Machine Learning Techniques for Classifying Network Anomalies and Intrusions

Zhida Li, Ana Laura Gonzalez Rios, Guangyu Xu, and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
Email: {zhidal, anag, joe_xu, ljilja}@sfu.ca

***Abstract*—Using machine learning techniques to detect network intrusions is an important topic in cybersecurity. A variety of machine learning models have been designed to help detect malicious intentions of network users. We employ two deep learning recurrent neural networks with a variable number of hidden layers: Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU). We also evaluate the recently proposed Broad Learning System (BLS) and its extensions. The models are trained and tested using Border Gateway Protocol (BGP) datasets that contain routing records collected from Réseaux IP Européens (RIPE) and BCNET as well as the NLS-KDD dataset containing network connection records. The algorithms are compared based on accuracy and F-Score.**

***Keywords*—Machine learning, recurrent neural networks, deep neural networks, broad learning system, intrusion detection**

I. Introduction

Machine learning algorithms have been used to successfully classify network anomalies and intrusions. Employed supervised learning algorithms include Support Vector Machine (SVM) [13], Long Short-Term Memory (LSTM) [8], [22], [25], [34], and Gated Recurrent Unit (GRU) [18], [42] as well as recently proposed Broad Learning System (BLS) [16], [17], [33]. Various intrusion detection systems (IDSs) [12], [15], [30] have been designed using machine learning [9] and deep neural networks such as stacked non-symmetric deep auto-encoder (NDAE) [37] and recurrent neural networks (RNNs) [43]. A number of algorithms (J48, naïve Bayes, naïve Bayes Tree, Random Forests, Random Tree, Multi-Layer Perception, and SVM) have been evaluated and compared [14], [38], [40], [44]. Network anomalies and intrusions [19], [45] have been classified using datasets such as Border Gateway Protocol (BGP) [1], [4] and NSL-KDD [3].

BLS and its extensions are alternatives to deep learning networks. They achieve comparable classification accuracy and require considerably shorter time for training than the conventional deep learning networks [17], [29] because of a small number of hidden layers. They also use pseudoinverse matrices rather than back-propagation during the training process.

The BGP and NSL-KDD datasets are benchmarks used for evaluating anomaly detection [10], [11], [20], [31] and intrusion [29] techniques. BGP datasets are routing records collected from Réseaux IP Européens (RIPE) that contain three types of anomaly attacks: Code Red I, Nimda, and Slammer [4] while BCNET [1] dataset contains regular traffic. The NSL-KDD dataset contains four types of intrusion attacks: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probe [3]. The NSL-KDD dataset is an improvement of the KDD'99 dataset [2], [35], [36] that was used in various IDSs [23], [26], [28], [32].

In this paper, we employ LSTM and GRU recurrent neural networks (RNNs) as well as BLS and its extensions. The paper is organized as follows. In Section II, we describe BGP and NSL-KDD datasets and their processing prior to classification. The LSTM and GRU algorithms with multi-layers are discussed in Section III. We introduce the BLS models in Section IV. In Section V, experimental procedure and training parameters are described. Classification models are compared based on accuracy and F-Score in Section VI. We conclude with Section VII.

II. Data Processing

We employ data collected from BGP and NSL-KDD datasets and label data as regular (0) and anomalous (1).

A. BGP Datasets

BGP datasets contain anomalous data points (the days of the attack) and regular data points (two days prior and two days after the attack). We extract 37 numerical features from BGP update messages [4]. In this paper, for each collected dataset (Code Red I, Nimda, and Slammer), we experiment with various number of training and test data points that are selected from periods of anomalies. Using 60 % of data for training and 40 % for testing generates the best performance results. The duration of anomalies and the number of data points in the BGP datasets are shown in Table I.

B. NSL-KDD Dataset

The KDDTrain^{+} and KDDTest^{+} are NSL-KDD training and test datasets, respectively. KDDTest^{-21} is a subset of the KDDTest^{+} dataset that does not include records correctly classified by 21 models [39]. The number of data points is shown in Table II [3]. Among 41 features, 38 are numerical and 3 are categorical ("protocol_type", "service", and "flag") features. The 3 categorical features have been converted to numerical features using the dummy coding method to generate 71 additional features. We use "pandas" Python library [5] to

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada under grant R31-611284.

978-1-7281-0397-6/19/$31.00 ©2019 IEEE

TABLE I
DURATION OF ANALYZED BGP EVENTS AND NUMBER OF DATA POINTS IN BGP DATASETS

	Regular (min)	**Anomaly** (min)	**Regular** (training dataset)	**Anomaly** (training dataset)	**Regular** (test dataset)	**Anomaly** (test dataset)
Code Red I	6,599	601	3678	362	2921	239
Nimda	3,678	3,522	3677	2123	1	1399
Slammer	6,330	870	3209	531	3121	339

create input matrices. Dimensions of KDDTrain^{+}, KDDTest^{+}, and KDDTest^{-21} matrices are 109×125,973, 109×22,544, and 109×11,850, respectively.

TABLE II
NUMBER OF DATA POINTS IN THE NSL-KDD DATASET

	Regular	**DoS**	**U2R**	**R2L**	**Probe**	**Total**
KDDTrain^{+}	67,343	45,927	52	995	11,656	125,973
KDDTest^{+}	9,711	7,458	200	2,754	2,421	22,544
KDDTest^{-21}	2,152	4,342	200	2,754	2,402	11,850

III. DEEP LEARNING: MULTI-LAYER NETWORKS

Deep neural networks enable learning to select important features from the input data. Their significant advantage is the back-propagation method that calculates gradients and updates the weights [21], [41]. Furthermore, they may achieve desired results by adjusting the number of hidden nodes, hidden layers, and optimization algorithms. They use nonlinear functions such as rectified linear unit (ReLU), logistic sigmoid, or tanh. The numbers of hidden nodes and layers are chosen depending on the size of the dataset. Note that adding hidden layers may not achieve higher accuracy because of "over-fitting".

We evaluate performance of two deep learning RNN models (LSTM and GRU) with 2 (LSTM$_2$ and GRU$_2$), 3 (LSTM$_3$ and GRU$_3$), and 4 (LSTM$_4$ and GRU$_4$) hidden layers. A model with 4 hidden layers is shown in Fig. 1.

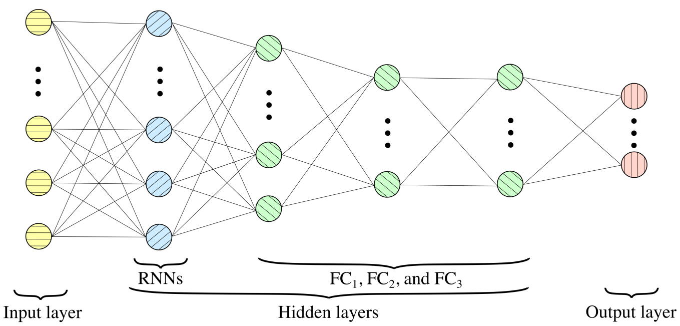

Fig. 1. Deep learning neural network model. It consists of 37 (BGP)/109 (NSL-KDD) RNNs, 80 FC$_1$, 32 FC$_2$, and 16 FC$_3$ fully connected (FC) hidden nodes.

IV. BROAD LEARNING SYSTEM

BLS [7], [17], [29] shows comparable performance and shorter training time than RNN. We evaluate performance of the original BLS, BLS with incremental learning, BLS with radial basis function (RBF-BLS), and BLS with cascades of mapped features (CFBLS), enhancement nodes (CEBLS), and both mapped features and enhancement nodes (CFEBLS).

A. Original BLS

The original BLS algorithm [17] constructs a set of enhancement nodes from the mapped inputs $\boldsymbol{X}$ as shown in Fig. 2. In the initialization step, a matrix $\boldsymbol{A}_n^m$ contains n groups of mapped features and m groups of enhancement nodes. Matrix $\boldsymbol{Z}^n$ is the collection of groups of mapped features $\boldsymbol{Z}^n \triangleq [\boldsymbol{Z}_1,\boldsymbol{Z}_2, ..., \boldsymbol{Z}_n]$ while matrix $\boldsymbol{H}^m$ contains groups of enhancement nodes $\boldsymbol{H}^m \triangleq [\boldsymbol{H}_1, \boldsymbol{H}_2, ..., \boldsymbol{H}_m]$.

Matrix $\boldsymbol{A_x}$ is constructed from $\boldsymbol{Z}_n$ and $\boldsymbol{H}_m$ as:

$$\boldsymbol{A}_x = [\ \phi(\boldsymbol{X}\boldsymbol{W}_{ei} + \boldsymbol{\beta}_{ei}) \mid \xi(\boldsymbol{Z}_x^n \boldsymbol{W}_{hj} + \boldsymbol{\beta}_{hj})\ ],$$
$$i = 1, 2, ..., n \text{ and } j = 1, 2, ..., m, \tag{1}$$

where ϕ and ξ are projection mappings while $\boldsymbol{W}_{ei}$, $\boldsymbol{W}_{hj}$ and $\boldsymbol{\beta}_{ei}$, $\boldsymbol{\beta}_{hj}$ are weights and bias parameters, respectively.

The BLS performance improves if the mapped features $\boldsymbol{Z}^n$, enhancement nodes $\boldsymbol{H}^m$, or input data points $\boldsymbol{X}$ are dynamically incremented. In BLS with incremental learning, matrix $\boldsymbol{A}_n^m$ may be updated by increasing the number of mapped features and/or enhancement nodes until a predefined training threshold is reached. Furthermore, an already trained BLS may easily accept new input data (nodes) $\boldsymbol{X}_a$ without a need for a new training cycle. Hence, this incremental learning reduces the training time when computing pseudoinverse matrices [17]. Equation (1) is modified to include additional mapped features $\boldsymbol{Z_{n+1}}$, enhancement nodes $\boldsymbol{H_{m+1}}$, and/or input nodes $\boldsymbol{X}_a$ as shown in Fig. 2. We employ an algorithm [17] that dynamically updates the mapped features and enhancement groups of nodes as well as input nodes.

B. RBF-BLS Extension

The BLS and incremental learning algorithms with enhancement nodes have been extended to include the radial basis function (RBF) [33] and create the RBF-BLS model. The RBF network includes one input, one hidden, and one output layer. The RBF function was implemented using Gaussian kernel:

$$\xi(x) \ = \ \exp\left(-\frac{\|\ x\ -\ c\ \|^2}{\gamma^2}\right). \tag{2}$$

If the input data are located in a narrow region around the central point c, the RBF generates significant non-zero responses. An RBF network with k hidden nodes relies on calculating pseudoinverse to perform rapid training. The weight vectors of the output $\boldsymbol{HW}$ are deduced from:

$$\begin{aligned}\boldsymbol{W} &= (\boldsymbol{H}^T\boldsymbol{H})^{-1}\boldsymbol{H}^T\boldsymbol{Y}\\ &= \boldsymbol{H}^{+}\boldsymbol{Y},\end{aligned} \tag{3}$$

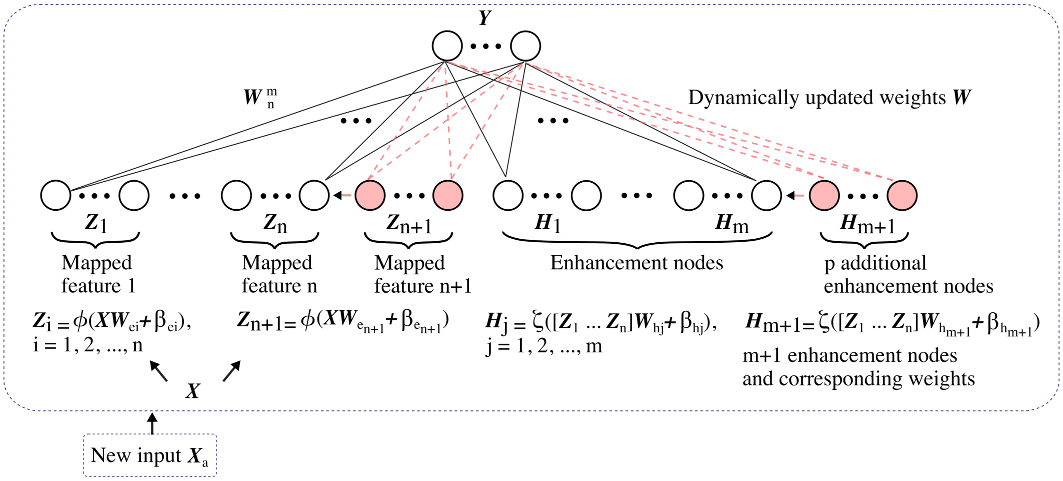
Y
W_n^m
Dynamically updated weights W
Z_1
Z_n
Z_{n+1}
H_1
H_m
H_{m+1}
Mapped feature 1
Mapped feature n
Mapped feature n+1
Enhancement nodes
p additional enhancement nodes
$Z_i = \phi(XW_{ei} + \beta_{ei})$, i = 1, 2, ..., n
$Z_{n+1} = \phi(XW_{e_{n+1}} + \beta_{e_{n+1}})$
$H_j = \zeta([Z_1 \dots Z_n]W_{hj} + \beta_{hj})$, j = 1, 2, ..., m
$H_{m+1} = \zeta([Z_1 \dots Z_n]W_{h_{m+1}} + \beta_{h_{m+1}})$
m+1 enhancement nodes and corresponding weights
X
New input X_a

Fig. 2. Module of the BLS algorithm with increment of mapped feature, enhancement nodes, and new input data [17].

where $\boldsymbol{W} = [w_1, w_2, ..., w_k]$ and $\boldsymbol{H} = [\xi_1, \xi_2, ..., \xi_k]$ are matrices of output weights and hidden nodes, respectively while $\boldsymbol{H}^+$ is the pseudoinverse of $\boldsymbol{H}$.

C. Cascades of Mapped Features and Enhancement Nodes

Several variants of the connections within and between the BLS mapped features and enhancement nodes were introduced [16]. These connections form a multi-layer architecture with cascades of mapped features, enhancement nodes, and both mapped features and enhancement nodes. In case of CFBLS, the new group $\boldsymbol{Z}_k$ of mapped features is created by using the previous group $(k-1)$. Groups of mapped features are formulated as:

$$\begin{aligned}\boldsymbol{Z}_k &= \phi(\boldsymbol{Z}_{k-1}\boldsymbol{W}_{ek} + \boldsymbol{\beta}_{ek}) \\ &\triangleq \phi^k(\boldsymbol{X}\ ;\ \{\boldsymbol{W}_{ei}, \boldsymbol{\beta}_{ei}\ \}_{i=1}^{k}), \text{for } k = 1, ..., n.\end{aligned} \quad (4)$$

Cascades of these groups $\boldsymbol{Z}^n \triangleq [\boldsymbol{Z}_1, ..., \boldsymbol{Z}_n]$ are used to generate the enhancement nodes $\{\boldsymbol{H}\}_{j=1}^{m}$. The first enhancement node in CEBLS is generated from mapped features. The subsequent enhancement nodes are generated from previous enhancement nodes creating a cascade:

$$\boldsymbol{H}_u \triangleq \xi^u(\boldsymbol{Z}^n\ ;\ \{\boldsymbol{W}_{hi}, \boldsymbol{\beta}_{hi}\}_{i=1}^{u}),\ for\ u = 1, ..., m, \quad (5)$$

where $\boldsymbol{W}_{hi}$ and $\boldsymbol{\beta}_{hi}$ are randomly generated. The CFEBLS architecture is a combination of the two cascading approaches.

V. Experimental Procedure

Classification performance depends on BLS parameters (number of mapped features, mapped groups, and enhancement nodes) as well as RNN parameters (number of hidden nodes, hidden layers, and optimization algorithms). The experimental procedure consists of four steps:

- *Step 1*: Normalize training and test datasets.
- *Step 2*: Train the RNN models and BLS using 10-fold validation. Tune parameters of the RNN and BLS models.
- *Step 3*: Test the RNN and BLS models.
- *Step 4*: Evaluate models based on accuracy and F-Score.

We implement deep learning models using a Dell Alienware Aurora with 32 GB memory, NVIDIA GeForce GTX 1080 GPU, and Intel Core i7 7700K CPU processor. We use the PyTorch [6] neural network library for building deep learning models. For LSTM, GRU, and BLS and its extensions, we generate results using Python 2.7.12. MATLAB implementation [7] of BLS and its extensions is converted to Python code in order to have consistent comparison. The deep learning models are trained using 50 epochs with learning rate $lr = 0.001$. We use 10 and 50 data points as input sequences for BGP and NSL-KDD datasets, respectively. In the training process, the model is optimized by using Adam algorithm [27] while "over-fitting" is avoided by employing dropout rate [24] of 50 %. The best results for LSTM$_2$ and GRU$_2$ are obtained with 16 fully connected hidden nodes (FC$_3$). LSTM$_3$ and GRU$_3$ offer the best performance with 32 FC$_2$ and 16 FC$_3$ fully connected hidden nodes. The sigmoid function is replaced by the RBF function for enhancement nodes in the RBF-BLS model. We implement the CFBLS, CEBLS, and CFEBLS models by modifying the original BLS functions. The BLS training parameters that generate the best performance results are listed in Table III.

TABLE III
Number of BLS Training Parameters

Parameters	**Code Red I**	**Nimda**	**Slammer**	**NSL-KDD**
Mapped features	100	500	100	100
Groups of mapped features	1	1	25	5
Enhancement nodes	500	700	300	100
Incremental learning steps	10	9	2	3
Data points/step	100	200	100	3000
Enhancement nodes/step	10	10	50	60

VI. Performance Evaluation

We evaluate the performance of the deep learning and broad learning classification models based on accuracy and F-Score. The training times for RNN and BLS models using the BGP and NSL-KDD datasets are shown in Table IV. The GRU

TABLE IV
TRAINING TIME FOR RNN AND BLS MODELS: BGP AND NSL-KDD DATASETS

Model	**Datasets**	**LSTM$_2$**	**LSTM$_3$**	**LSTM$_4$**	**GRU$_2$**	**GRU$_3$**	**GRU$_4$**	**BLS**	**RBF-BLS**	**CFBLS**	**CEBLS**	**CFEBLS**
		Python (CPU)										
Time (s)	BGP (Slammer)	224.52	259.91	819.78	54.12	60.76	759.82	21.53	18.68	18.89	32.36	32.13
	NSL-KDD	4481.73	4614.66	11478.62	1108.31	1161.80	11581.30	99.47	98.27	98.13	108.23	108.14
		Python (GPU)						MATLAB (CPU)				
Time (s)	BGP (Slammer)	30.74	34.94	38.82	31.03	35.46	40.22	1.36	1.20	1.03	5.49	5.98
	NSL-KDD	344.93	355.86	394.55	317.53	345.04	369.86	6.91	6.24	6.55	8.88	8.95

algorithm has shorter training time than the LSTM algorithm due to its simpler structure. Note that MATLAB employs the optimized function *mapminmax()*, which results in shorter training time for BLS and its extensions.

Performance of RNN and BLS models using the BGP datasets is shown in Table V and Table VI, respectively. RNN and BLS models offer variable performance with the best accuracy and F-Score in the range of 90 %–95 %.

TABLE V
PERFORMANCE OF RNN (LSTM AND GRU) MODELS: BGP DATASETS (PYTHON)

		Accuracy (%)			**F-Score** (%)
Model	**Training Dataset**	**Test**	**RIPE**	**BCNET**	**Test**
			(regular)	(regular)	
LSTM$_2$	Code Red I	94.08	83.75	60.49	68.89
	Nimda	78.36	47.15	48.61	87.87
	Slammer	92.98	92.99	85.97	72.42
LSTM$_3$	Code Red I	88.54	79.38	58.82	55.96
	Nimda	85.57	39.10	40.28	92.22
	Slammer	90.90	92.01	84.38	67.29
LSTM$_4$	Code Red I	86.96	75.00	57.01	51.53
	Nimda	92.00	26.94	35.21	95.83
	Slammer	92.49	92.22	86.18	70.72
GRU$_2$	Code Red I	87.47	80.07	60.21	52.97
	Nimda	70.71	48.96	58.26	82.83
	Slammer	91.88	93.33	90.90	69.42
GRU$_3$	Code Red I	88.07	79.44	60.56	53.51
	Nimda	80.21	38.40	44.24	89.02
	Slammer	91.76	95.21	90.83	68.72
GRU$_4$	Code Red I	91.84	77.50	60.07	63.87
	Nimda	87.36	35.00	39.38	93.25
	Slammer	92.14	92.15	90.35	70.11

TABLE VI
PERFORMANCE OF BLS MODEL AND ITS EXTENSIONS: BGP DATASETS (PYTHON)

		Accuracy (%)			**F-Score** (%)
Model	**Training Dataset**	**Test**	**RIPE**	**BCNET**	**Test**
			(regular)	(regular)	
BLS	Code Red I	94.97	69.79	65.21	66.38
	Nimda	76.57	70.69	54.93	86.73
	Slammer	87.65	75.62	68.40	57.68
RBF-BLS	Code Red I	95.92	90.69	73.96	70.07
	Nimda	57.92	70.63	57.22	73.36
	Slammer	91.21	90.55	70.76	64.57
CFBLS	Code Red I	95.16	69.38	61.74	71.08
	Nimda	55.71	68.06	58.26	71.56
	Slammer	89.28	71.25	61.81	60.99
CEBLS	Code Red I	94.94	70.69	60.35	65.22
	Nimda	66.43	74.10	54.51	79.83
	Slammer	91.01	87.71	82.43	66.38
CFEBLS	Code Red I	95.66	70.07	59.51	71.75
	Nimda	64.29	70.83	57.43	78.24
	Slammer	86.36	71.11	57.71	55.30

The RNNs and BLS performance using NSL-KDD dataset is shown in Table VII. The best performance is achieved using the LSTM$_4$ and GRU$_3$ RNNs while the CFBLS architecture offers the best results among BLS models.

TABLE VII
PERFORMANCE OF RNN (LSTM AND GRU) AND BLS MODELS: NSL-KDD DATASETS (PYTHON)

	Accuracy (%)		**F-Score** (%)	
Model	KDDTest^{+}	KDDTest^{-21}	KDDTest^{+}	KDDTest^{-21}
LSTM$_4$	82.78	66.74	83.34	76.21
GRU$_3$	82.87	65.42	83.05	74.06
CFBLS	82.20	67.47	82.23	76.29

Performance and training times of incremental BLS are shown in Table VIII. The incremental BLS algorithm requires shorter training time because the weights are updated based on only new data.

TABLE VIII
PERFORMANCE AND TRAINING TIME OF INCREMENTAL BLS MODEL: BGP AND NSL-KDD DATASETS (MATLAB)

Test	**Accuracy** (%)	**F-Score** (%)	**Time** (s)
Code Red I	94.37	65.10	0.926
Nimda	91.64	95.64	2.757
Slammer	89.31	63.07	2.805
KDDTest^{+}	81.34	81.99	32.99
KDDTest^{-21}	78.70	88.06	29.71

VII. CONCLUSION

We employed LSTM and GRU deep neural networks with a variable number of hidden layers. We also evaluated performance of BLS models that employ radial basis function (RBF), cascades of mapped features and enhancement nodes, and incremental learning. Compared to deep neural networks, BLS and cascade combinations of mapped features and enhancement nodes achieved comparable performance and shorter training time because of their wide and deep structure. BLS models consist of a small number of hidden layers and adjust weights using pseudoinverse instead of back-propagation. They also dynamically update weights in case of incremental learning. They better optimized weights due to additional data points for large datasets such as NSL-KDD. While increasing the number of mapped features and enhancement nodes as well as mapped groups led to better performance, it required additional memory and training time.

References

[1] BCNET [Online]. Available: http://www.bc.net/. Accessed: Feb. 21, 2019.

[2] KDD Cup 1999 Data [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html/. Accessed: Feb. 21, 2019.

[3] NSL-KDD Data Set [Online]. Available: https://www.unb.ca/cic/datasets/nsl.html/. Accessed: Feb. 21, 2019.

[4] RIPE NCC [Online]. Available: https://www.ripe.net/analyse/. Accessed: Feb. 21, 2019.

[5] Pandas [Online]. Available: https://pandas.pydata.org/. Accessed: Feb. 21, 2019.

[6] PyTorch [Online]. Available: https://pytorch.org/docs/stable/nn.html. Accessed: Feb. 21, 2019.

[7] Broadlearning [Online]. Available: http://www.broadlearning.ai/. Accessed: Feb. 21, 2019.

[8] Understanding LSTM Networks [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: Feb. 21, 2019.

[9] T. Ahmed, B. Oreshkin, and M. Coates, "Machine learning approaches to network anomaly detection," in *Proc. USENIX Workshop Tackling Comput. Syst. Problems with Mach. Learn. Techn.*, Cambridge, MA, USA, Apr. 2007, pp. 1–6.

[10] N. Al-Rousan, S. Haeri, and Lj. Trajković, "Feature selection for classification of BGP anomalies using Bayesian models," in *Proc. Int. Conf. Mach. Learn. Cybern., ICMLC 2012*, Xi'an, China, July 2012, pp. 140–147.

[11] N. Al-Rousan and Lj. Trajković, "Machine learning models for classification of BGP anomalies," in *Proc. IEEE Conf. High Perform. Switching Routing, HPSR 2012*, Belgrade, Serbia, June 2012, pp. 103–108.

[12] M. Bhuyan, D. Bhattacharyya, and J. Kalita, "Network anomaly detection: methods, systems and tools," *IEEE Commun. Surv. Tut.*, vol. 16, no. 1, pp. 303–336, Mar. 2014.

[13] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag, 2006.

[14] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Commun. Surv. Tut.,*, vol. 18, no. 2, pp. 1153–1176, 2016.

[15] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: a survey," *ACM Comput. Surv.*, vol. 41, no. 3, pp. 15:1–15:58, July 2009.

[16] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–14, Sept. 2018.

[17] C. L. P. Chen and Z. Liu, "Broad learning system: an effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[18] K. Cho, B. van Merriënboer, C. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translations," in *Proc. 2014 Conf. Empirical Methods Natural Lang. Process.*, Doha, Qatar, Oct. 2014, pp. 1724–1734.

[19] Q. Ding, Z. Li, S. Haeri, and Lj. Trajković, "Application of machine learning techniques to detecting anomalies in communication networks," pp. 47–70 and pp. 71–92, in *Cyber Threat Intelligence*, M. Conti, A. Dehghantanha, and T. Dargahi, Eds., Berlin: Springer, 2018.

[20] Q. Ding, Z. Li, P. Batta, and Lj. Trajković, "Detecting BGP anomalies using machine learning techniques," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Budapest, Hungary, Oct. 2016, pp. 3352–3355.

[21] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," *Neural Netw.*, vol. 18, no. 5–6, pp. 602–610, July/Aug. 2005.

[22] K. Greff, R. K. Srivastava, J. Koutnik, B. R. Steunebrink, and J. Schmidhuber, "LSTM: a search space odyssey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 10, pp. 2222–2232, Oct. 2017.

[23] W. Heyi, H. Aiqun, S. Yubo, B. Ning, and J. Xuefei, "A new intrusion detection feature extraction method based on complex network theory," in *Proc. 4th Int. Conf. Multimedia Inf. Netw. and Secur.*, Nanjing, China, Nov. 2012, pp. 852–856.

[24] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *Computing Research Repository (CoRR)*, abs/1207.0580, pp. 1–18, Jul. 2012.

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

[26] H. G. Kayack, A. N. Zincir-Heywood, and M. I. Heywood, "Selecting features for intrusion detection: a feature relevance analysis on KDD 99 intrusion detection datasets," in *Proc. 3rd Annu. Conf. Privacy Secur. Trust, PST 2005*, St. Andrews, NB, Canada, Oct. 2005, pp. 1–6.

[27] D. P. Kingma and J. Ba, "Adam: a method for stochastic optimization," in *Proc. 3rd Int. Conf. Learn. Representations*, San Diego, CA, USA, May 2015, pp. 1–15.

[28] W. Lee, S. J. Stolfo, and K. W. Mok, "Mining in a data-flow environment: experience in network intrusion detection," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery & Data Mining, KDD-99*, San Diego, CA, USA, Aug. 1999, pp. 114–124.

[29] Z. Li, P. Batta, and Lj. Trajković, "Comparison of machine learning algorithms for detection of network intrusions," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Miyazaki, Japan, Oct. 2018, pp. 4248–4253.

[30] L. Li, Y. Yu, S. Bai, Y. Hou, and X. Chen, "An effective two-step intrusion detection approach based on binary classification and k-NN," *IEEE Access*, vol. 6, pp. 12060–12073, Mar. 2018.

[31] Y. Li, H. J. Xing, Q. Hua, X.-Z. Wang, P. Batta, S. Haeri, and Lj. Trajković, "Classification of BGP anomalies using decision trees and fuzzy rough sets," in *Proc. IEEE Trans. Syst., Man, Cybern.*, San Diego, CA, USA, Oct. 2014, pp. 1331–1336.

[32] R. P. Lippmann, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. A. Zissman, "Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation," in *Proc. DARPA Inform. Survivability Conf. and Expo., DISCEX 2000*, Hilton Head, SC, USA, Jan. 2000, pp. 12–26.

[33] Z. Liu and C. L. P. Chen, "Broad Learning System: structural extensions on single-layer and multi-layer neural networks," in *Proc. 2017 Int. Conf. Secur., Pattern Anal., Cybern.*, Shenzhen, China, Dec. 2017, pp. 136–141.

[34] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proc. Euro. Symp. Artif. Neural Netw., Comput. Intell. Mach. Learn., ESANN 2015*, Bruges, Belgium, Apr. 2015, pp. 89–94.

[35] J. McHugh, "Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Trans. Inform. Syst. Secur.*, vol. 3, no. 4, pp. 262-294, Nov. 2000.

[36] A. A. Olusola, A. S. Oladele, and D. O. Abosede, "Analysis of KDD '99 intrusion detection dataset for selection of relevance features," in *Proc. World Congress Eng. Comput. Sci.*, San Francisco, CA, USA, Oct. 2010, pp. 162–168.

[37] N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi, "A deep learning approach to network intrusion detection," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 2, no. 1, pp. 41–50, Feb. 2018.

[38] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. Wireless Netw. Mobile Commun., WINCOM 2016*, Fez, Morocco, Oct. 2016, pp. 258–263.

[39] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *Proc. IEEE Symp. Comput. Intell. Secur. and Defense Appl., CISDA 2009*, Ottawa, ON, Canada, July 2009, pp. 1–6.

[40] D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Commun. Surv. Tut.,*, vol. 17, no. 1, pp. 70–91, 2015.

[41] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 3, pp. 229–256, May 1992.

[42] 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, Aug. 2018.

[43] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, Nov. 2017.

[44] J. Zhang and M. Zulkernine, "A hybrid network intrusion detection technique using random forests," in *Proc. First Int. Conf. Availability, Rel. Secur.*, Vienna, Austria, Apr. 2006, pp. 262–269.

[45] J. Zhang, J. Rexford, and J. Feigenbaum, "Learning-based anomaly detection in BGP updates," in *Proc. Workshop Mining Netw. Data*, Philadelphia, PA, USA, Aug. 2005, pp. 219–220.