Jack Moorin De Montfort University Intelligent Systems and Robotics Masters

Artificial Neural Networks and Deep Learning IMAT52355

‘A neural network model for detecting intrusions or attacks on a computer network’ Executive Summary

**Introduction:**

Within the executive summary document I will be summarising the neural network that was developed as part of my coursework for the ’Artificial Neural Networks and Deep Learning IMAT52355’ module, which I am currently undertaking at De Montfort University. The task of the assigned coursework was to design, program and trial a neural network program which would be trained on the KDD Cup 1999 dataset in an attempt to build a model that would be able to classify connections as being either a good connection or a potentially (malicious) bad.

**Dataset Initial Analysis:**

Before the neural network is developed, the provided dataset is analysed to assist in understanding the provided data and therein how to structure the network to provide optimal results. Initially the range of values within each column of the dataset is analysed and the total spread of values output, which enables the easy discovery of which columns hold continuous and discrete data and further the classification of columns which hold only a single value and can therefore be removed from the dataset as an introduction of unnecessary complexity. Dataset columns are further analysed with the implementation of pie chart drawing functionality which can be used to visualise the spread of values with a given column as a pie chart.

**Dataset Reduction:**

The dataset then goes through a pruning process in which columns within the dataset that are believed to be irrelevant to the classification of an attack are removed to reduce the overall complexity of the dataset and thereby reduce the amount of time needed to train the model. Irrelevant columns are defined as those which are empty and contain only NA values, those which have only a single unique value and therefore won’t impact the resultant prediction process and columns which have values highly corelated with that of another column as their data can therefore be represented by the column they’re correlated with. Before and after removal of highly correlated columns, the correlation within the dataset is visualized using a heat map to help present this correlation data.

**Define the Neural Network:**

Following analysis and reduction of the dataset, the neural network model is then defined. First, in order for the dataset to be able to be input into the graph as training data, it is encoded using one of two defined methods which can be alternated between by commenting the unused method and uncommenting the method to be used by the system. The first method uses mapping to encode the input data and defines a binary classification of the input data as either a normal or attack packet, while the second method uses a label encoder and the NumPy utilities function to\_categorical to encode the input data as ‘one hot’ vectors and defines a multiclass classification of the input data as one of a series of attack types including normal. The input and output columns of the dataset are then split into separate lists, the inputs are then scaled using the sklearn library’s MinMaxScaler to normalise them, the output column encoded as dummy indicators and the inputs simplified even further by being projected into a lower dimensionality space using linear dimensionality reduction. Now transformed into a format understandable by the network, the inputs and output are then partitioned into the inputs and output to be used in the training of the model and inputs and outputs to be used in its evaluation. The neural network is then defined as a TensorFlow Keras sequential model with five layers with units 32, 32, 32, 5 and number of output possibilities respectively before it is then compiled and trained.

**Evaluating the Neural Network:**

Now created, the performance of the neural network is then evaluated and visualised using a series of graphs output metrics. The first and second series of output graphs show the accuracy and loss of the network during training and validation respectively, using the data stored in the history object returned when the network was trained. The network is then tested using both the inbuilt testing functionality within the TensorFlow Keras library’s evaluate function and directly by manually testing the network using the test data. The network’s accuracy is then plotted in a final ‘Donut’ style pie chart graph.

**Testing Process:**

Multiple tests were carried out on the networks configuration parameters to find the combination which resulted in the most accurate final model. This process and the results compiled are documented in appendix A.

**Final Conclusions:**

Completing this coursework has served as a valuable learning experience as it has enabled the further development of both my programming skills within the Python programming language and my familiarity with and understanding of neural networks. Most notably improved was my understanding of sufficient ways to reduce the overall dataset used to train a neural network, namely the removal of highly correlated data attributes. In the event further time was available to complete the project a credible milestone would have been to enable the network to classify the given input by the more numerous intrusion type rather than attack category the intrusion lies within.

**Appendices:**

**A1)** Testing carried out on the correlation threshold used in data pruning:

The first of the carried-out tests concerns the defined correlation threshold value used when pruning the dataset used to train the Neural Network. Columns within the dataset that have a greater calculated Pearson correlation coefficient than the specified threshold to another given column are removed from the dataset. These columns are removed as their high correlation indicates that they present similar data and can be pruned from the dataset, making the process of training the network faster whilst not impacting on the overall accuracy.

In total a series of eight different values for the threshold were tested to determine the optimum threshold value that results in the best performance of the Neural Network. The below table shows the compiled testing data:

|  |  |  |  |
| --- | --- | --- | --- |
| Threshold: | Columns Pruned: | Final Accuracy: | Final Loss: |
| 0.5 | 18 | 98.73% | 0.016 |
| 0.6 | 16 | 98.83% | 0.015 |
| 0.7 | 14 | 99.16% | 0.012 |
| 0.75 | 13 | 99.21% | 0.012 |
| 0.8 | 13 | 98.86% | 0.014 |
| 0.85 | 12 | 99.13% | 0.014 |
| 0.9 | 10 | 98.86% | 0.014 |
| 0.95 | 8 | 99.00% | 0.018 |

Following the completed tests, the correlation threshold value which produced the best performance in the resultant neural network was determined to be 0.75, as it resulted in the highest accuracy and lowest loss of all the tested values.

**A2)** Testing carried out on the number of components used in principle component analysis:

The second set of tests carried out was on the number of components used by the sklearn library’s principal component analysis which reduces the neural networks inputs into dimensionality equal to the number of components set. The more components used the greater the number of inputs the network is trained on.

A total of eight different number of components were trialled to discover the optimum value and the results of these tests compiled into the below table.

|  |  |  |
| --- | --- | --- |
| Components: | Final Accuracy: | Final Loss: |
| 1 | 98.43% | 0.028 |
| 2 | 98.77% | 0.019 |
| 3 | 99.17% | 0.012 |
| 4 | 98.81% | 0.019 |
| 5 | 99.14% | 0.013 |
| 10 | 99.57% | 0.007 |
| 20 | 99.14% | 0.009 |
| 25 | 99.12% | 0.017 |

Following the completed tests on the number of components used, the value which resulted in the best performance was found to be 10, which had the highest accuracy and lowest loss in the resultant neural network.

**A3)** Testing carried out on the optimiser used when the network model is defined

The third set of tests carried out was on the optimiser used when the neural networks model is compiled. The optimiser defines how the network will calculate loss during training which will in turn be used to update the networks weights.

Eight different optimisers were tested to discover which results in the best performance for the network. The below table holds the data compiled during testing.

|  |  |  |
| --- | --- | --- |
| Optimiser: | Final Accuracy: | Final Loss: |
| ‘adam’ | 99.63% | 0.005 |
| ‘adamax’ | 99.14% | 0.016 |
| ‘ftrl’ | 79.22% | 0.296 |
| ‘nadam’ | 99.17% | 0.015 |
| ‘rmsprop’ | 99.11% | 0.016 |
| ‘sgd’ | 79.29% | 0.215 |
| ‘adadelta’ | 79.22% | 0.222 |
| ‘adagrad’ | 79.23% | 0.273 |

Following testing of the optimisers, the optimiser which resulted in the best overall performance of the neural network was found to be the ‘adam’ optimiser which resulted in the highest accuracy of the network and the lowest final loss value.