Foundations of Intelligent Security Systems

Project 2

Neural Networks in Intrusion Detection Design

Project By

Dhivya Govindarajan
Bhakti Bhadrecha
Jaydeep Untwal

Executive Summary

Technology is evolving everyday, we have become heavily dependent on it for almost all our activities that were once high time consuming and involved physical presence, e.g, depositing checks in bank accounts, signing contracts, etc. but with this convenience, comes great responsibility. This has made us vulnerable to malicious and tech savvy criminals to get unauthorized access to our confidential information. Just as we had security guards protecting our apartments and neighborhood banks, we need virtual guards to protect our information online. This project aims to design and implement an intrusion detection system (the above mentioned virtual guard) to monitor and alert us when it notices any suspicious activities like unauthorized access, modification of protected files and user spoofing. This will be achieved by analyzing network data of known attacks and normal behavior, which will help us prevent the system from future attacks.

The human brain is a very complicated structure. One of the most miraculous features of the brain is to develop memories and recognize patterns. To record a pattern in the brain, it develops a neuron which, takes few electro-chemical signals obtained from our senses as input, analyzes the behavior and then fires another electro-chemical signal to let the body react to it. Similar to these neurons in the brain, a field of computer science simulates this behavior to better identify patterns. This is known as 'Neural Networks'. In this project, we aim to use Neural Networks to identify the intrusions and detect the intrusions whenever the system encounters.

Specification

In this project, we are to study the Neural Network algorithm in detail. The data files are same as what we used for the project 1 analysis. There will be two sets of the data, testing and training datasets. This project focusses on developing an Intrusion Detection system which will detect the anomaly attacks and misuse attacks based on neural networks. In the last project, we used decision tree based classification model. Due to high number of attributes, decision trees have a lot of limitations. The limitations of decision trees include suffering from over and under fitting, highly complicated decision

trees and time consuming to build the decision tree. The dataset we are using in the project has values in the range of -1 to 1. Neural networks work great for such type of data. Also, neural networks do not have an issue with high number of attributes. The graphs and timing charts shown below will provide a better visualizations of the results.

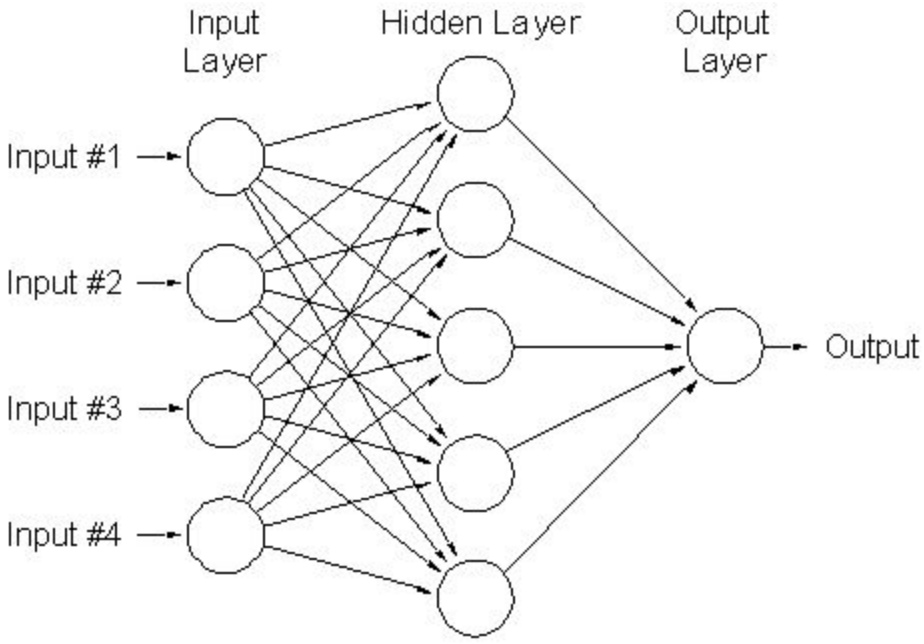
Methods and Techniques

This section describes the methods and techniques used in this project.

We used the Multi-layer perceptron architecture for our neural network system. We used various algorithms to train our network. This includes:

- Back Propagation
- Levenberg-Marquardt

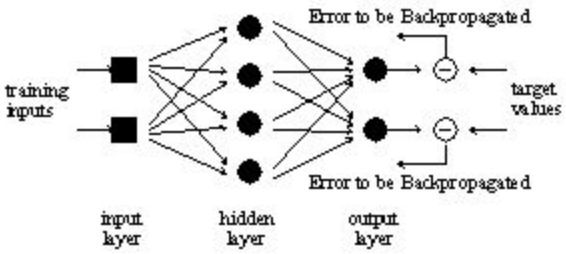
Multi-layer Perceptron:



Neural networks follow a layered architecture, it comprises of three main layers: Input layer, hidden layer and output layer. There are usually more than one hidden layer. This architecture works on layer-feed-layer basis. The output of one layer is used an input for another. The input layer has one or more edges connecting inward to it. These edges

have a weight in the range of [-1 to 1]. Each node in the input layer is then connected to one or more than one node in the hidden layer with a weight as a function of sum of the inputs to the node. Based on these inputs, the node in the hidden layer is connected to a node in the output layer with edges having biased weights and finally, the edges coming out of each node in the output layer are summed up to give out a final edge from the system. This system can be treated as a black box where several inputs are processed into one output.

Back Propagation:



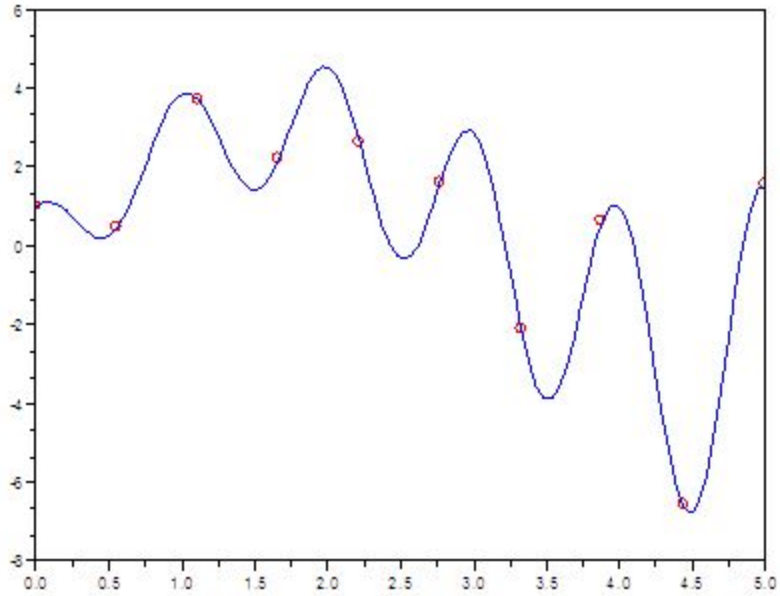
This algorithm aims to improve the generated model by minimizing the surface error. The error is calculated as a function of all the weights on the edges. The idea behind minimizing this error is to choose an outward edge with the steepest descent. This will ensure that we are moving along shorter distances to finally minimize the error. This error is propagated in backward direction to help improve the model.

This algorithm can be divided into two phases:

The propagation phase involves two steps, one forward propagation, where a simple multi-layered process is followed to generate output and second is the backward propagation where the errors are calculated for each neuron in the hidden layer.

The weight update phase involves two steps, where each edge weight is updated to the result of propagated output error and edge weight is set. Secondly, the ratio of the error gradient is subtracted from the edge weight.

Levenberg-Marquardt:



The Levenberg-Marquardt algorithm helps in various curve fitting problems that arise in multi class data. It is an iterative algorithm that builds up on an initial guess of a parameter vector. It is one of the fastest algorithms to train moderate-sized feedforward neural network. Matlab supports this algorithm using a library '*trainlm*'. Trainlm can train networks having weights, input and transfer function who have derivative functions.

We noticed that using 2 layers (default) and 10 neurons, we obtained the best results, this is described later in the results section.

The data files provided in the Project section of the mycourses are used for this project. It comprises of 23 data files out of which 22 are different kinds of attacks such as Back, Buffer Overflow, FTPWrite, GuessPassword, Imap, IPSweep, Land, LoadModule, MultiHop, Neptune, NMap, Perl, PHF, Pod, PortSweep, RootKit, Satan, Smurf, Spy, TearDrop, WarezClient and WarezMaster. Each file consists of 41 attributes and several data records. There are nine basic attributes: Duration, Protocol Type, Service Type, Status Flag, Total bytes sent to Destination & Source, etc. and 32 derived attributes which are

divided into three categories: Content, Timebased Traffic and Hostbased Traffic. We primarily used Matlab for our project as Matlab supports use of different Neural Network libraries. In order to do so, we had to preprocess the data to make it a valid input for the Matlab framework. Matlab accepts files for classification as two different files

- File 1: This file contains the data with attributes but not the class attribute
- File 2: This file contains only the class attribute

We wrote a script in 'R' to split the dataset into two: input_attacktype and output_attacktype. Also, as Matlab does not use attribute names, we had to manually remove header values using another 'R' script.

As Matlab does not interpret strings in the data, we converted the class values to numerical values. We maintained an alphabetical order to do the same, e.g, we had to consider 5 attack types, we converted the headers as follows:

- IPSweep -> 1
- Neptune -> 2
- Other -> 3
- PortSweep -> 4
- Satan -> 5
- Smurf -> 6

In other cases, where we only had to consider one particular attack, we treated the class values 'Attack' and 'Not Attack', e.g,

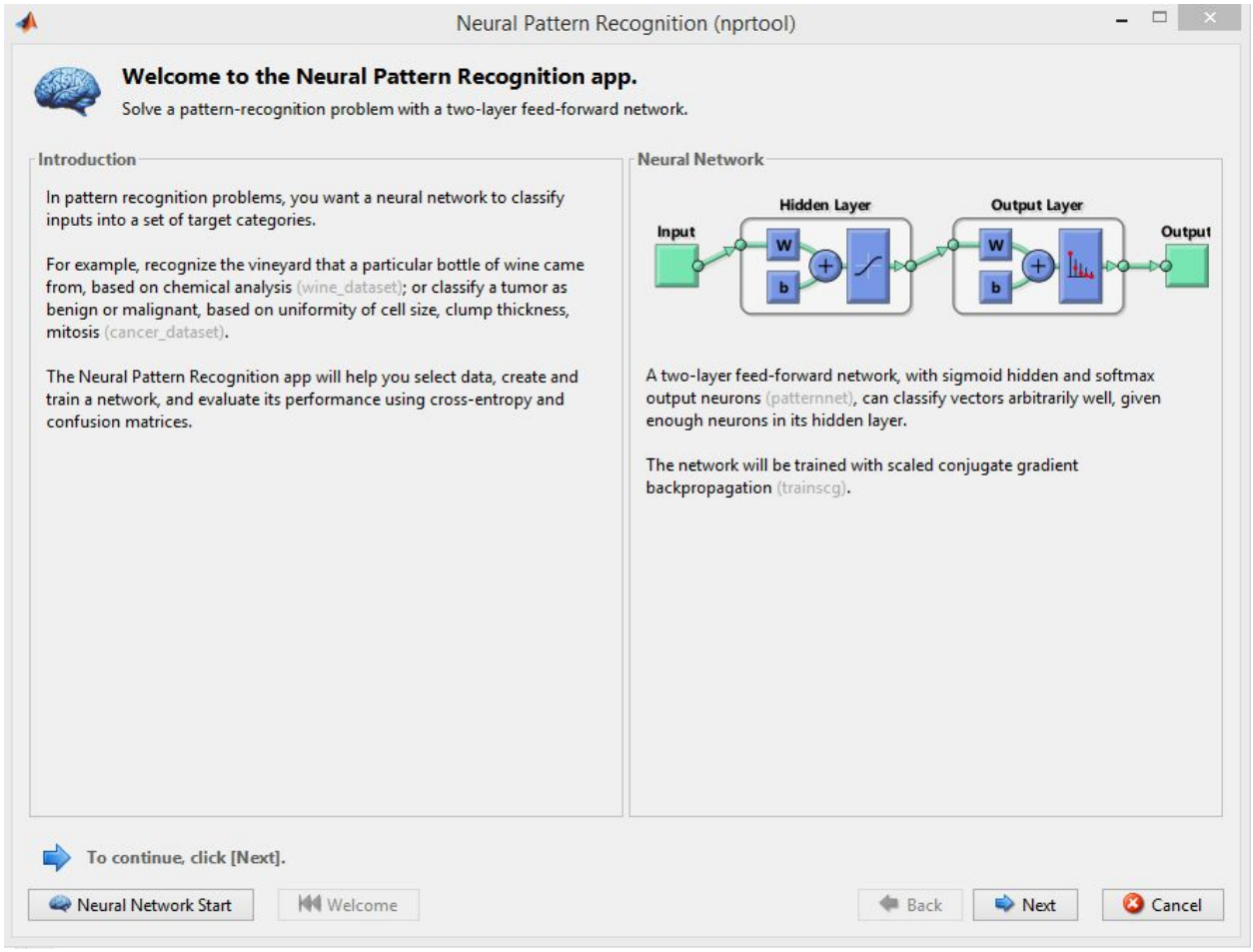
- Not Smurf Attack -> 1
- Smurf Attack -> 2

Matlab supports the inclusion of different libraries that are available, one package that we have used in this project is 'Neural Pattern Recognition tool' which allowed us to classify our data using GUI.

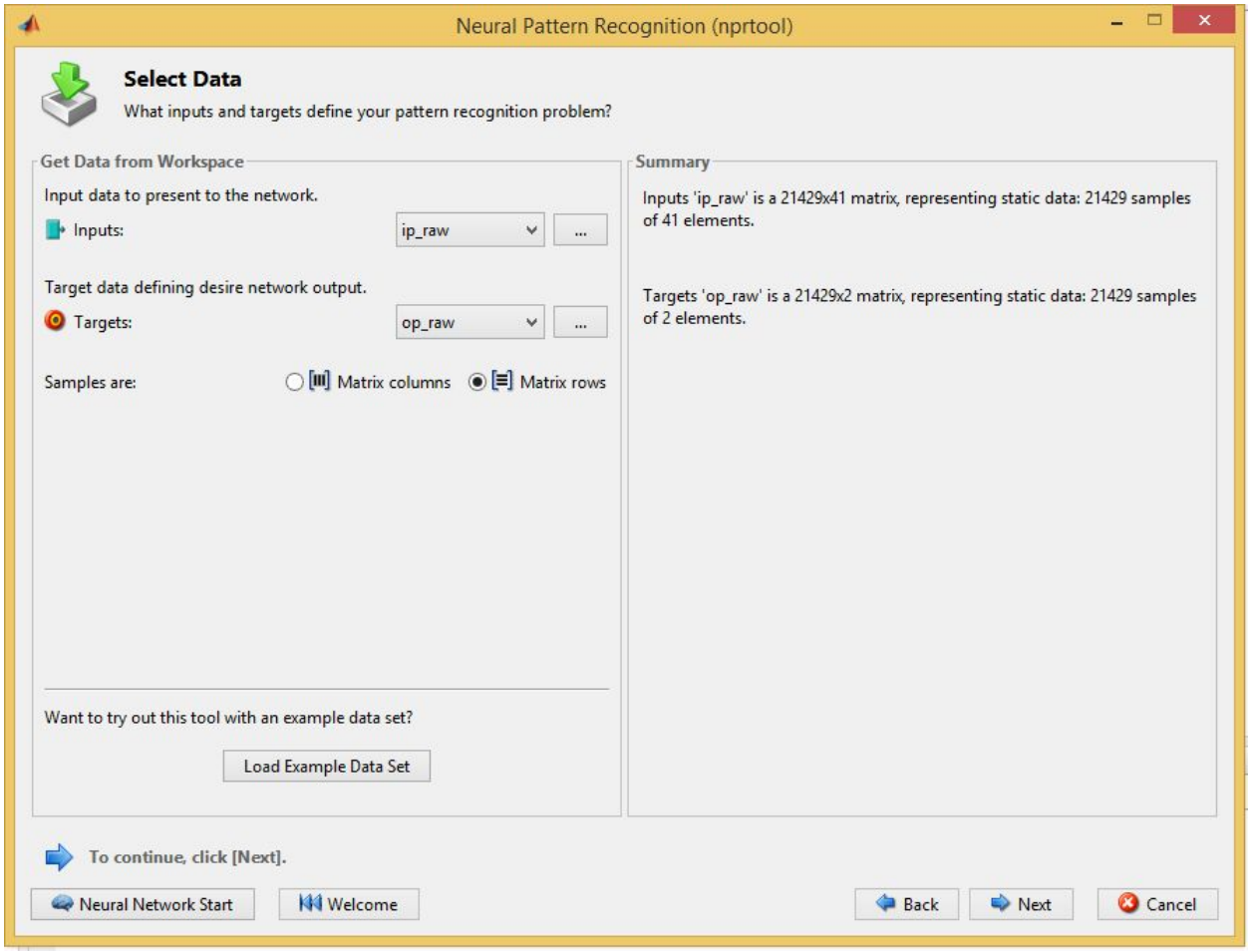
Implementation

As described in methods and techniques, we preprocessed our data to be used in the Neural Pattern Recognition tool in Matlab.

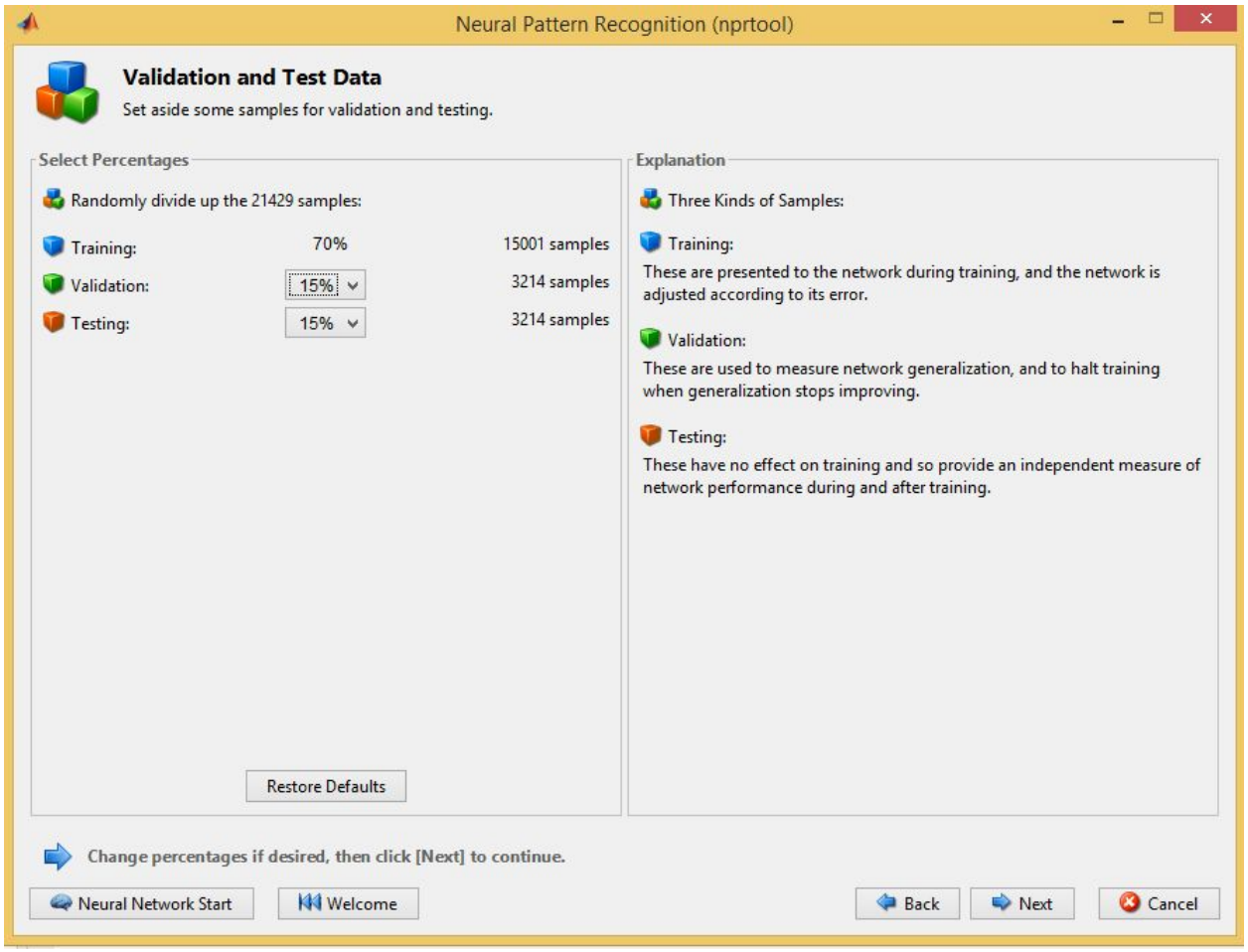
1. Open the NPR tool and you will see a screen as shown below



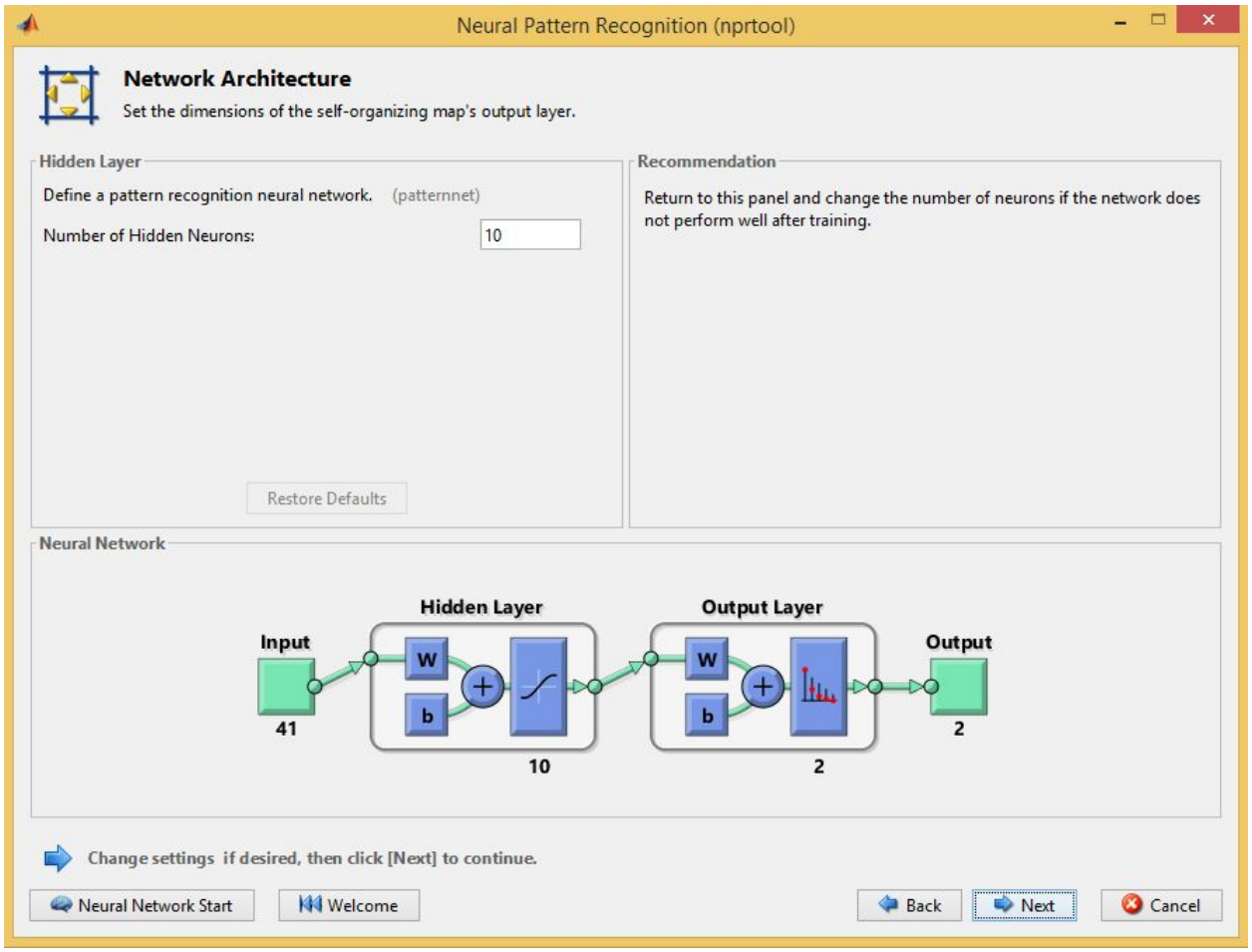
2. Select the input and the output data file in the workspace, you will also have to specify whether the data is in rows or columns



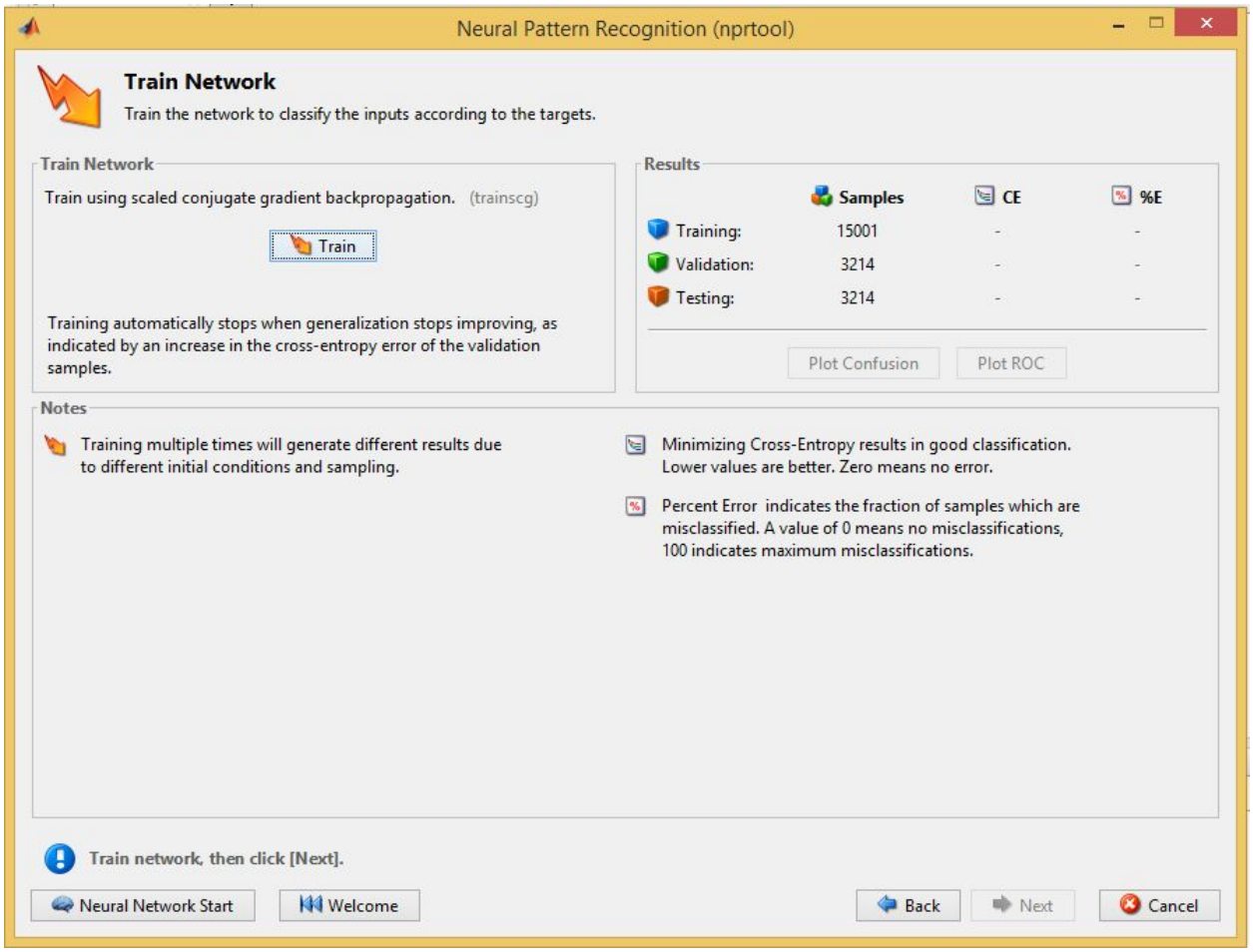
3. Select the percentage of data to be used for training, test and validation



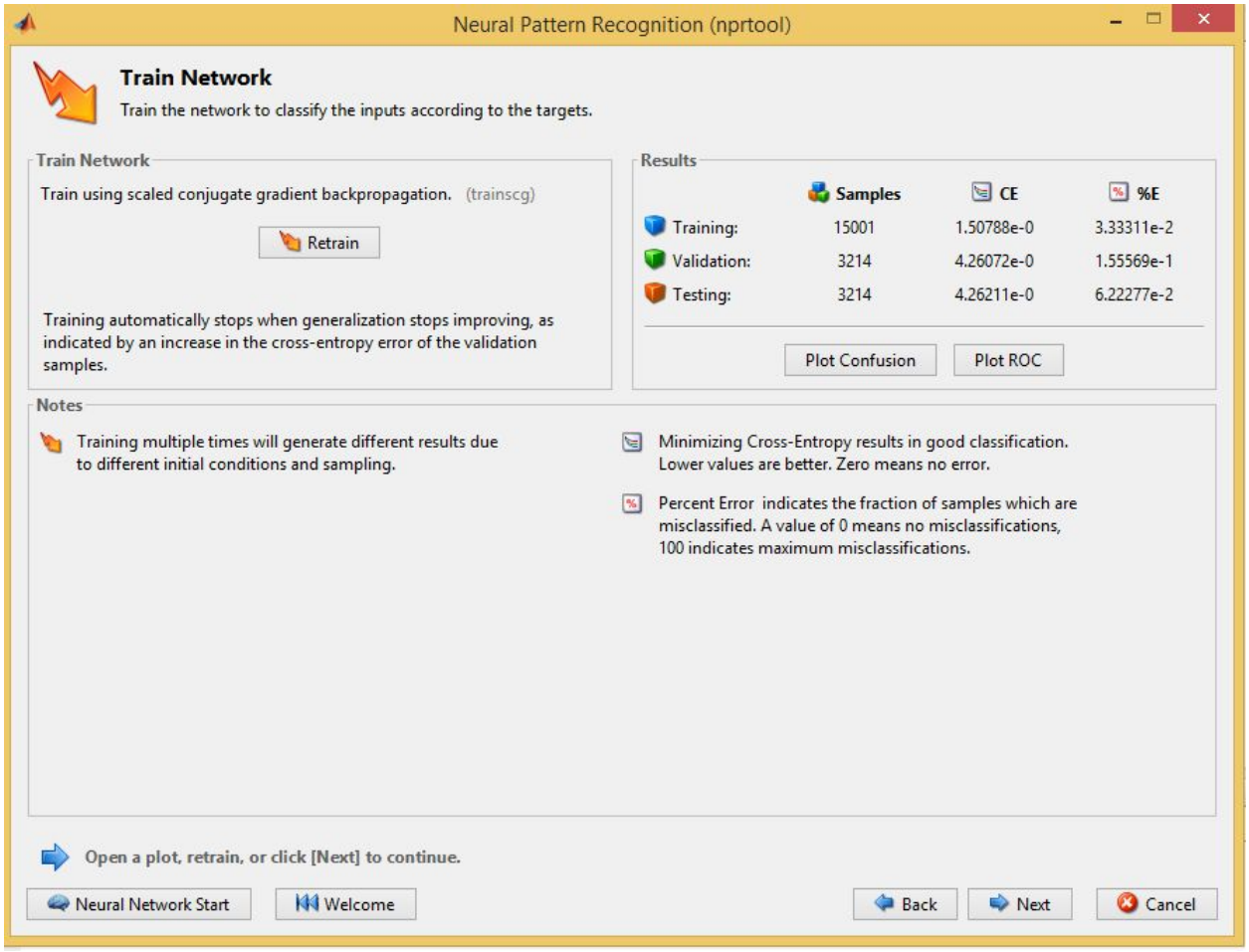
4. Select the number of neurons to be used in the hidden layer



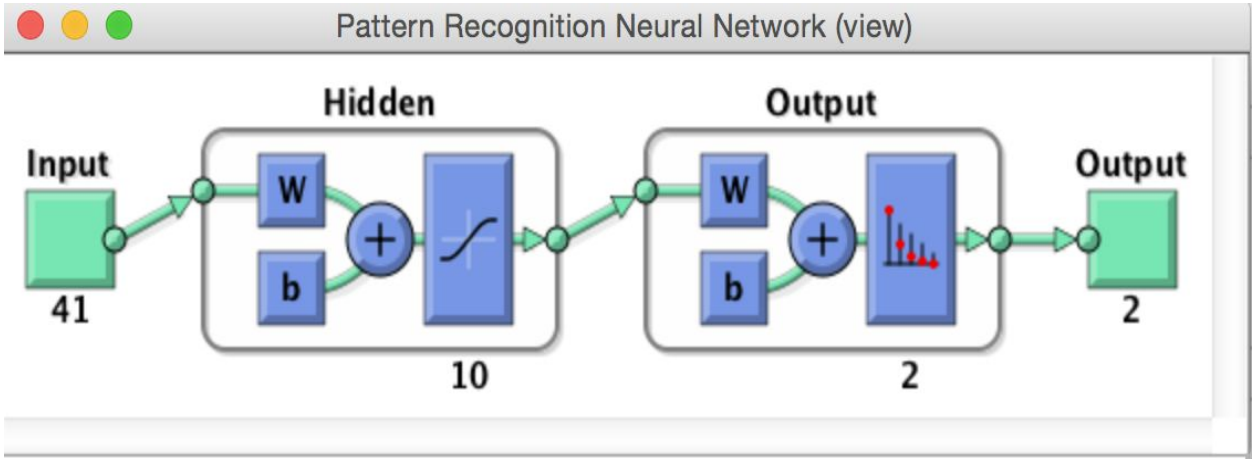
5. Click on the ‘train’ button to start training

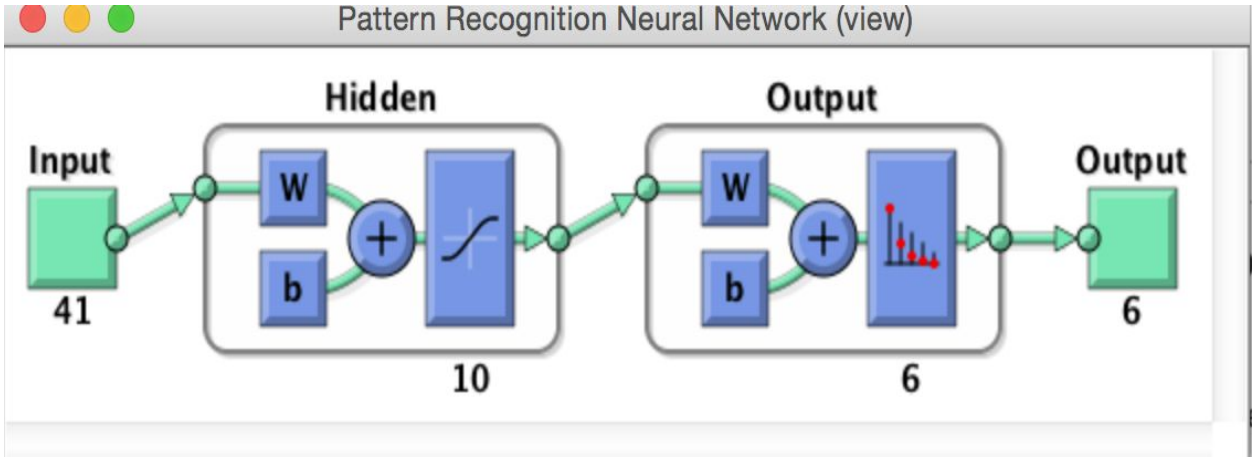


6. Once the training is complete, you will see the results and you can choose to plot the graphs or neural network for your current model



7. Here is a sample network with 2 and 6 output states





Results and Tests

Misuse Detection

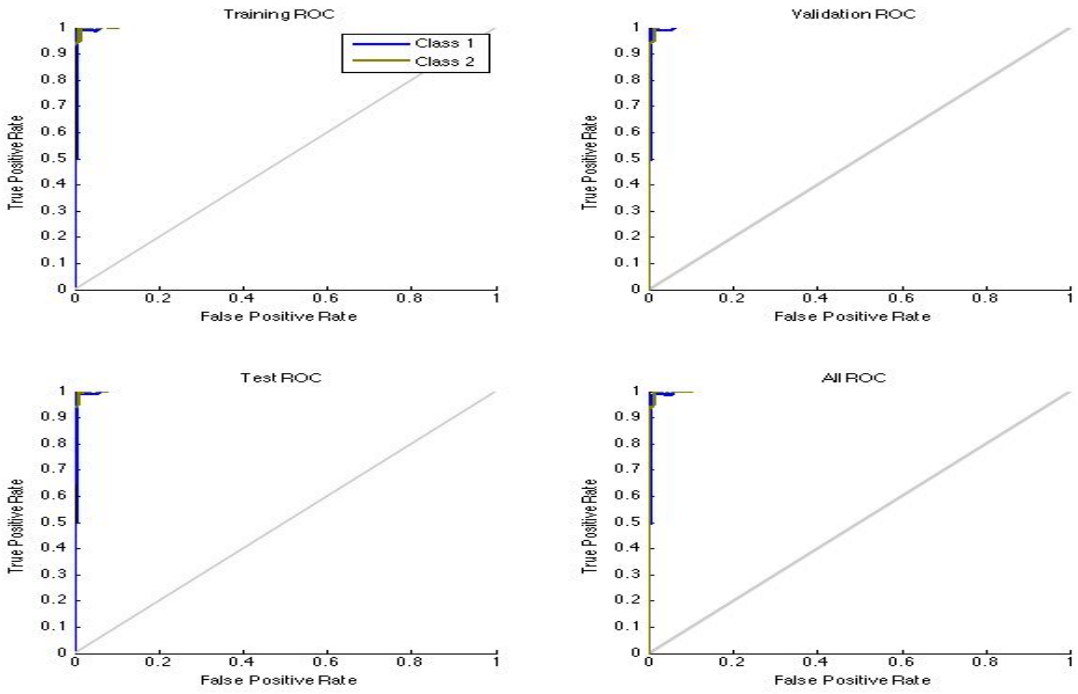
This section includes the analysis of misuse and anomaly detection. The misused include cases for : IPSweep, Neptune, PortSweep, Satan, Smurf and special case for all the attacks mentioned above.

IPSweep Analysis

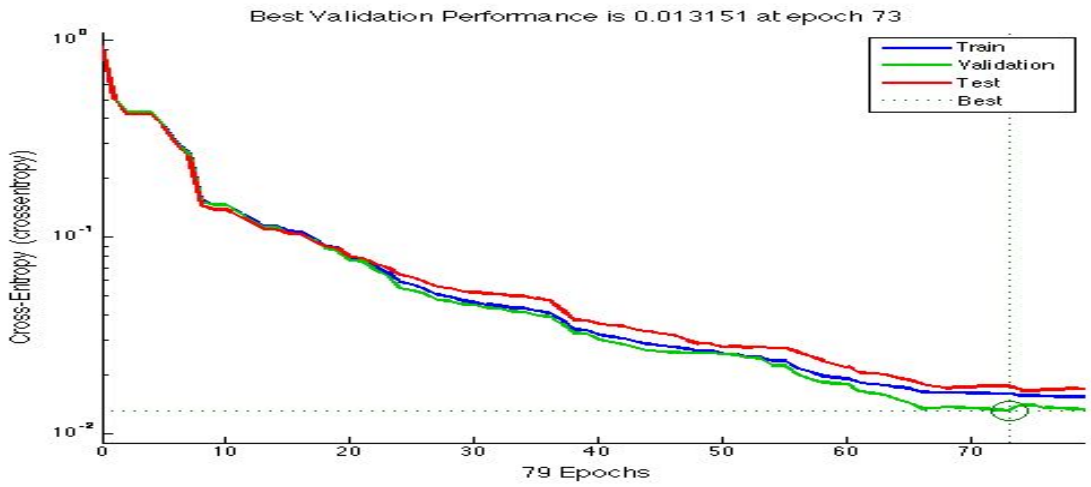
This section includes the analysis of the IPSweep attack. The dataset was prepared in such way that the data contains the records of the IPSweep attack only; the rest of the attack types are tagged as Not IPSweep. Therefore, there are two output states: IPSweep and Not IPSweep. Since, the MatLab cannot process/interpret the strings, the class values are converted to binary values: IPSweep is converted to the value 1 and Not IPSweep is converted to the value 2. The following is the confusion matrix of the IPSweep analysis.

In the confusion matrix, as mentioned before, the class values are converted to nominal values as: *Value 1 = IPSweep state Value 2 = Not IPSweep state*. The number of neurons used for the neural network model are 10. The dataset is split into two : training and testing sets. The confusion matrix for the training dataset implies that 99.6% of the records are trained correctly. The 29 records of the IPsweep, which is 0.2% of the dataset, are mis-trained as the Not IPSweep and the 33 records of the Not IPSweep, which is 0.2% of the dataset, is mistrained as IPSweep. Similarly, the validation confusion matrix came up with 99.7% accuracy, misclassifying only 2 instances of the IPSweep as Not IPSweep and 8 instances of Not IPSweep as IPSweep. The test confusion matrix resulted in 99.5% classification accuracy, misclassifying 9 instances of the IPSweep as Not IPSweep and 7 instances of the Not IPSweep as IPSweep. The overall accuracy of the model was 99.6% with very negligible amount of misclassification, 0.2%.

The below is the receiver operating characteristic curve. The curve defines the false positive and true positive rate. It could be interpreted that the more the curve is to the top and the left side of the plot, the better and accurate is the classification model. From the plot below, we can see that the curve almost reaches the value of 1 which is 100%.



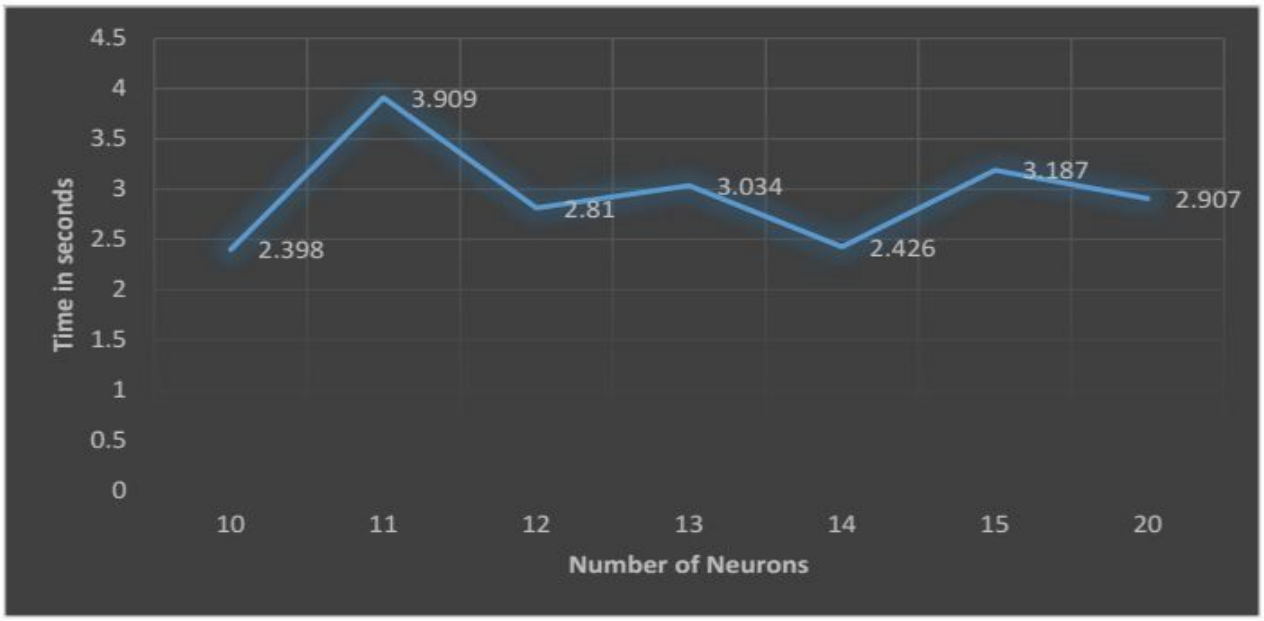
The below is the graph of Epochs vs Cross-entropy which gives the overall performance of the neural network's training, validation and test cases. Epoch is the single pass through the training set and the testing set. They are the number of iterations.



Since we used the matlab's built-in package for the modeling, it was complex to track the memory consumption of the model alone. however, the 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

Maximum possible array: 12552 MB (1.316e+10 bytes) *
Memory available for all arrays: 12552 MB (1.316e+10 bytes) *
Memory used by MATLAB: 1060 MB (1.111e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The graph below is the time consumption of the neural network recorded by the matlab using the Profiler function and is recorded and plotted using the Excel. The graph is plotted against the different number of neurons vs time in seconds.

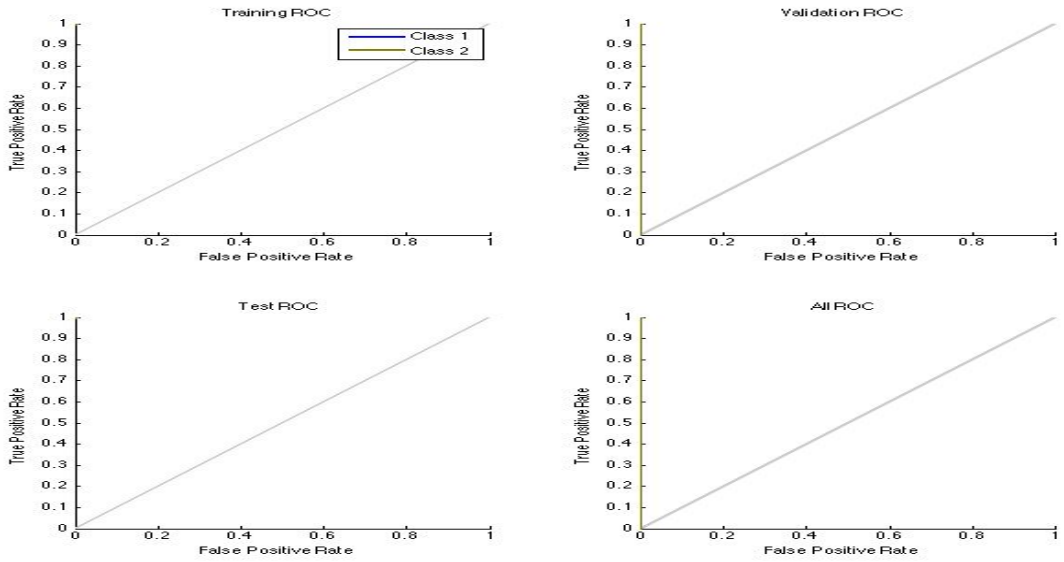


Neptune Analysis

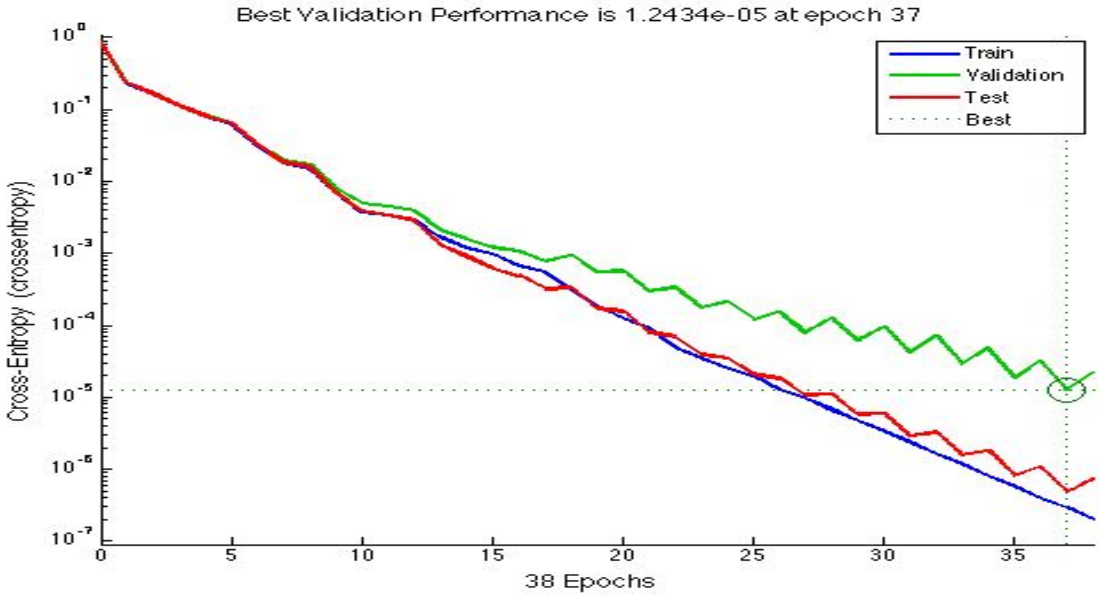
This section includes the analysis of the Neptune attack types. The dataset consists of two output states: Neptune and Not Neptune. The dataset is split into two sets : the training and the testing sets.The following is the confusion matrix of the Neptune neural network model.

In the confusion matrix, the class values are converted to nominal values as the following: *Value 1 = Neptune* *Value 2 = Not Neptune*. The number of neurons used for this neural network model are 10. The confusion matrix for the training dataset implies that 100% of the records are trained correctly with 0% of misclassification. The validation confusion matrix resulted with 100% accuracy. The test confusion matrix resulted in 100% classification accuracy, classifying all the data instances correctly. The overall accuracy of the model was 100%. There was not any necessity to change the number of neurons as the accuracy was 100%.

The following is the receiver operating characteristic curve. The curve defines the false positive and true positive rate. We can interpret that the model is accurate as the curve almost reaches the value of 1 which is 100%.



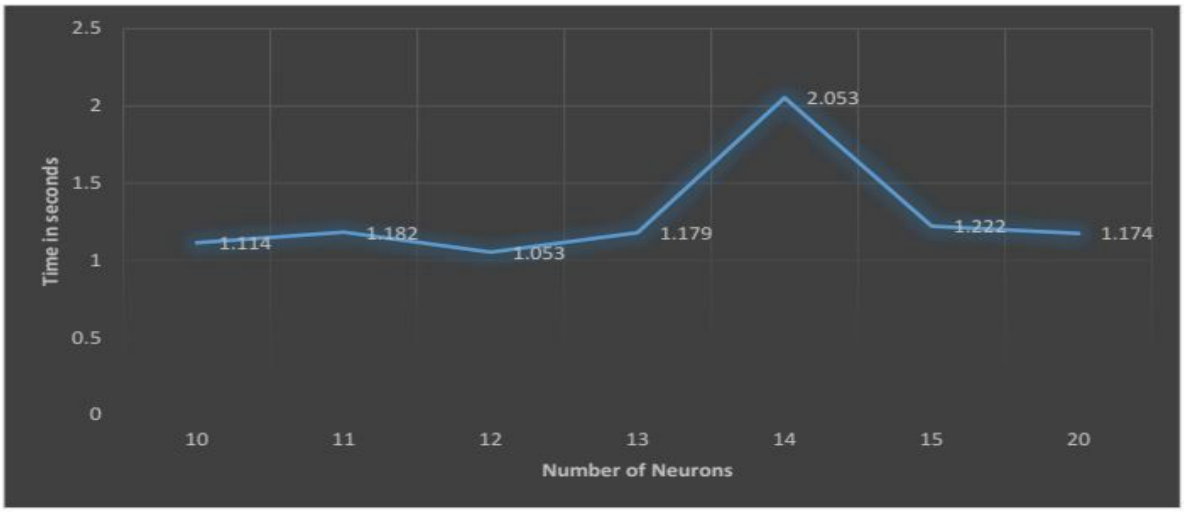
The below is the graph of Epochs vs Cross-entropy which gives the overall performance of the neural network's training, validation and test cases.



The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

Maximum possible array: 12568 MB (1.318e+10 bytes) *
Memory available for all arrays: 12568 MB (1.318e+10 bytes) *
Memory used by MATLAB: 1048 MB (1.099e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The following graph is the time consumption of the neural network for the Neptune attack.

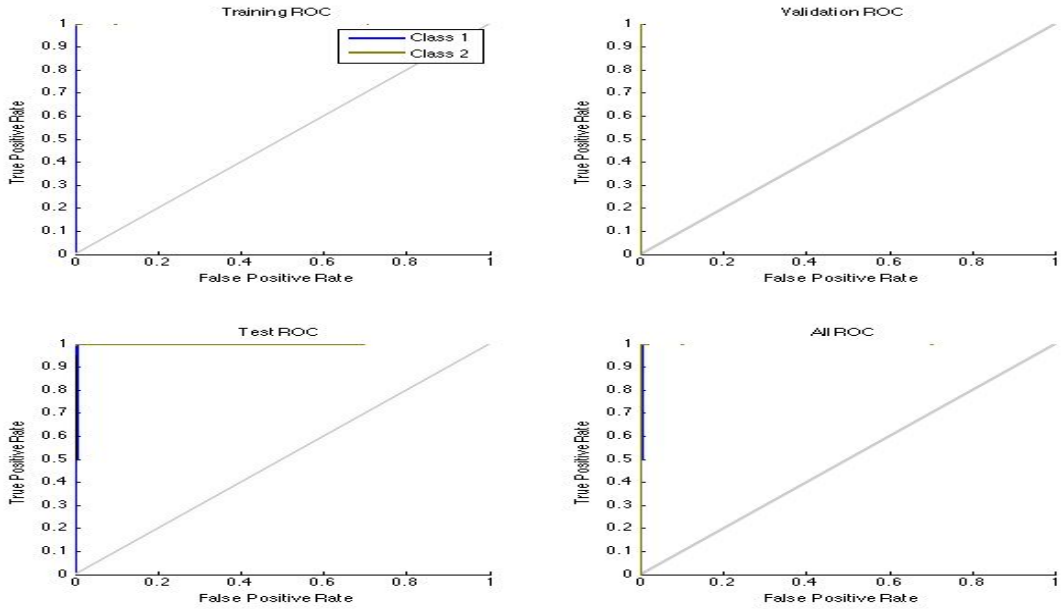


PortSweep

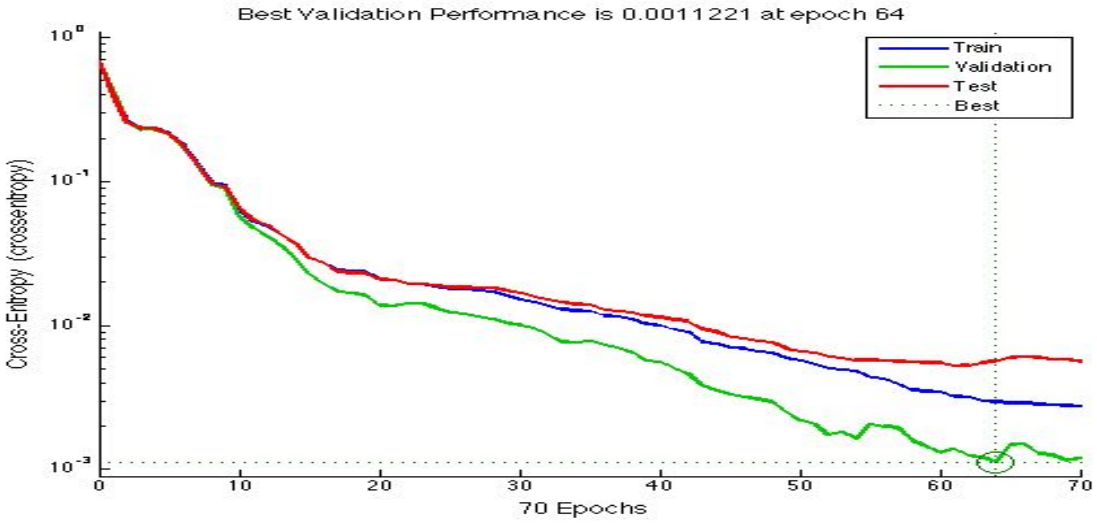
This section includes the analysis of the PortSweep attack data records. The dataset consists of two output states: PortSweep and Not PortSweep. The following is the confusion matrix of the PortSweep neural network model.

In the confusion matrix, the class values are converted to nominal values as the following: *Value 1 = Not PortSweep* *Value 2 = PortSweep*. The number of neurons used for this neural network model are 10. The confusion matrix for the training dataset resulted in 100% accuracy with 0% of misclassification. The validation confusion matrix implies that all the data records are classified correctly with 100% accuracy. The test confusion matrix resulted in 99.9% classification accuracy with 3 instances of Not PortSweep misclassified as PortSweep. The overall accuracy of the model was 100%. There was not any necessity to change the number of neurons as the accuracy was 100% by correctly classifying 18465 instances of the Not PortSweep and 2954 instances of the PortSweep class.

The following is the receiver operating characteristic curve. The curve with the false positive and true positive shows that the model is accurate as it reaches the value of 1.



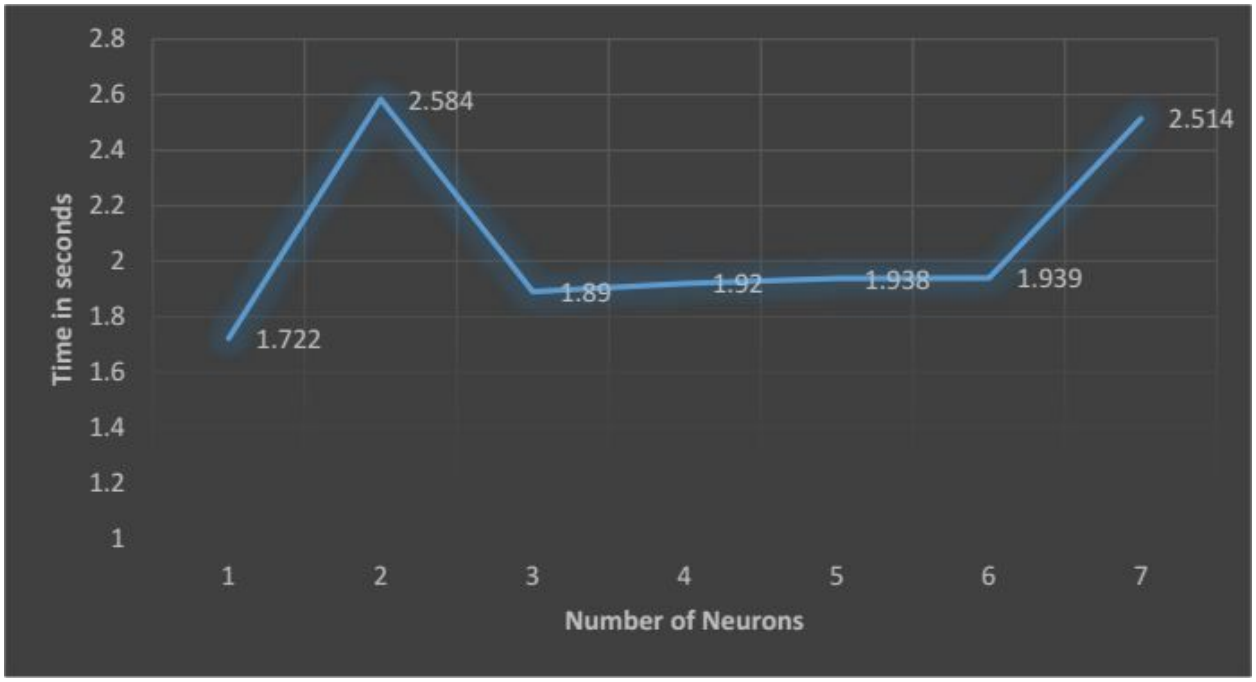
The graph of Epochs vs Cross-entropy which gives the best validation performance of the neural network by plotting number of epochs vs crossentrophy. In the case of PortSweep, the best validation performance occurred at the epoch 64th iteration.



The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

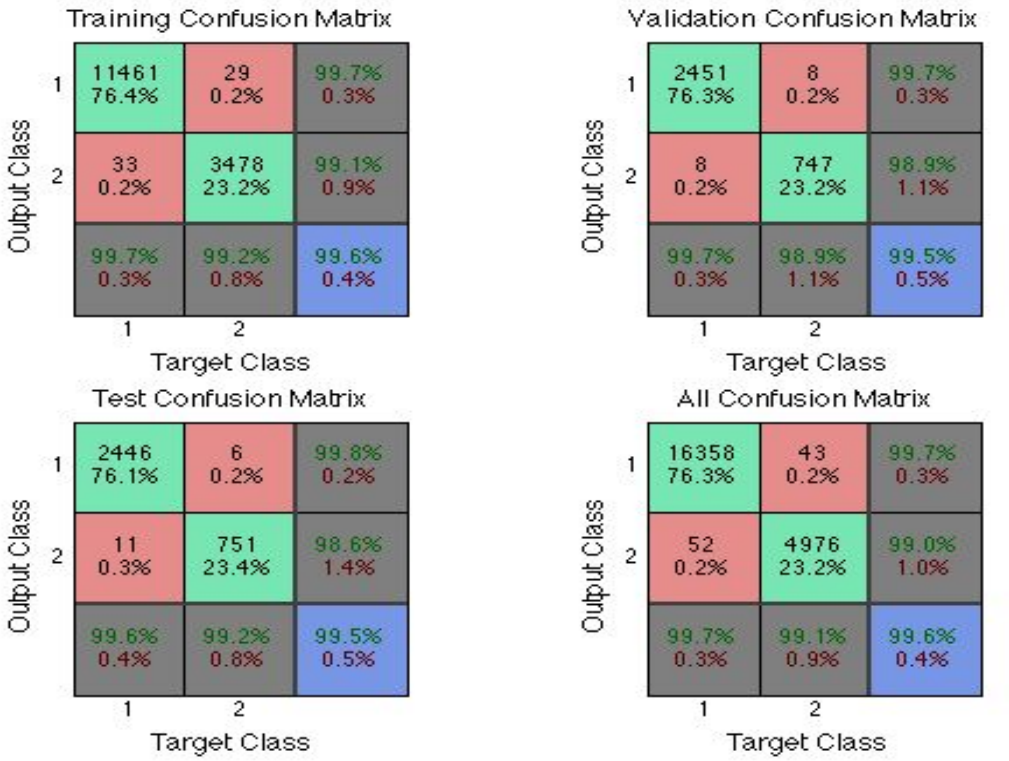
Maximum possible array: 12527 MB (1.314e+10 bytes) *
Memory available for all arrays: 12527 MB (1.314e+10 bytes) *
Memory used by MATLAB: 1065 MB (1.116e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The following graph is the time consumption of the neural network for the PortSweep attack.

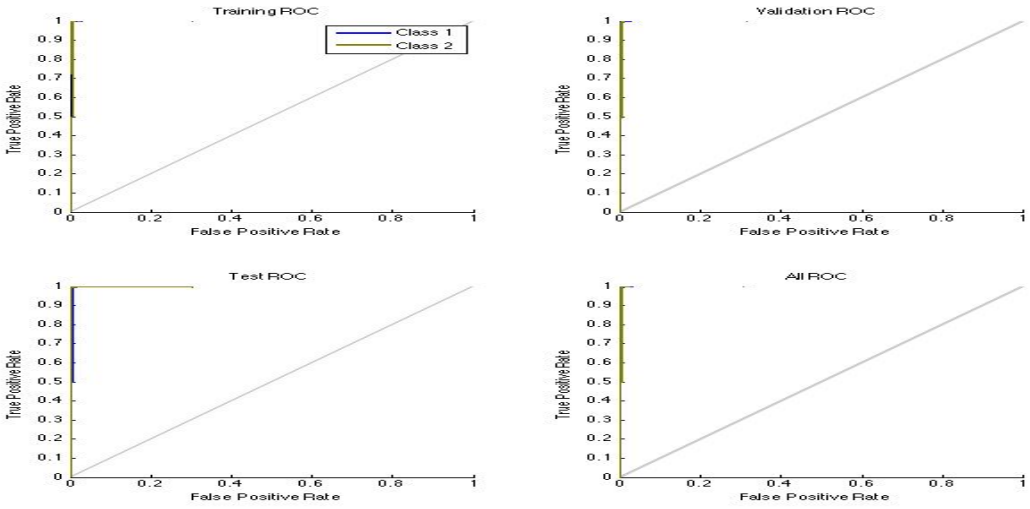


Satan Analysis

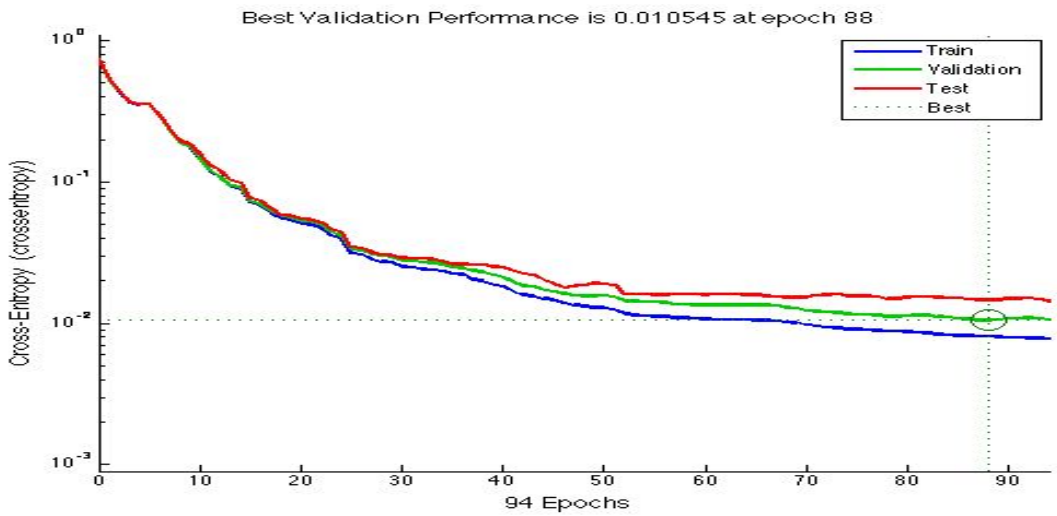
This section is the analysis of the Satan attack type. The dataset consists of two output states: Not Satan and Satan. The data is split into training and testing sets. The class values are converted to nominal values as the following: *Value 1 = Not Satan 2 = Satan*. The number of neurons used for this neural network model are 10. The confusion matrix for the training dataset resulted in 99.6% accuracy with 29 instances of Not Satan classified as Satan of 0.2% misclassification. The validation confusion matrix accuracy is 99.5% by misclassifying 8 instances of the Not Satan as Satan and misclassifying 8 instances of Satan as Not Satan. The test confusion matrix resulted in 99.5% classification accuracy with 6 instances of Satan misclassified as Satan and 11 instances of Satan misclassified as Not Satan. The overall accuracy of the model was 99.6% with 0.3% misclassification of Not Satan and 1% misclassification of Satan. The following is the confusion matrix of the Satan attack neural network model.



The following is the receiver operating characteristic curve where the curve of the false positive and true positive reaches 1 and implies that the model built is accurate.



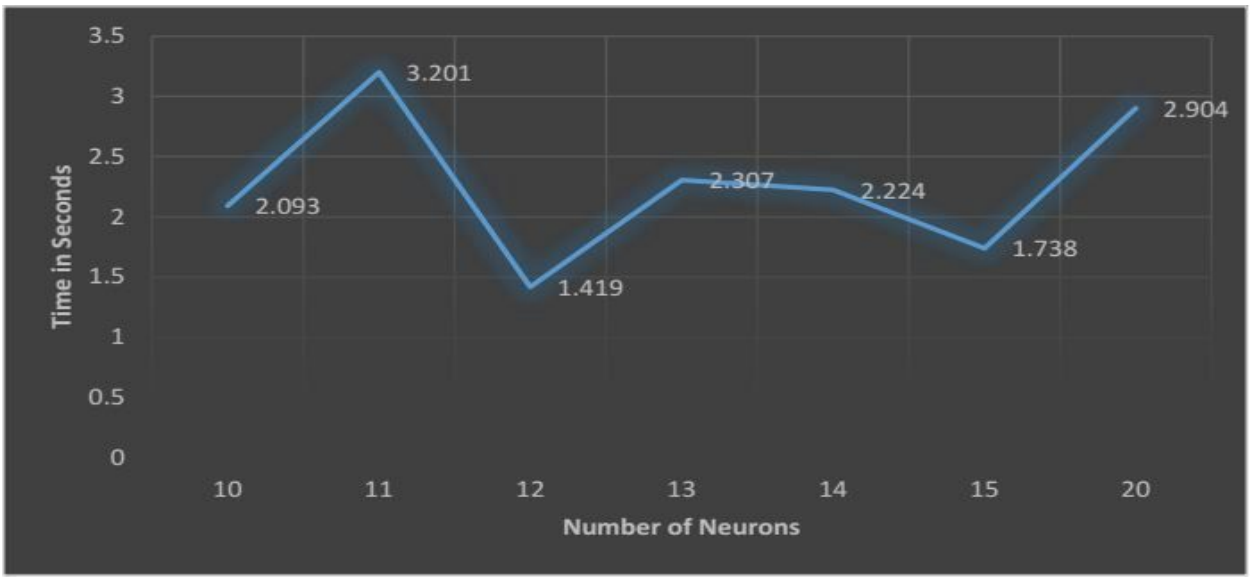
The following graph gives the best validation performance of the neural network by plotting number of epochs vs crossentrophy. The best validation performance of the Satan occurred at the epoch 88.



The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

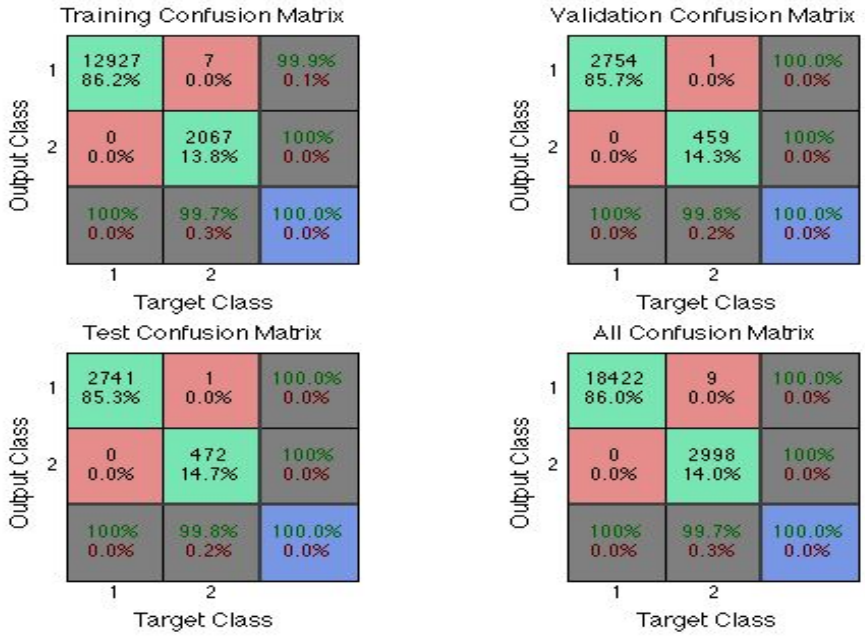
Maximum possible array: 12567 MB (1.318e+10 bytes) *
Memory available for all arrays: 12567 MB (1.318e+10 bytes) *
Memory used by MATLAB: 1054 MB (1.106e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

This is the time consumption graph of the neural network model for the Satan and Not Satan attacks dataset.

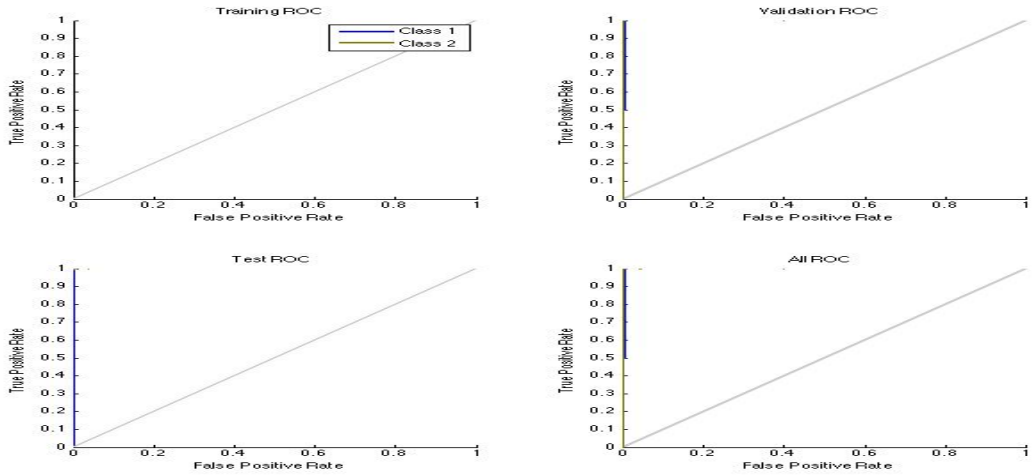


Smurf Analysis

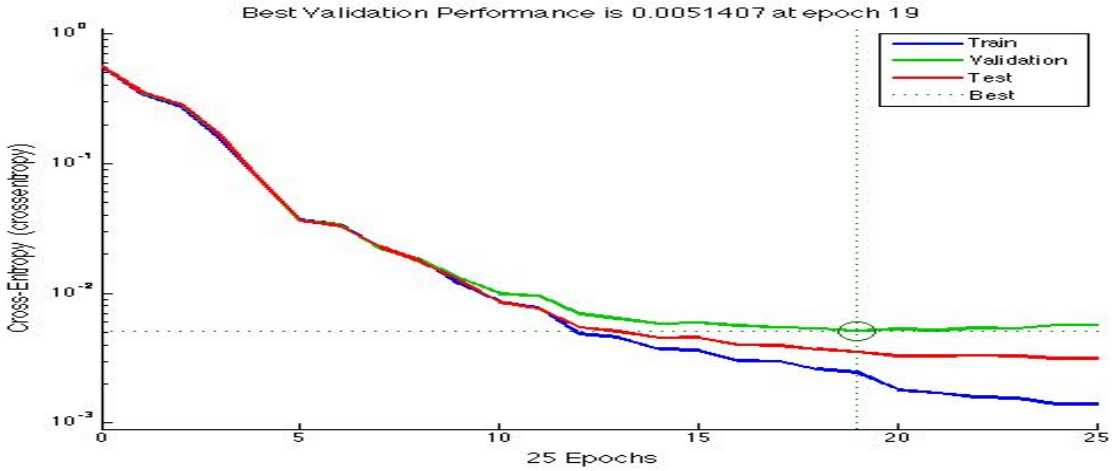
This section includes the analysis of the Smurf attack dataset. The dataset consists of two output states: Smurf and Not Smurf. The class values are converted from string to nominal values as the MatLab cannot interpret strings: *Value 1 = Not Smurf Value 2 = Smurf*. The number of neurons used for this neural network model are 10. The confusion matrix for the training dataset resulted in 100% accuracy with 0% of misclassification. The validation confusion matrix implies that all the data records are classified correctly with 100% accuracy. The test confusion matrix resulted in 100% classification accuracy. The overall accuracy of the model was 100% with zero misclassification rate. Every instances of the dataset are classified correctly. The confusion matrix is given below.



The graph following is the receiver operating characteristic curve where the curve of the false positive and true positive reaches 1 and implies that the model built is accurate.



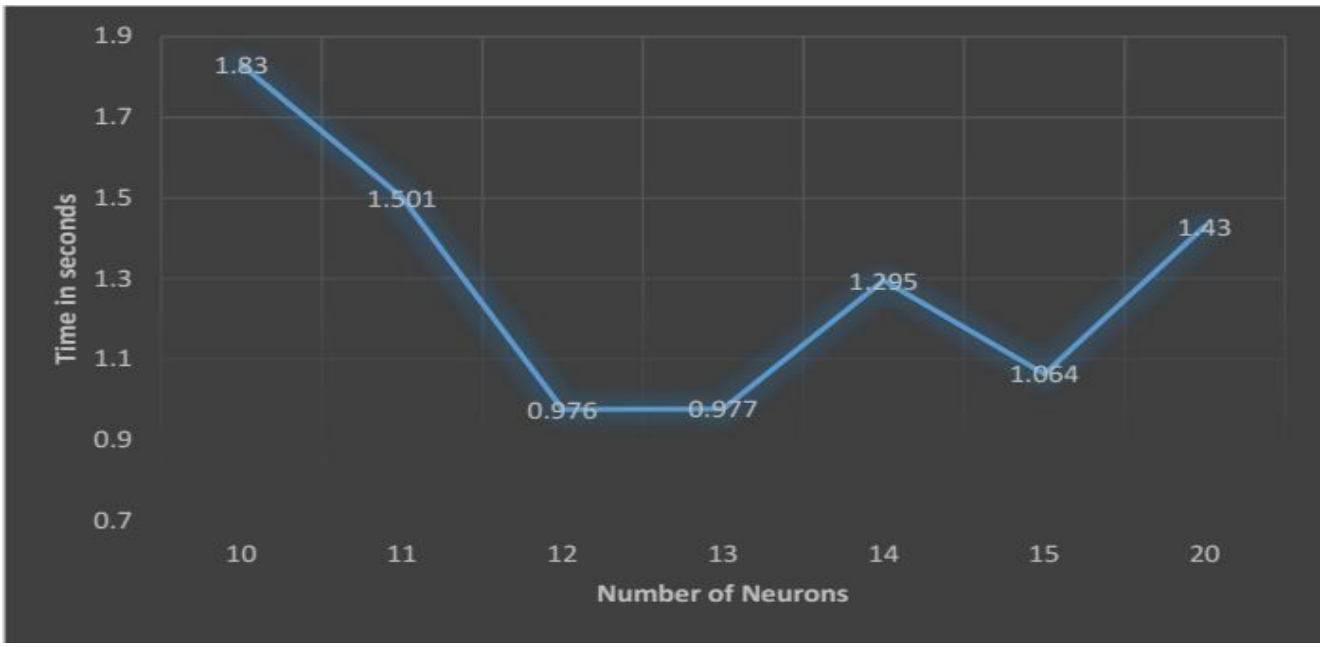
The graph below gives the best validation performance of the Smurf dataset which occurred at the epoch 19.



The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

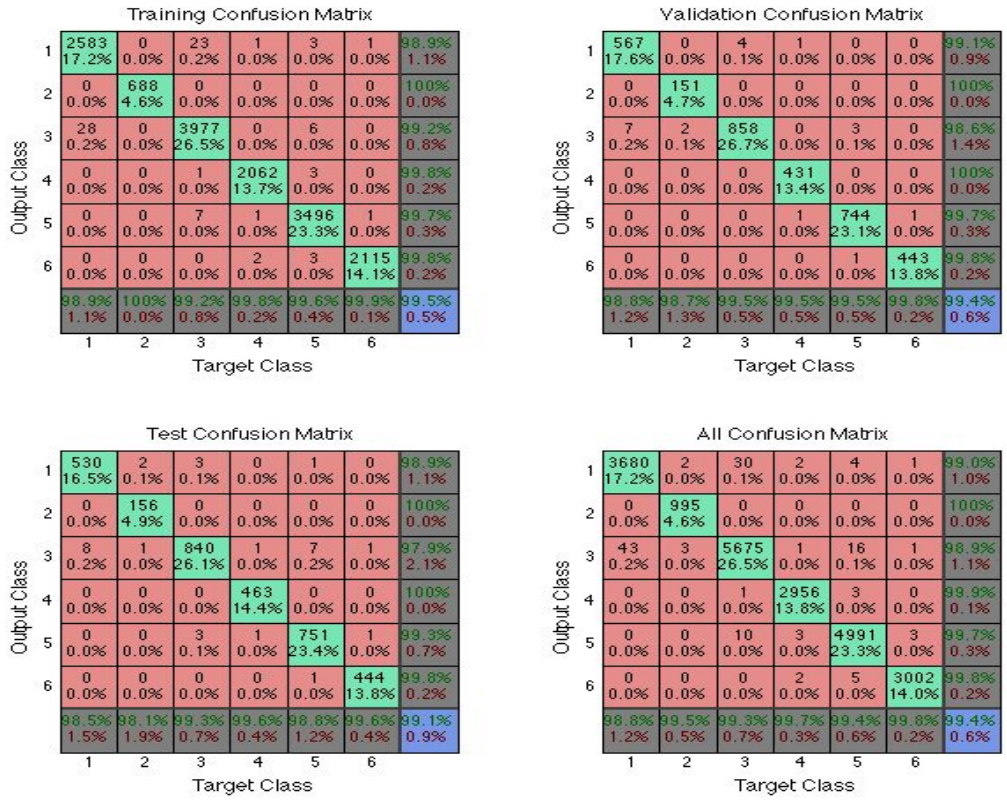
Maximum possible array: 12556 MB (1.318e+10 bytes) *
Memory available for all arrays: 12556 MB (1.318e+10 bytes) *
Memory used by MATLAB: 1060 MB (1.106e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The time consumed by the neural network model for the Smurf dataset is recorded and plotted for the better understanding of the NN efficiency. The graph is given below.

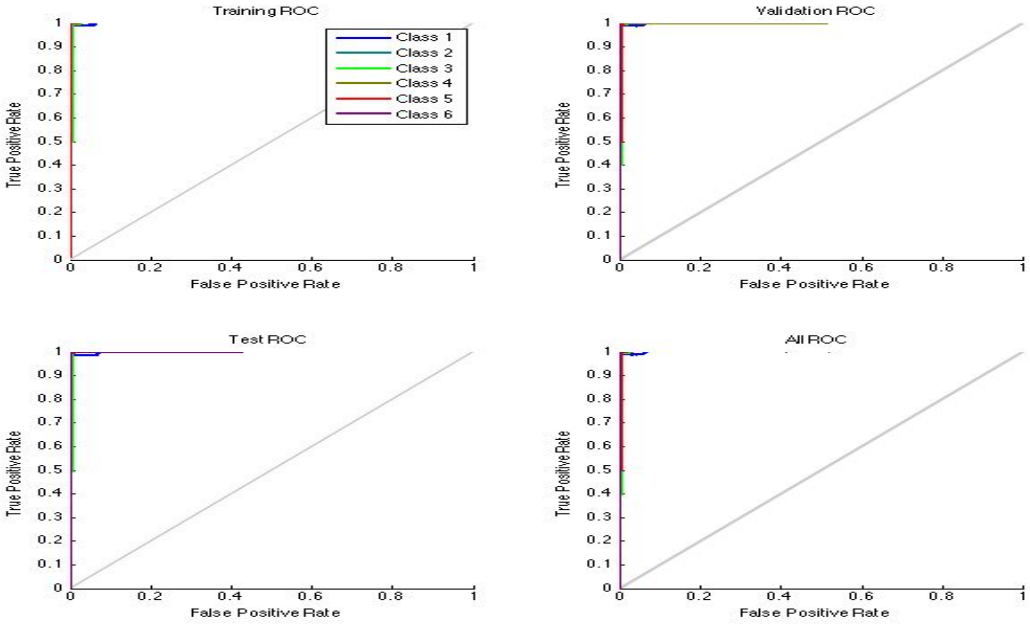


Special Case

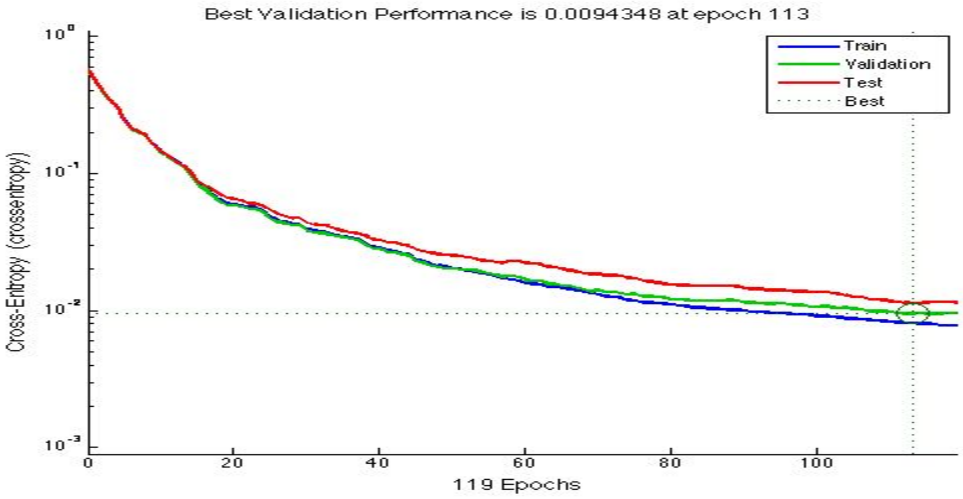
This section explains the analysis of the special case, which includes five attack types: IPSweep, Neptune, PortSweep, Satan, Smurf and the rest of the data records are tagged as other. This model consists of 6 states: IPSweep state, Neptune state, PortSweep state, Satan state, Smurf state and Other state. The class values are converted from string to nominal values as the MatLab cannot interpret strings: *Value 1 = IPSweep Value 2 = Neptune Value 3 = Other Value 4= PortSweep Value 5 = Satan Value 6 = Smurf*. The confusion matrix for the training dataset resulted in 99.5% accuracy with 23 instances of IPSweep misclassified as PortSweep, 1 instance of IPSweep misclassified as Satan, 3 instances of IPSweep misclassified as Smurf and 1 instance of IPSweep misclassified as Other. Similarly, all the other misclassifications from the training confusion matrix can be interpreted. The validation confusion matrix implies that 99.4% of the data records are classified correctly with IPSweep being misclassified 0.9%, Neptune classified 100% correctly, 1.4% misclassification of the PortSweep, 100% correct classification of Satan, 0.3% misclassification of Smurf and 0.2% misclassification of the other type. The test confusion matrix resulted in 99.1% classification accuracy rate and the misclassification rate can be interpreted as same as the training and validation confusion matrix. The overall accuracy of the model was 99.4% rate with 0.6% misclassification rate. The confusion matrix for the special case is given below.



The graph following is the receiver operating characteristic curve.



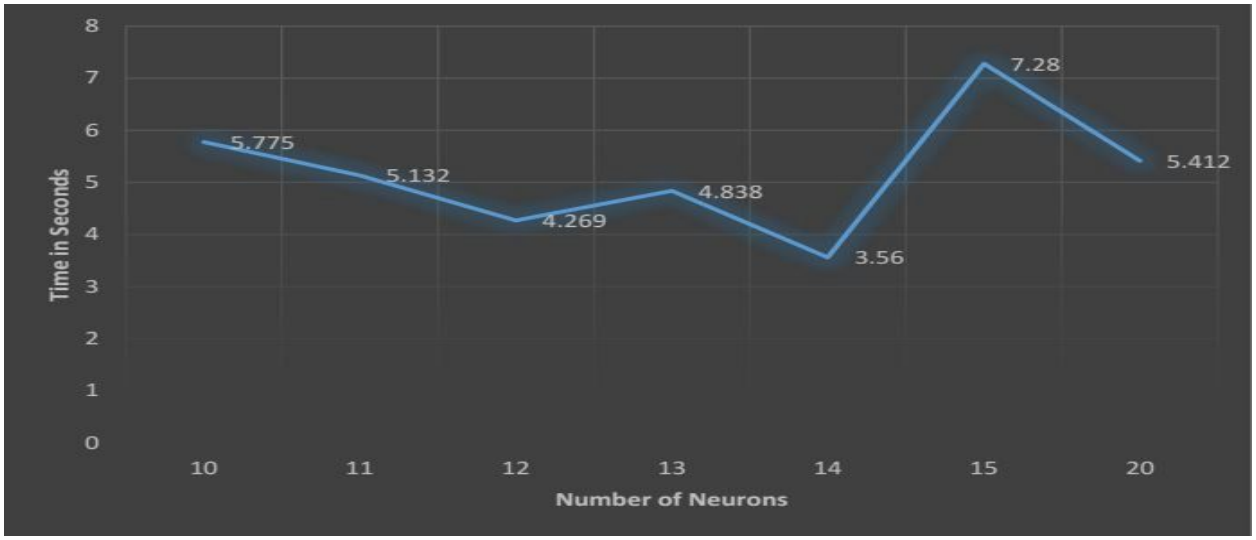
The graph below gives the best validation performance of the Smurf dataset which occurred at the epoch 113.



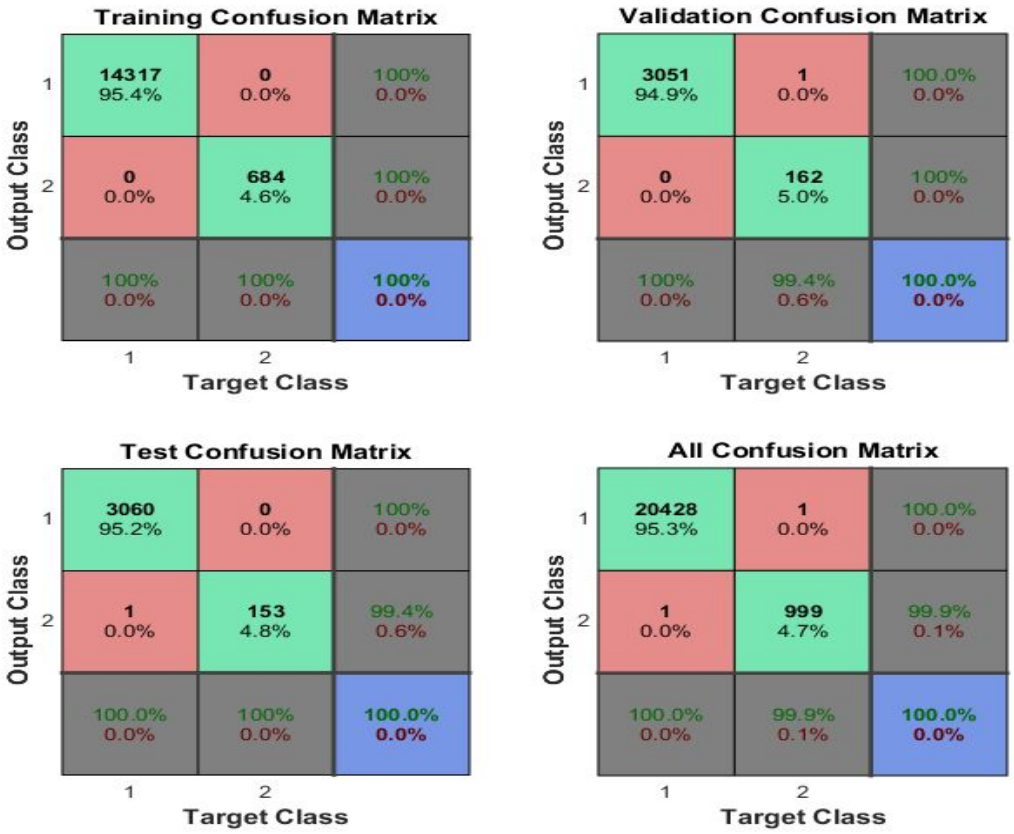
The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

Maximum possible array: 12555 MB (1.316e+10 bytes) *
Memory available for all arrays: 12555 MB (1.316e+10 bytes) *
Memory used by MATLAB: 1071 MB (1.123e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The time consumption graph of the NN model for the special case is given below.

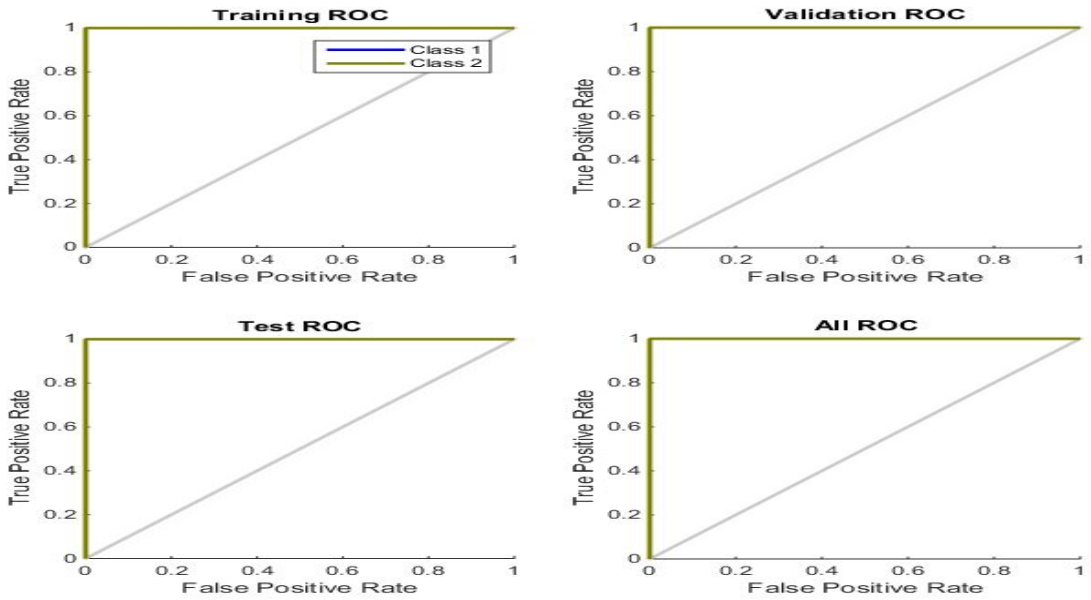


Anomaly Detection

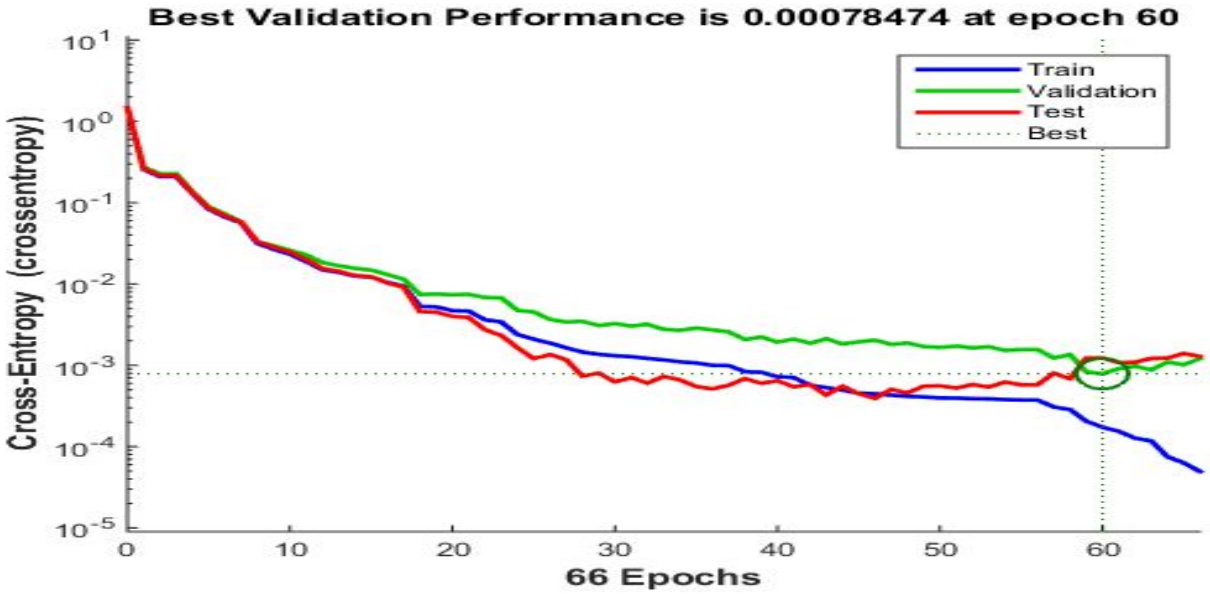


The given above is the confusion matrix of the anomaly detection. The class values are converted to nominal values as the following: *Value 1 = Attack* *Value 2 = Normal*. The number of neurons used for this neural network model are 10. The confusion matrix for the training dataset implies that 100% of the records are trained correctly with 0% of misclassification. The validation confusion matrix resulted with 100% accurate classification rate. The test confusion matrix resulted in 100% classification accuracy, classifying all the data instances correctly. The overall accuracy of the model was 100%. Every instances of the anomaly dataset (20428) are correctly classified either as Attack type or Normal type. There wasn't any necessity to change the number of neurons as the accuracy rate of the model calculated was 100% .

The following graph is the ROC plot which shows the false positive vs true positive rate.



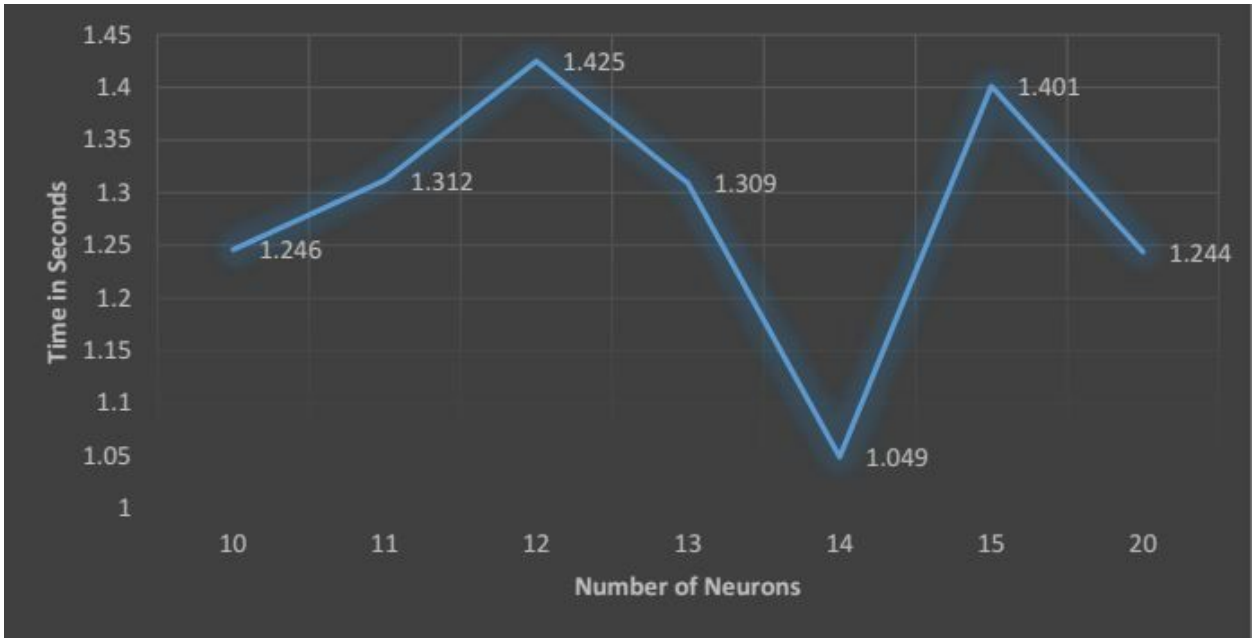
The graph below gives the best validation performance of the Smurf dataset which occurred at the epoch 60.



The 'memory' function was used to get the memory consumption for the default settings. The memory consumption of the NN by the MatLab is as follows:

Maximum possible array: 12601 MB (1.321e+10 bytes) *
Memory available for all arrays: 12601 MB (1.321e+10 bytes) *
Memory used by MATLAB: 1079 MB (1.132e+09 bytes)
Physical Memory (RAM): 8067 MB (8.459e+09 bytes)

The time consumed by the neural network model for the Anomaly detection is given below.



Development Process

The workflow for the project was divided as follows:

- **Research**: The research involved gathering knowledge about how Neural Networks work and understanding the application of the algorithms to classify the dataset. This was done by all the team members
- **Data Cleaning and Preparation**: As we were using MATLAB for our project, we were required to input two different files, one having only the class attributes and other file without the class attributes. So, splitting the dataset into two files using R tool and further cleaning was done by Bhakti Bhadrecha
- **Classifying the dataset**: Dhivya Govindarajan analyzed the algorithms, packages in the MatLab and applied the Neural Network model to the prepared dataset. The results of the analysis were shown in the earlier section.
- **Memory consumption and Time charts**: Jaydeep Untwal further worked on the results generated by the Neural Networks and came up with a function in Microsoft excel which enables us to draw the Memory Consumption and Timing graphs
- **Analyzing the results**: This was done by all the team members
- **Documentation:** This involved documenting all the above steps in a detailed manner. All the team members participated equally toward the completion of the project

References

- Project 1: Intrusion Detection System and Design, D. Govindarajan, B. Bhadrecha, J. Untwal, Rochester Institute of Technology
- https://en.wikipedia.org/wiki/Levenberg%E2%80%93Marquardt_algorithm
- https://en.wikipedia.org/wiki/Backpropagation
- https://en.wikipedia.org/wiki/Artificial_neural_network
- http://www.mathworks.com/help/nnet/ug/multilayer-neural-networks-and-backpropagation-training.html
- Introduction

  Image:http://www.unikaz.asia/sites/default/files/public/partners/photos/i-mozg/wallpaper-2870969.jpg
- Multi-layer perceptron: http://www.odec.ca/projects/2007/stag7m2/images/fig2.gif
- Back-propagation:

  http://engineeronadisk.com/V2/hugh_jack_masters/engineeronadisk-12.gif
- Curve for LMA: http://www.matrixlab-examples.com/image-files/curve-fit-005.gif