CSCI-735 Project Phase 2: Neural Network in Intrusion Detection System

Parijat Kawale
pk7145@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

Archit Joshi
aj6082@rit.edu
Rochester Institute of Technology
Rochester, New York, USA

1 EXECUTIVE SUMMARY

The present cybersecurity framework heavily depends on the use of Intrusion Detection Systems (IDS). These systems oversee networks or systems to detect any malevolent actions or breaches of policy. Although IDS are limited in their capacity to correct these issues, they play a crucial role in alerting users to potential threats. In phase 2 of the project, using our understanding from phase 1 we introduced Neural Networks into intrusion detection systems.

For the current phase of our project, we have used the KDD Dataset[5] which was retrieved from Kaggle[1]. The KDD dataset contains samples for the following attacks - snmpgetattack, named, xlock, smurf, ipsweep, multihop, xsnoop, sendmail, guess_passwd, saint, buffer_overflow, portsweep, pod, apache2, phf, udpstorm, warezmaster, perl, satan, xterm, mscan, processtable, ps, nmap, rootkit, neptune, loadmodule, back, httptunnel, worm, mailbomb, ftp_write, teardrop, land, sqlattack, snmpguess

Compared to phase 1, we designed and developed a neural network architecture. There are two different architectures taken into consideration, that is, one for misuse-based IDS and another for anomaly-based IDS. The main point of concern is designing the neural network architecture that would be most appropriate for misuse-based classification or anomaly-based classification. We have discussed the architecture of each neural network in upcoming sections. Even though the number of layers may stay the same, the way we calculate the loss, softmax and finalize the architecture based on accuracy as the output metric.

Overall, the final approach decided for the anomaly-based IDS is a 3-layered neural network with an Adam[6] optimizer, and categorical cross-entropy loss function. The misuse-based IDS is a 3-layered neural network with an Adam optimizer and a sparse categorical cross-entropy loss function. The accuracy achieved by anomaly-based IDS is 96% and by misuse-based IDS is 94%.

2 SPECIFICATION

During phase 1, the expectation was to use an existing IDS tool such as SNORT[3] or Suricata[4] to develop two different IDS i.e. anomaly-based IDS and misuse-based IDS. However, we conclude phase 1 by designing our own anomaly-based IDS and misuse-based IDS using Python.

In the current phase 2, we started with analyzing and understanding how neural networks work starting from how they are built to how we can optimize them using optimizers and loss functions. For the anomaly-based IDS and misuse-based IDS, we started analyzing which type of optimizer would be most appropriate in order to get the highest accuracy in terms of attack classification. Starting from working on stochastic gradient descent, the Adam optimizer gave us the best results for both IDS architectures. However, when it came to determining the loss function, the anomaly-based classification gave better results when the loss calculation function was based on categorical cross-entropy, while the misuse-based classification gave better results when the loss calculation function was based on sparse categorical cross-entropy. The concepts are discussed in future detail in sections 3.2 and 3.3

We use sci-kit-learn's train_test_split package to divide the KDD data into test and training sets with a ratio of 60:40. For misuse IDS, the test data was further divided into validation and test data while anomaly IDS had only test and training data. After that, the final model was then tested on the testing set for each IDS to classify different types of attacks and detect anomalies.

3 METHODS AND TECHNIQUES

3.1 Data preparation

In order to use the KDD dataset[5] for the custom IDS tool, we had to preprocess the data to clean it so that it could be used to develop the models. Initially, we filter the occurrence of attacks that have less than 2 records in total in the dataset as these would not be help to establish a base case. Later, we normalized the data and scaled it in the range [0,1]. This was achieved using the MinMaxScaler function in the sklearn.preprocessing[2] module. Using the scaler ensured that all the different outputs had an equal weightage as well as improved the convergence of the model based on the data.

After normalizing the data, we encode the training data using one-hot encoding technique. This encoding converts the output data, which is in a non-numerical format to a numerical format, that is, assigning a number to each type of categorical attribute. This is used to ensure that the machine learning model can understand and treat different classes of output based on the different output numbers assigned. The total data length resulting after the preprocessing is a packet record of 311029 packets in total, each containing 42 different attributes.

3.2 Misuse based IDS

The Misuse IDS uses a Sequential model from the Tensorflow Keras[7], which is a feedforward neural network and the data flows in one direction from input to output.

The input layer is determined by the shape of the training data, which corresponds to the number of features in the dataset after preprocessing. The model has two hidden layers. The first layer of the hidden layer consists of 64 neurons. It is aimed at providing enough capacity to learn complex patterns without being too large to cause overfitting or excessive computation. The second layer has

Conference'17, July 2017, Washington, DC, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

32 neurons that funnel the network. The output layer has a number of neurons equal to the number of unique attack types and uses a softmax function that converts logits to probabilities that sum up to one.

The learning rate is based on the Adam optimizer for the model. The reason for this is its effectiveness in various conditions as it has adaptive learning rate properties. The loss function used by the misuse-based IDS is sparse categorical cross-entropy as it is suitable for multi-class classification with integer labels. The model accuracy of the model is 96.65%. The false positive and false negative ratios of the model are 0.00095 and 0.03346 respectively.

3.3 Anamoly based IDS

The anomaly-based IDS utilizes a Sequential model from the Tensorflow Keras[7]. This model is a straightforward feedforward neural network, where data flows in one direction from input to output.

The architecture starts from the Input layer which is based on the shape of the training data, which corresponds to the number of features. Next, we have two hidden layers. The first layer consists of 64 neurons. This is done in order to provide enough capacity to learn complex patterns without being too large to cause overfitting. The second hidden layer consists of 32 neurons, which funnel the network towards the output. The hidden layer uses a Rectified linear unit (ReLU) to understand complex patterns and their efficiency in non-linear transformations.

The number of neurons in the output layers is based on the number of classes in the target variable, which uses the softmax activation for multi-class classification. The neural net model is trained with 10 epochs and a batch size of 32. The model compilation uses categorical cross-entropy as it deals with multi-class classification with one-hot encoded labels and the Adam optimizer that has good robustness and ability to adapt the learning rate during training. This model gives us a complete accuracy of 96.34% with a loss of 0.0745. The false positive and false negative ratios for anomaly-based IDS are 0.00104 and 0.03754 respectively.

4 IMPLEMENTATION

The Python code is dependent on the TensorFlow [7] and scikit-learn[2] library for its machine-learning models.

4.1 Structure

The overall structure is a directory named "pr2_6". Inside this directory, we have a python3 file that contains the custom models defined in their respective functions and a data folder that contains the actual data from the data-set.

4.2 Software and Hardware Requirements

- Cores: 4
- RAM: minimum 4GB
- Python3 installed
- Machine learning libraries: scikit-learn, numpy, pandas

4.3 Limitations

The machine should have at least four cores for the software to run properly. Please note that running the machine learning model takes a considerable amount of time.

4.4 User guide

Please note this user guide assumes that you have satisfied the software and hardware requirements mentioned in section 4.2

(1) Download the "CSCI-735-main.zip"
(2) Extract the zip file.
(3) Inside the directory, we have the data folder and and python file.
(4) Run the python file inside the directory location using: *python3 annClassifier.py*
(5) The program will generate heatmap for the confusion matrix after each classifier finishes running. Thus please close the heatmap so that the program resumes execution.

5 TESTS

In order to conduct testing on the model developed, we tried and tested different strategies to choose the best from all of them. Firstly, we tried different split strategies such as dividing the dataset into 60% training data and 40% testing data or 70% training data and 30% testing data, and so on. These splits were done to choose the best split of the data to get the maximum accuracy and have the maximum amount of data to train the model. The splits also allow for determining how well the model would be towards generalization. A higher proportion for the test set accompanied by a higher accuracy suggests that the model can handle unseen data efficiently.

For misuse-based IDS, the split was 60% training data and 40% testing data and for anomaly-based IDS, the split was 60% training data. The testing data is further divided into 50% testing data and 50% validation data.

Along with splitting the dataset into training and testing, we tuned the hyperparameters for each of the models - activation function, loss functions and the optimizer - to achieve the highest accuracy possible based on the dataset.

A combination of splitting the dataset into training and testing data and tuning the hyperparameters to achieve the best results resulted in the misuse-based IDS having an accuracy of attack detection to be 96% and anomaly-based IDS having an accuracy of 96% .

6 RESULTS

6.1 Results from the ANN models with all attacks

```
    accuracy                           0.96    186617
   macro avg       0.50      0.48      0.48    186617
weighted avg       0.94      0.96      0.95    186617

False Positive Ratio: 0.0010199142271783029
False Negative Ratio: 0.0367169121784189
```

Figure 1: Console output from Misuse-based IDS with accuracy and FP+FN ratios

As seen in Figure 1, the accuracy of the ANN model for misuse-based IDS is 96.% accuracy. The false positives and false negatives

for most attacks remain on the lower end which suggests that the classifier is correctly identifying the intrusions from the data. The entire output for the entire program can be seen in the console once the program is run. For simplicity, we have also included the screenshots in the appendix section which contains the program results.

```
    accuracy                           0.97     62206
   macro avg       0.54      0.54      0.53     62206
weighted avg       0.95      0.97      0.96     62206

Overall False Positive Ratio: 0.0009209033579673068
Overall False Negative Ratio: 0.032231617528855734
```

Figure 2: Console output from Anomaly-based IDS with accuracy and FP+FN ratios

As seen in Figure 2, the accuracy for the anomaly based ANN model is 97%. As this model is classified on a binary scale - anomalous or normal, we do not have false positive and false negative ratios for each label. We were able to successfully check for such instances in our model up to 97 times out of 100. The entire output is available in the appendix section.

6.2 Removing 2 attack labels from the training set to evaluate generalization capability

To check the generalization capacity of the IDS, we remove the two most frequently occurring labels in the data set from the training data partition. The test data still has those labels so we will see a fluctuation in the accuracy of the classifier as the model is not accustomed to detect the attacks it was not trained on. This is where creating new rules and monitoring false positive/negatives comes into picture as the programmers/designers constantly need to improve the IDS for the ever evolving cyber crime landscape.

For the anomaly based IDS the accuracy drops to 80.5% as per the Figure 3. The false positive rate is quite high, at 24.2%. This means that the model is likely to flag a lot of normal traffic as anomalous. The recall of the model is 1.0, which means that it is able to detect all of the anomalous traffic in the testing data. However, this is not very useful if the accuracy is low, as it means that the model is also likely to flag a lot of normal traffic as anomalous.

For the misuse based IDS the accuracy is far worse at barely 25%. Please note that this happens because we removed the 2 most frequently occurring attacks as shown in Figure 4 and misuse based IDS will operate on classifying labels itself unlike anomaly based IDS where it classifies in a binary fashion depending on what behaviour is classified. On the other hand if we removed the labels for attacks "xsnoop" and "snmpgetattack" which only account for barely 2.49% of the whole data our accuracy improves as seen in Figure 5 to about 96%. Thus it depends on how the classifier is trained and the proportion of attacks that are encountered and collected for the training data. This signifies the importance of frequently training the model on updated data so it stays up-to-date with emerging threats and is able to classify more accurately.

```
======== Anomaly based IDS with modified data
Attacks removed from training data : ['smurf.', 'neptune.']
They are still present in the testing data.Accuracy might be affected.
Model TRAINED
Model TESTED
True Positives: 100175
False Negatives: 0
False Positives: 24237
True Negatives: 0
Accuracy: 0.8051876024820757
Precision: 0.8051876024820757
Recall: 1.0
F1 Score: 0.8920819103509998

Process finished with exit code 0
```

Figure 3: Console output and metrics from Anomaly-based IDS with 2 most frequent attacks removed

```
======== Misuse based IDS with modified data
Attacks removed from training data : ['smurf.', 'neptune.']
They are still present in the testing data.Accuracy might be affected.
MODEL TRAINED
Prediction completed for test data
Accuracy: 0.24963963626036212
Label: normal.
False Positives: 7
False Negatives: 4
```

Figure 4: Console output and metrics from Misuse-based IDS with 2 most frequent attacks removed

```
======== Misuse based IDS with modified data
Attacks removed from training data : ['xsnoop.', 'snmpgetattack.']
They are still present in the testing data.Accuracy might be affected.
Proportion of data points in the dataset with labels in attacks_to_remove: 2.49%
MODEL TRAINED
Prediction completed for test data
Accuracy: 0.9636474704876833
Label: normal.
False Positives: 7
False Negatives: 4
```

Figure 5: Console output and metrics from Misuse-based IDS with xsnoop and snmpgetattack attacks removed

7 DEVELOPMENT PROCESS

The process of development started with firstly understanding the data and its attributes. The dataset used for our project is the raw KDD dataset[5]. This involved reviewing basic network concepts such as the packet structure which would allow us to understand what each attribute meant. Once the dataset was available and loaded into a pandas dataframe it was easy to access for further operations.

The next step in the process was to explore and attempt to clean the data to understand the data in a better way. This process was done together and the data was discussed amongst us. After the data pre-processing was complete, one of us started to work on the anomaly-based IDS design and another one of us started working on the misuse-based IDS design. However, after spending about a week to a week and a half, we were able to get a basic design of

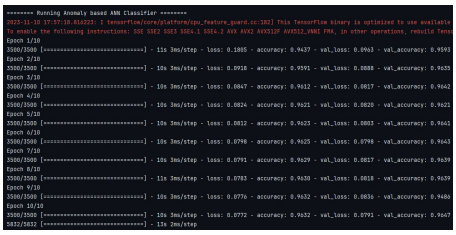
IDS up and running. Then, we worked on optimizing this neural net design using the Tensorflow Keras[7] library available in the Python programming language. The rest of the development was focused on hyperparameter tuning and testing it on the test data to get the false positive and negative ratios to understand the learning rate. Finally, we worked on the report detailing our process of development in detail.

REFERENCES

[1] [n. d.]. Kaggle. https://www.kaggle.com/
[2] [n. d.]. Scikit-Learn. https://scikit-learn.org/stable/
[3] [n. d.]. SNORT IDS tool. https://www.snort.org/
[4] [n. d.]. Suricata IDS tool. https://suricata.io/
[5] Steven Huang. [n. d.]. KDD Cup 1999 Data. https://www.kaggle.com/datasets/galaxyh/kdd-cup-1999-data
[6] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]
[7] Tensor-flow. [n. d.]. https://www.tensorflow.org/guide/keras/

A APPENDIX

A.1 Anomaly-Based IDS Console Output



(a) Console output from Anomaly-based IDS with 10 epochs

	precision	recall	f1-score	support
0	0.98	1.00	0.99	477
1	0.97	0.99	0.98	659
2	0.38	0.38	0.38	13
3	0.00	0.00	0.00	2
4	1.00	1.00	1.00	2620
5	1.00	0.94	0.97	95
6	1.00	0.97	0.99	184
7	1.00	1.00	1.00	5
8	0.00	0.00	0.00	1
9	1.00	1.00	1.00	3000
10	0.99	0.99	0.99	632
11	0.00	0.00	0.00	11
12	0.00	0.00	0.00	10
13	1.00	1.00	1.00	34801
14	1.00	1.00	1.00	50
15	0.85	0.99	0.92	36356
16	0.00	0.00	0.00	1
17	0.00	0.00	0.00	1
18	0.89	0.90	0.90	52
19	0.85	0.98	0.91	212
20	0.98	1.00	0.99	455
21	0.29	0.20	0.24	10
22	0.00	0.00	0.00	8
23	0.93	0.15	0.26	442
24	0.71	0.97	0.82	980
25	0.67	0.20	0.31	10
26	1.00	1.00	1.00	98455
27	0.00	0.00	0.00	4645
28	0.00	0.00	0.00	1444
29	1.00	1.00	1.00	1
30	0.00	0.00	0.00	7
31	0.00	0.00	0.00	1
32	0.93	0.94	0.93	961
33	0.00	0.00	0.00	1
34	1.00	0.20	0.33	5
35	0.50	0.50	0.50	2
36	0.62	0.62	0.62	8

(b) Console output from Anomaly-based IDS scikit classification report metrics

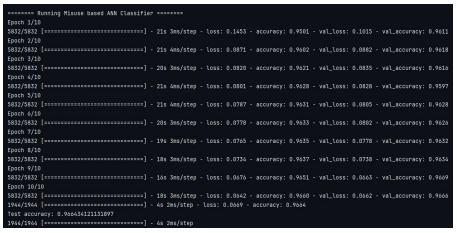
accuracy			0.96	186617
macro avg	0.58	0.54	0.54	186617
weighted avg	0.94	0.96	0.95	186617

False Positive Ratio: 0.001020360774563232
False Negative Ratio: 0.036732987884276355

(c) Console output from Anomaly-based IDS with accuracy and ratios

Figure 6: Console output from Anomaly-based IDS

A.2 Misuse-Based IDS Console Output



(a) Console output from Misuse-based IDS with 10 epochs

	precision	recall	f1-score	support
0	0.97	1.00	0.98	150
1	1.00	0.96	0.98	208
2	0.29	0.67	0.40	3
3	0.00	0.00	0.00	1
4	1.00	1.00	1.00	868
5	0.95	0.97	0.96	36
6	0.97	1.00	0.98	65
7	1.00	0.75	0.86	4
8	0.00	0.00	0.00	1
9	1.00	1.00	1.00	1026
10	0.98	1.00	0.99	224
11	0.00	0.00	0.00	4
12	0.00	0.00	0.00	5
13	1.00	1.00	1.00	11625
14	1.00	1.00	1.00	10
15	0.87	0.99	0.93	12102
16	0.00	0.00	0.00	1
17	0.00	0.00	0.00	1
18	0.95	0.95	0.95	21
19	1.00	0.90	0.95	82
20	0.98	1.00	0.99	147
21	0.00	0.00	0.00	6
22	0.00	0.00	0.00	3
23	0.92	0.15	0.25	156
24	0.68	0.99	0.80	307
25	0.00	0.00	0.00	4
26	1.00	1.00	1.00	32848
27	0.00	0.00	0.00	1545
28	0.73	0.56	0.63	446
29	0.00	0.00	0.00	1
30	0.00	0.00	0.00	2
32	0.93	0.96	0.94	298
33	0.00	0.00	0.00	1
34	0.00	0.00	0.00	3
35	0.00	0.00	0.00	1
36	0.00	0.00	0.00	1

(b) Console output from Misuse-based IDS scikit classification report metrics

accuracy			0.97	62206
macro avg	0.51	0.50	0.49	62206
weighted avg	0.94	0.97	0.95	62206

Overall False Positive Ratio: 0.0009590255418632103
Overall False Negative Ratio: 0.03356589396521236

(c) Console output from Misuse-based IDS with accuracy and ratios

Figure 7: Console output from Misuse-based IDS