Report

On

"Intrusion Detection System using Deep Learning Model"

As a part of Course

Cryptography (BITS F463)

By-

Aditya Tulsyan 2017B4A70740P

Ayush Singhal 2017A7PS0116P

Birla Institute of Technology and Science, Pilani

Sem I, 2019-20

Supervised by: Dr. Ashutosh Bhatia

OBJECTIVE

Building an **Anomaly-based Intrusion detection system** by training a deep learning model using Keras - TensorFlow APIs.

MOTIVATION

Intrusion is an effort that attempts to elude standard security mechanisms of the computer system. Intrusion detection is the process of monitoring and analyzing the events arising in a computer network to identify security breaches. Intrusion Detection System is the most important tool in maintaining the network's security.

INTRODUCTIONS

Intrusion detection is the process of identifying malicious activity targeted at computing and networking resources [1]. An Intrusion detection system monitors network traffic and raises an alert when suspicious activities are discovered. Intrusion detection systems are classified into two types- Network Intrusion Detection system and Host-based Intrusion detection system. Network intrusion detection systems are set up at points within the network to examine all the data going through the network itself. Host-based intrusion detection system run on hosts or devices present on the network. It detects incoming and outgoing packets from the device only and will give an alert if malicious activity is discovered.

An anomaly-based intrusion detection system is used to detect whether an activity is suspicious or not by comparing it with a statistical model of the past activity of the user. These have a better-generalized property as these models can be trained according to the applications and hardware configurations. Anomaly-based IDS is rendered the most effective among intrusion detection systems as they have no need to search for any specific pattern of the anomaly, but they rather just treat anything that does not match the profile as "Anomalous" [2].

Deep learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks [3]. It works by creating a statistical model based on the input and applies a non-linear transformation to its input until the output has reached an acceptable level of accuracy. "Deep" is inspired by the different number of processing layers the data has to go through.

Artificial neural networks are computing systems derived from the brain's neural networks. These systems learn to do tasks by considering examples. It is based on a group of connected units

called artificial neurons. Neurons having a connection can pass information from one to another. Each neuron has a weight associated with it that varies as learning proceeds. The neurons are organized in layers. Signals travel from the input layer to the output layer with the layers between them applying different kinds of transformation to them.

Dataset used in the model is CICIDS2017 [4]. It contains benign and most up to date common attacks. It also includes the results of the network traffic analysis with labeled flows based on timestamp, source, protocols, attacks, and other information in a CSV format.

In this project, we have trained a deep learning model to identify anomaly from the given data set. The model was trained on user data such as network packet information, software running information, system long events, operating system information, kernel information, etc.

METHODOLGY

IPython Notebook (Jupyter Notebook) was used as a computational environment to carry out the processing for easy creating and sharing of codes. Python modules such as pandas and numpy were used to batch process data. Matplotlib and Seaborn modules were used to represent and visualize the processed data in the form of charts.

- **Data Pre-Processing-**

Initial shape of dataset: 692703 rows x 79 columns

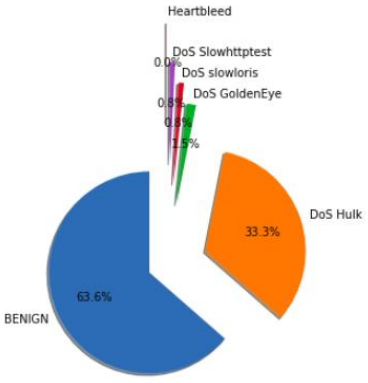
The first step in data preprocessing was to remove the rows with values infinity or nan in one of its columns.

This step resulted in around 1000 rows being dropped.

IDS dataset contains 79 parameters to represent the state of the machine in case of an attack. This is too large a value to be used to train the model. Hence, the rows with too large a variance or too little variance were removed as it will either not have a significant contribution in predicting the infiltration or will cause a large deviation and thus helps to avoid unnecessary time consumption during model training and testing.

This resulted in the dropping of 55 columns. Thus 24 parameters are present now to detect the state of the machine.

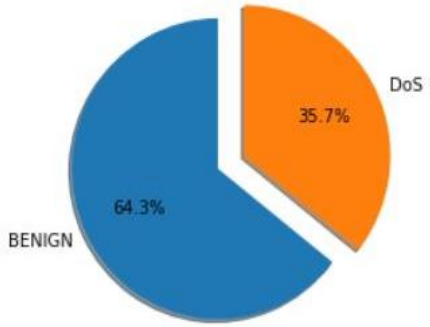
```
BENIGN              439972
DoS Hulk            230124
DoS GoldenEye        10293
DoS slowloris         5796
DoS Slowhttptest      5499
Heartbleed              11
Name:  Label, dtype: int64
```



According to observations, not much data was available for attacks such as DoS GoldenEye, Slowloris and other types of DoS attacks. So to get better accuracy, all such attacks are now instead classified as DoS attacks only. This is achieved by the following code.

```
# Since distribution opf various Denial of Service (DoS) is highly irregular, we have clubbed them for better results
df = df.replace('Heartbleed', 'DoS')
df = df.replace('DoS GoldenEye', 'DoS')
df = df.replace('DoS Slowhttptest', 'DoS')
df = df.replace('DoS slowloris', 'DoS')
df = df.replace('DoS Hulk', 'DoS')
df[' Label'].value_counts()
```

All this preprocessing resulted in 24 columns and around 700,000 rows as data. This is still too large a data for us to process due to constraints in our hardware. So, we have limited the number of rows to 100,000. Since the distribution is random, the ratio of benign to DoS attacks is maintained. The following is the final ratio of the processed data.



- **Training And Testing Data Generation-**

Importing various modules required to perform training and testing the data

Here we generate and separate the training and testing data to be used in our model. We replace the values in Label column that is Benign or DoS, by 0 or 1 so as to make it compatible with our model.Details of arrays X_train, X_test, y_train, y_test:

```
print('Train images shape:', X_train.shape)
print('Train labels shape:', y_train.shape)
print('Test images shape:', X_test.shape)
print('Test labels shape:', y_test.shape)
print('Train labels:', y_train)
print('Test labels:', y_test)
```

```
Train images shape: (80000, 23)
Train labels shape: (80000,)
Test images shape: (20000, 23)
Test labels shape: (20000,)
Train labels: [1 1 0 ... 1 1 0]
Test labels: [0 0 1 ... 1 0 1]
```

The dataset is divided into two parts, x for the input variables, and y for the output generation.

The input and output data are now divided into training and testing data using the train_test_split method. It will return the training and testing data for both x and y with a split in the ratio 0.8 to 0.2 respectively. Further, the data is converted from a data frame to a numpy array to make it compatible with processing in the model.

Since the data has large varying values in it, we need to normalize it before passing it to the model. The fit_transform function is used to transform it into the range 0 to 1.

- **Input Parameters-**

A total of 23 parameters are left after processing and preparing the data for input of Artificial Neural Networks. Here are the input parameters along with their significance.:

1. Destination Port: The source port serves analogous to the destination port, but is used by the sending host to help keep track of new incoming connections and existing data streams
2. Flow duration: Duration of the flow in Microsecond
3. Total Forward Packet: Total packets in the forward direction
4. Total Backward packets: Total packets in the backward direction
5. Fwd Packet Length max: Maximum size of the packet in the forward direction
6. Fwd Packet Length min: Minimum size of the packet in the forward direction
7. Fwd Packet Length Mean: Mean size of the packet in the forward direction
8. Fwd Packet Length Std: Standard deviation size of the packet in the forward direction
9. Bwd Packet Length Min: Minimum size of the packet in the backward direction
10. Bwd Packet Length Mean: Minimum size of the packet in the backward direction
11. Flow Bytes/s: Number of flow bytes per second
12. Flow Packets/s: Number of flow packets per second
13. Min Packet Length: Minimum length of a packet
14. Packet Length Mean: Mean length of a packet
15. Packet Length Std: Standard deviation length of a packet
16. PSH Flag Count: Number of packets with PUSH
17. ACK Flag Count: Number of packets with ACK
18. Down/Up Ratio: Download and upload ratio

19. Average Packet Size: Average size of the packet
20. Avg Fwd Segment Size: Average size observed in the forward direction
21. Avg Bwd Segment Size: Average number of bytes bulk rate in the forward direction
22. Subflow Fwd Packets: The average number of packets in a sub-flow in the forward direction
23. Subflow Bwd Packets: The Mean length of a packet

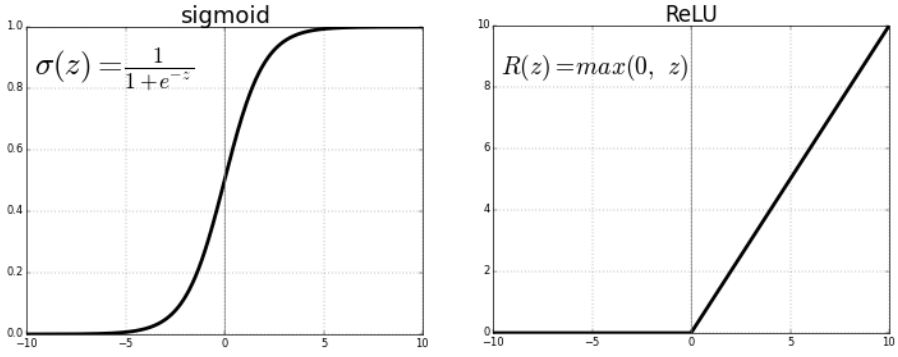
- **Deep Learning Model-**

We have used the Sequential Model for our neural network. The Sequential model API is a way of creating deep learning models where an instance of the Sequential class is created and model layers are created and added to it. The layers are created and passed to the Sequential as an array. As shown in the above code, we are adding layers piecewise via the Sequential object. All the layers are densely connected, that is, they are fully connected with each other.

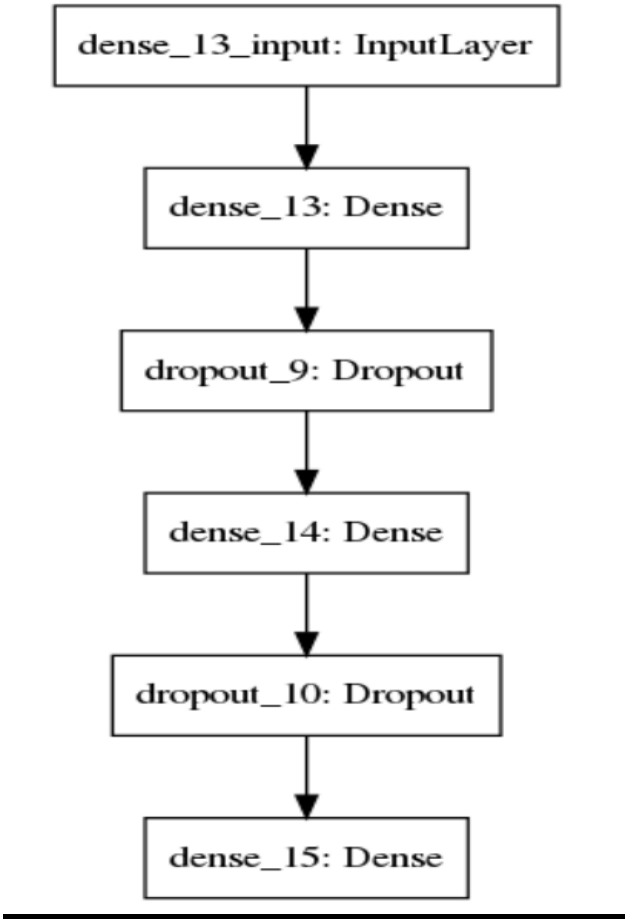
The first layer contains an input parameter of 23 and an output parameter of 256. The activation method used is ‘relu’. It stands for Rectified Linear Unit. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.

The second layer is a Dropout layer. It applies dropout to the input, that is, it randomly sets a fraction (0.5 in this case) of input units to 0 at each update during training time, which helps in preventing overfitting.

Next is a layer with 128 outputs and ‘relu’ activation. After that, we have another dropout layer with a 0.25 fraction rate. Last is our output layer with 1 output and activation function ‘sigmoid’. Sigmoid is an activation function of form f(x) = 1/(1+exp(-x)).



Model.compile configures the model for training. It defines the loss function, the optimizer and the metrics. A loss function is one of the two parameters required to compile a model. Here loss function is ‘binary_crossentropy’. Binary_crossentropy specifies that we have two classes between which we need to differentiate. It is mainly used for classification problems. The optimizer used here is SGD(Stochastic Gradient Descent Optimizer) with a learning rate of 0.01. The last parameter is metrics which is the list of metrics to be evaluated by the model during training and testing. Generally, we used [‘accuracy’] as the metric.



Graphical representation of the deep learning model used

- **Training the model**

Model.fit is used to train the model in Keras. Here we need to specify the input data(X_train), labels(Y_train), number of epochs/iterations(100) and batch size. This returns the history of model training. History consists of model accuracy and losses after each epoch. As the dataset is very big and we cannot fit complete data at once, we use batch size. Only this number of samples will be loaded into memory and processed.

Here is the training data for our model.

```
In [76]: model.fit(X_train, y_train, epochs=20,
                   verbose=1, batch_size=100)
```

```
Epoch 1/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.2288 - accuracy: 0.8868
Epoch 2/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.2206 - accuracy: 0.8891
Epoch 3/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.2157 - accuracy: 0.8916
Epoch 4/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.2100 - accuracy: 0.8946
Epoch 5/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.2040 - accuracy: 0.8965
Epoch 6/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.2004 - accuracy: 0.8984
Epoch 7/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1976 - accuracy: 0.8994
Epoch 8/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1928 - accuracy: 0.9041
Epoch 9/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1899 - accuracy: 0.9051
Epoch 10/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1852 - accuracy: 0.9086
Epoch 11/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1828 - accuracy: 0.9090
Epoch 12/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1813 - accuracy: 0.9116
Epoch 13/20
80000/80000 [==============================] - 2s 24us/step - loss: 0.1773 - accuracy: 0.9141
Epoch 14/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1742 - accuracy: 0.9155
Epoch 15/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1718 - accuracy: 0.9184
Epoch 16/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1692 - accuracy: 0.9195
Epoch 17/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1673 - accuracy: 0.9204
Epoch 18/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1656 - accuracy: 0.9218
Epoch 19/20
80000/80000 [==============================] - 2s 23us/step - loss: 0.1621 - accuracy: 0.9243
Epoch 20/20
80000/80000 [==============================] - 2s 22us/step - loss: 0.1603 - accuracy: 0.9256
```

```
Out[76]: <keras.callbacks.callbacks.History at 0x7fa5dd785080>
```

RESULTS

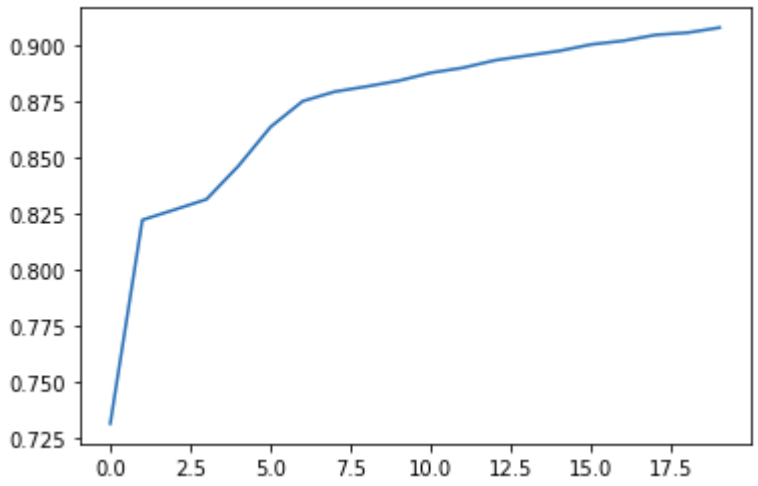
After applying the training model for 80000 training data and testing it over 20000 dataset, our two-layer neural network model with 256 and 128 neurons was successful in achieving an accuracy of over 90 percent. That is, our model can and will successfully predict an infiltration every 90 out of 100 times. Dataset used for testing the data containing 20000 rows of data will be chosen randomly from initial data during every iteration, hence making our results more reliable.

```
In [77]: score = model.evaluate(X_test, y_test, verbose=1)
         print('Test loss:', score[0])
         print('Test accuracy:', score[1])
```

```
20000/20000 [==============================] - 0s 19us/step
Test loss: 0.1791962167084217
Test accuracy: 0.9003999829292297
```

```
# Creating the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

```
array([[ 6977,   116],
       [ 1781, 11126]])
```



Epochs vs Accuracy

OBSERVATIONS

With successful identifications on our model, we can confidently proclaim that our model can be used as a back end engine to an intrusion detection system application that can be mounted on the border of any computer network. Due to hardware limitations and time constraints we restricted the data to 100k samples and maximum of 20 epochs. And the result we got from our model of a neural network with two hidden layers and 256 and 128 neurons is above 90 percent accuracy which seems quite satisfactory. From the result shown we have got, it can be easily concluded that on training whole 700k samples and for 50 - 100 epochs on can attain quite high accuracy on test data. Further, from the given graph, we can see the accuracy increase decreases as the number of epochs are increased and reach an asymptote of around 92%.

REFERENCES

[1]Nascimento, Gustavo, and Miguel Correia. “Anomaly-Based Intrusion Detection in Software as a Service.” 2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W), 2011, doi:10.1109/dsnw.2011.5958858.

[2]Mirza, Tamim. “Building an Intrusion Detection System Using Deep Learning.” Medium, Towards Data Science, 10 Oct. 2018, https://towardsdatascience.com/building-an-intrusion-detection-system-using-deep-learning-b9488332b321.

[3]“Artificial Learning, Machine Learning and Deep Learning: Know The Difference.” Systweak Software, 16 Dec. 2016, https://blogs.systweak.com/artificial-learning-machine-learning-and-deep-learning-know-the-difference/.

[4] “Artificial Learning, Machine Learning and Deep Learning: Know The Difference.” Systweak Software, 16 Dec. 2016, https://blogs.systweak.com/artificial-learning-machine-learning-and-deep-learning-know-the-difference/.

[5] Sharafaldin, Iman, et al. “Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization.” *Proceedings of the 4th International Conference on Information Systems Security and Privacy*, 2018, doi:10.5220/0006639801080116.