```
clc; clear all; close all;
%% Import test data
test_img = loadMNISTImages('t10k-images.idx3-ubyte');
test_lab = loadMNISTLabels('t10k-labels.idx1-ubyte');
test_lab = test_lab';
%% A using weight bias algorithm
tic
acc = zeros(6, 1);
for n = 1:6
    w =find_weights(5, n);
    % Test
    pattern=10;element=size(test_img, 2);pixel=size(test_img, 1);
    e=ones(pattern,element);
    for j=1:pattern
        for i=1:element
            y=sign(test_img(:,i)'*w(j,:)');
            if mod(j,10)==test_lab(i)
                d=1;
            else
                d=-1;
            end
            e(j,i)=d-y;
        end
    end
    sum=0;
    for j=1:pattern
        for i=1:element
            if e(j,i)~=0
                sum=sum+1;
            end
        end
    end
    % Find accuracy
    acc(n) = sum/element;
end
toc
%%
plot(1:6, acc * 100, 'LineWidth', 2);
title('Training Modes vs. Test Accuracy');
xlabel('Number of Training Modes in 10^4');
ylabel('Accuracy in Percentage');

clc; clear all; close all;
%%
% Load MNIST.
train_data = loadMNISTImages('train-images.idx3-ubyte');
train_labs = loadMNISTLabels('train-labels.idx1-ubyte');

targetValues = 0.*ones(10, size(train_labs, 1));
for n = 1: size(train_labs, 1)
    targetValues(train_labs(n) + 1, n) = 1;
end

units = 10;

alpha = 0.1;

activation = @logisticSigmoid;
dactivation = @dLogisticSigmoid;

batchSize = 100;
epochs = 500;

[w_between, w_output, error] = train(activation, dactivation, ...
    units, train_data, targetValues,epochs, batchSize, alpha);
```

```
66
67  test_data = loadMNISTImages('t10k-images.idx3-ubyte');
68  test_labs = loadMNISTLabels('t10k-labels.idx1-ubyte');
69
70  [correct, err] = validate(activation, w_between, w_output, test_data, test_labs);
71
72  fprintf('Classification errors: %d\n', err);
73  fprintf('Correctly classified: %d\n', correct);
74  fprintf('Accuracy: %d\n', correct/(err+correct));
75
76  function [w]=find_weights(times, modes)
77  data = loadMNISTImages('train-images.idx3-ubyte');
78  labels = loadMNISTLabels('train-labels.idx1-ubyte');
79  data = data(:, 1:modes*10000);
80  pattern=10;element=size(data, 2);pixel=size(data, 1);
81  alpha=0.1;% Learing rate
82  w=zeros(pattern,pixel);%weights
83  e=ones(pattern,element);%error
84  d=0;
85
86  for count=1:times
87      for j=1:pattern
88          for i=1:element
89              y=sign(data(:,i)'*w(j,:)');
90              if mod(j,10)==labels(i)
91                  d=1;
92              else
93                  d=-1;
94              end
95              e(j,i)=d-y;
96              w(j,:) = w(j,:)+alpha.*e(j,i).*data(:,i)';
97          end
98      end
99  end
100
101 end
102 function [w_bet, w_out, error] = train(act, deact, units, data, labs, epochs, batchSize, alpha)
103     trainingSetSize = size(data, 2);
104     datasize = size(data, 1);
105     pattern = size(labs, 1);
106     w_bet = rand(units, datasize);
107     w_out = rand(pattern, units);
108
109     w_bet = w_bet./size(w_bet, 2);
110     w_out = w_out./size(w_out, 2);
111
112     n = zeros(batchSize);
113
114     figure; hold on;
115
116     for t = 1: epochs
117         for k = 1: batchSize
118             % Select which input vector to train on.
119             n(k) = floor(rand(1)*trainingSetSize + 1);
120
121             % Propagate the input vector through the network.
122             inputVector = data(:, n(k));
123             hiddenActualInput = w_bet*inputVector;
124             hiddenOutputVector = act(hiddenActualInput);
125             outputActualInput = w_out*hiddenOutputVector;
126             outputVector = act(outputActualInput);
127
128             targetVector = labs(:, n(k));
129
130             % Backpropagate the errors.
```

```
            outputDelta = deact(outputActualInput).*(outputVector - targetVector);
            hiddenDelta = deact(hiddenActualInput).*(w_out'*outputDelta);

            w_out = w_out - alpha.*outputDelta*hiddenOutputVector';
            w_bet = w_bet - alpha.*hiddenDelta*inputVector';
        end

        % Calculate the error for plotting.
        error = 0;
        for k = 1: batchSize
            inputVector = data(:, n(k));
            targetVector = labs(:, n(k));

            error = error + norm(act(w_out*...
                act(w_bet*inputVector)) - targetVector, 2);
        end
        error = error/batchSize;

        plot(t, error,'*');
        xlabel('Epochs'), ylabel('Crossentropy');
        title('Error Rate vs. Number of Epochs');
    end
end
function [correct, err] = validate(act, w_bet, w_out, data, labels)

    testSetSize = size(data, 2);
    err = 0;
    correct = 0;

    for n = 1: testSetSize
        inputVector = data(:, n);
        outputVector = evaluate(act, w_bet, w_out, inputVector);

        class = decisionRule(outputVector);
        if class == labels(n) + 1
            correct = correct + 1;
        else
            err = err + 1;
        end
    end
end

function class = decisionRule(output)
    max = 0;
    class = 1;
    for i = 1: size(output, 1)
        if output(i) > max
            max = output(i);
            class = i;
        end
    end
end

function output = evaluate(act, w_bet, w_out, data)
    output = act(w_out*act(w_bet*data));
end
function y = logisticSigmoid(x)
    y = 1./(1 + exp(-x));
end
function y = dLogisticSigmoid(x)
    y = logisticSigmoid(x).*(1 - logisticSigmoid(x));
end
```