Expert Systems With Applications 185 (2021) 115524

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A bidirectional LSTM deep learning approach for intrusion detection

Yakubu Imrana [a], Yanping Xiang [a,*], Liaqat Ali [b,c], Zaharawu Abdul-Rauf [d]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China (UESTC), Chengdu 611731, China
[b] School of Information and Communication Engineering, University of Electronic Science and Technology of China (UESTC), Chengdu 611731, China
[c] Department of Electrical Engineering, University of Science and Technology, Bannu, Pakistan
[d] Department of Education, University for Development Studies (UDS), Tamale, Ghana

ARTICLE INFO

Keywords:
Machine learning
Deep learning
Recurrent neural networks
Bidirectional LSTM
Intrusion detection

ABSTRACT

The rise in computer networks and internet attacks has become alarming for most service providers. It has triggered the need for the development and implementation of intrusion detection systems (IDSs) to help prevent and or mitigate the challenges posed by network intruders. Over the years, intrusion detection systems have played and continue to play a very significant role in spotting network attacks and anomalies. Numerous researchers around the globe have proposed many IDSs to combat the threat of network invaders. However, most of the previously proposed IDSs have high rates of raising false alarms. Additionally, most existing models suffer the difficulty of detecting the different attack types, especially User-to-Root (U2R) and Remote-to-Local (R2L) attacks. These two types of attacks often appear to have lower detection accuracy for the existing models. Hence, in this paper, we propose a bidirectional Long-Short-Term-Memory (BiDLSTM) based intrusion detection system to handle the challenges mentioned above. To train and measure our model's performance, we use the NSL-KDD dataset, a benchmark dataset for most IDSs. Experimental results show and validate the effectiveness of the BiDLSTM approach. It outperforms conventional LSTM and other state-of-the-art models in terms of accuracy, precision, recall, and F-score values. It also has a much more reduced false alarm rate than the existing models. Furthermore, the BiDLSTM model achieves a higher detection accuracy for U2R and R2L attacks than the conventional LSTM.

1. Introduction

Information and Communication Technology (ICT), the Internet of Things (IoT), and other mobile devices have gained massive improvement in recent years. These technologies' massive growth has amounted to a significant increase in the number of individuals and many organizations depending and relying on wireless networks to accomplish various tasks. Regarding the rise in internet usage, most individuals' lives and how most organizations work have significantly changed. However, this rapid growth in Internet services and the large amount of information traffic have resulted in many security concerns in recent times. To deal with these concerns and make networks more secure, security researchers proposed many different techniques and ideas (Berman et al., 2019). Intrusion Detection Systems (IDSs) have proven to be one of the most promising, if not the best, ways to identify and deal with network intruders or invaders quickly. IDSs can identify already intruded network systems and systems that are experiencing intrusion.

Intrusion Detection Systems monitor network traffic and make a proper analysis of the network. It also spots possible attacks (anomalies) and unauthorized network access (Jang-Jaccard & Nepal, 2014). Ideally, IDS is computer software or program(s) that can gather and analyze various security criteria (metrics or parameters) concerning a network. In general, when the word IDS is mentioned, three key methodological concepts come to play: anomaly detection (Wenke et al., 1999), misuse detection (Cannady, 1998) and a hybrid of the two (Depren et al., 2005; Kim et al., 2014). With anomaly detection, the IDS flags when there is a deviation from a previously defined network state (Beqiri, 2009). Anomaly detection is good at spotting behavior that differs significantly from regular activity (Gregg, 2014). Misuse detection, on the other hand, is performed by comparing attack behaviors used to penetrate systems against recorded user activity (Cannady, 1998). It is essential to note that intrusions can be attacks from outside a network (outsider attacks) or authorized users seeking greater privileges (insider attacks). It could also come from privileged users attempting to misuse their privileges. Security researchers have carried

* Corresponding author.
E-mail addresses: yakubu.imrana@uds.edu.gh (Y. Imrana), 2650931580@qq.com (Y. Xiang), engr_liaqat183@yahoo.com (L. Ali), iris.zahra01@gmail.com (Z. Abdul-Rauf).

https://doi.org/10.1016/j.eswa.2021.115524
Received 19 June 2020; Received in revised form 27 June 2021; Accepted 29 June 2021
Available online 16 July 2021

out several commendable works in the domain of anomaly detection in computer networks and the internet as a whole. However, most of these works still suffer some drawbacks from the application perspective.

Many researchers proposed the use of machine learning (ML) techniques such as Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Random Forest (RF), and Naïve Bayes (NB) to detect and identify intruders of a network (Chandak et al., 2019; Horng et al., 2011; Koc et al., 2012; Manzoor & Kumar, 2017; Zhang & Zulkernine, 2006). These techniques are based on traditional ML and go with a higher computation cost. They do not get a deeper understanding of their datasets (they are shallow learners). They also give alerts that are not entirely true (i.e., they raise false alarms). Contrary to traditional ML, the latest approach referred to as deep learning has shown state-of-the-art performance on many problems (Liaqat, Ce et al., 2019; Liaqat, Shafqat et al., 2019) including intrusion detection. Deep learning provides automated tools for deep feature extraction. It gives a better representation of data for generating more improved models. Recurrent Neural Network (RNN) has become one of the most widely used approaches in deep learning for carrying out classifications and other evaluations on data sequences, building on today's research in the domain of intrusion detection (Kim & Kim, 2015; Tang et al., 2018). Moreover, RNN is a suitable method that can exhibit splendid outcomes in successive learning and enhance anomaly detection in a network system.

In this work, we propose to use a bidirectional Long-Short-Term-Memory based RNN model referred to as BiDLSTM for network intrusion detection. To train and measure our model's performance, we use the NSL-KDD dataset (UNB, 2009) that is publicly available for use in the University of New Brunswick (UNB) data repository. This work contributes meaningfully in the following ways:

i. Presents a development and implementation of an IDS using a bidirectional LSTM model that can accurately model and handle data processing.
ii. To the best of our knowledge, this is the first study that proposes the use of BiDLSTM for the intrusion detection problem.
iii. Proposes a model capable of learning the description of a data being normal or an attack-type from a labeled dataset and relating the acquired knowledge to make accurate classification on an unseen dataset.
iv. Achieves a better accuracy (of 7.59% and 4.45%) for intrusion detection than conventional LSTM. Additionally, the BiDLSTM model outperformed many other recently proposed approaches.

Subsequent sections of this work are as follows: Section 2 presents a brief explanation of deep learning approaches applied to intrusion detection. Section 3 provides a review of literature on RNN, LSTM, and intrusion detection. Section 4 presents the Bidirectional Long-Short-Term-Memory (BiDLSTM) model and describes the dataset used in this work. Section 5 provides the experiment description and the discussion of results. In Section 6, we present our conclusion and future works.

2. An overview of deep learning

Deep learning (DL) in artificial intelligence (AI) is a field that mimics the functioning of the human brain in the area of data processing and the generation of patterns for making effective decisions. It is a subset of ML that provides more advanced tools for generating models and uses a stratified form of ANN for the implementation of ML algorithms. Also referred to as deep neural learning (DNL), the edifices of deep learning are build over multiple layers linked to each other. DL can learn and convert their input data into multiple levels of abstraction of data representation (Bengio et al., 2013; LeCun et al., 2015). Examples of DL include deep belief networks (DBNs), deep neural networks (DNNs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs). These techniques yield commendable results close to or even beyond humans' reasoning, making them suitable for many problems in several research domains, including intrusion detection.

2.1. Recurrent neural networks

Recurrent Neural Network (RNN) as a subset of DL is a more suitable method for handling sequential data due to its recurrent (circular) manner of connection. It is a category of neural networks that utilize previous outputs as inputs while keeping hidden layers (Berman et al., 2019; Kim & Kim, 2015). RNNs can process any length of input and maintain a model's size with increasing input size. Unlike traditional feed-forward networks, RNNs can recall what they learned previously and guide their decisions based on acquired knowledge (Bengio et al., 1994). That is to say, RNNs do not only recall things during the training but also capable of recalling knowledge acquired from previous inputs in the process of creating outputs. Although RNNs can solve several research problems, they suffer from drawbacks such as vanishing gradients (Bengio et al., 1994; Pearlmutter, 1995; Pineda, 1987). This drawback makes them unable to learn long-term dependencies. To combat this drawback, Hochreiter and Schmidhuber (1995) proposed the Long-Short-Term-Memory (LSTM).

2.2. Long-Short-Term-Memory (LSTM)

LSTM is a recurrent neural network that utilizes a gating mechanism to learn long-term dependencies. It solves the problem of vanishing gradient found during the training of standard RNNs. LSTM models use multiple switch gates that enable them to circumvent units and, as a result, recall longer time steps (Hochreiter & Schmidhuber, 1997). Typically, the LSTM architecture has a memory referred to as cells that accept the current input and previous state as input (Hochreiter & Schmidhuber, 1995, 1997; Staudemeyer & Omlin, 2013). These cells choose what to keep and what to discard from memory and then combine the current input and previous state as the next input. By doing so, they can capture long-term dependencies (Le et al., 2017). This ability gives LSTMs an advantage over standard RNNs, so they gain much attention in recent times. Network security researchers utilize LSTMs to deal with the most impending security issues, including intrusion detection (Kim & Kim, 2015; Kim et al., 2016; Le et al., 2017; Staudemeyer, 2015; Staudemeyer & Omlin, 2013).

3. Related works

Intrusion detection in the domain of security has been a peculiar problem faced by most researchers. Machine learning techniques have proven to be one of the most efficient methods for combating intrusion in network systems. Most researchers in this domain have proposed several ML techniques. We discuss the most related of these techniques to our study in this section.

The authors in Ikram and Cherukuri (2016), Ingre and Yadav (2015), Nie et al. (2017), Parwez et al. (2017) and Reddy et al. (2016) proposed standard machine learning methods such SVM, RF, KNN, and NB for intrusion detection. Although these methods have yielded promising results over the years, they suffered from some immanent limitations that inspired deep neural network development. In Tang et al. (2016), Tang et al. proposed a deep learning approach for intrusion detection in network systems. They applied a DL technique to flow-based intrusion detection in a software-defined network. Their model was trained and tested on the NSL-KDD dataset and achieved a good result. With feed-forward DNN, the authors in Kasongo and Sun (2019) proposed an intrusion detection system based on DL. They combined a filter-oriented feature selection with feed-forward DNN to detect network intruders. The approach utilizes an information gain mechanism and has proven from experiment to outperform most existing traditional ML approaches.

The authors in Tang et al. (2018) proposed a recurrent neural network known as GRU-RNN, which uses a gating mechanism for detecting intrusions in software-defined networks. Their approach uses the NSL-KDD dataset for testing and evaluation. According to the authors, the

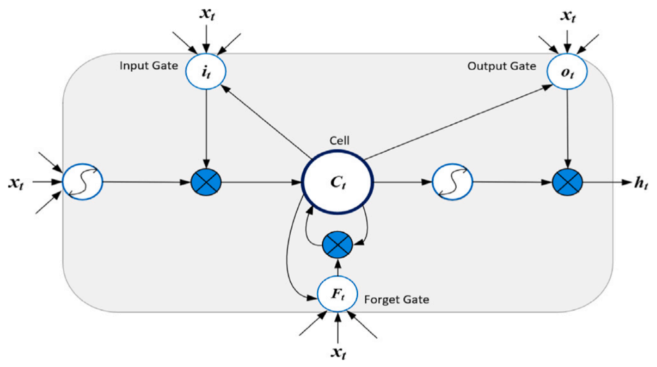


Fig. 1. Basic architecture of a long short-term memory cell

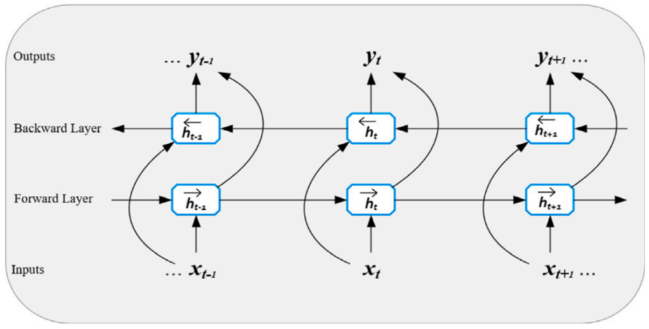


Fig. 2. Basic architecture of a bidirectional RNN.

GRU-RNN causes no deterioration to the network's performance and achieves greater accuracy in detecting anomalies. In Kim et al. (2016), Kim et al. presented a network intrusion detection model using LSTM–RNN. Their model was trained using the KDD Cup 1999 dataset and produced good accuracy, confirming DL's effectiveness on IDS. Fu et al. (2018) proposed a smart network attack detection system using LSTM–RNN. The system architecture comprises an input layer, a mean pooling layer, and a regression layer. The approach produced good performance results when trained with the NSL-KDD dataset.

In Staudemeyer (2015), Staudemeyer presented an LSTM model for network intrusion detection. The author used normal and malicious user's behaviors to model network traffic as time series. The model was trained on the DARPA and KDD Cup'99 datasets and experimented with different network topologies to evaluate the approach. The study also investigated different feature sets to detect attacks and establish training on networks specified for individual attack types. An IDS classifier was built in Le et al. (2017) using a recurrent neural network approach. The authors investigated six different optimizers for LSTM–RNN. According to the authors, the more suitable of the six optimizers is Nadam. It produced a good performance in detecting intruders compared to existing works. In Ishitaki et al. (2017), the writers presented a Deep Recurrent Neural Network (DRNN) based user behavior prediction method to monitor users' behavior on a Tor network. The authors constructed a Tor server and client used with a Wireshark network analyzer to collect data on the Tor network users and trained the DRNN model using this data.

4. Methodology

In this section, we first present the intuition of a conventional LSTM architecture. Then, we explain in detail the bidirectional LSTM (BiDLSTM) architecture. We further describe the NSL-KDD dataset used to train our model.

4.1. Conventional LSTM

A conventional LSTM has an identical control flow as RNNs, which processes data that carries on information as it propagates forward. The variations are the functions inside the cells of the LSTM. These functions allow the LSTM to retain or forget information. LSTM's fundamental principle is the cell state and the different gates. The cell state functions as a transit route that transmits relevant information throughout the data processing. It can be considered as the network's memory. The gates are diverse neural networks that determine which information is permissible on a cell state. During training, the gates will learn what information is necessary to retain or forget. There are three separate gates (i.e., the input, output, and forget gate) in an LSTM cell that controls information flow. The input gate determines which information from the current state is essential to add. The output gate defines what should be the next hidden state. The forget gate determines what is essential to retain from the previous state. Fig. 1 gives the structure of a conventional LSTM. The relationship between the inputs and outputs is described mathematically at time t and $t-1$ by the following equations:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{1}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2}$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{3}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_{t-1} + b_o) \tag{4}$$

$$h_t = o_t \tanh(c_t) \tag{5}$$

where c is the cell state. σ (the sigmoid function) and tanh denote the activation functions. x represents the input vector, the output is given by h_t. W and b denote the weights and biases parameters, respectively. f_t is the forget function which has the role of sieving out unwanted information. i_t (the input gate) and c induce new information in the cell state. o_t which is the output gate, outputs the relevant information.

4.2. Bidirectional LSTM (BiDLSTM)

The bidirectional LSTM augments standard LSTMs to improve a model's performance on classification issues. It trains two LSTMs on the input data. The first LSTM on the original input data and the other on a reversed replica of the input data. By this, more meaning is added to the network and achieves faster results. The concept behind BiDLSTM is very straightforward. It comprises duplicating the first recurrent layer in the network, then provides the input data in its original form as input to the first layer, and a reversed replica of the input data to the duplicated layer. This concept solves the problem of vanishing gradient in standard RNNs.

The training of BiDLSTM is on all past and future input information available within a particular time frame. Intuitively, BiDLSTM process input data in two directions (thus, from left-to-right and right-to-left) using a forward hidden layer and a backward hidden layer (Graves et al., 2013). In python, the Keras library uses a bidirectional layer wrapper to implement BiDLSTMs by taking the first LSTM layer as an argument. It allows users to define the merge mode, which specifies how to merge the forward and backward outputs before passing them on to the next layer (see Fig. 3). From Fig. 2, the output (y), forward hidden layer ($\overrightarrow{h}_t$), and the backward hidden layer ($\overleftarrow{h}_t$) are calculated as follows (Graves et al., 2013; Mousa & Schuller, 2017):

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}}x_t + W_{\overrightarrow{h}\overrightarrow{h}}\overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right) \tag{6}$$

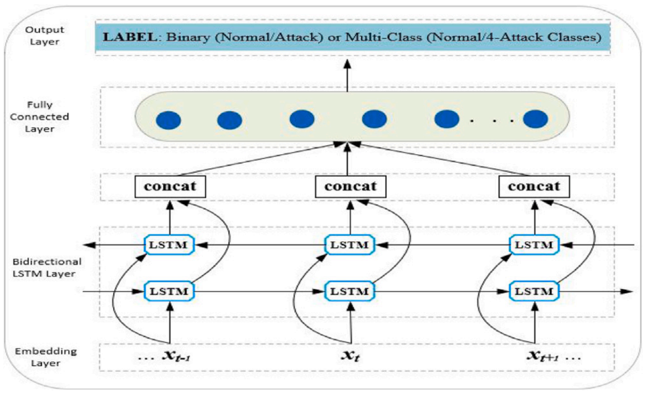


Fig. 3. BiDLSTM architecture.

Table 1
Feature list of NSL-KDD dataset.

Name of feature	Data type	Name of feature	Data type
duration	numeric	is_guest_login	numeric
protocol_type	non-numeric	count	numeric
service	non-numeric	srv_count	numeric
flag	non-numeric	serror_rate	numeric
src_bytes	numeric	srv_error_rate	numeric
dst_bytes	numeric	rerror_rate	numeric
land	numeric	srv_rerror_rate	numeric
wrong_fragment	numeric	same_srv_rate	numeric
urgent	numeric	diff_srv_rate	numeric
hot	numeric	srv_diff_host_rate	numeric
num_failed_logins	numeric	dst_host_count	numeric
logged_in	numeric	dst_host_srv_count	numeric
num_compromised	numeric	dst_host_same_srv_rate	numeric
root_shell	numeric	dst_host_diff_srv_rate	numeric
su_attempted	numeric	dst_host_same_src_port_rate	numeric
num_root	numeric	dst_host_srv_diff_host_rate	numeric
num_file_creations	numeric	dst_host_serror_rate	numeric
num_shells	numeric	dst_host_srv_serror_rate	numeric
num_access_files	numeric	dst_host_rerror_rate	numeric
num_outbound_cmds	numeric	dst_host_srv_rerror_rate	numeric
is_host_login	numeric		

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \tag{7}$$

$$y_t = W_{\overrightarrow{h}y}\overrightarrow{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y \tag{8}$$

where the term W denotes weight matrices ($W_{x\overrightarrow{h}}$ and $W_{x\overleftarrow{h}}$ are the forward input-hidden weight and backward input-hidden weight matrices respectively), the term b ($b_{\overrightarrow{h}}$ and $b_{\overleftarrow{h}}$) denotes the bias vectors in both directions, and the term $\mathcal{H}$ represents the hidden layer.

4.3. Description of dataset

The NSL-KDD dataset (Tavallaee et al., 2009; UNB, 2009) is one of the bench-marked datasets for evaluating Intrusion Detection Systems (IDS). It is an enhanced form of the KDDCup '99 dataset (Dua & Graff, 2017). The dataset comprises a training set (*KDDTrain*$^+$) with 125,973 traffic samples and two separate test sets (i.e., *KDDTest*$^+$ and *KDDTest*$^{-21}$). The *KDDTest*$^+$ has 22,544 traffic samples, and the *KDDTest*$^{-21}$ has 11,850 samples. Additionally, to make the intrusion detection more realistic, the test datasets include many attacks that do not appear in the training set (see Table 2). Thus, adding to the 22 types of attacks in the training set, 17 more different attack types exist in the test set.

Table 2
Attack categories of the different types of attacks.

Attack category		Types of attack	
		Training records	Test records
Attack	DoS	back, land, neptune, pod,smurf, teardrop	back, land, neptune, pod, smurf, teardrop, mailbomb, processtable, udpstorm, apache2, worm
	Probe	ipsweep, nmap, portsweep, satan	ipsweep, nmap, portsweep, satan, mscan, saint
	U2R	buffer-overflow, loadmodule, perl, rootkit	buffer-overflow, loadmodule, perl, rootkit, sqlattack, xterm, ps
	R2L	fpt-write, guess-passwd, imap, multihop, phf, spy, warezclient, warezmaster	fpt-write, guess-passwd, imap, multihop, phf, spy, warezmaster, xlock, xsnoop, snmpguess, snmpgetattack, httptunnel, sendmail, named
NORMAL		normal	normal

Table 3
Breakdown of traffic records in the NSL-KDD.

Attack category		Number of traffic records		
		KDDTrain$^+$	KDDTest$^+$	KDDTest^{-21}
Attack	DoS	45 927	7458	4342
	Probe	11 656	2421	2402
	U2R	52	200	200
	R2L	995	2754	2754
Normal		67 343	9711	2152
Total		**125 973**	**22 544**	**11 850**

The NSL-KDD dataset contains 41 features, including 3 non-numeric (i.e., *protocol_type*, *service* and *flag*) and 38 numeric features, as shown in Table 1. It has a class label grouped into two categories (anomaly and normal) for binary classification. For multi-class classification, we group the label into five categories (i.e., Normal, Denial of Service (DoS), User-to-Root (U2R), Remote-to-Local (R2L), and Probe). Table 3 gives a summary of the number of traffic records in the NSL-KDD dataset.

4.4. Data preparation

As mentioned in Section 4.3 above, the NSL-KDD dataset comes with 38 numeric and 3 non-numeric features. However, just as any RNN, the proposed BiDLSTM model only handles numerical data inputs. As a result, there is a need for us to convert all non-numeric features to numeric representations. The features (*protocol_type*, *service* and *flag*) are the non-numeric features in the NSL-KDD dataset that require transformation into numeric form. These three features are encoded and assigned integer values unique to each of them. After successfully transforming these features into numeric form, the next appropriate thing is feature scaling. Feature scaling ensures that the dataset is in the normalized form. The values of some features in the NSL-KDD dataset (e.g., *src_bytes* and *dst_bytes*) have uneven distribution, so we scale every feature's values within the range of (0, 1) using Min–Max scaling. By this, we ensure that our classifier does not produce biased outcomes. The expression for the Min–Max feature scaling is as follows:

$$Z' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{9}$$

Here, Z' represents the new value (scaled), and X denotes the original value.

Algorithm 1: BiDLSTM Training

```
1  Load Dataset
2  for Data in Training and Test Sets do
3      Extract Features (x)
4      Extract Labels (y)
   Input: Features Extracted
   Output: Classifications
5  for Features in x do
6      if Feature = Nonnumerical then
7          Encode Feature using Keras Library
8  Scale Features with $Z' = \frac{X - X_{\min}}{X_{\max} - X_{\min}}$
9  for $i$ from 1 → $n$ do
10     Start: K = 10
11     Split Training set into K-groups
12     Load BiDLSTM model
13     Fit model with K-1 group
14     Validate model with remaining Kth group
15     Repeat until all K-groups are used as validation set
16 Test model on Test sets ($NSL\text{-}KDDTest^{+}$ and $NSL\text{-}KDDTest^{-21}$)
```

4.5. Performance metrics

To evaluate our model's performance, we calculate the accuracy, recall, specificity, and false alarm rate. We also investigated the model's precision and F-score. Each of these metrics is explained and derived as follows:

(i) *Accuracy (ACC)*: This is the ratio of the number of correctly detected intrusions to the total number of traffic records:-

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

(ii) *Recall*: It refers to the ratio of the number of intrusion records correctly detected as intrusions to the overall anomalies:-

$$Recall = \frac{TP}{FN + TP} \tag{11}$$

(iii) *Specificity*: This is the percentage of normal records correctly detected as normal:-

$$Specificity = \frac{TN}{FP + TN} \tag{12}$$

(iv) *False Alarm Rate (FAR)*: It is the percentage of normal behaviors classified as intrusive behaviors:-

$$FAR = \frac{FP}{TN + FP} \tag{13}$$

(v) *Precision*: This refers to the ratio of the true anomalous records to the overall traffic records identified as intrusions:-

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

(vi) *F-Score*: It refers to the harmonic mean of the precision and true positive rate:-

$$F - Score = 2\left(\frac{1}{Precision^{-1} + TPR^{-1}}\right) \tag{15}$$

5. Experiment and results

In this section, we present the implementation of BiDLSTM and discuss the experimental findings. We compare the model's performance with state-of-the-art methods trained and tested on the same dataset (i.e., the NSL-KDD dataset). Also, we present a comparison of results with some recently published methods on the NSL-KDD dataset.

Table 4
Summary of the proposed model architecture.

Layer	Type	Output shape	Number of parameters
1	Embedding	41, 64	8 062 272
2	Bidirectional LSTM	41, 128	66 048
3	Dropout	41, 128	0
4	Bidirectional LSTM	64	41 216
5	Dropout	64	0
6	Fully connected (ReLU)	64	4160
7	Dropout	64	0
8	Fully connected (ReLU)	32	2080
9	Dropout	32	0
10	Fully connected (Sigmoid or softmax)	2 for binary 5 for multi-class	66 for binary 165 for multi-class

5.1. Implementation

The proposed model is a bidirectional LSTM implemented in python programming language using TensorFlow and Keras. The Adaptive Moment Estimation (Adam) algorithm is the optimizer used to update the model's weights with a learning rate of 0.001. The loss functions used are the binary cross-entropy for binary classification and the categorical cross-entropy for multi-class classification. As shown in Fig. 3, the model starts by mapping inputs to their representations using an embedding layer. It then feeds the embeddings to the LSTM layers with two processing directions. The first in the forward direction and the other in the reversed direction. The LSTM outputs are then fed to fully connected layers with the rectified linear unit (ReLU) as an activation function. Ideally, the fully connected layers learn and compile the extracted data by the LSTM layers to form a final output that passes through an output layer for classification. Finally, we apply a dropout probability of 0.2 to the layers to ensure that our model does not over-fit the data. Table 4 displays a summary of the proposed model architecture.

The model's performance is validated using a stratified K-fold cross-validation method with K set to 10. The stratified K-fold ensures that the sample percentage for each of the classes is equal in every fold. The process first shuffles the dataset and then splits it into K groups. Then fit the model with K-1 (10–1) folds and validated with the Kth folds remaining (9 folds). This process repeats until the last K-fold. Thus, it repeats until every K-fold serves as the test set. We record each fold's scores as depicted in Fig. 4 and then take the mean of these scores as the model's performance.

5.2. Results

This section presents a discussion of the experimental results and findings. The proposed BiDLSTM model was trained using a personal computer (PC) with Intel Core i5-9300H CPU and 8 GB RAM. It runs on NVIDIA GeForce GTX 1050 Ti having 4 GB dedicated GDDR5 VRAM. We conducted two experiments to evaluate the model's performance. The first experiment is a binary classification, and the second is a multi-class classification. For each experiment, we first implement the conventional LSTM and compare the performance with that of the bidirectional LSTM approach. To compare with state-of-the-art machine learning methods (i.e., J48, NB, NB Tree, SVM, RF, RT Multi-Layer Perceptron (MLP)) presented in Tavallaee et al. (2009), we simultaneously designed contrast experiments on the NSL-KDD dataset.

5.2.1. Experiment 1: Binary classification

In this experiment, we performed a binary classification (anomaly and normal) using all 41 features in the NSL-KDD dataset for conventional LSTM and the bidirectional LSTM. For this classification, the conventional LSTM model's performance is evaluated using the confusion matrices depicted in Fig. 5a and b. In contrast, Fig. 6a and b are used to measure the bidirectional LSTM model's performance.

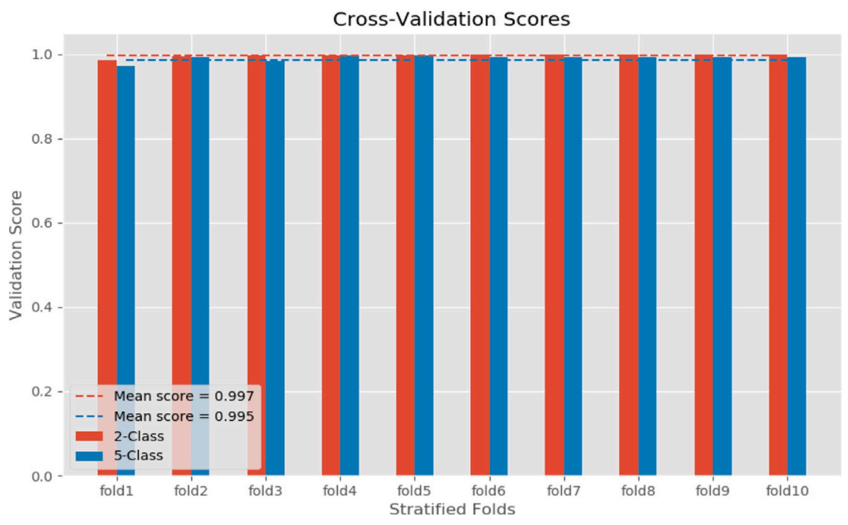


Fig. 4. Cross-validation scores over folds. The bars represent the validation score for each of the 10 folds for the two classification problems and the dashed lines indicate the mean validation score for the classification problems (i.e. 2-class and 5-class classification)

Table 5
Conventional LSTM model performance for binary classification.

Performance measure	Test dataset	
	KDDTest^{+}	KDDTest^{-21}
Accuracy	89.81%	79.87%
Recall	84.03%	79.54%
False alarm rate	2.55%	18.63%
Specificity	97.45%	81.37%
Precision	97.75%	95.06%
F-score	90.38%	86.61%

Table 6
Bidirectional LSTM model performance for binary classification.

Performance measure	Test dataset	
	KDDTest^{+}	KDDTest^{-21}
Accuracy	94.26%	87.46%
Recall	90.79%	88.32%
False alarm rate	1.15%	16.40%
Specificity	98.85%	83.60%
Precision	99.05%	96.04%
F-score	94.74%	92.02%

Tables 5 and 6 summarize the performance results obtained by the conventional LSTM and the BiDLSTM. As shown in Table 7, we compared the performance with other techniques mentioned in Hadeel et al. (2020), Javaid et al. (2016), Tavallaee et al. (2009) and Yin et al. (2017). The results prove that the BiDLSTM classifier is superior in detecting network anomalies based on the accuracy, precision, recall, specificity, and F-score values.

From Fig. 7, it can be seen that the proposed BiDLSTM obtained a higher accuracy for the 2-class classification than the other existing IDSs trained with the NSL-KDD dataset. The proposed model obtained a training accuracy of 99.95%, a testing accuracy of 94.26%, and 87.46% on the KDDTest^{+} and the KDDTest^{-21} datasets, respectively, which is superior to the results obtained by the other existing models. From Table 7, the BiDLSTM model improves the detection accuracy of the convention LSTM model by 4.45% on the KDDTest^{+} and 7.59% on KDDTest^{-21} datasets. Additionally, it obtained a greater precision rate of 99.05% and 96.04%, respectively, on the KDDTest^{+} and KDDTest^{-21} datasets compared to the other models. Furthermore, the BiDLSTM model obtained a better F-score with a reduced rate of raising false alarms, giving it an edge over the existing methods in detecting anomalies.

Table 7
Accuracy of BiDLSTM against existing methods-binary classification.

Method	Accuracy in percentage	
	KDDTest^{+}	KDDTest^{-21}
J48 (Tavallaee et al., 2009)	81.05%	63.97%
Naïve Bayes (Tavallaee et al., 2009)	76.56%	55.77%
NB tree (Tavallaee et al., 2009)	82.02%	66.16%
Random forest (Tavallaee et al., 2009)	80.67%	63.26%
Random tree (Tavallaee et al., 2009)	81.59%	58.51%
Multi-Layer Perceptron (Tavallaee et al., 2009)	77.41%	57.34%
SVM (Tavallaee et al., 2009)	69.52%	42.29%
RNN (Yin et al., 2017)	83.28%	68.55%
STL (Javaid et al., 2016)	88.39%	–
Sigmoid_PIO (Hadeel et al., 2020)	86.90%	–
Cosine_PIO (Hadeel et al., 2020)	88.30%	–
Conventional LSTM	**89.81%**	**79.87%**
Proposed BiDLSTM	**94.26%**	**87.46%**

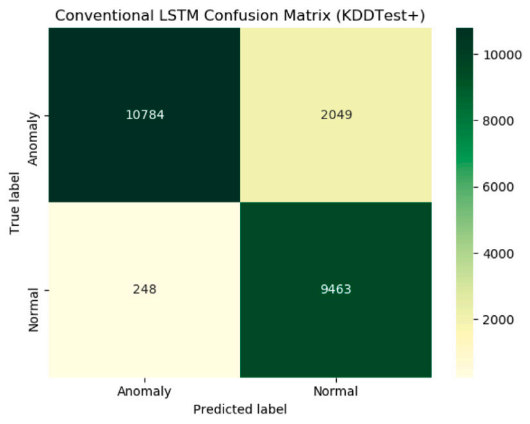
5.2.2. *Experiment 2: Multi-class classification*

The second experiment presents a 5-class (Normal, DoS, Probe, R2L, and U2R) classifier trained using all 41 features. Here, we investigate conventional LSTM and BiDLSTM model performance for multi-class classification using the NSL-KDD dataset. To compare the performance of the proposed approach with the state-of-the-art methods (i.e., J48, NB, NB Tree, SVM, RF, RT, and MLP), we trained and tested these models for the multi-class classification using WEKA (WEKA, 2016). It is an open-source data mining and machine learning software accessed via a graphical user interface, Java API, or standard terminal applications.

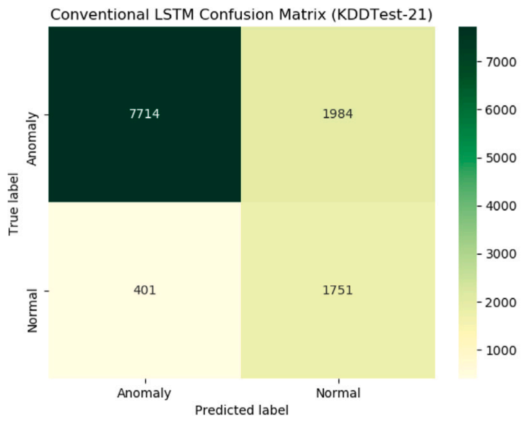
We also compared the BiDLSTM approach to some published methods on the same dataset (i.e., RNN-IDS, SCDNN, and MDPCA-DBN). Fig. 8a and b depict the confusion matrices used to evaluate the conventional LSTM. In contrast, Fig. 9a and b depict the matrices used to evaluate the BiDLSTM. The results are summarized in Tables 8 and 9. Contrary to the binary classification, both models provide lower detection accuracies for the multi-class classification.

From Tables 10 and 11, the proposed BiDLSTM not only improves the performance of the conventional LSTM, but it also has a higher detection accuracy than the state-of-the-art IDS models. Compared to the existing IDS models, the proposed BiDLSTM model achieved a greater accuracy of 91.36% and 82.05% for the KDDTest^{+} and the KDDTest^{-21}, respectively. Additionally, in raising false alarms, the proposed model achieved a lower rate of 0.88% for the KDDTest^{+} and 4.20% for the KDDTest^{-21} compared to other algorithms.

As shown in Tables 12 and 13 the BiDLSTM also outperforms some recently published models. From the two tables, MDPCA-DBN

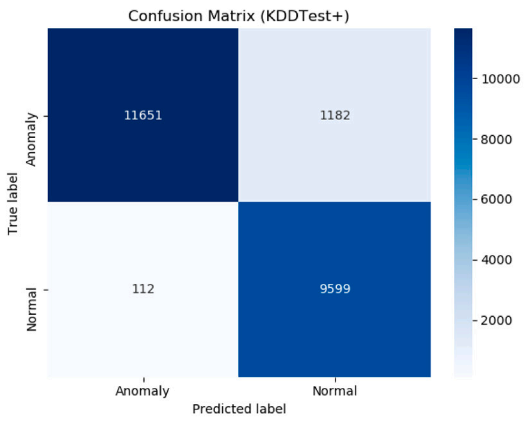


(a) Conventional LSTM Confusion Matrix for KDDTest^{+} (Binary Classification)

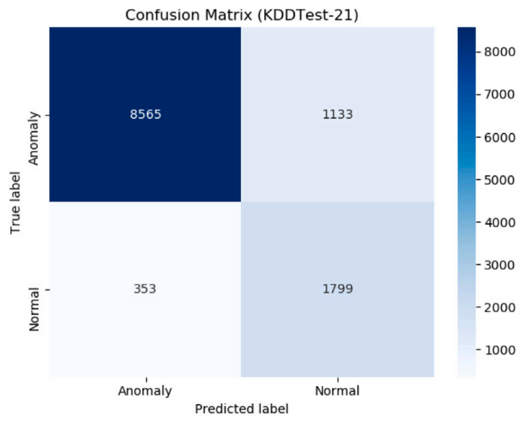


(b) Conventional LSTM Confusion Matrix for KDDTest^{-21} (Binary Classification)

Fig. 5. Conventional LSTM confusion matrices for binary classification



(a) Bidirectional LSTM Confusion Matrix for KDDTest^{+} (Binary Classification)



(b) Bidirectional LSTM Confusion Matrix for KDDTest^{-21} (Binary Classification)

Fig. 6. BiDLSTM confusion matrices for binary classification

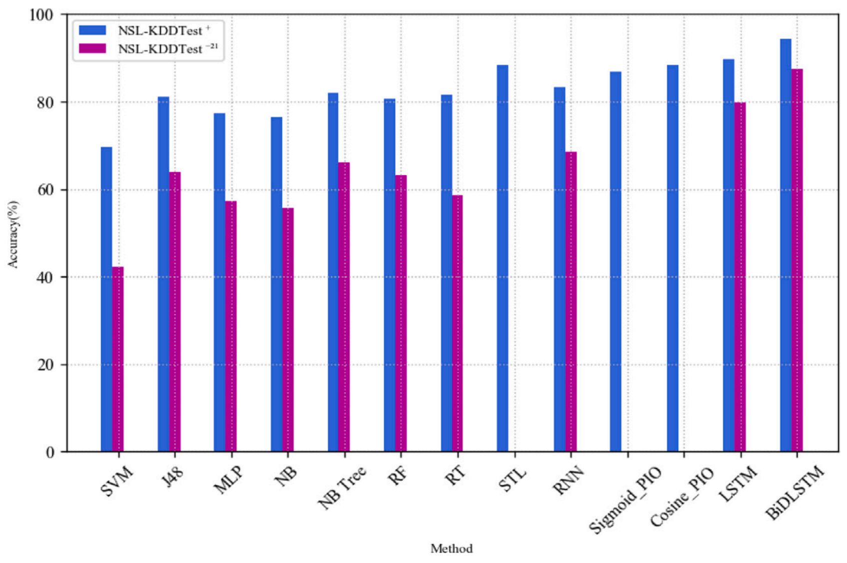
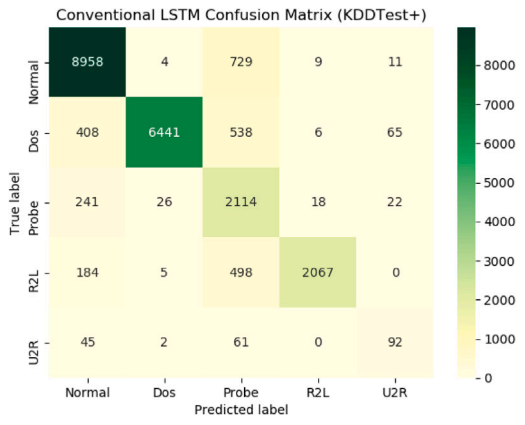
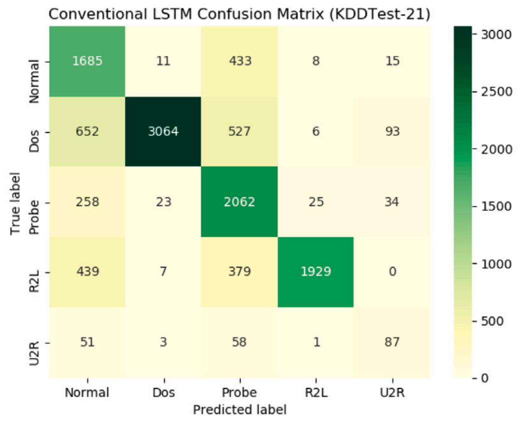


Fig. 7. Performance comparison for binary classification.

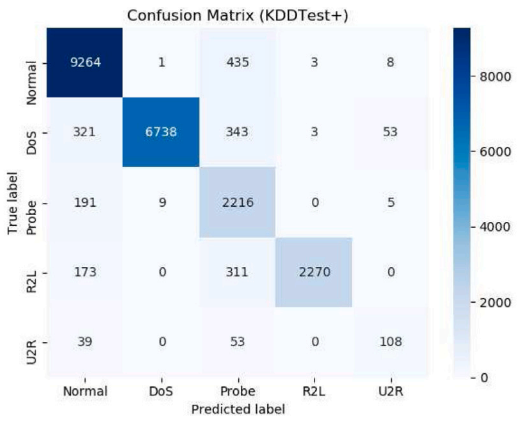


(a) Conventional LSTM Confusion Matrix for KDDTest$^+$ (Multi-Class Classification)

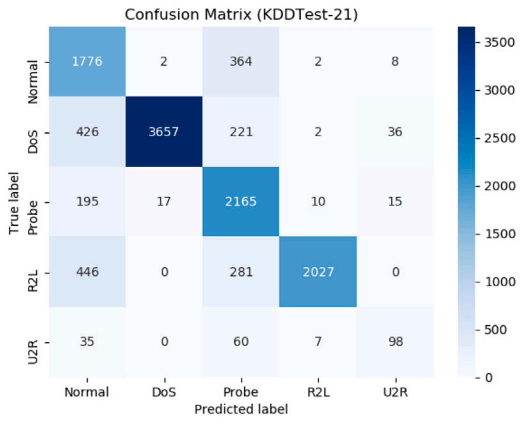


(b) Conventional LSTM Confusion Matrix for KDDTest^{-21} (Multi-Class Classification)

Fig. 8. Conventional LSTM confusion matrices for multi-class classification

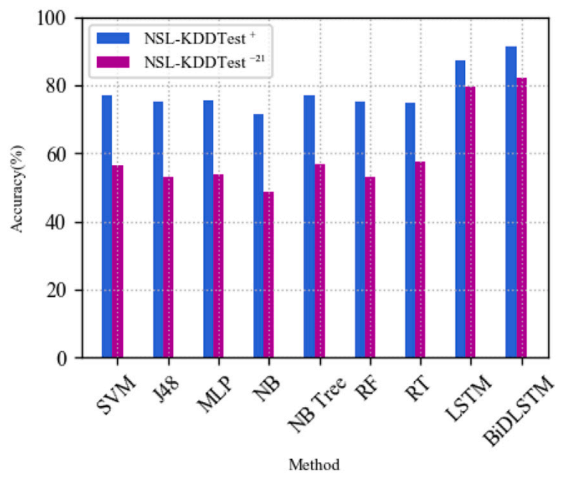


(a) Bidirectional LSTM Confusion Matrix for KDDTest$^+$ (Multi-Class Classification)

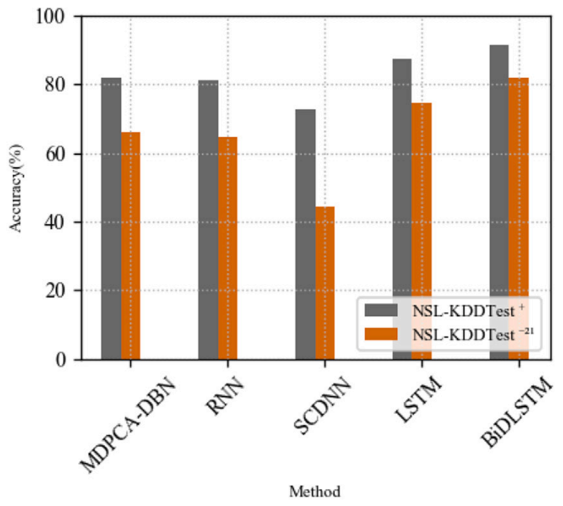


(b) Bidirectional LSTM Confusion Matrix for KDDTest^{-21} (Multi-Class Classification)

Fig. 9. BiDLSTM confusion matrices for multi-class classification



(a) Comparison of detection accuracy against state-of-the-art methods for multi-class classification



(b) Comparison of detection accuracy against other methods in literature for multi-class classification

Fig. 10. Performance comparison for multi-class classification

Table 8
Conventional LSTM model performance for multi-class classification.

Dataset	Class	Performance measure				
		Recall	FAR	Specificity	Precision	F-score
KDDTest^{+}	Normal	92.25%	6.84%	93.16%	91.07%	91.66%
	DoS	86.36%	0.25%	99.75%	99.43%	92.44%
	Probe	87.32%	9.07%	90.93%	53.65%	66.47%
	R2L	75.05%	0.17%	99.83%	98.43%	85.17%
	U2R	46.00%	0.44%	99.56%	48.42%	47.18%
KDDTest^{-21}	Normal	78.30%	14.44%	85.56%	54.62%	64.35%
	DoS	70.57%	0.59%	99.41%	98.58%	82.26%
	Probe	85.85%	14.79%	85.21%	59.61%	70.36%
	R2L	70.04%	0.44%	99.56%	97.97%	81.69%
	U2R	43.50%	1.22%	98.78%	37.99%	40.56%

Table 9
Bidirectional LSTM model performance for multi-class classification.

Dataset	Class	Performance measure				
		Recall	FAR	Specificity	Precision	F-score
KDDTest^{+}	Normal	95.40%	5.64%	94.36%	92.75%	94.06%
	DoS	90.34%	0.07%	99.93%	99.85%	94.86%
	Probe	91.53%	5.68%	94.32%	65.99%	76.69%
	R2L	82.43%	0.03%	99.97%	99.74%	90.26%
	U2R	54.00%	0.30%	99.70%	62.07%	57.75%
KDDTest^{-21}	Normal	82.53%	11.36%	88.64%	61.71%	70.62%
	DoS	84.22%	0.25%	99.75%	99.48%	91.22%
	Probe	90.13%	9.80%	90.20%	70.04%	78.83%
	R2L	73.60%	0.23%	99.77%	98.97%	84.42%
	U2R	49.00%	0.51%	99.49%	62.42%	54.90%

Table 10
Comparison between state-of-the-art methods using KDDTest^{+} for multi-class classification.

Method	Performance in percentage				
	Accuracy	Precision	Recall	FAR	F-score
SVM	77.12%	80.80%	77.12%	15.90%	73.90%
J48	75.23%	80.30%	75.23%	16.70%	71.3%
Multi-Layer Perceptron	75.66%	0.00%	75.70%	16.60%	0.00%
Naïve Bayes	71.48%	76.30%	71.50%	12.30%	71.40%
Random forest	77.07%	82.20%	77.10%	16.2%	73.10%
Random tree	75.13%	79.3%	75.10%	16.90%	70.50%
NB tree	74.65%	78.23%	74.70%	14.60%	74.48%
Conventional LSTM	**87.26%**	**90.34%**	**87.26%**	**4.03%**	**88.03%**
Proposed BiDLSTM	**91.36%**	**92.81%**	**91.36%**	**0.88%**	**91.67%**

Table 11
Comparison between state-of-the-art methods using KDDTest^{-21} for multi-class classification.

Method	Performance in percentage				
	Accuracy	Precision	Recall	FAR	F-score
SVM	56.59%	77.20%	56.60%	10.3%	56.60%
J48	53.22%	76.6%	53.20%	11.50%	51.70%
Multi-Layer Perceptron	53.81%	0.00%	53.80%	11.20%	0.00%
Naïve Bayes	48.57%	62.60%	48.60%	11.00%	50.20%
Random forest	56.79%	80.20%	56.80%	10.60%	55.3%
Random tree	53.29%	75.30%	53.30%	11.2%	50.80%
NB tree	57.58%	76.43%	57.60%	10.40%	65.69%
Conventional LSTM	**74.49%**	**81.53%**	**79.49%**	**5.96%**	**75.76%**
Proposed BiDLSTM	**82.05%**	**85.91%**	**82.05%**	**4.20%**	**82.77%**

has higher precision scores of 97.27% on the KDDTest^{+} and 95.51% on KDDTest^{-21} compared to BiDLSTM. However, BiDLSTM obtained better recall values of 91.36% and 82.05% on the two test datasets compared to the MDPCA-DBN. Because of the good recall values, the BiDLSTM outperformed MDPCA-DBN for the F-Score values. BiDLSTM achieved F-scores of 91.67% for the KDDTest^{+} and 82.77% for KDDTest^{-21} compared to the other existing models. In a nutshell, the proposed BiDLSTM, compared to the existing models, shows superiority in detecting intrusions, as shown in Fig. 10a and b.

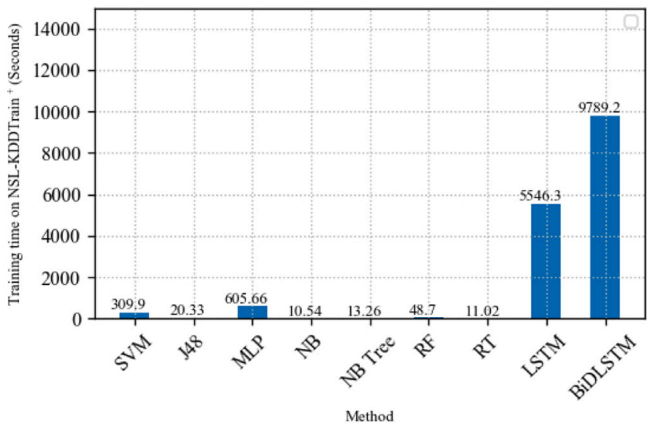


Fig. 11. Training time in seconds for different methods on KDDTrain^{+}.

5.3. Computational complexity and runtime analysis

This subsection presents the computational complexity and discusses the time required to train and test the proposed approach.

5.3.1. Computational complexity

We analyzed the overall time computational complexity of our proposed approach in respect to the building blocks of the algorithm design; input feature scaling, K-fold cross-validation, and the BiDLSTM. In scaling all features during training, the time complexity is $\mathcal{O}(ZS)$. Where Z is the number of feature and S is the feature size. For the K-fold cross-validation, we specifically use the stratified K-fold cross-validation, which requires $\mathcal{O}(Kn)$ time complexity. Where K denotes the number of times the algorithm goes through the data and n is the sample size. The main component of the algorithm, BiDLSTM trains two LSTMs. Thus, it trains one LSTM in the forward direction and the other in the reversed direction. The time required to train one LSTM in the forward direction is calculated as; $\mathcal{O}((QH) + (QM_cB_s) + (HU_f) + (M_cB_sU_f))$. Where, M_c is the number of memory cell blocks, B_s denotes the size of the cell blocks ($B_s > 0$), Q represents the number of output units, H denotes the number of hidden units. U_f represents the number of units connected forward to the hidden units, gates, and memory cells. The same time is required for the reversed direction. Hence, the total time complexity of the BiDLSTM model is calculated as; $\mathcal{O}(2[(QH) + (QM_cB_s) + (HU_f) + (M_cB_sU_f)]) = \mathcal{O}(W)$. Where W is the total number of weights optimized in the network. Therefore, the overall time complexity of the proposed approach is $\mathcal{O}(ZS + Kn + W)$. Table 14 presents the computational complexities of the different algorithms used in this study.

5.3.2. Runtime analysis

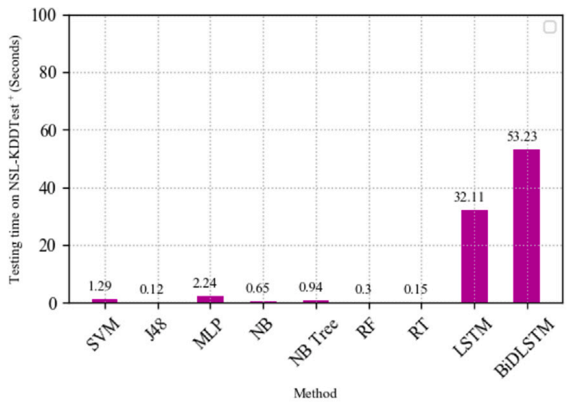
In this subsection, we discuss the training and testing timings of the methods presented in this research. To compare the various techniques' timings, we carried out all experiments on a PC with Intel Core i5-9300H and 8 GB memory. Fig. 11 depicts the training time of all methods on the NSL-KDDTrain^{+} dataset. We can observe that BiDLSTM has the highest training time, followed by the LSTM model. Whereas the training times of the other methods are comparable. As shown in Fig. 11, it takes approximately 9789 s to train the BiDLSTM model. The reason is that the entry shape is the dimension matrix of the data length and the number of features applied. The state-of-the-art models require lesser training and evaluation time because they are shallow learners. However, in this domain, the essential trait of a model is its ability to effectively and accurately detect network intrusions. Hence, there is a trade-off between performance and training time. Thus, the other methods may take less time to train than BiDLSTM, but the performance of BiDLSTM stands tall in terms of effective intrusion detection. For all models used in this analysis, evaluation times are shown in Fig. 12a and b for the NSL-KDDTest^{+} and NSL-KDDTest^{-21} datasets.

Table 12
Comparison between other methods in literature using KDDTest$^+$ for multi-class classification.

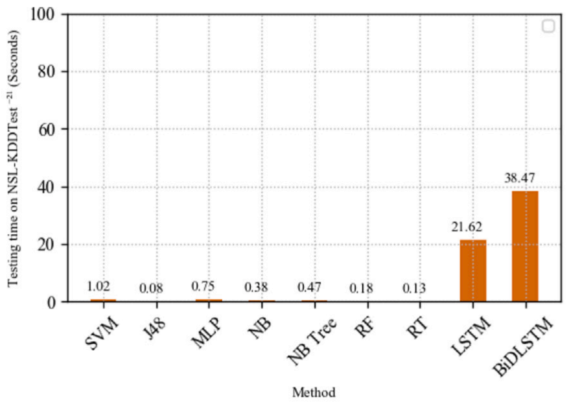
Method	Performance in percentage				
	Accuracy	Precision	Recall	FAR	F-score
MDPCA-DBN (Yang et al., 2019)	82.08%	97.27%	70.51%	2.62%	81.75%
SCDNN (Ma et al., 2016)	72.64%	–	57.48%	27.36%	–
RNN (Yin et al., 2017)	81.29%	83.07	81.29	12.42	79.25
Conventional LSTM	**87.26%**	**90.34%**	**87.26%**	**4.03%**	**88.03%**
Proposed BiDLSTM	**91.36%**	**92.81%**	**91.36%**	**0.88%**	**91.67%**

Table 13
Comparison between other methods in literature using KDDTest^{-21} for multi-class classification.

Method	Performance in percentage				
	Accuracy	Precision	Recall	FAR	F-score
MDPCA-DBN (Yang et al., 2019)	66.18%	95.51%	61.57%	13.06%	74.87%
SCDNN (Ma et al., 2016)	44.55%	–	37.85%	55.45%	–
RNN (Yin et al., 2017)	64.67%	–	–	–	–
Conventional LSTM	**74.49%**	**81.53%**	**79.49%**	**5.96%**	**75.76%**
Proposed BiDLSTM	**82.05%**	**85.91%**	**82.05%**	**4.20%**	**82.77%**



(a) Testing time in seconds for different methods on KDDTest$^+$



(b) Testing time in seconds for different methods on KDDTest^{-21}

Fig. 12. Comparison of test timings in seconds for the various methods used in the study

5.4. Limitations

The experimental results indicate that the proposed approach can be employed to detect intrusions in network systems effectively. However, more research work still needs to be carried out on this domain. Compared to the other models, the developed BiDLSTM approach provided a better detection accuracy rate on identifying different attack types. However, as pointed out in the complexity and run time analysis of the BiDLSTM model, the developed method has a higher complexity and requires more training time than the conventional LSTM model and other machine learning models. These factors are the main limitations in intrusion detection based on BiDLSTM.

6. Conclusion and future works

This work proposed the application of a deep learning approach (i.e., bidirectional Long-Short-Term Memory) to detect network intrusions. The proposed approach showed good performance and achieved accurate results. To validate our model's performance, the NSL-KDD dataset, which is extensively utilized by most researchers as the benchmark dataset for intrusion detection, was used to train the model. After the experiment, the BiDLSTM model obtained a higher accuracy, recall, and F-score than the conventional LSTM model and other existing intrusion detection models in literature. The proposed model not only efficiently improves the overall anomaly detection rate but also the detection rate of each attack class (i.e., Normal, DoS, Probe, R2L, and U2R) especially, R2L and U2R attacks. In the future, we plan to develop and explore the performance of integrated systems that would integrate some state-of-the-art feature selection methods with conventional LSTM and BiDLSTM models.

CRediT authorship contribution statement

Yakubu Imrana: Conceptualization, Formal analysis, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Yanping Xiang:** Investigation, Software, Resources, Supervision, Writing – review & editing. **Liaqat Ali:** Formal analysis, Methodology, Validation, Visualization, Writing – original draft. **Zaharawu Abdul-Rauf:** Formal analysis, Methodology, Validation, Visualization, Writing – original draft.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This research is a sub project of Independent Scientific Research Project, China (No. ZZKY-ZX-03-02-04)

Table 14
Computational complexities of the different algorithms used in this paper.

Algorithm	Complexity	Parameter description
SVM	$\mathcal{O}(N^2Z)$	N: is the number of samples
Multi-Layer Perceptron	$\mathcal{O}(ZL_{n1} + L_{n1}L_{n2} + \ldots)$	Z: is the number of features
Naïve Bayes	$\mathcal{O}(NZ)$	L_{ni}: is the number of neurons at layer i
Random forest	$\mathcal{O}(N\log(N)d.k)$	d: is the dimensionality of data
Random tree	$\mathcal{O}(ZN\log(N))$	k: is the number of decision trees
J48	$\mathcal{O}(NZ^2)$	h: is the tree height
NB Tree	$\mathcal{O}(NZ + h)$	N, Z and h
Conventional LSTM	$\mathcal{O}((QH) + (QM_cB_s) + (HU_f) + (M_cB_sU_f))$	M_c: is the number of memory cell blocks, B_s: is the size of the cell blocks ($B_s > 0$)
Proposed BiDLSTM	$\mathcal{O}(2[(QH) + (QM_cB_s) + (HU_f) + (M_cB_sU_f)])$	Q: is the number of output units, H: is the number of hidden units. U_f: is the number of units connected forward to the hidden units, gates, and memory cells.

References

Bengio, Y., Courville, A., & Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *Transactions on Neural Networks*, *5*(2), 157–166.

Beqiri, E. (2009). Neural networks for intrusion detection systems. In *Global security, safety, and sustainability. ICGS3 2009. Communications in computer and information science, Vol. 45* (pp. 156–165). Berlin, Heidelberg: Springer.

Berman, D. S., Buczak, A. L., Chavis, J. S., & Corbett, C. L. (2019). A survey of deep learning methods for cyber security. *Information*, *10*(4), 122.

Cannady, J. (1998). Artificial neural networks for misuse detection. In *National information systems security conference* (pp. 443–456).

Chandak, T., Shukla, S., & Wadhvani, R. (2019). An analysis of "A feature reduced intrusion detection system using ANN classifier. *Expert Systems with Applications*, *130*, 79–83.

Depren, O., Topallar, M., Anarim, E., & Ciliz, M. K. (2005). An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, *29*(4), 713–722.

Dua, D., & Graff, C. (2017). *UCI machine learning repository-KDD cup 1999 data set*. University of California, Irvine, School of Information and Computer Sciences, http://archive.ics.uci.edu/ml.

Fu, Y., Lou, F., Meng, F., Tian, Z., Zhang, H., & Jiang, F. (2018). An intelligent network attack detection method based on RNN. In *2018 IEEE third international conference on data science in cyberspace (DSC), Guangzhou* (pp. 483–489). IEEE.

Graves, A., Mohamed, A. R., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing, Vancouver, BC* (pp. 6645–6649). IEEE.

Gregg, M. (2014). *Certified Ethical Hacker (CEH) cert guide*. USA: Pearson Education, Inc..

Hadeel, A., Ahmad, S., Khair, S., & Eddin (2020). A feature selection algorithm for intrusion detection system based on pigeon inspired optimizer. *Expert Systems with Applications*, *148*, Article 113249.

Hochreiter, S., & Schmidhuber, J. (1995). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780.

Hochreiter, S., & Schmidhuber, J. (1997). LSTM can solve hard long time lag problems. In *Proceedings of the 9th international conference on neural information processing systems* (pp. 473–479). MIT Press.

Horng, S.-J., Su, M.-Y., Chen, Y.-H., Kao, T.-W., Chen, R.-J., Lai, J.-L., & Perkasa, C. D. (2011). A novel intrusion detection system based on hierarchical clustering and support vector machines. *Expert Systems with Applications*, *38*(1), 306–313.

Ikram, T. S., & Cherukuri, A. K. (2016). Improving accuracy of intrusion detection model using PCA and optimized SVM. *Journal of Computing and Information Technology - CIT*, *24*(2), 133–148.

Ingre, B., & Yadav, A. (2015). Performance analysis of NSL-kdd dataset using ANN. In *2015 international conference on signal processing and communication engineering systems, Guntur* (pp. 92–96). IEEE.

Ishitaki, T., Obukata, R., Oda, T., & Barolli, L. (2017). Application of deep recurrent neural networks for prediction of user behavior in tor networks. In *2017 31st international conference on advanced information networking and applications workshops (WAINA), Taipei* (pp. 238–243). IEEE.

Jang-Jaccard, J., & Nepal, S. (2014). A survey of emerging threats in cybersecurity. *Journal of Computer and System Sciences*, *80*(5), 973–993.

Javaid, A., Niyaz, Q., Sun, W., & Alam, M. (2016). A deep learning approach for network intrusion detection system. In *Proceedings of the 9th EAI international conference on bio-inspired information and communications technologies (Formerly BIONETICS)* (p. 6). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Kasongo, S. M., & Sun, Y. (2019). A deep learning method with filter based feature engineering for wireless intrusion detection system. *IEEE Access*, *7*, 38597–38607.

Kim, J., & Kim, H. (2015). Applying recurrent neural network to intrusion detection with hessian free optimization. In *Revised selected papers of the 16th international workshop on information security applications - Volume 9503* (pp. 357–369). Springer-Verlag.

Kim, J., Kim, J., Le, T., & Kim, H. (2016). Long short term memory recurrent neural network classifier for intrusion detection. In *2016 international conference on platform technology and service (PlatCon), Jeju* (pp. 1–5). IEEE.

Kim, G., Lee, S., & Kim, S. (2014). A novel hybrid intrusion detection method integrating anomaly detection with misuse detection. *Expert Systems with Applications*, *41*(4), 1690–1700.

Koc, L., Mazzuchi, T. A., & Sarkani, S. (2012). A network intrusion detection system based on a hidden Naïve Bayes multiclass classifier. *Expert Systems with Applications*, *39*(18), 13492–13500.

Le, T., Kim, J., & Kim, H. (2017). An effective intrusion detection classifier using long short-term memory with gradient descent optimization. In *2017 international conference on platform technology and service (PlatCon), Busan* (pp. 1–6). IEEE.

LeCun, Y., Y., B., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

Liaqat, A., Ce, Z., Mingyi, Z., & Yipeng, L. (2019). Early diagnosis of Parkinson's disease from multiple voice recordings by simultaneous sample and feature selection. *Expert Systems with Applications*, *137*, 22–28.

Liaqat, A., Shafqat, K., Ullah, Noorbakhsh, G., Amiri, Imrana, Y., Iqbal, Q., Adeeb, N., & Redhwan, N. (2019). A feature-driven decision support system for heart failure prediction based on χ^2 statistical model and Gaussian Naive Bayes. *Computational and Mathematical Methods in Medicine*, *2019*.

Ma, T., Wang, F., Cheng, J., Yu, Y., & Chen, X. (2016). A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks. *Sensors*, *16*(10), 1701.

Manzoor, I., & Kumar, N. (2017). A feature reduced intrusion detection system using ANN classifier. *Expert Systems with Applications*, *88*(C), 249–257.

Mousa, A., & Schuller, B. (2017). Contextual bidirectional long short-term memory recurrent neural network language models: A generative approach to sentiment analysis. In *Proceedings of the 15th conference of the European chapter of the association for computational linguistics: Volume 1, Long Papers'* (pp. 1023–1032). Association for Computational Linguistics.

Nie, L., Jiang, D., & Lv, Z. (2017). Modeling network traffic for traffic matrix estimation and anomaly detection based on Bayesian network in cloud computing networks. *Annals of Telecommunications*, *72*, 297–305.

Parwez, M. S., Rawat, D. B., & Garuba, M. (2017). Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, *13*(4), 2058–2065.

Pearlmutter, B. A. (1995). Gradient calculations for dynamic recurrent neural networks: A survey. *Transactions on Neural Networks*, *6*(5), 1212–1228.

Pineda, F. J. (1987). Generalization of backpropagation to recurrent and higher order neural networks. In *Proceedings of the 1987 international conference on neural information processing systems* (pp. 602–611). MIT Press.

Reddy, R. R., Ramadevi, Y., & Sunitha, K. V. N. (2016). Effective discriminant function for intrusion detection using SVM. In *2016 international conference on advances in computing, communications and informatics (ICACCI), Jaipur* (pp. 1148–1153). IEEE.

Staudemeyer, R. C. (2015). Applying long short-term memory recurrent neural networks to intrusion detection. *South African Computer Journal (SACJ)*, *56*, 136–154.

Staudemeyer, R. C., & Omlin, C. W. (2013). Evaluating performance of long short-term memory recurrent neural networks on intrusion detection data. In *Proceedings of the South African institute for computer scientists and information technologists conference* (pp. 218–224). Association for Computing Machinery.

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2016). Deep learning approach for network intrusion detection in software defined networking. In *2016 international conference on wireless networks and mobile communications (WINCOM), Fez* (pp. 258–263). IEEE.

Tang, T. A., Mhamdi, L., McLernon, D., Zaidi, S. A. R., & Ghogho, M. (2018). Deep recurrent neural network for intrusion detection in SDN-based networks. In *2018 4th IEEE conference on network softwarization and workshops (NetSoft), Montreal, QC* (pp. 202–206). IEEE.

Tavallaee, M., Bagheri, E., Lu, W., & Ghorbani, A. (2009). A detailed analysis of the KDD CUP 99 data set. In *2009 IEEE symposium on computational intelligence for security and defense applications, Ottawa* (pp. 1–6). IEEE.

UNB (2009). NSL-KDD dataset. https://www.unb.ca/cic/datasets/nsl.html.

WEKA (2016). The workbench for machine learning. https://www.cs.waikato.ac.nz/ml/weka.

Wenke, L., Stolfo, S. J., & Mok, K. W. (1999). A data mining framework for building intrusion detection models. In *Proceedings of the 1999 IEEE symposium on security and privacy (Cat. No.99CB36344)* (pp. 120–132). IEEE.

Yang, Y., Zheng, K., Wu, C., Niu, X., & Yang, Y. (2019). Building an effective intrusion detection system using the modified density peak clustering algorithm and deep belief networks. *Applied Science*, *9*(2), 238.

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *IEEE Access*, *5*, 21954–21961.

Zhang, J., & Zulkernine, M. (2006). A hybrid network intrusion detection technique using random forests. In *First international conference on availability, reliability and security (ARES'06)* (pp. 262–269). IEEE.