Time-related Network Intrusion Detection Model: A Deep Learning Method

†Yun Lin, †Jie Wang, †Ya Tu, ‡*Lei Chen, †Zheng Dou
†*College of Information and Communication Engineering, Harbin Engineering University*
Harbin, P.R.China
‡*Department of Information Technology, College of Engineering and Computing, Georgia Southern University*
Georgia, U.S.A.
Corresponding Author: LChen@georgiasouthern.edu

***Abstract*—Network Intrusion Detection System (NIDS) has become a strong tool to alarm attacks in computer and communication systems. Machine learning, especially deep learning, has made huge success in fields of industry and academic. Network intrusion activity can be a time series event. In this paper, we adopt a time-related deep learning approach to detect network intrusions. A stacked sparse autoencoder (SSAE) is first built to extract the features with the greedy layer-wise strategy. And then, we propose a time-related intrusion detection system based on the variants of Recurrent Neural Network (RNN). We study the performance of proposed approach on the binary classification with a benchmark dataset UNSW-NB15. Based on the study of time-related parameter $time\_step$, it is proved that our time-related model is effective for intrusion detection. The experiment results show that the accuracy of the proposed approach reaches over 98% and the false alarm rate is as low as 1.8%. The performance of our model is superior to some approaches based on deep neural network or some machine learning methods.**

***Index Terms*—Network Intrusion Detection, Stacked Sparse Autoencoder, Recurrent Neural Network, Time-related Model**

I. Introduction

With the increase of network penetration rate and the rapid development of network application, cyber security has become one of the biggest challenges that network developers and managers have to face [1]. Activities that attempt to bypass security mechanisms are usually considered as intrusions. A large amount of network intrusion incidents are reported each year. Network Intrusion Detection System is regarded as a very important foundation and prerequisite for identifying malicious activities and handling cyber attacks in network traffic. NIDS monitors the network traffic and scans the system for suspicious activities and alerts the alarm.

Many studies on NIDS involve machine learning. Since 2006, the theory of deep learning introduced by Hinton has achieved remarkable results in academic and industry [2]. The most frequently used deep learning methods include Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Generative Adversarial Network (GAN) and Restricted Boltzmann Machine (RBM). Deep learning is proved to be a promising method in different fields such as computer vision, medical, finance, and advertising et al. [3].

The behaviours of network intrusions can also be time-related events. RNN is proved to well process time-related sequence data, making it popular to handle problems such as machine translation, image captioning, and speech recognition. Some existed methods upon RNN are presented to solve network intrusion detection problems. In 2016, Kim et al. explored the possibility of applying RNN to intrusion detection and used a variant of RNN to build an intrusion detection model. [4]. They extracted instances from KDD Cup99 dataset completing the experiment on finding the super parameters and evaluating the model performance. In 2017, Yin et al. used standard RNN to build an IDS and evaluated their approach with a benchmark dataset NSL-KDD [5]. They studied the influence of hidden nodes and learning rate. However, the accuracy of the proposed model was not quite satisfying. In 2018, Xu et al. constructed a new DNN model which applied Gated Recurrent Unit (GRU) and Multilayer Perceptron (MLP) to extract the data information [6]. Their simulation results showed that the GRU cell can be more effective than the Long Short-term Memory (LSTM) cell for intrusion detection. In [7], Anani et al. used the full KDD Cup99 dataset to compare the model detection performance based on LSTM, Bidirectional Long Short-term Memory (BiLSTM), Skip-LSTM and GRU. The result illustrated that the GRU achieved superior performance than others. In [8], to enhance the ability of classification, Agarap et al. first built a GRU model and then introduced the linear support vector machine to replace the softmax classifier. Similarly, the L2-SVM loss function was adopted replacing the cross-entropy function. In [9], authors used the unsupervised version of different variants of RNN cell to construct an autoencoder for intrusion detection. In [10], Roy et al. picked samples from UNSW-NB15 dataset and built a BiLSTM RNN network. Five features were selected reaching an accuracy over 95%.

However, most of previous work ignore the influence of the length of history information on performance. And the data they use is only part of the dataset which is not consistent with the real traffic. In this paper, we adopt different variants of RNN combined with stacked sparse autoencoder for building an intrusion detection system. We have found that the use of autoencoder helps to represent the data. And the number of historical data truly has impact on the detection performance. As for the result, our approach can reach more than 98% in detection rate, less than 1.8% in false alarm rate. The main contributions include the followings:

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

1) We present a stacked sparse autoencoder to extract low-dimension features. Parameters, such as the value of sparse constant and the number of layers and units, are taken into consideration. The greedy layer-wise strategy is adopted to train the autoencoder for better performance.
2) We compare the performance of four different variants of RNN cell. We adopt the batch normalization strategy to take the full use of learning ability of deep networks. By comparing the impact of different $timestep$ on performance, our time-related model is proved to be effective for time series problems.
3) We use the entire UNSW-NB15 dataset instead of partial data for experiments. Our approach achieves high detection accuracy and low false alarm rate in binary classification problems.

The remainder of this paper is organized as follows. Section II provides related methodology of autoencoder and variants of RNN. In Section III, the details of the proposed work are illustrated. Section IV describes the experimental results and analysis. Section V draws the conclusion of this paper and casts our future work.

II. Basic Theory

A. Sparse Autoencoder

Autoencoder (AE) has a three-layer structure, including the input layer, the hidden layers and the output layer [11]. The autoencoder can be divided into encoder and decoder. Encoder is usually for data compression and feature extraction. Due to the reduction of the number of hidden layer neurons, encoding maps the same type of data to specific neurons to complete dimensionality reduction. And the process to transform the extracted data into the shape of input data is called decoding.

Encoding can well implement the non-linear transformation from high dimensional data space to low dimensional data space.

$$H = f_\theta(X) = \sigma(W_{ij}X + b_{ij}) \tag{1}$$

where W is the weight matrix and b stands for the bias vector. And σ is the representation of activation function. Decoding can be regarded as a reverse process of encoding.

$$Y = g_\theta(X) = \sigma(W_{jk}X + b_{jk}) \tag{2}$$

In the decoding phase, the Mean Squared Error is usually adopted as the reconstruction error function J:

$$J(W,b) = \frac{1}{2N}\sum_{n=1}^{N}\|Y_n - X_n\|^2 \tag{3}$$

Sparse AE is a variant of the autoencoder [12]. Sparseness is reflected in the fact that only a few neurons are activated by introducing a constraint. In this way, the activated neurons can be data-dependent: different inputs will result in different neurons activated. Selective activation of neurons helps to make encoded data well characterized and easier to distinguish.

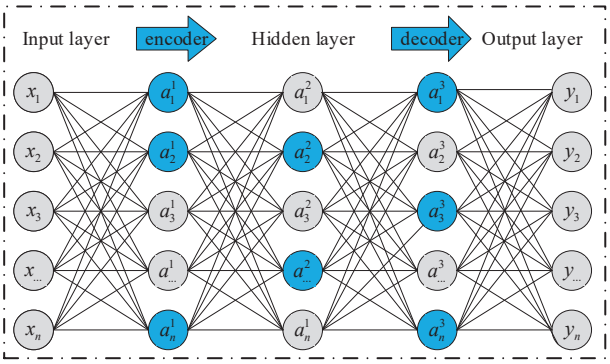


Fig. 1. Structure of Stacked Sparse Autoencoder.

Generally, a constant ρ is the proportion of activated neurons. The average activation can be $\hat{\rho}_j = \frac{1}{N}\sum_{i=1}^{N} n_j(x_i)$ where N is the number of hidden neurons. The Kullback-Leibler (KL) divergence can ensure that $\hat{\rho}_j$ approaches the constant ρ

$$KL(\rho\|\hat{\rho}_j) = \rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j} \tag{4}$$

Therefore the error function in the sparse autoencoder is as follow:

$$J_{sparse} = J(W,b) + \mu\sum_{j=1}^{N} KL(\rho\|\hat{\rho}_j) \tag{5}$$

where μ is a constant.

B. The variants of RNN

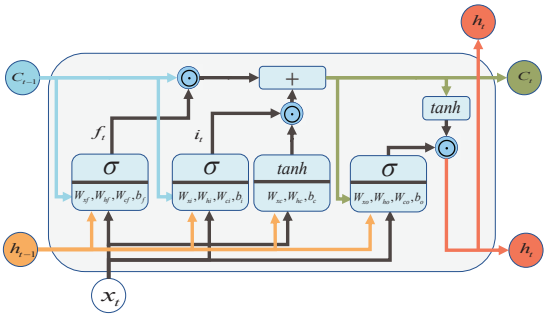
The greatest challenge of RNN is the gradient vanishing problem [13]. To tackle this, a number of variants of RNN are introduced.

Long Short-Term Memory is proposed by Hochreiter and Schmidhuber who used three gates to solve the gradient vanishing problem [14]. Gating mechanism works through storing the history memory so that the activation value h_t of the hidden unit at time step t can be calculated using information at different times [15]. The structure of a LSTM cell is shown in Fig.2(a). The process to obtain a updated activation of neural unit is as follow:

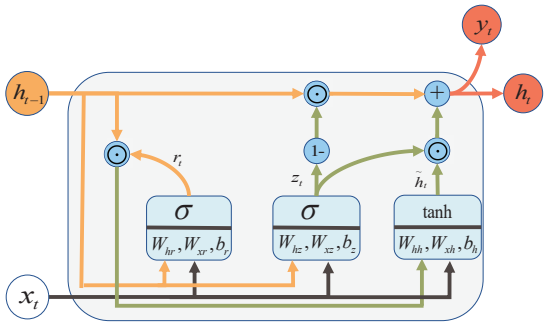
$$\begin{cases} i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \\ f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \\ c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \\ o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \\ h_t = o_t \odot \tanh(c_t) \end{cases} \tag{6}$$

where W is the weight matrix and b is the bias vector. σ and $tanh$ are activation functions.

Similar to LSTM, the Gated Recurrent Unit model uses only two gates to address the problem of vanishing gradient [16]. The structure of GRU cell is shown in Fig.2(b). r_t stands for the reset gate and z_t is update gate. Suppose the current



(a) Structure of the LSTM Cell.



(b) Structure of the GRU Cell.

Fig. 2. Variants of Recurrent Neural Network.

input is x_t and the last time step activation is r_{t-1}. h_t, r_t and z_t are calculated as follows:

$$\begin{cases} r_t = \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ z_t = \sigma(x_t W_{xz} + h_{t-1} W_{hz} + b_z) \end{cases} \tag{7}$$

The reset of history h_{t-1} is then obtained through the Hadamard Product of r_t and h_{t-1}. A new candidate activation can be obtained:

$$\tilde{h}_t = \tanh(x_t W_{xh} + W_{hh}(r_t \odot h_{t-1}) + b_h) \tag{8}$$

Finally, the activation of hidden unit at time t is updated:

$$h_t = (1 - z_t) h_{t-1} + z_t \tilde{h}_t \tag{9}$$

III. Proposed Work

In this section, we propose our time-related intrusion detection model. The framework of proposed model is shown in Fig.3.

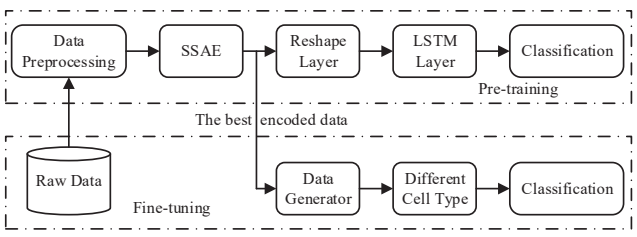


Fig. 3. Proposed Framework for Intrusion Detection using Deep Learning.

A. Dataset

UNSW-NB15 created by Moustafa et al. overcomes the shorcoming existed in KDDCUP'99 and gradually becomes one of benchmark dataset for intrusion detection [17]. The normal records are collected from real network environment. And there exist nine different kinds of attacks which include Fuzzers, Reconnaissance, Shellcode, Analysis, Backdoors, DoS, Exploits, Generic, and Worms. Every sample in UNSW-NB15 has 42 features as described in table I where $attack\_cat$ and $label$ are labels. The features can be divided into Flow features, Basic features, Content features, Time features and Additional generated features. In this research, the entire UNSW-NB15 dataset is adopted for performance evaluation.

TABLE I
UNSW-NB15 Dataset

Number	Description	Number	Description
1	dur	23	dwin
2	proto	24	tcprtt
3	service	25	synack
4	state	26	ackdat
5	spkts	27	smean
6	dpkts	28	dmean
7	sbytes	29	trans_depth
8	dbytes	30	response_body_len
9	rate	31	ct_srv_src
10	sttl	32	ct_state_ttl
11	dttl	33	ct_dst_ltm
12	sload	34	ct_src_dport_ltm
13	dload	35	ct_dst_sport_ltm
14	sloss	36	ct_dst_src_ltm
15	dloss	37	is_ftp_login
16	sinpkt	38	ct_ftp_cmd
17	dinpkt	39	ct_flw_http_mthd
18	sjit	40	ct_src_ltm
19	djit	41	ct_srv_dst
20	swin	42	is_sm_ips_ports
21	stcpb	43	attack_cat
22	dtcpb	44	label

In order to meet the requirements of the input format in neural network, data preprocessing is needed. One-hot encoding is used to transform the nominal features, $service$, $state$ and $proto$ into numerical values. The Min-Max technique is thereafter adopted to normalize the data as follow:

$$x^* = \frac{x - min}{max - min} \tag{10}$$

B. Model training

The model training consists of two steps: pre-training and fine-tuning. The pre-training is aimed to train a stacked sparse autoencoder that produces well separated data as described in the upper half of Fig.3. The optimal weights and bias of each layer in the stacked sparse auto-encoded network will be sequentially obtained through the greedy layer-wise pre-training method [18].

The fine-tuning process helps to find the optimal structure of the time-series model, including the cell type, the number of layers and the number of units. We adopt four different kinds of cell types, LSTM, BiLSTM, GRU and Bi-directional

GRU (BiGRU). Based on this, different number of layers and neuron units as well as the parameter $time\_step$ are tested to obtain the optimal result as shown in the lower half of Fig.3.

C. Evaulation

For the binary classification problem, the confusion matrix is defined in table II. A few metrics based on the confusion matrix are used to evaluate the simulation results.

TABLE II
CONFUSION MATRIX FOR BINARY CLASSIFICATION

Predicted \ Reality	Attack	Normal
Attack	TP	FP
Normal	FN	TN

Based on the above definition, $Accuracy$, False Alarm Rate (FAR), $Precision$ and $Recall$ can be obtained:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{11}$$

$$FAR = \frac{1}{2}(\frac{FP}{FP + TP} + \frac{FN}{FN + TN}) \tag{12}$$

$$Precision = \frac{TP}{TP + FN} \tag{13}$$

$$Recall = \frac{TP}{TP + FP} \tag{14}$$

IV. EXPERIMENT

In this research, Python is used for coding and Keras is used to build the neuron network. In the data preprocessing stage, the dimension of the input transforms from 42 to 196. The value of each feature ranges from 0 to 1.

Model training contains two stages. The purpose of the first stage is to find a proper SSAE to create well-separately encoded-data. Based on the theoretical analysis and our previous work experience, it is clear that the hidden layer number, the unit number, and the sparse constant are among the main factors determining the performance of SSAE. For this purpose, experiments are conducted to determine a suitable SSAE structure. Different SSAE are trained using the greedy layer-wise method and a single LSTM layer is connected after the SSAE. The $time\_step$ equals to 1 and the results are shown in table III.

From table III, it is easy to find that the SSAE $[128, 32, 32]$ with $\rho = 0.04$ achieves the best performance and therefore is selected as the encoded data generator.

In the second stage, we use the encoded data as the input of the time-related model based on variants of RNN. Four different kinds of variants include LSTM, BiLSTM, GRU and BiGRU. We use the $tanh$ activation function and the $Adam$ optimizer. Based on experience, we test on a few structures and finally decide to adopt a two-layer cell structure $[24, 12]$.

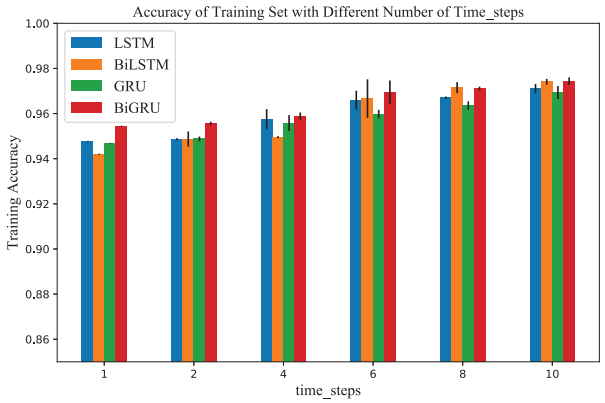
TABLE III
THE ACCURACY AND FAR OF STACKED SPARSE AUTOENCODER WITH DIFFERENT PARAMETERS

SSAE	ρ	Acc(train)	Acc(test)	FAR(train)	FAR(test)
None	None	0.9403	0.8709	0.0670	0.1385
[100, 50]	0.02	0.9486	0.8874	0.0604	0.1177
	0.04	0.9467	0.8889	0.0611	0.1171
	0.06	0.9457	0.8760	0.0621	0.1211
[81, 36]	0.02	0.9473	0.8793	0.0603	0.1213
	0.04	0.9478	0.8948	0.0614	0.1103
	0.06	0.9493	0.8713	0.0608	0.1335
[128,64,32]	0.02	0.9485	0.8934	0.0604	0.1120
	0.04	0.9499	0.8892	0.0607	0.1176
	0.06	0.9389	0.8895	0.0631	0.1180
[128,32,32]	0.02	0.9462	0.8871	0.0596	0.1229
	0.04	**0.9557**	**0.9004**	**0.0587**	**0.1061**
	0.06	0.9420	0.8931	0.0613	0.1166

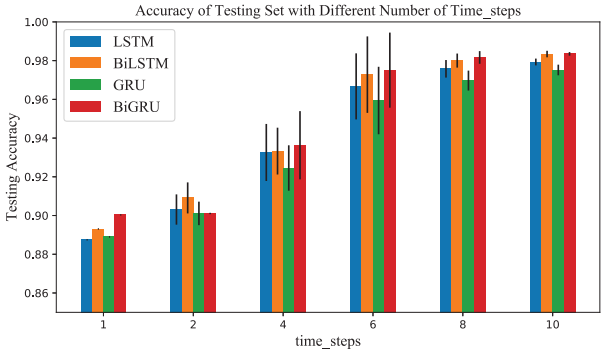
We also add a fully-connected layer with 6 neurons. The output layer has only one neuron. To verify the effectiveness of the model for time series data, experiments are conducted on different $time\_step$. All samples in the dataset are used and the results are shown in Fig.4.

In Fig.4(a) and Fig.4(b), it is obvious that with the increase of $time\_step$, the accuracy improves step by step both in training and testing dataset. For the training dataset, the accuracy improvement is less significant with the increase of $time\_step$. This may be due to the fact that the separability of the data itself reaches the limit. For this reason, feature engineering should be applied to help improve performance. Impressively, the promotion of performance due to $time\_step$ growth is very obvious on the testing set. To better characterize the impact of $time\_step$, we introduce a gain ratio that is the value of accuracy improvement for every two steps. Starting from $time\_step = 2$, a vertical line serves as the indicator of the increase of accuracy: the longer the line, the greater the increase of accuracy. Generally, the development of the model has two periods: the fast-ascension period and the slow-convergence period which are evidential in our model. Although the indicator reaches its max value when $time\_step$ equals 6, the actual detection rate still has room to improve, therefore it is still considered in the fast-ascension period. When $time\_step = 10$, the accuracy improves to a relatively high level due to its smallest gain ratio. We also do the experiment with $time\_step = 12$. However, we find the performance improvement is very insignificant but the cost of time and computation is very expensive. 8 can serve as an optimized candidate value for the parameter of $time\_step$. Fig.4(c) shows the comparison of FAR using different $time\_step$ values, where FAR is retained at a low level when $time\_step = 8$, again indicating the optimal value of $time\_step$ being 8.

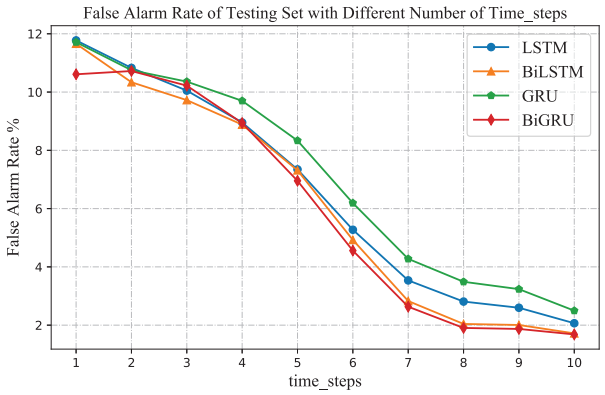
BiGRU's performance stands out during the experiments when $time\_step$ is greater than 4, where it performs better than other cells both in $Accuracy$ and FAR. Based on this, in the next step, we adopt the BiGRU-based model to finish the testing stage as shown in Fig.5.



(a) Comparison of Accuracy using different Cells with different time_steps on Training Dataset



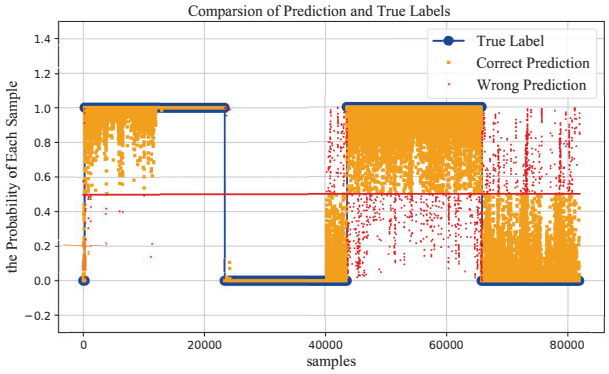
(b) Comparison of Accuracy using different Cells with different time_steps on Testing Dataset.



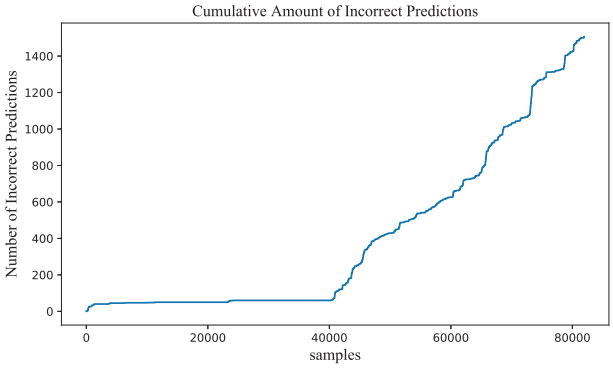
(c) Comparison of FAR using different Cells with different time_steps on Testing Dataset.

Fig. 4. The Experiment Results on Training and Testing Dataset.

In Fig.5(a), the blue bold dots represent the real label of testing samples. The orange middle dots mean the correct predictions and the red small dots are incorrectly classified samples. In Fig.5(b), the curve represents the accumulation number of incorrect samples as time goes on. In the beginning, despite the fluctuation found in the classified data (orange dots), satisfying performance can still be achieved. However, as time moves along with more samples, the performance of the model has declined, roughly starting at the 40,000th sample. There may be two possible reasons: it may be due to



(a) Comparison of Predictions and True Labels using BiGRU with 8 time_steps



(b) Cumulative Amount of Wrong Predictions During the Testing Phase.

Fig. 5. The Detection Process on Testing Dataset using BiGRU with 8 time_steps.

the emergence of new types of attacks as time goes on. For some features, there exist certain values in the testing dataset that are not available in the training dataset. Another reason may be the fluctuation of $Normal$ data, especially in features like $proto$ and $state$. Online learning could be a good solution to these problems. In our future research, we may consider adding online operation to this model.

TABLE IV
COMPARISON BETWEEN OUR PROPOSED MODEL WITH OTHER MACHINE LEARNING ALGORITHMS

Method	Accuracy	Precision	Recall
Decision Tree(DT) [19]	86.94%	91.28%	82.03%
Support Vector Machine(SVM) [20]	88.89%	92.25%	84.79%
Gradient Boosting Tree(GBT) [21]	93.13%	92.38%	92.84%
Bidirectional LSTM [10]	95.71%	**100%**	96.0%
Deep feed-forward neural network [22]	92.5%	98.2%	**99%**
Our BiGRU Model	**98.17%**	98.24%	98.43%

It is relatively meaningful to compare the performance of the proposed approach with other existing methods. We have tried to cover some of the most recently presented models based on UNSW-NB15. The comparison result shown in table IV proves the advancement of our approach.

V. Conclusion

In this paper, we propose a time series analysis model for network intrusion detection. A stacked sparse autoencoder is used to extract features that can represent the raw data in a low-dimension way. The greedy layer-wise strategy is adopted to find a group of optimal hyperparameters and the output of SSAE is used as the input of the time-related model in the next step. Four different variants of RNN are used and time-step is regarded as an important factor for the study. The experiment results show that the BiGRU with 8 time-steps has the optimal overall performance considering the actual accuracy and the accuracy gaining ratio of increasing time-steps. The accuracy reaches over 98% while the FAR is retained as low as 1.8%. In our future research, we will focus on improving the performance of the training dataset by introducing attention mechanism. In addition, we will use the online learning strategy to improve the robustness of the model.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (61771154) and the Fundamental Research Funds for the Central Universities (HEUCFG201830, GK2080260148). This paper is also funded by the International Exchange Program of Harbin Engineering University for Innovation-oriented Talents Cultivation.

Meantime, all the authors declare that there is no conflict of interests regarding the publication of this article. We greatly appreciate the helpful input and suggestions from the reviewers.

References

[1] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: Methods, systems and tools," *IEEE Commun. Surv. Tuts.*, vol. 16, no. 1, pp. 303–336, 2014.

[2] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2014.

[3] D. Kwon, H. Kim, J. Kim, C. S. Sang, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Clust. Comput.*, no. 5, pp. 1–13, 2017.

[4] J. Kim, J. Kim, H. L. T. Thu, and H. Kim, "Long short term memory recurrent neural network classifier for intrusion detection," in *Proc. Int. Conf. Platform Technol. Service*, pp. 1–5, 2016.

[5] C. L. Yin, Y. F. Zhu, J. L. Fei, and X. Z. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, no. 99, pp. 21954–21961, 2017.

[6] C. Xu, J. Shen, X. Du, and F. Zhang, "An intrusion detection system using a deep neural network with gated recurrent units," *IEEE Access*, vol. 6, pp. 48697–48707, 2018.

[7] W. Anani and J. Samarabandu, "Comparison of recurrent neural network algorithms for intrusion detection based on predicting packet sequences," in *2018 IEEE Canadian Conf. on Electrical & Computer Engineering (CCECE)*, pp. 1–4, IEEE, 2018.

[8] A. F. M. Agarap, "A neural network architecture combining gated recurrent unit (gru) and support vector machine (svm) for intrusion detection in network traffic data," in *Proc. of the 2018 10th Int. Conf. on Machine Learning and Computing*, pp. 26–30, ACM, 2018.

[9] A. H. Mirza and S. Cosan, "Computer network intrusion detection using sequential lstm neural networks autoencoders," in *Proc. IEEE Sign. Process. Commun. Appl. Conf. (SIU)*, pp. 1–4, IEEE, 2018.

[10] B. Roy and H. Cheung, "A deep learning approach for intrusion detection in internet of things using bi-directional long short-term memory recurrent neural network," in *2018 28th Int. Telecommun. Netw. and Appl. Conf. (ITNAC)*, pp. 1–6, IEEE, 2018.

[11] B. Yan and G. Han, "Effective feature extraction via stacked sparse autoencoder to improve intrusion detection system," *IEEE Access*, vol. 6, pp. 41238–41248, 2018.

[12] A. Ng *et al.*, "Sparse autoencoder," *CS294A Lecture notes*, vol. 72, no. 2011, pp. 1–19, 2011.

[13] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] J. Chen, H. Jing, Y. Chang, and Q. Liu, "Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process," *Reliab. Eng. Syst. Saf.*, vol. 185, pp. 372–382, 2019.

[16] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep recurrent neural network for intrusion detection in sdn-based networks," in *Proc. IEEE NetSoft*, pp. 202–206, IEEE, 2018.

[17] N. Moustafa and J. Slay, "The evaluation of network anomaly detection systems: Statistical analysis of the unsw-nb15 data set and the comparison with the kdd99 data set," *Information Security Journal: A Global Perspective*, vol. 25, no. 1-3, pp. 18–31, 2016.

[18] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Advances Neural Inf. Process. Syst.*, pp. 153–160, 2007.

[19] H. M. Anwer, M. Farouk, and A. Abdel-Hamid, "A framework for efficient network anomaly intrusion detection with features selection," in *2018 9th International Conference on Information and Communication Systems (ICICS)*, pp. 157–162, IEEE, 2018.

[20] Q. Tian, J. Li, and H. Liu, "A method for guaranteeing wireless communication based on a combination of deep and shallow learning," *IEEE Access*, vol. 7, pp. 38688–38695, 2019.

[21] Y. Zhou, M. Han, L. Liu, J. S. He, and Y. Wang, "Deep learning approach for cyberattack detection," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 262–267, IEEE, 2018.

[22] A.-H. Muna, N. Moustafa, and E. Sitnikova, "Identification of malicious activities in industrial internet of things based on deep learning models," *Journal of Information Security and Applications*, vol. 41, pp. 1–11, 2018.