EEE443 Neural Networks

Homework 1 Report

Name: Oğuz Altan

ID: 21600966

Question 1)

2.1

We can define error as

$error^n = y^n - h(x^n, w) \Rightarrow$ we have $\boxed{\underset{w}{\text{argmin}} \sum_n (error^n)^2 + \beta \sum_i w_i^2}$

$$\tilde{w} = \underset{w}{\text{argmax}}\ P(w|error)$$

Bayes' Rule

$$= \underset{w}{\text{argmax}} \frac{P(error|w)P(w)}{P(error)} = \underset{w}{\text{argmax}}\ P(error|w)P(w)$$

using natural logarithm

$$= \text{argmax}\ \ln(P(error|w)P(w)) = \text{argmax}\left[\ln(P(error|w)) + \ln(P(w))\right]$$

Finding argmax of $\ln(P(error|w)) + \ln(P(w))$ is same as finding argmin of $-\ln(P(error|w)) - \ln(P(w))$

thus, $\tilde{w} = \text{argmin}\left[-\ln(P(error|w)) - \ln(P(W))\right]$

We are given that maximum a posteriori estimate for the network weights are obtained by solving:

$$\text{argmin}\left[\sum_n (error^n)^2 + \beta \sum_i w_i^2\right]$$

Seeing the similarity between the maximum a posteriori estimate and the equation

$$\tilde{w} = \text{argmin}\left[-\ln(P(error|w)) - \ln(P(w))\right]$$

we can say that

$$-\ln(P(w)) \sim \beta \sum_i w_i^2 \Rightarrow -\ln(P(w)) = \gamma \beta \sum_i w_i^2$$

Therefore, $\boxed{P(w) = e^{-\gamma \beta \sum w_i^2}}$ $\qquad \sum w_i^2 = WW^T$

$\boxed{P(w) = e^{-\gamma \beta WW^T}}$ prior probability distribution of the network weights

2.2

Question 2)

In this question, we are to design a neural network with a single hidden layer with four input neurons (with binary inputs) and a single ouput neuron, assuming a step function as the activation function, to implement:

$$(X_1 OR\ NOT\ X_2)\ XOR\ (NOT\ X_3 OR\ NOT\ X_3)$$

which can also be written as

$$(X_1 + \bar{X}_2) \oplus (\bar{X}_3 + \bar{X}_4)$$

This function can be simplified as:

$$(X_1 + \bar{X}_2)\overline{(\bar{X}_3 + \bar{X}_4)} + \overline{(X_1 + \bar{X}_2)}(\bar{X}_3 + \bar{X}_4)$$

where OR operation refers to summation and AND to multiplication. Further simplifications:

$$[(X_1 + \bar{X}_2)(X_3X_4)] + [(\bar{X}_1X_2)(\bar{X}_3 + \bar{X}_4)]$$

$$[X_1X_3X_4 + \bar{X}_2X_3X_4] + [\bar{X}_1X_2\bar{X}_3 + \bar{X}_1X_2\bar{X}_4]$$

a) This simplified logic equation forms 4 neurons of hidden layer (AND operation). Truth tables of each neuron and calculation weights and biases of each neuron are shown below:

Truth Table for $X_1X_3X_4$

X1	X3	X4	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Corresponding activation functions for each row of truth table is given on left, whereas the simplified constraints for step function as activation function are given on the right below:

$f(-\theta) < 0$	$\theta > 0$
$f(w_4 - \theta) < 0$	$\theta > w_4$
$f(w_3 - \theta) < 0$	$\theta > w_3$
$f(w_3 + w_4 - \theta) < 0$	$\theta > w_3 + w_4$
$f(w_1 - \theta) < 0$	$\theta > w_1$
$f(w_1 + w_4 - \theta) < 0$	$\theta > w_1 + w_4$
$f(w_1 + w_3 - \theta) < 0$	$\theta > w_1 + w_3$
$f(w_1 + w_3 + w_4 - \theta) > 0$	$\theta < w_1 + w_3 + w_4$

Weights and biases chosen arbitrarily, complying with the inequalities above are:

$$w_1 = 2 \qquad \theta = 4.5$$
$$w_2 = 0$$
$$w_3 = 2$$
$$w_4 = 2$$

Truth Table for $\overline{X}_2 X_3 X_4$

X2	X3	X4	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Corresponding activation functions for each row of truth table is given on left, whereas the simplified constraints for step function as activation function are given on the right below:

$f(-\theta) < 0$	$\theta > 0$
$f(w_4 - \theta) < 0$	$\theta > w_4$
$f(w_3 - \theta) < 0$	$\theta > w_3$
$f(w_3 + w_4 - \theta) > 0$	$w_3 + w_4 > \theta$
$f(w_2 - \theta) < 0$	$\theta > w_2$
$f(w_2 + w_4 - \theta) < 0$	$\theta > w_2 + w_4$
$f(w_2 + w_3 - \theta) < 0$	$\theta > w_2 + w_3$
$f(w_2 + w_3 + w_4 - \theta) < 0$	$\theta < w_2 + w_3 + w_4$

Weights and biases chosen arbitrarily, complying with the inequalities above are:

$$w_1 = 0$$
$$w_2 = -1$$
$$w_3 = 1$$
$$w_4 = 2$$
$$\theta = 2.25$$

Truth Table for $\overline{X}_1 X_2 \overline{X}_3$

X1	X2	X3	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Corresponding activation functions for each row of truth table is given on left, whereas the simplified constraints for step function as activation function are given on the right below:

$f(-\theta) < 0$	$\theta > 0$
$f(w_3 - \theta) < 0$	$\theta > w_3$
$f(w_2 - \theta) > 0$	$w_2 > \theta$
$f(w_2 + w_3 - \theta) < 0$	$w_2 + w_3 < \theta$
$f(w_1 - \theta) < 0$	$\theta > w_1$
$f(w_1 + w_3 - \theta) < 0$	$\theta > w_1 + w_3$
$f(w_1 + w_2 - \theta) < 0$	$\theta > w_1 + w_2$
$f(w_1 + w_2 + w_3 - \theta) < 0$	$\theta < w_1 + w_2 + w_3$

Weights and biases chosen arbitrarily, complying with the inequalities above are:

$$w_1 = -2$$
$$w_2 = 2$$
$$w_3 = -1$$
$$w_4 = 0$$
$$\theta = 1.75$$

Truth Table for $\overline{X}_1 X_2 \overline{X}_4$

X1	X2	X4	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Corresponding activation functions for each row of truth table is given on left, whereas the simplified constraints for step function as activation function are given on the right below:

$f(-\theta) < 0$	$\theta > 0$
$f(w_4 - \theta) < 0$	$\theta > w_4$
$f(w_2 - \theta) > 0$	$w_2 > \theta$
$f(w_2 + w_4 - \theta) < 0$	$\theta > w_2 + w_4$
$f(w_1 - \theta) < 0$	$\theta > w_1$
$f(w_1 + w_4 - \theta) < 0$	$\theta > w_1 + w_4$
$f(w_1 + w_2 - \theta) < 0$	$\theta > w_1 + w_2$
$f(w_1 + w_2 + w_4 - \theta) < 0$	$\theta > w_1 + w_2 + w_4$

Weights and biases chosen arbitrarily, complying with the inequalities above are:

$$w_1 = -1$$
$$w_2 = 2$$
$$w_3 = 0$$
$$w_4 = -1$$
$$\theta = 1.25$$

In the truth table below for the 4-input OR gate, Neuron 1 is $X_1X_3X_4$, Neuron 2 is $\bar{X}_2X_3X_4$, Neuron 3 is $\bar{X}_1X_2\bar{X}_3$ and Neuron 4 is $\bar{X}_1X_2\bar{X}_4$.

Truth Table for $Neuron\ 1 +\ Neuron\ 2 +\ Neuron$ 3 + Neuron 4

Neuron 1	Neuron 2	Neuron 3	Neuron 4	Output
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Corresponding activation functions for each row of truth table is given on left, whereas the simplified constraints for step function as activation function are given on the right below:

$f(-\theta) < 0$ $\quad\quad \theta > 0$
$f(w_4 - \theta) > 0$ $\quad\quad w_4 > \theta$
$f(w_3 - \theta) > 0$ $\quad\quad w_3 > \theta$
$f(w_3 + w_4 - \theta) > 0$ $\quad\quad \theta > w_3 + w_4$
$f(w_2 - \theta) > 0$ $\quad\quad \theta > w_2$
$f(w_2 + w_4 - \theta) > 0$ $\quad\quad \theta > w_2 + w_4$
$f(w_2 + w_3 - \theta) > 0$ $\quad\quad \theta > w_2 + w_3$
$f(w_2 + w_3 + w_4 - \theta) > 0$ $\quad\quad \theta > w_2 + w_3 + w_4$

$f(w_1 - \theta) > 0$ $\quad\quad w_1 > \theta$
$f(w_1 + w_4 - \theta) > 0$ $\quad\quad w_1 + w_4 > \theta$
$f(w_1 + w_3 - \theta) > 0$ $\quad\quad w_1 + w_3 > \theta$
$f(w_1 + w_3 + w_4 - \theta) > 0$ $\quad\quad w_1 + w_3 + w_4 > \theta$
$f(w_1 + w_2 - \theta) > 0$ $\quad\quad w_1 + w_2 > \theta$
$f(w_1 + w_2 + w_4 - \theta) > 0$ $\quad\quad w_1 + w_2 + w_4 > \theta$
$f(w_1 + w_2 + w_3 - \theta) > 0$ $\quad\quad w_1 + w_2 + w_3 > \theta$
$f(w_1 + w_2 + w_3 + w_4 - \theta) > 0$ $\quad\quad w_1 + w_2 + w_3 + w_4 > \theta$

Weights and biases chosen arbitrarily, complying with the inequalities above are:

$$w_1 = 1$$
$$w_2 = 1$$
$$w_3 = 1$$
$$w_4 = 1$$
$$\theta = 0.75$$

b) The output of the vector calcualted by MATLAB is:

```
The output of the network taking inputs in binary from 0 to 15 is the vector:
    0    0    0    1    1    1    1    0    0    0    0    1    0    0    0    1
```

This output vector is correct, according to the truth table of the $(X_1 + \bar{X}_2) \oplus (\bar{X}_3 + \bar{X}_4)$ showing that the network achieves 100% performance in implementing the desired logic.

c) To minimize the error caused by the Gaussian Noise and to make the network have the most robust decision boundary, I prefer to select biases in the exact middle of lower and higher values of the constraints. The values that are found this way give the optimal performance of the network.

d) We are to generate 400 input samples by first concatenating 25 samples from each input vector and then, we generate a random noise vector of length 2 for each training sample, assuming a zero-mean Gaussian distribution with an std of 0.2.

X_noise
4x400 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0.0214	-0.1928	-0.4023	-0.0086	0.0687	-0.0381	-0.2448	7.9191e-04	0.8246	1.1965	0.8679	1.3464	1.0707	0.8175	1.3333
2	0.0499	0.0852	-0.3786	-0.2320	1.2463	1.0743	1.0667	1.0692	0.2451	0.0192	0.1308	-0.0207	1.1145	1.1054	0.9736
3	0.0220	0.3051	1.0021	0.8378	-0.0492	0.0232	1.3086	0.8269	-0.1032	-0.1079	1.2149	0.9537	-0.0269	0.0467	1.0221
4	-0.1406	0.9679	-0.0755	1.0600	0.0097	1.2080	-0.1400	0.9605	0.0657	1.1128	-0.0387	1.1104	0.1703	0.8278	0.3166
5															
6															

Figure 1: First 15 column of new X matrix with Gaussian noise

The classification performance of the non robust and robust networks on the validation samples is calculated by MATLAB:

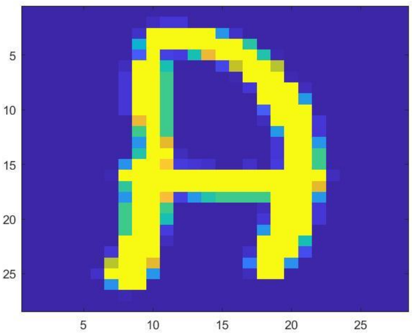
```
Correctness percentage of not robust network is:
    85

Correctness percentage of robust network is:
   90.7500
```

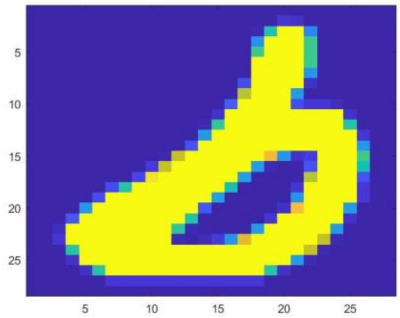
It can be seen that the performance of the network increases as it uses robust configuration of weights and biases. Therefore, using optimal weights and biases that make the network most robust increase the performance of the network.

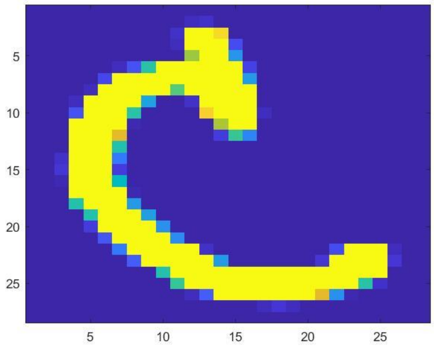
Question 3) 8.1

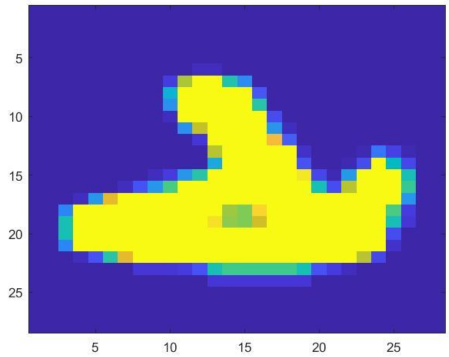
a)

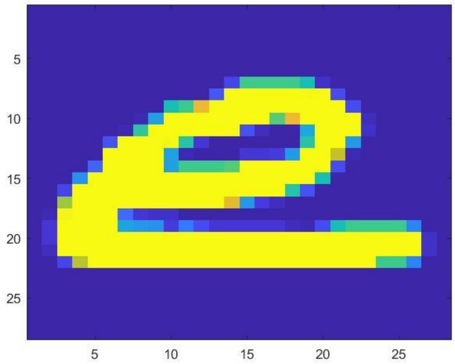
Sample images from each class:

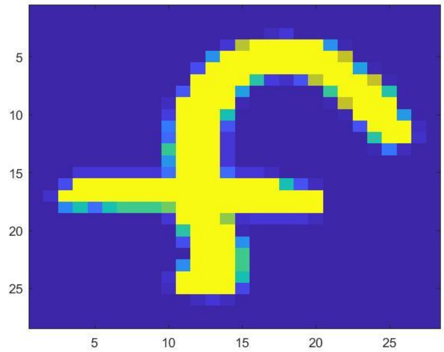


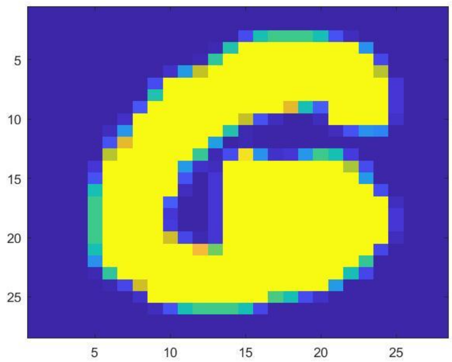


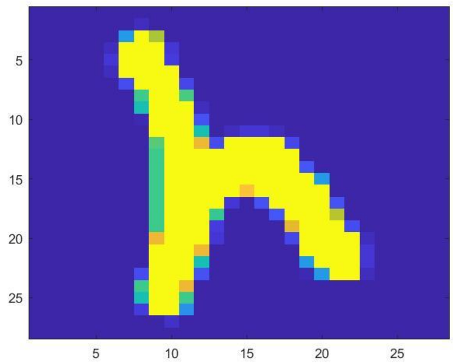


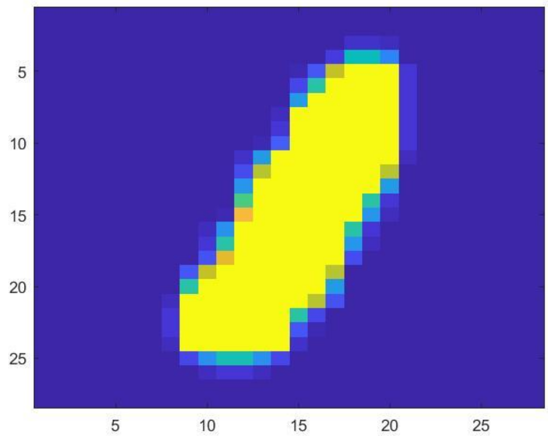


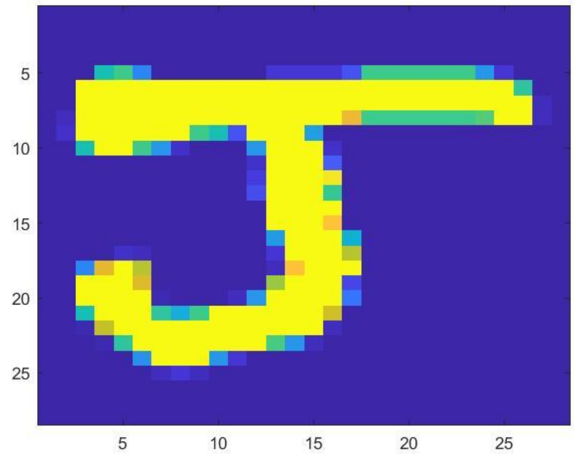


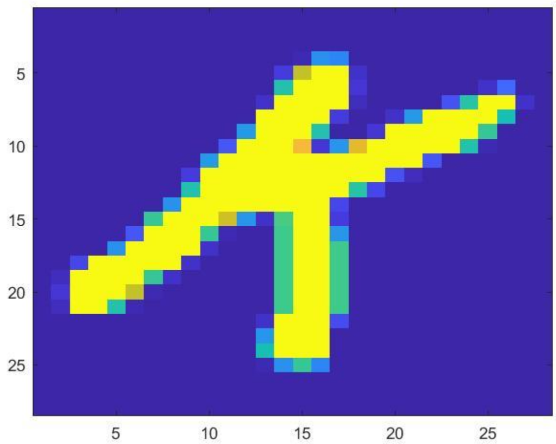


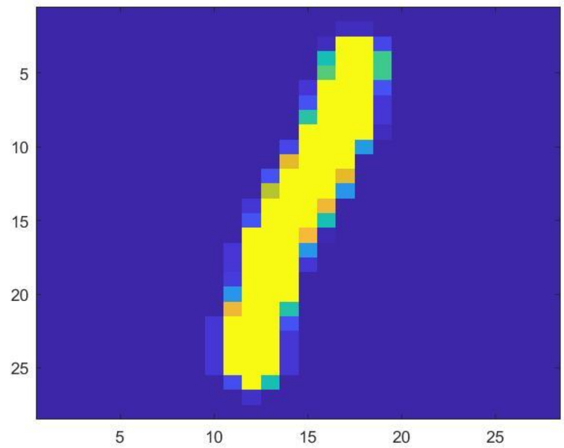


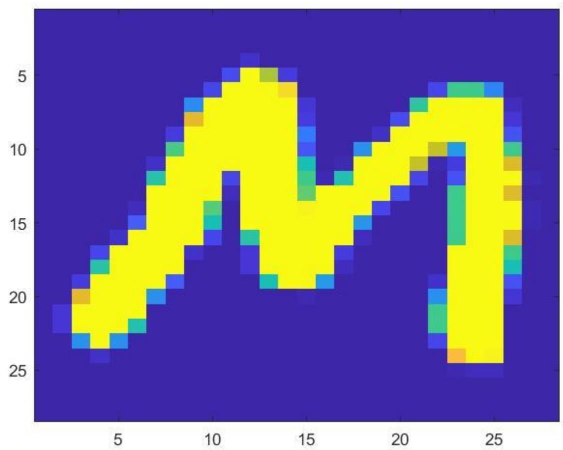


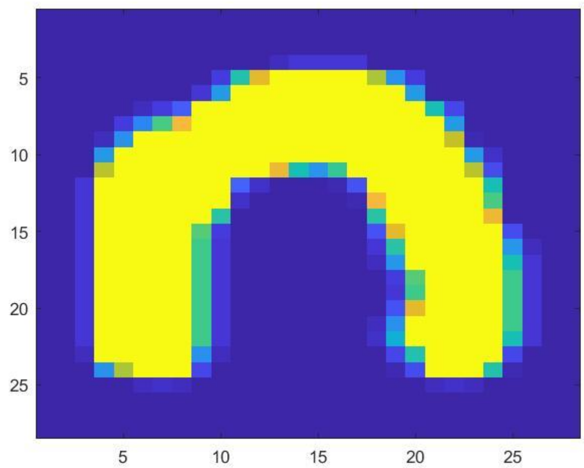

5
10
15
20
25
5
10
15
20
25

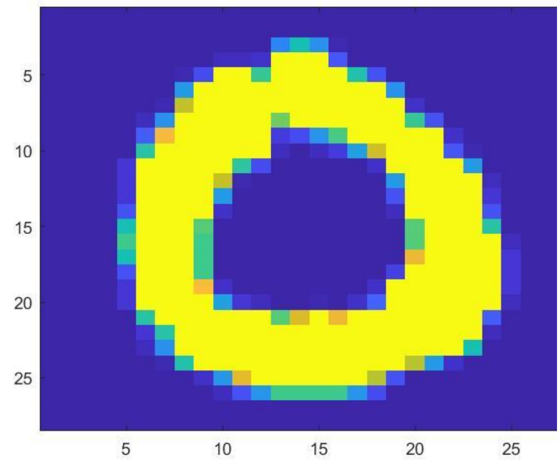

5
10
15
20
25
5
10
15
20
25

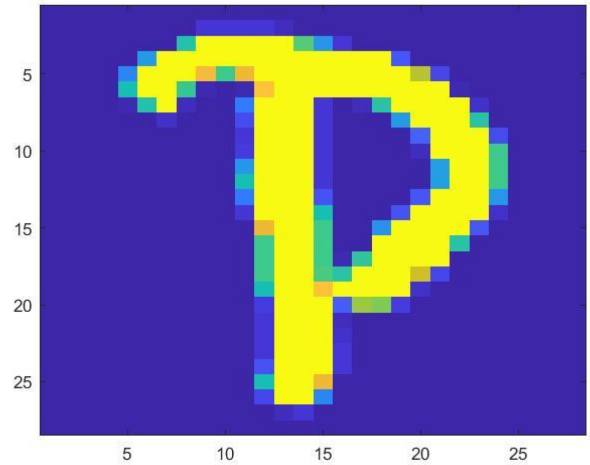

5
10
15
20
25
5
10
15
20
25

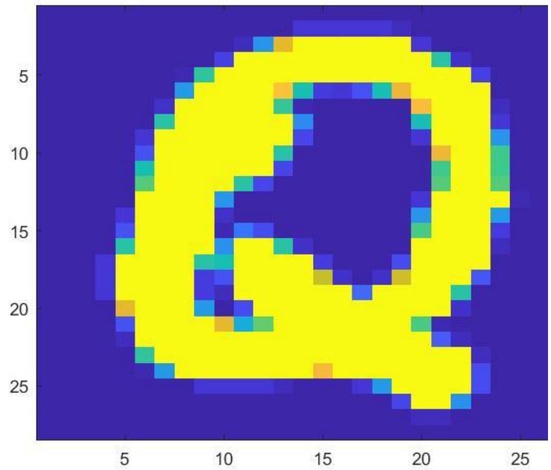

5
10
15
20
25
5
10
15
20
25

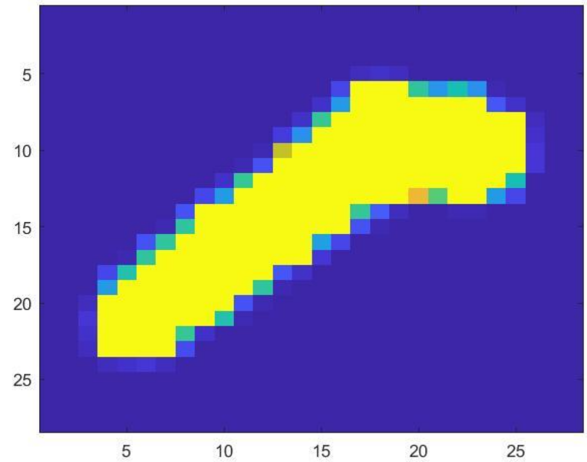

5
10
15
20
25
5
10
15
20
25

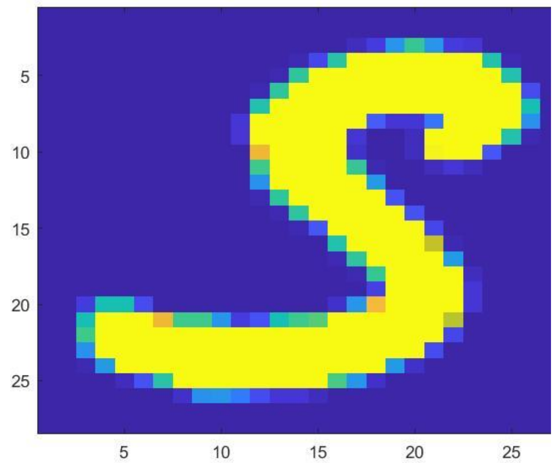

5
10
15
20
25
5
10
15
20
25

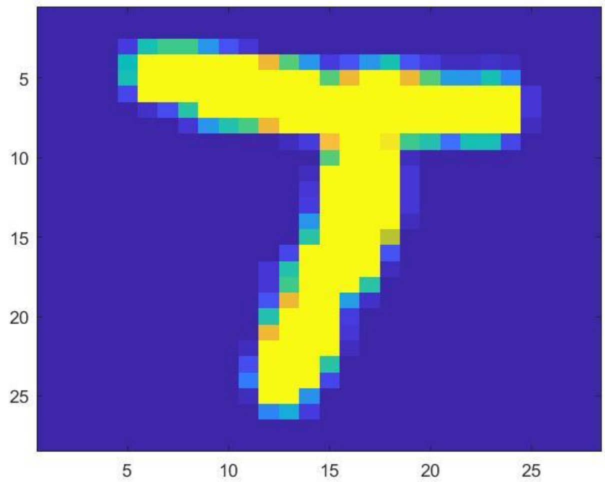

5
10
15
20
25
5 10 15 20 25


5
10
15
20
25
5 10 15 20 25


5
10
15
20
25
5 10 15 20 25


5
10
15
20
25
5 10 15 20 25


5
10
15
20
25
5 10 15 20 25


5
10
15
20
25
5 10 15 20 25

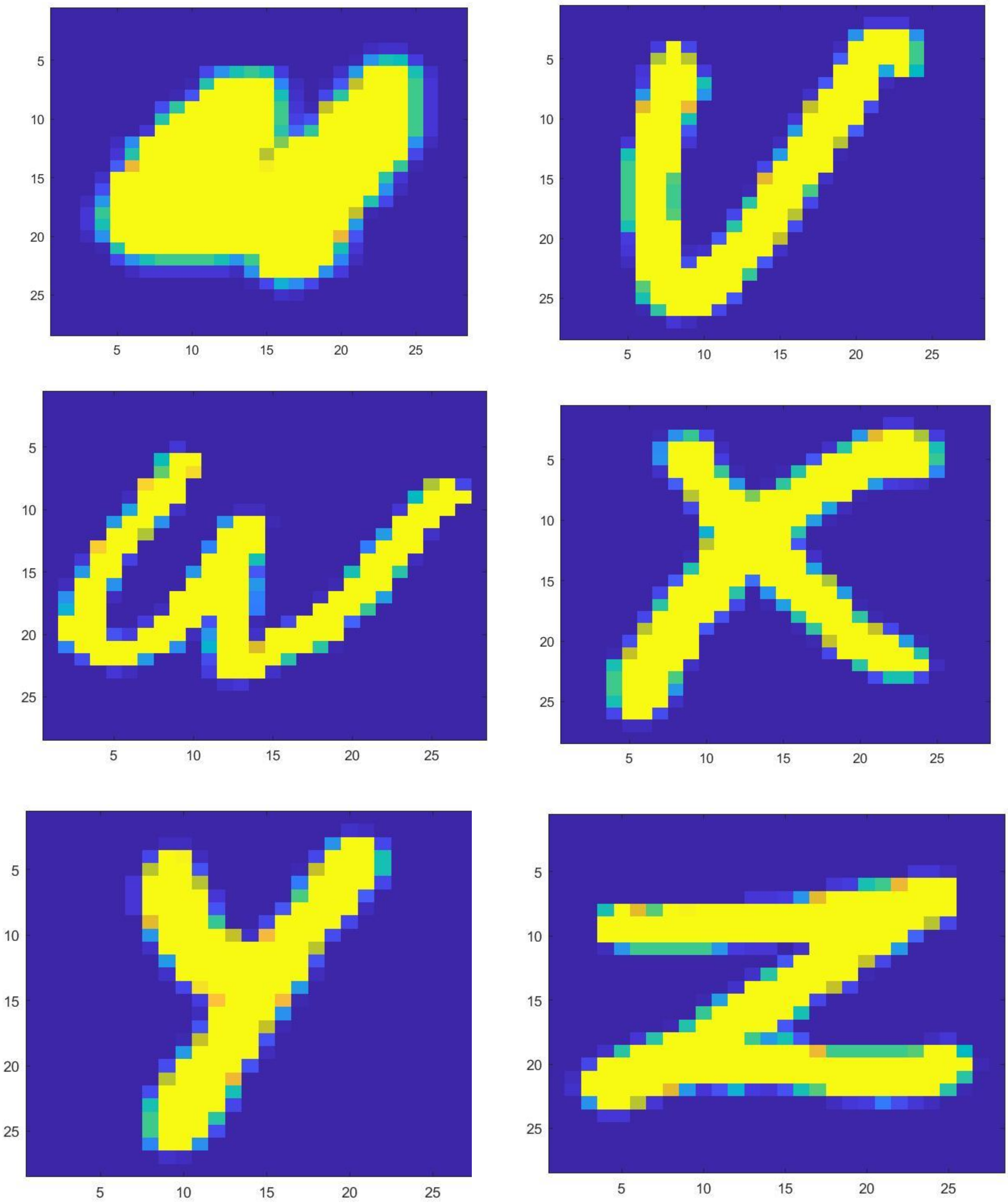
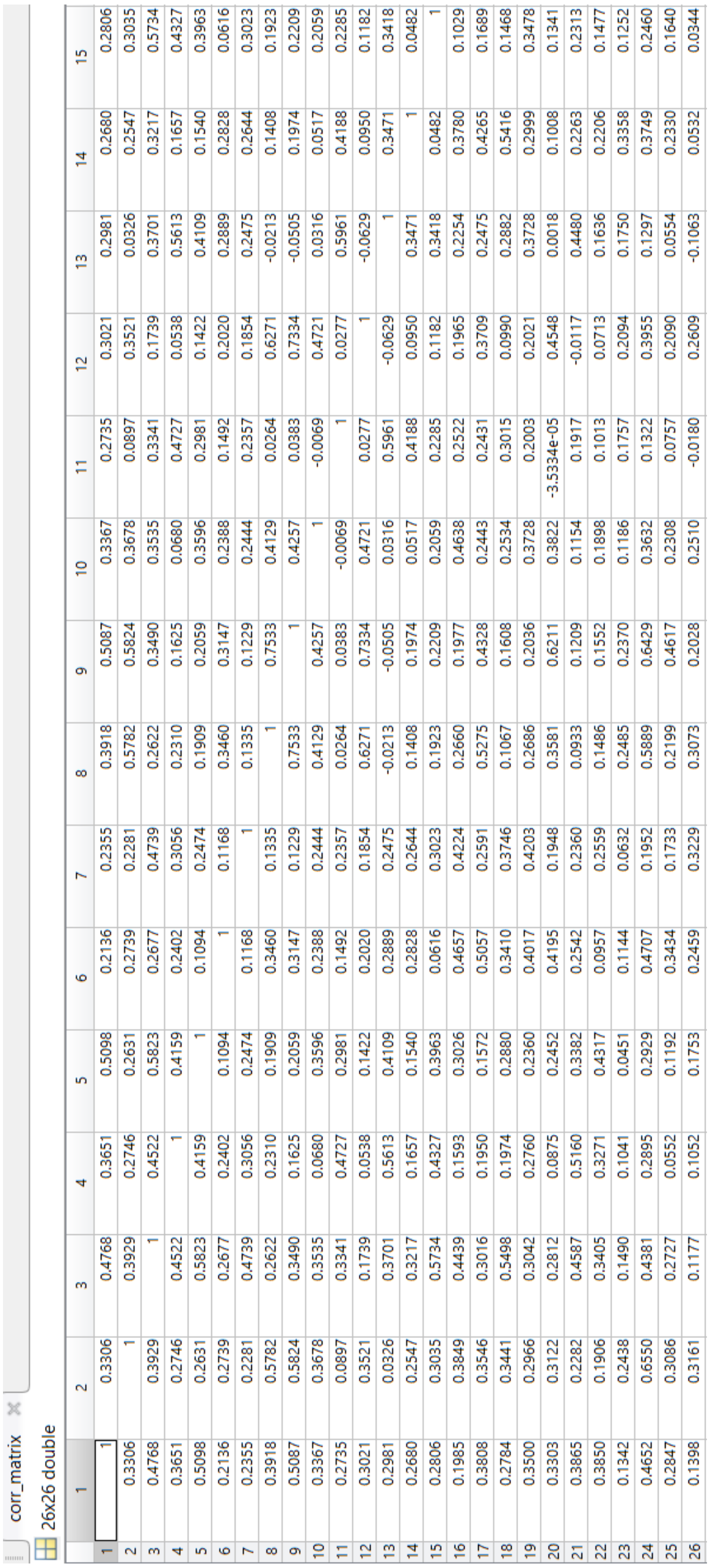


Figure 2: Letter Samples from English Alphabet

The first 15 columns of correlation coefficients matrix can be seen below. Diagonal of this matrix consists of 1's, as the correlation coefficient between two same images is 1.



corr_matrix

26x26 double

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0.3306	0.4768	0.3651	0.5098	0.2136	0.2355	0.3918	0.5087	0.3367	0.2735	0.3021	0.2981	0.2680	0.2806
2	0.3306	1	0.3929	0.2746	0.2631	0.2739	0.2281	0.5782	0.5824	0.3678	0.0897	0.3521	0.0326	0.2547	0.3035
3	0.4768	0.3929	1	0.4522	0.5823	0.2677	0.4739	0.2622	0.3490	0.3535	0.3341	0.1739	0.3701	0.3217	0.5734
4	0.3651	0.2746	0.4522	1	0.4159	0.2402	0.3056	0.2310	0.1625	0.0680	0.4727	0.0538	0.5613	0.1657	0.4327
5	0.5098	0.2631	0.5823	0.4159	1	0.1094	0.2474	0.1909	0.2059	0.3596	0.2981	0.1422	0.4109	0.1540	0.3963
6	0.2136	0.2739	0.2677	0.2402	0.1094	1	0.1168	0.3460	0.3147	0.2388	0.1492	0.2020	0.2889	0.2828	0.0616
7	0.2355	0.2281	0.4739	0.3056	0.2474	0.1168	1	0.1335	0.1229	0.2444	0.2357	0.1854	0.2475	0.2644	0.3023
8	0.3918	0.5782	0.2622	0.2310	0.1909	0.3460	0.1335	1	0.7533	0.4129	0.0264	0.6271	-0.0213	0.1408	0.1923
9	0.5087	0.5824	0.3490	0.1625	0.2059	0.3147	0.1229	0.7533	1	0.4257	0.0383	0.7334	-0.0505	0.1974	0.2209
10	0.3367	0.3678	0.3535	0.0680	0.3596	0.2388	0.2444	0.4129	0.4257	1	-0.0069	0.4721	0.0316	0.0517	0.2059
11	0.2735	0.0897	0.3341	0.4727	0.2981	0.1492	0.2357	0.0264	0.0383	-0.0069	1	0.0277	0.5961	0.4188	0.2285
12	0.3021	0.3521	0.1739	0.0538	0.1422	0.2020	0.1854	0.6271	0.7334	0.4721	0.0277	1	-0.0629	0.0950	0.1182
13	0.2981	0.0326	0.3701	0.5613	0.4109	0.2889	0.2475	-0.0213	-0.0505	0.0316	0.5961	-0.0629	1	0.3471	0.3418
14	0.2680	0.2547	0.3217	0.1657	0.1540	0.2828	0.2644	0.1408	0.1974	0.0517	0.4188	0.0950	0.3471	1	0.0482
15	0.2806	0.3035	0.5734	0.4327	0.3963	0.0616	0.3023	0.1923	0.2209	0.2059	0.2285	0.1182	0.3418	0.0482	1
16	0.1985	0.3849	0.4439	0.1593	0.3026	0.4657	0.4224	0.2660	0.1977	0.4638	0.2522	0.1965	0.2254	0.3780	0.1029
17	0.3808	0.3546	0.3016	0.1950	0.1572	0.5057	0.2591	0.5275	0.4328	0.2443	0.2431	0.3709	0.2475	0.4265	0.1689
18	0.2784	0.3441	0.5498	0.1974	0.2880	0.3410	0.3746	0.1067	0.1608	0.2534	0.3015	0.0990	0.2882	0.5416	0.1468
19	0.3500	0.2966	0.3042	0.2760	0.2360	0.4017	0.4203	0.2686	0.2036	0.3728	0.2003	0.2021	0.3728	0.2999	0.3478
20	0.3303	0.3122	0.2812	0.0875	0.2452	0.4195	0.1948	0.3581	0.6211	0.3822	-3.5334e-05	0.4548	0.0018	0.1008	0.1341
21	0.3865	0.2282	0.4587	0.5160	0.3382	0.2542	0.2360	0.0933	0.1209	0.1154	0.1917	-0.0117	0.4480	0.2263	0.2313
22	0.3850	0.1906	0.3405	0.3271	0.4317	0.0957	0.2559	0.1486	0.1552	0.1898	0.1013	0.0713	0.1636	0.2206	0.1477
23	0.1342	0.2438	0.1490	0.1041	0.0451	0.1144	0.0632	0.2485	0.2370	0.1186	0.1757	0.2094	0.1750	0.3358	0.1252
24	0.4652	0.6550	0.4381	0.2895	0.2929	0.4707	0.1952	0.5889	0.6429	0.3632	0.1322	0.3955	0.1297	0.3749	0.2460
25	0.2847	0.3086	0.2727	0.0552	0.1192	0.3434	0.1733	0.2199	0.4617	0.2308	0.0757	0.2090	0.0554	0.2330	0.1640
26	0.1398	0.3161	0.1177	0.1052	0.1753	0.2459	0.3229	0.3073	0.2028	0.2510	-0.0180	0.2609	-0.1063	0.0532	0.0344

Figure 3: Correlation Matrix

The variability for within-class is lower than for across-class because the images in the same class are more similar than two images in different classes. This situation can also be observable in the correlation coefficient matrix below. In conclusion, for two images that are in the same class, their correlation is stronger than for two images that are in different class (across- class).

The correlation coefficient matrix can be visualised as below:

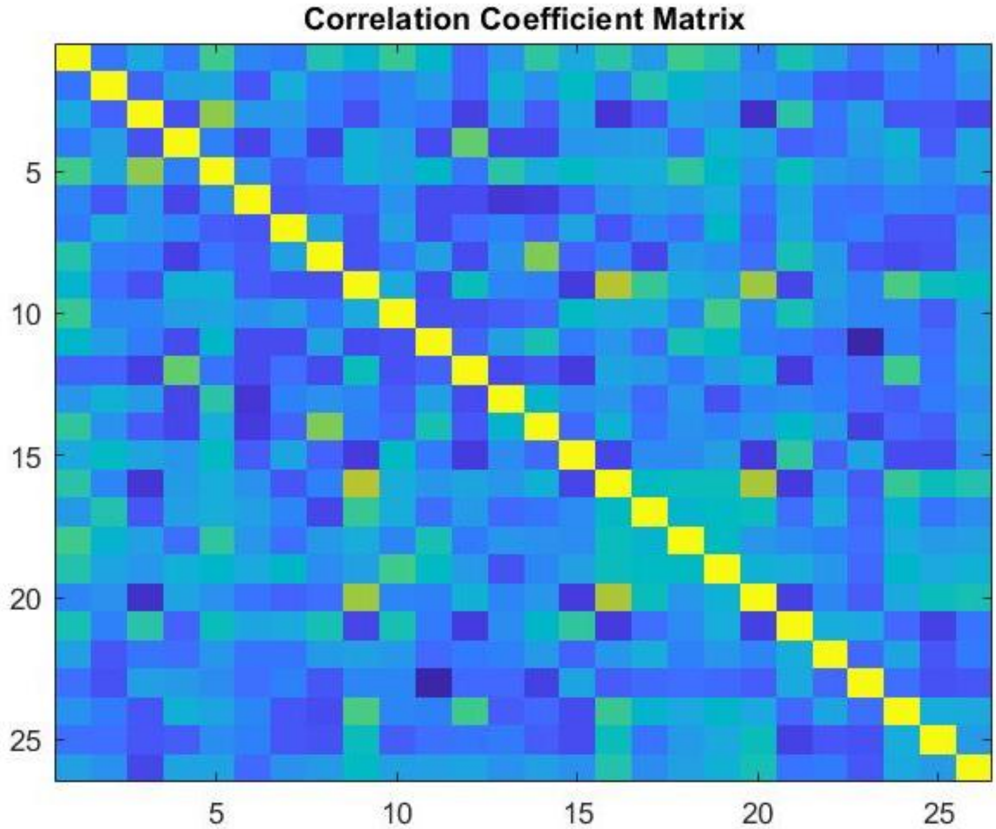
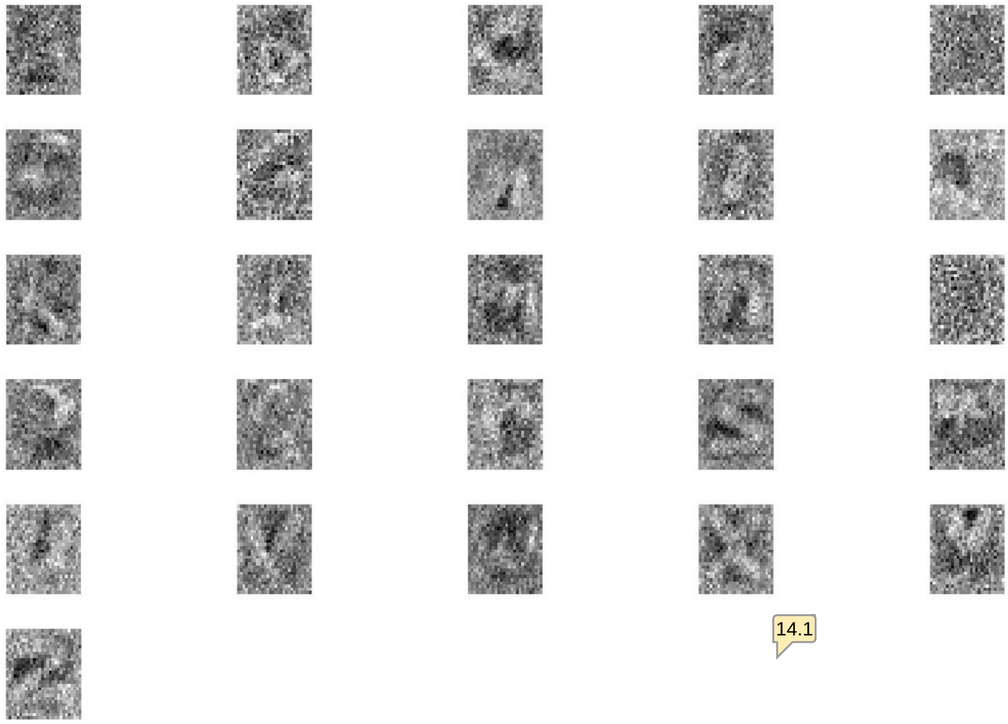


Figure 4: Correlation Matrix within Image Representation

b) Optimal learning rate is found to be around 0.1. The final network weights for each digit as a separate image can be seen below:



14.1

Figure 5: Final Network Weights for Each Digit as Separate Images

It can be seen that network predicts the letters in the alphabet with a moderate performance and accuracy. This is due to single layer perceptron architecture of network, which is more primitive compared to the multi layer neural networks, which maye give higher performance. The results are not perfect but letters can be read.

c)

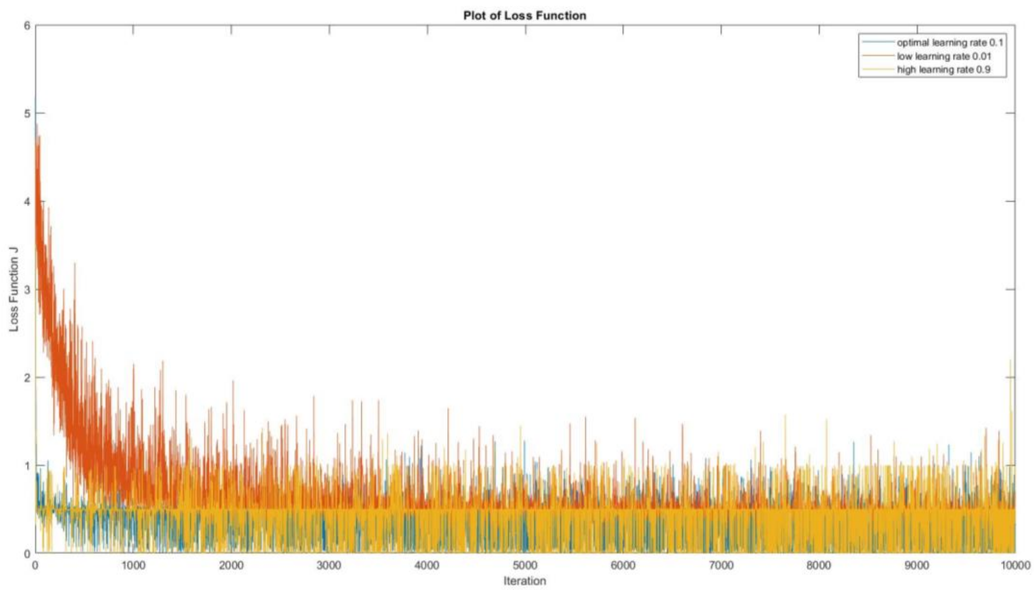


Figure 6: Loss Function for 3 Different Learning

As it can be seen from the plot of the loss function for optimal learning rate, low learning rate and high learning rate, the most optimal minimization of loss function occurs for optimal learning rate.

d) The performance of the trained networks using all samples in the test data are evaluated using MATLAB and results for three different learning rates can be seen below:

```
The correctness for learning rate nu 0.1 is = 57.461538
The correctness for low learning rate nu 0.001 is = 14.230769
The correctness for high learning rate nu 0.9 is = 32.307692
```

APPENDIX

```
function oguz_altan_21600966_hw1(question)
clc
close all

switch question
    case '1'
        disp('1')
        %% question 1 code goes here
        disp('The answer of this question is analytical
and can be found on the report.');

    case '2'
        disp('2')
        %% question 2 code goes here

        %hidden layer (AND gate) input weights and bias
        w1 = [2 0 2 2];
        w2 = [0 -1 1 2];
        w3 = [-2 2 -1 0];
        w4 = [-1 2 0 -1];
        theta = [4.5 2.25 1.75 1.25];
        W_hidden = [w1;w2;w3;w4];

        disp('hidden layer input weights matrix is: ');
        disp(W_hidden);
        disp('hidden layer bias weight vector is: ');
        disp(theta);

        %output layer (OR gate) input weigths and bias
        W_or = [1 1 1 1];
        theta_or = 0.75;

        disp('output layer input weights vector is: ')
        disp(W_or);
        disp('output layer bias weight is: ');
        disp(theta_or);
```

```
        bina_input = decimalToBinaryVector(0:15); %creates
matrix of binary equivalents of 0 to 15
        disp('Matrix of binary equivalents of decimal 0 to
15: ');
        disp(bina_input);

        %calculates the network output: hidden layer takes
weights and theta and uses
        %weighted sum minus bias and puts the result in
activation
        %function. Output layer makes similar calculation
using OR layer
        %weights, which are outputs of hidden layer and
again puts the result into activation function. The
        %result is nn_out
        nn_out =
unitStep(W_or*unitStep(W_hidden*bina_input'-theta')-
theta_or);
        disp('The output of the network taking inputs in
binary from 0 to 15 is the vector: ')
        disp(nn_out)

        %generate 400 input samples by concatenating 25
samples from each input vector
        %and adds gaussian noise with std of 0.2
        std = 0.2;
        new_X = repmat(bina_input',1,25);
        noise = std*randn(4,400);
        X_noise = new_X + noise;
        disp('New X matrix with Gaussian noise:');
        disp(X_noise);

        %not robust network
        %calculates network output using the same way that
is explained for
        %nn_out
        nn_out_tt =
unitStep(W_or*unitStep(W_hidden*new_X-theta')-theta_or);
        nn_out_noised =
unitStep(W_or*unitStep(W_hidden*X_noise-theta')-theta_or);
        notrobustcorrect = sum(nn_out_tt ==
nn_out_noised)/4; %compares the original vs noised network
        disp('Correctness percentage of not robust network
is: ')
        disp(notrobustcorrect);
```

```
        %robust network
        robust_hidden_theta = [5 2.5 1.5 1.5];
        robust_theta_or = 0.5;

        nn_out_tt_robust =
unitStep(W_or*unitStep(W_hidden*new_X-
robust_hidden_theta')-robust_theta_or);
        nn_out_noised_robust =
unitStep(W_or*unitStep(W_hidden*X_noise-
robust_hidden_theta')-robust_theta_or);

        %compares the original vs noised network and
divides the number of times
        %that they are equal by 4 to find the correctness
percentage
        robustcorrect = sum(nn_out_tt_robust ==
nn_out_noised_robust)/4;

        disp('Correctness percentage of robust network is:
')
        disp(robustcorrect);

    case '3'
        disp('3')
        %% question 3 code goes here
        load('assign1_data1.mat')

        %part a
        %find random indexes for each class
        rand_vector = [];
        for i = 1:26
            rand_vector = [rand_vector randi([(i-1)*200
i*200])];
        end
        rand_vector;

        %print sample images for each class
        for i = 1:26
            figure;
            image(trainims(:,:,rand_vector(i)));
        end
        figure;
        image(trainims(:,:,randi([5000 5200])))

        %construct correlation matrix for each pair of
images
        bina_input = [];
```

```
        for i = 1:26
            matt = trainims(:,:,rand_vector(i));
            col_vec = matt(:); %reshapes the image matrix
into a column vector
            bina_input = [bina_input col_vec];
        end

        bina_input = double(bina_input);
        corr_matrix = corrcoef(bina_input);
        disp('The correlation coefficient matrix: ');
        disp(corr_matrix);

        imagesc(corr_matrix);
        title('Correlation Coefficient Matrix');

        %using one hot encoding for labels to make them
appropriate
        %to use in error calculation
        onehot = zeros(26,5200);
        for i = 1:5200
            onehot(trainlbls(i),i) = 1;
        end

        %constructing weights and learning rate (nu)
matrices
        std = 0.1;
        W = 0.1*randn(785,26); %Gaussian noise weights
        nu_opt = 0.1; %learning rate
        W_low = W;
        W_high = W;
        loss_vec = [];

        %10000 iterations and updates using gradient
descent
        for i = 1:10000
            rand_image_index = randi([1 5200]); %randomly
selects an index in [1 5200]
            trainims_in_matrix =
trainims(:,:,rand_image_index); %creates image matrix from
trainims dataset
            x = double(trainims_in_matrix(:)); %casting
uint8 to double
            x = [x ;-1]; %adding bias to the weights
matrix
            x = x/255; %normalization
            v = W'*x ; %weighted sum
```

```
            y = sigmoid(v); %applying activation function
to v
            der_act = sigmoid(v) .* (1 - sigmoid(v));
%derivative of the sigmoid function
            error = (onehot(:,rand_image_index)-y);
%difference between the desired and realized output of the
network
            grad = x * (error .* der_act)'; %gradient for
gradient descent
            W = W + nu_opt * grad; %updating weights
matrix
            loss = 1/2 * error' * error; %Mean Square
Error
            loss_vec = [loss_vec loss]; %storing loss for
each iteration into a vector
        end

        figure;
        for k = 1:26
            subplot(6,5,k),imshow(reshape(W(2:785,k),[28
28]), [min(W(2:785,k)) max(W(2:785,k))]);
        end

        %testing the trained network using test images
        W_test = W;
        correct = 0;
        output_tot = [];
        max_value = [];

        for i = 1:1300
            rand_image_index = i; %randomly selects an
index in [1 5200]
            testims_in_matrix =
testims(:,:,rand_image_index);
            x_test = double(testims_in_matrix(:));
%casting uint8 to double
            x_test = [x_test;-1]; %adding bias
            x_test = x_test/255;
            output_test = sigmoid(W_test.'*x_test);
            [max_value, label_index] = max(output_test);
            if (testlbls(i) == label_index)
                correct = correct+1;
            end
        end
        fprintf('The correctness for optimal learning rate
nu 0.1 is = %f\n',correct/1300*100);
```

```
        %%%%%%%%%%%%%%%%%% lower learning rate nu
        nu_low = 0.001;
        loss_vec_low = [];
        for i = 1:10000
            rand_image_index = randi([1 5200]); %randomly
selects an index in [1 5200]
            trainims_in_matrix =
trainims(:,:,rand_image_index); %creates image matrix from
trainims dataset
            x = double(trainims_in_matrix(:)); %casting
uint8 to double
            x = [x ;-1]; %adding bias to the weights
matrix
            x = x/255; %normalization
            v = W_low'*x ; %weighted sum
            y = sigmoid(v); %applying activation function
to v
            der_act = sigmoid(v) .* (1 - sigmoid(v));
%derivative of the sigmoid function
            error = (onehot(:,rand_image_index)-y);
%difference between the desired and realized output of the
network
            grad = x * (error .* der_act)'; %gradient for
gradient descent
            W_low = W_low + nu_low * grad; %updating
weights matrix
            loss_low = 1/2 * error' * error; %Mean
Squarred Error
            loss_vec_low = [loss_vec_low loss_low];
%storing loss for each iteration into a vector
        end

        W_test_low = W_low;
        correct_low = 0;
        output_tot_low = [];
        max_value_low = [];
        for i = 1:1300
            rand_image_index = i;
            testims_in_matrix =
testims(:,:,rand_image_index);
            x_test = double(testims_in_matrix(:));
            x_test = [x_test;-1];
            x_test = x_test/255;
            output_test = sigmoid(W_test_low.'*x_test);
            [max_value_low, label_index] =
max(output_test);
            if (testlbls(i) == label_index)
```

```
                correct_low = correct_low+1;
            end
        end
        fprintf('The correctness for low learning rate nu 
0.001 is = %f\n',correct_low/1300*100);


        %%%%%%%%%%%%%%%%%% higher learning rate nu
        nu_high = 0.9;
        loss_vec_high = [];
        for i = 1:10000
            rand_image_index = randi([1 5200]); %randomly 
selects an index in [1 5200]
            trainims_in_matrix = 
trainims(:,:,rand_image_index); %creates image matrix from 
trainims dataset
            x = double(trainims_in_matrix(:)); %casting 
uint8 to double
            x = [x ;-1]; %adding bias to the weights 
matrix
            x = x/255; %normalization
            v = W_high'*x ; %weighted sum
            y = sigmoid(v); %applying activation function 
to v
            der_act = sigmoid(v) .* (1 - sigmoid(v)); 
%derivative of the sigmoid function
            error = (onehot(:,rand_image_index)-y); 
%difference between the desired and realized output of the 
network
            grad = x * (error .* der_act)'; %gradient for 
gradient descent
            W_high = W_high + nu_high * grad; %updating 
weights matrix
            loss_high = 1/2 * error' * error; %Mean 
Squarred Error
            loss_vec_high = [loss_vec_high loss_high]; 
%storing loss for each iteration into a vector
        end

        W_test_high= W_high;
        correct_high = 0;
        output_tot_high = [];
        max_value_high = [];

        for i = 1:1300
            rand_image_index = i;
```

```
            testims_in_matrix =
testims(:,:,rand_image_index);
            x_test = double(testims_in_matrix(:));
            x_test = [x_test;-1];
            x_test = x_test/255;
            output_test = sigmoid(W_test_high.'*x_test);
            [max_value_high, label_index] =
max(output_test);
            if (testlbls(i) == label_index)
                correct_high = correct_high+1;
            end
        end
        fprintf('The correctness for high learning rate nu
0.9 is = %f\n',correct_high/1300*100);

        %plotting loss functions for three different
learning rates on the
        %same figure to compare easily
        iter = [1:10000];
        figure;
        plot(iter,loss_vec);
        hold on;
        plot(iter,loss_vec_low);
        plot(iter,loss_vec_high);
        title('Plot of Loss Function');
        legend('optimal learning rate 0.1', 'low learning
rate 0.01', 'high learning rate 0.9');
        xlabel('Iteration');
        ylabel('Loss Function J');

end
end

function outStep = unitStep(t)
outStep = t >= 0;
end

function sigmoidOut = sigmoid(v)
sigmoidOut = 1./(1+exp(-v));
end
```

Index of comments

2.1 23/25

2.2 -2
Form is correct, but not normalized.

3.1 26/30

3.2 10/10

7.1 5/5

7.2 6/10

7.3 not very clear

7.4 5/5

8.1 35/45

13.1 -2
within class variability

13.2 how did you calculate -3

14.1 no sgd equations -5