EEE443 Neural Networks

Homework 2 Report

Name: Oğuz Altan

ID: 21600966

Question 1)

The cost function is given as:

$$C_1 = \frac{1}{2}\sum_n (y^n - w^T x^n)^2$$

a) The derivations are needed for minimization of both cost function C_1 and C_2. To simplify the notations, vectorized notation is to be used, where y^n corresponds to Y and x^n to X.

Cost function C_1 can be written as:

$$C_1 = \frac{1}{2}(Y - w^T X)(Y - w^T X)^T$$

$$= \frac{1}{2}(YY^T - YX^T w - w^T XY^T - w^T XX^T w)$$

The terms $YX^T w$ and $w^T XY^T$ are both scalar and therefore equal to each other. Thus, we can rewrite our cost function C_1 as:

$$C_1 = \frac{1}{2}(YY^T - 2YX^T w + w^T XX^T w)$$

To find the minimum of C_1, we take the derivative of C_1 and equate it to 0, for some w. This can be mathematically written as:

$$\underset{w}{\operatorname{argmin}}\, C_1 = \frac{dC_1}{dw} = 0$$

As we take the derivative with respect to w, we can omit the term YY^T as it is constant and independent of the variable w. Therefore,

$$\underset{w}{\operatorname{argmin}}\, C_1 = -YX^T w + \frac{1}{2} w^T XX^T$$

We are given another cost function:

$$C_2 = \frac{1}{2} w^T A w - b^T w$$

Minimizing C2:

$$\underset{w}{\operatorname{argmin}}\, C_2 = \frac{1}{2} w^T A w - b^T w$$

Equating these two cost function equations, we get:

$$A = XX^T \; and \; b = XY^T$$

As we have found A and b, we can write derivatives of two cost functions C_1 and C_2 as:

$$\frac{dC_1}{dw} = \frac{1}{2}(XX^T + X^T X)w - XY^T = \Delta \mathrm{w}$$

$$\frac{dC_2}{dw} = \frac{1}{2}(A + A^T)w - b = \Delta \mathrm{w}$$

b) As we know $\Delta w^{\sim} = \Delta w$

$$\frac{1}{2}(A + A^T)w^{\sim} = \frac{1}{2}(A + A^T)w - b$$

$$2\ \frac{1}{2}(A + A^T)^{-1}(A + A^T)w^{\sim} = 2\ \ (A + A^T)^{-1}(\frac{1}{2}(A + A^T)w - b)$$

$$w^{\sim} = w - 2\ \ (A + A^T)^{-1}b$$

Using that change of variable, then we can write C_3 as:

$$C_3 = \frac{1}{2}\Delta w^{\sim \mathrm{T}} A w^{\sim}$$

$$C_3 = \frac{1}{2}(w - 2(A + A^T)^{-1}b)^T A(w - 2(A + A^T)^{-1}b)$$

As C_3 is a loss function and we want to minimize it, we take its derivate with respect to w:

$$\frac{dC_3}{dw} = \frac{1}{2}(A(w - 2(A + A^T)^{-1}b) + A^T(w - 2(A + A^T)^{-1}b))$$

$$= \frac{1}{2}\big((A + A^T)(w - 2(A + A^T)^{-1}b)\big) = \frac{1}{2}(A + A^T)w - b$$

which is same update rule as what we have found for part a.

c) We have update rule of C_2 and we assume that A is symmetric and positive-definite matrix, meaning that eigenvalues of A is real and positive-definite. Therefore, we have:

$$\frac{dC_2}{dw} = Aw - b$$

For the update rule, we can state:

$$w(t + 1) = w(t) + \ \Delta w(t + 1)\ = w(t) - \frac{dC_2}{dw(t)}\eta$$

$$= w(t) - \big(-b + Aw(t)\big)\eta = (I - \ \eta A)w(t) + \eta b$$

As the weight vector A can be rewritten in term of eigenvectors of $w(t)$ using eigenvalue decomposition, we can rewrite the update rule as:

$$w(t + 1) = (I - \eta A)\left(\sum_k a_k^t\, v_k\right) + \eta b = \sum_k a_k^t\,(1 - \eta\lambda_k)v_k + \eta b$$

The condition for stability for such a system having propery of causalty and linearly independent eigenvectors:

$$1 > (1 - \eta\lambda_k) > -1$$

Considering this stability condition, tha maximum learning rate η_{max} can be obained as follows:

$$\eta_{max} = \frac{2}{max_k \lambda_k}$$

Question 2

a and b)

The results for a single hidden layer neural network architecture are given below. This architecture has 50 hidden layer neurons and works for 300 epoch. The backpropagation algorithm is stochastic gradient descent on mini-batches. Various weights initializations, which are taken from Gaussian distribution, multiplied by already calculated standart deviations, are tested. These standart deviations are inverse squares of number of inputs of a neuron in that corresponding layer. Then, I chose an appropriate set of network parameters and made simulations on MATLAB. The error metrics which are used throughout this experiment are mean square error (MSE) and mean classification error (MCE), that shows the percentage of correctly classified images.

First of all, for all of the figures and plots, the zigzag pattern can be explained by the stochastic gradient descent algorithm on mini-batches. As the parameters updates are applied at the end of the process of every mini-batch, using the mean of gradients, the corresponding curves are not very smooth, because we do not update every parameters and plot the error metrics for every data sample in the dataset, rather we do them after mini-batches of data samples from the dataset.

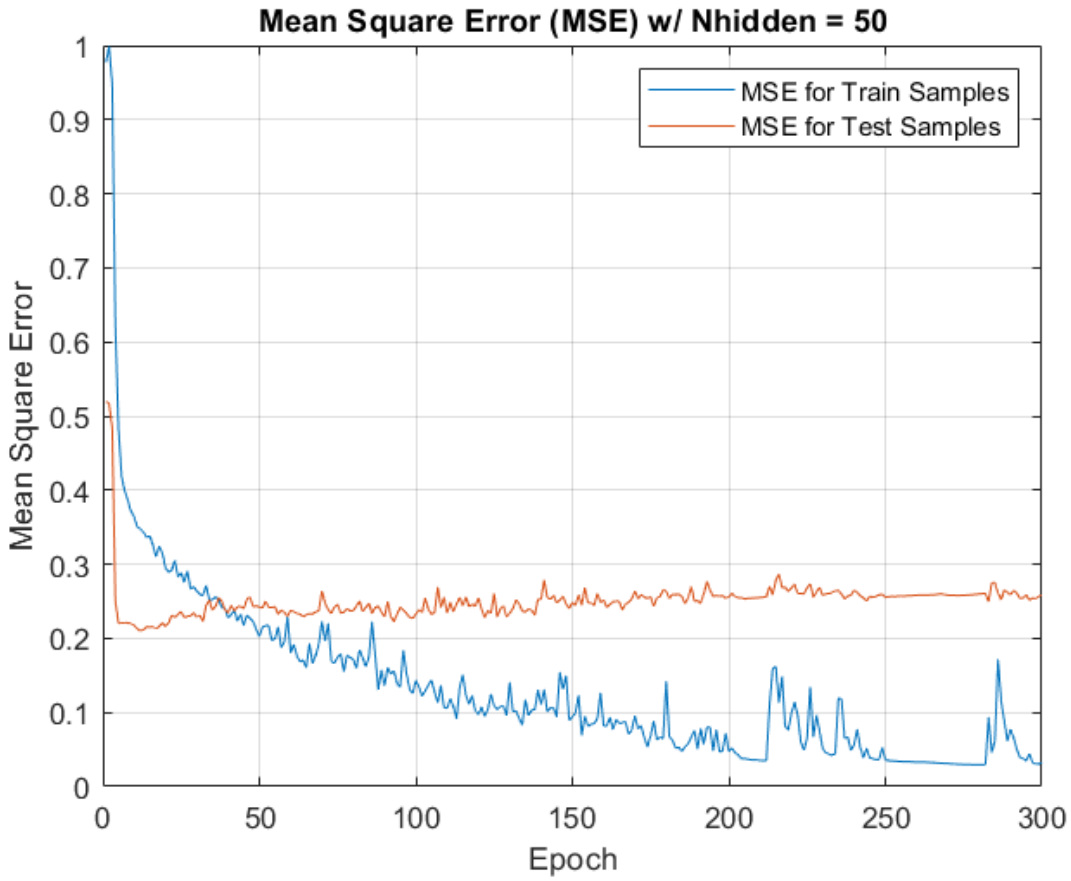

Figure 1: Mean Square Error Plots for 50 Hidden Neurons Architecture

As it can be seen from the figure, the mean square errors converge to 0 for 300 epochs, which shows that the network learns how to perform binary classification on cat versus car problem, using feedforward and backpropagation algortihms.

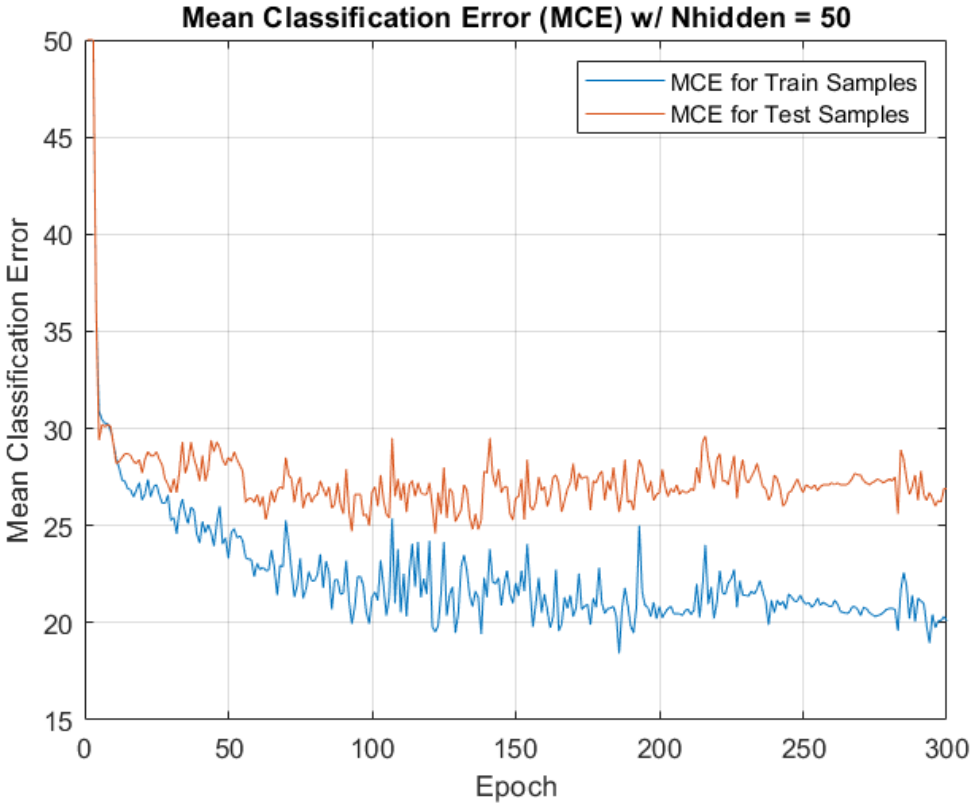

Figure 2: Mean Classification Error Plots for 50 Hidden Neurons Architecture

From the figure above, it can be observed that mean classification error increases looking at the overall plot. This error metric does not increase for every iteration, because we use stochastic gradient descent on mini-batches. However, looking at the overall shape, it can be stated that the number of correctly estimated ground truth label increases. At the end of 300th epoch, the percentage of correct predictions is between 78-79%.

For both error metrics, looking at train vs testing sets, we see that performance of the network is better on training sets. The reason for that is, as the network uses the same train set to optimize its parameters, or learns, when it uses the same train set to check its performance, it can predict the correct output better than predicting the output of the test set, the set of data that is totally unknown and new to the network.

The mean square error takes the square of prediction error, which means that for every bad prediction, squaring the error increases the negative effect of the error and decreases the performance of the network. For our case, as we use hyperbolic tangent function, our errors are between -1 and 1. When we take the square of a number between -1 and 1, the error gets much smaller. This may result in underestimating the network's ineffectiveness, as although the error is much bigger, when we use MSE, we consider the smaller version of the actual prediction error. Using these informations about MSE and observing the MSE plots of our network, it can be commented that MSE may not be an adequate predictor of classification error.

c)

In this part, we train three different neural networks with different number of single hidden layer neurons. In the network in part a and b, we use 50 hidden layer neurons. For the other two networks that we train to compare with our first network, we use 200 and 10 hidden layer neurons. The corresponding mean square error plots are shown in the Figure 3, above.

The first thing to observe here is the shape of mean square error curves. These curves are not very smooth but rather has zigzag shape. Looking at the overall curves, it can be observed that MSE for all three neural networks and for both train and test data sets, decreases and converges to zero. Again, the performance of the networks are better on train sets than test sets, for the same reasons explained above.

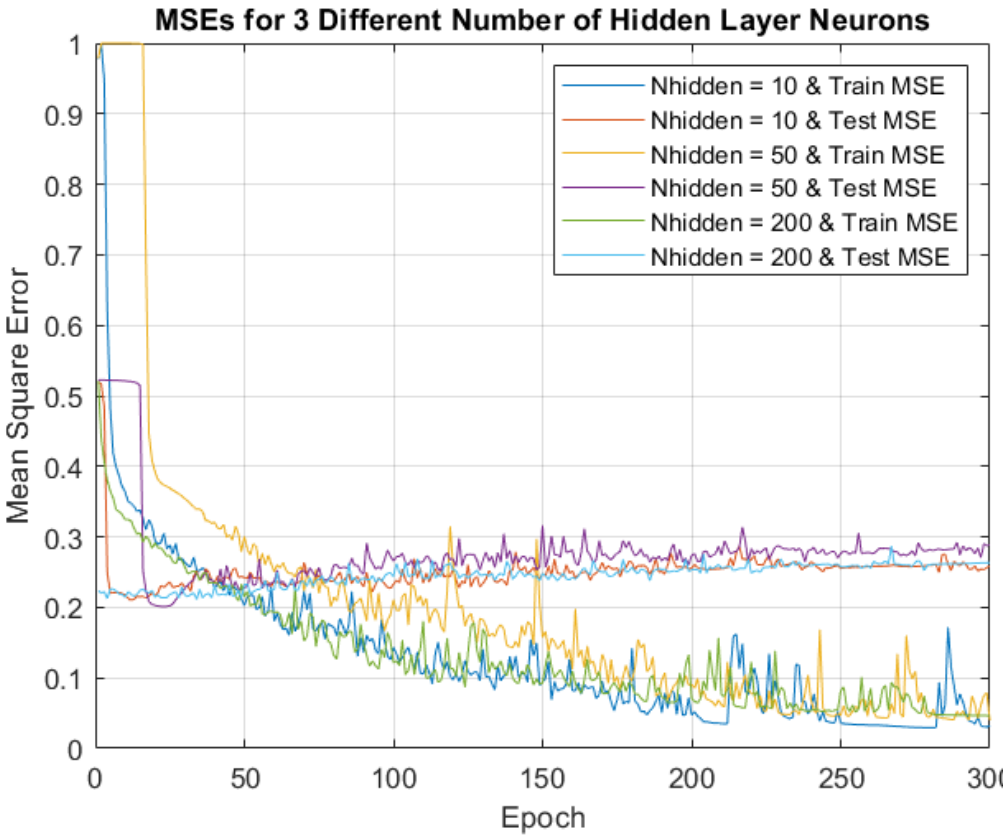

Figure 3: Mean Square Error (MSE) Plots for Three Different Single Hidden Layer Architecture

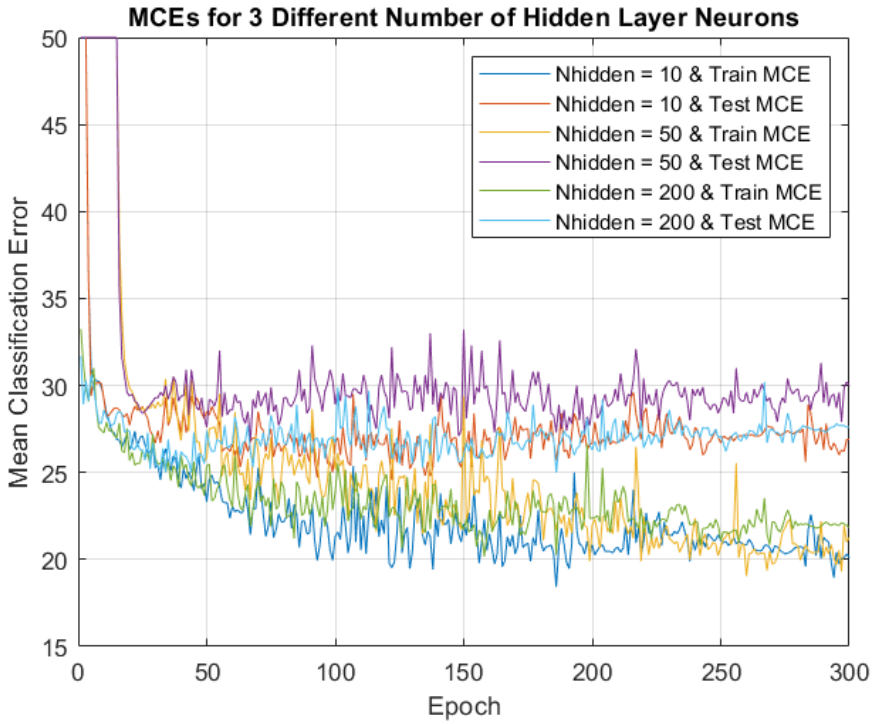

Figure 4: Mean Classification Error (MCE) Plots for Three Different Single Hidden Layer Architecture

d) In this part, we use a neural network with two hidden layers.

Figure 5: Mean Square Error (MSE) Plots of Two Hidden Layers Architecture

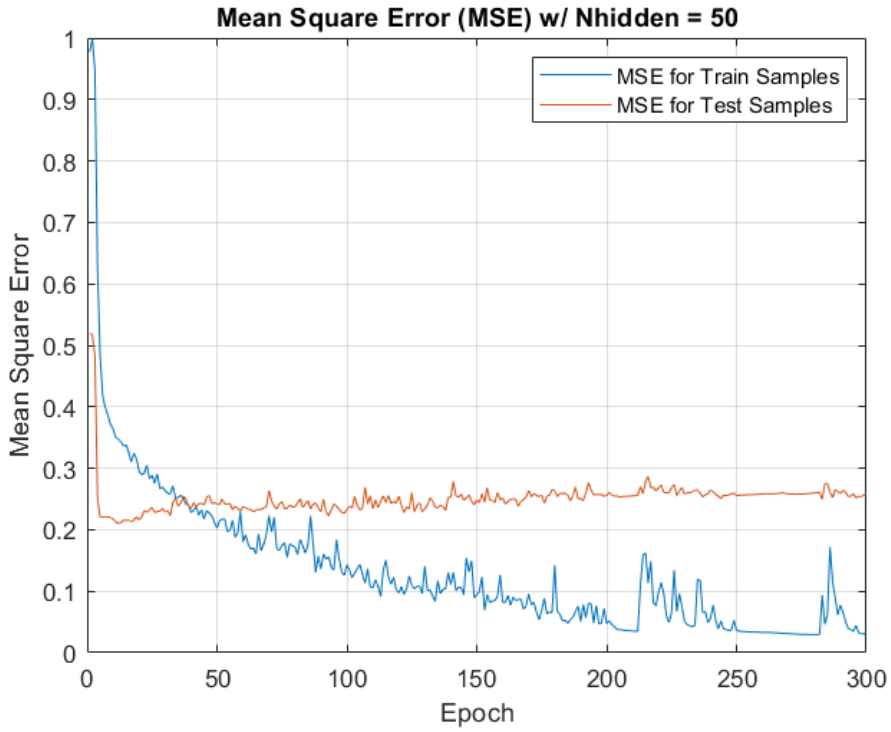

Figure 6: Mean Square Error (MSE) Plots for 50 Hidden Neurons Architecture

To compare both networks easily, the MSE and MCE plots of the single hidden layer network, implemented in part a is pasted under the MSE and MCE plots of two hiden layer network. Observing both plots above, Figure 5 and 6, it can be seen that MSE curve for two hidden layers network converges to 0 much faster than single hidden layer network, with a more smooth curve.

Figure 7: Mean Classification Error (MCE) Plots of Two Hidden Layers Architecture

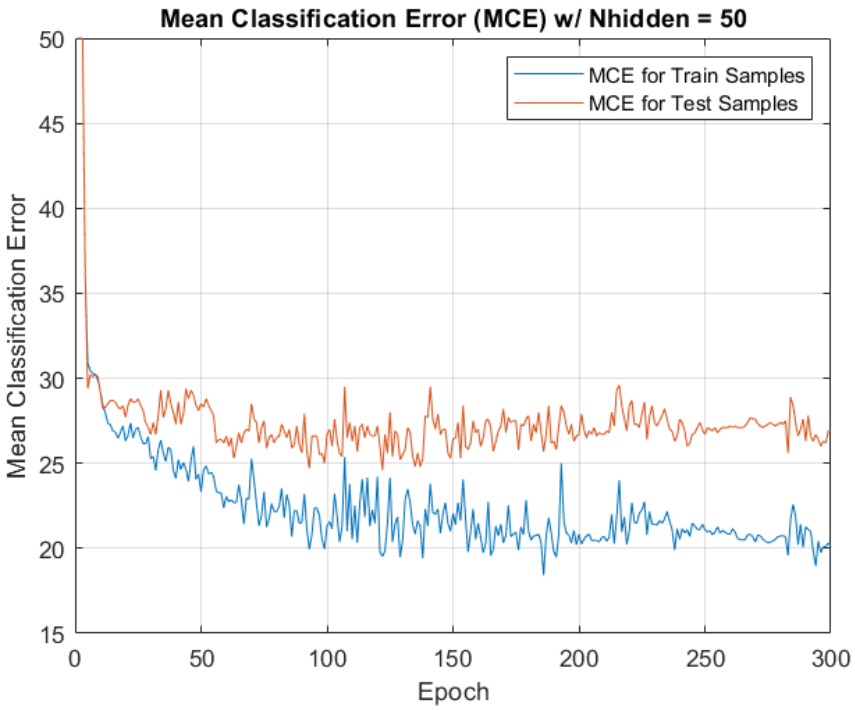

Figure 8: Mean Classification Error (MCE) Plots for 50 Hidden Neurons Architecture

Observing, this time, MCE plots above, Figure 7 and 8, it can be seen that classification performance of the two hidden layer network is better than of the single hidden layer network, althought the difference is not major but small. After 300 epochs, the percentage of correctly classifying the data is very close to 80% for two hidden layers network.

d) This time, we use momentum method on the same two hidden layers neural network implemented in part d.

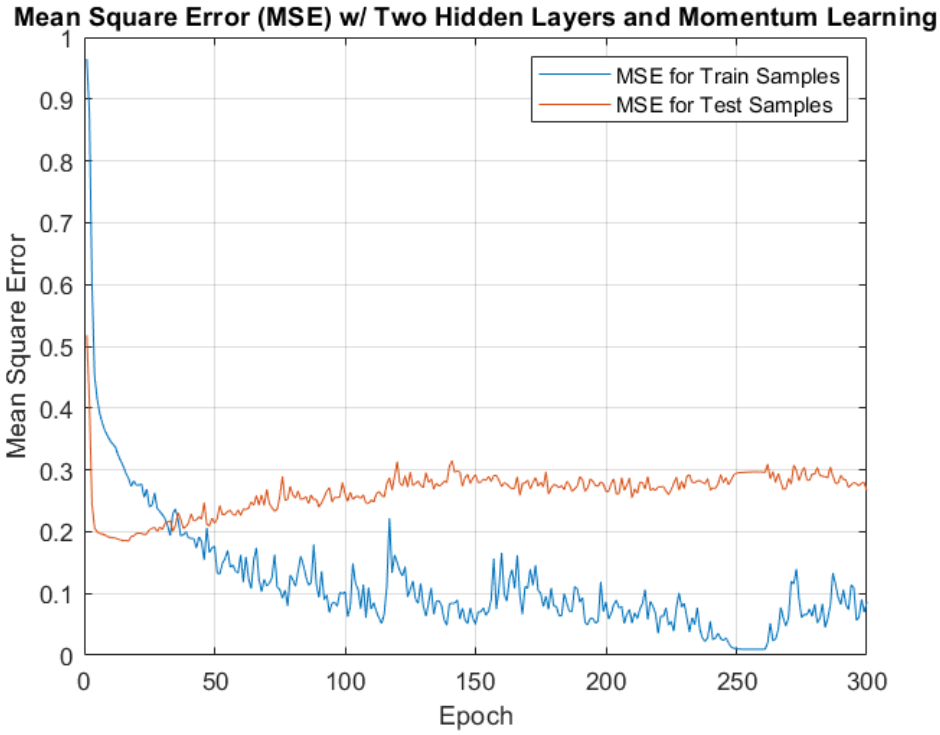

Figure 9: Mean Square Error (MSE) Plots for Two Hidden Layers Network with Momentum Method

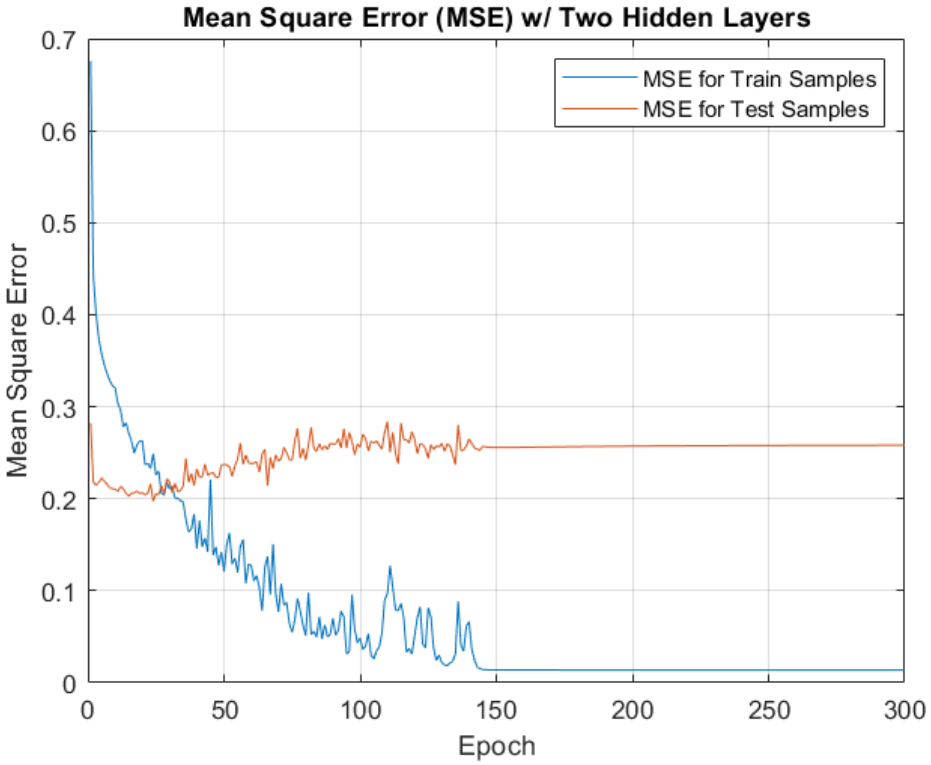

Figure 10: Mean Square Error (MSE) Plots for Two Hidden Layers Network

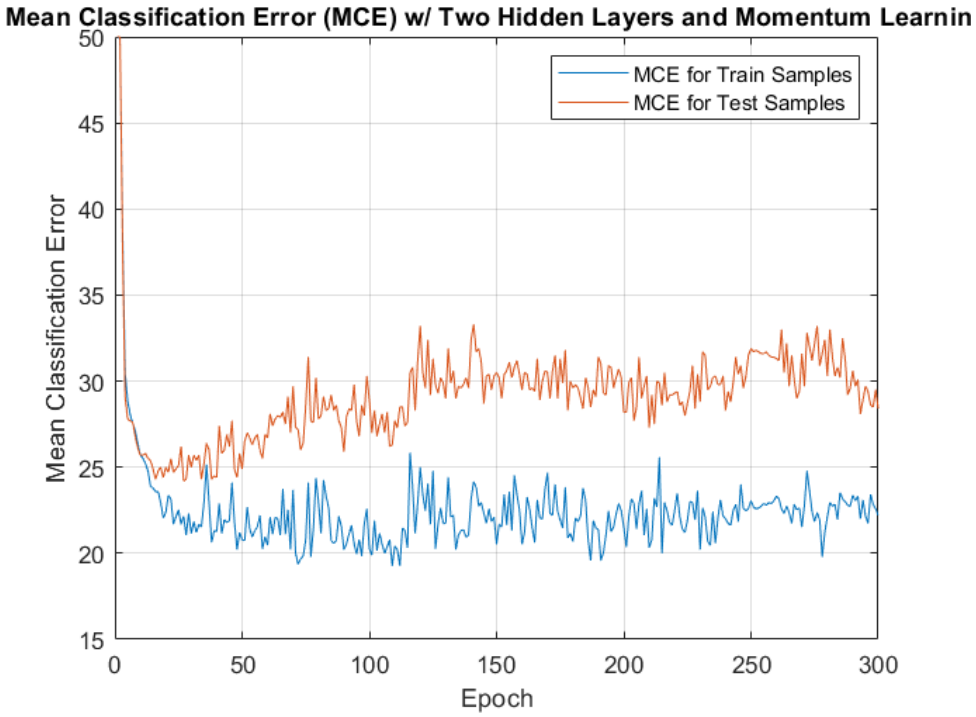

Figure 11: Mean Classification Error (MCE) Plots for Two Hidden Layers Network with Momentum

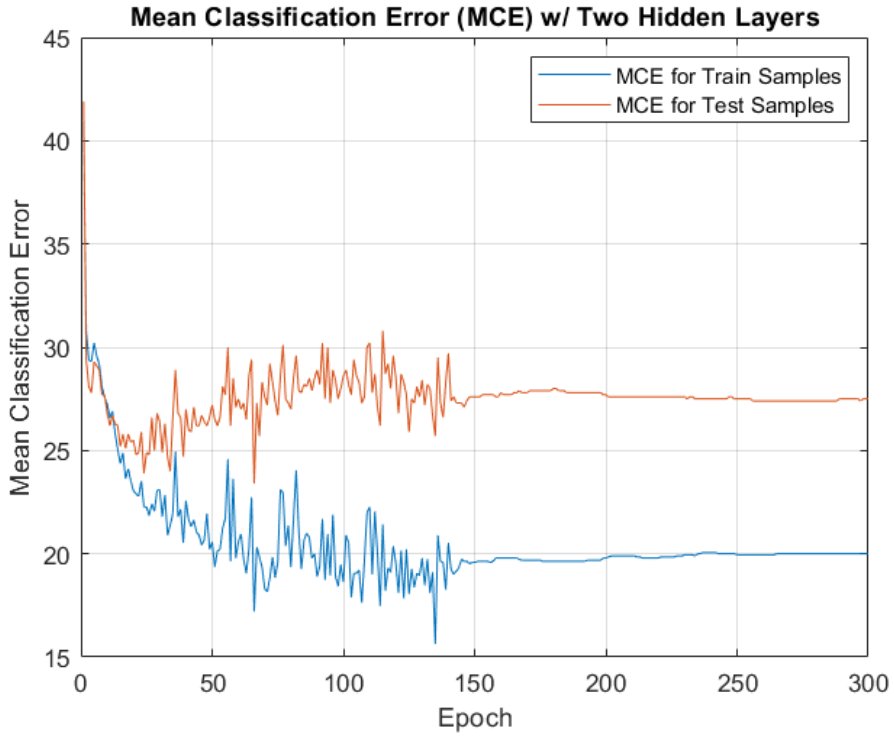

Figure 12: Mean Classification Error (MCE) Plots for Two Hidden Layers Network

Learning with momentum is used to smooth out the curves without slowing down the learning to much. We use moving average of the individual weight changes corresponding to the single data samples. Therefore, using this method with stochasting gradient descent algorithm on mini-batches smooths out error metrics plots.

The MSE and MCE plots of the network in part d and e are put top and bottom for the sake of easiness of the comparisons. Looking a the figures 9 and 10, it can be commented that the MSE curve for the network with momentum method is smoother but converges to 0 slower than of the network without momentum method, as that method is actually a trade-of between the smoothness and the learning speed.

3)

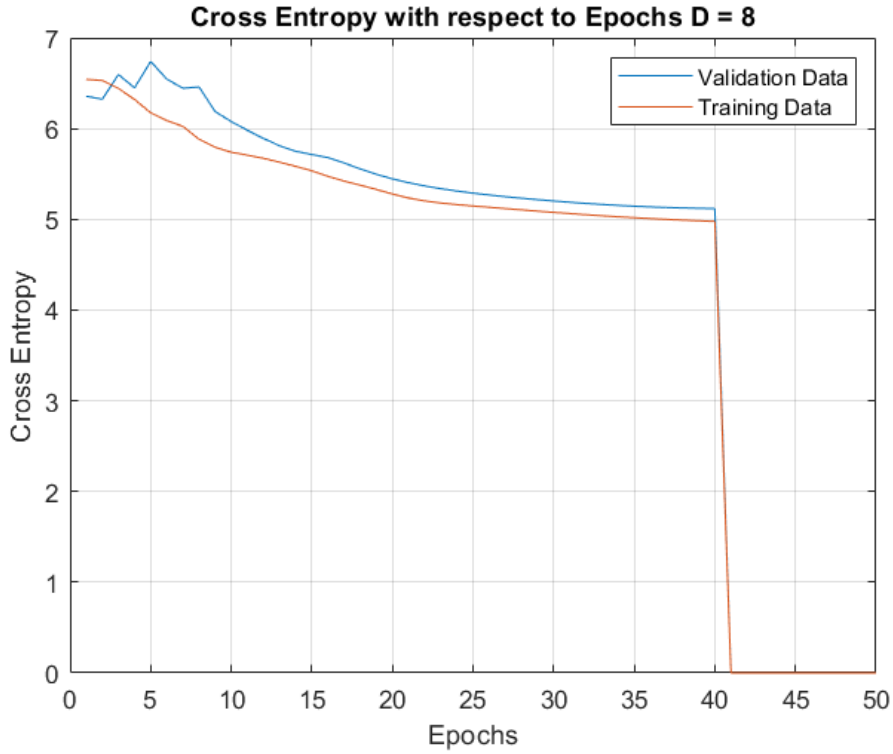

Figure 13: Cross Entropy with respect to Epoch for pair (D,P) = 8,64

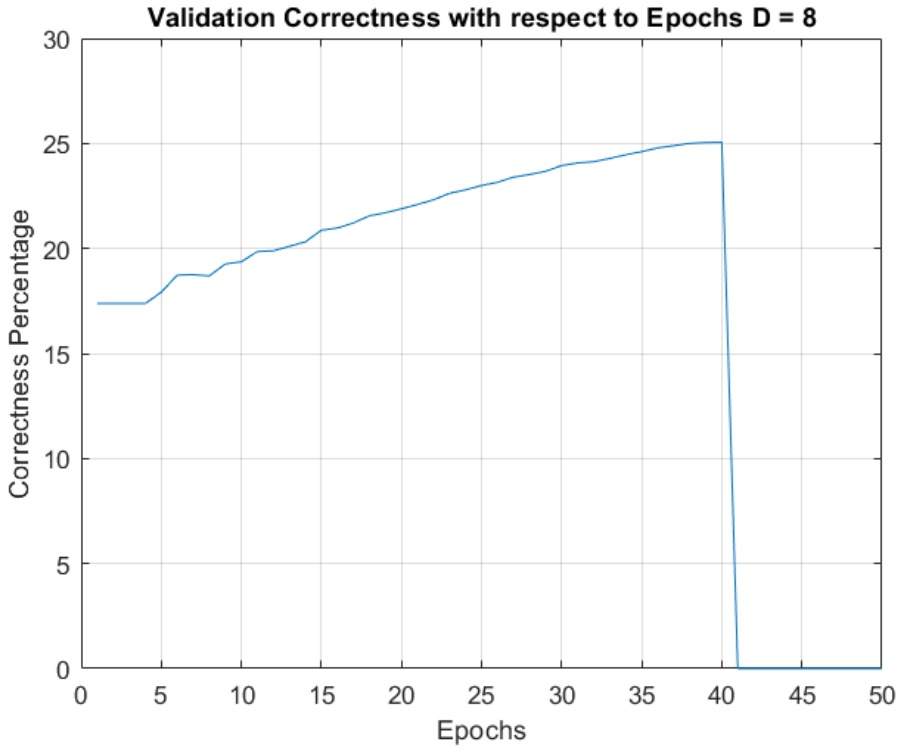

Figure 14: Validation Correctness with respect to Epoch for pair (D,P) = 8,64

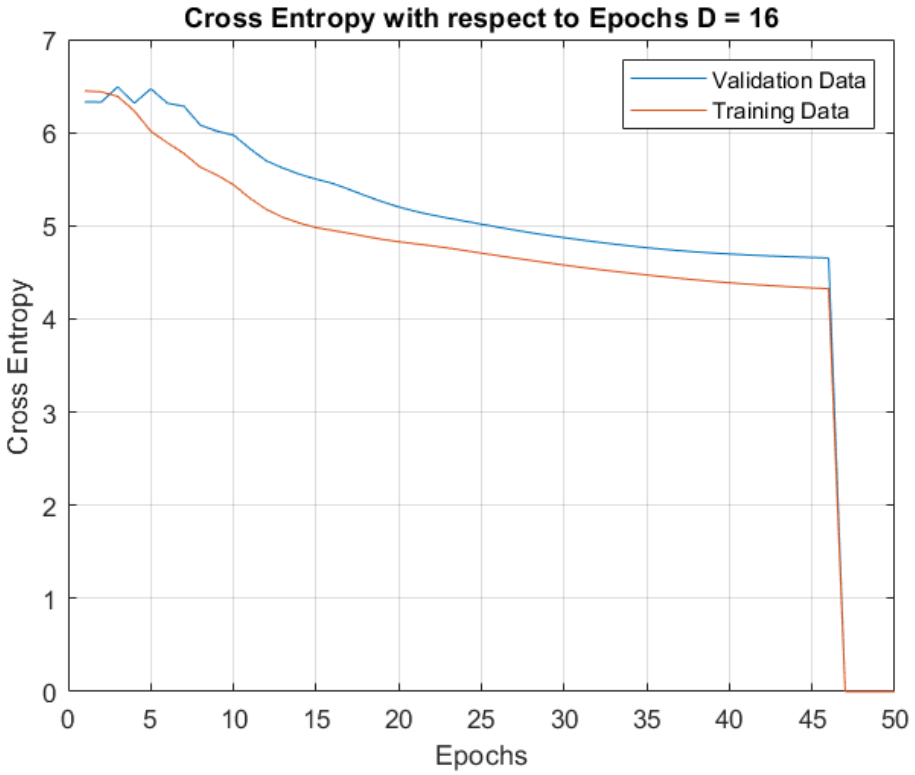

Figure 15: Cross Entropy with respect to Epoch for pair (D,P) = 16,128

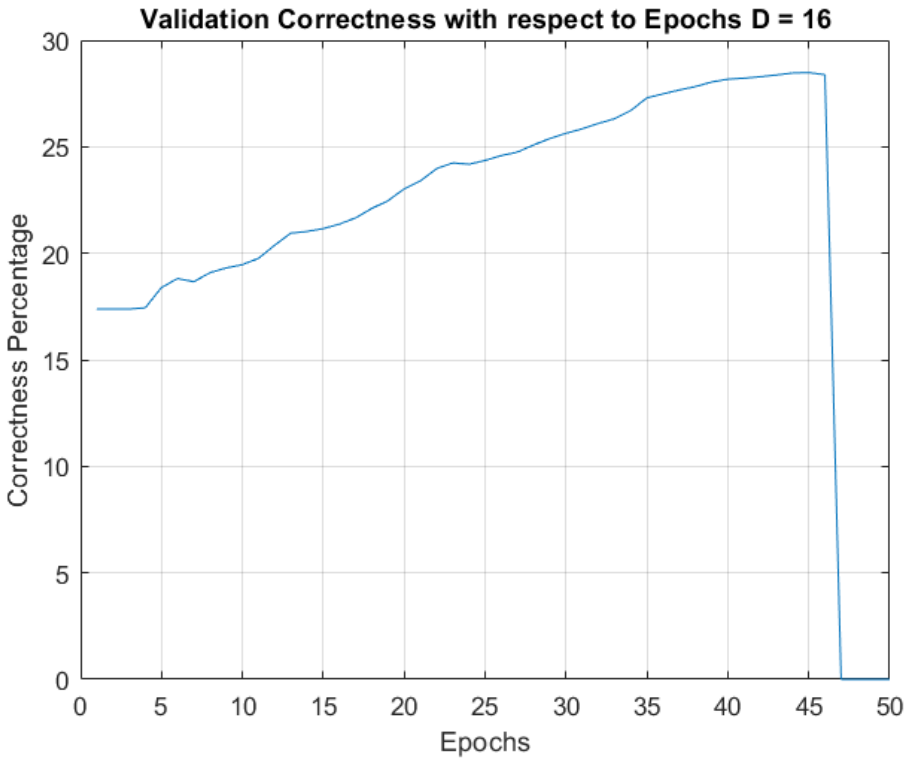

Figure 16: Validation Correctness with respect to Epoch for pair (D,P) = 16,128

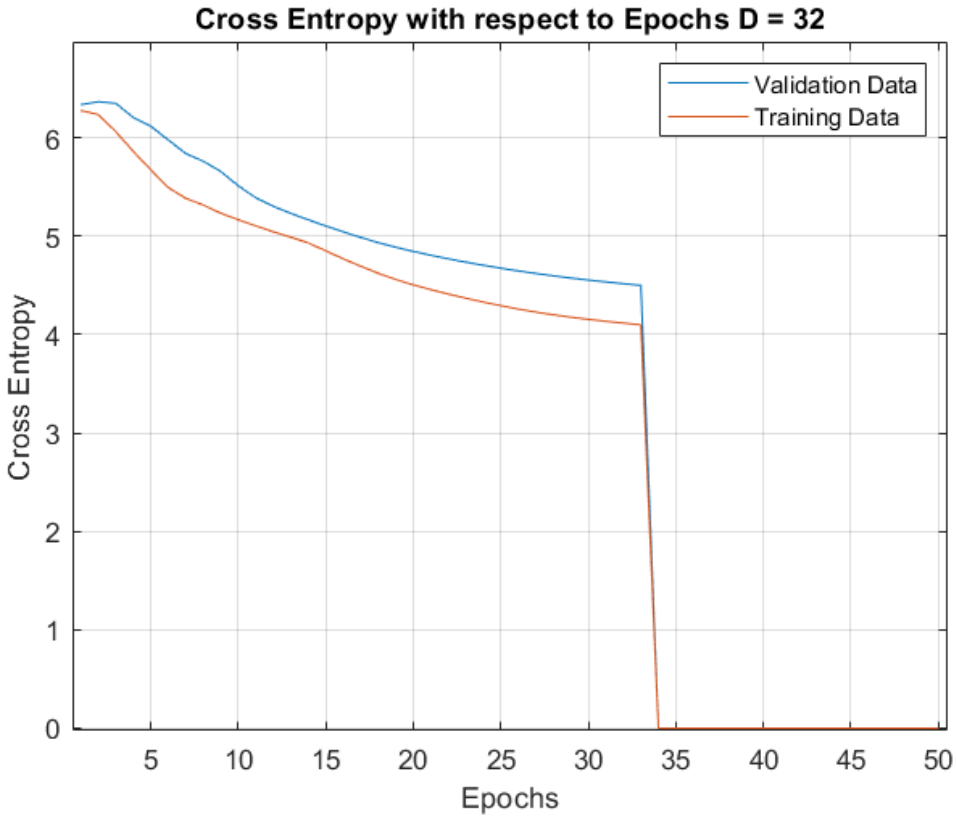

Figure 17: Cross Entropy with respect to Epoch for pair (D,P) = 32,256

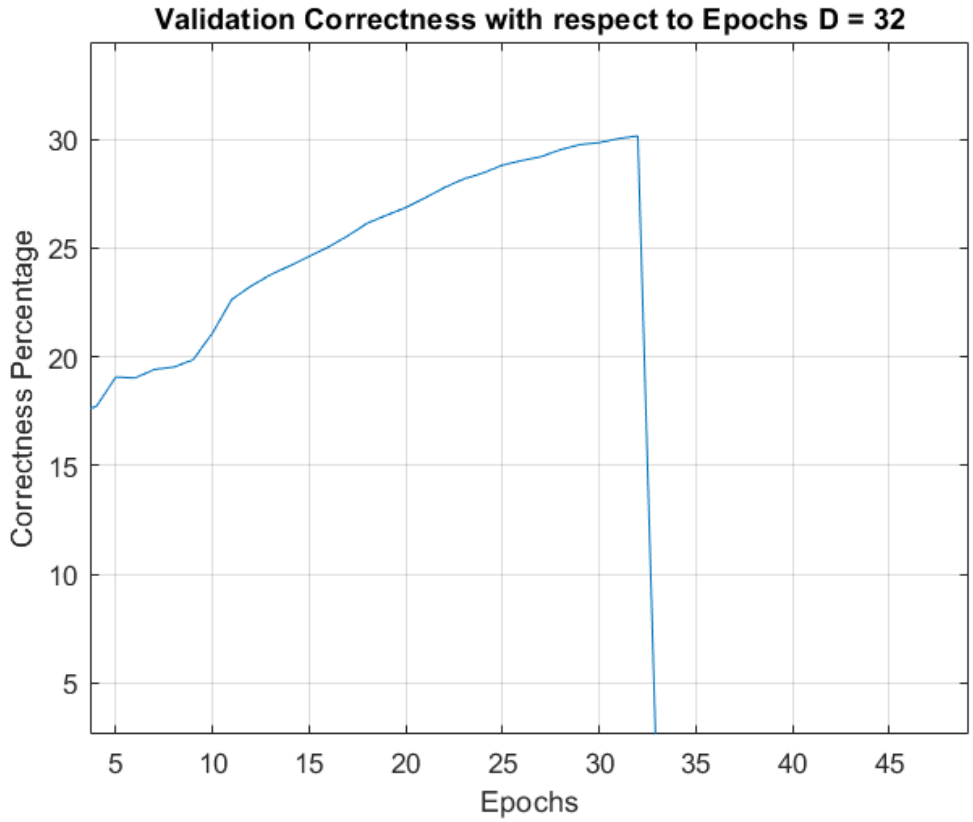

Figure 18: Validation Correctness with respect to Epoch for pair (D,P) = 32,256

In this question, we train a model of neural network for natural language processing. In our case, we analyze sequence of words. Given a trigram, network predicts the fourth word that can come after trigram, namely three words. Among the possible encoding techniques, one hot encoding is preferred in our model. Given a vocabulray of size 250 words, we map word indexes to a vector of size 250 and hot encoding results for input, output and index is computed in MATLAB. These one hot vectors forms embedding matrix and this matrix is used in the network. In the first part, we try the network for 3 different hidden layer neuron numbers compare how three different models converge. Softmax activation function is used to compute probability of output classes that add up to 1, where the class with highest probability is the most probable word that may come after trigram. After feedforward, we apply backpropagation algorithm to robust the network and increase the success of the network. Among the possible loss functions, cross entropy is preferred in this network, as this loss function looks at the difference of probabilistic distributions. The way that the network is tested is using cross validation and cross validation is used to compute the error of the predictions of the network.

Due to early stop, namely algorithm stops based on the cross-entropy error on the validation data, network with pair (D,P) = (32,256) stops at 33th epoch, (D,P) = (16,128) at 46th epoch and (D,P) = (8,64) at 40th epoch.

a) From the validation correctness and cross entropy plots of the three neural networks with (D,P) pairs (8,64), (16,128), (32,256) respectively, it can be seen that the higher the number of hidden layer neurons for such structure, the more succesful the network in prediction and learing but also more prone to over-fitting. Observing the three validation correctness plots, it can be stated that the neural netwok with parameters (D,P) = (32,256) is the most successful one as correctness percentage is higher than other two and reaches stability before other two.

b) For this part, I pick some sample trigrams from the test data and generate predictions for the fourth word via the trained neural network. I also list the top 10 candidates for the fourth word for each of 5 sample trigrams. The table for 5 sample trigrams and related candidate fourth words, in a fashion that the word with higher probability has lower index than the one with lower probability, namely the most probable word has index 1 and the most irrelevant among them has index 10, can be found below:

Trigram	1	2	3	4	5	6	7	8	9	10
not for long	few	year	few	.	it	he	what	know	have	and
Going to get	we	.	i	they	you	it	to	?	is	,
with another way	.	.	he	no	she	is	i	we	that	you
did not do	.	,	for	?	to	now	at	ago	over	about
up with another	said	says	.	‘s	was	has	does	is	would	did

Table 1: Predicting Fourth Word for Given Trigrams

The embedding in this network does not only care the similarity but also the frequency of the word used in the trigram structure. Analyzing the sensitivity of the network predictions, we can see that the words that are used in similar contexts are more successful to predict than the words that are

irrelevent compared to other words. One of the other facts about this network and word predicting algorithm is that rather than learning and using the meaning and semantics of the words, network cares the frequency of the usage of the words more.

APPENDIX:

```
function oguz_altan_21600966_hw2(question)
clc
close all

switch question
    case '1'
        disp('1')
        %% question 1 code goes here
        disp('The solution of this question is analytical and can be found on the
report');
    case '2'
        disp('2')
        %% question 2 code goes here
        clear
        clc
        load('assign2_data1.mat')
        %imagesc(trainims(:,:,50))

        %creating network parameters
        num_images = size(trainims,3);
        N_hidden = 50;
        std1 = (1025)^(-1/2); %std for hidden layer
        std2 = (N_hidden)^(-1/2); %std for output layer
        W1 = std1*randn(N_hidden,1025); %weights of hidden node
        W2 = std2*randn(1,N_hidden+1); %weights of output node

        batch_size = 38;
        learn_rate = 0.25;
        grad_acc_out = zeros(size(W2));
        grad_acc_hid = zeros(size(W1));
        epoch = 300;
        error_singleim = zeros(1,batch_size);
        error_singleim_test = zeros(1,batch_size);
        error_ave_train = zeros(1,epoch);
        ran_indexes = randperm(num_images);
        mini_batch = 1;

        correct_train_vec = zeros(1,epoch);
        correct_test_vec = zeros(1,epoch);

        %feed forward part

        for epo = 1:epoch

            for i = 1:num_images

                trainims_in_column = trainims(:,:,ran_indexes(i)); %creates image
matrix from mini batch dataset
                x = double(trainims_in_column(:))/255; %casting uint8 to double
                x = [x ;-1]; %adding bias to the weights matrix

                %first feedforward for hidden layer
                v1 = W1*x ; %weighted sum input of hidden layer
                y1 = tanh(v1); %applying activation function to sum
                deriv_act1 = (1-y1.^2); %derivative of the tanh function

                %second feedforward for output layer
                y1 = [y1;-1]; %adding bias to the hidden layer output
                v2 = W2*y1 ; %weighted sum input of output layer
                out_nn = tanh(v2); %applying activation function to output layer
input sum
                deriv_act2 = (1-out_nn.^2); %derivative of the tanh function

                %backpropogation
                error_output = 2*trainlbls(ran_indexes(i))-1-out_nn; %error of
output of the network for single image
```

```
                delta_out = deriv_act2*error_output; %gradient for output layer

                er_hid = W2'*delta_out;
                delta_hidden = deriv_act1.*er_hid(1:(end-1)); %gradient for hidden
layer

                grad_acc_out = grad_acc_out+learn_rate*delta_out*y1'; %output
gradient accumulator
                grad_acc_hid = grad_acc_hid+learn_rate*delta_hidden*x'; %hidden
gradient accumulator

                error_singleim(i) = 1/2*(abs(error_output).^2); %mean square error
for single image in a mini-batch

                %updates parameters after each mini-batch is processed,
                if (mini_batch == batch_size)
                    W2 = W2 + grad_acc_out/batch_size; %updating output weights
matrix
                    W1 = W1 + grad_acc_hid/batch_size; %updating hidden weights
matrix

                    grad_acc_out = zeros(size(W2)); %resetting gradient
accumulators for the next mini-batch
                    grad_acc_hid = zeros(size(W1));

                    mini_batch = 1;
                else
                    mini_batch = mini_batch + 1;
                end
            end

            %one epoch is finished

            error_ave_train(epo) = (sum(error_singleim)/num_images);

            %testing mse and mean square error for each epoch

            %mce and mse for test samples
            W_test1 = W1;
            W_test2 = W2;
            y1_test = 0;
            correct_test = 0;
            out_nn_test = 0;

            for i = 1:size(testims,3)

                testims_in_column = testims(:,:,i);
                x_test = double(testims_in_column(:)/255); %casting uint8 to double
                x_test = [x_test;-1]; %adding bias

                %first feedforward for hidden layer
                %weighted sum input of hidden layer

                y1_test = [tanh(W_test1*x_test);-1]; %adding bias to the hidden
layer output

                %second feedforward for output layer
                %weighted sum input of output layer

                out_nn_test = tanh(W_test2*y1_test); %applying activation function
to output layer input sum

                error_output_test = 2*testlbls(i)-1-out_nn_test; %error of output
of the network for single image
                error_singleim_test(i) = 1/2*(abs(error_output_test).^2); %mean
square error for single image in a mini-batch

                if(sign(out_nn_test) == 2*testlbls(i)-1)
```

```
                correct_test = correct_test+1;
            end
        end

        error_ave_test(epo) = (sum(error_singleim_test)/1900);
        correct_test_vec(epo) = correct_test;

        %mce and mse for train samples
        W_train1 = W1;
        W_train2 = W2;
        y1_train = 0;
        correct_train = 0;
        out_nn_train = 0;

        for i = 1:size(trainims,3)

            trainims_in_column = trainims(:,:,i);
            x = double(trainims_in_column(:)/255); %casting uint8 to double
            x = [x;-1]; %adding bias

            %first feedforward for hidden layer
            %weighted sum input of hidden layer

            y1_train = [tanh(W_train1*x);-1]; %adding bias to the hidden layer
output

            %second feedforward for output layer
            %weighted sum input of output layer

            out_nn_train = tanh(W_train2*y1_train); %applying activation
function to output layer input sum

            if(sign(out_nn_train) == 2*trainlbls(i)-1)
                correct_train = correct_train+1;
            end
        end

        correct_train_vec(epo) = correct_train;
    end

    iter = [1:epoch];

    %Plot MCEs
    figure;
    plot(iter,100-(correct_train_vec*100/1900));
    title('Mean Classification Error (MCE) w/ Nhidden = 50');
    xlabel('Epoch');
    ylabel('Mean Classification Error');
    grid on
    hold on
    plot(iter,100-(correct_test_vec*100/1000));
    legend('MCE for Train Samples', 'MCE for Test Samples');
    grid on
    hold off

    figure;
    plot(iter,error_ave_train);
    title('Mean Square Error (MSE) w/ Nhidden = 50');
    xlabel('Epoch');
    ylabel('Mean Square Error');
    grid on
    hold on
    plot(iter,error_ave_test);
    legend('MSE for Train Samples', 'MSE for Test Samples');
    grid on
    hold off

    % hidden layer neurons H_high = 200
```

```
        %creating network parameters
        num_images = size(trainims,3);
        N_hidden_high = 200;
        std1 = (1025)^(-1/2); %std for hidden layer
        std2 = (N_hidden_high)^(-1/2); %std for output layer
        W1_high = std1*randn(N_hidden_high,1025); %weights of hidden node
        W2_high = std2*randn(1,N_hidden_high+1); %weights of output node
        batch_size = 38;
        learn_rate = 0.25;
        grad_acc_out = zeros(size(W2_high));
        grad_acc_hid = zeros(size(W1_high));
        epoch = 300;
        error_singleim_high = zeros(1,batch_size);
        error_singleim_test_high = zeros(1,batch_size);
        error_epoch = 0;
        error_ave_train_high = zeros(1,epoch);
        ran_indexes = randperm(num_images);
        mini_batch = 1;

        correct_train_vec_high = zeros(1,epoch);
        correct_test_vec_high = zeros(1,epoch);

        %feed forward part

        for epo = 1:epoch

            for i = 1:num_images

                trainims_in_column = trainims(:,:,ran_indexes(i)); %creates image
matrix from mini batch dataset
                x = double(trainims_in_column(:))/255; %casting uint8 to double
                x = [x ;-1]; %adding bias to the weights matrix

                %first feedforward for hidden layer
                v1 = W1_high*x ; %weighted sum input of hidden layer
                y1 = tanh(v1); %applying activation function to sum
                deriv_act1 = (1-y1.^2); %derivative of the tanh function

                %second feedforward for output layer
                y1 = [y1;-1]; %adding bias to the hidden layer output
                v2 = W2_high*y1 ; %weighted sum input of output layer
                out_nn = tanh(v2); %applying activation function to output layer
input sum
                deriv_act2 = (1-out_nn.^2); %derivative of the tanh function

                %backpropogation
                error_output = 2*trainlbls(ran_indexes(i))-1-out_nn; %error of
output of the network for single image
                delta_out = deriv_act2*error_output; %gradient for output layer

                er_hid = W2_high'*delta_out;
                delta_hidden = deriv_act1.*er_hid(1:(end-1)); %gradient for hidden
layer

                grad_acc_out = grad_acc_out+learn_rate*delta_out*y1'; %output
gradient accumulator
                grad_acc_hid = grad_acc_hid+learn_rate*delta_hidden*x'; %hidden
gradient accumulator

                error_singleim_high(i) = 1/2*(abs(error_output).^2); %mean square
error for single image in a mini-batch

                %updates parameters after each mini-batch is processed,
                if (mini_batch == batch_size)
                    W2_high = W2_high + grad_acc_out/batch_size; %updating output
weights matrix
```

```
                    W1_high = W1_high + grad_acc_hid/batch_size; %updating hidden
weights matrix

                    grad_acc_out = zeros(size(W2_high)); %resetting gradient
accumulators for the next mini-batch
                    grad_acc_hid = zeros(size(W1_high));

                    mini_batch = 1;
                else
                    mini_batch = mini_batch + 1;
                end
            end

            %one epoch is finished

            error_ave_train_high(epo) = (sum(error_singleim_high)/num_images);

            %testing mse and mean square error for each epoch

            %mce and mse for test samples
            W_test1_high = W1_high;
            W_test2_high = W2_high;
            y1_test_high = 0;
            correct_test_high = 0;
            out_nn_test_high = 0;

            for i = 1:size(testims,3)

                testims_in_column = testims(:,:,i);
                x_test = double(testims_in_column(:)/255); %casting uint8 to double
                x_test = [x_test;-1]; %adding bias

                %first feedforward for hidden layer
                %weighted sum input of hidden layer

                y1_test_high = [tanh(W_test1_high*x_test);-1]; %adding bias to the
hidden layer output

                %second feedforward for output layer
                %weighted sum input of output layer

                out_nn_test_high = tanh(W_test2_high*y1_test_high); %applying
activation function to output layer input sum

                error_output_test_high = 2*testlbls(i)-1-out_nn_test_high; %error
of output of the network for single image
                error_singleim_test_high(i) = 1/2*(abs(error_output_test_high).^2);
%mean square error for single image in a mini-batch

                if(sign(out_nn_test_high) == 2*testlbls(i)-1)
                    correct_test_high = correct_test_high+1;
                end
            end

            error_ave_test_high(epo) = (sum(error_singleim_test_high)/1900);
            correct_test_vec_high(epo) = correct_test_high;

            %mce and mse for train samples
            W_train1_high = W1_high;
            W_train2_high = W2_high;
            y1_train_high = 0;
            correct_train_high = 0;
            out_nn_train_high = 0;

            for i = 1:size(trainims,3)

                trainims_in_column = trainims(:,:,i);
                x = double(trainims_in_column(:)/255); %casting uint8 to double
```

```
                x = [x;-1]; %adding bias

                %first feedforward for hidden layer
                %weighted sum input of hidden layer

                y1_train_high = [tanh(W_train1_high*x);-1]; %adding bias to the
hidden layer output

                %second feedforward for output layer
                %weighted sum input of output layer

                out_nn_train_high = tanh(W_train2_high*y1_train_high); %applying
activation function to output layer input sum

                if(sign(out_nn_train_high) == 2*trainlbls(i)-1)
                    correct_train_high = correct_train_high+1;
                end
            end

            correct_train_vec_high(epo) = correct_train_high;
        end

        iter = [1:epoch];

        %Plot MCEs
        figure;
        plot(iter,100-(correct_train_vec_high*100/1900));
        title('Mean Classification Error (MCE) w/ Nhidden = 200');
        xlabel('Epoch');
        ylabel('Mean Classification Error');
        grid on
        hold on
        plot(iter,100-(correct_test_vec_high*100/1000));
        legend('MCE for Train Samples', 'MCE for Test Samples');
        grid on
        hold off

        figure;
        plot(iter,error_ave_train_high);
        title('Mean Square Error (MSE) w/ Nhidden = 200');
        xlabel('Epoch');
        ylabel('Mean Square Error');
        grid on
        hold on
        plot(iter,error_ave_test_high);
        legend('MSE for Train Samples', 'MSE for Test Samples');
        grid on
        hold off

        %hidden layer neurons H_low = 10

        %creating network parameters
        num_images = size(trainims,3);
        N_hidden_low = 10;
        std1 = (1025)^(-1/2); %std for hidden layer
        std2 = (N_hidden_low)^(-1/2); %std for output layer
        W1_low = std1*randn(N_hidden_low,1025); %weights of hidden node
        W2_low = std2*randn(1,N_hidden_low+1); %weights of output node
        batch_size = 38;
        learn_rate = 0.25;
        grad_acc_out = zeros(size(W2_low));
        grad_acc_hid = zeros(size(W1_low));
        epoch = 300;
        error_singleim_low = zeros(1,batch_size);
        error_singleim_test_low = zeros(1,batch_size);
        error_epoch = 0;
        error_ave_train_low = zeros(1,epoch);
        ran_indexes = randperm(num_images);
```

```
        mini_batch = 1;

        correct_train_vec_low = zeros(1,epoch);
        correct_test_vec_low = zeros(1,epoch);

        %feed forward part

        for epo = 1:epoch

            for i = 1:num_images

                trainims_in_column = trainims(:,:,ran_indexes(i)); %creates image
matrix from mini batch dataset
                x = double(trainims_in_column(:))/255; %casting uint8 to double
                x = [x ;-1]; %adding bias to the weights matrix

                %first feedforward for hidden layer
                v1 = W1_low*x ; %weighted sum input of hidden layer
                y1 = tanh(v1); %applying activation function to sum
                deriv_act1 = (1-y1.^2); %derivative of the tanh function

                %second feedforward for output layer
                y1 = [y1;-1]; %adding bias to the hidden layer output
                v2 = W2_low*y1 ; %weighted sum input of output layer
                out_nn = tanh(v2); %applying activation function to output layer
input sum
                deriv_act2 = (1-out_nn.^2); %derivative of the tanh function

                %backpropogation
                error_output = 2*trainlbls(ran_indexes(i))-1-out_nn; %error of
output of the network for single image
                delta_out = deriv_act2*error_output; %gradient for output layer

                er_hid = W2_low'*delta_out;
                delta_hidden = deriv_act1.*er_hid(1:(end-1)); %gradient for hidden
layer

                grad_acc_out = grad_acc_out+learn_rate*delta_out*y1'; %output
gradient accumulator
                grad_acc_hid = grad_acc_hid+learn_rate*delta_hidden*x'; %hidden
gradient accumulator

                error_singleim_low(i) = 1/2*(abs(error_output).^2); %mean square
error for single image in a mini-batch

                %updates parameters after each mini-batch is processed,
                if (mini_batch == batch_size)
                    W2_low = W2_low + grad_acc_out/batch_size; %updating output
weights matrix
                    W1_low = W1_low + grad_acc_hid/batch_size; %updating hidden
weights matrix

                    grad_acc_out = zeros(size(W2_low)); %resetting gradient
accumulators for the next mini-batch
                    grad_acc_hid = zeros(size(W1_low));

                    mini_batch = 1;
                else
                    mini_batch = mini_batch + 1;
                end
            end

            %one epoch is finished

            error_ave_train_low(epo) = (sum(error_singleim_low)/num_images);

            %testing mse and mean square error for each epoch
```

```
            W_test1_low = W1_low;
            W_test2_low = W2_low;
            y1_test_low = 0;
            correct_test_low = 0;
            out_nn_test_low = 0;

            for i = 1:size(testims,3)

                testims_in_column = testims(:,:,i);
                x_test = double(testims_in_column(:)/255); %casting uint8 to double
                x_test = [x_test;-1]; %adding bias

                %first feedforward for hidden layer
                %weighted sum input of hidden layer

                y1_test_low = [tanh(W_test1_low*x_test);-1]; %adding bias to the
hidden layer output

                %second feedforward for output layer
                %weighted sum input of output layer

                out_nn_test_low = tanh(W_test2_low*y1_test_low); %applying
activation function to output layer input sum

                error_output_test_low = 2*testlbls(i)-1-out_nn_test_low; %error of
output of the network for single image
                error_singleim_test_low(i) = 1/2*(abs(error_output_test_low).^2);
%mean square error for single image in a mini-batch

                if(sign(out_nn_test_low) == 2*testlbls(i)-1)
                    correct_test_low = correct_test_low+1;
                end
            end

            error_ave_test_low(epo) = (sum(error_singleim_test_low)/1900);
            correct_test_vec_low(epo) = correct_test_low;

            %mce for train samples
            W_train1_low = W1_low;
            W_train2_low = W2_low;
            y1_train_low = 0;
            correct_train_low = 0;
            out_nn_train_low = 0;
            correct_train_low = 0;

            for i = 1:size(trainims,3)

                trainims_in_column = trainims(:,:,i);
                x = double(trainims_in_column(:)/255); %casting uint8 to double
                x = [x;-1]; %adding bias

                %first feedforward for hidden layer
                %weighted sum input of hidden layer

                y1_train_low = [tanh(W_train1_low*x);-1]; %adding bias to the
hidden layer output

                %second feedforward for output layer
                %weighted sum input of output layer

                out_nn_train_low = tanh(W_train2_low*y1_train_low); %applying
activation function to output layer input sum

                if(sign(out_nn_train_low) == 2*trainlbls(i)-1)
                    correct_train_low = correct_train_low+1;
                end
            end
```

```
            correct_train_vec_low(epo) = correct_train_low;
        end

        iter = [1:epoch];

        %Plot MCEs
        figure;
        plot(iter,100-(correct_train_vec_low*100/1900));
        title('Mean Classification Error (MCE) w/ Nhidden = 10');
        xlabel('Epoch');
        ylabel('Mean Classification Error');
        grid on
        hold on
        plot(iter,100-(correct_test_vec_low*100/1000));
        legend('MCE for Train Samples', 'MCE for Test Samples');
        grid on
        hold off

        figure;
        plot(iter,error_ave_train_low);
        title('Mean Square Error (MSE) w/ Nhidden = 10');
        xlabel('Epoch');
        ylabel('Mean Square Error');
        grid on
        hold on
        plot(iter,error_ave_test_low);
        legend('MSE for Train Samples', 'MSE for Test Samples');
        grid on
        hold off

        %comparing plots

        iter = [1:epoch];
        figure;
        plot(iter,error_ave_train);
        grid on
        title('MSEs for 3 Different Number of Hidden Layer Neurons')
        xlabel('Epoch');
        ylabel('Mean Square Error');
        hold on;
        plot(iter,error_ave_test);
        plot(iter,error_ave_train_high);
        plot(iter,error_ave_test_high);
        plot(iter,error_ave_train_low);
        plot(iter,error_ave_test_low);
        legend('Nhidden = 10 & Train MSE','Nhidden = 10 & Test MSE', 'Nhidden = 50
& Train MSE','Nhidden = 50 & Test MSE', 'Nhidden = 200 & Train MSE','Nhidden = 200
& Test MSE');
        hold off

        figure;
        plot(iter,100-(correct_train_vec*100/1900));
        grid on
        title('MCEs for 3 Different Number of Hidden Layer Neurons')
        xlabel('Epoch');
        ylabel('Mean Classification Error');
        hold on;
        plot(iter,100-(correct_test_vec*100/1000));
        plot(iter,100-(correct_train_vec_high*100/1900));
        plot(iter,100-(correct_test_vec_high*100/1000));
        plot(iter,100-(correct_train_vec_low*100/1900));
        plot(iter,100-(correct_test_vec_low*100/1000));
        legend('Nhidden = 10 & Train MCE','Nhidden = 10 & Test MCE', 'Nhidden = 50
& Train MCE','Nhidden = 50 & Test MCE', 'Nhidden = 200 & Train MCE','Nhidden = 200
& Test MCE');
        hold off

        % two hidden layers
```

```
        %creating network parameters

        N1_hidden = 80;
        N2_hidden = 50;
        std1 = (1025)^(-1/2); %std for hidden layer
        std2 = (N1_hidden)^(-1/2); %std for output layer
        std3 = (N2_hidden)^(-1/2);
        W1_twohid = std1*randn(N1_hidden,1025); %weights of hidden node
        W2_twohid = std2*randn(N2_hidden,N1_hidden+1); %weights of output node
        W3_twohid = std3*randn(1,N2_hidden+1);
        num_images = size(trainims,3);
        batch_size = 38;
        learn_rate = 0.25;
        grad_acc_out = zeros(size(W3_twohid));
        grad_acc_hid2 = zeros(size(W2_twohid));
        grad_acc_hid1 = zeros(size(W1_twohid));
        epoch = 300;
        error_singleim_twohid = zeros(1,batch_size);
        error_ave_twohid = zeros(1,epoch);
        ran_indexes = randperm(num_images);
        mini_batch = 1;
        correct_train_vec_twohid = zeros(1,epoch);
        correct_test_vec_twohid = zeros(1,epoch);

        %saving weights for momentum
        W1_mom = W1_twohid;
        W2_mom = W2_twohid;
        W3_mom = W3_twohid;

        %feed forward part

        for epo = 1:epoch

            for i = 1:num_images

                trainims_in_column = trainims(:,:,ran_indexes(i)); %creates image
matrix from mini batch dataset
                x_train_twohid = double(trainims_in_column(:))/255; %casting uint8
to double
                x_train_twohid = [x_train_twohid ;-1]; %adding bias to the weights
matrix

                %first feedforward for first hidden layer
                v1 = W1_twohid*x_train_twohid ; %weighted sum input of hidden layer
                y1 = tanh(v1); %applying activation function to sum
                deriv_act1 = (1-y1.^2); %derivative of the tanh function

                %second feedforward for second hidden layer
                y1 = [y1;-1];
                v2 = W2_twohid*y1;
                y2 = tanh(v2);
                deriv_act2 = (1-y2.^2);

                %third feedforward for output layer
                y2 = [y2;-1];
                v3 = W3_twohid*y2;
                out_nn_twohid = tanh(v3);
                deriv_act3 = (1-out_nn_twohid.^2);

                %backpropogation
                error_output_twohid = 2*trainlbls(ran_indexes(i))-1-out_nn_twohid;
%error of output of the network for single image
                delta_out = deriv_act3*error_output_twohid; %gradient for output
layer

                er_hid2 = W3_twohid'*delta_out;
```

```
                delta_hidden2 = deriv_act2.*er_hid2(1:(end-1)); %gradient for
hidden layer

                er_hid1 = W2_twohid'*delta_hidden2;
                delta_hidden1 = deriv_act1.*er_hid1(1:(end-1)); %gradient for
hidden layer

                grad_acc_out = grad_acc_out+learn_rate*delta_out*y2'; %output
gradient accumulator
                grad_acc_hid2 = grad_acc_hid2+learn_rate*delta_hidden2*y1'; %hidden
gradient accumulator
                grad_acc_hid1 =
grad_acc_hid1+learn_rate*delta_hidden1*x_train_twohid'; %hidden gradient
accumulator

                error_singleim_twohid(i) = 1/2*(abs(error_output_twohid).^2); %mean
square error for single image in a mini-batch

                %updates parameters after each mini-batch is processed,
                if (mini_batch == batch_size)
                    W3_twohid = W3_twohid + grad_acc_out/batch_size; %updating
output weights matrix
                    W2_twohid = W2_twohid + grad_acc_hid2/batch_size; %updating
output weights matrix
                    W1_twohid = W1_twohid + grad_acc_hid1/batch_size; %updating
hidden weights matrix

                    grad_acc_out = zeros(size(W3_twohid));
                    grad_acc_hid2 = zeros(size(W2_twohid));
                    grad_acc_hid1 = zeros(size(W1_twohid));

                    mini_batch = 1;
                else
                    mini_batch = mini_batch + 1;
                end
            end

            %one epoch is finished

            error_ave_train_twohid(epo) = (sum(error_singleim_twohid)/num_images);

            %testing mse and mean square error for each epoch

            %mce and mse for test samples
            W_test1_twohid = W1_twohid;
            W_test2_twohid = W2_twohid;
            W_test3_twohid = W3_twohid;
            y1_test_twohid = 0;
            y2_test_twohid = 0;
            correct_test_twohid = 0;
            out_nn_test_twohid = 0;

            for i = 1:size(testims,3)

                testims_in_column = testims(:,:,i);
                x_test_twohid = double(testims_in_column(:)/255); %casting uint8 to
double
                x_test_twohid = [x_test_twohid;-1]; %adding bias

                %first feedforward for first hidden layer
                %weighted sum input of first hidden layer

                y1_test_twohid = [tanh(W_test1_twohid*x_test_twohid);-1]; %adding
bias to the hidden layer output

                %second feedforward for second hidden layer
                %weighted sum input of second hidden layer
```

```
                y2_test_twohid = [tanh(W_test2_twohid*y1_test_twohid);-1]; %adding
bias to the hidden layer output

                %third feedforward for output layer
                %weighted sum input of output layer
                out_nn_test_twohid = tanh(W_test3_twohid*y2_test_twohid); %applying
activation function to output layer input sum

                error_output_test_twohid = 2*testlbls(i)-1-out_nn_test_twohid;
%error of output of the network for single image
                error_singleim_test_twohid(i) =
1/2*(abs(error_output_test_twohid).^2); %mean square error for single image in a
mini-batch

                if(sign(out_nn_test_twohid) == 2*testlbls(i)-1)
                    correct_test_twohid = correct_test_twohid+1;
                end
            end

            error_ave_test_twohid(epo) = (sum(error_singleim_test_twohid)/1900);
            correct_test_vec_twohid(epo) = correct_test_twohid;

            %mce and mse for train samples
            W_train1_twohid = W1_twohid;
            W_train2_twohid = W2_twohid;
            W_train3_twohid = W3_twohid;
            y1_train_twohid = 0;
            y2_train_twohid = 0;
            correct_train_twohid = 0;
            out_nn_train_twohid = 0;

            for i = 1:size(trainims,3)

                trainims_in_column = trainims(:,:,i);
                x = double(trainims_in_column(:)/255); %casting uint8 to double
                x = [x;-1]; %adding bias

                %first feedforward for first hidden layer
                %weighted sum input of first hidden layer

                y1_train_twohid = [tanh(W_train1_twohid*x);-1]; %adding bias to the
hidden layer output

                %second feedforward for second hidden layer
                %weighted sum input of second hidden layer

                y2_train_twohid = [tanh(W_train2_twohid*y1_train_twohid);-1];
%adding bias to the hidden layer output

                %third feedforward for output layer
                %weighted sum input of output layerr
                out_nn_train_twohid = tanh(W_train3_twohid*y2_train_twohid);
%applying activation function to output layer input sum

                if(sign(out_nn_train_twohid) == 2*trainlbls(i)-1)
                    correct_train_twohid = correct_train_twohid+1;
                end
            end

            correct_train_vec_twohid(epo) = correct_train_twohid;
        end

        iter = [1:epoch];

        %Plot MCEs
        figure;
        plot(iter,100-(correct_train_vec_twohid*100/1900));
        title('Mean Classification Error (MCE) w/ Two Hidden Layers');
```

```
xlabel('Epoch');
ylabel('Mean Classification Error');
grid on
hold on
plot(iter,100-(correct_test_vec_twohid*100/1000));
legend('MCE for Train Samples', 'MCE for Test Samples');
grid on
hold off

%Plot MSEs
figure;
plot(iter,error_ave_train_twohid);
title('Mean Square Error (MSE) w/ Two Hidden Layers');
xlabel('Epoch');
ylabel('Mean Square Error');
grid on
hold on
plot(iter,error_ave_test_twohid);
legend('MSE for Train Samples', 'MSE for Test Samples');
grid on
hold off


% learning with momentum

%creating network parameters
%using same weight parameters for part d) two hidden layer nn

N1_hidden = 80;
N2_hidden = 50;
std1 = (1025)^(-1/2); %std for first hidden layer
std2 = (N1_hidden)^(-1/2); %std second hidden layer
std3 = (N2_hidden)^(-1/2); %std for outut layer

num_images = size(trainims,3);
batch_size = 38;
learn_rate = 0.25;

grad_acc_out = zeros(size(W3_mom));
grad_acc_hid2 = zeros(size(W2_mom));
grad_acc_hid1 = zeros(size(W1_mom));

grad_acc_out_prev = zeros(size(W3_mom));
grad_acc_hid2_prev = zeros(size(W2_mom));
grad_acc_hid1_prev = zeros(size(W1_mom));

epoch = 300;
error_singleim_twohid_mom = zeros(1,batch_size);
error_ave_twohid_mom = zeros(1,epoch);
ran_indexes = randperm(num_images);
mini_batch = 1;
correct_train_vec_twohid_mom = zeros(1,epoch);
correct_test_vec_twohid_mom = zeros(1,epoch);
momentum = 0.3

%feed forward part

for epo = 1:epoch

    for i = 1:num_images

        trainims_in_column = trainims(:,:,ran_indexes(i)); %creates image
matrix from mini batch dataset
        x_train_twohid_mom = double(trainims_in_column(:))/255; %casting
uint8 to double
        x_train_twohid_mom = [x_train_twohid_mom ;-1]; %adding bias to the
weights matrix
```

```
                %first feedforward for first hidden layer
                v1 = W1_mom*x_train_twohid_mom ; %weighted sum input of hidden
layer
                y1 = tanh(v1); %applying activation function to sum
                deriv_act1 = (1-y1.^2); %derivative of the tanh function

                %second feedforward for second hidden layer
                y1 = [y1;-1];
                v2 = W2_mom*y1;
                y2 = tanh(v2);
                deriv_act2 = (1-y2.^2);

                %third feedforward for output layer
                y2 = [y2;-1];
                v3 = W3_mom*y2;
                out_nn_twohid = tanh(v3);
                deriv_act3 = (1-out_nn_twohid.^2);

                %backpropogation
                error_output_twohid_mom = 2*trainlbls(ran_indexes(i))-1-
out_nn_twohid; %error of output of the network for single image
                delta_out = deriv_act3*error_output_twohid_mom; %gradient for
output layer

                er_hid2 = W3_mom'*delta_out;
                delta_hidden2 = deriv_act2.*er_hid2(1:(end-1)); %gradient for
hidden layer

                er_hid1 = W2_mom'*delta_hidden2;
                delta_hidden1 = deriv_act1.*er_hid1(1:(end-1)); %gradient for
hidden layer

                grad_acc_out = grad_acc_out+learn_rate*delta_out*y2'; %output
gradient accumulator
                grad_acc_hid2 = grad_acc_hid2+learn_rate*delta_hidden2*y1'; %hidden
gradient accumulator
                grad_acc_hid1 =
grad_acc_hid1+learn_rate*delta_hidden1*x_train_twohid_mom'; %hidden gradient
accumulator

                error_singleim_twohid_mom(i) =
1/2*(abs(error_output_twohid_mom).^2); %mean square error for single image in a
mini-batch

                %updates parameters after each mini-batch is processed,
                if (mini_batch == batch_size)
                    W3_mom = W3_mom + (grad_acc_out +
grad_acc_out_prev*momentum)/batch_size; %updating output weights matrix
                    W2_mom = W2_mom + (grad_acc_hid2 +
grad_acc_hid2_prev*momentum)/batch_size; %updating output weights matrix
                    W1_mom = W1_mom + (grad_acc_hid1 +
grad_acc_hid1_prev*momentum)/batch_size; %updating hidden weights matrix

                    grad_acc_out_prev = grad_acc_out;
                    grad_acc_hid2_prev = grad_acc_hid2;
                    grad_acc_hid1_prev = grad_acc_hid1;

                    grad_acc_out = zeros(size(W3_mom)); %resetting gradient
accumulators for the next mini-batch
                    grad_acc_hid2 = zeros(size(W2_mom));
                    grad_acc_hid1 = zeros(size(W1_mom));

                    mini_batch = 1;
                else
                    mini_batch = mini_batch + 1;
                end
            end
```

```
            %one epoch is finished

            error_ave_train_twohid_mom(epo) =
(sum(error_singleim_twohid_mom)/num_images);

            %testing mse and mean square error for each epoch

            %mce and mse for test samples
            W_test1_twohid_mom = W1_mom;
            W_test2_twohid_mom = W2_mom;
            W_test3_twohid_mom = W3_mom;
            y1_test_twohid_mom = 0;
            y2_test_twohid_mom = 0;
            correct_test_twohid_mom = 0;
            out_nn_test_twohid_mom = 0;

            for i = 1:size(testims,3)

                testims_in_column = testims(:,:,i);
                x_test_twohid_mom = double(testims_in_column(:)/255); %casting
uint8 to double
                x_test_twohid_mom = [x_test_twohid_mom;-1]; %adding bias

                %first feedforward for first hidden layer
                %weighted sum input of first hidden layer

                y1_test_twohid_mom = [tanh(W_test1_twohid_mom*x_test_twohid_mom);-
1]; %adding bias to the hidden layer output

                %second feedforward for second hidden layer
                %weighted sum input of second hidden layer

                y2_test_twohid_mom = [tanh(W_test2_twohid_mom*y1_test_twohid_mom);-
1]; %adding bias to the hidden layer output

                %third feedforward for output layer
                %weighted sum input of output layer
                out_nn_test_twohid_mom =
tanh(W_test3_twohid_mom*y2_test_twohid_mom); %applying activation function to
output layer input sum

                error_output_test_twohid_mom = 2*testlbls(i)-1-
out_nn_test_twohid_mom; %error of output of the network for single image
                error_singleim_test_twohid_mom(i) =
1/2*(abs(error_output_test_twohid_mom).^2); %mean square error for single image in
a mini-batch

                if(sign(out_nn_test_twohid_mom) == 2*testlbls(i)-1)
                    correct_test_twohid_mom = correct_test_twohid_mom+1;
                end
            end

            error_ave_test_twohid_mom(epo) =
(sum(error_singleim_test_twohid_mom)/1900);
            correct_test_vec_twohid_mom(epo) = correct_test_twohid_mom;

            %mce and mse for train samples
            W_train1_twohid_mom = W1_mom;
            W_train2_twohid_mom = W2_mom;
            W_train3_twohid_mom = W3_mom;
            y1_train_twohid_mom = 0;
            y2_train_twohid_mom = 0;
            correct_train_twohid_mom = 0;
            out_nn_train_twohid_mom = 0;

            for i = 1:size(trainims,3)

                trainims_in_column = trainims(:,:,i);
```

```
                x = double(trainims_in_column(:)/255); %casting uint8 to double
                x = [x;-1]; %adding bias

                %first feedforward for first hidden layer
                %weighted sum input of first hidden layer

                y1_train_twohid_mom = [tanh(W_train1_twohid_mom*x);-1]; %adding
bias to the hidden layer output

                %second feedforward for second hidden layer
                %weighted sum input of second hidden layer

                y2_train_twohid_mom =
[tanh(W_train2_twohid_mom*y1_train_twohid_mom);-1]; %adding bias to the hidden
layer output

                %third feedforward for output layer
                %weighted sum input of output layerr
                out_nn_train_twohid_mom =
tanh(W_train3_twohid_mom*y2_train_twohid_mom); %applying activation function to
output layer input sum

                %error_output_train_twohid = 2*trainlbls(i)-1-out_nn_train_twohid;
%error of output of the network for single image
                %error_singleim_train_twohid(i) =
1/2*(abs(error_output_train_twohid).^2); %mean square error for single image in a
mini-batch

                if(sign(out_nn_train_twohid_mom) == 2*trainlbls(i)-1)
                    correct_train_twohid_mom = correct_train_twohid_mom+1;
                end
            end

            correct_train_vec_twohid_mom(epo) = correct_train_twohid_mom;
        end

        iter = [1:epoch];

        %Plot MCEs
        figure;
        plot(iter,100-(correct_train_vec_twohid_mom*100/1900));
        title('Mean Classification Error (MCE) w/ Two Hidden Layers and Momentum
Learning');
        xlabel('Epoch');
        ylabel('Mean Classification Error');
        grid on
        hold on
        plot(iter,100-(correct_test_vec_twohid_mom*100/1000));
        legend('MCE for Train Samples', 'MCE for Test Samples');
        grid on
        hold off

        %Plot MSEs
        figure;
        plot(iter,error_ave_train_twohid_mom);
        title('Mean Square Error (MSE) w/ Two Hidden Layers and Momentum
Learning');
        xlabel('Epoch');
        ylabel('Mean Square Error');
        grid on
        hold on
        plot(iter,error_ave_test_twohid_mom);
        legend('MSE for Train Samples', 'MSE for Test Samples');
        grid on
        hold off

    case '3'
        disp('3')
```

```
%% question 3 code goes here

%loading data and setting network parameters
load('assign2_data2.mat')

D = 8;
P = 64;
learn = 0.15;
momentum = 0.85;
std = 0.01;

batch_size = 250;
batch_num = length(traind)/ batch_size;
epoch_num = 50;

%initalizing weights with normal distribution
W_embed = std*randn(batch_size,D);
W_hid = randn(P,D+1)*std;
W_out =  randn(250,P+1)*std;

%shuffling train dataset index
rand_in = randperm(length(traind));

%setting up gradient accumulators for layers
grad_acc_out = zeros(size(W_out));
grad_acc_hid = zeros(size(W_hid));
grad_acc_embed = zeros(size(W_embed));

%creating gradient accumulators for previous gradients in
%momentum method
grad_acc_out_prev = zeros(size(W_out));
grad_acc_hid_prev = zeros(size(W_hid));
grad_acc_embed_prev = zeros(size(W_embed));

%
train_cross_ent = zeros(1,epoch_num);
valid_nn_out = ones(P+1,length(vald));
val_correct = zeros(1,epoch_num);
val_cross_ent = zeros(1,epoch_num);

%runs over epoches
for i = 1:epoch_num

    disp("Epoch No: " + i)

    %runs over mini-batches

    for j = 1:batch_num

        %takes train sample data
        train_ind = rand_in((1+(j-1)*batch_size):batch_size*j);
        train_data = double(trainx(:,train_ind));

        %creating words matrix from zeros
        words_mat = zeros(batch_size,250);

        %network output matrix for train samples
        train_out = zeros(250,batch_size);

        %fills words matrix with assigned values for words
        for k = 1:batch_size
            words_mat(k,train_data(1,k)) = words_mat(k,train_data(1,k))+1;
            words_mat(k,train_data(2,k)) = words_mat(k,train_data(2,k))+1;
            words_mat(k,train_data(3,k)) = words_mat(k,train_data(3,k))+1;
            train_out(traind(train_ind(k)),k) = 1;
        end

        nn_input = [(words_mat*W_embed)'; repmat(-1,[1,batch_size])];
```

```
                %feedforwarding algorithm
                vh = W_hid*nn_input;
                yh = [1./(1+exp(-vh));repmat(-1,[1,250])];
                vo = W_out*yh;
                yo = normSoftMax(vo);

                % backpropagation algorithm
                err_out = train_out - yo;
                grad_out = (err_out*yh')/batch_size;
                err_hid = (W_out'*err_out).*(yh.*(1-yh));
                grad_hid = ((err_hid(1:end-1,:))*nn_input')/batch_size;
                err_embed = (W_hid'*err_hid(1:(end-1),:));
                grad_embed = ((err_embed(1:end-1,:)*(words_mat)))'/batch_size;

                %update the weights at the end of mini-batch
                W_out = W_out + (learn*grad_out+ momentum* grad_acc_out_prev);
                W_hid = W_hid + (learn*grad_hid + momentum* grad_acc_hid_prev);
                W_embed = W_embed + (learn*grad_embed + momentum*
grad_acc_embed_prev);

                %setting up previous gradients accumulators
                grad_acc_out_prev = learn*grad_out;
                grad_acc_hid_prev = learn*grad_hid;
                grad_acc_embed_prev = learn*grad_embed;

                train_cross_ent(i) = - mean(sum(train_out.*log2(yo)));

            end

            val_data = double(valx);
            words_mat = zeros(length(vald),250);
            val_out = zeros(250,length(vald));

            %
            for k = 1:length(vald)
                words_mat(k,val_data(1,k)) = words_mat(k,val_data(1,k))+1;
                words_mat(k,val_data(2,k)) = words_mat(k,val_data(2,k))+1;
                words_mat(k,val_data(3,k)) = words_mat(k,val_data(3,k))+1;
                val_out(vald(k),k) = 1;
            end

            %
            nn_input = [(words_mat*W_embed)'; repmat(-1,[1,length(vald)])];
            valid_vh = W_hid*nn_input;
            valid_nn_out(1:(end-1),:) = 1./(1+exp(-valid_vh));
            valid_nn_o = normSoftMax(W_out*valid_nn_out);
            [m,k] = max (valid_nn_o);

            % Validation Errors
            val_cross_ent(i) = - mean(sum(val_out.*log2(valid_nn_o)));
            if (- mean(sum(val_out.*log2(valid_nn_o)))< 4.5)
                break
            end
            val_correct(i) = mean(k == vald)*100;

            %early stop process with threshold 0.05
            if(i>10)
                if(val_correct(i) - val_correct(i-1) < 0.05 && val_correct(i-1) -
val_correct(i-2) < 0.05)
                    disp("Early Stop");
                    break;
                end
            end
            disp("Correct: " + val_correct(i) + "%");
        end

        figure;
```

```
        save('d_validation er.mat', 'val_cross_ent', 'val_correct');

        plot(1:epoch_num, val_correct)
        grid on;
        title(sprintf('Validation Correctness with respect to Epochs D = %d', D))
        xlabel('Epochs')
        ylabel('Correctness Percentage')

        figure
        plot(1:epoch_num, val_cross_ent)
        grid on;

        hold on

        plot(1:epoch_num, train_cross_ent)
        grid on;
        title(sprintf('Cross Entropy with respect to Epochs D = %d', D))
        xlabel('Epochs')
        ylabel('Cross Entropy')
        legend('Validation Data','Training Data')

        % Testing the network with testing data
        test_index = randperm(length(testd));
        test_index = test_index(1:5);
        test_data = double(testx(:,test_index));
        test_yh = ones(P+1,length(test_index));
        test_words = zeros(length(test_index),250);
        test_out = zeros(250,length(test_index));

        for k = 1:length(test_index)
            test_words(k,test_data(1,k)) = test_words(k,test_data(1,k))+1;
            test_words(k,test_data(2,k)) = test_words(k,test_data(2,k))+1;
            test_words(k,test_data(3,k)) = test_words(k,test_data(3,k))+1;
            test_out(testd(test_index(k)),k) = 1;
        end

        %feedforward to predict the words
        nn_input = [(test_words*W_embed)'; repmat(-1,[1,length(test_index)])];
        test_vh = W_hid*nn_input;
        test_yh(1:end-1,:) = 1./(1+exp(-test_vh));
        test_yo = normSoftMax(W_out*test_yh);
        [m, t] = maxk(test_yo,10);

        %Prints trigram, label word and predicted fourth words with
        %associated probabilities
        for k= 1:5
            disp("Trigram: " + words(1,testx(1,testd(test_index(k)))) + " " +
words(1,testx(2,testd(test_index(k))) )+ " " +
words(1,testx(3,testd(test_index(k)))));
            disp("Label: " + words(1,testd(test_index(k))));

            for z = 1:10
                disp("Predicted fourth word " + z + " is: " + words(1,t(z,k)) + "
with probability: " + m(z,k));
            end
        end
end
end

function y = normSoftMax(x)
normz = exp(x- max(x));
y = normz./sum(normz);
end
```