# EEE443 Neural Networks

# Homework 2 Report

# Name: Oğuz Altan

# ID: 21600966

Question 1)

The cost function is given as:

1. The derivations are needed for minimization of both cost function and . To simplify the notations, vectorized notation is to be used, where corresponds to and to .

Cost function can be written as:

The terms and are both scalar and therefore equal to each other. Thus, we can rewrite our cost function as:

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

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

We are given another cost function:

Minimizing C2:

Equating these two cost function equations, we get:

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

1. As we know

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

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

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

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

For the update rule, we can state:

As the weight vector can be rewritten in term of eigenvectors of using eigenvalue decomposition, we can rewrite the update rule as:

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

Considering this stability condition, tha maximum learning rate can be obained as follows:

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.

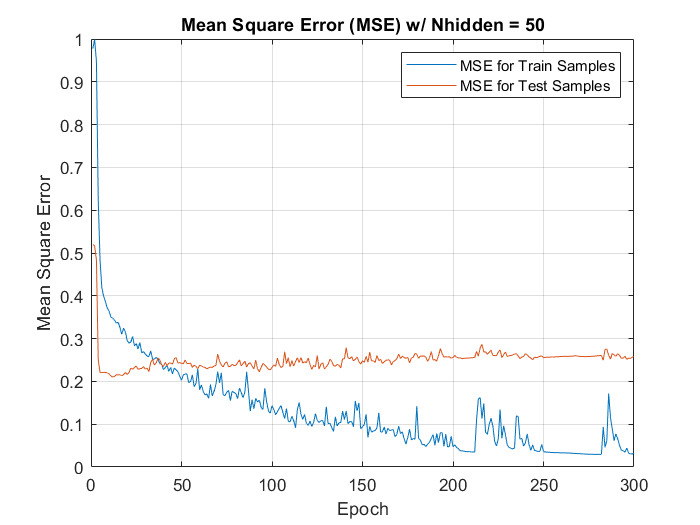


Figure 1: Mean Square Error Plots for 50 Hidden Neurons Architecture

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.

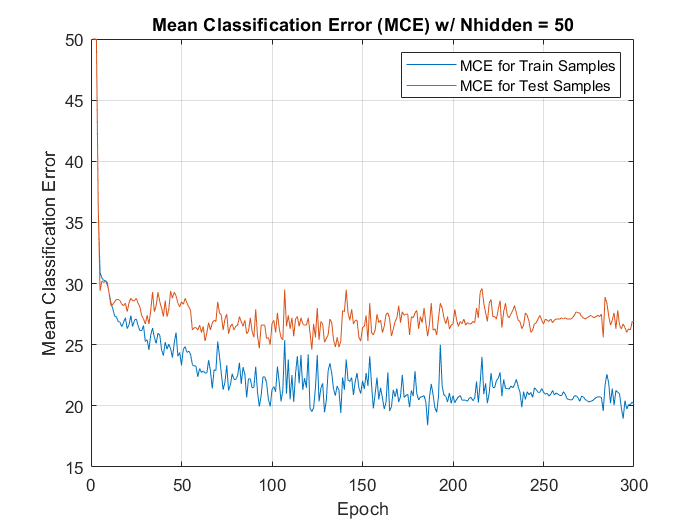


Figure 2: Mean Classification Error Plots for 50 Hidden Neurons Architecture

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

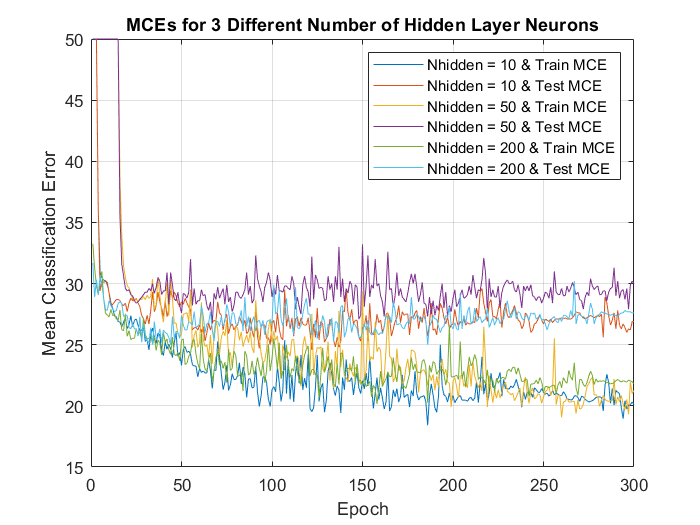
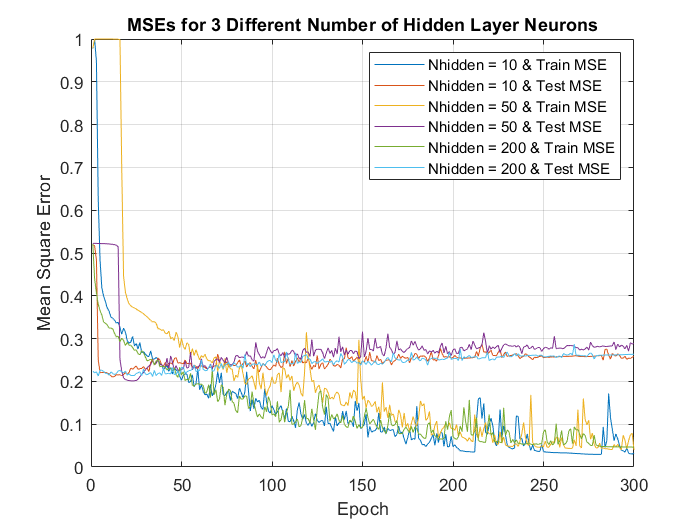


Figure 4: Mean Classification Error (MCE) Plots for Three Different Single Hidden Layer Architecture

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

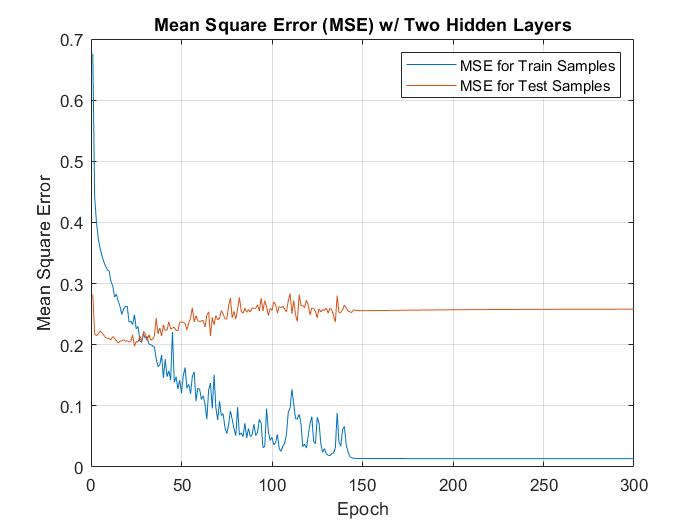


Figure 5: Mean Square Error (MSE) Plots of Two Hidden Layers Architecture

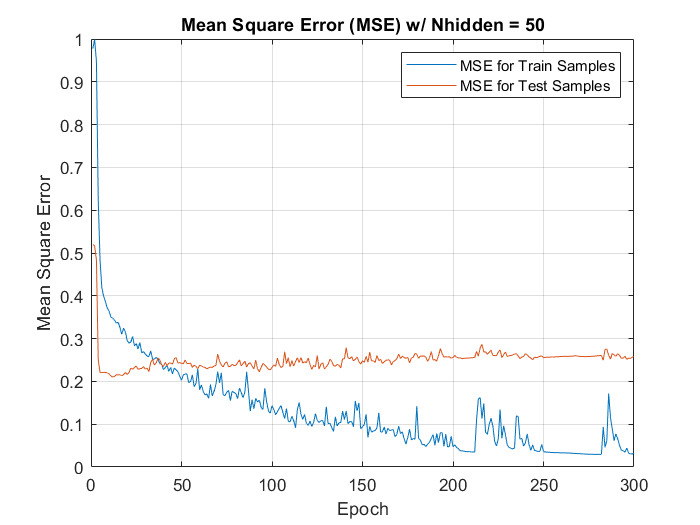


Figure 6: Mean Square Error (MSE) Plots for 50 Hidden Neurons Architecture

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.

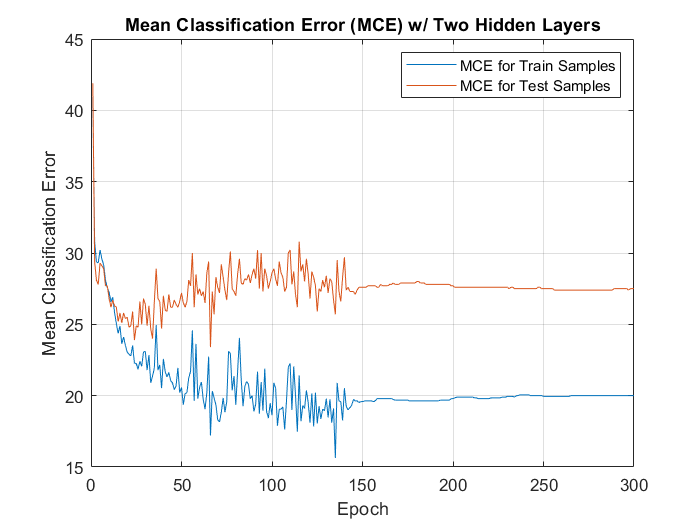


Figure 7: Mean Classification Error (MCE) Plots of Two Hidden Layers Architecture

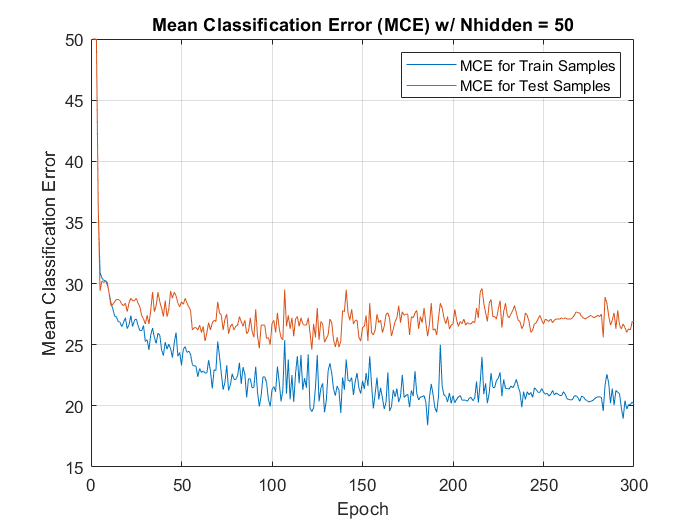


Figure 8: Mean Classification Error (MCE) Plots for 50 Hidden Neurons Architecture

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.

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

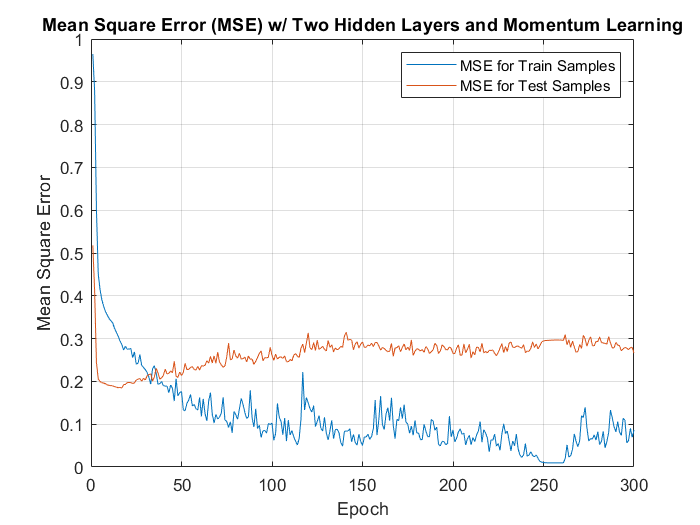


Figure 9: Mean Square Error (MSE) Plots for Two Hidden Layers Network with Momentum Method

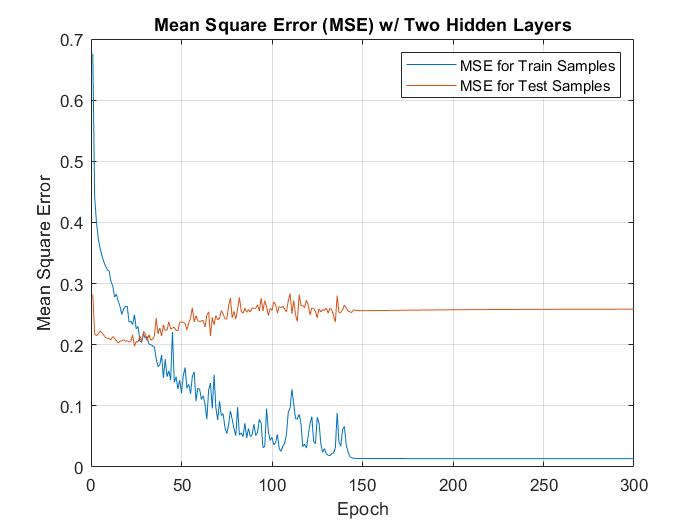


Figure 10: Mean Square Error (MSE) Plots for Two Hidden Layers Network with Momentum Method

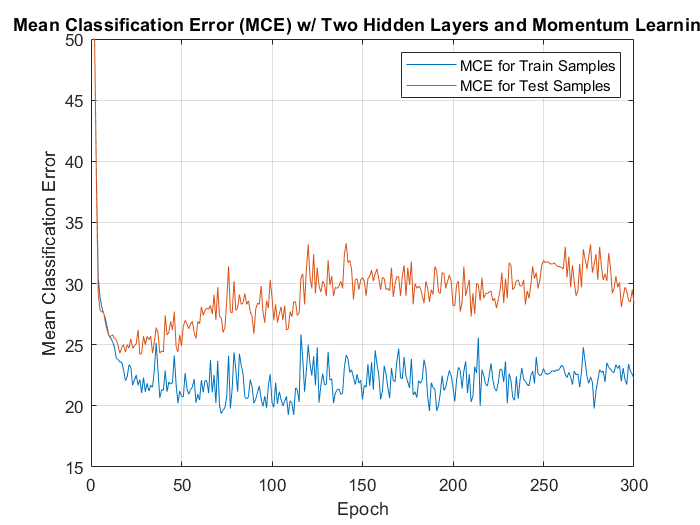


Figure 11: Mean Classification Error (MCE) Plots for Two Hidden Layers Network with Momentum Method

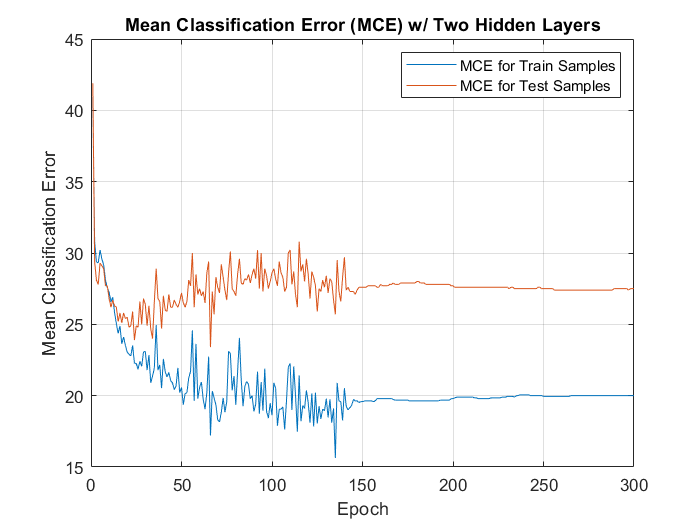


Figure 12: Mean Classification Error (MCE) Plots for Two Hidden Layers Network

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)

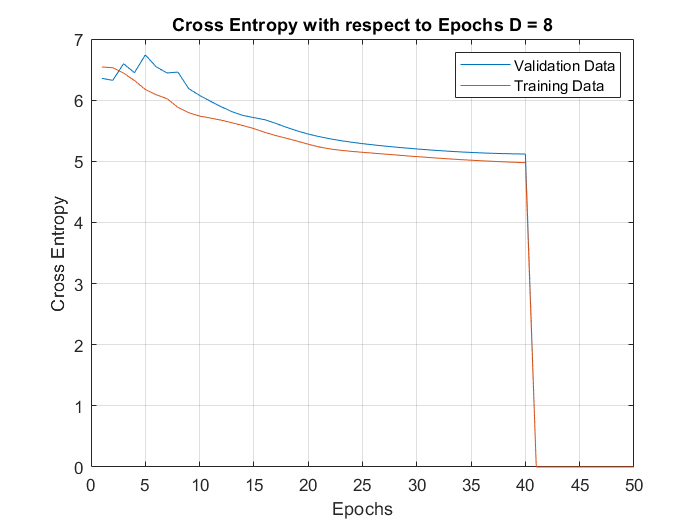


Figure 13: Cross Entropy with respect to Epoch for pair (D,P) = 8,64

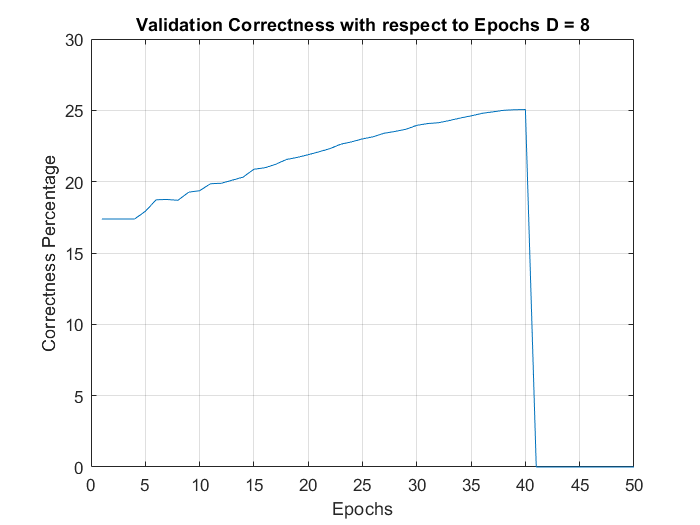


Figure 14: Validation Correctness with respect to Epoch for pair (D,P) = 8,64

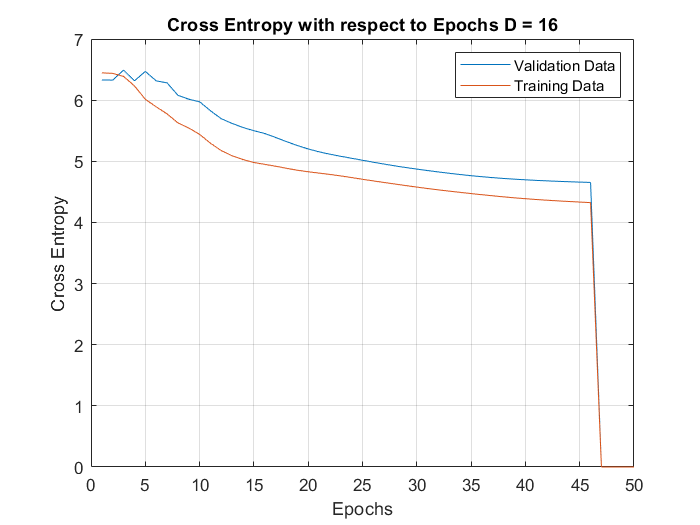


Figure 15: Cross Entropy with respect to Epoch for pair (D,P) = 16,128

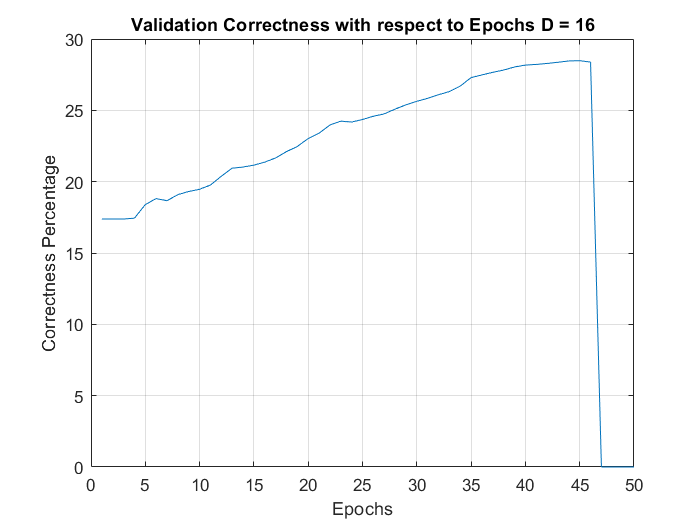


Figure 16: Validation Correctness with respect to Epoch for pair (D,P) = 16,128

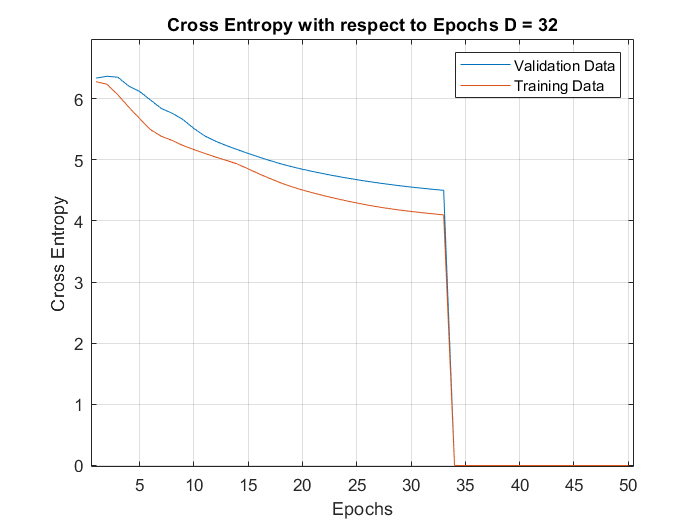


Figure 17: Cross Entropy with respect to Epoch for pair (D,P) = 32,256

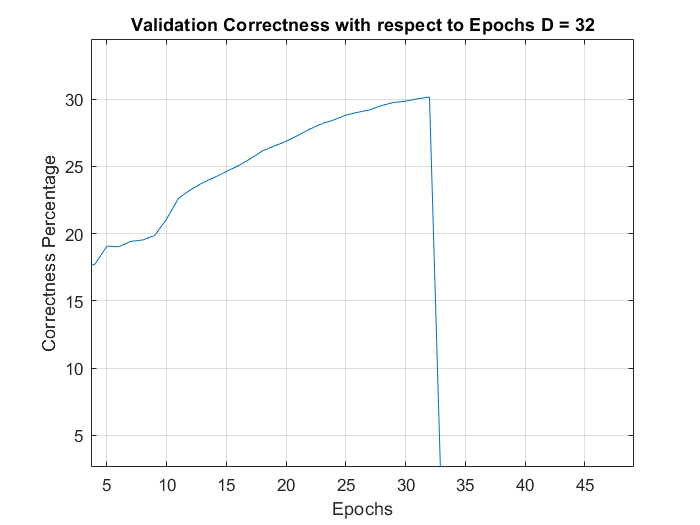


Figure 18: Validation Correctness with respect to Epoch for pair (D,P) = 32,256

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.

1. 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.
2. 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