ASSIGNMENT REPORT
GROUP 06

SCS 3120 – Machine Learning and Neural Networks

Group Members – Group 06

	Name	**Registration no**	**Signature**
1	Kumarasiri B.A.N.P	2013CS061	
2	Radeeshani M.D.P.	2013CS095	
3	Chandrathilake K.A.A.P	2013IS006	
4	Jayawardena U.U.A	2013IS025	
5	Thathsaranee B.L	2013IS057	

Part 1- Supervised Network

In the first question we have been asked to build and train a supervised neural network to classify 4 English letters (R, Q, K, W) and put them into correct classes. We implemented the network using MATLAB. Usage of a supervised neural network for this type of questions is obvious since we know all the output classes and there are exactly four. Hence we created a Feed Forward Back propagation Neural Network and trained it to deliver the target output. Following are the general steps in Supervised Training.

1. Weights are randomly set
2. Target is identified
3. Input is applied to the network
4. Actual output of the network is calculated
5. Error = Target - Actual Output is calculated
6. Weights are updated until the error is minimized

Delta Rule

$$\Delta W = \eta \, \partial x$$

ΔW = Wnew – W old
η- Learning Rate Parameter
∂- Error (= Target ~ Network output)
x - Input

In a Feed Forward Back propagation network, net value is calculated in forward direction and the error is propagated backwards from the output nodes to the input nodes. This procedure is done until the error becomes minimum.

Input for the Neural Network

In the question, we have been told to prepare 30 unique images of each character in 12 x 10 dimension. From those 30, we have to use 22 images from each character for the training and remaining 8 images for testing purposes. Altogether we have 88 images for training and 24 images for testing. Our data set consist of handwritten letters which are different from each other.

Pre-processing

We need to take all our input images to one standard format so that training process will be a lot easier. So all the images will be pre-processed before using them to training or testing. And also pre-processing part is really important because it eliminates the useless pixels.

We have implemented pre-processing as a separate function. Function will be executed in the following order.

1. Converting RGB of the image to a gray scale image
   ```
   grayImage = rgb2gray(Img)
   ```
2. Convert the gray scale image into a binary image based on a certain threshold value
   ```
   I=im2bw(grayImage,graythresh(grayImage))
   ```
3. Perform the image dilation on the image
   ```
   se=strel('square',1)
   I=imdilate(~I,se)
   ```
4. Resize all the pre-processed images to the given dimension (12 x 10)
   ```
   processImages=imresize(I,[10 12])
   ```

After pre-processing we saved those 120 images in a different folder.

Targets for the Network

Since we need to identify 4 different outputs, we used 4 digit binary patterns to identify them. Output layer consists of 4 nodes and binary patters for targets are as follows.

1. 0100 binary pattern for letter K
2. 0001 binary pattern for letter Q
3. 0010 binary pattern for letter R
4. 1000 binary pattern for letter W

Creating the Neural Network

MATLAB command 'newff()' is used to create a Feed Forward Back propagation network.

```
net = newff(RInput,RTarget,[20 20],{'tansig', 'tansig'})
```

Here 'RInput' is the training dataset and 'RTarget' is the targets we have mentioned above. We have used 2 hidden layers each with 20 nodes. Tan sigmoid function is used as the transfer function.

Train the network

```
net = train(net,RInput,RTarget)
```

Network which returned by the newff function will be provided to this function along with input image set and target image set.

Test the network

```
TestingOutput=sim(net, TestInput)
```

We provide test image dataset to this function as inputs and they will process according the network we have built. We can check the final output from TestingOutput.

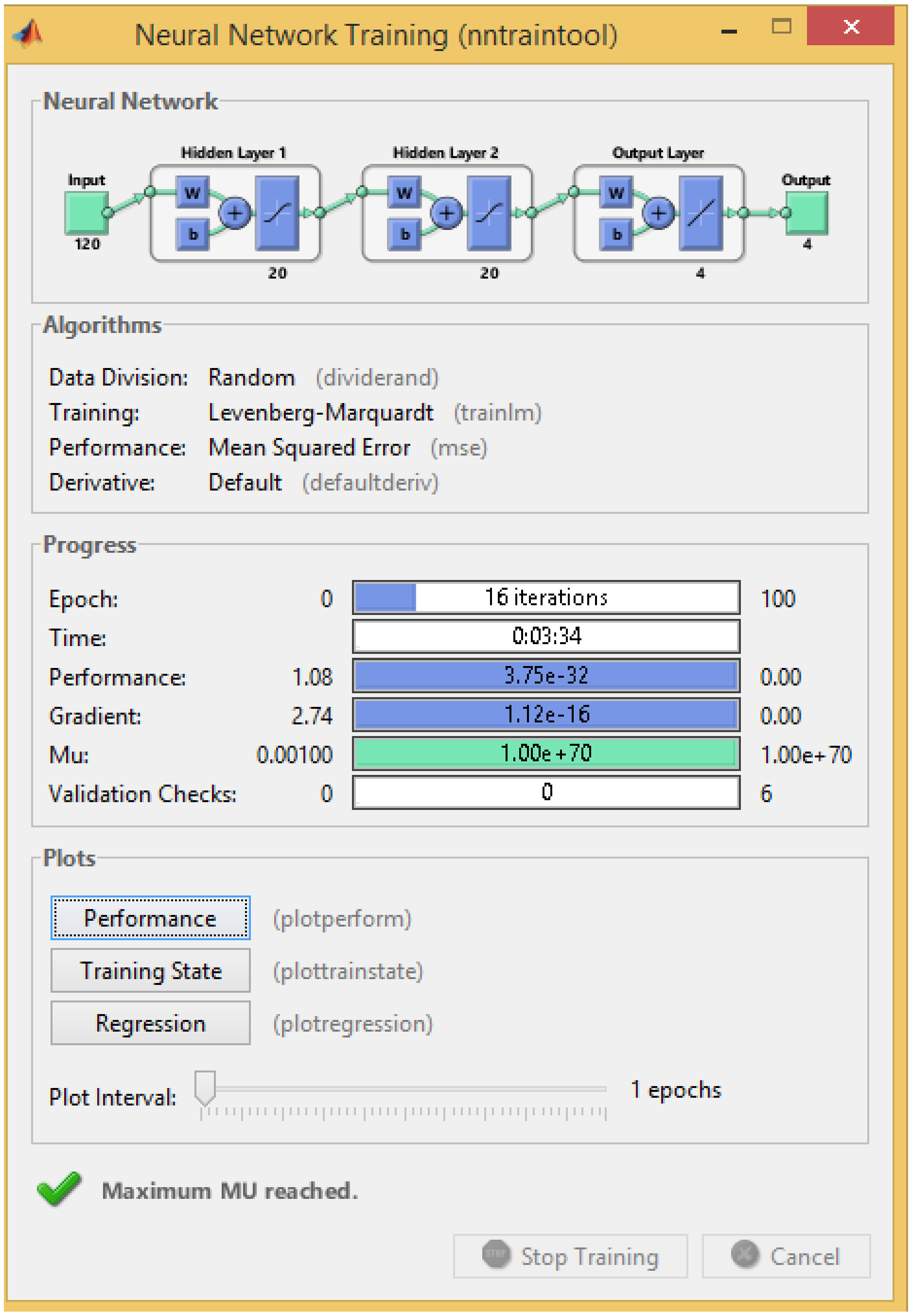
Sample input images

We used letters which are totally different from each other. All letters are written using a pencil of image editor. (Not different fonts)

After Pre-processing

Network Details

Train the Neural Network



Algorithms

Data Division: Random (dividerand)
Training: Levenberg-Marquardt (trainlm)
Performance: Mean Squared Error (mse)
Derivative: Default (defaultderiv)

Progress

Epoch:	0	16 iterations	100
Time:		0:03:34	
Performance:	1.08	3.75e-32	0.00
Gradient:	2.74	1.12e-16	0.00
Mu:	0.00100	1.00e+70	1.00e+70
Validation Checks:	0	0	6

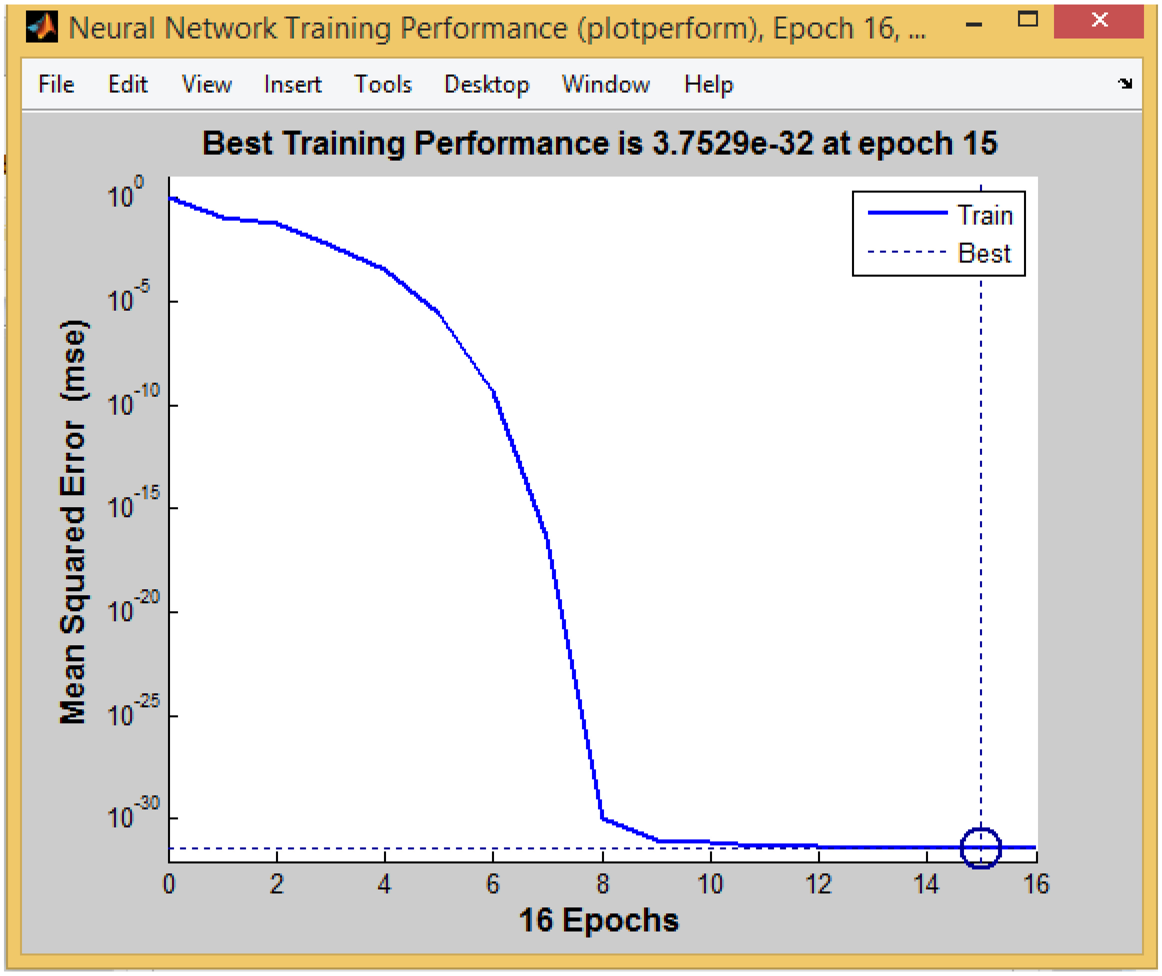
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)

Plot Interval: 1 epochs

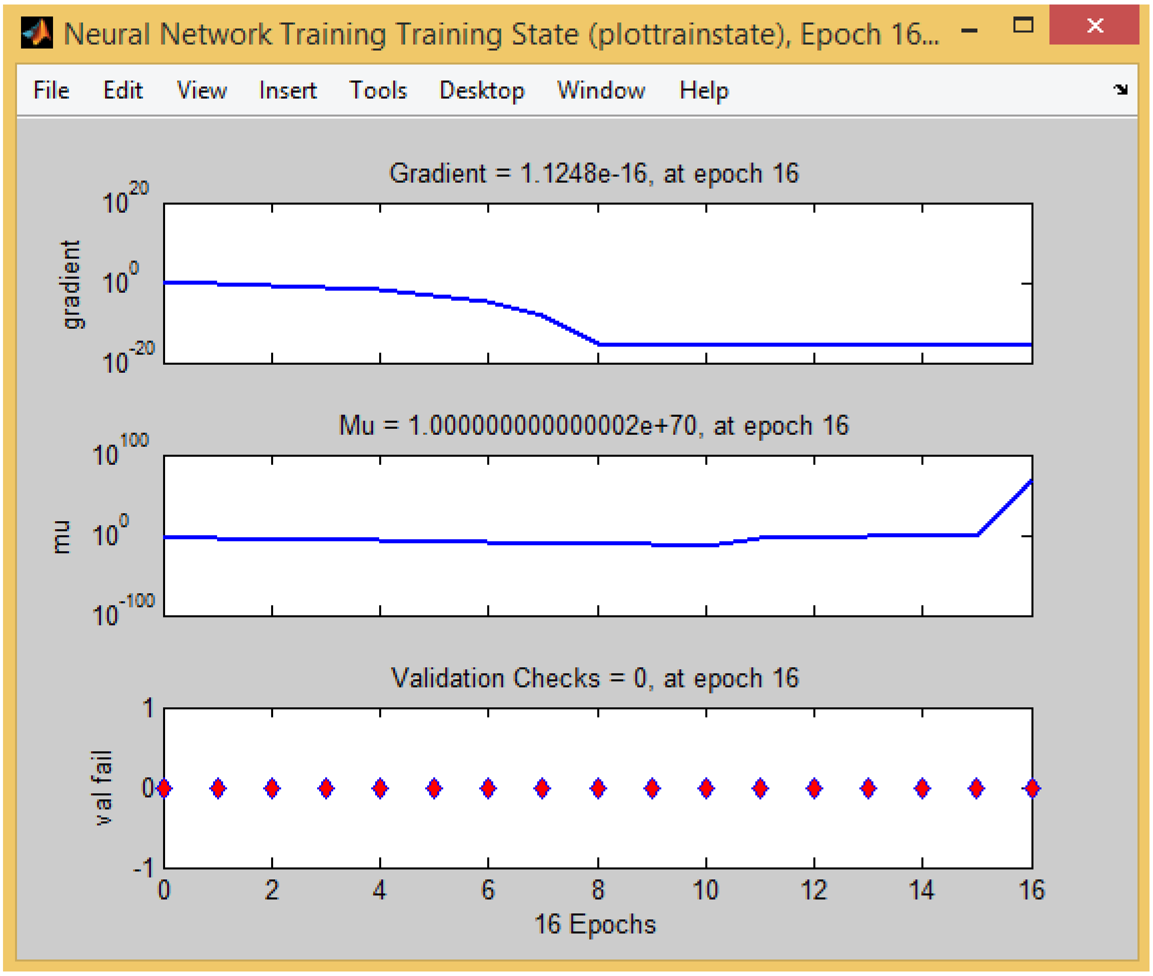
Maximum MU reached.

Here training ended up with an error 3.75x e -32. As the error is pretty much low, this network will provide accurate results.

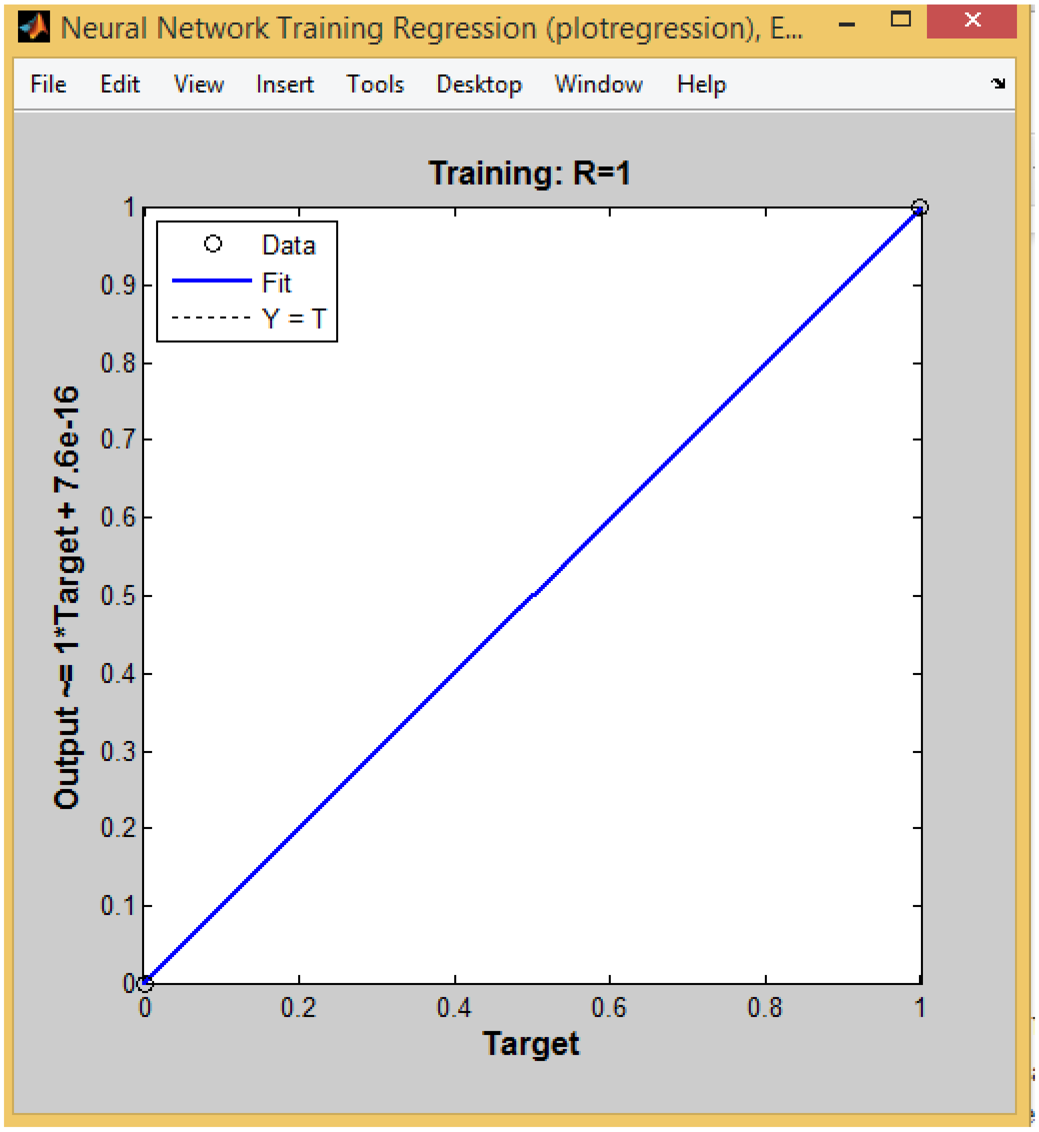
Training Performance



Training State



Training Regression



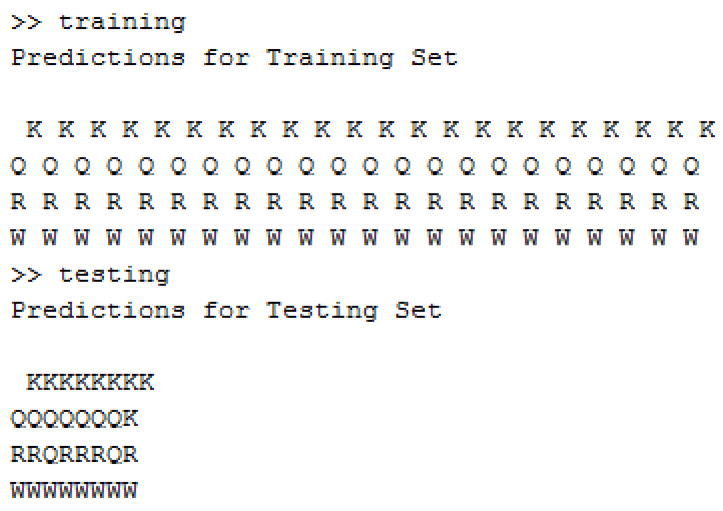
<u>Real Input:</u>

K K K K K K K K K K K K K K K K K K K K K K
Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q
R R R R R R R R R R R R R R R R R R R R R R
W W W W W W W W W W W W W W W W W W W W W W

<u>ANN Predicted Input:</u>

K K K K K K K K K K K K K K K K K K K K K K
Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q
R R R R R R R R R R R R R R R R R R R R R R
W W W W W W W W W W W W W W W W W W W W W W

Output Of the neural network (Predictions for training and testing set)



```
>> training
Predictions for Training Set

 K K K K K K K K K K K K K K K K K K K K K K
Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q Q
R R R R R R R R R R R R R R R R R R R R R R
W W W W W W W W W W W W W W W W W W W W W W
>> testing
Predictions for Testing Set

 KKKKKKKK
QQQQQQQK
RRQRRRQR
WWWWWWWW
```

Accuracy of the training set
All letters are trained correctly by the neural network.

Letter	Output	Accuracy as a fraction	Accuracy as a percentage
K	KKKKKKKKKK	10/10	100%
Q	QQQQQQQQQQ	10/10	100%
R	RRRRRRRRRR	10/10	100%
W	WWWWWWWWWW	10/10	100%

Accuracy of the testing set

Letter	Output	Accuracy as a fraction	Accuracy as a percentage
K	KKKKKKKK	8/8	100%
Q	QQQQQQQK	7/8	87.5%
R	RRQRRRQR	6/8	75%
W	WWWWWWWW	8/8	100%

Part 2 - Unsupervised Network

Under this section we are to train a neural network using the unsupervised learning. Unlike in supervised learning (previous section), here no target is given. Unsupervised learning allows us to approach problems with little or no idea what our results should look like (Here we do not give the target together with the training set). We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by **clustering** the data based on relationships among the variables in the data.

Q: Consider the four variables p, q, r and s having values within the given ranges.

$0.2 < p < 0.4$ $0.5 < q < 0.7$ $0.9 < r < 1.2$ $1.3 < s < 1.6$

Create 200 random values for each p, q, r and s.

Here we have clustered the cluster inputs within four ranges into four groups as **p**,**q**,**r** and **s**. As target is not given network decides the most appropriate class for the given input by seeking the most relevant node in the hidden layer.
As the initial step we generated random real numbers within the four given ranges.

Four ranges were as follows.

p	q	r	s
0.2 - 0.4	0.5-0.7	0.9-1.2	1.3-1.6

We used the MATLAB command “rand” to generate the random numbers within a specified range. Here we generated random numbers, including 200 from each range.

```
n=200;
p=0.2+rand(1,n)*(0.4-0.2);  % generate 200 random numbers in the range of 0.2 to 0.4
q=0.5+rand(1,n)*(0.7-0.5);  % generate 200 random numbers in the range of 0.5 to 0.7
r=0.9+rand(1,n)*(1.2-0.9);  % generate 200 random numbers in the range of 0.9 to 1.2
s=1.3+rand(1,n)*(1.6-1.3);  % generate 200 random numbers in the range of 1.3 to 1.6
```

Q: Build the 4 x 1000 matrix X in the following arrangement

Class 1	Class 2	Class 3	Class 4	Class 5
1 - 200	**201 - 400**	**401 - 600**	**6001 - 800**	**801 - 1000**
p	**q**	**r**	**s**	**s**
q	**r**	**s**	**p**	**r**
r	**s**	**p**	**q**	**q**
s	**p**	**q**	**r**	**p**

In order to get 4x1000 matrix X, concatenated the p,q,r,s matrices using the following matlab command.

```
X = [p q r s s ; q r s p r ; r s p q q ; s p q r p];
```

Q: Design a Self Organising Map (SOM) to classify the above 5 classes.

I. Build a Training Set (X1) from X using all data values in X except the EVERY FOURTH Data Value and Extract EVERY FOURTH Data Value to the Test Set (X2)

Test data set X2 (4x250 matrix) was created by extracting every fourth column of matrix X. Reaming 750 columns were included in matrix X1 which was used as the training data set.

```
X1= zeros(4,750);              %  create 4 x 750 matrix filled with zeros
X2= zeros(4,250);              %  create 4 x 250 matrix filled with zeros

cnt1=1;                        %  counter variables to manipulate index
cnt2=1;
for i=1:1:1000
   if(mod(i,4)==0)             % if column no is divisible by 4
      X2(:,cnt1)= X(:,i);      % insert column to X2 matrix
      cnt1 = cnt1+1;
   else
      X1(:,cnt2) = X(:,i);     % insert column to X1 matrix
      cnt2 = cnt2+1;
   end
End
```

II. Create the network - Self Organizing Map (SOM)

Our next step was to create the network - the Self Organizing Map (SOM). For that we used the built in matlab function ‘*newsom’.*

```
SOM_net = newsom(minmax(X1),[5 1]);

SOM_net.trainParam.epochs = 200;
SOM_net.trainParam.goal = 0.01;
```

Here we used ‘newsom’ command with minimum and maximum values of our Training Set and the output lattice is set to be 5x1 in dimension. The network is set to be trained into 200 epochs.
The error goal was set to 0.01

III. Train the Network you have designed with X1

‘Train’ command was used to train the SOM. Using the above created training set we trained the SOM.

```
SOM_net_trained =train(SOM_net,X1);
```

IV. Obtain the trained Weight Set - W

Using the following we obtained the weight vector.

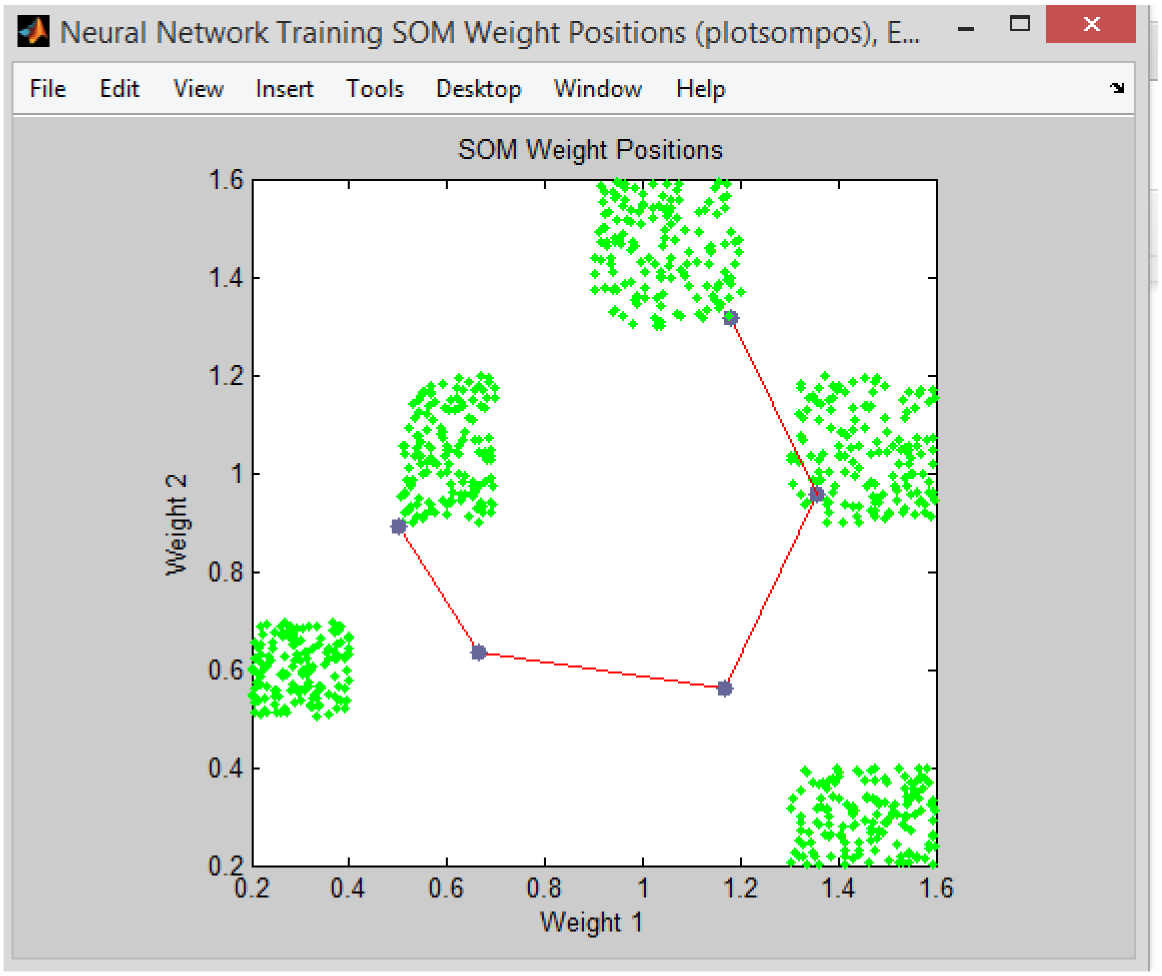
```
w=SOM_net_trained.iw{:,:};
fprintf 'Weights (After Training): '\n;
```

To display the weight matrix following code is used.

```
disp(w);
```

Weights after training is as follows.

```
Weights (After Training):
    1.1925    1.3175    0.3997    0.4973
    1.3508    0.9665    0.5242    0.5666
    1.1578    0.5606    0.7172    0.9725
    0.6602    0.6405    1.0437    1.0633
    0.4993    0.9094    1.3188    0.6801
```

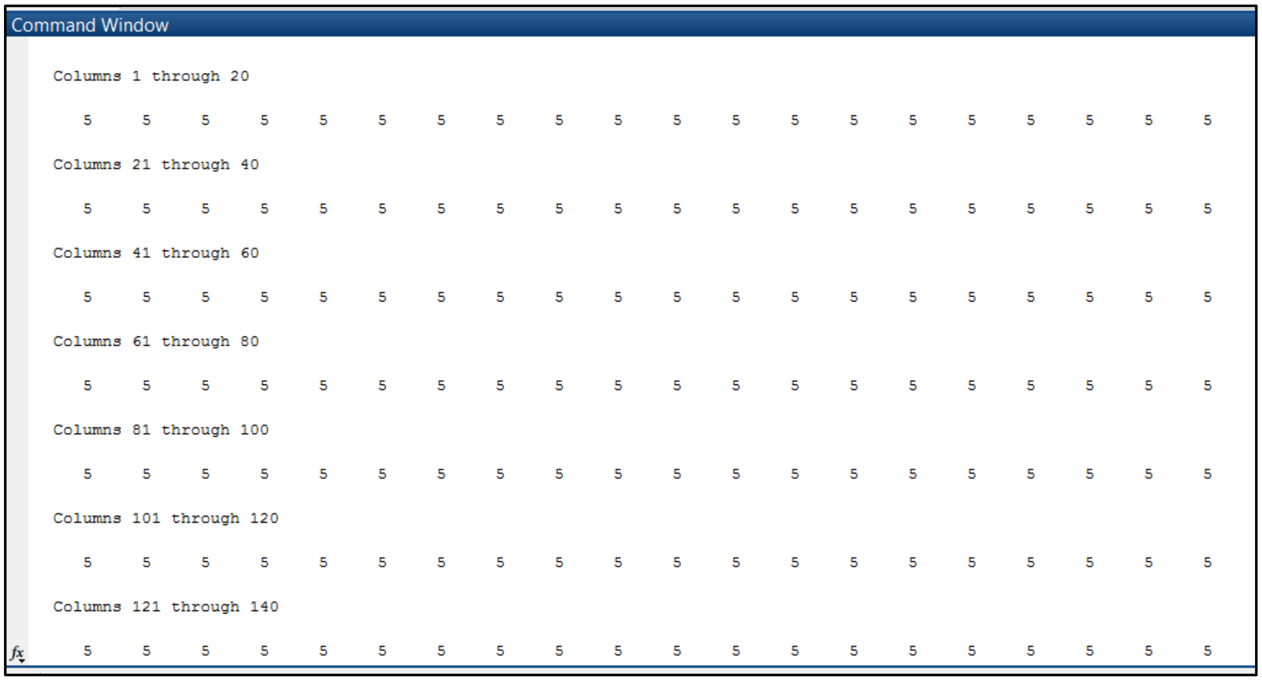

Neural Network Training SOM Weight Positions (plotsompos), E...
File Edit View Insert Tools Desktop Window Help
SOM Weight Positions
Weight 1
Weight 2
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6

V. Clustering the training set and plotting the results

'Sim' function in Matlab was used to simulate the the network created. Simulations outputs returned by sim function was saved in the varibale *simulated_training.* Returned vector was then converted into indices using vec2ind command.

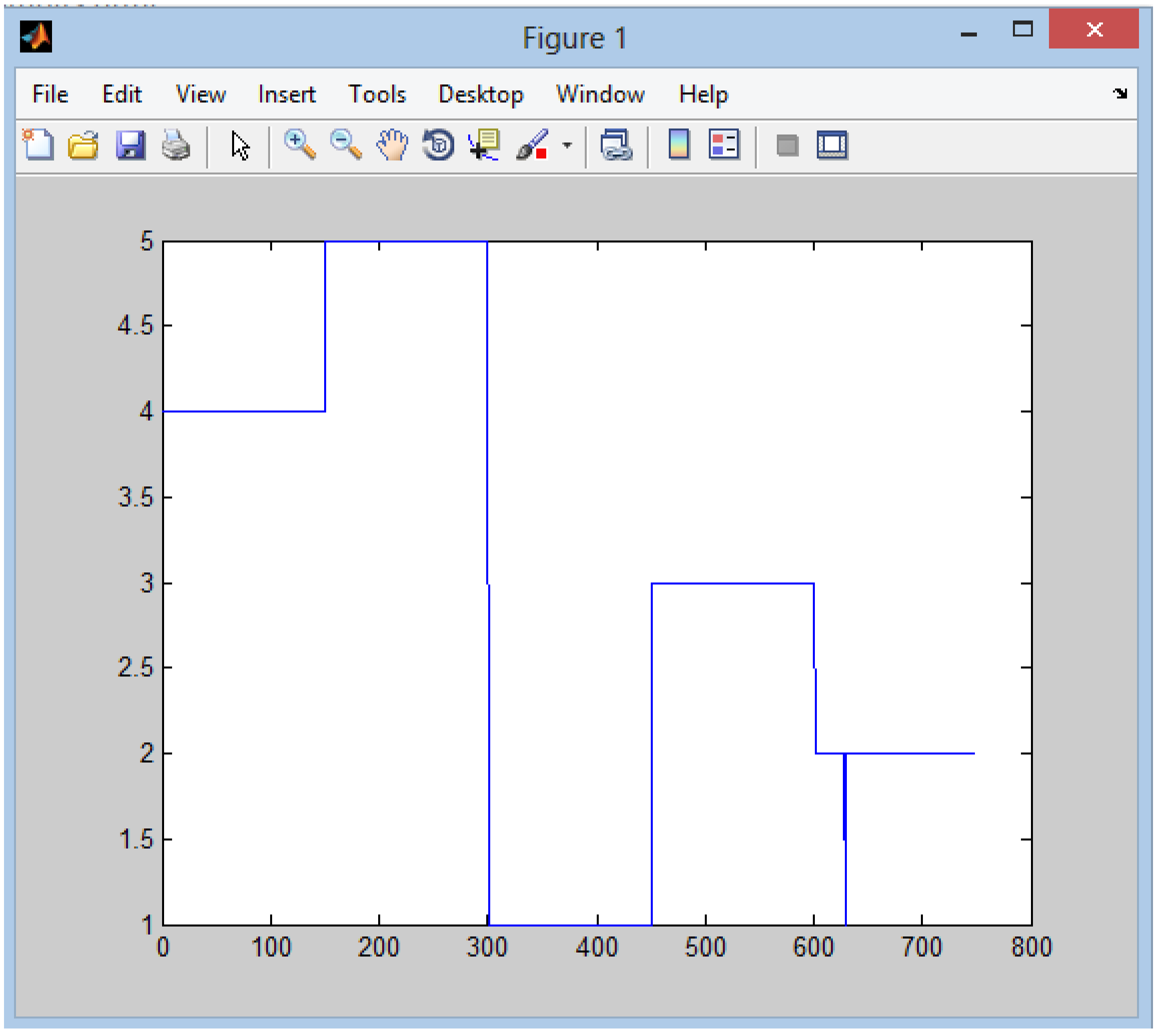
```
simulated_training=sim(SOM_net_trained,X1);
TrainingIndices=vec2ind(simulated_training);
disp(TrainingIndices);
plot(TrainingIndices);
```

disp(TrainingIndices); command is used to display cluster details (outputs of the training set) and those results can be viewed through the command window as in the following interface.

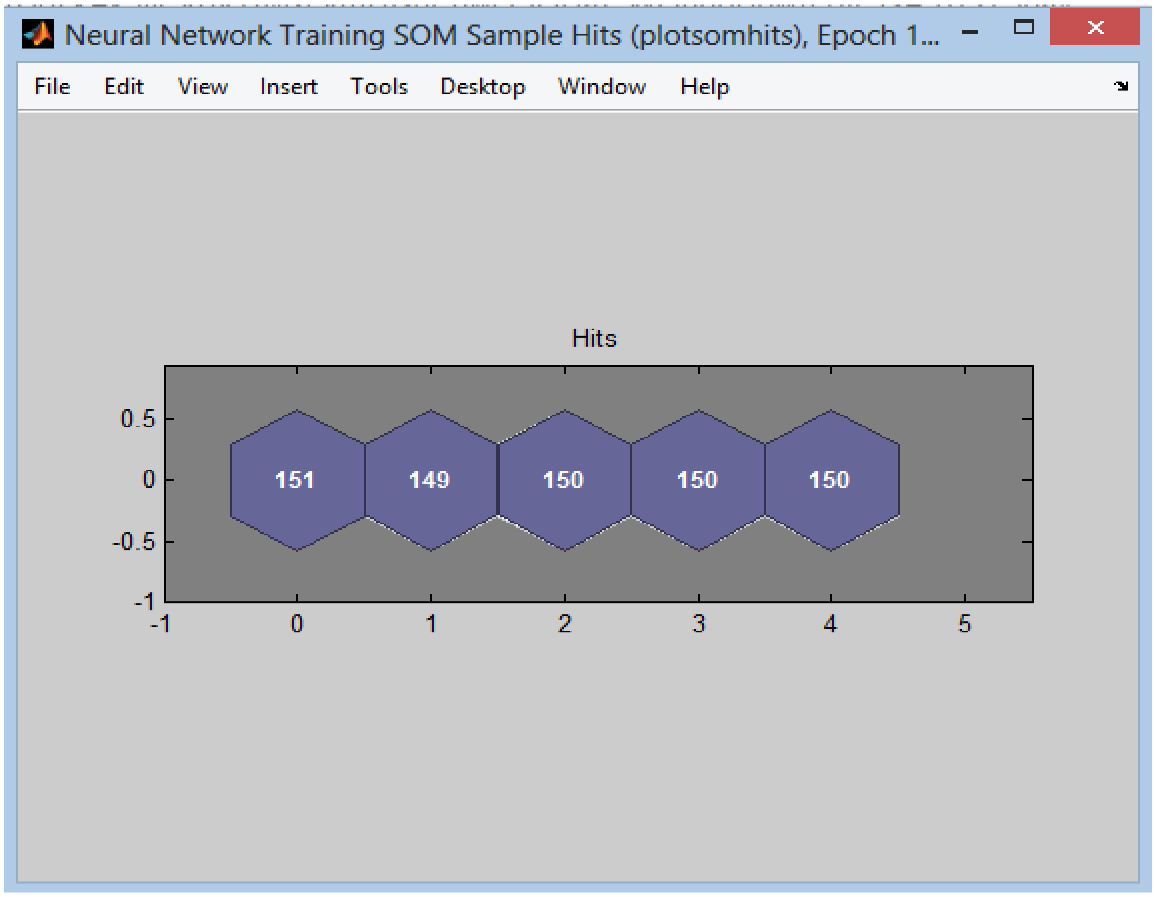


```
  Columns 1 through 20

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 21 through 40

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 41 through 60

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 61 through 80

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 81 through 100

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 101 through 120

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 121 through 140

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5
```

Those training outputs can be plotted in a map using "plot(TrainingIndices)" command.



SOM sample hits were appeared as follows.



Train Results

1 - 150 → Class 1
151 - 300 → Class 2
301 - 450 → Class 3
451 - 600 → Class 4
601- 750 → Class 5

Error in training

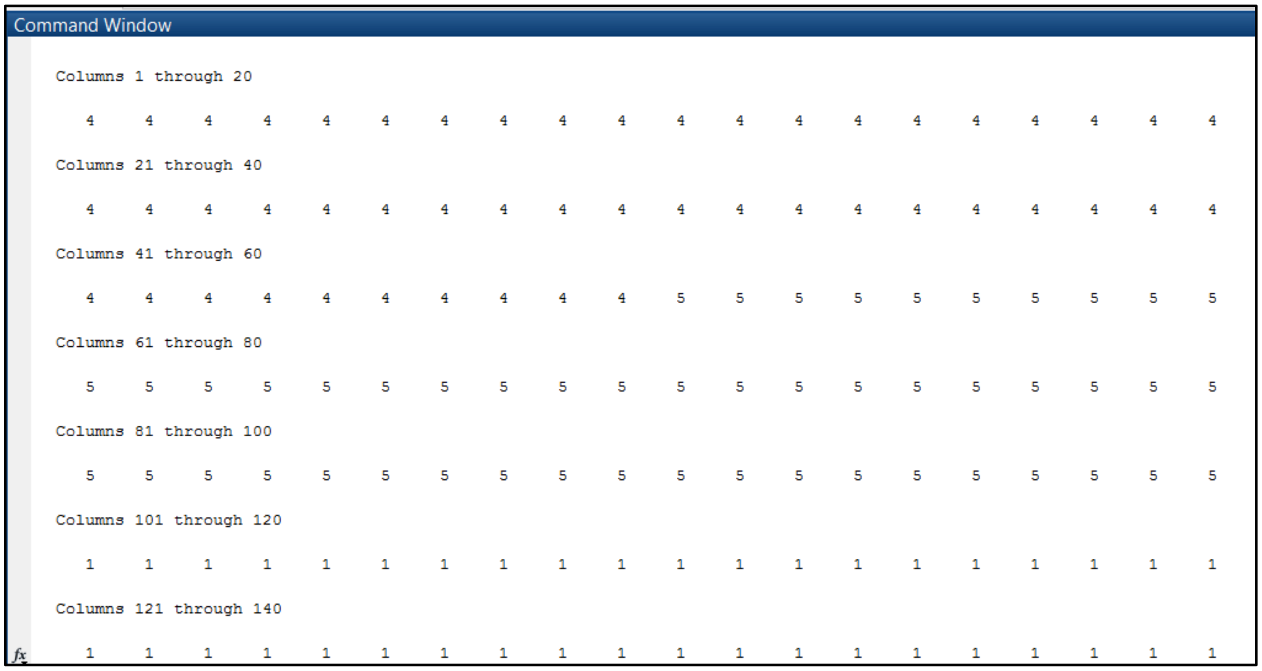
Error in 1- 150 section = 0.6667%
Error in 151 - 300 section = 0.6667%
Error in 301 - 450 section = 0%
Error in 451 - 600 section = 0%
Error in 601- 750 section = 0%

VI. Verifying the testing set

After the training, SOM is tested using Test data set.

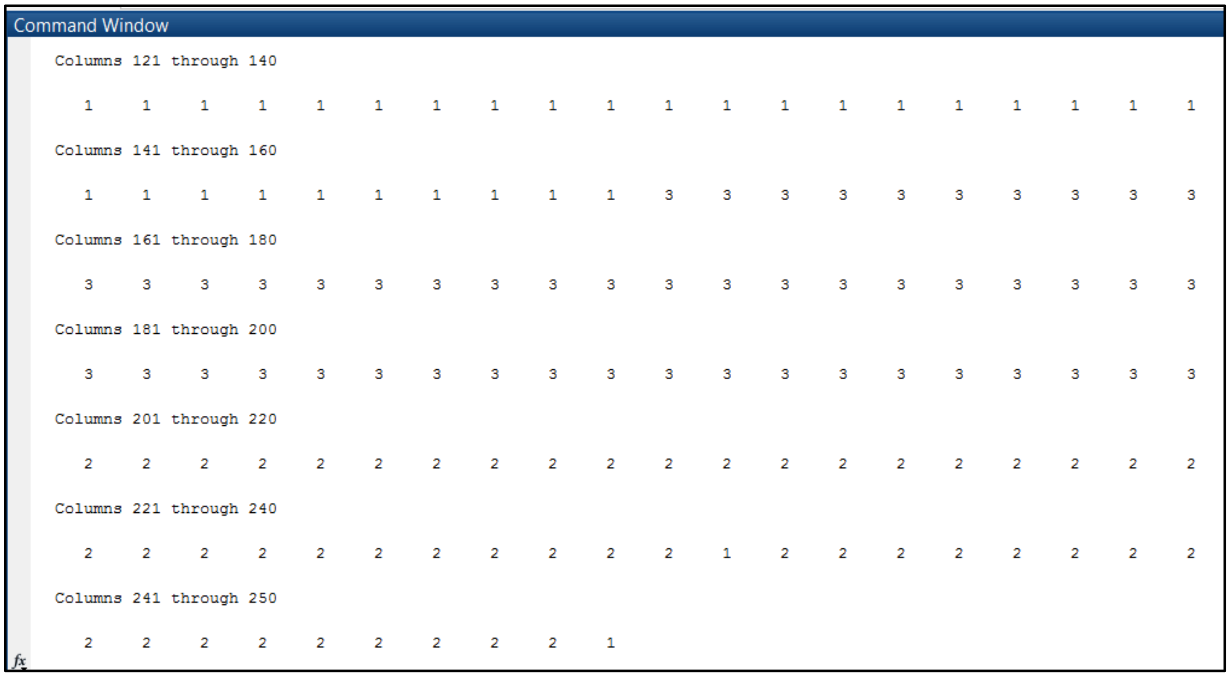
```
simulated_testing=sim(SOM_net_trained,X2);
TestingIndices=vec2ind(simulated_testing);
disp(TestingIndices);
plot(TestingIndices);
```

The following result will be displayed.



```
Command Window

  Columns 1 through 20

     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4

  Columns 21 through 40

     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4     4

  Columns 41 through 60

     4     4     4     4     4     4     4     4     4     4     5     5     5     5     5     5     5     5     5     5

  Columns 61 through 80

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 81 through 100

     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5     5

  Columns 101 through 120

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 121 through 140

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
```

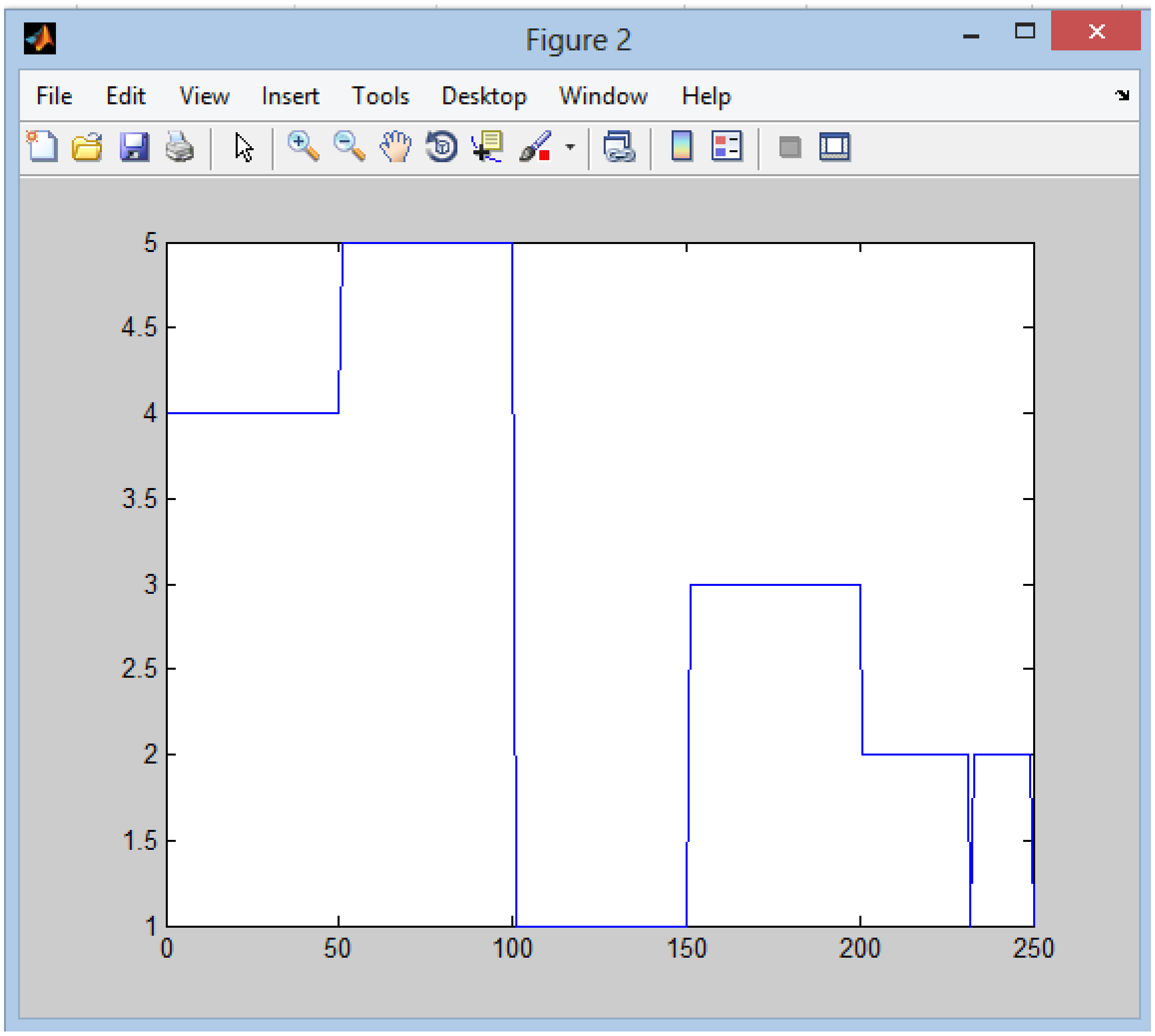
]



Command Window

```
  Columns 121 through 140

     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

  Columns 141 through 160

     1     1     1     1     1     1     1     1     1     1     3     3     3     3     3     3     3     3     3     3

  Columns 161 through 180

     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3

  Columns 181 through 200

     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3     3

  Columns 201 through 220

     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2

  Columns 221 through 240

     2     2     2     2     2     2     2     2     2     2     2     1     2     2     2     2     2     2     2     2

  Columns 241 through 250

     2     2     2     2     2     2     2     2     2     1
```

The test results can also be graphed using the MATLAB command *'plot'*
plot(TestingIndices)



Test results

1 - 50 → Class 1
51 - 100 → Class 2
101 -150 → Class 3
151 - 200 → Class 4
201 - 250 → Class 5

Error in Testing

Error in 1- 50 Section = 0%
Error in 51-100 Section = 0%
Error in 101-150 Section = 0%
Error in 151-200 Section = 2%

Conclusion

Question 01 - Supervised Network

Our training process accuracy is 100% since all the letters were correctly predicted. But in the testing process, only letters K and W got 100% accuracy. Letter Q accuracy was 87.5% while letter R accuracy was 75%. Mean squared error of the testing process is 0.019. Even though this is a small value we cannot conclude that the network is 100% accurate.

Question 02 - Unsupervised Network

During each and every instance an error rate was gained. The error could occur at any class out of the five classes. During the training error rate was 0.667% and during the testing the error rate was 2%. The error rate did not go beyond the percentage, 2%. Therefore we cannot conclude that the network is 100% accurate.