

# Purpose

The purpose of this lab report is to outline the process the group was involved in designing a neural network in matlab. Initial work will involve using Matlab’s built in nntool GUI to understand how ANN works. A command line script will then be written to implement an automated ANN solution. The idea is to let the network learn how to classify given facial image data and map them to an emotion from 6 choices.

# Materials

1. Matlab R2014a
2. Artificial Neural Networks GUI tool (nntool)
3. Data from provided emotions\_data.mat file

# Procedure

The documentation provided for Neural Network Design in the Matlab help outlines the following steps:

1. Collect data
2. Create the network
3. Configure the network
4. Initialize the weights and biases
5. Train the network
6. Validate the network
7. Use the network

Data provided in emotions\_data.mat file contain

1. A matrix x of dimensions 448×136, (There are 448 number of examples and 136 is the dimensionality of the feature vector computed by concatenating the coordinates of 68 facial points).
2. A vector y of dimensions 448×1, containing the emotion labels of the corresponding examples. These labels are numbered from 1 to 6 and correspond to an emotion follows:

1=anger

2=disgust

3=fear

4=happiness

5=sadness

6=surprise

The data needs to be prepared before it can be used with nntool.

## Preparing data

Before we can use the y label vector, we need to convert it to a matrix of rows of ‘1’s and ‘0’s corresponding to the value of the original vector cell. This was done by creating a matlab function, convertNum(inputMatrix)

function OUT = convertNum (inMat)

[m,n] = size(inMat);

outMat = zeros(m,6);

for j = 1:m

OUTA = [0 0 0 0 0 0];

OUTA(inMat(j)) = 1;

outMat(j,1:6) = OUTA;

end

OUT = outMat;

end

this function simply converts the number to a row of 6 binary numbers with the column corresponding to the number set to ‘1’ leaving the other columns at 0. E.g 1=100000,2=010000,3=001000…

we then feed y into this function:

ymat = convertNum(y)

We then transpose ymat:

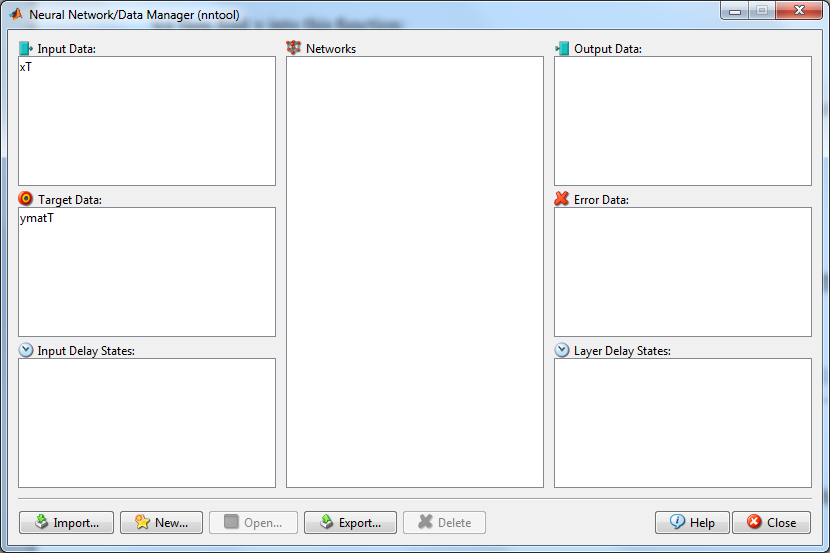
ymatT=ymat’

We next need to also transpose x so that the number of rows in ymat corresponds to the number of columns in the new matrix:

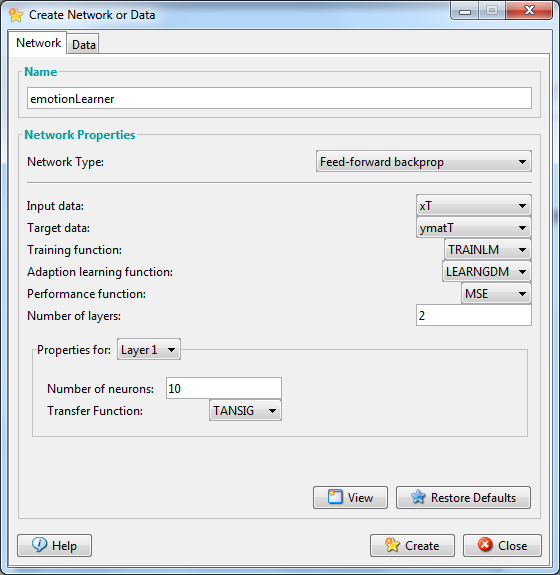
xT = x’

## Creating and training the Network using Matlab nntool

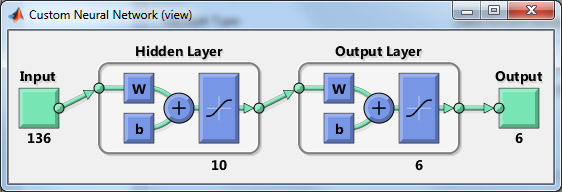
using nntool, xT was fed as input data while ymatT was fed as the target data as seen in figure below:

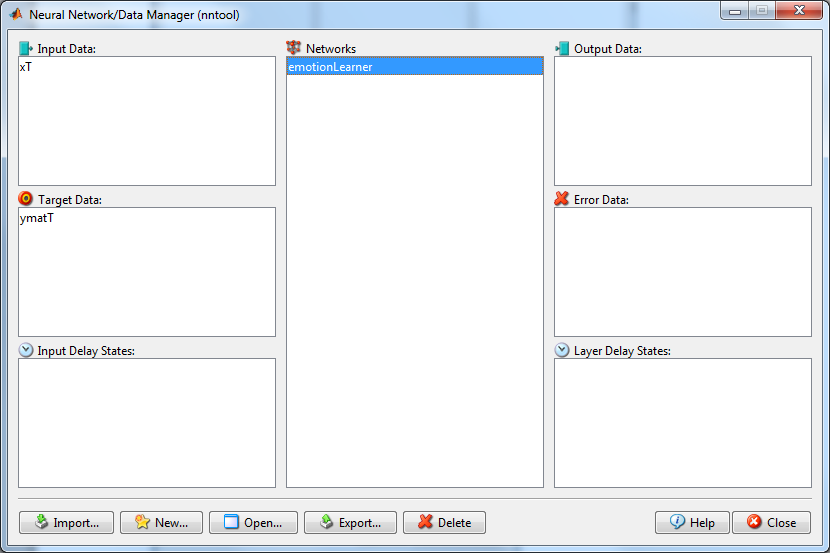


A new neural network learner was created using this data:

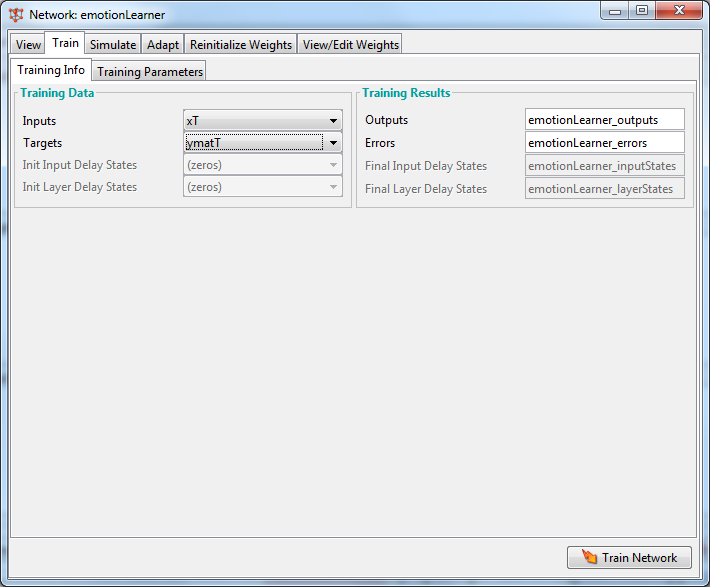


This creates a 2 layer network(one hidden and one output) with 136 row inputs according to xT with 6 rows of target outputs according to ymatT. Adjusting the number of layers would create additional hidden layers. This can be viewed as below:

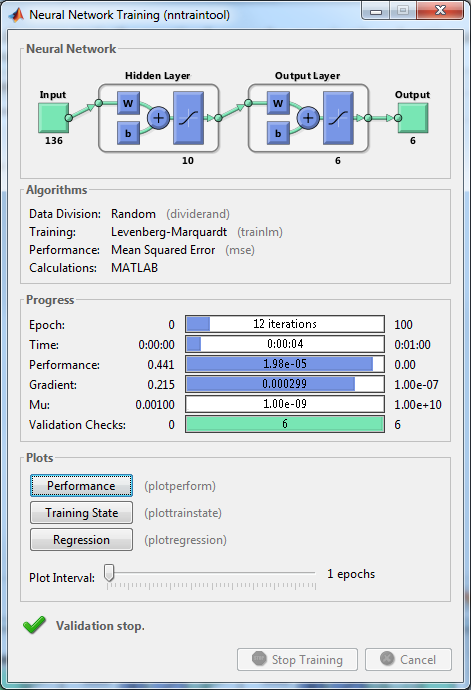


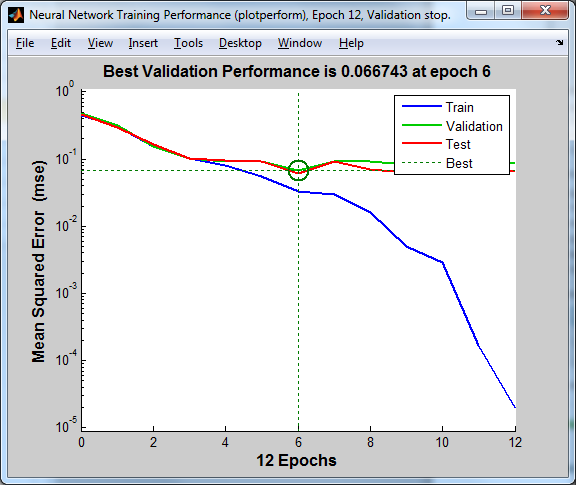


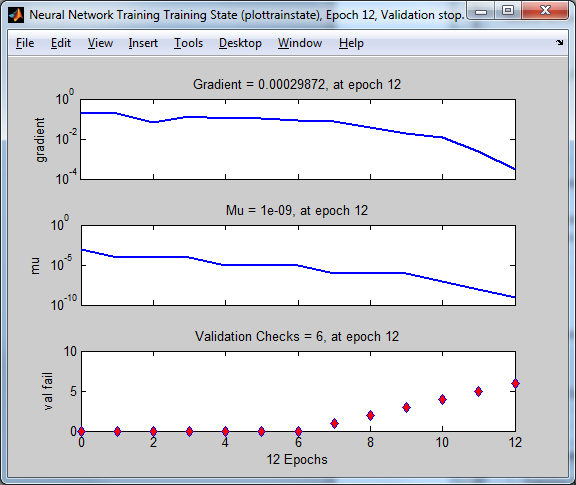
We then can configure the emotionLearner network by opening it and setting our training info and parameters:

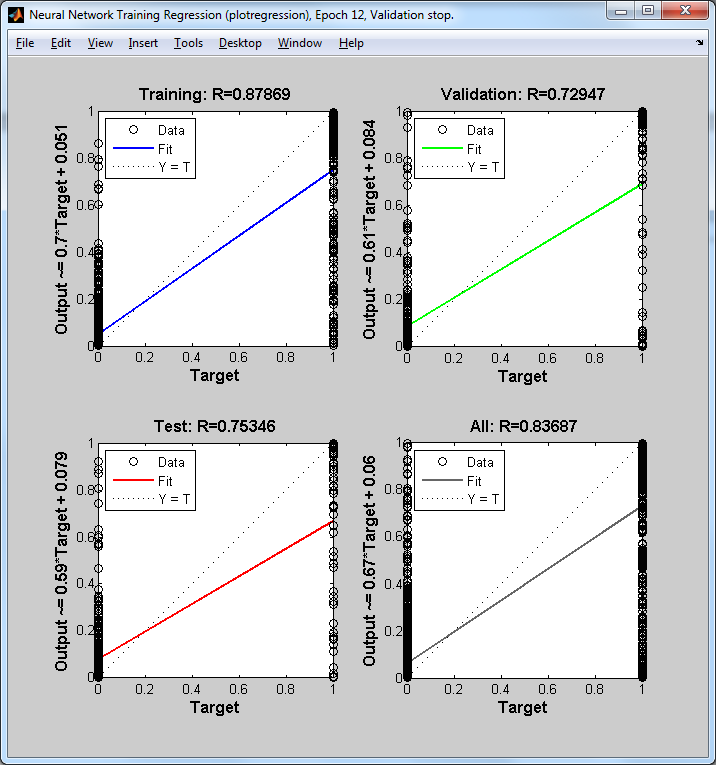


We begin with a 2 layer setting(1 hidden + 1 output), 100 epochs (learning rate) showing at a rate of 5 epochs with a time of 60.



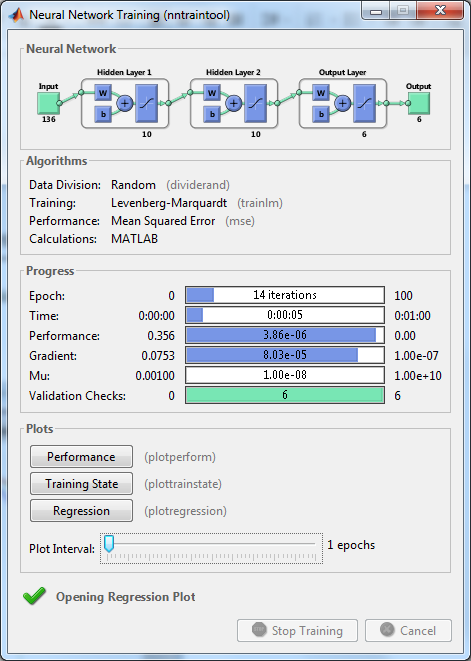


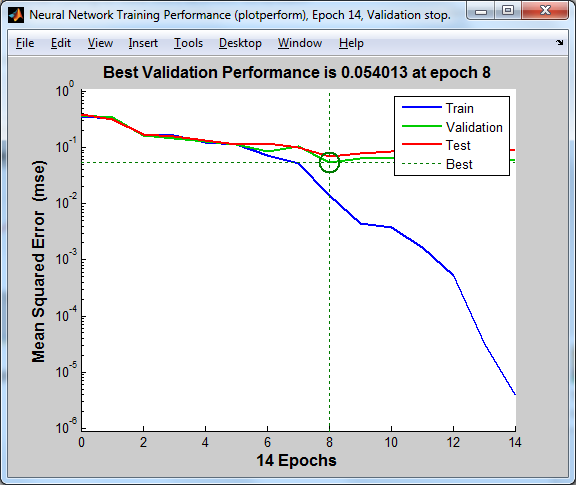




Regression plots here would not help as the data is meant for a classification activity.

Using 2 hidden layers gives a different best performance in validation:





To make better sense of the data, the output and targets were fed to a custom script to come up with some statistics(downloaded from <http://uk.mathworks.com/matlabcentral/fileexchange/46035-confusion-matrix--accuracy--precision--specificity--sensitivity--recall--f-score> ).

We first had to convert the outputs to a vector for it to work:

>>new\_output=convert1D(emotionLearner\_outputs)

>> stat = confusionmatStats(new\_output,y)

stat =

confusionMat: [6x6 double]

accuracy: 0.6190

sensitivity: [6x1 double]

specificity: [6x1 double]

precision: [6x1 double]

recall: [6x1 double]

Fscore: [6x1 double]

The confusion matrix recall and precision output is tabled below:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** | **recall** | **precision** |
| **1** | 16 | 3 | 0 | 0 | 1 | 0 | 80.00% | 34.04% |
| **2** | 6 | 55 | 0 | 1 | 1 | 0 | 87.30% | 91.67% |
| **3** | 7 | 0 | 48 | 0 | 2 | 0 | 84.21% | 70.59% |
| **4** | 12 | 1 | 16 | 112 | 0 | 1 | 78.87% | 99.12% |
| **5** | 6 | 0 | 1 | 0 | 52 | 0 | 88.14% | 89.66% |
| **6** | 0 | 1 | 3 | 0 | 2 | 101 | 94.39% | 99.02% |

The accuracy was found to be at 61.90%

## Implementing ANN via matlab command line scripting

To implement ANN using matlab command line scripting tool, the process would be similar:

1. prepare data
2. train
3. validate
4. test

However, the data needs to be prepared further as we do not have the automatic data splitting functions provided with nntool. We make use of N-fold splitting, and as suggested by the lab manual, 10-fold.

To implement 10-fold cross validation, We need to write a script in matlab to do so. The basic idea is that we first need to split the sample data into 3 parts:

1. Training data
2. Testing data
3. Validation data

A network is then created using the newff function, configured, and then trained with the train function using the training data and tested with the training data with the sim function. (ANNver5.m)

The data is then as before converted to a vector to be fed into the confusionmatStat function

Using this script, multiple configurations were carried out, giving different results:

Using the following parameters:

hidden\_layers=[10 10]; %number of neurons for each hidden layer

layer\_functions={'tansig' 'tansig' 'tansig'};

epochs = 500;

show = 25;

time = 200;

goal=0;

min\_grad=1e-07;

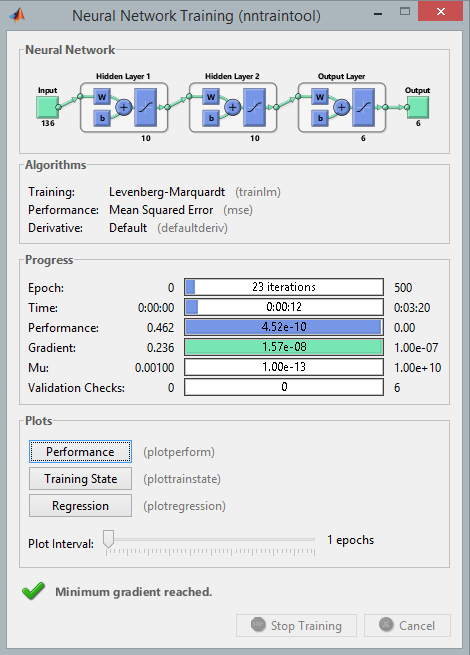
max\_fail=6;

mu = 0.001;

mu\_dec=0.1;

mu\_inc=10;

mu\_max=10000000000



The following confusion matrix was obtained:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |  | **recall** | **precision** |
| **1** | 25 | 8 | 4 | 1 | 11 | 1 |  | 50.00% | 48.08% |
| **2** | 11 | 32 | 3 | 1 | 4 | 6 |  | 56.14% | 61.54% |
| **3** | 5 | 3 | 35 | 10 | 6 | 7 |  | 53.03% | 56.45% |
| **4** | 1 | 3 | 5 | 89 | 1 | 3 |  | 87.25% | 82.41% |
| **5** | 5 | 4 | 8 | 1 | 38 | 1 |  | 66.67% | 54.29% |
| **6** | 5 | 2 | 7 | 6 | 10 | 66 |  | 68.75% | 78.57% |

Accuracy at 55.5%

## 

# Observations

*These are statements of what you witnessed throughout the lab while you were carrying out your procedure.  They must be true observations.*

# Data/Results

*Here is where you present your data - usually in a neat and organized data table.*

TODO: confusion matrix

# Calculations

*This section is where you show any mathematical calculations you needed to complete during this experiment.  This is an important section because it shows where you derived your final results.*

# Analysis

*Explain what happened in your experiment by demonstrating your understanding of the experimental process.*

# Conclusion

*Explain all the concepts and vocabulary that were used in the lab in detail.  This should be at least a ½ page statement that answers the questions posed in your purpose.  Avoid personal feelings and emotions.*

# References

1. Matlab help documentation
2. Lab Manual
3. G54MLE Lectures
4. Bishop Chapter 5
5. Duda and Hart
6. Coursera