

DDoS Detection Using Long Short-Term Memory Recurrent Neural Network

Youngho Kim

June 2020

1 Introduction

Due to the great raise of the companies, government, and even the schools are in danger of the distributed denial of service(DDoS) attacks. The DDoS is achieved by using attacking bots or compromised Internet of Things (IoT) devices to exhaust the capacity of the server or the network bandwidth and applications resources[3].

According to the research held by Nexusguard in 2016, it proved that the frequency of DDoS attacks has been increased enormously by 83% in the second quarter of 2016[2]. According to Mansfield-Devine, the tremendous increase in DDoS attacks is related to the attackers' motivational factors towards the money, revenge, and destruction to perform other attacks [1].

To prevent from being attacked, many researchers tried several machine learning methods to increase the accuracy of their detection models. However, new attacks always come to the threat of the society. In order to at least decrease the denial of service attacks, the analysis of 120K of the 22 different attacking packets was used to classify the normal and abnormal packets that is being received from the client.

Through out the report, the strength of the LSTM RNN(Long Short-term Memory Recurrent Neural network) will be introduced and handled to make a model to detect the DDoS attacks. Also, well-known methods such as SVM and KNN algorithms will be covered in this report.

2 Data pre-processing

Index of /iscxdownloads
















Name	Last modified	Size	Description
 Parent Directory		-	
 CIC-IDS-2017/	2018-06-07 17:12	-	
 CIC-IDS/	2019-04-05 11:03	-	
 CICAndMal2017/	2018-08-31 14:05	-	
 CICAndroidAdGMalware.>	2017-10-21 17:00	-	
 CICFlowMeter/	2018-07-05 08:25	-	
 ISCX-AndroidBot-2015/	2016-02-02 13:35	-	
 ISCX-AndroidValidati.>	2016-02-02 08:55	-	
 ISCX-Bot-2014/	2016-01-28 09:12	-	
 ISCX-IDS-2012/	2016-05-24 11:57	-	
 ISCX-SlowDoS-2016/	2017-06-02 08:42	-	
 ISCX-Tor-NonTor-2017/	2018-11-15 08:38	-	
 ISCX-URL-2016/	2019-03-05 16:15	-	
 ISCX-VPN-NonVPN-2016/	2016-06-30 11:15	-	
 NSL-KDD/	2015-12-09 13:43	-	

Figure 1: KDD 120k packet dataset

The NSL-KDD data-set from University of New Brunswick Lab includes 125,973 packets that has log of 22 different types of attacks. The documentation was downloaded by following the link: <https://iscxdownloads.cs.unb.ca/iscxdownloads>. It contains 42 features with labels in it. The label depicts whether the packet is normal or not(attack or not) [4].

2.1 Data cleansing

Since NSL-KDD data-set was clean that it does not contain unnecessary features or string values, it was great to use as a train set for the LSTM model. It labeled the packets as "normal" and "anomaly" to classify whether the packet is from the attacker or not. Also, by using "OneHotEncoder" and "OrdinalEncoder" libraries from "sklearn" library, it was helpful to alter the order-like values to numeric numbers (e.g: TCP, UDP to 0, 1).

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...
1	0	udp	other	SF	146	0	0	0	0	0	...
2	0	tcp	private	S0	0	0	0	0	0	0	...
3	0	tcp	http	SF	232	8153	0	0	0	0	...
4	0	tcp	http	SF	199	420	0	0	0	0	...

5 rows × 42 columns

Figure 2: Features of the KDD dataset

To train the LSTM model, two different train sets are necessary. One is called X_train that contains the features of the packets and the other is y_train that contains the labels of each of the packets whether it classifies the attack packets. To test, X_test and y_test are also used that has 40% of random packets from the train set. The test set contains random attacking packets so that it can objectively measure the accuracy of the model.

2.2 Data features and Attack types

The NSL-KDD data-set contains 42 clean and well-defined features. Features such as protocol type, flag, attack type, and service were re-sized to be used in Tensorflow. In order to use LSTM model, 3d shape array was used for the 42 features.

The 42 features of NSL-KDD Dataset.					
S/N	Field Name	Description	S/N	Field Name	Description
1	duration	continuous	22	is_guest_login	symbolic
2	protocol type	symbolic	23	count	continuous
3	service	symbolic	24	srv_count	continuous
4	flag	symbolic	25	error_rate	continuous
5	src_bytes	continuous	26	error_rate	continuous
6	dst_bytes	continuous	27	srv_error_rate	continuous
7	Land	symbolic	28	srv_error_rate	continuous
8	wrong_fragment	continuous	29	same_srv_rate	continuous
9	urgent	continuous	30	diff_srv_rate	continuous
10	Hot	continuous	31	srv_diff_host_rate	continuous
11	num_failed_logins	continuous	32	dst_host_count	continuous
12	logged_in	symbolic	33	dst_host_srv_count	symbolic
13	num_compromised	continuous	34	dst_host_same_srv_rate	continuous
14	root_shell	continuous	35	dst_host_diff_srv_rate	continuous
15	su_attempted	continuous	36	dst_host_same_src_port_rate	continuous
16	num_root	continuous	37	dst_host_srv_diff_host_rate	continuous
17	num_file_creations	continuous	38	dst_host_error_rate	continuous
18	num_shells	continuous	39	dst_host_srv_error_rate	continuous
19	num_access_file	continuous	40	dst_host_error_rate	continuous
20	num_outbound_cmds	continuous	41	dst_host_srv_error_rate	continuous
21	is_host_login	symbolic	42	Attack type	label

Figure 3: 42 different features

The best way to train a model is to train the specific attack types to prevent from being attacked by the same type of attacks. To reinforce the model, the 22 different types of attacks were used to train the model. Each attack has different values of the features [4].

The 22 Attack Types of NSL-KDD Dataset.					
S/N	Name	Attack Type	S/N	Name	Attack Type
1	back	DoS	12	perl	u2r
2	buffer_overflow	u2r	13	phf	r2l
3	ftp_write	r2l	14	pod	DoS
4	guess_passwd	r2l	15	portsweep	probe
5	imap	r2l	16	rootkit	u2r
6	ipsweep	probe	17	satan	probe
7	land	DoS	18	smurf	DoS
8	loadmodule	u2r	19	spy	r2l
9	multihop	r2l	20	teardrop	DoS
10	neptune	DoS	21	warezclient	r2l
11	nmap	probe	22	warezmastert	r2l

Figure 4: 22 different types of attacks

3 Packet Analysis with Tensorflow

Tensorflow has the great libraries that contains RNN, DNN, and many different machine learning methods. The Tensorflow Keras was used for this research. 2 hidden layers were used for the deep neural network, and mean absolute error method was used to check the loss from the data while learning. "adam" method was used for the DNN optimization. According to the documentation, "Adam is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks" [1]. Figure 5 shows the sample loss and accuracy for the trained data-set using LSTM method.

```
history = model.fit(X_train, y_train, epochs=30, validation_data = (X_test, y_test))# validation_data=(X_test, y_test)
```

Train on 125973 samples, validate on 52977 samples

Epoch 1/30
125973/125973 [=====] - 19s 151us/sample - loss: 0.0518 - acc: 0.9612 - val_loss: 0.0374 - val_acc: 0.9708

Epoch 2/30
125973/125973 [=====] - 17s 133us/sample - loss: 0.0347 - acc: 0.9733 - val_loss: 0.0289 - val_acc: 0.9786

Epoch 3/30
125973/125973 [=====] - 17s 132us/sample - loss: 0.0284 - acc: 0.9856 - val_loss: 0.0245 - val_acc: 0.9880

Epoch 4/30
125973/125973 [=====] - 17s 135us/sample - loss: 0.0237 - acc: 0.9887 - val_loss: 0.0219 - val_acc: 0.9884

Epoch 5/30
125973/125973 [=====] - 16s 125us/sample - loss: 0.0216 - acc: 0.9898 - val_loss: 0.0200 - val_acc: 0.9904

Epoch 6/30
125973/125973 [=====] - 15s 121us/sample - loss: 0.0201 - acc: 0.9904 - val_loss: 0.0214 - val_acc: 0.9909

Epoch 7/30
125973/125973 [=====] - 15s 119us/sample - loss: 0.0191 - acc: 0.9906 - val_loss: 0.0204 - val_acc: 0.9911

Epoch 8/30
125973/125973 [=====] - 15s 118us/sample - loss: 0.0184 - acc: 0.9909 - val_loss: 0.0187 - val_acc: 0.9915

Epoch 9/30
125973/125973 [=====] - 15s 120us/sample - loss: 0.0179 - acc: 0.9911 - val_loss: 0.0194 - val_acc: 0.9914

Epoch 10/30
125973/125973 [=====] - 16s 128us/sample - loss: 0.0172 - acc: 0.9912 - val_loss: 0.0158 - val_acc: 0.9918

Figure 5: Result of LSTM method

3.1 Model Evaluation

According to Figure 6, from 52977 sample test packets, the value of loss was 0.013 and value of accuracy was 0.9929 accordingly. This result of evaluation shows that LSTM RNN is almost a perfect method to prevent and protect the 22 different types of attacks.

```
In [24]: val_loss, val_acc = model.evaluate(X_test, y_test)
print("Value Loss: ", val_loss)
print("Value Accuracy: ", val_acc)

52977/52977 [=====] - 2s 43us/sample - loss: 0.0133 - acc: 0.9930
Value Loss: 0.013315000836290137
Value Accuracy: 0.9929781
```

Figure 6: Model Evaluation

3.2 Training and Validation

After 30 epochs of train, Figure 7 shows the extremely low loss of the model. This proves that the model is catching the attacking packets correctly. The final value of the loss is 0.0111

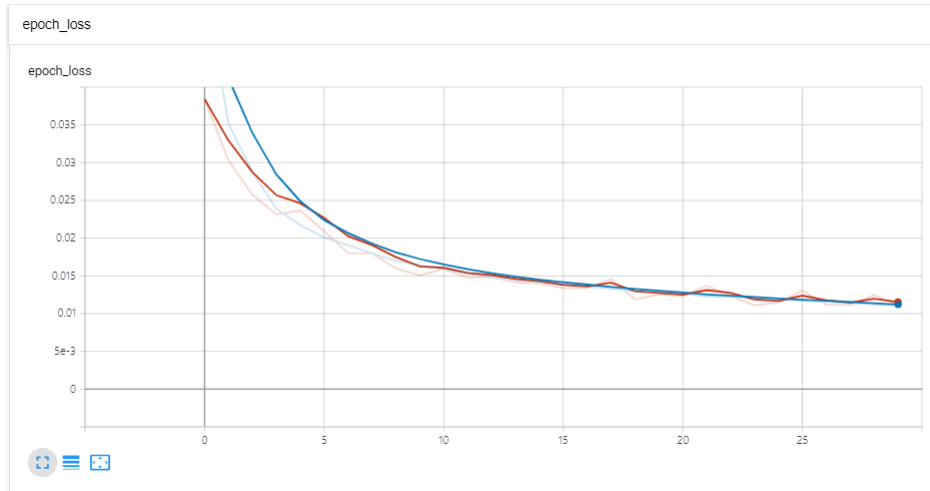


Figure 7: Training and Validation loss graph in TensorBoard

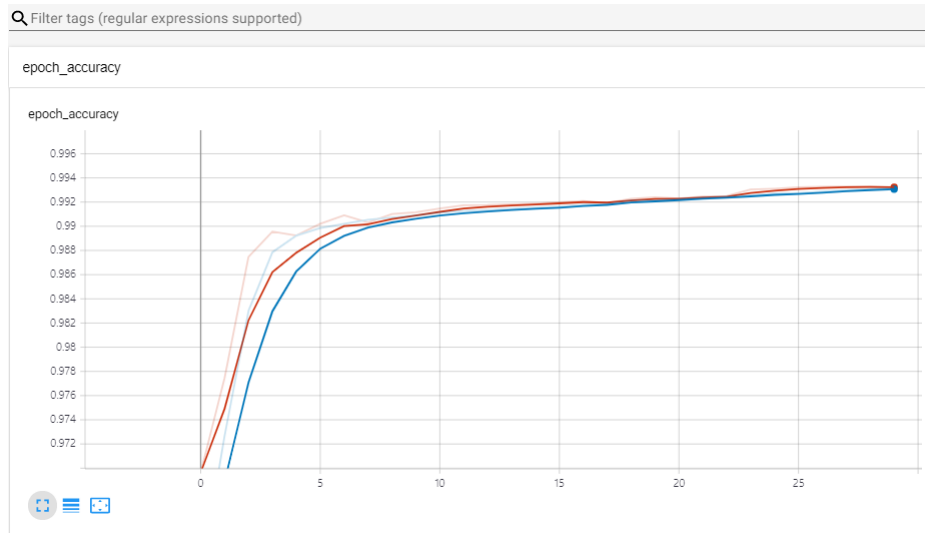


Figure 8: Training and Validation accuracy graph in TensorBoard

Name	Smoothed	Value	Step	Time	Relative
DDoS_LSTM_RNN1591881473\train	0.993	0.9931	29	Thu Jun 11, 22:26:35	8m 20s
DDoS_LSTM_RNN1591881473\validation	0.9932	0.9932	29	Thu Jun 11, 22:26:35	8m 20s

Figure 9: TensorBoard Contexts

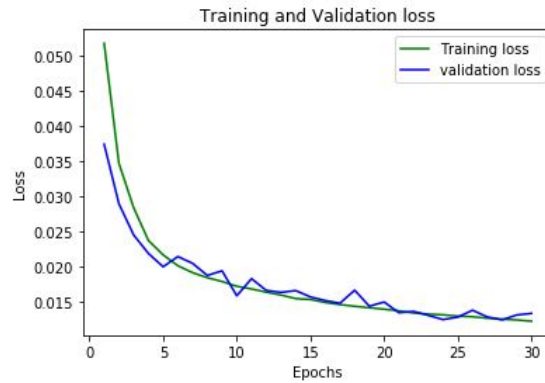


Figure 10: Training and Validation loss graph

The accuracy of the model was extremely high. However, there were almost no changes to the accuracy even if the epoch is increased to 40. The accuracy of the final model is near to 99.29%.

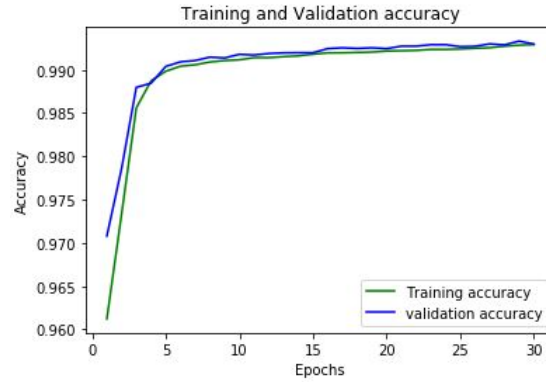


Figure 11: Training and Validation accuracy graph

4 Compare to SVM model

The well known approach to detect the DDoS is SVM approach that uses hyper-plane and labels to classify the packets. In order to compare LSTM RNN model to SVM model, same data-set was used to train and test the model. For the classifier, the linear kernel was used to fit the models. Figure 9 depicts the accuracy of the SVM model.

```
from sklearn import svm
classifier = svm.SVC(kernel = 'linear')
classifier.fit(X_train, y_train)
y_predict = classifier.predict(X_test)

#Evaluation (result)
from sklearn import metrics
print("Accuracy is: ", metrics.accuracy_score(y_test, y_predict))
```

Accuracy is: 0.9601336429016366

Figure 12: Accuracy of SVM method

5 Detection Demo

By using random test packets (about 11000 packets), the finalized result was tested in VSCode environment.

References

- [1] Vitaly Bushaev. Adam — latest trends in deep learning optimization., 2018.
- [2] A. Rathod D. Kshirsagar, S. Sawant and S. Wathore. *Procedia Computer Science, 85, International Conference on Computational Modelling and Security*. D.Kshirsagar, 2016.
- [3] NEXUSGUARD. q2-2016-ddosthreat-report, 2017.
- [4] Peter.Ken.Bediako. *Long Short-Term Memory Recurrent Neural Network for detecting DDoS flooding attacks within TensorFlow Implementation framework*. Peter.Ken.Bediako, 2017.