Project Part 3 Documentation

Foundations of Artificial Intelligence

Divesh Badod(db9932@rit.edu)

Prof. Leonid Reznik

1. Executive Summary

We are tackling this problem with a machine learning model. From the dataset generated by the ES, we are building a neural network which through regression will learn about the security evaluation used in the ES model and we will test it out to predict the security score of the devices as closer to the expert system as possible. Expert System is computationally expensive hence its deployment in different devices will questionable since not all devices will be able to handle the computation power required by the expert system, hence we are building a neural network model training it on the dataset created by the expert system to work as an expert system on devices which will not be able to handle the computations for the expert systems. The neural network will be easier to deploy and the computations required after the system is trained is low. Neural network models work as real-life biological neural networks with 3 layers consisting of neurons amongst which the hidden layer consists of the neuron of varied numbers. This model was introduced with the idea of human brain function in mind, with neurons working as simple processors of this network and the connection between them forms a network of neurons analogous to a human's neural network. The neurons compute the weighted sum of the input signals and compare the result with the threshold value. If the net input is lesser than the threshold then the output gets a value of -1 otherwise the neuron becomes activated with the given activation function and the output value becomes +1. This is the basis of how the neurons work and multiple models are ranging from using 1 single neuron called the perceptron to multi-layer perceptron and as the name suggests it consists of multiple neurons. There is a recurrent neural network, long-short term neural network, convolutional neural network and many more depending on what the requirements of the problem are.

For our case, we will be using a multi-layer perceptron neural network from the Keras library of python programming language. We are tackling our problem with regression technique. All the basic neural network uses the technique of backpropagation which adjusts the weight of the inputs to the neurons the get to the target value with the least percentage of error possible. This is also implemented in our project, the neurons in the hidden layer were one of the many changing factors of the system. There were multiple datasets which I created throughout the experiment with entries as big as 800,000, but the neural network was trained with a dataset of 60,000 entries with 100 epochs. The epochs are a tricky variable, an epoch means how many times the neural network cycles through the training dataset. But there is no guarantee that a larger number of epochs will lead to the greater accuracy it is considered an art in AI to determine how many numbers of epochs should a neural network train through. Its impact on our implementation is discussed below. The activation function is decided depending on what you want as the output, this is the function that triggers the neurons in the neural network and helps it in improving the weights of the inputs. There are several activation functions too like Step function, Sign function, Sigmoid function, and Linear function, for this project we have implemented Sigmoid function. The most important tool in this project is the Keras package followed by the Matlab package both of their functions and implementation is described in the following sections. The memory consumption, absolute errors, regression technique and implementation, loss of the model everything else required for performance evaluation of the neural network was calculated and presented in the implementation section.

2. Requirements

Requirements presented below will give you the basic idea of what this project will work since the neural network is involved there will be performance issues with outdated hardware and software, so the idealistic requirements are presented below.

- **Software Requirements**

The requirements for this project will be any device with python and its packages installed. Since the latest python is 3.8 there are some packages which need updating from the previous version, so I recommend 3.7 since I have used 3.7 for building this project. The latest OS is the fundamental requirement respective of which software you use be it macOS, Windows or Linux, see to it that the latest OS is installed. The following list shows which are the latest OS in the market right now: -

- Windows 10
- Linux 3.0.0
- MacOS 10.15

These are the most recognised OS around the globe right now hence I have specified the latest versions of only these three. There are multiple operating systems which use Linux at its core but they are their independent OS with Ubuntu being the most well-known, hence I would recommend to all the users with different OS to use the latest update because of the packages and the python itself being used for this project. Any outdated package or library will crash the program immediately.

- **Hardware Requirements**

Hardware requirements can vary since they don’t affect the implementation very much but since I have used Keras I would recommend that you have a GPU integrated into your hardware system. It's not mandatory but it will improve the performance of this project significantly. Now the following are some basic requirements regarding the hardware of your system.

- Processor: - Intel (i5, i7, i9), AMD (Z+ Series, Zen 2 Series)
- GPU: - The CPUs mentioned above are enough for the GUI to load but Nvidia and AMD has a varied range of graphics processors
- RAM: - Above 2gb
- Storage: - Above 1gb

Processor, RAM, and GPUs are the most reliant on factors to run this project the better and latest available hardware installed will be more beneficial. Since neural networks are to be trained and tested to work like an ES there will be a lot of lag while you execute, there are graphs to be generated and datasets to be trained and tested on. 60,000 entries in a dataset require a lot more hardware reliant system and the number of epochs and the dataset size was considered in such a way that all the systems are manageably able to run this project without the system crashing or the neural network just training for hours and hours without generating any output. Hence a good RAM, processor and GPU will significantly improve the project’s performance and get you the output instantly.

3. Implementation

The implementation will be divided into 2 parts one describing the theory of Machine learning concepts which I have used for building this project and the next section will describe the packages and the libraries which support these theories and how they were implemented.

- **Machine Learning Concepts implemented:**
    - **Neural Network: -** Artificial neural networks are the most researched upon topics in the field of AI mostly because it is the closest technology to imitate workings of a human brain. The basic concepts of a neural network are used in a large variety of applications like classification problems, data-processing problems, and various prediction problems (used in this project). A neural network consists of one or more neurons, which is the basic processing element. A neuron has one or more inputs and outputs which are individually weighted. In multilayer perceptron, one output of a neuron can be the input to other neurons. These inputs are computed by summating their weights and the output is produced which is normalized using an activation function.
    - **Activation function:** - The basic idea behind the activation function is to activate the neuron or not. Since the neurons calculate the weighted sum of the input and compare it to a threshold to check whether it should be fired or not, an activation function is used to determine the firing of the neuron. Various functions can be used like Step, Sigmoid, Sign, Linear, Tanh, ReLu function.
    - **Optimizers:** - Optimizers are algorithms or methods used to change the attributes of your neural network such as weights and learning rate to reduce the losses of the model. There are various optimizers you can use like Gradient Descent, Stochastic Gradient Descent, Mini-Batch Gradient Descent, Momentum, Adagrad, AdaDelta, Adam etc.
- **Project Implementation of these concepts:**
    - **Keras API: -** Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. The core data structures are layers and models which work as neural networks when called upon. The simplest model is the sequential model working as an MLP if given multiple layers otherwise working as an SLP or a single neuron. The Keras philosophy is to transform convoluted things to simple implementation while allowing the user to be fully in control when they need to. Keras follows the principle of progressive disclosure of complexity, it makes it easy to get started, yet it makes it possible to handle arbitrarily advanced use cases, only requiring incremental learning at each step.
        - Requirements for Keras:
            - Python 3.5–3.8
            - Ubuntu 16.04 or later
            - Windows 7 or later

- ➢ macOS 10.12.6 or later

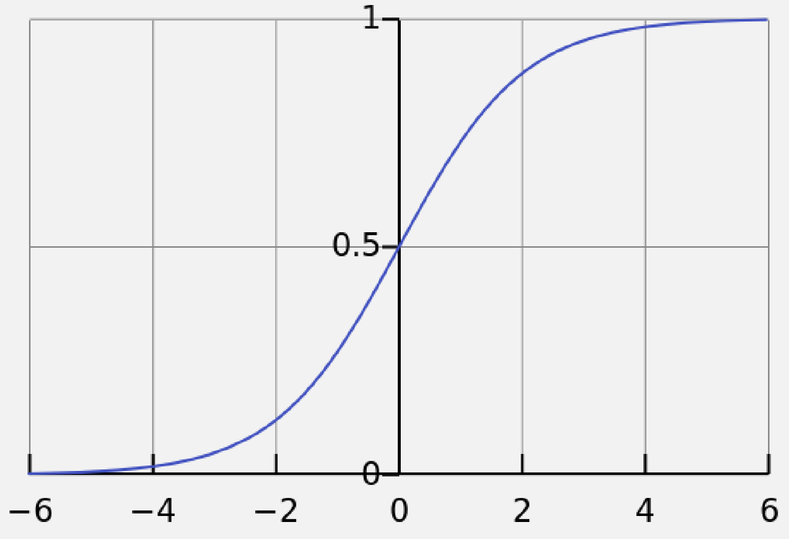
With simple lines like

```
model = keras.Sequential([layers.Dense(40, activation='sigmoid',
input_shape=[len(X_train.keys())]), layers.Dense(1, activation='sigmoid')])
model.compile(loss='mean_squared_error', optimizer='adam',
metrics=['mean_absolute_error', 'mean_squared_error'])



history = model.fit(X_train, Y_train, epochs=100, validation_split=0.3,
verbose=False)



loss, ae, se = model.evaluate(X_test, Y_test, verbose=0)
```

you can train and test a simple neural network and evaluate its performance.

- **Activation function(**activation='sigmoid'**): -** For the implementation of this project we have used the sigmoid activation function.

$$A = \frac{1}{1+e^{-x}}$$



A sigmoid function looks like a smooth step function in the shape of “S” or so-called “Sigmoid curve”. Looking at this curve makes you realize that this activation function works very well with the continuous values within the range of 0 to 1 which is exactly how I have prepared the dataset’s security rating outputs. This helps majorly in keeping the activation bounds in range without blowing up the activation of neurons in the implemented neural network.

- **Optimizer(**optimizer='adam'**): -** For the implementation of this project I have used the Adam(Adaptive moment estimation) optimizer since it is regarded as the best optimizer out of the other options mentioned above, as it trains neural networks with lesser time and much more efficiently compared to the others though it is only a little computationally expensive that doesn’t affect the performance too much. Adam works with momentums of first and second order. The perception behind Adam is that we don’t want to train so fast just because we can take over the minimum, we want to decrease the velocity a little bit for a careful search. In addition to storing an exponentially decaying average of past squared gradients**,** Adam also keeps an exponentially decaying average of past gradients.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}.$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}.$$

First and second-order of momentum.

- **Matplotlib library: -** This is a plotting library for Python programming language. It provides an object-oriented API for embedding plots into applications using GUI toolkits like Tkinter, wxPython, Qt, or GTK+. It can help in plotting graphs like Line, Histogram, Scatter, 3D, and image plot out of which I have implemented the Scatter and the Line plot. The results for those are in the results section.

- **Datasets: -** The ES implementation with all the metrics combined and values generated out of those metrics lead up to the creation of dataset with entries as large as 839,000. This will surely take a toll on the performance of this project and even though the neural network will train and evaluate itself with much more vigour. But a dataset with this much entries is not necessary even though I can work with it just by changing a single line in the implemented code. The entire experimentation process with the dataset is described in the following section.

- **Memory Profiler: -** This is a module in python which helps in checking the memory consumption of the python code being implemented and line by line analysis of the program created. Since the memory required for the program to run is extremely high, I have used this module to check the memory usage of the said implemented neural network. The memory usage and time took was entirely dependent on the number of neurons being used and the epochs for training the dataset. The result section further elaborates on the results produced by this module while implementing the application.

4. Experimentation

Experimentation of this project was dependent on 3 factors datasets, the number of neurons and the number epochs the neural network will go through for training purposes. The other factors were shown in the implementation were directly finalized with a careful review of the concepts implemented and the requirements of this project so there was no experimentation done regarding those variables.

- **Datasets: -** As mentioned above that the ES implementation gave so many combinations that the actual dataset leads up to 839,000 entries. To run the implemented neural network this much entries are not required but they don't harm the performance of the neural network in anyway but it affects the time taken by the application to complete the training, testing and evaluating the performance of the said neural network, which roughly takes up about 10-20 mins and this would've hurt the implementation of this application in other devices with lesser computational power or even GPUs. The GPUs accelerate the performance since the TensorFlow works on GPUs, but other devices might not have a separate GPU integrated into their hardware. Hence a reasonable size was considered of only 60,000 entries, keeping the application runnable for devices with lesser quality hardware or no GPUs. There was a third dataset I experimented with which contained only 300 entries, but this gave the neural network lesser entries to train on thus affecting the evaluation (Mean absolute error ranged from 53% to 55%). Hence after a few experimentations, I converged to the conclusion of using the dataset with 60,000 entries, so that the application is runnable on different devices and the neural network has a great number of values to train itself thus giving us a better performance.

- **Epochs:** - In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset. Usually, training a neural network takes more than a few epochs. In other words, if we feed a neural network the training data for more than one epoch in different patterns, we hope for a better generalization when given a new test data. Increasing the number of epochs doesn't necessarily mean that the network will converge to better results hence to decide on the number of epochs the network should run is an extremely necessary step. The time taken by the neural network to train gets considerably higher with the greater number of epochs, and with no guarantee of better results, it is beneficial to start with a smaller number and end at the point at which you might consider the results to be exceptional. Now I experimented with 50, 100, 500 and 1000 epochs. With the increasing number, the time taken for training was increasing exponentially and the results were more or less the same hence I considered keeping the number to 100 so that the neural network gets a good amount of time to train in a limited amount of time.

- **Number of Neurons:** - This is again a more heuristic approach to find the number of neurons like epochs but I started with the number of neurons half the size of the inputs and from there incremented it by doubling the neurons in the hidden layer for experimenting. I found out that the minimum mean absolute error when I used 10 neurons in the hidden layer. Even though it is an arbitrary selection since most of it is decided by intuition, I find that the best result was in the range of 10 to 15 neurons in the hidden layer.

5. Results

Results of this project are generated via graphs of line plot and scatter plot using the Matplotlib library. The results will be classified into the categories of Errors, Scatter, Loss and Memory consumption. These will be further classified with the number of neurons implemented. All these results were found with the epoch 100 and validation split 0.3

- **Memory Consumption: -**

Number of Neurons	Memory usage	Time Taken
5	302.4 MiB	470.46413373947 secs
10	303.3 MiB	488.16351246833 secs
20	303.2 MiB	473.81182646751 secs
40	303.2 MiB	480.98107743263 secs

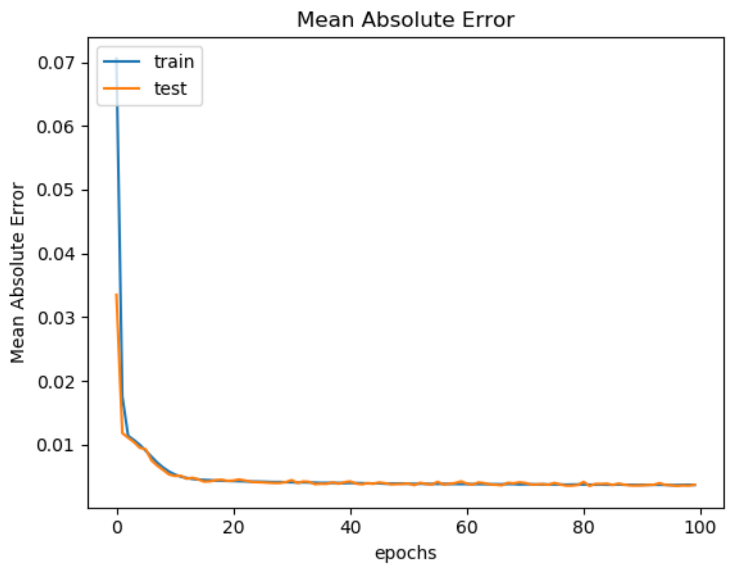
As represented here the memory consumption and time taken doesn't vary much even if the number of neurons was increased this only varies if the epoch is changed.

- **Errors: -** The is represented with two different calculations of errors, one being means absolute error and mean squared error after testing. The goal of these results was to reach towards 0 as close as possible and the following table and graph images represent the error scores. Again, this is done for 100 epochs and the table shows the mean calculated after 100 epochs and the graphs show each value.

Number of Neurons	**Mean absolute error**
5	0.003668108955025673
10	0.003048771293833852
20	0.0032363939099013805
40	0.0029367711395025253

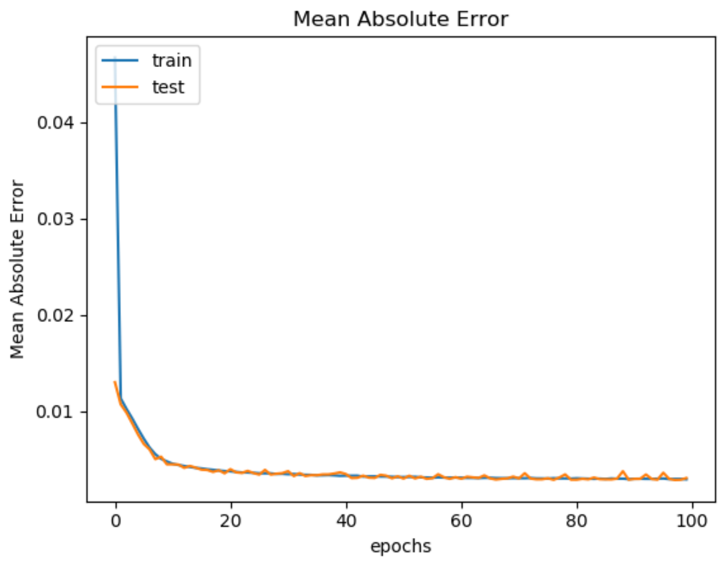
These are the mean values for all the epochs combined the graphs will illustrate the mean values for each epoch.

- **Mean Absolute error: -**
  With 5 neurons



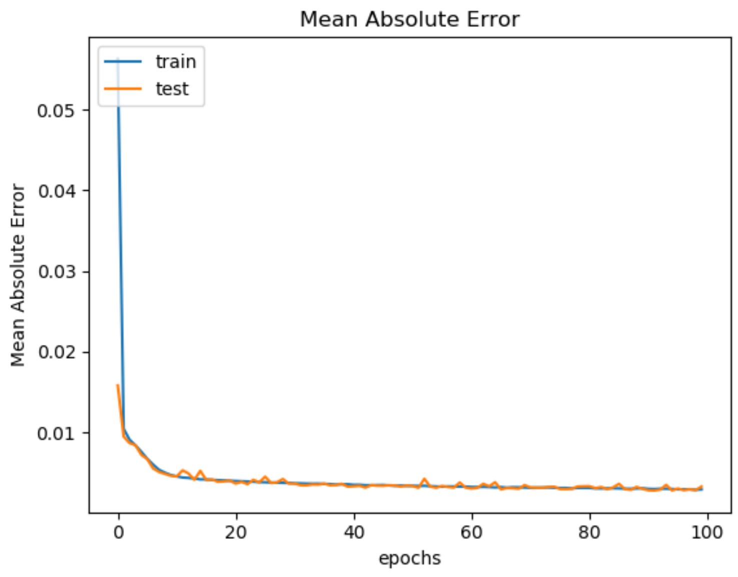
The value here is ranging from 70% to less than 10% for training and from 30% to again less than 10% for testing.

With 10 neurons



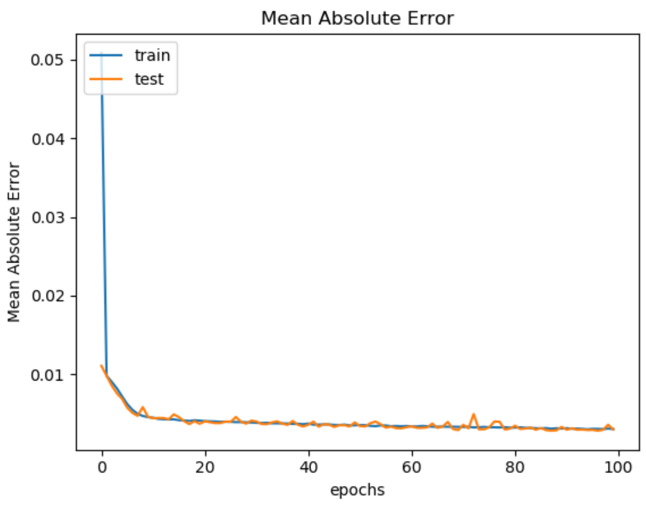
The value here is ranging from 50% to about 1% for training and from slightly above 10% to again about 1% for testing.

With 20 neurons



The value here is ranging from 60% to about1% for training and from slightly above 10% to again about 1% for testing.

With 40 neurons

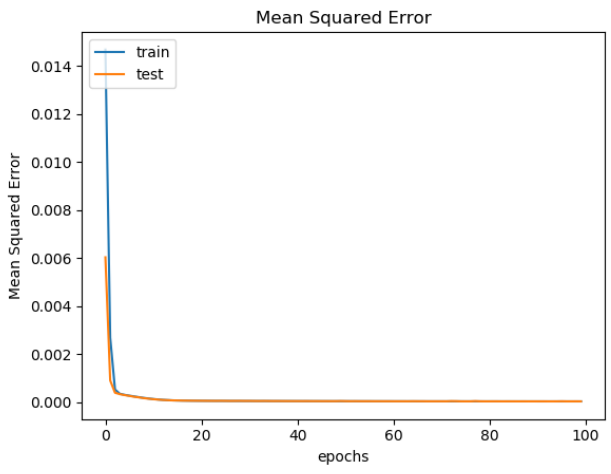


The value here is ranging from 50% to about 1% for training and from 10% to again about 1% for testing.

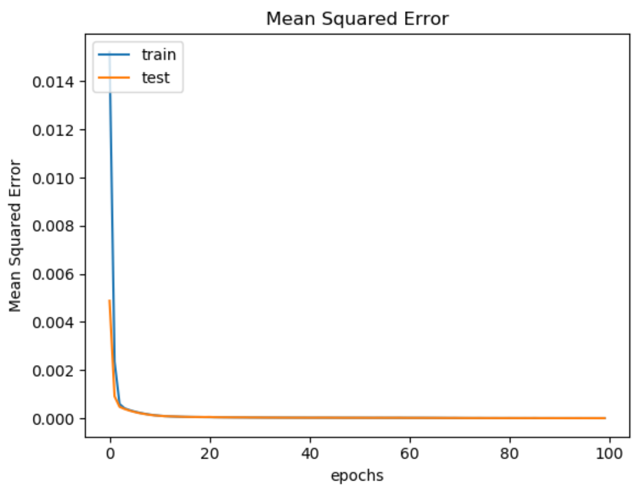
- **Model Loss: -** Loss function takes in the predicted score coming out from our model together with true values, compares it and gives us some quantitative value of how good or bad those predictions are for the training dataset. For our project, we are using the Mean squared error loss function which is calculated as the average of the squared differences between the predicted and actual values. Below is the outcome of the loss function or so-called Mean Squared Error values for the experimented neurons.

Neurons	**Mean squared error**
5	0.000031955820304574445
10	0.000020049801605637185
20	0.000021088098947075196
40	0.000021284429749357514

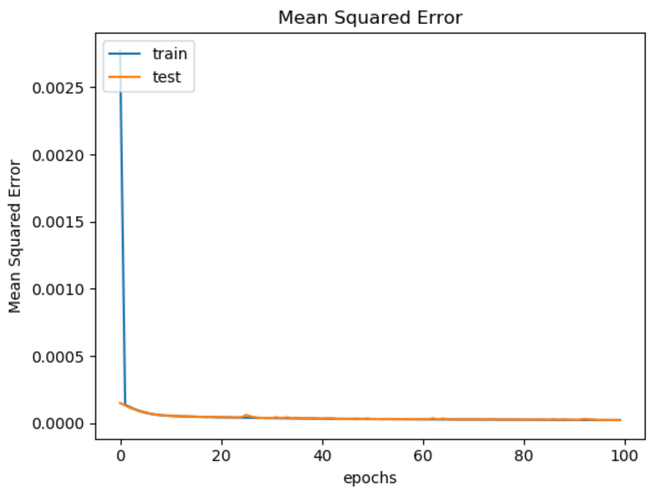
With 5 neurons



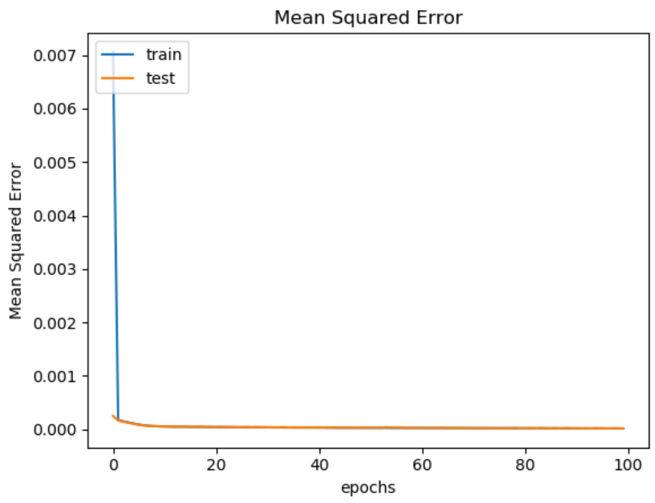
With 10 neurons



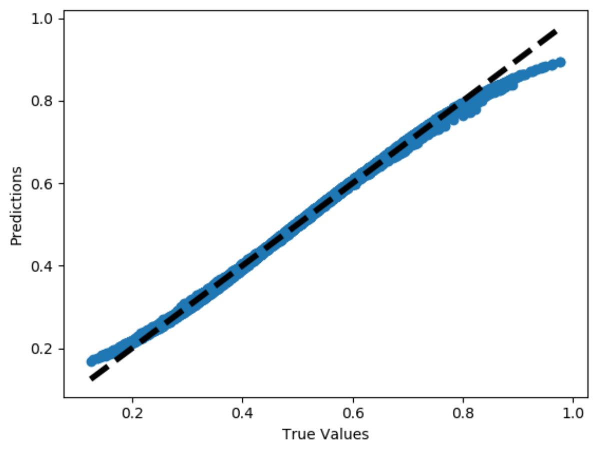
With 20 neurons



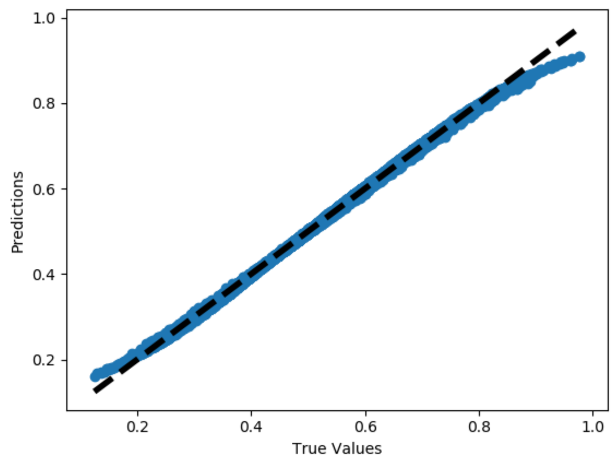
With 40 neurons



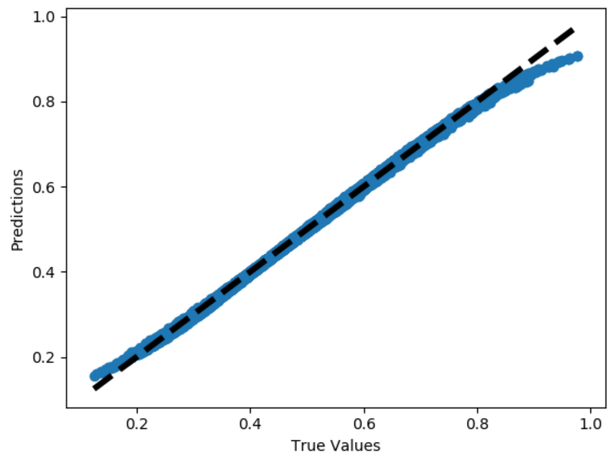
- **Scatter Plots: -** A scatter plot is defined as a graph of plotted points that show the relationship between the two sets of data. In my project, the two sets are the predicted and the actual values of the test data. Below are the scatter plots concerning the number of neurons used. Again, this is done for 100 epochs.
  With 5 neurons



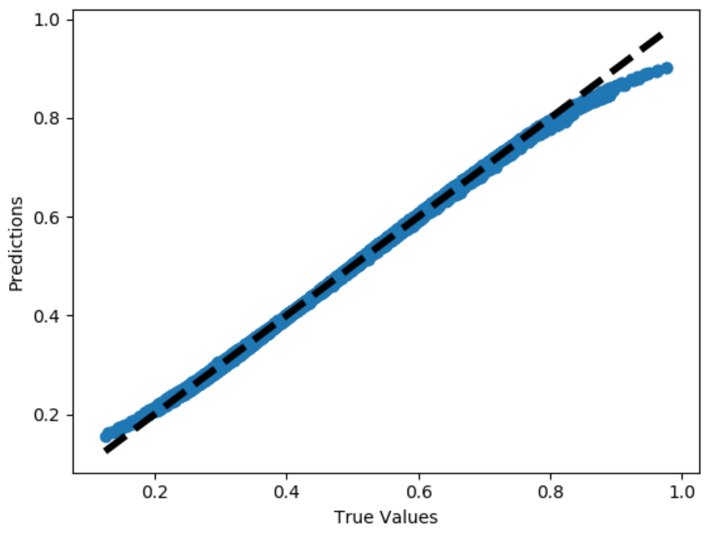
With 10 neurons



With 20 neurons



With 40 neurons



The scatter plots show that the actual values and the predicted values of the test data were extremely close thus confiding that the neural network built was very well trained and evaluated.

The results shown here will vary with different datasets and even by changing the number of epoch or neurons. The goal of this project was to implement a neural network and train it to predict values as close to the ES system as possible and with the results presented above it is certain that the neural network is showing you the system security rating with almost 0% error rate when compared to the ES system built in the previous part of the project.