;           % Hebbian update: W_new = W_old + y_i * x_i
end

% Output
fprintf('\nQ2)');
fprintf('\nFinal weight vector after continuous Hebbian learning:\n');
disp(W);
```

```
Q2)
Final weight vector after continuous Hebbian learning:
    1.8277   -3.7044    2.4454   -0.7832
```

## Q3) Perceptron Training with Noisy Data Probe

```
%clear; clc;

% Input Patterns
X = [ 1.0  -2.0   0.0  -1.0;
      0.0  -1.5  -0.5  -1.0;
     -1.0   1.0   0.5  -1.0 ];    % Each row is an input vector

T = [-1; -1; 1];                  % Target outputs

% Initial Weights
w = [1.0  -1.0   0.0   0.5];      % Initial weight vector
eta = 1;                          % Learning rate
max_iter = 50;

% Training Loop
for iter = 1:max_iter
    error_count = 0;
    for i = 1:size(X,1)
        x_i = X(i,:);
        y = sign(dot(w, x_i));
        if y ~= T(i)
            w = w + eta * T(i) * x_i;
            error_count = error_count + 1;
        end
    end
    if error_count == 0
        break;
    end
end
```

```
% Output
fprintf('\nQ3)');
fprintf('\nFinal trained weight vector:\n');
disp(w);
fprintf('Training completed in %d iterations.\n', iter);

% Noisy Data Probe
fprintf('\nTesting with noisy inputs:\n');

% Add small Gaussian noise to each original input
rng(0);  % For reproducibility
noise_level = 0.2;
X_noisy = X + noise_level * randn(size(X));

for i = 1:size(X_noisy,1)
    x_i = X_noisy(i,:);
    y = sign(dot(w, x_i));
    fprintf('Noisy input %d: %s → Output: %d (Target: %d)\n', ...
        i, mat2str(round(x_i,2)), y, T(i));
end
```

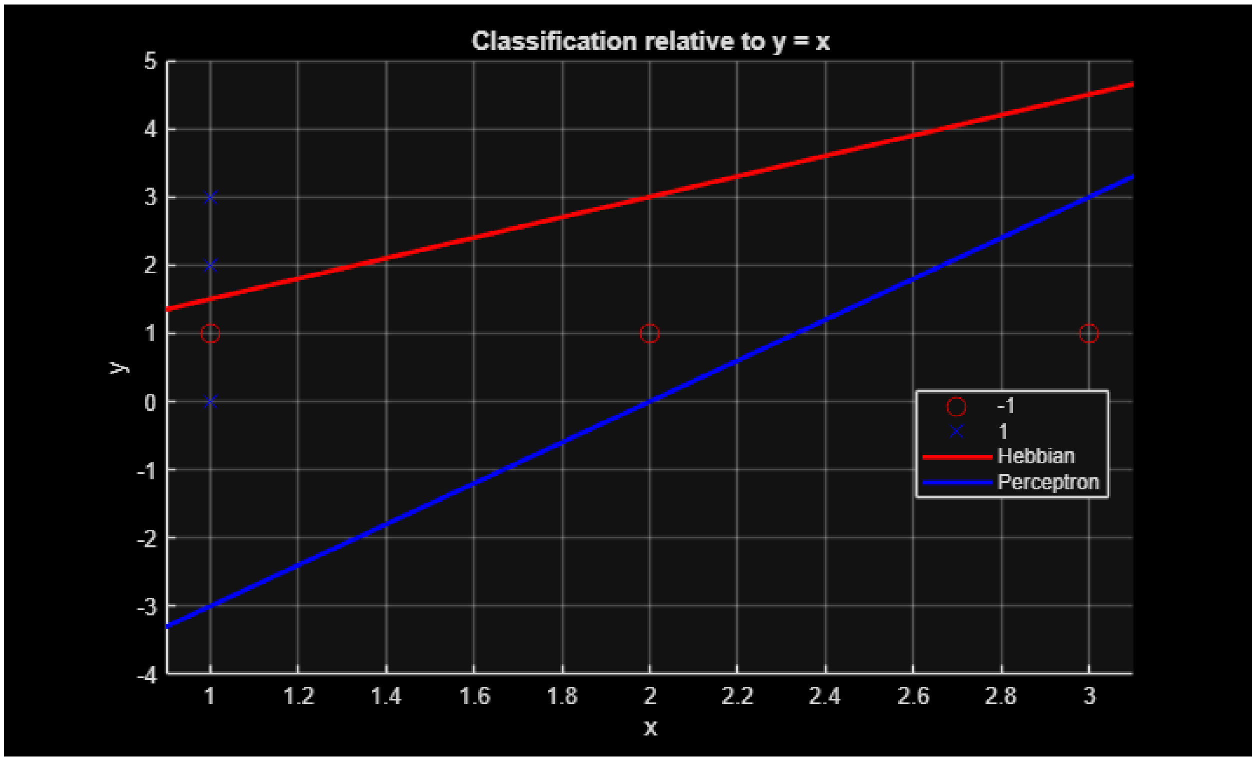
```
Q3)
Final trained weight vector:
   -1.0000    2.0000    0.5000    0.5000

Training completed in 2 iterations.

Testing with noisy inputs:
Noisy input 1: [1.11 -1.83 -0.09 -0.45] → Output: -1 (Target: -1)
Noisy input 2: [0.37 -1.44 -0.43 -1.27] → Output: -1 (Target: -1)
Noisy input 3: [-1.45 0.74 1.22 -0.39] → Output: 1 (Target: 1)
```

## Q4) Classification of points relative to y = x using Hebbian and Perceptron rules

```
%clear; clc;

% Generate Sample Data
% Class C1: points below y = x → target = -1
% Class C2: points above y = x → target = +1

X = [1 1; 1 0; 2 1; 1 2; 3 1; 1 3];   % Sample points
T = [-1; 1; -1; 1; -1; 1];            % Targets based on y < x or y > x

X_aug = [X ones(size(X,1),1)];        % Add bias term

% (a) Hebbian Learning
w_hebb = [0 0 0];                     % Initial weights
for i = 1:size(X_aug,1)
    w_hebb = w_hebb + T(i) * X_aug(i,:);
```

```
end

fprintf('Hebbian weights: [%.2f %.2f %.2f]\n', w_hebb);

% (b) Perceptron Learning
w_perc = [0 0 0];                    % Initial weights
eta = 1; max_iter = 50;

for iter = 1:max_iter
    error_count = 0;
    for i = 1:size(X_aug,1)
        y = sign(dot(w_perc, X_aug(i,:)));
        if y ~= T(i)
            w_perc = w_perc + eta * T(i) * X_aug(i,:);
            error_count = error_count + 1;
        end
    end
    if error_count == 0
        break;
    end
end

fprintf('Perceptron weights: [%.2f %.2f %.2f]\n', w_perc);
fprintf('Perceptron converged in %d iterations.\n', iter);

% Plotting
figure; hold on;
gscatter(X(:,1), X(:,2), T, 'rb', 'ox', 8);

% Decision boundary: w1*x + w2*y + bias = 0 → y = -(w1*x + bias)/w2
x_vals = linspace(min(X(:,1))-1, max(X(:,1))+1, 100);

% Hebbian boundary
y_hebb = -(w_hebb(1)*x_vals + w_hebb(3)) / w_hebb(2);
plot(x_vals, y_hebb, 'r-', 'LineWidth', 2, 'DisplayName','Hebbian');

% Perceptron boundary
y_perc = -(w_perc(1)*x_vals + w_perc(3)) / w_perc(2);
plot(x_vals, y_perc, 'b-', 'LineWidth', 2, 'DisplayName','Perceptron');

xlabel('x'); ylabel('y'); title('Classification relative to y = x');
legend; grid on;
```

```
Hebbian weights: [-3.00 2.00 0.00]
Perceptron weights: [-3.00 1.00 6.00]
Perceptron converged in 50 iterations.
```



*Published with MATLAB® R2025a*