

University of Colombo School of Computing (UCSC)

SCS3120/IS3017/CS3120 Machine Learning and Neural Computing

Assignment 02 (Practical Assignment)

Report Group 03 UCSC

Group Members (UCSC)

Serial No	Index No	Signature
1	14000522	
2	14000628	
3	14001004	
4	14020752	
5	14020793	

(Date of Submission : 21/07/2017)

Table of Content

Question 01	2
Motivation	2
Resources	2
Input Data Table	2
Preparation of Data	3
Creating the Network and Training	5
Testing and Analysis	7
Summary - How to run the network	9
 Question 02	 10
Motivation	10
Resources	10
Preparation of Data	10
Designing the Network	11
Training the Network	11
Results	12
Validation	13
Accuracy	15

Question 01 - Supervised Training

Motivation

The motivation of the question is to classify images with a single sinhala letter correctly to its class. We have given 3 different sinhala letters ඉ, ඊ, උ to classify into 3 classes.

Our task is to implement an Artificial Neural Network (ANN) based solution to carry out above scenario.

We have implemented a feedforward backpropagation neural network as our solution.

Resources

In order to design the network and simulate, we have used **Octave Software** along with 2 external libraries ‘**image**’ and ‘**nnet**’ from Octave-Forge

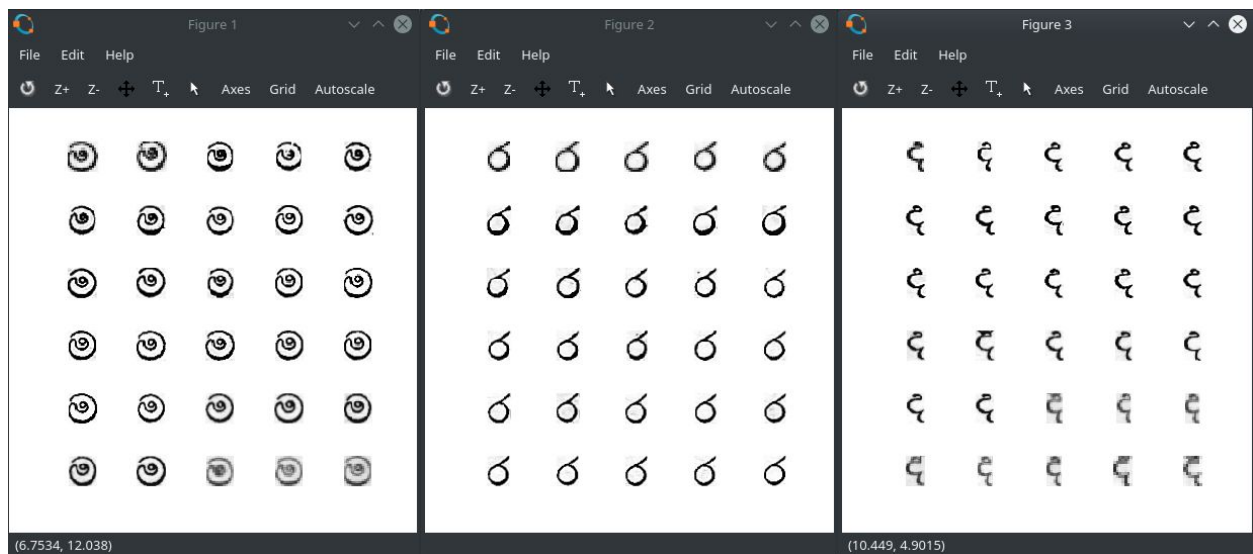
Image package - <https://octave.sourceforge.io/image/>

Nnet package - <https://octave.sourceforge.io/nnet/>

Input Data Table

Character image data taken from LMS (“Group03 UCSC.mat”)

Input data table assigned to our group plotted as follows.



Preparation of Data

Preparing the data and making datasets is the most crucial part in supervised learning.

Data preparation is done in 3 stages. ***Enhancing the shape***, ***Making the datasets*** and ***Declaring target values***.

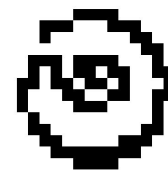
Enhancing the Shape (code file: cleaning_func.m)

In order to train a network we did some transformations to the character images and enhanced the shape of characters

Original Character Image



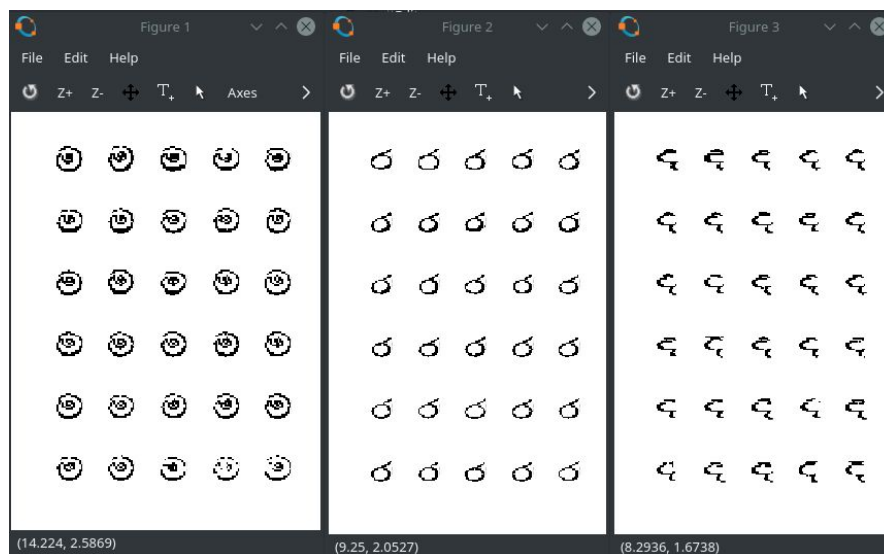
Transformed Character Image



In order to transform the image, we implemented a separate function (cleaning_func)

Steps in enhancing,

1. Cropped the image using its binary image matrix
 - Rows containing only ones (1 = white) till finding a 0 containing row from top and bottom are removed from original image matrix.
 - Columns containing only ones (1 = white) till finding a 0 containing column from left and right are removed from original image matrix.
2. Resized the cropped image to make it 14x14 in dimension
3. Made the resized image binary



(90 images after enhancing the shapes)

Making the Datasets (code file: make_dataset.m)

Resizing the data makes each character image is a 14x14 matrix. Then we make this a column vector to 196x1 dimensions.

After cleaning the 90 character images, those are separated into 3 sets.

1. Training dataset - 21 images (70%)
2. Validation dataset - 3 images (10%)
3. Testing dataset - 6 images (20%)

* Validation dataset is used to **prevent overfitting**.

** In the question it has started to use 22 images for training and 8 for testing, but since we are using an extra dataset for validation we had to reduce training set for 21 and testing set for 6.

Declaring the Target Values (code file: make_dataset.m)

We have 3 classes as outputs. Therefore our approach is to use a single output node for each class. So the outputs for each class are as follows.

- 100 as the target for 🍌 (“Ma”)
- 010 as the target for 🍌 (“Ra”)
- 001 as the target for 🍌 (“Da”)

Target Values for Training Set (21*3 = 63 images)			
Targets for 🍌 : y1	11111111111111111111	00000000000000000000	00000000000000000000
Targets for 🍌 : y2	00000000000000000000	11111111111111111111	00000000000000000000
Targets for 🍌 : y3	00000000000000000000	00000000000000000000	11111111111111111111










Target Values for Validation Set (3*3 = 9 images)			
Targets for 🍌 : y1	111	000	000
Targets for 🍌 : y2	000	111	000
Targets for 🍌 : y3	000	000	111

Creating the Network and Training (code file: training.m)

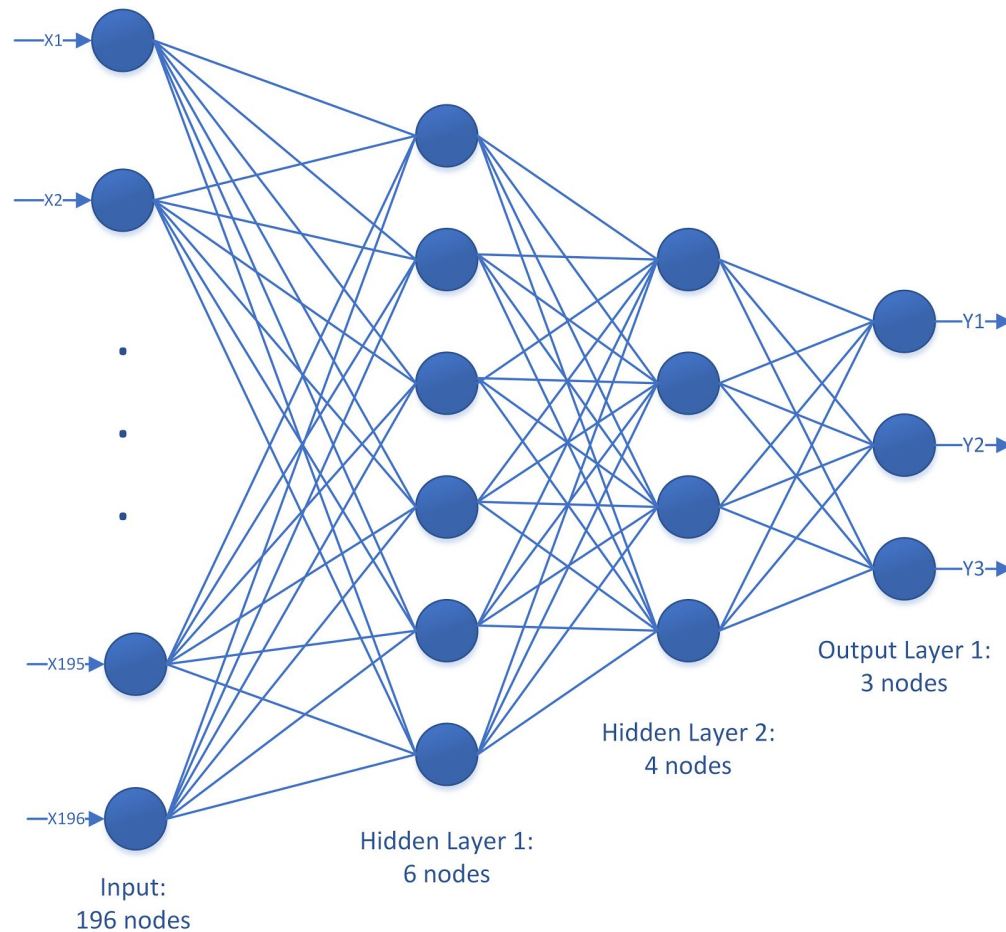
Network implemented using **newff** function.

In order to create a suitable network we had to train and test the network several times.

Here are some of the results of training with different hidden nodes and hidden layers.

   1 hidden layer with 1 node (no validation set) Accuracy : 38.8%	   1 hidden layer with 6 nodes Accuracy : 77%
   2 hidden layer with 6 and 4 nodes Accuracy : 100%	
Network structure created using $\text{net} = \text{newff}(\text{Pr}, [6, 4, 3]);$ PR = min_max values of training set [6, 4, 3] = [hidden layer 1 # nodes, hidden layer 2 # nodes, # output nodes]	

We managed to get a good success rate (100% accuracy) with 2 hidden layers. Network structure is as follows,



Training

```
net = train(net, Training_Set, Targets, [], [], Validation_Set);
```

```
>> training
```

```
** Warning in INIT
** Network net.inputs{1}.range has a row with equal min and max values.
** Constant inputs do not provide useful information.
```

```
TRAINLM, Epoch 0/50, MSE 1.34485/0, Gradient 293.742/1e-10
TRAINLM, Epoch 24/50, MSE 0.00413758/0, Gradient 0.0452719/1e-10
TRAINLM, Validation stop.
```

```
Training completed
```

Once we find a good trained network, we save it to a file so that we don't want to train the network each time. File is saved as ***"trained_network.mat"***

Testing and Analysis (code file: testimg_func.m | testing.m)

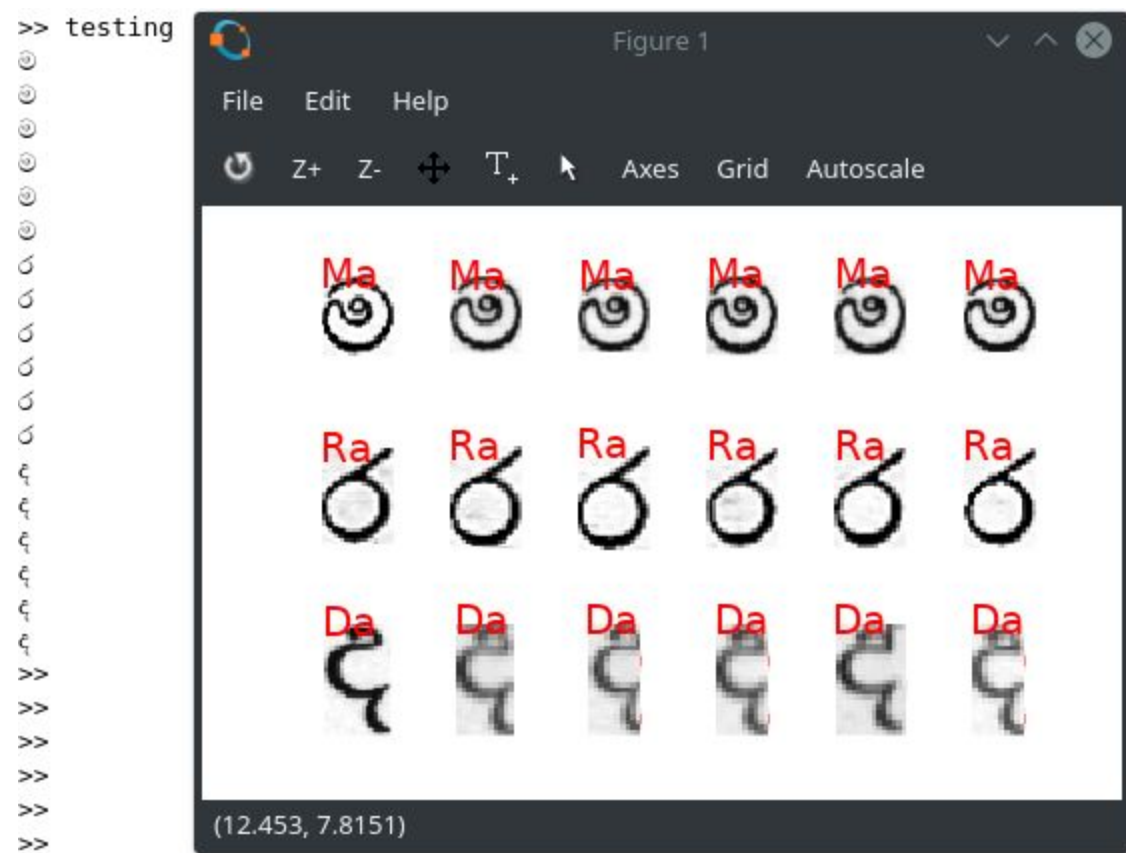
```
round ( sim (net, img_test) );
```

sim function is used to simulate the network and test.

Now, we can test the network using the testing data.

* Note that the network output class is printed with **red** on top of each character image

* Original image is displayed, but cleaned and transformed image is used to classify.



Here we could obtain **100% accuracy** for each testing character.

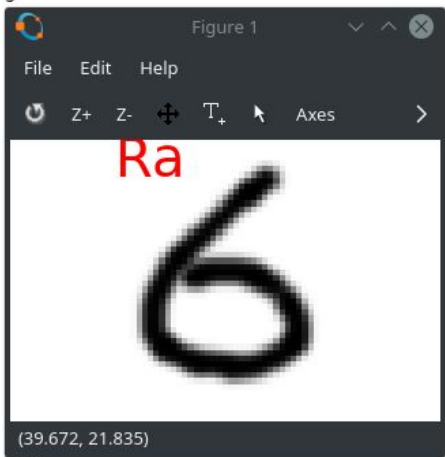
Testing some more images

To simulate the network, we wrote a separate function called “testimg_func”. To use that function you need only the trained network which we saved as “trained_network.mat”.

```
load trained_network.mat;           # load net
img = imread ("test_images/1.png"); # reads an image file
testimg_func(img, net);             # simulate the network
```

To test the network further, we have drawn 3 letters using an image drawing tool and simulated. It also gave 100% accuracy.

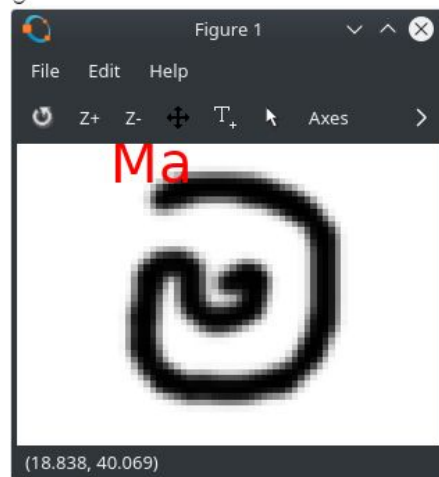
```
>> test1 = imread("test_images/1.png");
>> testimg_func(test1,net);
```



```
>> test2 = imread("test_images/2.png");
>> testimg_func(test2,net);
```



```
>> test3 = imread("test_images/3.png");
>> testimg_func(test3,net);
```



Summary - How to run the network

Option 01: (Working directory = q1 directory)

1. Run group03q1.m
 >> group03q1

Option 02 : (Working directory = q1/ CodeFiles directory)

Make the working space of Octave as the directory containing code files files (CodeFiles directory),

To train and simulate testing.

1. Run make_dataset.m file
 >> make_dataset
2. Run training.m file
 >> training
3. Run testing.m file
 >> testing

```
>> make_dataset
Dataset completed
>>
>> training

** Warning in INIT
** Network net.inputs{1}.range has a row with equal min and max values.
** Constant inputs do not provide useful information.

TRAINLM, Epoch 0/50, MSE 4.07435/0, Gradient 542.859/1e-10
TRAINLM, Epoch 14/50, MSE 3.63058e-05/0, Gradient 0.530402/1e-10
TRAINLM, Validation stop.

Training completed
>>
>> testing
⊙
⊙
⊙
⊙
⊙
⊙
⊙
```

To simulate the network without training.

1. Load trained network
 >> load trained_network.mat;
2. Load an image/matrix
 >> img = imread("img.png");
3. Simulate the network
 >> testimg_func(img, net);



Question 02

Motivation

The motivation of the question is to cluster a randomly generated dataset into 5 target classes. The approach chosen here is to use a Self Organizing Map (SOM) in carrying out unsupervised classification.

Resources

For the purpose of this study MATLAB software along with its inbuilt Neural Network Toolbox has been used in preparation of data, training and testing the network.

Preparation of Data

As per the instructions of the assignment, four random vectors (p, q, r, s) were generated each having 200 components using following code in MATLAB,

```
p = 0.2 * rand(1, 200) + 0.6;  
q = 0.2 * rand(1, 200) + 0.9;  
r = 0.2 * rand(1, 200) + 1.3;  
s = 0.2 * rand(1, 200) + 1.7;
```

Consequently, these vectors were concatenated in permutations given in the assignment to create a 4 x 1000 matrix with following code,

```
row1 = [p, q, r, s, s];  
row2 = [q, r, s, p, r];  
row3 = [r, s, p, q, q];  
row4 = [s, p, q, r, p];  
  
X = [row1; row2; row3; row4];
```

Every fourth column in the dataset is chosen as the test dataset (X2 (4 x 250)) while rest of the columns were chosen as training data (X1 (4 x 750)).

```

for i = 1:1000
    if mod(i, 4) == 0
        X_Test = [X_Test, X(:, i)];
        continue
    end
    X_Train = [X_Train, X(:, i)];
End

```

Designing the Network

Since the original problem specifies 5 classes, the SOM was designed with 5 output nodes. Output nodes were fully connected to four input nodes.

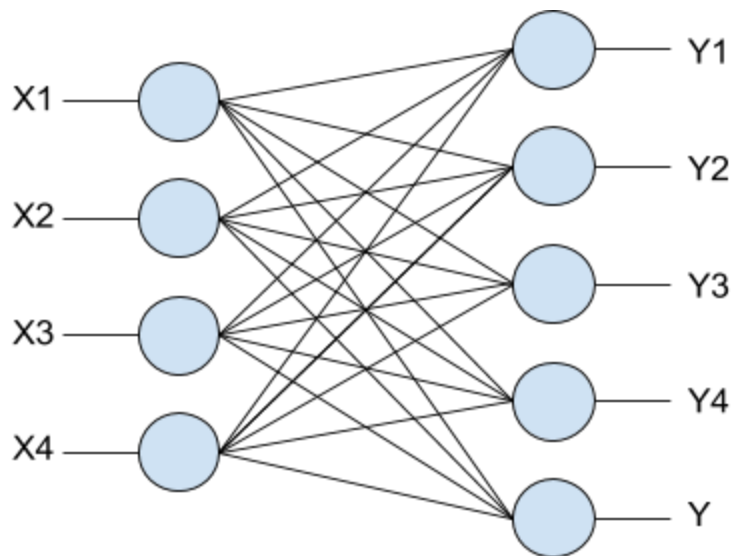


Figure: The structure of the SOM used in classification with inputs labelled X and outputs Y

Training the Network

The network was trained against the data in training set (X_Train) for 1000 epochs,

```

sMap = selforgmap([5, 1]);
sMap.trainParam.epochs = 1000;

[sMap, tr] = train(sMap, X_Train);

```

Results

After training, the network has determined the final weight matrix,

1.4053	1.8028	0.7038	1.0067
1.7998	1.4047	1.0101	0.7024
1.8004	0.7038	1.0065	1.4052
1.0065	1.4075	1.7980	0.7049
0.7062	1.0073	1.4055	1.7994

Table: Final weight matrix. Vertical axis for output nodes and horizontal axis for input nodes

The SOM has recognized five clusters within the training dataset.

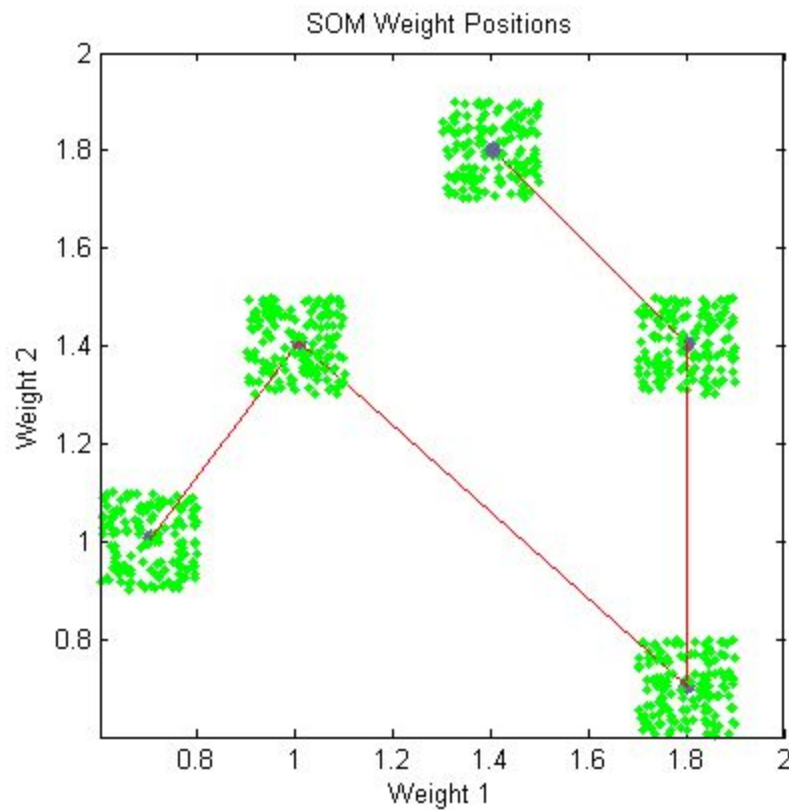


Figure: Position of SOM nodes in data-space after training. Green dots represents the input vectors and grey dots (obscured by green dots) represents SOM nodes while red lines represent neighboring nodes.

The final network has classified 150 input vector for each som node which is plotted as follows,

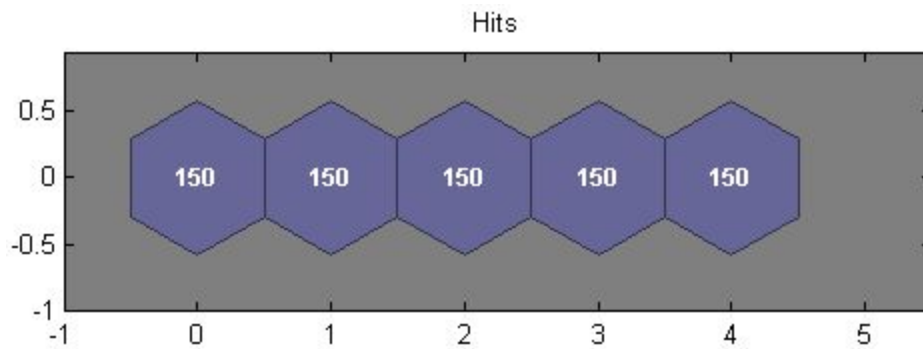


Figure: 150 input vectors in each output class

Validation

In order to validate the results of the network the testing set (X_Test) has been fed into the network,

```
test = vec2ind(sMap(X_Test))
```

Which resulted in following classification,

Columns 1 through 15

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

Columns 16 through 30

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

Columns 31 through 45

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

Columns 46 through 60

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

Columns 61 through 75

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

Columns 76 through 90

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

Columns 91 through 105

4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1

Columns 106 through 120	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Columns 121 through 135	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Columns 136 through 150	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Columns 151 through 165	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Columns 166 through 180	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Columns 181 through 195	3	3	3	3	3	3	3	3	3	3	3	3	3	3
Columns 196 through 210	3	3	3	3	3	2	2	2	2	2	2	2	2	2
Columns 211 through 225	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Columns 226 through 240	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Columns 241 through 250	2	2	2	2	2	2	2	2	2					

Figure: Output classification for the testing dataset

Since the testing data was chosen as every fourth column from the original dataset, definitions of the classes in original problem is remapped to assess the network performance

Class	Original	Remapped	Class assigned by SOM
Class 1	1 - 200	1 - 50	5
Class 2	201 - 400	51 - 100	4
Class 3	401 - 600	101 - 150	1
Class 4	601 - 800	151 - 200	3
Class 5	801 - 1000	201 - 250	2

Table: Remapping of original ranges and classes determined by SOM

Accuracy

Class (SOM)	Expected	Actual	Accuracy (Expected % Actual)
5	50	50	100
4	50	50	100
1	50	50	100
3	50	50	100
2	50	50	100

Table: The accuracy of classification for SOM determined classes

From the table it is apparent that the network has a 100% accuracy in classifying the provided dataset.