Comparison of Machine Learning Algorithms for Detection of Network Intrusions

Zhida Li, Prerna Batta, and Ljiljana Trajković
Simon Fraser University
Vancouver, British Columbia, Canada
{zhidal, pbatta, ljilja}@sfu.ca

***Abstract*—Detecting, analyzing, and defending against network intrusions is an important topic in cyber security. Various detection systems have been designed using machine learning techniques that help detect malicious intentions of network users. We apply Recurrent Neural Networks (RNNs) and Broad Learning System (BLS) machine learning algorithms to classify known network intrusions. The developed models are trained and tested using the NSL-KDD dataset containing information about intrusion and regular network connections. The algorithms are used to classify various types of intrusion classes and regular data and are compared based on accuracy and F-Score. Performance results indicate that the BLS algorithm shows comparable performance with shorter training time.**

***Index Terms*—Machine learning, recurrent neural networks, deep neural networks, broad learning system**

I. Introduction

With the development of fast computing platforms, neural network-based algorithms have proved useful in detecting various anomalies such as network intrusions. Intrusion detection and protection mechanisms are used to detect network anomalies and attacks. Various detection systems have been designed using machine learning techniques that help detect malicious intentions of network users.

Intrusion datasets are important for identifying, testing, and validating effective intrusion detection methods. The quality of collected data affects effectiveness of anomaly detection techniques. Several datasets used for intrusion detection are publicly available.

The most widely used datasets are KDD Cup 1999 (KDD'99) [1] and NSL-KDD [2]. An early corpus of data to model network traffic collected at US Air Force bases was developed for the DARPA'98 Intrusion Detection System (IDS) evaluation program [3]–[5]. DARPA'98 consists of 7 weeks of collected network traffic containing nearly 4 GB of compressed raw *tcpdump* data that include 5 millions of connection records each having 100 bytes. The KDD'99 intrusion dataset, based on data captured for DARPA'98, is a benchmark widely used for evaluation of anomaly detection techniques. The KDD'99 dataset consists of 41 connection features [6]: 3 categorical and 38 numerical features. Data points are labeled as either intrusion or regular data. There are four types of intrusion attacks: Denial of Service (DoS), User to Root (U2R), Remote to Local (R2L), and Probe. The NSL-KDD dataset is a randomly selected subset of KDD'99 after redundant data were removed [7].

Various intrusion detection techniques have been proposed over the last decade. The information gain method and rough sets were employed for feature selections [6], [8]. A hybrid intrusion detection system was proposed to identify misuse and anomaly intrusions using random forests [9]. Network (NIDS) [10] and recurrent neural network (RNN-IDS) [11] intrusion detection systems were proposed and compared to various machine learning algorithms such as J48, naive Bayes (NB), NB Tree, Random Forests (RF), Random Tree (RT), Multilayer Perception (MP), and Support Vector Machine (SVM). Training time is often of concern when employing deep learning algorithms trained using only CPUs. Hence, a stacked non-symmetric deep auto-encoder (NDAE) that combines deep learning offered by NDAE and shallow learning offered by the RF classifier has been proposed and implemented using a graphics processing unit (GPU) [12]. A hybrid framework is proposed [13] to include binary classifier (BC) modules based on the C4.5 algorithm [14] and an aggregation module generating two output classes: certain and uncertain. The data points from both classes are imported to a k-NN module that determines the class of uncertain data points. The Broad Learning System (BLS) [15], an alternative to deep learning networks, offers desired classification accuracy with shorter training time when using the Modified National Institute of Standards and Technology (MNIST) [16] and the New York University Object Recognition Benchmark [17] datasets.

In this paper, we employ three variants of RNNs (Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Bidirectional LSTM (Bi-LSTM)) and the BLS algorithm.

The paper is organized as follows: In Section II, we describe the NSL-KDD dataset and its processing for training and test. The LSTM, GRU, and Bi-LSTM algorithms are described in Section III. In Section IV, experimental procedure, a deep learning framework, and BLS training parameters are introduced. The two-way and five-way classification results are compared based on accuracy and F-Score in Section V. We conclude with Section VI.

II. Data Processing

In this Section, we introduce four types of intrusion attacks and employed training and test datasets. We use unbalanced

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

datasets for model training where the number of intrusions is smaller than the number of regular data points. We consider four types of intrusion attacks described in Table I [10].

TABLE I
NSL-KDD Dataset: Types of Intrusion Attacks

Type	Intrusion attacks
DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, processtable, udpstorm, apache2, worm
U2R	buffer-overflow, loadmodule, perl, rootkit, sqlattack, xterm, ps
R2L	fpt-write, guess-passwd, imap, multihop, phf, spy, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
Probe	ipsweep, nmap, portsweep, satan, mscan, saint

In two-way classification, all intrusions are treated as one class. We consider anomaly and regular classes that correspond to 1 and 0, respectively. In five-way classification, the targets are regular, DoS, U2R, R2L, and Probe, labeled as $0, 1, 2, 3,$ and 4, respectively. We employ the *KDDTrain*$^{+}$ dataset for training and *KDDTest*$^{+}$ and *KDDTest*$^{-21}$ datasets for test. The *KDDTrain*$^{+}$ and *KDDTest*$^{+}$ are entire NSL-KDD training and test datasets, respectively. *KDDTest*$^{-21}$ is a subset of the KDD'99 dataset that does not include records correctly classified by 21 models (7 classifiers used 3 times) [7]. The number of data points in the NSL-KDD dataset is shown in Table II [2]. The four top-rank features [9] are described in Table III while their traces are shown in Fig. 1. The remaining 37 features are described in [6].

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

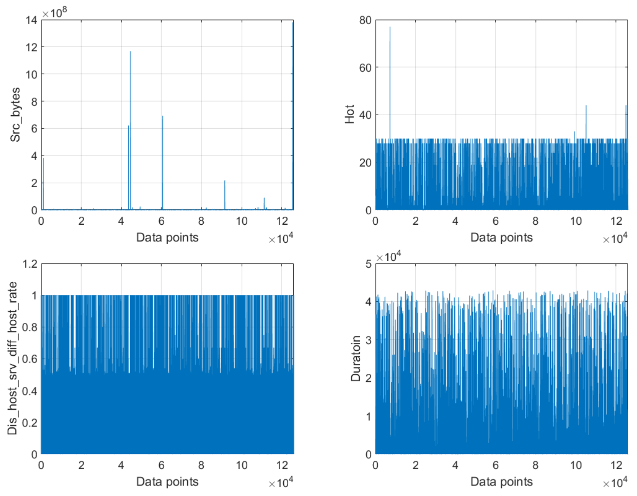


Fig. 1. *KDDTrain*$^{+}$ dataset: traces of "Src_bytes" (top left), "Hot" (top right), "Dis_host_srv_diff_host_rate" (bottom left), and "Duration" (bottom right) features.

Prior to training, each feature has been normalized with mean 0 and standard deviation 1. The categorical features 2, 3, and 4 are converted to numerical features by using dummy encoding. Feature selection algorithms were not used to extract the most relevant features and the evaluated classification algorithms employed the entire *KDDTrain*$^{+}$ dataset.

III. Classification Algorithms

Five classification algorithms were used for comparison.

A. Long Short-Term Memory Neural Network

An important advantage of recurrent neural networks (RNNs) is their ability to use contextual information between input and output sequences. However, the conventional RNNs may store only a limited range of contextual information. This affects the information propagation through the network and results in a diminishing influence of hidden layers. The LSTM neural network was proposed to address this deficiency [18] and to overcome long-term dependency and vanishing gradient problems [19]. Eight LSTM variants of the most commonly used LSTM architecture [20], [21] were evaluated [22] to show that none could significantly enhance the classification performance.

We select a single-hidden-layer LSTM module shown in Fig. 2 for the intrusion detection. An LSTM cell called the *memory block* [23] is composed of: (a) *forget gate* f_t, (b) *input gate* i_t, and (c) *output gate* o_t. The *forget gate* discards the irrelevant memories according to the cell state, the *input gate* controls the information that will be updated in the LSTM cell, and the *output gate* works as a filter that controls the output. The logistic sigmoid and network output functions are denoted by σ and $\tanh$, respectively. The output of the LSTM cell is connected to the LSTM output layer.

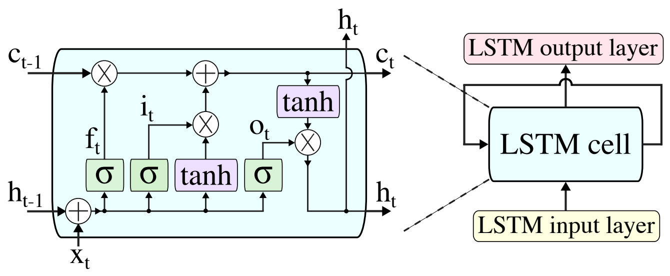


Fig. 2. Repeating module for the LSTM neural network. States c_{t-1} and c_t are the previous and current cell states, respectively.

The outputs of the *forget gate* f_t, the *input gate* i_t, and the *output gate* o_t at time *t* are [24]:

$$f_t = \sigma(W_{if}x_t + b_{if} + U_{hf}h_{t-1} + b_{hf}) \quad (1)$$
$$i_t = \sigma(W_{ii}x_t + b_{ii} + U_{hi}h_{t-1} + b_{hi}) \quad (2)$$
$$o_t = \sigma(W_{io}x_t + b_{io} + U_{ho}h_{t-1} + b_{ho}), \quad (3)$$

where $\sigma(\cdot)$ is the logistic sigmoid function, x_t is the current input vector, h_{t-1} is the previous output vector, W_{if}, U_{hf}, W_{ii}, U_{hi}, W_{io}, and U_{ho} are weight matrices, and b_{if}, b_{hf}, b_{ii}, b_{hi}, b_{io}, and b_{ho} are bias vectors. The output i_t of the *input gate* decides if the information will be stored in the cell state. The sigmoid function is used to update the information.

TABLE III
NSL-KDD FEATURES: LISTED ARE DEFINITIONS, TYPES, AND DESCRIPTIONS FOR FOUR TOP RANK FEATURES

Feature	Definition	Type	Description
1	duration	continuous	length (number of seconds) of the connection
5	src_bytes	continuous	number of data bytes from source to destination
10	hot	continuous	number of "hot" indicators
37	dst_host_srv_diff_host_rate	continuous	no. of connections to different hosts

The cell state c_t is calculated as:

$$c_t = f_t * c_{t-1} + i_t * tanh(W_{ic} x_t + b_{ic} + U_{hc} h_{t-1} + b_{hc}), \quad (4)$$

where $*$ denotes element-wise multiplications and the $tanh$ function is used to to create a vector for the next cell state.

The output of the LSTM cell is:

$$h_t = o_t * tanh(c_t). \quad (5)$$

B. Gated Recurrent Unit Neural Network

The Gated Recurrent Unit (GRU) structure shown in Fig. 3 is a special case of LSTM and has a simpler structure. To make predictions, it employs gated mechanisms to control input, memory, and available information in the current timestep. While an LSTM cell consists of three gates, a GRU cell contains only two gates: *reset* r_t and *update* z_t [25], [26]. The *reset gate* determines how the new input information is combined with the previous memory content while the *update gate* defines the content stored at the current timestep.

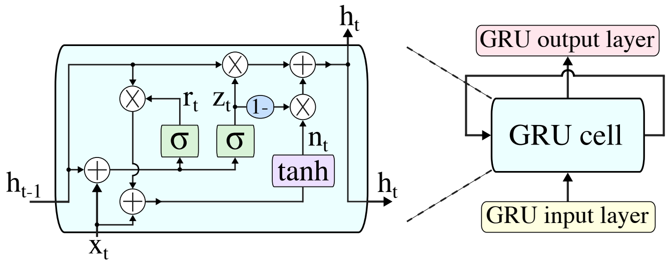


Fig. 3. Repeating module for the GRU neural network.

The outputs of the *reset gate* r_t and the *update gate* z_t at time *t* are [24]:

$$r_t = \sigma(W_{ir} x_t + b_{ir} + U_{hr} h_{t-1} + b_{hr}) \quad (6)$$
$$z_t = \sigma(W_{iz} x_t + b_{iz} + U_{hz} h_{t-1} + b_{hz}), \quad (7)$$

where σ is the sigmoid function, x_t is the input, h_{t-1} is the previous output of the GRU cell, W_{ir}, U_{hr}, W_{iz}, and U_{hz} are the weight matrices, and b_{ir}, b_{hr}, b_{iz}, and b_{hz} are the bias vectors.

The output of the GRU cell is:

$$h_t = (1 - z_t) * n_t + z_t * h_{t-1}, \quad (8)$$

where n_t is:

$$n_t = tanh(W_{in} x_t + b_{in} + r_t * (U_{hn} h_{t-1} + b_{hn})). \quad (9)$$

W_{in} and U_{hn} are the weight matrices while b_{in} and b_{hn} are the bias vectors.

C. Bidirectional LSTM Neural Network

The Bidirectional Recurrent Neural Network (BRNN) [27] utilizes additional information for prediction. It contains a forward and a backward layer that connect to the same output layer. Forward and backward cell states are not connected and no delays are introduced to include future information. The Bi-LSTM neural network [21] (a version of BRNN) shown in Fig. 4, is an extended form of the one-way LSTM.

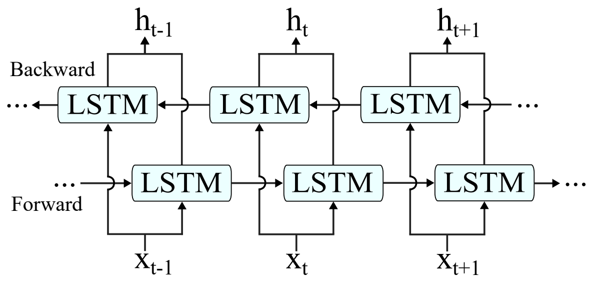


Fig. 4. Module structure for the Bi-LSTM neural network.

D. Deep Learning Neural Network

We consider a deep learning neural network model shown in Fig. 5. It is based on the LSTM, GRU, and Bi-LSTM algorithms, which do not require a priori selection of features. The "pandas" [28], a Python data analysis library, was used to create input matrices from *KDDTrain*$^+$, *KDDTest*$^+$, and *KDDTest*$^{-21}$ datasets. The three categorical features "protocol_type", "service", and "flag" are converted to numerical features using the *pandas.get_dummies()* function generating 71 additional features. The *nn.LSTM()* and *nn.GRU()* functions are used to create the LSTM, GRU, and Bi-LSTM models.

The input layer consists of 109 nodes (each node corresponds to one feature) that serve as inputs to the RNN layer. The length of the employed input time sequence is 50 data points. The hidden layers contain one RNN (LSTM, GRU, or Bi-LSTM) and two fully-connected layers (FC$_1$ with ReLU function and FC$_2$). A dropout layer with the rate 0.5 is added before FC$_2$. In case of two-way and five-way classifications, the model consists of 109 RNN, 80 FC$_1$, and 32 FC$_2$ hidden nodes. For five-way classification, the number of FC$_2$ output nodes is 5. In the training phase, we use cross entropy function to calculate loss. In the test phase, the probability for each target class is calculated in the output layer using the *softmax* function [29].

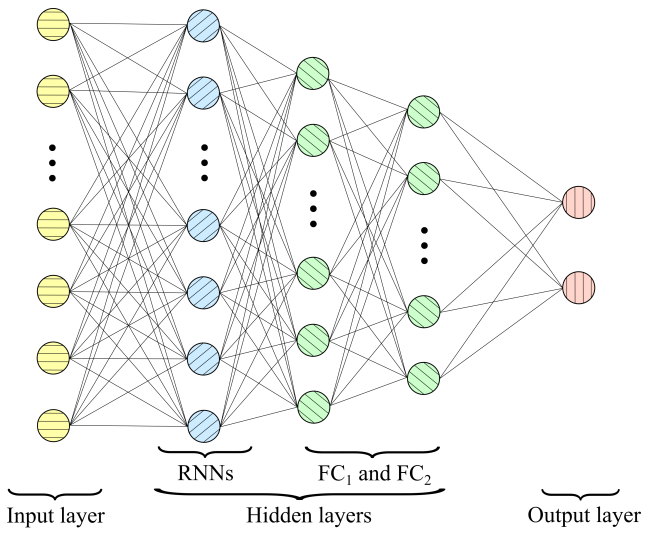


Fig. 5. Deep learning neural network model. The model consists of 109 RNN, 80 FC$_1$, and 32 FC$_2$ hidden nodes.

E. Broad Learning System

The Broad Learning System (BLS) is based on Random Vector Functional-Link Neural Network (RVFLNN). It creates mapped features and enhancement nodes. Three variants of BLS algorithms [15] were proposed to increase the number of mapped features, enhancement nodes, and input data. A module of the BLS algorithm is shown in Fig. 6. We first select a certain number of data points from the entire training dataset to be used as the input patterns. These data points are used to form additional feature and enhancement nodes. The remaining training data are included in the system as new input $\boldsymbol{X}_a$ in every step of the incremental learning, resulting in updating weights.

In the initialization step, matrix $\boldsymbol{A}_n^m$ contains n groups of mapped features and m groups of enhancement nodes. The additional feature and enhancement nodes are introduced as [15]:

$$\boldsymbol{A}_x = [\ \boldsymbol{Z}_x^n \mid \xi(\boldsymbol{Z}_x^n \boldsymbol{W}_{h1} + \boldsymbol{\beta}_{h1}), ..., \xi(\boldsymbol{Z}_x^n \boldsymbol{W}_{hm} + \boldsymbol{\beta}_{hm})\ ], \tag{10}$$

$$\boldsymbol{Z}_x^n = [\phi(\boldsymbol{X}_a \boldsymbol{W}_{e1} + \boldsymbol{\beta}_{e1}), ..., \phi(\boldsymbol{X}_a \boldsymbol{W}_{en} + \boldsymbol{\beta}_{en}], \tag{11}$$

where ξ and ϕ are projection mappings, $\boldsymbol{W}_{hi}$ and $\boldsymbol{\beta}_{hi}$ ($i = 1, 2, ..., m$), and $\boldsymbol{W}_{ej}$ and $\boldsymbol{\beta}_{ej}$ ($j = 1, 2, ..., n$), are weights and bias, respectively. The updating matrix is defined as:

$$^x\boldsymbol{A}_n^m = \begin{bmatrix} \boldsymbol{A}_n^m \\ \boldsymbol{A}_x^T \end{bmatrix}. \tag{12}$$

It is calculated using the pseudoinverse matrix:

$$(^x\boldsymbol{A}_n^m)^+ = [\ (\boldsymbol{A}_n^m)^+ - \boldsymbol{B}\boldsymbol{D}^T | \boldsymbol{B})\ ], \tag{13}$$

where $\boldsymbol{D}^T = \boldsymbol{A}_x^T (\boldsymbol{A}_n^m)^+$,

$$\boldsymbol{B}^T = \begin{cases} (\boldsymbol{A}_x^T - \boldsymbol{D}^T \boldsymbol{A}_n^m)^+, & \text{if } \boldsymbol{A}_x^T - \boldsymbol{D}^T \boldsymbol{A}_n^m \neq 0 \\ (1 + \boldsymbol{D}^T \boldsymbol{D})^{-1} (\boldsymbol{A}_n^m)^+ \boldsymbol{D}, & \text{if } \boldsymbol{A}_x^T - \boldsymbol{D}^T \boldsymbol{A}_n^m = 0. \end{cases} \tag{14}$$

The updated weights are calculated as:

$$^x\boldsymbol{W}_n^m = \boldsymbol{W}_n^m + (\boldsymbol{Y}_a^T - \boldsymbol{A}_x^T \boldsymbol{W}_n^m)\boldsymbol{B}, \tag{15}$$

where $\boldsymbol{Y}_a$ are the corresponding labels of $\boldsymbol{X}_a$.

IV. Experimental Procedure

The experimental procedure consists of five steps:

- *Step 1*: Convert categorical features into numerical features using dummy coding for training and test datasets.
- *Step 2*: Normalize training and test datasets.
- *Step 3*: Tune the model parameters during the 10-fold validation.
- *Step 4*: Test the LSTM, GRU, Bi-LSTM, and BLS models using *KDDTest*$^+$ and *KDDTest*$^{-21}$ datasets.
- *Step 5*: Evaluate derived models based on accuracy and F-Score for binary and multi classes.

Training and testing were performed using a Dell Alienware Aurora with 32 GB memory, NVIDIA GeForce GTX 1080 GPU, and Intel Core i7 7700K processor. A deep learning network model was implemented using PyTorch [24], a neural network library designed for Python deep learning framework. Its *torch.autograd()* function provides automatic differentiation when back-propagation is required, leading to reduced code complexity. The *torch.cuda()* function was used to activate the GPU acceleration. When compiling the LSTM, GRU, and Bi-LSTM networks, the "Adam" optimizer [30] was used because of its superior performance when dealing with large datasets and high-dimensional parameter spaces. We trained the model using 50 epochs with learning rate $lr = 0.001$. The results of the BLS [31] algorithm are generated using MATLAB R2017b. The *bls_train_inputenhance()* function is utilized for training. The BLS training parameters are shown in Table IV.

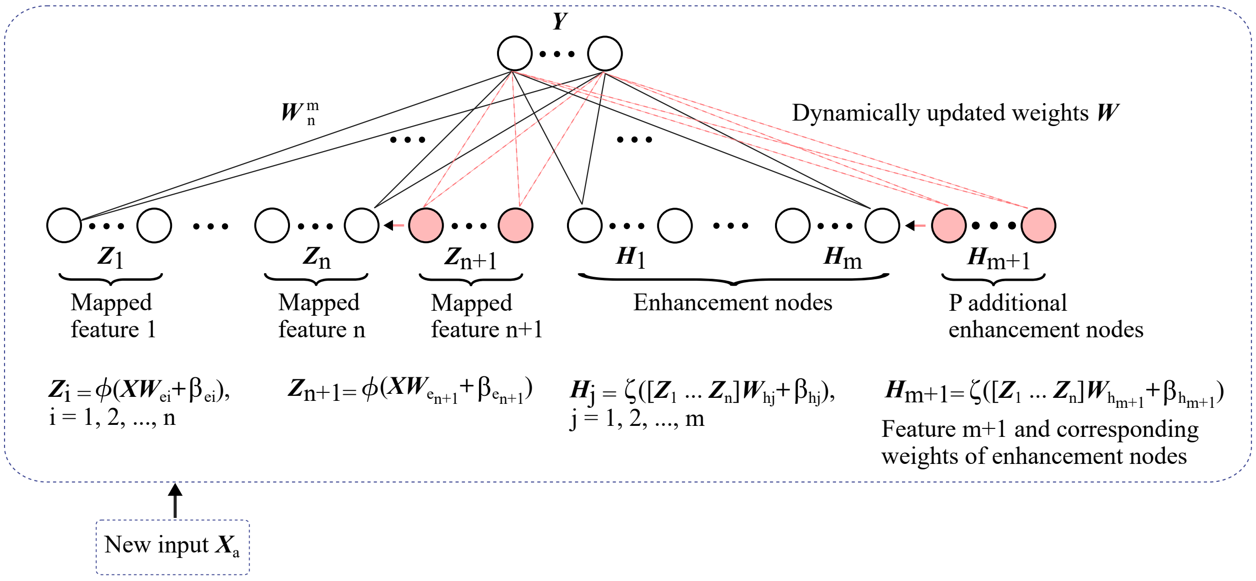
TABLE IV
BLS Training Parameters

Number	
Feature nodes per window	10
Windows	10
Enhancement nodes	200
Added input patterns per incremental step	1,000
Added enhancement nodes per incremental step	60
Steps of incremental learning	4

The four algorithms (LSTM, GRU, Bi-LSTM, and BLS) use the same training and test datasets shown in Table V.

TABLE V
The NSL-KDD Training and Test Datasets

Model	Training dataset	Test dataset
LSTM, GRU, Bi-LSTM, and BLS	*KDDTrain*$^+$	*KDDTest*$^+$ *KDDTest*$^{-21}$



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

Fig. 6. Module of the BLS algorithm with increment of mapped feature, enhancement nodes, and new input data [15].

V. Performance Evaluation

A. Evaluation Metrics

The confusion matrix shown in Table VI is used to evaluate performance of classification algorithms. True positive (TP) and false negative (FN) are the number of intrusion data points that are classified as intrusion and regular, respectively. False positive (FP) and true negative (TN) are the number of regular data points that are classified as intrusion and regular, respectively. The performance measures are:

$$\text{Accuracy} = \frac{\text{TP + TN}}{\text{TP + TN + FP + FN}}, \quad (16)$$

$$\text{F-Score} = 2 \times \frac{\text{precision} \times \text{sensitivity}}{\text{precision + sensitivity}}, \quad (17)$$

where:

$$\text{precision} = \frac{\text{TP}}{\text{TP + FP}} \text{ and sensitivity} = \frac{\text{TP}}{\text{TP + FN}}. \quad (18)$$

TABLE VI
Confusion Matrix

	Predicted class	
Actual class	Intrusion (positive)	Regular (negative)
Intrusion (positive)	TP	FN
Regular (negative)	FP	TN

B. Performance Comparison

We measure the classification performance of the considered deep learning and broad learning models based on accuracy and F-Score. For two-way and five-way classifications, the BLS model generated high accuracies when using the *KDDTest*$^{+}$ and *KDDTest*$^{-21}$ datasets: 84.15 % and 72.64 % shown in Table VII and 82.47 % and 70.30 % shown in Table VIII. The BLS model also achieved high F-Scores (84.68 % and 80.61 %) compared to algorithms shown in Table IX. The BC+k-NN algorithm offered the best performance in case of two-way classification [13]. No total F-Score was reported [13] for five-way classification. We have not compared results that used different number of NSL-KDD test data [12].

The BLS algorithm performs well in classification tasks. Although the conventional RNN algorithm often achieves high accuracy, its performance is limited to using time sequential input data. The GRU algorithm has fewer parameters and, therefore, it has shorter training time while the LSTM exhibits better performance with larger datasets. Reported accuracies and F-Scores were achieved with 10 and 50 epochs for BLS and the three variants of RNNs, respectively. Results generated using the RNN-IDS model [11] required 100 epochs on a CPU platform. The BLS approach has the shortest training time (21.92 s), as shown in Table X.

VI. Conclusion

Three types of RNNs and a BLS have been described and employed to detect network intrusions. When using the *KDDTest*$^{+}$ and *KDDTest*$^{-21}$ datasets, the BLS model shows better performance than the three implemented deep recurrent neural networks (LSTM, GRU, and Bi-LSTM) and most reported results. Performance of BLS depends on the number of mapped features and enhancement nodes, which have to be chosen a priori. Even though performance improves with increased number of nodes, they require additional memory and training time. An advantage of the BLS model is that it requires considerably shorter time for training than the conventional deep learning networks.

References

[1] KDD Cup 1999 Data [Online]. Available: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed: July 18, 2018.

TABLE VII
TWO-WAY CLASSIFICATION: ACCURACY

Model		LSTM	GRU	Bi-LSTM	BLS	J48	NB	NB Tree	[7] RF	RT	MP	SVM	[10] NIDS	[11] RNN-IDS	[13] BC+k-NN
Accuracy (%)	*KDDTest*$^+$	82.68	82.87	81.03	84.15	81.05	76.56	82.02	80.67	81.59	77.41	69.52	75.75	83.28	94.92
	KDDTest^{-21}	64.32	65.42	64.31	72.64	63.97	55.77	66.16	63.26	58.51	57.34	42.29	N/A	68.55	91.35

TABLE VIII
FIVE-WAY CLASSIFICATION: ACCURACY

Model		LSTM	GRU	Bi-LSTM	BLS	J48	NB	NB Tree	RF	[11] RT	MP	SVM	RNN-IDS
Accuracy (%)	*KDDTest*$^+$	79.56	80.17	79.44	82.47	74.60	74.40	75.40	74.00	72.80	78.10	74.00	81.29
	KDDTest^{-21}	60.51	60.75	60.80	70.30	51.90	55.77	55.40	50.80	49.70	58.40	50.70	64.67

TABLE IX
TWO-WAY CLASSIFICATION: F-SCORE

Model		LSTM	GRU	Bi-LSTM	BLS	[13] BC+k-NN
F-Score (%)	*KDDTest*$^+$	82.76	83.05	81.23	84.68	95.39
	KDDTest^{-21}	73.18	74.60	73.49	80.61	94.49

TABLE X
TRAINING TIME

Model	LSTM	GRU	Bi-LSTM	BLS	[11] RNN-IDS
Time (s)	355.86	345.04	497.66	21.92	5,516

[2] NSL-KDD Data Set [Online]. Available: https://web.archive.org/web/20150205070216/http://nsl.cs.unb.ca/NSL-KDD/. Accessed: July 18, 2018.

[3] 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' 00)*, Hilton Head, SC, USA, Jan. 2000, pp. 12–26.

[4] S. J. Stolfo, W. Fan, W. Lee, A. Prodromidis, and P. K. Chan, "Cost-based modeling for fraud and intrusion detection: Results from the JAM project," in *Proc. DARPA Inform. Survivability Conf. and Expo. (DISCEX' 00)*, Hilton Head, SC, USA, Jan. 2000, pp. 130–144.

[5] 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. Security*, vol. 3, no. 4, pp. 262-294, Nov. 2000.

[6] 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 Security and Trust (PST)*, St. Andrews, NB, Canada, Oct. 2005, pp. 1–6.

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

[8] 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.

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

[10] 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)*, Fez, Morocco, Oct. 2016, pp. 258–263.

[11] 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, pp. 21954–21961, Nov. 2017.

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

[13] 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.

[14] J. R. Quinlan, *C4.5: Programs for Machine Learning*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1993.

[15] 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.

[16] The MNIST Database of Handwritten Digits [Online]. Available: http://yann.lecun.com/exdb/mnist/. Accessed: July 18, 2018.

[17] The NORB dataset, v1.0 [Online]. Available: https://cs.nyu.edu/~ylclab/data/norb-v1.0/. Accessed: July 18, 2018.

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

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

[20] 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.

[21] A. Graves, N. Jaitly, and A.-R. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM," in *Proc. IEEE Workshop Autom. Speech Recognition Understanding*, Olomouc, Czech Republic, Dec. 2013, pp. 273–278.

[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] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *Proc. Annual Conf. Int. Speech Commun. Assoc. (INTERSPEECH)*, Singapore, Sept. 2014, pp. 338–342.

[24] PyTorch [Online]. Available: https://pytorch.org/docs/stable/nn.html/. Accessed: July 18, 2018.

[25] 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 Language Process. (EMNLP)*, Doha, Qatar, Oct. 2014, pp. 1724–1734.

[26] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: encoder-decoder approaches," in *Proc. 8th Workshop Syntax, Semantics Structure in Statistical Transl. (SSST-8)*, Doha, Qatar, Oct. 2014, pp. 103–111.

[27] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[28] Pandas [Online]. Available: https://pandas.pydata.org/. Accessed: July 18, 2018.

[29] C. M. Bishop, *Pattern Recognition and Machine Learning*. Secaucus, NJ, USA: Springer-Verlag, 2006, p. 115.

[30] 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.

[31] Broadlearning [Online]. Available: http://www.broadlearning.ai/. Accessed: July 18, 2018.