Network Intrusion Detector

Lucas Saechao, Ashley Thor, Noah Venethongkham

218794239, 219334909, 219660117

CSC 180 - Intelligent Systems

Project 2

May 12, 2021

Distinguishing Between Good and Bad Connections

Abstract

The following report describes the motivation, challenges, and methods used in order to train two neural networks that will predict the intentions of a network connection on some system. This model is trained using the KDD Cup 1999 Dataset, “Computer Network Intrusion Detection”.

Problem Statement

Software that protects computer networks from unauthorized intrusion or access provides a real, and universal value to individuals and enterprises around the world. This type of software is known as an Intrusion Detection System (IDS), and is intended to monitor a network for malicious activity. The advent and explosion of machine learning models, and software in the last decade provide opportunities to develop new techniques in order to determine good or bad actors in a system.

This project aims to build a Network IDS (NIDS) using both a convolutional neural network, and a fully connected neural network, and provide an accurate, predictive model that can distinguish bad connections, intrusions, or attacks, from genuine, well-intentioned network connections.

Methodology

In order to train this dataset, several Python library functions were utilized from Scikit-learn, TensorFlow, Keras, Numpy, and Pandas. To prepare the dataset for training, several preliminary steps were necessary. First, data that would not have been helpful to determining the outcome must be removed. This includes duplicate records, and records with missing values. We approached this problem as a binary classification problem, which means that we only cared about good or bad outcomes in general, without regard for what "type" of intrusion the bad outcome would have been. To accomplish this, we provide a function `make_binary(input_str)` which the outcome features are passed as an input, and it would then map '0' to "normal" connections, and '1' for any other connection. Next, we want to remove any records that are unhelpful for the purpose of training this data. The final step we took is encoding the data, or normalizing the data into numerical features.

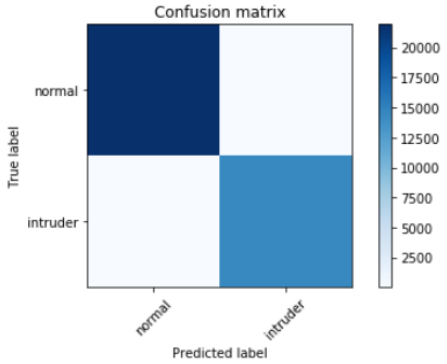
After preprocessing the dataset, we began to build two neural networks. The first is a dense neural network with three layers: An input layer with 200 neurons, a hidden layer with 150 neurons, and an output layer, with two neurons. Their activator functions are relu, sigmoid, and softmax. This model was compiled using the adam optimizer, and the sparse categorical crossentropy loss function. The second model is a convolutional neural network (CNN), and this required additional preprocessing - we needed to allow the model to view the data as an image. After this, the model's data had a shape of N records, 1, 121, and a single alpha channel. The CNN featured several layers: a relu input layer with 200 neurons, and a 1x5 kernel, a relu convolutional layer of 150 neurons and a 1x4 kernel, a 1x2 max pooling layer, a dropout layer, a fully connected

relu layer, another dropout layer, and an output layer with two neurons. This model was compiled using categorical crossentropy, and the adam optimizer.

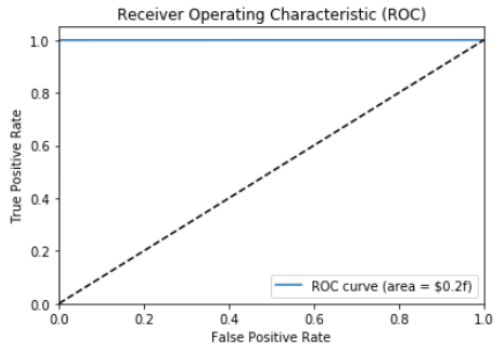
Experimental Results and Analysis

Noah and Ashley worked together in determining an effective convolutional neural network design where we modeled an accuracy of .99 along with other favorable metrics including F1, precision, recall, and support.

```
Accuracy: 0.9990109074923758
Averaged F1: 0.9990108257029516
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     21938
           1       1.00      1.00      1.00     14459

    accuracy                           1.00     36397
   macro avg       1.00      1.00      1.00     36397
weighted avg       1.00      1.00      1.00     36397

Log Loss: 0.03416228661121941
```



	precision	recall	f1-score	support
0	1.00	1.00	1.00	21938
1	1.00	1.00	1.00	14459
accuracy			1.00	36397
macro avg	1.00	1.00	1.00	36397
weighted avg	1.00	1.00	1.00	36397



Figures 1-3 Showcase the precision, recall f1-score, support, confusion matrix, and ROC graphs for the convolutional neural network.

This model design was tested with multiple dropout layers to prevent overfitting. Lucas even trained the CNN model using different test training splits: 25/75, 50/50, 75/25 to gauge how the amount of test data changed the results and it largely left the accuracy unchanged.

Ashley built the dense neural network with 3 layers, using the activators relu, sigmoid, and softmax to achieve favorable results in test accuracy being .99.

```
Test loss: 0.003936686476780898
Test accuracy: 0.9989284873008728
```

Noah and Ashley worked together to write a method on how the model performs with a variety of activators and optimizers, with a total combination of 18, we have determined that relu, sigmoid, adam & tanh, sigmoid, adam is best.

```
[('relu', 'relu', 'adam'), ('relu', 'relu', 'sgd'), ('relu', 'sigmoid', 'adam'), ('relu', 'sigmoid', 'sgd'), ('relu', 'tanh',
'adam'), ('relu', 'tanh', 'sgd'), ('sigmoid', 'relu', 'adam'), ('sigmoid', 'relu', 'sgd'), ('sigmoid', 'sigmoid', 'adam'), ('si
gmoid', 'sigmoid', 'sgd'), ('sigmoid', 'tanh', 'adam'), ('sigmoid', 'tanh', 'sgd'), ('tanh', 'relu', 'adam'), ('tanh', 'relu',
'sgd'), ('tanh', 'sigmoid', 'adam'), ('tanh', 'sigmoid', 'sgd'), ('tanh', 'tanh', 'adam'), ('tanh', 'tanh', 'sgd')]
18
```

For the Convolutional neural network we have found at relu, relu, relu, adam is the best combination

```
Test loss: 1.7984585051351585e-05
Test accuracy: 1.0
```

Task Division and Project Reflection

For this project, we are working as a team of three, and the tasks were divided accordingly. Lucas worked on preprocessing the data and set up the code to load the data, train the model, and prepare the CNN. Noah and Ashley experimented using different optimizers to see which set of settings and neuron counts would provide the

greatest and most generalized CNN model fitting. Ashley also worked on implementing the fully connected neural network (DNN). Noah and Ashley worked together to write a method on how the DNN model performs with a variety of activators and optimizers.

Challenges

By far, the largest challenge of this project seemed to be attempting to allow the CNN to view the input data as an image. In preprocessing the data, Lucas worked out how to accomplish this, and shaped the data into a 1x121 grayscale image. Another challenge came after performing our analysis - our model seemed to be unnaturally accurate, and we worried that we were overfitting the model. Lucas trained the model using different test training splits: 25/75, 50/50, 75/25, in order to gauge how well the model generalized the input data. Each test training split yielded highly desirable results to which our team concluded that the problem must be significantly easier to model. Noah and Ashley were having difficulties in writing a method to test various activators and optimizers; however, after much effort they were able to iterate over 3 lists (layers). In addition, we tried to implement regularization, however, encountered problems configuring it to our model. In the end, we decided that we did not have enough time to add it to our code.

Other Insights

We did not test a large variety of combinations of hidden layers. We only tested the DNN model with 2-3 hidden layers and went up to 200 neurons. Likewise, we only

tested the CNN model with a handful of hidden layers and went up to 200 neurons. If we had more time, we could have tried to test a larger variety of hidden layers and neurons in order to produce the best model without overfitting. In addition, we used 10% of the original dataset per the instructions for the project, however, with a larger dataset such as 20% of the original dataset, our models can be trained to predict whether an incident is normal or if it is an intruder. However, more time would be required with our current hardware specifications.