Computer Network Intrusion Detection Using Sequential LSTM Neural Networks Autoencoders

Ali H. Mirza and Selin Cosan
Department of Electrical and Electronics Engineering
Bilkent University, Ankara 06800, Turkey
{mirza,cosan}@ee.bilkent.edu.tr

***Abstract*—In this paper, we introduce a sequential autoencoder framework using long short term memory (LSTM) neural network for computer network intrusion detection. We exploit the dimensionality reduction and feature extraction property of the autoencoder framework to efficiently carry out the reconstruction process. Furthermore, we use the LSTM networks to handle the sequential nature of the computer network data. We assign a threshold value based on cross-validation in order to classify whether the incoming network data sequence is anomalous or not. Moreover, the proposed framework can work on both fixed and variable length data sequence and works efficiently for unforeseen and unpredictable network attacks. We then also use the unsupervised version of the LSTM, GRU, Bi-LSTM and Neural Networks. Through a comprehensive set of experiments, we demonstrate that our proposed sequential intrusion detection framework performs well and is dynamic, robust and scalable.**

Keywords. Intrusion detection, LSTM, autoencoders, unsupervised learning, sequential data.

I. Introduction

A. Preliminaries

Anomaly detection is of pivotal interest in the field of network intrusion detection [1], medical diagnosis [2], fraud detection [3] etc. The basic assumption is that a huge amount of normal data originates from a particular distribution but unknown. Whereas, few unlikely and rare observations, i.e., anomalies, originate from different unknown distributions [4]. In some domains like network intrusion detection, anomaly detection is of prime importance. This is because, a single malicious attack, i.e., an anomaly is sufficient enough to override the system and may cause severe damage [5]. For example, an unusual and out of the routine traffic in a computer network depicts the presence of a network attack.

Intrusion detection is a branch of computer network security that aims to automatically and efficiently detect the computer network attacks [6]. Intrusion detection promises the confidentiality and integrity of a system [1]. The origin of the computer network attack can be local as well as remote [7]. Various levels of security help in protection against the computer network attacks. According to [8], computer network security is a cyclic process involving three steps namely; prevention, detection and recovery. The computer attack data is sequential in nature and is of variable length. Moreover, in an online setting, we do not know beforehand whether the incoming data is malicious or not. Hence, the learning framework is unsupervised [9] for the network data in this case.

978-1-5386-1501-0/18/$31.00

In this paper, we propose a sequential autoencoder framework using Deep Neural Networks (DNNs) [10]. However, the DNNs can provide only limited performance in modelling time series and processing temporal data [11]. As a result, recurrent neural networks (RNNs) [12] are introduced, which not only handle the temporal data but also handle the time dependencies in the data. In order to cope and handle the variable length data sequence, RNNs are used to first convert the variable length data to fixed length data. Autoencoders [13] work on the fixed length data sequence in an unsupervised framework and detect subtle anomalies in the data. The use of sequential autoencoders is two-fold. First, it helps in reducing the dimension of the input data [14]. Second, autoencoders work as a feature extraction block that extracts the useful and more reliable features out of the data. Although, regular autoencoder can work on sequential data by fixing the data size, usually by padding all sequences with zero vectors to the length of the longest sequence [13]-[14]. In contrast, recurrent auto encoders can compress variable length sequences into fixed length representations [15]. Moreover, recurrent networks reuse their weight matrix for all time steps. Therefore, they can generalize dependencies between nearby frames to other positions in the sequence. As a special case of the RNNs, we use sequential long short-term memory (LSTM) autoencoders.

B. Related Work

Artificial Neural Networks (ANNs) are used in the design of intrusion detection system (IDS) [16] employing efficient backpropagation. Support Vector Machines (SVM) [17], Self-Organized Maps (SOM) [18] and Random Forests[19] are used as efficient classifiers to perform network intrusion detection. Such classifiers suffer from the deficiency that they require a fixed length data sequence to work on [17]-[19]. A lot of work is done under the supervised as well as a semi-supervised framework [20]. Work on data set like NSL-KDD [21] is under the umbrella of the supervised and semi-supervised framework. While, in real-time, this is not such a scenario. Recently, long short term memory (LSTM) neural networks allow serendipitous discovery of important long and short-term features in time series. [22] made use of an LSTM autoencoder to reconstruct video frames. LSTM and CRF autoencoders are

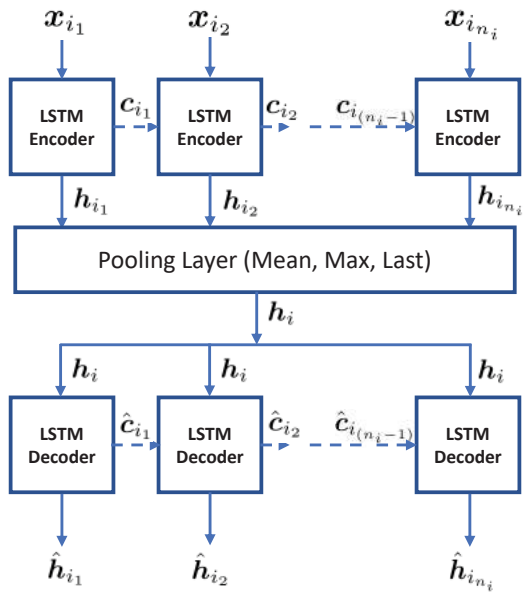

Fig. 1. Detailed description of the LSTM Sequential Autoencoder Model using the output of pooling layer, i.e., $\boldsymbol{h}_i$ as the input to all the stages of LSTM-decoder part.

similar in that they are both sequential variants of the standard autoencoder [23].

C. Contributions

Our main contributions are as follows:

- We developed an online sequential unsupervised framework for network intrusion detection using LSTM-autoencoders.
- The proposed framework is dynamic and scalable as it works on both fixed as well as variable length network data sequences.
- The proposed framework works efficiently as a feature extractor and also make use of the past information in order to make correct decisions.

II. Problem Description

In this paper, all vectors are column vectors and denoted by boldface lower case letters. Matrices are represented by boldface upper case letters. For a vector $\boldsymbol{u}$, $|\boldsymbol{u}|$ is the $\ell 1$-norm and $\boldsymbol{u}^T$ is the ordinary transpose. For a vector $\boldsymbol{x_t}$, $x_{t,i}$ and $\boldsymbol{x_{ti}}$ are the i^{th} element and the i^{th} column of the vector $\boldsymbol{x}_t$, respectively. Similarly, for a matrix $\boldsymbol{W}$, w_{ij} is the i^{th} row and j^{th} column entry. We observe the input data sequence

$$\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_n,$$

denoted by $\{\boldsymbol{X}_t\}_{t=1}^n$, where n represents the total number of observations. Here, for each observation $\boldsymbol{X}_t$, we have $\boldsymbol{X}_t = [\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_i}, \ldots, \boldsymbol{x}_{t_{n_t}}]$, $\boldsymbol{x}_{t_i} \in \mathbb{R}^d$, $t = 1, \ldots, n$, where $n_t \in \mathbf{Z}^+$ is the length of the individual input sequence and may vary for each input sequence.

We use the RNN to process the variable length input, such as $\boldsymbol{X}_t$, to extract the sequential natured information from the data. For each $\boldsymbol{X}_t$, the framework for the generic RNN for the i^{th} column of $\boldsymbol{X}_t$ is given as follows [12]:

$$\boldsymbol{h}_{t_i} = \kappa(\boldsymbol{W}\boldsymbol{x}_{t_i} + \boldsymbol{R}\boldsymbol{h}_{(t-1)_i}),$$

where $\boldsymbol{h}_{t_i} \in \mathbb{R}^m$ is the state vector and $\boldsymbol{x}_{t_i} \in \mathbb{R}^d$ is the input vector for $i = 1, \ldots, n_t$. The RNN coefficient weight matrices are $\boldsymbol{R} \in \mathbb{R}^{m \times m}$ and $\boldsymbol{W} \in \mathbb{R}^{m \times d}$. The function $\kappa(\cdot)$ is commonly set to $\tanh(\cdot)$ and apply pointwise to vectors.

Now that we have the formulation of the RNN network, we extract the sequential information by driving each column of $\boldsymbol{X}_t$ to the encoder part of the RNN network. For each $\boldsymbol{X}_t$, the output is given by

$$\boldsymbol{h}_{t_i} = \kappa_\phi^{enc}(\boldsymbol{x}_{t_i}, \boldsymbol{h}_{(t-1)_i}), \tag{1}$$

where $\boldsymbol{h}_{t_i}$ is the output of the i^{th} RNN-encoder unit and ϕ is the parameter set of the RNN-encoder part. After whole of the sequence is passed through the RNN-encoder, we get $\{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}$. We then perform three types of pooling operation on $\{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}$, i.e., mean, max and last pooling. The mean, last and max pooling operations are computed as follows:

$$\boldsymbol{h}_i = \frac{\sum_{j=1}^{n_i} \boldsymbol{h}_{t_j}}{n_i} \tag{2}$$

$$\boldsymbol{h}_i = \boldsymbol{h}_{t,n_i} \tag{3}$$

$$\boldsymbol{h}_i = \max_j \{\boldsymbol{h}_{t_i}\}_{i=1}^{n_i}, \tag{4}$$

where j is the index for the number of rows of $\boldsymbol{h}_{t_i}$. After the pooling operation, we pass $\boldsymbol{h}_i$ to the RNN-decoder part which reconstructs the input as follows:

$$\hat{\boldsymbol{h}}_{t_i} = \kappa_\psi^{dec}(\boldsymbol{h}_i, \hat{\boldsymbol{h}}_{(t-1)_i}) \tag{5}$$

$$\hat{\boldsymbol{x}}_{t_i} = \rho(\hat{\boldsymbol{h}}_{t_i}), \tag{6}$$

where $\{\hat{\boldsymbol{x}}_{t_i}\}_{i=1}^{n_i}$ is the reconstructed input and ψ is the parameter set for RNN-decoder part. The function $\rho(\cdot)$ is commonly set to $\tanh(\cdot)$ and apply pointwise to vectors. After we retrieve the reconstructed input, we evaluate the mean square loss, i.e., $\sum_{i=1}^{n_i} ||\boldsymbol{x}_{t_i} - \hat{\boldsymbol{x}}_{t_i}||^2$ and update the corresponding LSTM-encoder and decoder parameters accordingly.

As a special case of the RNNs, we use the LSTM neural network with only one hidden layer defined as follows [24]:

$$\tilde{\boldsymbol{c}}_t = g\left(\boldsymbol{W}^{(\tilde{c})}\boldsymbol{x}_t + \boldsymbol{R}^{(\tilde{c})}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(\tilde{c})}\right) \tag{7}$$

$$\boldsymbol{i}_t = \sigma\left(\boldsymbol{W}^{(i)}\boldsymbol{x}_t + \boldsymbol{R}^{(i)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(i)}\right) \tag{8}$$

$$\boldsymbol{f}_t = \sigma\left(\boldsymbol{W}^{(f)}\boldsymbol{x}_t + \boldsymbol{R}^{(f)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(f)}\right) \tag{9}$$

$$\boldsymbol{c}_t = \boldsymbol{D}_t^{(i)}\tilde{\boldsymbol{c}}_t + \boldsymbol{D}_t^{(f)}\boldsymbol{c}_{t-1} \tag{10}$$

$$\boldsymbol{o}_t = \sigma\left(\boldsymbol{W}^{(o)}\boldsymbol{x}_t + \boldsymbol{R}^{(o)}\boldsymbol{h}_{t-1} + \boldsymbol{b}^{(o)}\right) \tag{11}$$

$$\boldsymbol{h}_t = \boldsymbol{D}_t^{(o)} l(\boldsymbol{c}_t), \tag{12}$$

where $\boldsymbol{c}_t \in \mathbb{R}^m$ is the state vector, $\boldsymbol{x}_t \in \mathbb{R}^p$ is the input vector and $\boldsymbol{h}_t \in \mathbb{R}^m$ is the output vector. Here, $\boldsymbol{i}_t$, $\boldsymbol{f}_t$ and $\boldsymbol{o}_t$ are the input, forget and output gates, respectively. In (10) and (12), $\boldsymbol{D}_t^{(i)} = \text{diag}(\boldsymbol{i}_t)$, $\boldsymbol{D}_t^{(f)} = \text{diag}(\boldsymbol{f}_t)$ and $\boldsymbol{D}_t^{(o)} = \text{diag}(\boldsymbol{o}_t)$.

The functions $g(\cdot)$ and $l(\cdot)$ apply to vectors pointwise and commonly set to $\tanh(\cdot)$. Similarly, the sigmoid function $\sigma(\cdot)$ applies pointwise to the vector elements. The weight matrices are set to appropriate dimensions.

Remark 1: *The RNN Autoencoder framework discussed in (1)-(6) also applies on the LSTM neural network defined in (7)-(12). The detailed description of the sequential LSTM autoencoder framework is shown in Fig. 1. The encoder and decoder equations for the LSTM network modifies as follows:*

$$\boldsymbol{h}_{t_i} = \kappa_{\phi}^{enc}(\boldsymbol{x}_{t_i}, \boldsymbol{c}_{t_{i-1}}) \tag{13}$$

$$\hat{\boldsymbol{h}}_{t_i} = \kappa_{\psi}^{dec}(\boldsymbol{h}_i, \hat{\boldsymbol{h}}_{t_{i-1}}, \hat{\boldsymbol{c}}_{t_{i-1}}) \tag{14}$$

$$\hat{\boldsymbol{x}}_{t_i} = \rho(\hat{\boldsymbol{h}}_{t_i}), \tag{15}$$

where $\boldsymbol{h}_i$ is the LSTM state vector obtained after pooling operation as mentioned in (2)-(4).

A. Error Function and Threshold

During the reconstruction phase in the sequential LSTM autoencoder, there is an error associated with the reconstructed input. The reconstruction error for sequence $\boldsymbol{X}_i = [\boldsymbol{x}_{t_1}, \ldots, \boldsymbol{x}_{t_{n_i}}]$ is given as follows:

$$Error(i) = \sum_{i=1}^{n_i} ||\boldsymbol{x}_{t_i} - \hat{\boldsymbol{x}}_{t_i}||^2, \tag{16}$$

where $Error(i)$ is the reconstruction error for sequence $\boldsymbol{X}_i$. Based on this error measure, we update the corresponding weights of encoder and decoder part of the LSTM-autoencoder framework.

Remark 2: *For normal data sequences, the value of reconstruction error is less than the reconstruction error for anomalous data sequence. As a result, in order to classify the data as an anomaly, we assign a threshold value τ. The value of τ is critical, as it is directly related to the accuracy of the system. Table 1, shows the best achievable f1-score for a particular threshold value τ.*

III. Experiments

In this section, we demonstrate the performance of our proposed algorithm using intrusion detection evaluation data set (ISCX IDS 2012) [25]. Network payloads are captured for seven days. There are around 1.8 millions of connections for FTP, SMTP, HTTP, SSH, IMAP, and POP3. Around five percent of connections are labelled as an anomaly. These anomalies come from a diverse set of multi-stage attacks. Some connections do not have packet payloads at the source or/and destination ports. Since packet payloads are used as input in our systems, we disregard the connections without payloads at both ports, regarding anomaly occurrence rate must remain almost the same after this operation.

Each network payload, captured at both source and destination ports, is regarded as sequential character-based input. The payloads are used in hexadecimal format, so we have a total of 64 characters to be considered as our vocabulary. By using one hot encoding, characters are converted to numerical features resulting in 64-dimensional vectors.

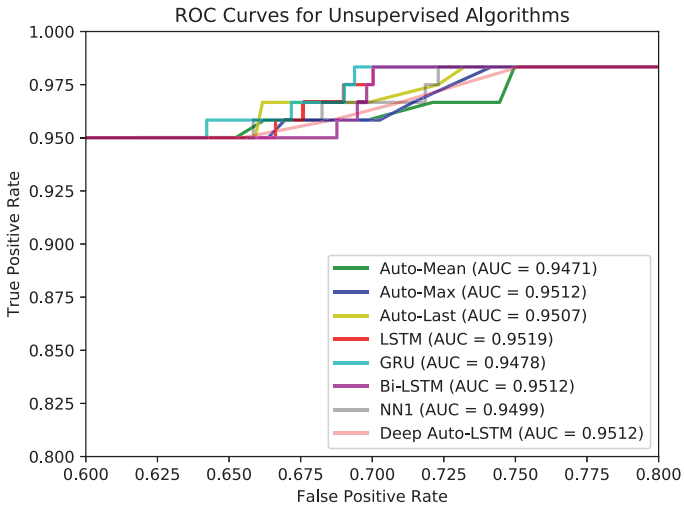

Fig. 2. ROC curves for the proposed sequential LSTM autoencoders along with several unsupervised algorithms for network intrusion detection.

We randomly split the data set into training and test sets with percentage 90 and 10, respectively. Among the training set, 20k of connections are chosen randomly to be used in our experiments. For all the splits, anomaly occurrence rate remains the same. As our training method, Adam [26] is employed with default parameters presented in the original work. The objective function is mean squared error. Batch size is chosen as 64. All the LSTM autoencoders have a unit size of 64 at both encoder and decoder layers. Sigmoid activation is used at the output layer. For all the experiments, 20k of data is split into training and test sets with size 16000 and 4000, respectively. The training set is further split into training and validation sets with 80/20 ratio. First, we compare various systems with the proposed LSTM autoencoders as shown in Fig. 2. We also added more layers in the proposed algorithm, i.e., 2 layers each in encoder and decoder part, and call it Deep Auto-LSTM networks. For the case of the RNNs, the LSTM network, the GRU network, and the bidirectional LSTM networks all with one layer is applied separately with linear regression at the end. In addition, feed-forward neural networks with one layer are trained with the SGD algorithm [27]. We use the validation set to obtain the receiver operating characteristics (ROC) and choose the threshold τ corresponding to the best AUC score. Then, the test set is evaluated with the fixed threshold τ and f1 scores are evaluated as shown in Table. I. The receiver operating characteristics with the corresponding AUC scores for the proposed sequential LSTM autoencoders and other unsupervised algorithms is shown in Fig. 2.

We carry out all the experiments using 5-fold cross-validation. We show the ROC curve for one of the experiments for the LSTM autoencoder with mean pooling in Fig. 3. The ROC and AUC scores for all the folds are shown in Fig. 3. The average AUC score is 0.96 with a standard deviation of 0.01.

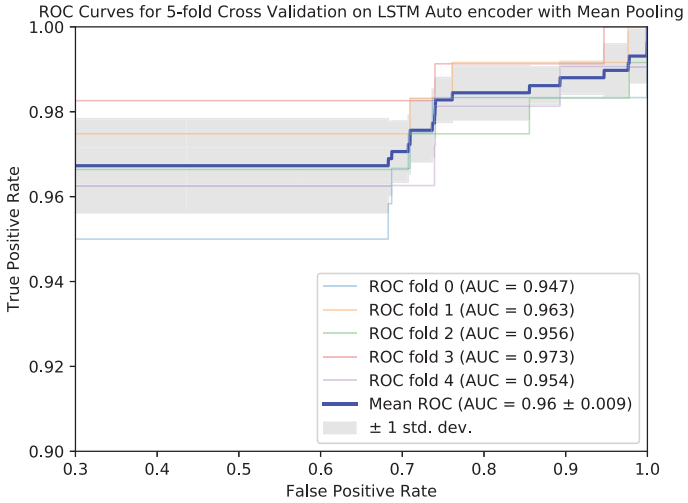

Fig. 3. 5-fold cross-validation ROC curve for LSTM Autoencoder with Mean Pooling.

TABLE I
PERFORMANCE METRICS TABLE SHOWING THE THRESHOLD, F1-SCORE AND AUC SCORE VALUES FOR THE SEQUENTIAL LSTM AUTOENCODERS AND OTHER UNSUPERVISED ALGORITHMS.

Unsupervised Learning Algorithms	Threshold	f1-score	AUC Score
LSTM	0.008	0.8409	**0.9519**
GRU	0.006	0.8102	0.9478
Bi-LSTM	0.008	0.8461	**0.9512**
NN1 - 1 layer	0.008	0.8352	0.9499
LSTM Autoencoder with Last Pooling	0.076	0.8409	0.9507
LSTM Autoencoder with Max Pooling	0.076	**0.8538**	**0.9512**
LSTM Autoencoder with Mean Pooling	0.079	0.8072	0.9471
Deep Auto LSTM	0.076	**0.8538**	**0.9512**

IV. CONCLUSION

In this paper, we propose a sequential LSTM autoencoder for executing computer network intrusion detection in an unsupervised manner. We introduced three types of pooling layer in the proposed algorithm. We select a suitable threshold value that helps in achieving the best possible f1-score for our proposed algorithm. We validate the performance of our proposed algorithm on the ISCX IDS 2012 data set. We also carry out unsupervised network intrusion detection on the LSTM, GRU, Bi-LSTM and feed-forward neural networks. Through an extensive set of experiments, we demonstrate that our proposed algorithm manages to achieve best f1 and AUC score. Out of the proposed algorithms, the LSTM autoencoder with max pooling and Deep Auto LSTM networks showed the best f1-score.

REFERENCES

[1] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[2] W.-K. Wong, A. W. Moore, G. F. Cooper, and M. M. Wagner, "Bayesian network anomaly pattern detection for disease outbreaks," in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 808–815, 2003.

[3] T. Fawcett and F. Provost, "Activity monitoring: Noticing interesting changes in behavior," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 53–62, ACM, 1999.

[4] M. Markou and S. Singh, "Novelty detection: a reviewpart 1: statistical approaches," *Signal processing*, vol. 83, no. 12, pp. 2481–2497, 2003.

[5] J. P. Anderson, "Computer security threat monitoring and surveillance," *Technical Report, James P. Anderson Company*, 1980.

[6] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, no. 2, pp. 222–232, 1987.

[7] S. J. Stolfo, S. M. Bellovin, S. Hershkop, A. D. Keromytis, S. Sinclair, and S. W. Smith, *Insider attack and cyber security: beyond the hacker*, vol. 39. Springer Science & Business Media, 2008.

[8] C. Shields, "Machine learning and data mining for computer security, chapter an introduction to information assurance," *Springer*, 2005.

[9] K. Leung and C. Leckie, "Unsupervised anomaly detection in network intrusion detection using clusters," in *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pp. 333–342, Australian Computer Society, Inc., 2005.

[10] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[11] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "Lstm: A search space odyssey," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2017.

[12] L. Medsker and L. Jain, "Recurrent neural networks," *Design and Applications*, vol. 5, 2001.

[13] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, p. 4, ACM, 2014.

[14] V. Kustikova and P. Druzhkov, "A survey of deep learning methods and software for image classification and object detection," *OGRW2014*, vol. 5, 2014.

[15] O. Fabius and J. R. van Amersfoort, "Variational recurrent auto-encoders," *arXiv preprint arXiv:1412.6581*, 2014.

[16] H. Debar, M. Becker, and D. Siboni, "A neural network component for an intrusion detection system," in *Research in Security and Privacy, 1992. Proceedings., 1992 IEEE Computer Society Symposium on*, pp. 240–250, IEEE, 1992.

[17] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[18] T. Kohonen, "The self-organizing map," *Neurocomputing*, vol. 21, no. 1-3, pp. 1–6, 1998.

[19] A. Liaw, M. Wiener, *et al.*, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[20] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, "Semi-supervised network traffic classification," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 35, pp. 369–370, ACM, 2007.

[21] S. Revathi and A. Malathi, "A detailed analysis on nsl-kdd dataset using various machine learning techniques for intrusion detection," *International Journal of Engineering Research and Technology. ESRSA Publications*, 2013.

[22] N. Srivastava, E. Mansimov, and R. Salakhudinov, "Unsupervised learning of video representations using lstms," in *International conference on machine learning*, pp. 843–852, 2015.

[23] W. Ammar, C. Dyer, and N. A. Smith, "Conditional random field autoencoders for unsupervised structured prediction," in *Advances in Neural Information Processing Systems*, pp. 3311–3319, 2014.

[24] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[26] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*, pp. 177–186, Springer, 2010.