UNIVERSITY OF Cincinnati

Homework 3: Intelligent Systems Simple Classifiers

Presented by:
Aswin Balasubramaniam

Course Name/Number: Intelligent Systems (EECE 6036)
Class section: 001
Due Date: November 12th, 2019

Problem 1: Classification using Backpropagation with Momentum

Problem Summary:

In this problem a two-layer feed-forward neural network is implemented. The network is trained using backpropagation with momentum to classify the handwritten numbers (0 to 9) correctly. The provided MNIST dataset is used to train and test the network generated.

System Description:

Number of hidden neurons: The number of hidden neurons was set to varying values from 150 to 200 during the testing phase. Due to time constraints and limitations of code execution time an exhaustive investigation of the effect of hidden neurons was not conducted, but it was observed that 150 hidden neurons provided the desired results of hit rates for the given set of parameters (learning rates, momentum, and epochs).

Learning rate: The learning rate was set to a small value of 0.1 when tests were conducted. The learning rates for both the layers were set to the same value. This generated desired results and hence was not changed.

Momentum: The momentum parameter, alpha, was set to a value between 0 and 1. To get the errors to converge to a minimum value alpha was set to 0.5. Together with a learning rate of 0.1 and alpha of 0.5 the observed hit rates were desirable and were set as the parameters for the network.

Output threshold criteria: The initial output threshold criteria was set following the suggestion given in the homework document. A threshold was set to decide if a neuron's output should be a 0 or 1. Thresholds of 0.75 and 0.55 to detect 1's and 0.25 and 0.1 to detect 0's were set. This did not provide consistent results as the maximum values from the network kept fluctuating leading to network outputs to be set as 0's. The criteria that was implemented searched for the highest output value (highest firing neuron) generated using the network and set that value to 1 while setting the remaining values to 0.

Rule for choosing the initial weights: The weights were initialized following the Xavier initialization method formulated by Glorot & Bengio. The weights were initialized to random values between (-a, +a) where $a \sim \sqrt{\frac{6}{N_S+N_T}}$ where Ns = number of neurons in the source layer and N_T = number of neurons in target layer. When the weights were initially to set to higher values (-1 to 1), the calculated weight and input sums were of high values and generated sigmoid values of 1 consistently.

Stopping criteria: The stopping criteria for training was determined after multiple trials of using epoch values ranging from 100 to 200. The number of epochs was set to 100 as it generated desired results and increasing the hit rate above 170 started decreasing the hit rate values observed for testing set despite increasing the hit rate values for the training set. Setting the epoch to a value higher than 170 made the network to start overfitting and thereby losing the ability to generate desired results.

Results:

Table 1.1 shows the confusion matrix generated using the training dataset of the final network (last epoch). Table 1.2 shows the confusion matrix generated using the testing dataset. Figure 1.1 shows the time series plot of training set error values vs. epochs.

Confusion Matrix of Training Set for Final Network												
		Classification of Numbers										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
True Values of Numbers	**0**	362	0	0	0	0	0	0	0	0	0	362
	1	0	464	1	0	0	0	1	0	0	0	466
	2	0	1	409	1	3	2	2	2	2	0	422
	3	0	0	3	391	0	0	0	4	1	0	399
	4	0	0	1	0	404	0	0	0	0	0	405
	5	1	0	0	0	0	361	1	0	0	1	364
	6	1	0	0	0	1	1	380	0	0	0	383
	7	1	0	0	0	0	0	0	405	0	0	406
	8	0	0	1	0	0	0	0	0	385	0	386
	9	2	2	0	4	2	1	1	0	2	393	407
	Total	367	467	415	396	410	365	385	411	390	394	**4000**

Table 1.1: Confusion matrix of training set for final network

Confusion Matrix of Testing Set												
		Classification of Numbers										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**Total**
True Values of Numbers	**0**	94	0	0	0	0	1	2	0	1	0	98
	1	0	102	0	0	0	0	1	1	1	0	105
	2	1	0	97	1	2	0	0	3	4	0	108
	3	0	1	2	91	0	4	0	1	2	0	101
	4	0	0	0	0	91	0	1	0	0	3	95
	5	1	0	0	1	0	85	1	0	3	1	92
	6	2	1	0	0	0	2	74	0	0	0	79
	7	0	3	4	0	2	0	0	95	0	2	106
	8	0	0	1	0	1	2	2	0	96	1	103
	9	0	1	0	1	7	0	0	1	3	100	113
	Total	98	108	104	94	103	94	81	101	110	107	**1000**

Table 1.2: Confusion matrix of testing set

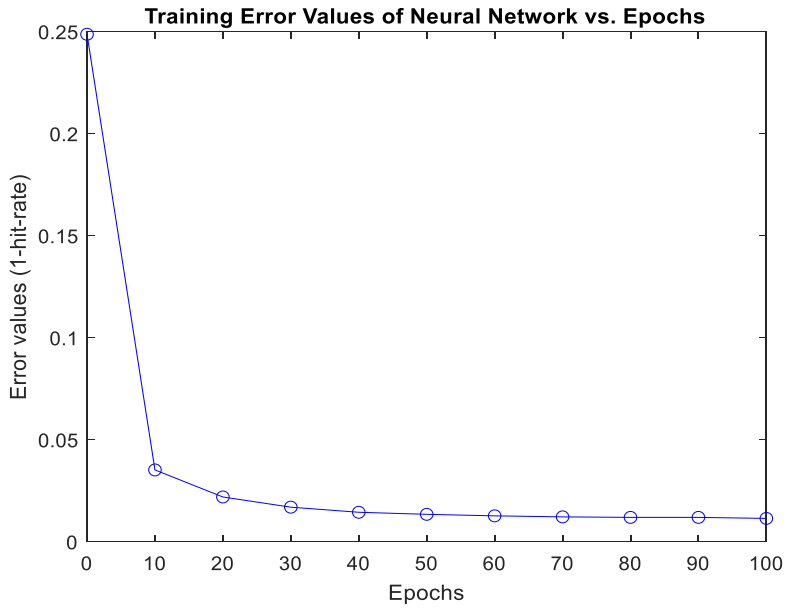

Figure 1.1: Plot of error values vs. epochs for the training set

Analysis of Results:

Using the obtained results it can be seen that the generated two-layered neural network was able to classify the images very accurately using the training and testing datasets. In Table 1.1 the output of the final network using the training dataset provides a hit rate of 98.85% and in Table 1.2 the output of the network using the test set provides a hit rate of 91.9%. Figure 1.1 shows how the error rate decreases as the number of epochs are increased.

Using the confusion matrix of the training and testing sets in Table 1.1 it can be seen that numbers 0, 1, 4, 5, 6, and 8 were detected with a higher accuracy value when compared to the numbers 2, 3, 7 and 9. This suggests that numbers 2, 3 and 9 are relatively difficult to classify using the generated network. Using more varied datasets for training would probably help improve the classification and increase the hit rate values.

Problem 2: Reconstruction using Auto-Encoder

Problem Summary:

In this problem an autoencoder is trained using the same data in Problem 1, to reconstruct the input that is provided to the network and obtain a good set of features that represent the data set. A one hidden layer feedforward network with 784 inputs, 784 outputs, and 150 hidden neurons is trained. The network is trained using backpropagation with momentum and J_2 loss function is used to evaluate its performance.

System Description:

Learning rate: The learning rate was set to 0.01 when tests were conducted. The learning rates for both the layers were set to the same value. This generated desired results when compared to a learning rate of 0.1 and produced lower loss function values within the specified number of training epochs.

Momentum: The momentum parameter, alpha, was set to 0.5 to get the errors to converge to a minimum value quickly. Together with a learning rate of 0.01 and alpha of 0.5 the observed J_2 loss function values were low.

Rule for choosing the initial weights: The weights were initialized following the Xavier initialization method formulated by Glorot & Bengio as mentioned in Problem 1.

Stopping criteria: The number of epochs was set to 100 as it generated desired results. Other values of epochs were not tested due to time constraints.

Results:

Figure 2.1 shows the mean of loss function values of the training (final network) and testing datasets. Figure 2.2 shows mean of loss function values for each number on the training (final network) and testing datasets. Figure 2.3 shows the time series plot of mean loss function value of training set vs. epochs.

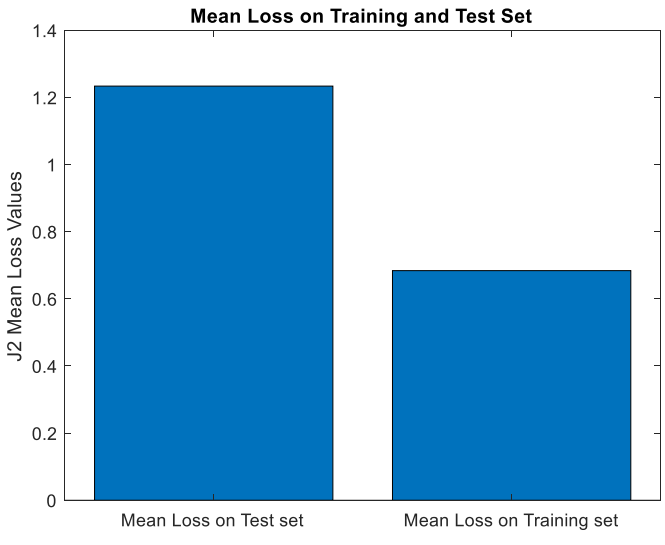

Figure 2.1: Mean of loss function value on training and testing dataset

Figure 2.2: Mean of loss function value for each number on training and testing dataset

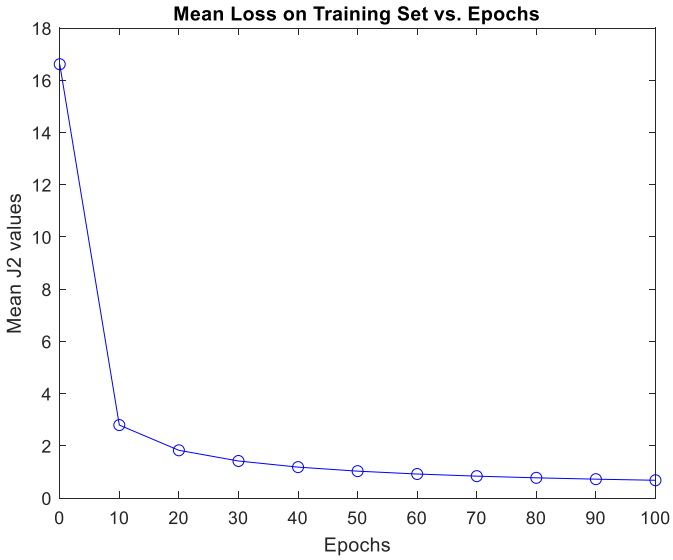

Figure 2.3: Plot of mean of loss function value on training dataset vs. epochs

Features:

Figure 2.4 shows the images of features learned by 20 random hidden neuron weights in problem 2. Figure 2.5 shows the images of features learned by corresponding 20 random hidden neuron weights in problem 1.

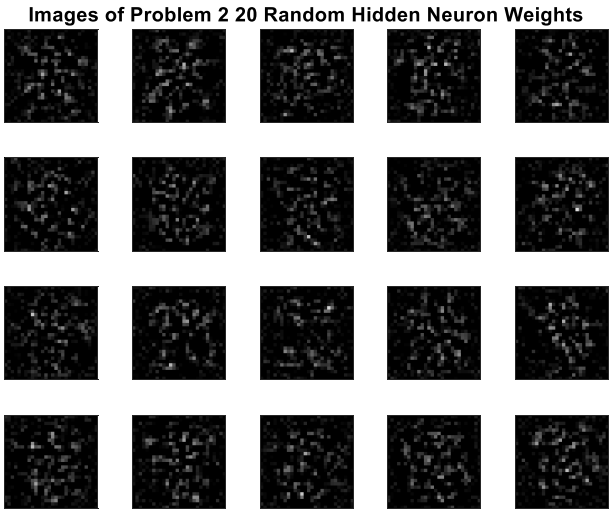

Figure 2.4: Image of features learned by the 20 random hidden neurons in problem 2

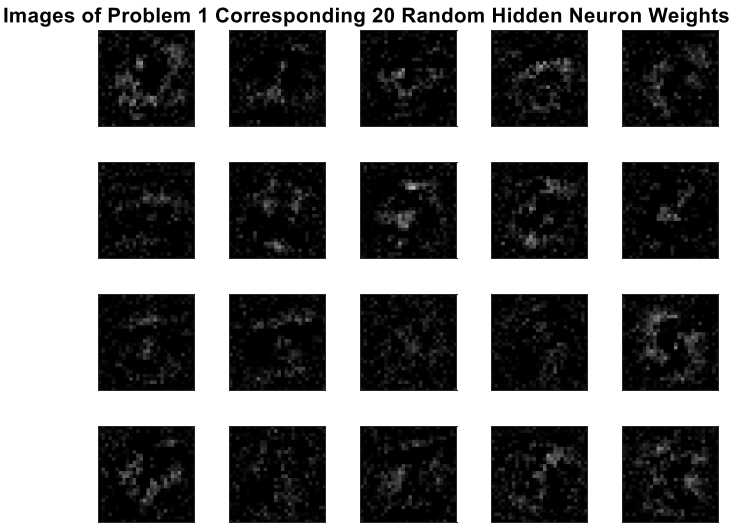

Figure 2.5: Image of features learned by the corresponding 20 random hidden neurons in problem 1

Comparing Figures 2.4 and 2.5 it can be seen that the features learned by the same hidden neurons in both the problems are different. The features learned by a neuron is a combination of varying levels of the intensity value of different pixels and weights of the output layer. It is evident that each hidden neuron

does not learn a specific geometry of a number but is learning various combinations of intensities of a particular digit which makes up the features of an input image.

Sample Outputs:

Figure 2.5 shows the images of 8 random samples from the test set and the corresponding outputs generated using the autoencoder network.

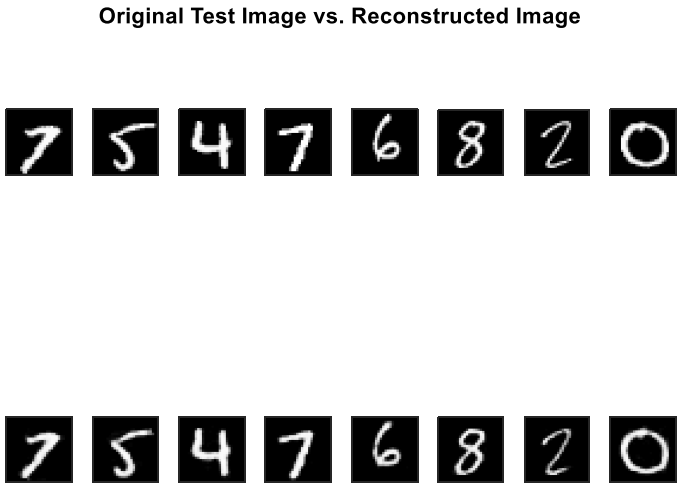

Figure 2.5: Image of 8 random samples from test set and the corresponding outputs generated using the network

Analysis of Results:

Using the results shown in Figure 2.5 one can see that the network was successful in reconstructing the inputs with very minimal changes. The loss function values obtained using both the training and testing sets fell in the desired range of values. In Figure 2.1 the mean of loss function value for the training set was lower than for the testing set, indicating that the network was trained to produced desired results using the data available in the training set. In Figure 2.2 the lower values of loss function for the number 1 indicates the 1 is easier to reconstruct during training and testing. This can be attributed to the non-complex geometry of the number when compared to other digits like 2 or 8 that produced higher loss function values due to its complex geometries of curves and lines. Figure 2.3 shows the effect number of epochs have on the loss function values on the training set. Using the plot, it can be seen that as the number of epochs for training is increased the mean loss function value decreases and starts to saturate.

Appendix:

Problem 1 Code: Language: MATLAB

```
%%********** Homework 3: Multi Layer Feed Forward Neural Network *********%%
% ***** Problem 1: Training Using Back Propogation including Momentum
% ***** Name: Aswin Balasubramaniam ***** %
% ***** M#: 07525504 ***** %
% ***** Course: Intelligent Systems (EECE6036)*****%
% ************************************************************************%%
clear all;
close all;
clf;
clc;
input_layer = 'Input the number of required hidden layers: ';
hidd_layer = input(input_layer);                                % Get user's input for number of
hidden layers Note: The code is written to handle only a single layer and this is a place holder
input_neuron = 'Input the number of hidden neurons to be implemented: ';
h_n = input(input_neuron);                                      % Get user's input for number of
hidden neurons in the layer
o_n = 10;                                                       % Variable for the number of output
neurons

% Reading Data in Text File
img_data = load('MNISTnumImages5000.txt');
img_label = load('MNISTnumLabels5000.txt');

% Randomizing the data points, splitting the data to training and
% testing sets and saving them as txt files
r_num = randperm(size(img_data,1));
r_ord_data = img_data(r_num,:);
r_ord_lab = img_label(r_num,:);
tr_data = r_ord_data(1:4000,:);
tr_lab = r_ord_lab(1:4000,:);
te_data = r_ord_data(4001:5000,:);
te_lab = r_ord_lab(4001:5000,:);
dlmwrite('HW3_Train_Data.txt', tr_data);
dlmwrite('HW3_Test_Data.txt', te_data);
dlmwrite('HW3_Train_Label.txt', tr_lab);
dlmwrite('HW3_Test_Label.txt', te_lab);

% Neural Network Variables Initialization
epo = 100;                                  % Number of epochs
eta_1 = 0.1;                                % Learning rate for input layer
eta_2 = 0.1;                                % Learning rate for output layer
alpha = 0.5;                                % Parameter for momentum
true_train(epo+1,1) = (0);                  % Empty array for training true counts
true_test = 0;                              % Empty array for testing true counts
conf_train(1:10,1:10) = (0);                % Empty array for confusion matrix for training set
conf_test(1:10,1:10) = (0);                 % Empty array for confusion matrix for testing set
w_ip = normrnd(0, sqrt(6/(h_n+length(tr_data(1,:)))), [h_n, length(tr_data(1,:))+1]); % Input
weights being initialized using Xavier initializtion method
w_op = normrnd(0, sqrt(6/(h_n+o_n)), [o_n, h_n+1]); % Output weights being initialized using
Xavier initializtion method
d_lay2(1,1:o_n) = (0);                      % Empty array to store the delta values of layer 2
d_w_op(1:o_n, 1:h_n+1) = (0);               % Empty array to store the delta weight values for
output weights
d_w_ip(1:h_n, 1:length(tr_data(1,:))+1) = (0);  % Empty array to store the delta weight values
for input weights

% Training the Neural Network
for k = 1:epo+1
    for i=1:length(tr_data)                 % Loop that goes over all the training data
        t_label = tr_lab(i);                % Stores the label of the training point
        actual_op(1:o_n) = (0);             % Creates empty matrix to save values of actual
output
        actual_op(t_label+1) = 1;           % Adds 1 (actual neuron output) to the location of
corresponding neuron
```

```
        lay1_neu_sum = (w_ip*([1 tr_data(i,:)]'))';      % layer 1 sum
        lay1_op = 1./(1+exp(-lay1_neu_sum));             % Layer 1 output

        lay2_neu_sum = (w_op*([1 lay1_op]'))';           % Layer 2 sum
        lay2_op = 1./(1+exp(-lay2_neu_sum));             % Layer 2 output

        d_lay2 = (actual_op - lay2_op).*lay2_op.*(1-lay2_op);    % Delta of layer 2: Back
propogation
        d_w_op = eta_2*d_lay2'*([1 lay1_op]) + alpha*d_w_op;     % Delta of layer 2 weights
        w_op = w_op + d_w_op;                                    % Updated weights for layer 2

        d_lay1_sum = w_op(1:o_n,2:h_n+1)'*d_lay2';               % Delta of layer 1: Back
propogation
        d_lay1 = lay1_op.*(1-lay1_op).*d_lay1_sum';
        d_w_ip =  eta_1*d_lay1'*([1 tr_data(i,:)]) + alpha*d_w_ip;  % Delta of layer 1 weights
        w_ip = w_ip + d_w_ip ;                                   % Updated weights for layer 1

        [x,y] = max(lay2_op);                            % Applies threshold to network output to
generate the required output
        lay2_op(y) = 1;
        lay2_op(lay2_op<x) = 0;
        if (isequal(lay2_op, actual_op))                 % Checks if the actual output is the same
as generated output
            true_train(k) = true_train(k) + 1;
        end
        if k==epo+1                                      % Updates the confusion matrix for
trainig at the end of training
            conf_train(find(actual_op == max(actual_op)), find(lay2_op == max(lay2_op))) =
conf_train(find(actual_op == max(actual_op)), find(lay2_op == max(lay2_op))) + 1;
        end
    end
end

% Testing the Neural Network
for test = 1:length(te_data)                             % Loop that goes over all the testing
data
    tst_label = te_lab(test);                            % Stores the label of the testing point
    actual_op(1:o_n) = (0);                              % Creates empty matrix to save values of
actual output
    actual_op(tst_label+1) = 1;                          % Adds 1 (actual neuron output) to the
location of corresponding neuron

    lay1_neu_sum = (w_ip*([1 te_data(test,:)]'))';       % layer 1 sum
    lay1_op = 1./(1+exp(-lay1_neu_sum));                 % Layer 1 output

    lay2_neu_sum = (w_op*([1 lay1_op]'))';               % layer 2 sum
    lay2_op = 1./(1+exp(-lay2_neu_sum));                 % Layer 2 output

    [x,y] = max(lay2_op);                                % Applies threshold to network output to
generate the required output
    lay2_op(y) = 1;
    lay2_op(lay2_op<x) = 0;
    if (isequal(lay2_op, actual_op))
        true_test = true_test + 1;
    end                                                  % Updates the confusion matrix for
testing
    conf_test(find(actual_op == max(actual_op)), find(lay2_op == max(lay2_op))) =
conf_test(find(actual_op == max(actual_op)), find(lay2_op == max(lay2_op))) + 1;
end

% Output Generation
% Calculate the error values during training
hit_rates_train = true_train/length(tr_data);
E_train = 1 - hit_rates_train;
E_plot_x = [0 10:10:epo];
E_train_plot = [E_train(1) (E_train(11:10:epo+1))'];

% Plot the error values of the training data
figure(1);
plot(E_plot_x, E_train_plot, 'bo-');
title('Training Error Values of Neural Network vs. Epochs');
xlabel('Epochs');
```

```
ylabel('Error values (1-hit-rate)');
xticks([0 10:10:epo]);

% Saving neural network weight values
dlmwrite('HW3P1_Trained_IPweights.txt', w_ip);
dlmwrite('HW3P1_Trained_OPweights.txt', w_op);
```

Problem 2 Code: Language: MATLAB

```
%%********* Homework 3: Multi Layer Feed Forward Neural Network *********%%
% ***** Problem 2: Auto Encoder
% ***** Name: Aswin Balasubramaniam ***** %
% ***** M#: 07525504 ***** %
% ***** Course: Intelligent Systems (EECE6036)*****%
% ************************************************************************%%
clear all;
close all;
clf;
clc;
hidd_layer = 1;                          % Variable for number of hidden layers
h_n = 150;                               % Variable for number of hidden neurons in hidden layer
o_n = 784;                               % Variable for number of output neurons

% Reading Randomized Training & Testing Data & Labels Saved in Problem 1
tr_data = load('HW3_Train_Data.txt');
tr_lab = load('HW3_Train_Label.txt');
te_data = load('HW3_Test_Data.txt');
te_lab = load('HW3_Test_Label.txt');

tic                                      % Start timer to record time of program run

% Neural Network Variables Initialization
epo = 100;                               % Number of epochs
eta_1 = 0.01;                            % Learning rate for input layer
eta_2 = 0.01;                            % Learning rate for output layer
alpha = 0.5;                             % Parameter for momentum
J2_train = [];                           % Empty array for J2 loss function training data
J2_test = [];                            % Empty array for J2 loss function training data
J2_epoch_train(1:epo+1) = (0);           % Empty array for J2 loss function training data per
epoch
w_ip = normrnd(0, sqrt(6/(h_n+length(tr_data(1,:)))), [h_n, length(tr_data(1,:))+1]); % Input
weights being initialized using Xavier initializtion method
w_op = normrnd(0, sqrt(6/(h_n+o_n)), [o_n, h_n+1]); % Output weights being initialized using
Xavier initializtion method
d_lay2(1,1:o_n) = (0);                   % Empty array to store the delta values of layer 2
d_w_op(1:o_n, 1:h_n+1) = (0);            % Empty array to store the delta weight values for output
weights
d_w_ip(1:h_n, 1:length(tr_data(1,:))+1) = (0); % Empty array to store the delta weight values for
input weights

% Training the Neural Network
for k = 1:epo+1
    for i=1:length(tr_data)               % Loop that goes over all the training data
        t_label = tr_lab(i);              % Stores the label of the training point
        actual_op = tr_data(i,:);         % Stores the input to network to be compared with output
of network

        lay1_neu_sum = (w_ip*([1 tr_data(i,:)]'))';      % layer 1 sum
        lay1_op = 1./(1+exp(-lay1_neu_sum));             % Layer 1 output

        lay2_neu_sum = (w_op*([1 lay1_op]'))';           % Layer 2 sum
        lay2_op = 1./(1+exp(-lay2_neu_sum));             % Layer 2 output

        d_lay2 = (actual_op - lay2_op).*lay2_op.*(1-lay2_op);    % Delta of layer 2: Back
propogation
        d_w_op = eta_2*d_lay2'*([1 lay1_op]) + alpha*d_w_op;     % Delta of layer 2 weights
        w_op = w_op + d_w_op;                                    % Updated weights for layer 2

        d_lay1_sum = w_op(1:o_n,2:h_n+1)'*d_lay2';               % Delta of layer 1: Back
propogation
```

```
        d_lay1 = lay1_op.*(1-lay1_op).*d_lay1_sum';
        d_w_ip =  eta_1*d_lay1'*([1 tr_data(i,:)]) + alpha*d_w_ip;  % Delta of layer 1 weights
        w_ip = w_ip + d_w_ip ;                                      % Updated weights for layer 1

        error_mat_e = (actual_op-lay2_op).^2;                       % Calculating the J2 loss
function for the input being analyzed every epoch
        error_e = 0.5*sum(error_mat_e);
        J2_epoch_train(k) = J2_epoch_train(k) + error_e;
        % Calculating the J2 loss function at the end of epoch for evey
        % number seperately
        if k == epo
            error_mat = (actual_op-lay2_op).^2;
            error = 0.5*sum(error_mat);
            J2_train(t_label+1,end+1) = error;
        end
    end
end

% Testing the Neural Network
for test = 1:length(te_data)                                        % Loop that goes over all the
testing data
    tst_label = te_lab(test);                                       % Stores the label of the testing
point
    actual_op = te_data(test,:);                                    % Stores the input to network to
be compared with output of network

    lay1_neu_sum = (w_ip*([1 te_data(test,:)]'))';                  % layer 1 sum
    lay1_op = 1./(1+exp(-lay1_neu_sum));                            % Layer 1 output

    lay2_neu_sum = (w_op*([1 lay1_op]'))';                          % layer 2 sum
    lay2_op = 1./(1+exp(-lay2_neu_sum));                            % Layer 2 output

    error_mat = (actual_op-lay2_op).^2;                             % Calculating the J2 loss
function for the input being analyzed every epoch
    error = 0.5*sum(error_mat);
    J2_test(tst_label+1,end+1) = error;
end

% Output Generation
% Calculate the mean J2 loss function values for training, testing and
% individuual numbers in training and testing data at the final epoch
train_num = sum(J2_train~=0,2);
train_sum = sum(J2_train,2);
mean_J2train_num = train_sum./train_num;
mean_J2train_all = sum(train_sum)/sum(train_num);

test_num = sum(J2_test~=0,2);
test_sum = sum(J2_test,2);
mean_J2test_num = test_sum./test_num;
mean_J2test_all = sum(test_sum)/sum(test_num);

% Plot the J2 loss function values of the neural network every 10th epoch
figure(1)
Graph1_labels = categorical({'Mean Loss on Training set','Mean Loss on Test set'});
Loss_mean = [mean_J2train_all; mean_J2test_all];
bar(Graph1_labels, Loss_mean);
title('Mean Loss on Training and Test Set');
ylabel('J2 Mean Loss Values');

% Plot the mean J2 loss function values of the neural network for indiviual
% numbers in training and testing data set
figure(2)
Graph2_labels = categorical({'Mean Loss 0','Mean Loss 1','Mean Loss 2','Mean Loss 3','Mean Loss
4','Mean Loss 5','Mean Loss 6','Mean Loss 7','Mean Loss 8','Mean Loss 9'});
Loss_num_mean = [mean_J2train_num(1) mean_J2test_num(1); mean_J2train_num(2) mean_J2test_num(2);
mean_J2train_num(3) mean_J2test_num(3); mean_J2train_num(4) mean_J2test_num(4);
mean_J2train_num(5) mean_J2test_num(5); mean_J2train_num(6) mean_J2test_num(6);
mean_J2train_num(7) mean_J2test_num(7); mean_J2train_num(8) mean_J2test_num(8);
mean_J2train_num(9) mean_J2test_num(9); mean_J2train_num(10) mean_J2test_num(10)];
bar(Graph2_labels, Loss_num_mean);
title('Mean Loss on Training and Test Set for each number');
```

```
ylabel('J2 Mean Loss Values');
legend ('Training set','Test set');

% Plot the mean J2 loss function values of the neural network in training and testing data set
figure(3)
J2train_plot_x = [0 10:10:epo];
J2_train_plot = [J2_epoch_train(1)/4000 (J2_epoch_train(11:10:epo+1)/4000)];
plot(J2train_plot_x, J2_train_plot, 'bo-');
title('Mean Loss on Training Set vs. Epochs');
xlabel('Epochs');
ylabel('Mean J2 values');
xticks([0 10:10:epo]);

% Images of 20 random hidden neuron feature using their weights
% Code reference: Dr. Ali Minai's code showMNISTnum.m
figure(4)
r_hidd_neu = randperm(150,20);
dlmwrite('HW3P2_20RandomHiddenNeuronNum.txt', r_hidd_neu);
U = w_ip(r_hidd_neu, 2:785);
for i=1:4
    for j = 1:5
        v = reshape(U((i-1)*5+j,:),28,28);
        subplot(4,5,(i-1)*5+j)
        image(64*v)
        colormap(gray(64));
        set(gca,'xtick',[])
        set(gca,'xticklabel',[])
        set(gca,'ytick',[])
        set(gca,'yticklabel',[])
        set(gca,'dataaspectratio',[1 1 1]);
    end
end
a1 = axes;
t1 = title('Images of Problem 2 20 Random Hidden Neuron Weights');
a1.Visible = 'off';
t1.Visible = 'on';

% Images of corresponding 20 random hidden neurons features in P1 using their weights
% Code reference: Dr. Ali Minai's code showMNISTnum.m
figure(5)
P1_w_ip = load('HW3P1_Trained_IPweights.txt');
U = P1_w_ip(r_hidd_neu, 2:785);
for i=1:4
    for j = 1:5
        v = reshape(U((i-1)*5+j,:),28,28);
        subplot(4,5,(i-1)*5+j)
        image(64*v)
        colormap(gray(64));
        set(gca,'xtick',[])
        set(gca,'xticklabel',[])
        set(gca,'ytick',[])
        set(gca,'yticklabel',[])
        set(gca,'dataaspectratio',[1 1 1]);
    end
end
a2 = axes;
t2 = title('Images of Problem 1 Corresponding 20 Random Hidden Neuron Weights from P2');
a2.Visible = 'off';
t2.Visible = 'on';

% Images of random original 8 test data used and data recreated using the network
% Code reference: Dr. Ali Minai's code showMNISTnum.m
figure(6)
ntk_img = [];
r_test_data = randperm(1000,8);
org_img = te_data(r_test_data, :);
for t = 1:length(r_test_data)
    lay1_neu_sum = (w_ip*([1 org_img(t,:)]'))';
    lay1_op = 1./(1+exp(-lay1_neu_sum));
```

```
    lay2_neu_sum = (w_op*([1 lay1_op]'))';
    lay2_op = 1./(1+exp(-lay2_neu_sum));
    ntk_img(t, :) = lay2_op;
end
for i=1:2
    if i==1
        U = org_img;
    else
        U = ntk_img;
    end
    for j = 1:8
        v = reshape(U(j,:),28,28);
        subplot(2,8,(i-1)*8+j)
        image(64*v)
        colormap(gray(64));
        set(gca,'xtick',[])
        set(gca,'xticklabel',[])
        set(gca,'ytick',[])
        set(gca,'yticklabel',[])
        set(gca,'dataaspectratio',[1 1 1]);
    end
end
a3 = axes;
t3 = title('Original Test Image vs. Reconstructed Image');
a3.Visible = 'off';
t3.Visible = 'on';

% Saving neural network weight values
dlmwrite('HW3P2_Trained_IPweights.txt', w_ip);
dlmwrite('HW3P2_Trained_OPweights.txt', w_op);

toc                                     % Start timer to record time of program run
```