Lab-3 Character Recognition using a Back-propagation Neural Network

Software: MATLAB, Neural Network Toolbox

Character recognition is one of the most widely used biometric traits for authentication of person as well as document. In this assignment a Back propagation Neural network (BPNN) is designed to model the way in which a few characters are recognized.
Each student has to design and train a Back-propagation Neural Network to recognize 10 characters (uppercase letters, A-Z, and numbers, 0-9). Select any 10 characters from your name and/or student ID. Please note that repeating characters are not allowed. Represent each letter by an m by n grid of Boolean values. The size of a character must be bigger than 7 rows and 5 columns.

There are two methods available in MATLAB: Neural Network Toolbox, the *nftool*, *nntool* and *command-line*. However, *nftool* is specialized in fitting problem only. Therefore, it is not covered in this lab assignment. Use the *nntool* GUI or *command-lines* for design, training and simulation of the back-propagation network.

Objective: To carry out the character recognition activity using Back Propogation Neural Network

Tasks: Follow instructions as given in Page3 and complete the following Tasks.

(i) Prepare the data set such as inputs /output similar with Lab - 2.

(ii) Design neural network by varying Parameters such as number of layers, number of neuron in each layer and training algorithm.

(iii) Demonstrate the effectiveness of data handling through performance.

(iv) Comment your results

(v) Using 4 additional training algorithms, vary training parameters, tabulate the results of all training algorithm and comment the best possible set of variables based on the network performance.

Train the network without noise (1 sample) and with noise (4 samples**).** To generate the noise sample, follow step 3 in the following procedure. Once the network is trained well, demonstrate how the network identifies any of the learned object when the Boolean values of a character are input to the back-propagation network. Test the network using object with noise and without noise.
The available Training algorithms are TRAINBFG, TRAINBFGC, TRAINBR, TRAINC, TRAINCGB, TRAINCGF, TRAINCGP, TRAINGD, TRAINGDA,

TRAINGDM, TRAINGDX, TRAINLM, TRAINOSS, TRAINR, TRAINRP, TRAINSCG. Fix the following parameters and comment on the training time and errors:

- Epochs
- Number of Hidden Layers
- Number of Neurons in Each Hidden Layer
- Transfer Function of Each Layer

Report Format

You are required to prepare a formal report for this lab assignment. Please make sure that your name and Student ID is included in the first page of your softcopy /hardcopy report. Your report will be assessed based on the demonstration shown and how the work done is documented. However, the following aspects are essential:

1. Details of characters chosen (10 characters taken from Name/ID)
2. I/O preparation for training.
3. The network architecture of the working network.
4. Training parameters of the working network.
5. Comparative study on the performances with different training algorithms

You might also find the other topics useful for report preparation. Please take note that this doesn't mean 'copy and paste directly from the help file is allowed'. This documentation is available at Help >> Product Help >> Neural Network Toolbox.

Report Submission

This report is worth 10 marks of your internal marks.

Assessment rubrics: Marking rubrics to evaluate Lab-3 Character Recognition using a Back-propagation Neural Network.
Note: Maximum mark provided for this Experiment is 10

Preliminary studies (Max. Mark 1): Preparation of Data sets such as Inputs/ output/ Test/Testout/ without noise/with noise Demonstration and explanation is poor (40%)/ acceptable (60%)/Excellent (100%)	Usage of NN tool box (Max.Mark: 1). Display the level/ ability in using and demonstrating the effectiveness of data handling; Not proficient (20%)/ Novice level (40%)/ Intermediate level (60%/commendable (100%)	NN Training with Training algorithms. Number layers, Number of Neuron in each layer to show the varaition. (Max. Mark: 2) Methods and agruments to support decisions: presented poorly (40% / Vague (60%) / Clearly 100%)	Simulation results: Adequate Modification and training. Show error as minimum as possible (or) Regression R =1 suring simulation (Max. Mark (2). Little information on results (20%) Analysis of results (50%) Critical assessment of results(100%)	Comparative study on the performance: Max. Mark (4) Discuss and Demonstrate atleast 4 Training Algorithm and comment your results. Achievements made are poor (20%), Acceptable (50%) and Commendable/examplary (100%)		

Step-1

Choice of Characters

Consider the characters N, G, O, –, E, 1, 8, and 3. There are 8 characters in total. This example demonstrates the recognition of all 8 characters.

Step-2

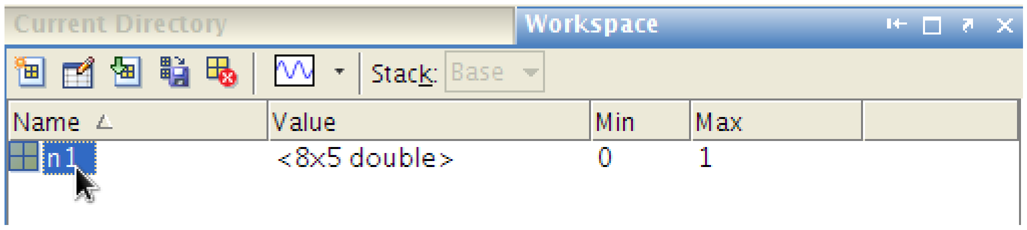
Creating a character in Matlab

Let's say we want the character size to be 8x5, we'd first need to create the array. Here's one possible way to do it:-

```
>> n1 = zeros(8,5);
```

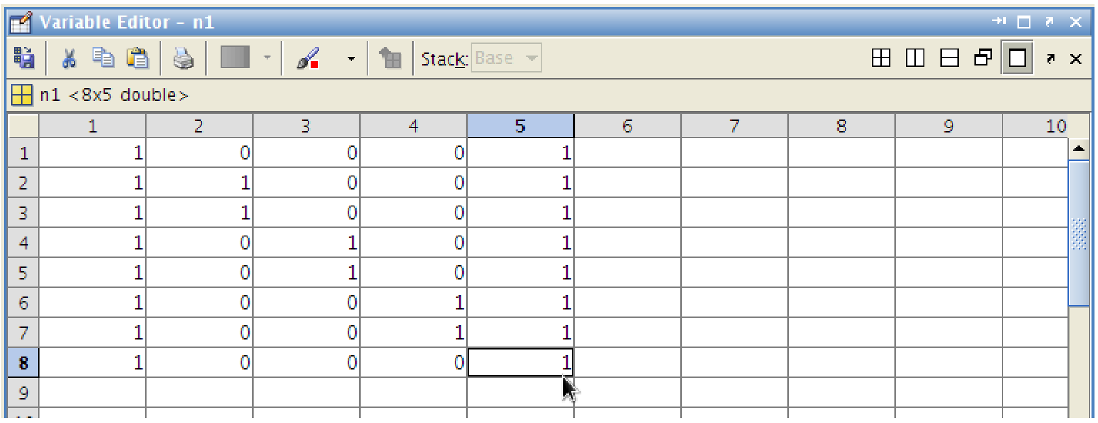
This creates an array named n1, with 8x5 size and filled with zeros. You can edit the array graphically using Matlab's Variable Editor. Either run the following command or double-click on your variable in the Workspace tab:-

```
>> openvar('n1')
```



Name	Value	Min	Max
n1	<8x5 double>	0	1

Change the values within the variable to create the chosen letter. For example, here's the letter N:-

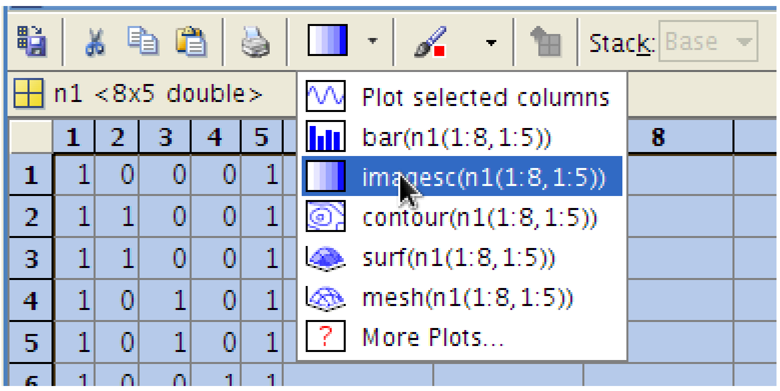


n1 <8x5 double>

	1	2	3	4	5	6	7	8	9	10
1	1	0	0	0	1					
2	1	1	0	0	1					
3	1	1	0	0	1					
4	1	0	1	0	1					
5	1	0	1	0	1					
6	1	0	0	1	1					
7	1	0	0	1	1					
8	1	0	0	0	1					
9										

Of course, you can change the column width to get a better view of how it would actually look like in an image. To preview your character, you can use either imagesc or imshow. To access imagesc, select the entire array using *Ctrl-A* and select imagesc from the toolbar. Alternatively, you could run imshow:-

```
>> imshow(n1)
```



	1	2	3	4	5
1	1	0	0	0	1
2	1	1	0	0	1
3	1	1	0	0	1
4	1	0	1	0	1
5	1	0	1	0	1
6	1	0	0	1	1

In either case, the image is likely to consist of non-square pixels. To correct this, the following command should be run in the Command Window, without closing the figure:-

```
>> axis square
```

Step-3

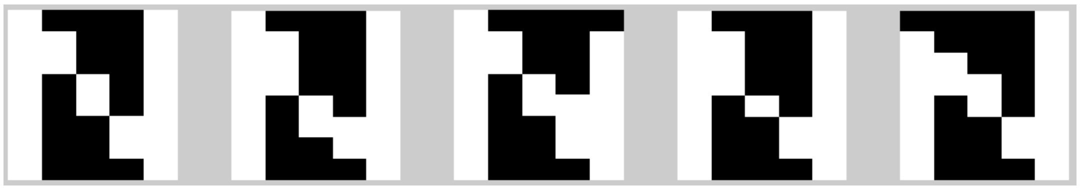
Adding Noise to characters

To create additional 'noisy' versions of an existing character, first the character should be copied to new variables:-

```
>> n2=n1;
>> n3=n1;
>> n4=n1;
>> n5=n1;
```

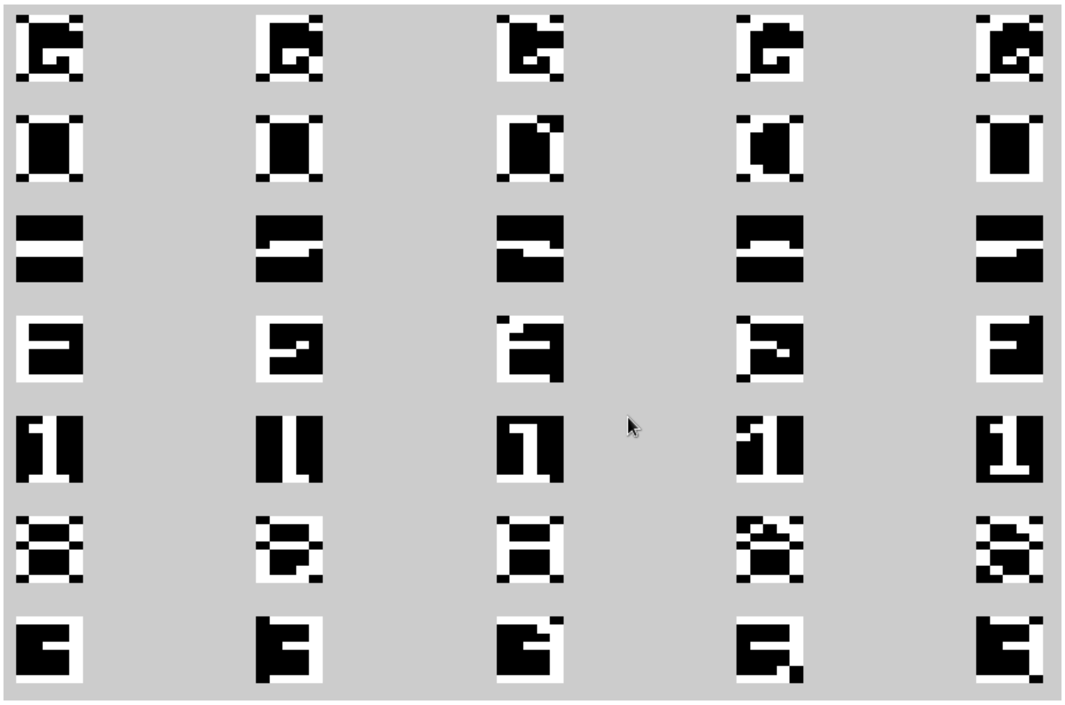
Then, each character is modified by inverting (changing a 0 to a 1 or a 1 to a 0) some pixels. This should be done for characters n2 till n5. Character n1 should be left unchanged (no noise). The subplot command allows for displaying all the images at the same time. Each image has had 2 or 3 pixels inverted.

```
>> subplot(1,5,1),imshow(n1),axis square
>> subplot(1,5,2),imshow(n2),axis square
>> subplot(1,5,3),imshow(n3),axis square
>> subplot(1,5,4),imshow(n4),axis square
>> subplot(1,5,5),imshow(n5),axis square
```



Saving the Matlab workspace frequently is a good idea, to prevent loss of any work in progress.

Samples of remaining characters



The first character in each row is the image without noise. Together with the first example, there are 8 characters in this data-set.

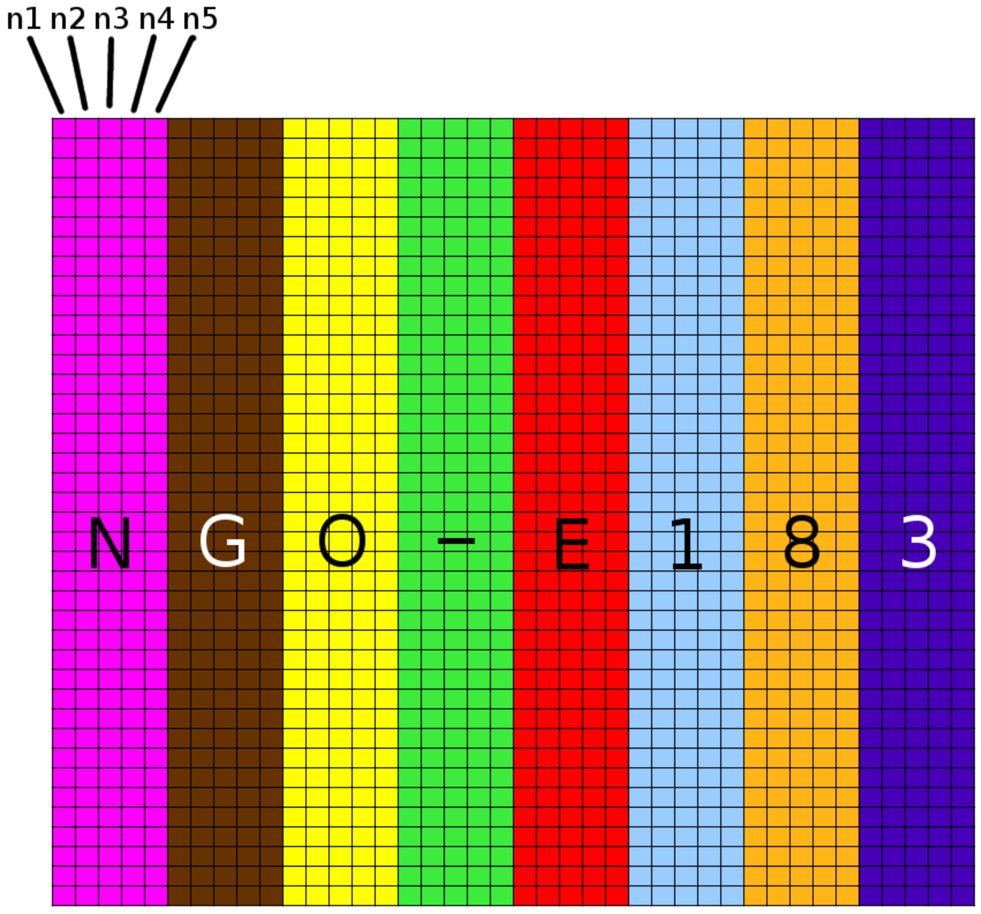
Step-4

Preparing the Training Input

The neural network toolbox input takes each column of the input matrix as a separate unit. The input characters so far need to be converted to a matrix of one column per character. The reshape command is used to create rows for each character. Then, the matrix is transposed so that each column contains the values from a single character.

```
>> traininput = [reshape(n1,1,40); reshape(n2,1,40); reshape(n3,1,40);
>>   reshape(n4,1,40); reshape(n5,1,40);
>>   reshape(n1,1,40); reshape(g2,1,40); reshape(g3,1,40);
>>   reshape(g4,1,40); reshape(g5,1,40);
>>   reshape(o1,1,40); reshape(o2,1,40); reshape(o3,1,40);
>>   reshape(o4,1,40); reshape(o5,1,40);
>>   reshape(hyphen1,1,40); reshape(hyphen2,1,40); reshape(hyphen3,1,40);
>>   reshape(hyphen4,1,40); reshape(hyphen5,1,40);
>>   reshape(e1,1,40); reshape(e2,1,40); reshape(e3,1,40);
>>   reshape(e4,1,40); reshape(e5,1,40);
>>   reshape(one1,1,40); reshape(one2,1,40); reshape(one3,1,40);
>>   reshape(one4,1,40); reshape(one5,1,40);
>>   reshape(eight1,1,40); reshape(eight2,1,40); reshape(eight3,1,40);
>>   reshape(eight4,1,40); reshape(eight5,1,40);
>>   reshape(three1,1,40); reshape(three2,1,40); reshape(three3,1,40);
>>   reshape(three4,1,40); reshape(three5,1,40)];
>> traininput = traininput';
```

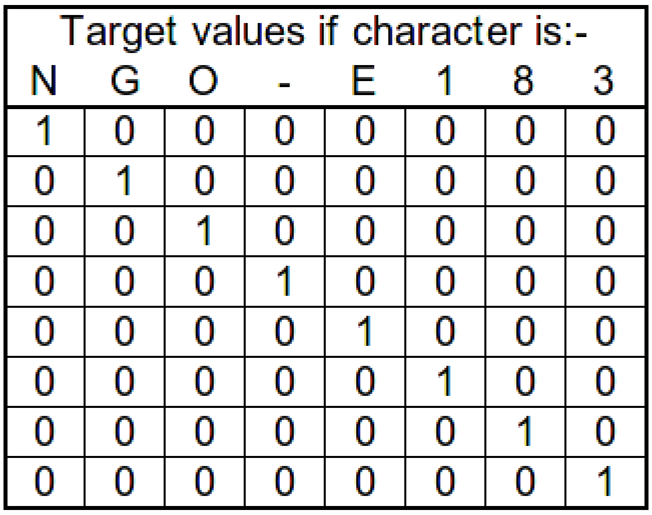
There are many ways to prepare the proper column-wise input for neural network training besides what has been shown here. The final result can be similar to this:-



Step-5

Preparing the Training target

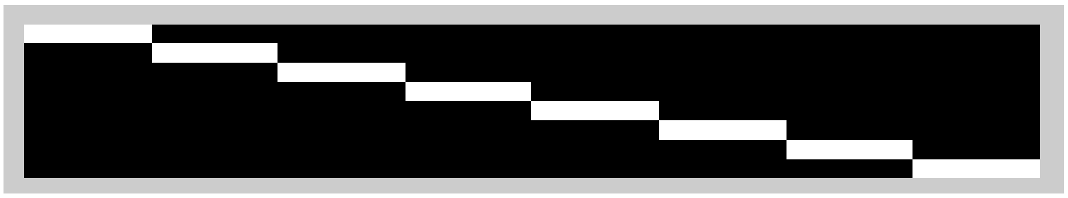
Just as there are many ways to prepare the training inputs, so also there are many ways to represent the output target. One simple way is to have 8 outputs for 8 recognizable characters, with the output representing the correct character having the value 1 when all the rest have value 0 to that character's input, as shown below.



Target values if character is:-							
N	G	O	-	E	1	8	3
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

While we can use the same method as previously used to create the characters, another method is to create the entire array from the command line. This can be done by creating the array and then setting the necessary pixels to 1. Note that the array has 8 rows for 8 different characters and 40 columns for 40 different inputs (5 per character).

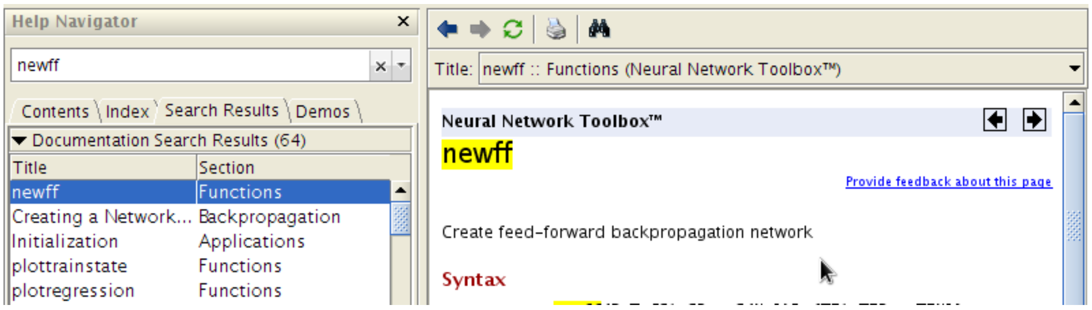
```
>> traintarget = zeros(8,40);
>> traintarget(1,1:5)=1;
>> traintarget(2,6:10)=1;
>> traintarget(3,11:15)=1;
>> traintarget(4,16:20)=1;
>> traintarget(5,21:25)=1;
>> traintarget(6,26:30)=1;
>> traintarget(7,31:35)=1;
>> traintarget(8,36:40)=1;
>> imshow(traintarget)
```



Step-6

Create a Feed Forward Artificial Neural Network (newff)

The next step is to create a neural network. An ANN in Matlab has the following components: input layer, hidden layer(s), output layer, transfer function(s), training algorithm, weights, and biases. As with all other Matlab commands, you should search the Help file and read up on the function, especially the syntax and the necessary inputs/provided outputs.


Help Navigator
newff
Contents
Index
Search Results
Demos
Title: newff :: Functions (Neural Network Toolbox™)
Neural Network Toolbox™
Provide feedback about this page
Syntax

Documentation Search Results (64)

Title	Section
newff	Functions
Creating a Network...	Backpropagation
Initialization	Applications
plottrainstate	Functions
plotregression	Functions

newff

Create feed-forward backpropagation network

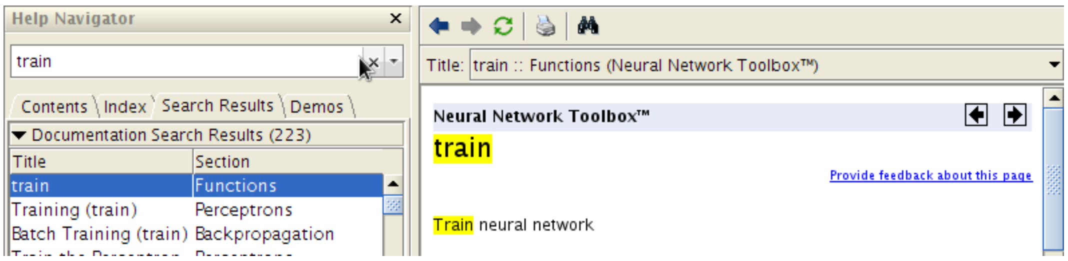
For example, to create a network for this problem, various decisions must be made. In the case of a network with 20 neurons in 1 hidden layer using the '*tansig*' function, with an output transfer function of '*purelin*' and using the '*trainlm*' training algorithm, the following command is used.

```
>> net = newff( traininput, traintarget, [20], {'tansig', 'purelin'}, 'trainlm' );
```

If, on the other hand, the network should have 850 neurons in 2 hidden layers, 500 in the first hidden layer and 350 in the second hidden layer, both using the '*tansig*' function, with an output transfer function of '*purelin*' and using the '*trainbfg*' training algorithm, the following command is used.

```
>> net = newff( traininput, traintarget, [500,350], {'tansig','tansig', 'purelin'}, 'trainbfg'
);
```

Training the ANN (train)


Help Navigator
train
Contents
Index
Search Results
Demos
Title: train :: Functions (Neural Network Toolbox™)
Neural Network Toolbox™
Provide feedback about this page

Documentation Search Results (223)

Title	Section
train	Functions
Training (train)	Perceptrons
Batch Training (train)	Backpropagation

train

Train neural network

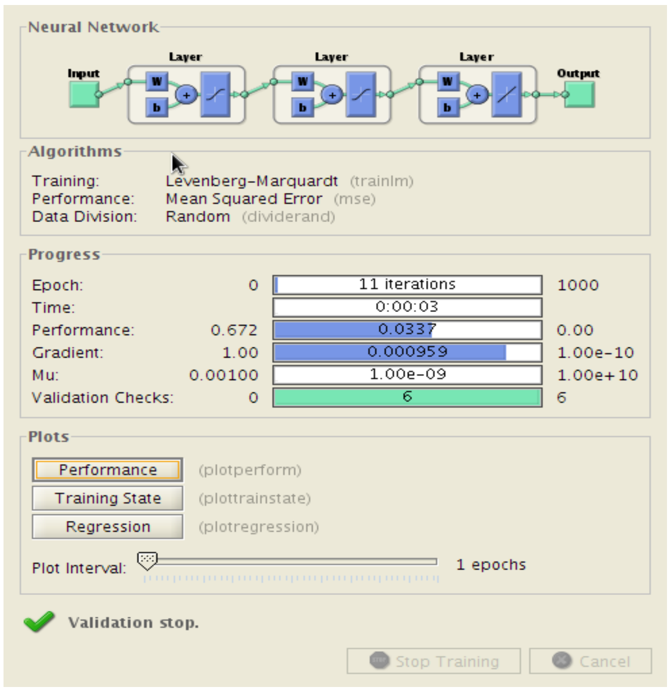
Training the network can be done using:-

```
>> [out_net, trained_record, trained_output, nn_error] = train(net, traininput,
traintarget);
```

Before doing the training using that command, it is normally better to set some of the parameters to something other than the default value. This will need to be done in any case when trying to further improve the quality of the ANN. Note that the below are just suggested values. The Help file contains explanations on what these parameters and others do.

```
>> net.trainParam.show = 50;
>> net.trainParam.lr = 0.05;
>> net.trainParam.epochs = 300;
>> net.trainParam.goal = 1e-5;
```

The variable '*net*' here contains an untrained network, the parameters of which can be set. If another network is created in future, those parameters would also have to be set. Also, the output network inherits all the parameters of the untrained network, but with modified weights and biases as a consequence of the training.



```
Algorithms
Training:        Levenberg-Marquardt  (trainlm)
Performance:     Mean Squared Error  (mse)
Data Division:   Random  (dividerand)

Progress
Epoch:                     0   11 iterations   1000
Time:                          0:00:03
Performance:           0.672   0.0337          0.00
Gradient:               1.00   0.000959        1.00e-10
Mu:                  0.00100   1.00e-09        1.00e+10
Validation Checks:         0   6               6

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

Plot Interval:   1 epochs

Validation stop.
```

Caution: If you cancel your network using Ctrl-C instead of the Stop Training button, your network has been partially altered and may not work, requiring you to create a new one.

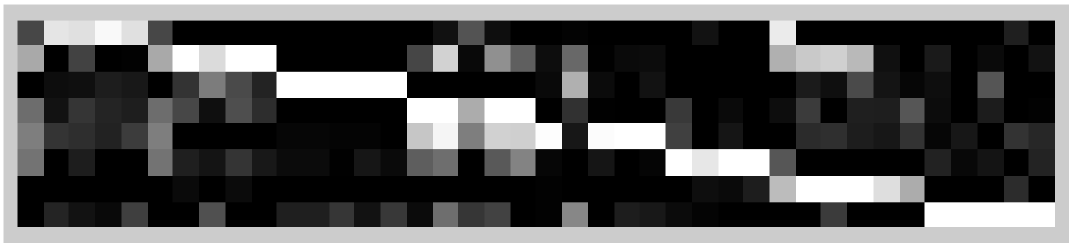
Click on the button labelled '*Performance*'. The plot has three lines, because the input samples and targets vectors are randomly divided into three sets. 60% of the vectors are used to train the network. 20% of the vectors are used to validate how well the network generalized. Training on the training vectors continues as long the training reduces the network's error on the validation vectors. After the network memorizes the training set (at the expense of generalizing more poorly), training is stopped. This technique automatically avoids the problem of overfitting, which plagues many optimization and learning algorithms. Refer to “Automated Data Division During Network Creation” in the help file. There are other plots as well, which may be useful to you when training your networks.

<u>Step-7</u>

Simulating ANN output (sim)

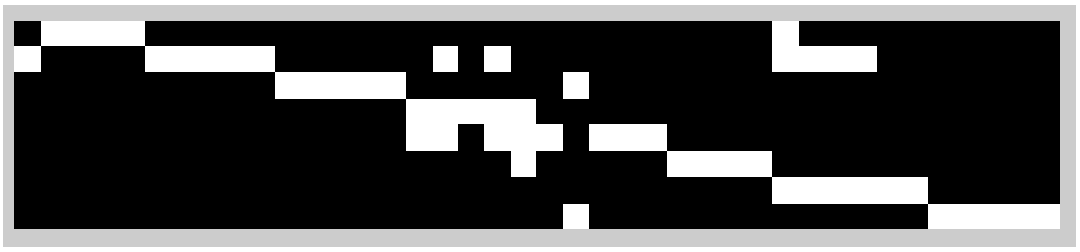
The training process *may* give a functioning network for character recognition. Testing is required to see if this is the case. The sim command simulates the output of a network if some input(s) is applied. If the training inputs are applied to the trained network, the similarity of the output to the target results determines whether the network can be defined as having properly learned the problem.

```
>> output = sim(out_net, traininput);
>> imshow(output)
```



It is sometimes helpful to round the output of the network for better results.

```
>> output_rounded = round(output);
>> imshow(output_rounded)
```



The output of sim can guide the development of the neural network. Ideally, it is possible for rounded results to be perfect, this depends on the dataset you've created and the training parameters you've used.

Improving the Performance

These are some parameters which can be varied to improve the result:-

- Learning Rate
- Epochs
- Goal
- Number of Hidden Layers
- Number of Neurons in Each Hidden Layer
- Transfer Function of Each Layer
- Training Algorithm

This is a trial and error process. Therefore, keep trying until your network is able to recognize all the training samples. REMEMBER! The result for the trained samples MUST BE PERFECT!

Once perfect results have been obtained, move on to next section (**Lab Activities**) and work on the tasks given.

Lab Activities

Complete the following activities and discuss your results. These activities are similar to tasks done in Lab 2.

A. Create **new** inputs to test the ANN. The new inputs should be the same character but should **NOT** be from the training input (Insert noises at different locations).

B. Using the **training input variables**, change the following and discuss the results.

   i. Change the number of neurons/layers and study the effects on the network performance.

   ii. Use different training algorithms (other than what you used during the first part of the lab) and perform a comparative study of the algorithms. Choose FOUR additional algorithms and compare the performance of the algorithms that you have implemented. Discuss the results.