Anushree Sitaram Das
ad1707@rit.edu
Foundation of Artificial Intelligence
Project Report

Use of Machine Learning Technique to Implement ES

Summary

In the previous project, an android application was build implementing an expert system for security evaluation. While an ES design has many advantages, its implementation requires a high computational power and the available expert system tools have not yet produced a stable and reliable version for mobile development. To address these concerns machine learning techniques can be used to improve the performance of the android application. Machine learning is the construction and study of algorithms that can learn from data. The differentiating feature of machine learning techniques is that it operates by building a model based on inputs and using that to make predictions or decisions, rather than following only explicitly programmed instructions. The same process of building a model based on inputs and using that to make predictions or decision can be applied in evaluating the security of an android device. To decide the final features of the model for example how many layers should it have, how many neurons should each hidden layer have, a thorough study have to performed examining all the possibilities and coming up with the best result. As a result, this report describes the empirical study of number of neurons affect the performance of the neural network.

Requirements

To overcome the shortcomings of expert system, machine learning techniques can be used to improve the security evaluation of android smartphone described in the previous project. Following tasks are needed to be performed according to the problem statement:

- Investigate a novel approach to an ES implementation on mobile devices by utilizing machine learning (ML) techniques to approximate the hyper-surface that is produced by the ES.

- Find a good approximation of the ES that could be easily implemented on a resource constrained computation platform.
- Research the choice of the number of neurons in a hidden layer and conduct an empirical study of different choices by computer simulation.
- Write and execute a code that will allow to conduct the empirical study and process the result.

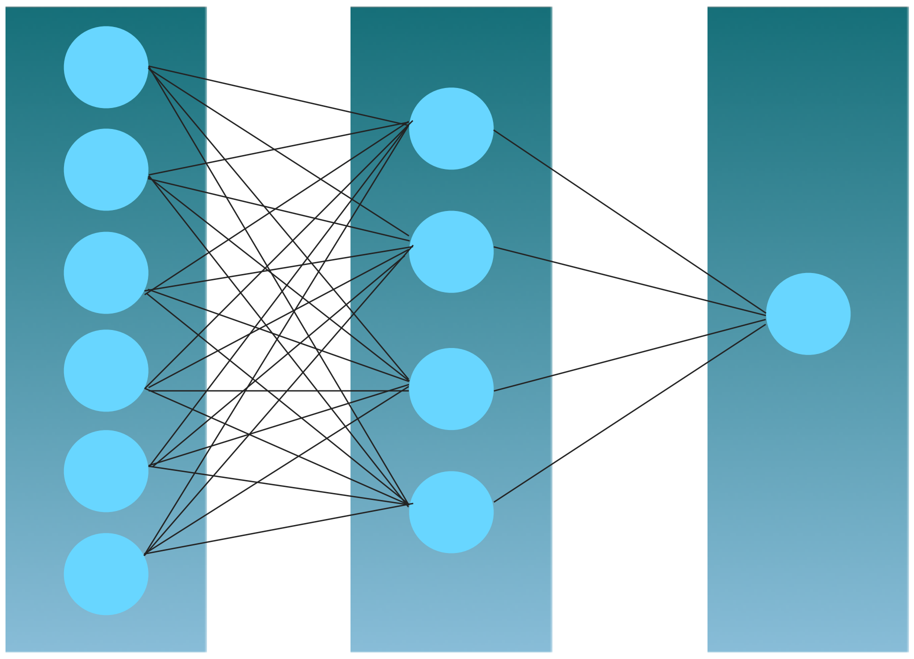
The machine learning techniques used can be artificial neural networks (ANN) and specifically their multilayer perceptron (MLP) model or any other ML methods. The structure of the model should have three-layers with an input layer, a hidden layer and an output layer. The research should be based on how number of neurons influence the neural network's performance and resource consumption.

Implementation

Since Machine learning uses algorithms that can learn from data, the first order of business is to build dataset. To do so I generated all possible combinations (permutations) of all inputs and apply them to the ES model. But inputs with security score around 9 and 10 were very few compare to others. So the dataset was sorted accordingly to have equal number of input cases for each range of score. After generating the dataset, it was subdivided into the training set (around 70%) and testing set (around 30%).

The next step is build the model to which the dataset can be feed. For this project, I have used tensorflow[1] library in python to simulate neural networks and study the relation between neurons and the neural network's performance and resource consumption. The neural network is composed of 3 layers:

Input Layer + Hidden Layer + Output Layer



The model with three layers are build using Keras API[2] provided by tensor flow. The input layer has 6 neurons, since there are 6 metrics as security metrics: API level, Security Patch Year, Security Patch Month, Device Lock set, Root Access granted and Number of applications with dangerous permissions. The hidden layer will have variable number of neurons to perform the experiment. The output layer has only one neuron which gives the security rate of an android smartphone. The activation function used in hidden layer is RELU(rectified linear units) function[3]. The rectified linear activation function[3] is a piecewise linear function that will output the input directly if is positive, otherwise, it will output zero. The optimizer used to compile this model is Adam[4]. Adam[4] is the most used and fastest learning optimizer. Loss calculated using MSE(mean squared error) and MAE(mean absolute error). I have used MAE as a major factor to evaluate performance because it is useful in predicting the difference between the actual output and the predicted output.

To demonstrate how performance (approximation errors) and resource consumption depend on the number of neurons, graphs are plotted using matplotlib[5], a python library.

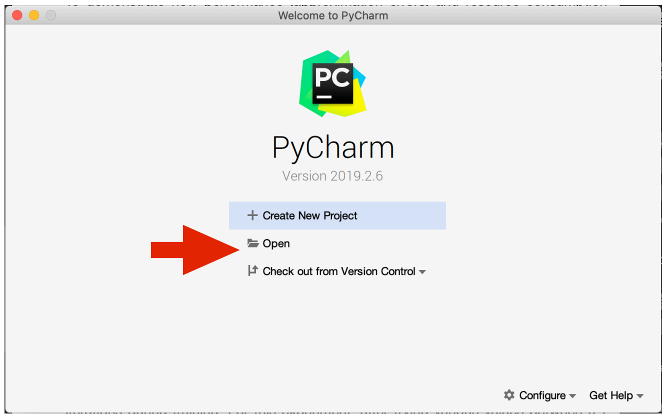
User's Guide

To run the code which carries out the experiment, you require an environment which runs python 3.7. Also the environment should have the following libraries installed in it:

- matplotlib[5] - used for plotting graphs
- numpy[6] - used to create arrays to store values for plotting
- pandas[7] - used to read data from csv file and store in a dataframe
- tensorflow[1] - used to create neural network
- memory_profiler[8] - used to see memory consumption
- time[9] - used to see time consumed

To run the python code provided follow the steps mentioned below:

1. Download Pycharm[10] from https://www.jetbrains.com/pycharm/download
2. Install Pycharm by following the instructions given on https://www.jetbrains.com/help/pycharm/installation-guide.html#silent
3. Launch Pycharm IDE
4. Click Open as shown in the figure and go to the folder where the neuralnetwork.py code is and select it.



5. Go to File -> Other Settings -> Preferences for new projects.
6. Install all the libraries mentioned above.
7. Right-click anywhere on code and click Run' neuralnetwork'

Experiments

First, we need to decide values for several factors like learning rate, number of iterations during training. For this experiment, after trying various values between 0.1 and 0.001, the most suitable learning rate was 0.06. After trying 2, 5,10 and 20 number of iterations, the following analysis was made: smaller number of iterations were giving larger MAE while larger number of iterations is giving smaller MAE but taking up too much time. So, the number of iterations for this project was decided to be 10.

To find how the number of neurons affect the error, memory usage and time, the following algorithm was followed:

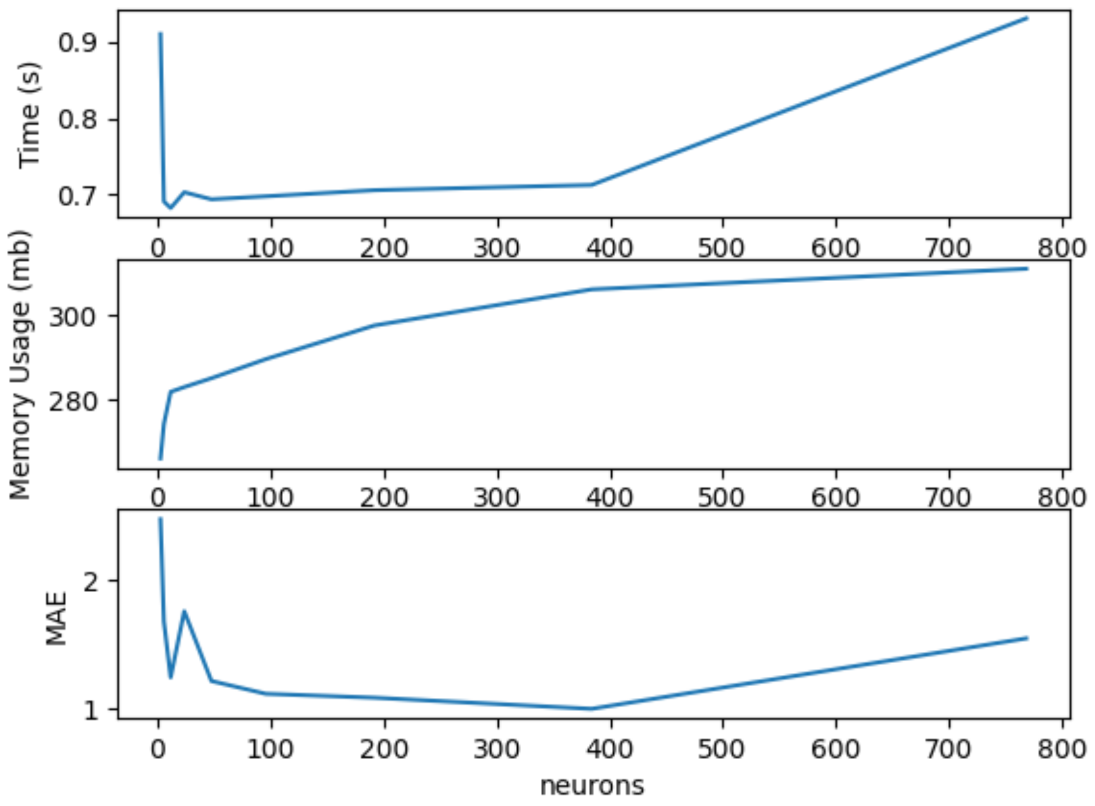
1. Choose the initial number of neurons in the hidden layer equal to about a half of a number of inputs, i.e, 3. Weights should be randomly generated by the neural network model.
2. Train the neural network with the goal to achieve a reasonable error for the training set. After the training is completed record the final error achieved and the time and memory consumed.
3. Take the neural network produced by the training and apply the testing set to it. Record final error achieved and the time and memory consumed.
4. Make the number of neurons in the hidden layer about twice as much as in the previous step, repeat steps 2-3.
5. Repeat step 4 a few times till we get the error values pretty low (around 0.1). Build plots demonstrating how the performance and resource consumption depend on the number of neurons.

The following section explains the output of the experiment and the graphs supporting the results.

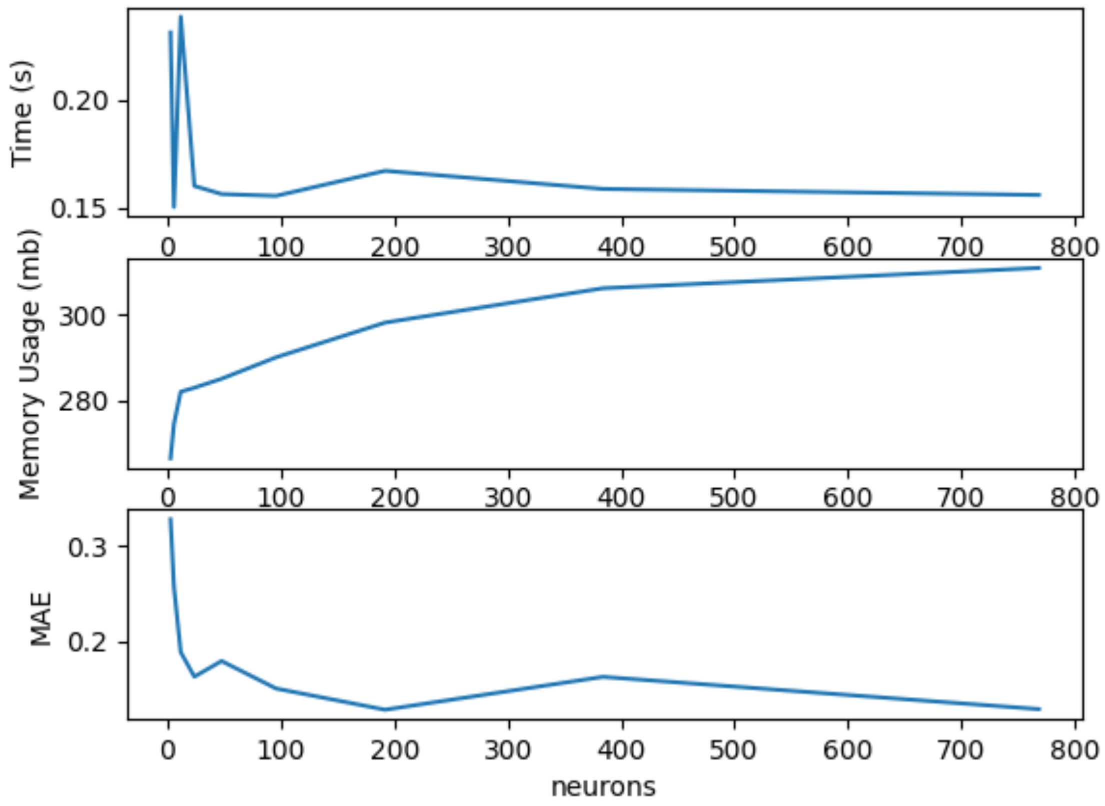
Results

After performing the experiment described in the previous section, the following graphs were obtained. Graph 1 shows how the number of neurons affect the MAE, the time taken to train and memory consumption during training. Graph 2 shows how the number of neurons affect the MAE, the time taken to train and memory consumption during testing.

Graph 1



Graph 2



As demonstrated in Graph 1, the MAE starts to increase after number of neurons increases after 400 during training. But on the other hand, MAE slightly rises when number of neurons is around 400 while testing. Apart from this little contradiction, when the number of neurons crosses 400 the memory usage during training and testing ,both reach a near constant value. In terms of time taken, it increases gradually during training after crossing 400 neurons but for testing the time is almost constant and very low after crossing 400 neurons.

As a result, number of neurons suitable for the hidden layer of the neural network model of this project is around 400 neurons based on the observations made from the result of the experiment performed.

References

1. Tensorflow https://www.tensorflow.org/
2. Keras API https://www.tensorflow.org/api_docs/python/tf/keras
3. RELU(rectified linear units) function https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU
4. Adam Optimizer https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam
5. matplotlib https://matplotlib.org/
6. numpy https://numpy.org/
7. pandas https://pandas.pydata.org/
8. memory-profiler https://pypi.org/project/memory-profiler/
9. time https://docs.python.org/3/library/time.html
10. Pycharm IDE https://www.jetbrains.com/pycharm/