Applying Long Short-Term Memory Recurrent Neural Network for Intrusion Detection

Sara Althubiti
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
saalthub@aggies.ncat.edu

William Nick
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
wmnick@aggies.ncat.edu

Janelle Mason
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
jcmason@aggies.ncat.edu

Xiaohong Yuan
Albert Esterline
Department of Computer Science
North Carolina A&T State University
Greensboro, NC, USA
{xhyuan, esterlin}@ncat.edu

***Abstract*—These days, web applications are used extensively. While organizations benefit from the new abilities they provide, the chance of being targeted is increased, which may cause massive system damage. It is thus important to detect web application attacks. Web intrusion detection systems (IDSs) are important for protecting systems from external users or internal attacks. There are however, many challenges that arise while developing a powerful IDS for unexpected and irregular attacks. Deep Learning approaches provide several methods, and they can detect known and unknown attacks. Long Short-Term Memory (LSTM) is a type of Recurrent Neural Network (RNN) and has the ability to remember values over arbitrary intervals. LSTM is a suitable method to classify and predict known and unknown intrusions. In this work, we propose a deep learning approach to construct an IDS. We apply LSTM RNNs and train the model using the CSIC 2010 HTTP dataset. An LSTM model using the Adam optimizer can construct an efficient IDS binary classifier with an accuracy rate of 0.9997.**

Keywords*— *Intrusion detection system, Long Short-Term Memory, Recurrent Neural Network

I. INTRODUCTION

With the rapid development of the Internet, web applications have been widely used and are often under the threat of attack due to extensive usage. Cyber-attackers may cause massive damage to the system in several ways such as stealing data, destroying a specific target, disrupting access, or compromising a website. Web Intrusion Detection Systems (WIDSs) must be developed to detect abnormal behaviors and activities in a system to reduce anticipated risks of attacks.

Intrusion detection system (IDS) is a powerful technique initially implemented for detecting abnormal activities in a target application or a computer. An IDS has two main techniques [1], signature-based (misuse) detection and anomaly-based detection. Misuse detection is utilized to detect known attacks by using pattern matching or rule-based methods. On the other hand, anomaly detection is employed for detecting both known and unknown attacks by studying their behavior.

Prominent researchers studied the effectiveness of different intrusion detection techniques. A clustering-based approach is used to protect web servers from attacks using malicious HTTP (Hypertext Transfer Protocol) access patterns [2]. A survey of various machine learning techniques to build web intrusion detection systems such as Random Forest (RF), Logistic Regression (LR), and Decision Tree (DT) is presented in [3]. A combination of the K-means clustering algorithm and the J48 Decision Tree algorithm as an artificial intelligence machine learning technique was proposed to reduce the false positive rate in abnormal intrusion detection data [4].

In the research reported here, we implement a Long Short-Term Memory (LSTM) model for intrusion detection using the CSIC 2010 HTTP dataset. Then we compile the model utilizing an Adam optimizer, seeking an optimal solution for the binary intrusion classification problem using accuracy rate as a performance measure.

II. RELATED WORK

First, Traditional IDS approaches such as machine learning and pattern recognition do not involve AI (artificial intelligence) characteristics. Their ability to detect dynamic and complex cyber-attacks on web applications is limited compared to deep learning approaches because the later include multilayer processing delivering better accuracy. Hence, we expect a deep learning approach for intrusion detection, RNN-IDS, to be able to extract better data representations to generate better models.

In [5], an IDS model with deep learning was constructed by applying an LSTM architecture for a Recurrent Neural Network (RNN), and the IDS model was trained using the KDD Cup 1999 dataset. In the training stage, extracted instances from KDD Cup 1999 dataset are used. Experiments were done with a range of values to find a proper learning rate and hidden-layer size. In the testing stage, 10-fold cross-validation was used to measure the performance and compare it to other IDS classifiers. It was found that the LSTM-RNN classifier detected attacks accurately: it has the highest DR (Detection Rate) and accuracy although its FAR (False Alarm Rate) is barely better than that of the other classifiers. The relation between the dataset and the initial weight values was studied to improve the FAR by investigating the performance change when all hyper-parameters are the same. A deep learning approach was confirmed as an effective approach for implementing IDSs.

A deep learning approach was also used to find the most appropriate optimizer among six optimizers (RMSprop, Adagrad, Adadelta, Adam, Adamax, and Nadam) for LSTM RNN [6]. The LSTM RNN model with a Nadam optimizer was an efficient intrusion detection approach with 97.54% accuracy rate, 98.95% detection rate, and 9.98% false alarm rate.

In [7], an RNN-IDS was proposed as a deep learning approach to study the performance of binary model classification and multi-class classification. The effects of the number of neurons and of different learning rates on the performance of the proposed model were investigated. Also, the proposed approach was compared with other approaches, which are Support Vector Machine (SVM), J48, Random Forest (RF), naïve Bayes, and Artificial Neural Network (ANN) using the NSL-KDD dataset. Results showed that the RNN-IDS is a very appropriate classification model with higher accuracy, lower false acceptance rate, and superior performance in both binary and multi-class classification approaches compared to previously used methods. The RNN-IDS model shows marked accuracy improvement for intrusion detection.

In [8], the performance of Long Short-Term Memory Recurrent Neural Networks (LSTM-RNNs) on classifying intrusion detection data was evaluated on a time series model of the KDDCup99 dataset. Confusion matrix, accuracy, the Receiver Operating Characteristic (ROC) curve and the corresponding Area-Under-the-Curve (AUC) are the appropriate performance measures applied to the dataset. Their results show that ROC-curves are well suited for selecting well-performing neural networks. Trained LSTM neural networks learned at least partially all five traffic classes (normal, denial of service, network probes, user-to-root, remote-to-local attacks). In most trained networks, target neurons representing normal traffic and network probes indicated excellent performance, and the target neuron representing DoS attacks attains almost perfect discrimination between attacks and other traffic.

III. Method

RNN is a well-known model in deep learning [6]. RNNs are used to identify images and text and generally provide high performance. An RNN, however, has the disadvantage of not capturing long-term dependencies, which might connect the current task with the previous one and the base of vanishing gradient descent. Consequently, LSTM is introduced to capture long-term dependencies [9]. The primary goal of LSTM is to accomplish disappearing gradient descent, which is an optimization algorithm for finding weights in artificial neural networks in a way that avoids long-term dependency issues.

For long-term memory purposes, the hidden node is replaced by an LSTM cell as depicted in Fig. 1. Each LSTM cell comprises three principal gates, the input gate, forget gate, and output gate; beside, which is the cell state at time *t*. The input gate governs the new value entering the cell, the forget gate regulates the remaining value in the cell, and the output gate adjusts the used value to calculate the output activation of the LSTM cell. A decision made by a sigmoid layer about what information will be removed from the cell state is the initial step in the LSTM cell. The information comes from, and the output is a number between 0 and 1 for each number in the cell state, where 0 denotes "complete removal" and 1 denotes "complete retaining".

Next, a decision is made on what latest information will be stored in the cell state, and this step has two parts. First, a sigmoid layer i_t decides which values will be updated. Next, a tanh layer generates a vector of weights that might be added to the state. Then we update the state by combining the two parts. The old cell state c_{t-1} must then be updated into the new cell state c_t. Next, a product of $c_{t-1}$and f_t is added to the product of i_t and *tanh*() to attain new values for the update in each state. The last step is to control the output, which relies on the filtered version of the cell state. A sigmoid layer decides what portions of the cell state are going to the output. That part of the cell state will be input to *tanh* and the result will provide the output.

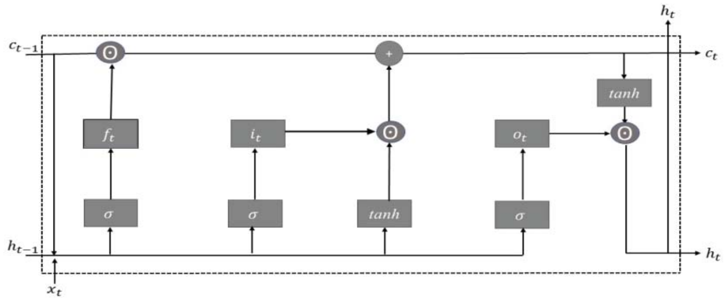

Fig. 1. Long Short-Term Memory

Define The equations to compute the values of the previous the three gates are as follows.

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

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

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (3)$$

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

$$h_t = o_t \odot tanh(c_t) \quad (5)$$

Here x_t, h_t, and c_t denote the input layer, hidden layer, and cell state at time t, respectively. b_i, b_f, b_c, and b_o correspond to bias at the input gate, forget gate, cell state, and the output gate, respectively. σ is the sigmoid function, $\odot$ corresponds to element-wise multiplication., W_{xi}, W_{xf}, W_{xc}, and W_{xo} are weights matrices that link the input layer to the input gate, forget gate, cell state, and output gate, respectively. W_{hi}, W_{hf}, W_{hc}, W_{ho} are weight matrices that attach hidden layer to the input gate, forget gate, cell state, and output gate, respectively. Finally, W_{ci}, W_{cf}, and W_{co} are weight matrices that join the cell state layer to the input gate, forget gate, and output gate, respectively.

IV. Adam Optimizer

Adam is an optimizer that adjusts iterative weights in training data. It is also required for compiling deep learning models implemented in Python using Keras, a deep learning library. And it is an alternative technique to compute learning rates for all parameters and works well in practice. Adam configuration is easy since the default parameter configuration performs well on most problems. Adam retains a decaying average of previous gradients m_t and then gets gradients concerning the stochastic object at timestep t as follows.

$$g_t \leftarrow \nabla_\Theta f_t(\theta_{t-1}) \tag{6}$$

$$m_t = \beta_1 m_{t-1}(1-\beta_1)g_t \tag{7}$$

$$v_t = \beta_2 v_{t-1}(1-\beta_2)g_t^2 \tag{8}$$

Here $\mathrm{g_t}$, β_1, and β_2 are the gradient of the objective function, the exponential decay rate for the first moment estimates, and the exponential decay rate for the second moment estimates, respectively. $\mathrm{m_t}$ and $\mathrm{v_t}$ are estimations of the mean and the uncentered variance of the gradients. Also, as m_t and $\mathrm{v_t}$ are reset as vectors of 0's, the authors of Adam noted that they are biased to zero, particularly through the initial time steps, and specifically when decay rates are low. They counter these biases by calculating bias-corrected first- and second-moment estimates:

$$m_t^{new} = \frac{m_t}{1-\beta_1^t} \tag{9}$$

$$v_t^{new} = \frac{v_t}{1-\beta_2^t} \tag{10}$$

Updated parameters are used, which produces Adam update rule:

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{v^{new}}+\in} m_t^{new} \tag{11}$$

The projected default values for β_1, β_2, and $\in$ are 0.9, 0.999, and 10^{-8}, respectively. There is ample evidence that Adam works well in practice and compared to other adaptive learning algorithms.

V. Dataset and Feature Extraction

The CSIC 2010 HTTP dataset [11] was used for the experiment; it consists of generated traffic targeted to an e-commerce Web application. The dataset has 36,000 normal requests and more than 25,000 abnormal requests. In this data, the requests are labeled as normal or abnormal and include several attacks, such as SQL injection, buffer overflow, information gathering, file disclosure, CRLF injection, XSS, and so on. Reading through [12], we found nine features as listed in Table 1 that we considered significant for the detection process (see Table 1).

TABLE I. Names of 9 features that are considered relevant for the detection of Web attacks in the CSIC-2010 HTTP dataset.

Feature Name
Length of the request
Length of the arguments
Number of arguments
Number of digits in the arguments
Length of the path
Number of letters in the arguments
Number of letter chars in the path
Number of 'special' chars in the path
Maximum byte value in the request

VI. Evaluation Metrics

In our model, the most important performance indicator (Accuracy, AC) of intrusion detection is used to measure the performance of the LSTM-RNN model. In addition to the accuracy, we introduce the precision and recall. In the following, TP and TN refer to the number of true positives and negatives, respectively, and FP and FN refer to the number of false positives and negatives, respectively. P = TP + FN is the number of (possibly misclassified) positive observations, and N = TN + FP is the number of (possibly misclassified) negative observations. Accuracy is the percentage of the number of records classified correctly versus total the records:

$$\frac{\mathrm{TP+TN}}{\mathrm{TP+TN+FP+FN}} \tag{12}$$

Precision is the proportion of positive predictions that are correct:

$$\frac{\mathrm{TP}}{\mathrm{TP+FP}} \tag{13}$$

Recall is the proportion of all positive observations that are classified as such:

$$\frac{\mathrm{TP}}{\mathrm{TP+FN}} \tag{14}$$

VII. Experiments

In our experiments, we split the original dataset into two groups, where 67% of the data is used as training data and 33% is used as data to test. Furthermore, both the training and the testing sets contain abnormal instances. In this section, we conducted two experiments. The first experiment was finding hyperparameter values to get the best performance of IDS model. Secondly, we measured the performance with the hyperparameter values attained from the first experiment. We used a neural network that contained three layers, hidden, input, and output. We trained the LSTM model that contained an input layer with nine neurons, which corresponds to the nine features, a hidden layer with six neurons, and an output layer that either produced normal or abnormal results with a single neuron. The number of iterations was set to 100 epochs after conducting some experimentation. We initialize the network weights to the default uniform weights, which ranged between 0 and 0.05. While compiling the model, we specified the loss function to assess the weights and use logarithmic loss, which is defined in Keras as "**binary_crossentropy"** since the problem is a binary classification. However, the loss function intends to measure variation between predicted and actual values. Then we fit the model on the data using the **fit()** function and specified the amount of epochs and the batch size. After we trained the model on the complete dataset, we are able to evaluate the performance on the same dataset through accuracy and model loss.

A. Hyperparameters Initialization

To enhance the training result of the neural network model hyperparameters are used to accomplished this. First, we defined hyperparameter values, as shown in Table 2, and then applied them to the LSTM model. The learning rate regulates the weight update at the end of each batch and the best results were attained using 0.01. A single hidden layer was used for small datasets. Using more hidden layers makes the model deeper and provides more accurate results. The number of epochs is the frequency of passes through the entire training set. Batch size is the number of patterns exposed to the network before updated weights. Both the number of epochs and batch size are sensitive to memory and their used values are optimal. Next, we implemented the LSTM RNN model with the Adam optimizer with parameters as shown in Table 3. Adam is originated from adaptive moment estimate. We used Adam optimizer for several reasons such as easy implementation, efficient computation, less memory requirements, and suitable for large data.

TABLE II. THE HYPERPARAMETERS' VALUES OF THE MODEL

Name of hyperparameters	**Value**
Learning rate	0.01
Number of hidden layers	6
Number of epochs	100
Batch size	50

TABLE III. THE OPTIMIZER VALUES OF THE MODEL

Name of hyperparameters	**Value**
Learning rate	0.001
Beta_1	0.9
Beta_2	0.999
Epsilon	1e-08
Decay	0.0

VIII. EXPERIMENTAL RESULTS AND FINDINGS

Examining the selected optimizer for LSTM RNN to detect intrusion is a crucial step. Hence, we implemented the proposed method on IDS using CSIC 2010 HTTP dataset with Adam optimizer. We used the same approach as our experimentation, where we divided the dataset into two sets, 67% was used for training data and 33% was used for testing data. We configured the Adam optimizer learning rate to the value 0.01, which attained better-classifying results and better performance as shown in Table 4. We were able to conclude that training the LSTM RNN model with the Adam optimizer on IDS achieves a decent result for binary classifying on IDS.

TABLE IV. THE AVERAGE CLASSIFICATION PERFORMANCE

Optimizer	**Accuracy**	**Recall**	**Precision**
Adam	0.9997	0.995	0.995

We evaluated the model performance on the classification results for the dataset using accuracy, which is a common method to evaluate the model effectiveness. In the model, the accuracy scores deliver the required point of comparison while evaluating LSTM not only in the training dataset, but also testing dataset as depicted in Figure 2. The accuracy of the method providing context is shown when comparing the prediction of the training dataset to the testing dataset.

The model has a good fit since the model performance is good on training and testing sets of data and it can be identified from Figure 2, where the training and testing loss decrease and stabilize close to each other. LSTM is trained using fit() function, where it returns a variable named history that keeps record of the loss and accuracy during model compilation. Also, history is able to callback records of training metrics for an epoch, which comprises the loss and accuracy of classification problems for testing dataset. In the model, accuracy plot depicted in Fig. 2, shows that both datasets are trained well and obtained high accuracy. From the loss plot, the model has comparable performance on both training and validation datasets (labeled test) unless parallel plots start to depart it may be interpreted as a sign to stop training at an earlier epoch.

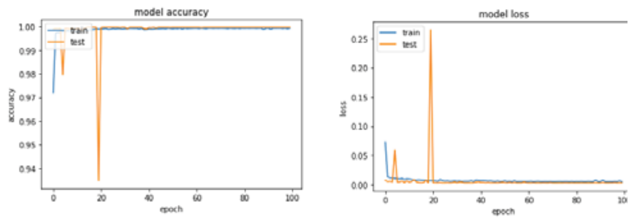

Fig. 2. Model accuracy and loss (nine features)

The purpose of this study was to show how deep learning could be effective with different sets of features to classify HTTP requests as normal and abnormal traffic by applying them on the CSIC 2010 HTTP dataset. [13] selected the five most significant features due to lower accuracy, but when deep learning is used the full set of features attains high accuracy. This study showed that using deep learning techniques has a high accuracy in five compared to nine features resulted in 99.57% and 99.97% compared to [13] results as shown in Table 4. The five feature model results are depicted in Figure 3 as model accuracy and loss.

TABLE V. THE AVERAGE CLASSIFICATION PERFORMANCE

Methods	**RF**	**LR**	**J48**	**ABc**	**SGDc**	**NB**
Accuracy	99.94	99.94	99.94	99.94	99.88	88.83
Precision	99.90	99.90	99.90	99.90	99.90	89.00
Recall	99.90	99.90	99.60	99.90	99.90	88.80

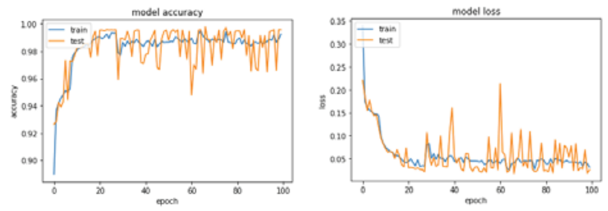

Fig. 3. Model accuracy and loss (five features)

IX. CONCLUSION

In this paper, we found that Adam optimizer is appropriate for the LSTM RNN model in detecting intrusion. We conclude that LSTM RNN model using Adam optimizer can construct an efficient IDS binary classifier. This classifier performance is measured with an accuracy rate of 0.9944. In short, in future work, we will apply LSTM to more recent intrusion detection datasets and evaluate the performance of complex LSTMs with different optimizers.

REFERENCES

[1] Mell, P., Hu, V., Lippmann, R., Haines, J., & Zissman, M. (2003). An overview of issues in testing intrusion detection systems.

[2] Pereira, H., & Jamhour, E. (2013, May). A clustering-based method for intrusion detection in web servers. In *Telecommunications (ICT), 2013 20th International Conference on* (pp. 1-5). IEEE.

[3] Pham, T. S., & Hoang, T. H. (2016, October). Machine learning techniques for web intrusion detection—A comparison. In *Knowledge and Systems Engineering (KSE), 2016 Eighth International Conference on* (pp. 291-297). IEEE.

[4] Landress, A. D. (2016, March). A hybrid approach to reducing the false positive rate in unsupervised machine learning intrusion detection. In *SoutheastCon, 2016* (pp. 1-6). IEEE.

[5] Kim, J., Kim, J., Thu, H. L. T., & Kim, H. (2016, February). Long Short Term Memory Recurrent Neural Network Classifier for Intrusion Detection. In *Platform Technology and Service (PlatCon), 2016 International Conference on* (pp. 1-5). IEEE.

[6] Kim, J., & Kim, H. (2017, February). An Effective Intrusion Detection Classifier Using Long Short-Term Memory with Gradient Descent Optimization. In *Platform Technology and Service (PlatCon), 2017 International Conference on* (pp. 1-6). IEEE.

[7] 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.

[8] Staudemeyer, R. C., & Omlin, C. W. (2013, October). 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). ACM.

[9] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.

[10] D. Kingma, J.Ba. ”Adam: A method for Stochastic Optimization”. 3rd International Conference for Learning Representations, arXiv: 1412.6980, 2015.

[11] Giménez, C.T., Villegas, A.P., and Marañón, G.A., ―HTTP Dataset CSIC 2010,‖ CSIC (Spanish Research National Council), 2012, http://www.isi.csic.es/dataset/

[12] Nguyen, H.T., et al. "Application of the generic feature selection measure in detection of web attacks.‖ Computational Intelligence in Security for Information Systems. Berlin: Springer, 2011, 25-32.

[13] Althubiti, S., Yuan, X., & Esterline, A. (2017). Analyzing HTTP requests for web intrusion detection.