An Enhanced Deep Autoencoder-based Approach for DDoS Attack Detection

SAMAR SINDIAN, SAMER SINDIAN
CCE Department – Faculty of Engineering
Islamic University of Lebanon (IUL)
P.O.Box: 30014 -Wardanyeh
LEBANON
samar.sindian@iul.edu.lb, samer.sindian@iul.edu.lb
www.iul.edu.lb

Abstract: - Intrusion detection systems play a crucial role in preventing security threats and defending networks from attacks. Among the attacks, distributed Denial-of-Service (DDoS) attacks literally get into the network and, in addition, they are terribly troublesome to avoid. With the advent of unknown threats, traditional machine learning approaches are impacted by lower detection rates and higher false-positive rates. As a result, the DDoS detection system requires an over-performing machine learning classifier with minimal false-positive and high detection accuracy. In this context, we propose an Improved Deep Sparse Autoencoder-based Framework (EDSA) for DDoS Attack Detection with a cost minimization strategy. The sparse autoencoder is used for dataset extraction functionality, while the softmax layer is used for traffic classification as malicious or bengin. However, intrusion detection includes the risk elements of inaccurate prediction; hence, we have used research metrics such as accuracy, precision, detection rate and specificity for our model analysis. The proposed solution uses the CICDDoS 2019 datasets and demonstrates high detection accuracy with a much less false positives percentage.

Key-Words: - Denial-of-service - Deep learning neural network - Cost minimization - Detection accuracy- False reduction – Autoendcoder – Security

Received: September 29, 2020. Revised: October 30, 2020. Accepted: November 19, 2020.
Published: December 9, 2020.

1 Introduction

Nowadays, DDoS attacks are increasingly easy to implement. The Intrusion Detection System (IDS) is a special security tool that is being used by the network experts to keep the network safe and secure from network attacks which can come from many different sources [2]. It has emerged as one of the basic and powerful tool in order to deal with data security and availability issues over the communication networks. The necessity to filter false alarms in the event that the user (device or security administrator) is overloaded with data is one major limitation of current IDS technologies.

IDSes, including active and passive, network-based and host-based, and knowledge-based and behavior-based, are categorized in several different ways: An active IDS (now more widely referred to as an intrusion prevention system) is a system that is configured without any intervention needed by an operator to automatically block suspected attacks in progress. A passive IDS is a system that is only designed to monitor and evaluate the behavior of network traffic and alert an operator to possible vulnerabilities and attacks. A network-based IDS usually consists of a network computer (or sensor) with a promiscuous-mode Network Interface Card (NIC) and a separate management interface. The IDS is located along a line or boundary of the network and controls all traffic on that line. A host-based IDS involves the installation of small programs (or agents) on individual systems to be tracked. Only the individual host systems on which the agents are mounted can be monitored by a host-based IDS; it does not track the entire network. To detect active intrusion attempts, a knowledge-based (or signature-based) IDS references a database of prior attack profiles and documented device vulnerabilities. IDS based on information is actually more popular than IDS based on actions. To detect successful intrusion attempts, a behavior-based (or statistical anomaly-based) IDS references a baseline or learned pattern of regular device operation. Deviations from this baseline or trend cause an alarm.

These attacks have a major influence of the networks and the systems as they include network performance, data security, loss of intellectual

property [3] and a real liability for the compromised notes or networks data and that is why we need a powerful IDS. Researchers are considerably reviewing the DDoS detection techniques. Neural networks are considered as one of the foremost applied methods in IDS systems. In short, there are many contributions to our proposed DDoS detection model: it introduces a novel hybrid malicious network flow detection technique focused on Autoencoder and deep neural networks. The suggested model can also prevent overfitting to predefined malicious patterns. The key motivation of this study is the concept that an autoencoder model could create a more precise classifier model behind a deep neural network model that is comparable to the traditional neural network model to detect malicious computer network traffic. Our key tasks are to create a data representation model using autoencoder techniques and to create a malicious network flow detection model using a deep neural network.

The remainder of this paper is organized as follows. The related works are introduced in Section II. Section III describes the deep neural network (DNN), auto encoder and the dropout techniques. Section IV proposes a novel intrusion detection model and shows in detail how the model works. Section V demonstrates the experimental details and results. Finally, Section VI highlights some conclusions and further work.

2 Related Work

DDoS and DoS mitigation has been researched for many years, and several, different approaches have been proposed to enhance the IDS using artificial neural networks. However, deep learning models have recently been used in the field of intrusion detection. Deep learning methods can automatically extract features and perform classification, such as AutoEncoder [1][2], DNN [3], and recurrent neural network (RNN) [4]. The introduction of deep learning into the security systems made it possible to these systems to screen benign from malicious traffic separately.

A neuro-based clustering algorithm for both wired and wireless networks was proposed by the authors of [5]. The detection of anomalies is carried out at regular intervals to monitor the analyzed traffic by means of statistical variance.
Detection of the change detects the statistical variance of the volume of traffic. The NS2 simulator was used for implementing this algorithm for a different dataset and showed better performance.

Grzegorczyk et al. in [6], suggested a DDoS defense system that includes decision tree attack detection and traceback of attackers with traffic pattern matching. It is based on the observation that the network traffic under DDoS attack would differ from the normal traffic situation and the decision tree (C4.5) generating algorithm is applied.

Stacked AutoEncoders are used to detect attacks with an overall precision of 98.60 percent on IEEE 802.11 networks [7]. A hybrid method combining spectral clustering and deep neural networks was presented by Ma et al. [8] to detect attacks on the NSL-KDD dataset with an overall precision of 72.64 percent. A software-defined network (SDN) intrusion detection system with an accuracy of 89 percent was built using the gated recurrent unit recurrent neural network (GRU-RNN) [9]. In order to detect attacks, Shone et al . [1] used a stacked non-symmetric AutoEncoder and random forest (RF).

Muna et al. [10] suggested a deep learning model-based anomaly detection technique for Internet Industrial Control Systems (IICSs), using deep auto-encoders for feature extraction and deep feedforward neural networks for classification.

3 Deep Neural Network

Deep learning refers to a class of algorithms for machine learning. Deep learning extracts features and uses nonlinear functions to transform them. The method of learning can be either supervised or unsupervised. Unsupervised feature learning is able to learn discriminative and effective features from a large amount of unlabeled data **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**. DDoS attacks are hard to detect in the sense of network security. Therefore, an efficient solution to attack detection can be given by unsupervised feature learning. In the proposed context, most representative features from the CICDDoS 2019 dataset **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** are extracted by the Sparse Autoencoder (SAE) combined with the denoising module. Then, with dropout, the learned features are fed into a neural network classifier. The following sections explain the details of the framework.

3.1 Auto-encoder

An auto-encoder is a symmetrical neural network that, by minimizing reconstruction errors, can learn the functions in an unsupervised learning manner.

One of the key benefits of using this type of model is its ability to evaluate the important parts of the input by pressuring it to learn the useful properties of the data it provides during training. In autoencoders, dimensionality reduction and regularization are primarily two methods used for dimension selection. Dimensionality reductions arise when there are fewer nodes in each hidden layer of the model than in the previous layer. Regularization chooses the nodes that have the greatest positive effect on the outcome of the model and eliminates the effect of the other nodes. The auto-encoder's basic structure is shown in Fig. 1, where in the hidden layer it attempts to learn an approximation such that the input data at the output layer can be reconstructed perfectly.

The transformation of input data into code from high-dimensional space into low-dimensional space is the mechanism of the encoder network.

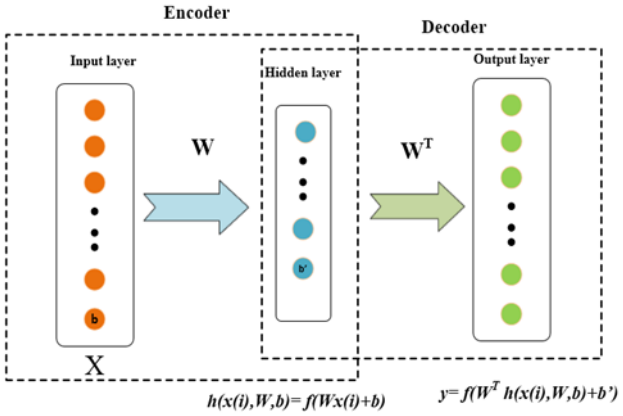

Fig. 1 the structure of auto-encoder neural network

However, the auto-encoder's fundamental issues, such as simply copying the input layer to the hidden layer, make it ineffective to extract meaningful features, although its output can be a perfect input data retrieval. A SAE can learn relatively sparse features as an extension of the auto-encoder by adding a sparse penalty term (a regularizing function) to the usual loss function of an autoencoder.

Using Kullback-Leibler divergence is one way to enforce a sparsity penalty. KL-divergence, used to calculate their similarities or dissimilarities, is of the divergence between two distributions of probability.

3.2 Scaling Data

Improving model accuracy, reducing loss, and improving convergence times are the reasons for scaling distinct features to better relate to each other. The SAE is built on the architecture of a neural network, and one way a neural network learns is by looking at changes in the traffic flows of the network features.

A difficulty with this is that the absolute values of features are perceived by deep learning based models, and not the relative difference. The loss feature is aggressive, first trying to maximize it on massive values. That is why, in order to maintain the weights of all input features of equal relative importance, we should scale the data.

In this research, by mapping the IP (Internet Protocol) address to an integer representation, a preprocessing feature is applied to the CICDDoS 2019 dataset [12]. Both the source IP address (Src IP) and the destination IP address (Dst IP) are included in the mapped IP. These two are transformed to a number representation of an integer. This research divides the knowledge into a training set and a test set with a 75:25 ratio.
New statistical features are extracted from network traffic using the autoencoder in the preprocessing stage.

The normalization module continued in this step, followed by feature extraction. The normalization module's input is the extracted functionality. To normalize the extracted features, the min-max normalization procedure is used in such a way that the training dataset consists of (0 , 1). The above value is fed to the machine learning algorithm as an input. To scale the attributes, the range (0 , 1) is used. In the classifier, the actual number between 0 and 1 is given in Eq. 1:

$$x_{norm} = \frac{x_i - x_{min}}{x_{\max} - x_{min}} \qquad (1)$$

where x_i is the value of a particular feature, x_{min} is the minimum value, $x_{\max}$ is the maximum value, and x_{norm} is the normalized value of the input that lies between [0,1].

3.3 Sparse Auto-encoder

For the auto encoder, a neural network of three layers identical to that in Fig. 1 can be constructed, where the sigmoid function is selected as the network activation function. The input layer consists of *D* neurons, and the hidden layer consists of *C* neurons, where the input vector dimension is *D*, and the function vector dimension is *C*.

The objective is to learn and obtain a function expression on a hidden layer for the unlabeled input

data *X*. To efficiently represent the input vector, the encoder converts it to code (features). The decoder input is the hidden layer output, and the decoder output is the auto-encoder output. From the code created by the encoder, the decoder tries to reconstruct the original input vector.

Let us consider X as an input vector as X={x_1 , x_2, x_3, ….x_n} where x_i is the input vector represent the original features. The mapping of input to output in encoder can be given by Eq. (2)

$$h(x(i), W, b) = f(Wx(i) + b),\ i = 1, \dots n \qquad (2)$$

Where f is an activation function, for which we used the sigmoid function defined as in Eq. 3

$$f(z) = \frac{1}{1+\exp(-z)} \qquad (3)$$

The weights between the input layer and the hidden layer are denoted by *W* and the biases are represented by *b* so that the output,

$$y = f(W^T h(x(i), W, b) + b') \qquad (4)$$

is identical to the input or refactored. Where W^T is the weight matrix which represents weight of the link connecting hidden layer neuron to the neurons in the output layer of the network, y is the reconstructed input vector, b' is the bias vector.

The auto encoder is trained by minimizing the sparse cost (loss) function defined as:

$$C_{sparse}(W, b) = E_{MSE} + E_{Reg} + E_{sparsity} \qquad (5)$$

where E_{MSE} , E_{Reg} , $E_{sparsity}$ represents the mean square error, the regularization factor and the sparsity factor respectively. The mean square error, E_{MSE} can be calculated by

$$E_{MSE} = \frac{1}{n}\sum_{i=1}^{n} ||x_i - y_i||^2 \qquad (6)$$

Where y is the reconstruction of x.

In the training data set, DNN learn every point, thus leading to overfitting of the model. This is a problem with deep networks, as the model results in poor output on new test data. To overcome this problem, the regularization factor E_{Reg} is taken into account in the objective function that can be

determined using

$$E_{Reg} = \frac{\lambda}{2}\left(\sum_{l=1}^{l_s-1}\sum_{i=1}^{D}\sum_{j=1}^{C} W_{ij}\,(l)\right) \qquad (7)$$

where λ is chosen to control the regularization term of all the weights in a particular layer, l denotes the layer number and ls denotes the total number of layers

Sparsity constraint enables a model to learn from the data about the interesting features. One solution is to incorporate an additional term in the loss function during training to penalize the KL divergence **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** in order to place a sparsity constraint on the hidden units. Sparsity factor $E_{sparsity}$can be calculated using,

$$E_{sparsity} = \beta \sum_{j=1}^{C} KL(\rho||\,\rho_j) \qquad (8)$$

where β is the sparsity weight term and $KL(\rho||\,\rho_j)$ is the KL divergence given by

$$KL(\rho||\,\rho_j) = \rho\, log\frac{\rho}{\rho_j} + (1-\rho)\log\frac{1-\rho}{1-\rho_j} \qquad (9)$$

where sparsity parameter constant is given by ρ , whereas ρj represents average activation value of jth neuron in the hidden layer which can be calculated using

$$\rho_j = \frac{1}{n}\sum_{i=1}^{n}[h_j(x(i), W, b)] \qquad (10)$$

where $h_j(x(i), W, b)$ represents the activation function of the jth neuron in the hidden layer of autoencoder.

The sparse penalty term actually works on the hidden layer to control the number of “active” neurons. To control the number of “active” neurons, the sparse penalty term actually works on the hidden layer. "In practice, the neuron is called “active” if the output of a neuron is close to 1, otherwise it is" inactive". Most of the time, it is easier to keep the neurons of the hidden layer “inactive”.

In this process, the average activation of each hidden neuron ρ_j is expected to be close to zero, namely the neurons of the hidden layer is mostly "inactive". To achieve this, the sparse term represented in Eq. 8 is added to the objective function in Eq. 5 that penalizes ρ_j if it deviates significantly from ρ .

This penalty function possesses the property that $KL(\rho||\rho_j) = 0$ if $\rho_j = \rho$. Otherwise, it increases monotonically as ρ_j diverges from ρ, which acts as the sparsity constraint. The cost function of the neural network (Eq. 5) can now be written as:

$$C\ (W,b) = \frac{1}{n}\sum_{i=1}^{n}\left||x_i - y_i|\right|^2 + \frac{\lambda}{2}\left(\sum_{l=1}^{l_s-1}\sum_{i=1}^{D}\sum_{j=1}^{C} W_{ij}(l)\right) \quad (11)$$

Adding the sparse penalty term to the cost function, it can be modified as:

$$C_{sparse}(W,b) = \frac{1}{n}\sum_{i=1}^{n}\left||x_i - y_i|\right|^2 + \frac{\lambda}{2}\left(\sum_{l=1}^{l_s-1}\sum_{i=1}^{D}\sum_{j=1}^{C} W_{ij}(l)\right) + \beta\sum_{j=1}^{C}\rho\, log\frac{\rho}{\rho_j} + (1-\rho)\log\frac{1-\rho}{1-\rho_j} \quad (12)$$

The optimal parameters of W and b need to be defined during the coding process. As the sparse cost function shown in equation (9) is directly related to the parameters W and b, it can be solved in order to obtain these two parameters by minimizing $C_{sparse}(W,b)$.

The back-propagation algorithm can be used to understand this, where the stochastic gradient descent method is used for training and the parameters W and b in each iteration can be modified as:

$$W_{ij} := W_{ij} - \varepsilon\frac{\partial}{\partial W_{ij}} C_{sparse}(W,b) \quad (13)$$

$$b_i := b_i - \varepsilon\frac{\partial}{\partial b_i} C_{sparse}(W,b) \quad (14)$$

where ε is the learning rate.

To compute the average activation δ_j in order to get the sparse error, a forward pass on all training examples is used, then the back-propagation algorithm works to update the parameters. After that, the SAE can learn efficient sparse feature representations.

3.4 Dropout

Dropout is a technique that, when training a neural network with small training data set [15], can help minimize 'overfitting'. Generally, the overfitting issue arises when the known training data set is insufficient, leading to poor output on the test data.

The dropout technique is applied to SAE-based DNN training in this study in order to avoid complex co-adaptations to training data and repeatedly prevent the same features from being extracted.

Technically, by setting the output of certain hidden neurons to zero, the "dropout" can be realized so that these neurons will not be active in the training phase for forward propagation.

It should be noted that certain differences exist between the training phase and the dropout testing process. During testing, the dropout is switched off, which ensures that the outputs of all hidden neurons during testing will not be masked. This will help to enhance the SAE-based DNN's feature extraction and classification capability.

4 Proposed Framework

A two-hidden-layer sparse auto-encoder with sigmoid activation functions is used in the proposed EDSA system. In the CICDDoS 2019 dataset, the input layer has 80 neurons representing the selected features (explained in section V) and the complete description of these features is presented in **Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.**.

The structure of the EDSA method is shown in Fig.2, where it is composed of three steps:

(1) Feature extraction using sparse autoencoder and normalization
(2) Deep neural network training
(3) Classification research for one of the two classes: Benign and Malicious

Step (1): The SAE is used to learn characteristics from data and initialize the DNN structure by taking advantage of unsupervised learning, as shown in Fig.2 (a). Autoencoder is a feature engineering technique focused on a neural network that can learn the hidden features of the data through an iterative training process. Autoencoder discovers the association and intermediate relationship between the individual attributes in this learning process and derives the optimal knowledge from the features thus extracting the most representative features.

The original features are first used to train the SAE, using the following steps:

I) Set up the learning rate, sparse rate and denoising parameters, dropout rate, etc., and initialize the weight W and b randomly.
II) Use the training method of stochastic batches in the forward propagation algorithm to measure the average sparsity activation ρ_j for sparsity.
III) Measure as in Eq. (12): the sparse cost function.
IV) Update the W and b parameters based on Eq. (13) and Eq. (14).

Step (2): Then, to train a neural network classifier with a dropout module for DoS attack diagnosis, the learned sparse feature representation of the auto-encoder is used as shown in Fig.2 (b). In addition, in a well-trained SAE, the parameters of the DNN classifier with the corresponding parameters are initialized and then further modified as follows:

I) To initialize the first layer of the DNN, use the parameters of the SAE.
II) Set the training parameters and the dropout rate and perform the forward propagation algorithm to extract for classification the labeled features.
III) Using Eq. (11), compute the mean square error for the cost function of the DNN.
IV) Perform the algorithm of back-propagation the same as before with the exception of the sparse term
V) To change the weights and fine-tune the entire network, except for the sparse term (set sparse penalty term to 0), perform the back-propagation algorithm the same as before.

Step (3): Finally, in order to identify network traffic as either benign (normal) or malicious (DoS attack), the test data set is used to verify the efficacy of the presented SAE-based DNN as shown in Fig.2 (c).

5 Experimental Results

The suggested method experiment is conducted by the CICIDS 2017 [17] dataset instead of the KDD cup dataset using Python [18]. While, for many years, the KDD'99 dataset has been the trusted dataset for researchers, the key reason for not using it in our work is that it is highly redundant, containing traffic from almost 20 years ago.

In addition, the total NSL-KDD (which is an improved version of KDD-CUP99) instances are 125,923 in training and 22,544 in testing, while ICIDS2017 has 2,830,108 instances and is generated on the basis of real network traffic.

CICIDS 2017 was developed over a span of 5 days within an emulated environment and includes network traffic in a packet-based and bidirectional flow-based format.

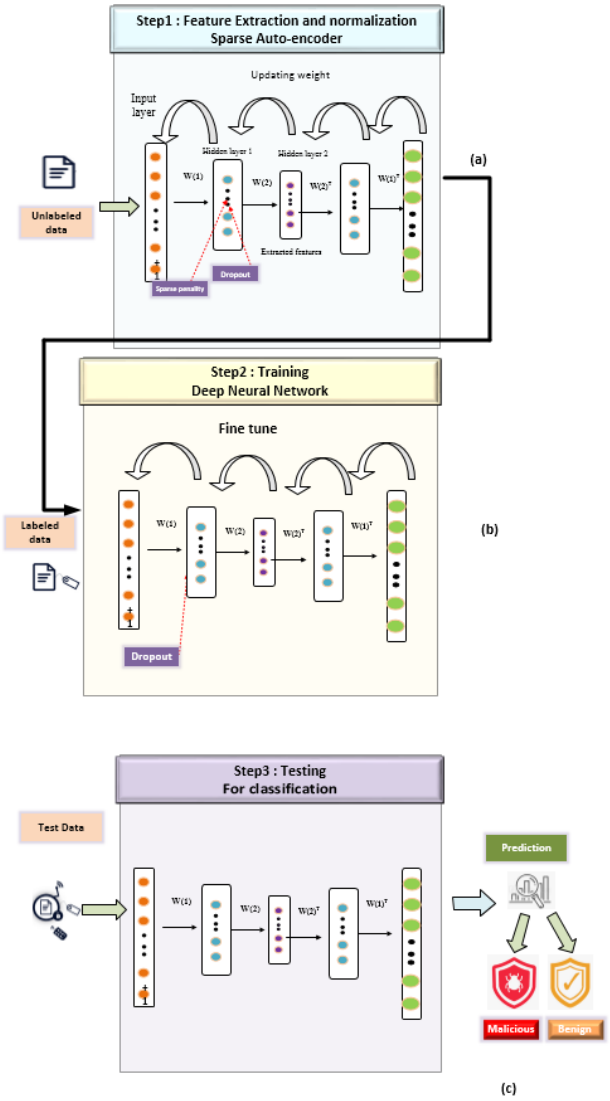

Fig. 2 Structure of the EDSA network

For our DDoS intrusion detection system we used the CICDDoS2019. The latter contains benign and the most up-to-date common DDoS attacks, which resembles the true real-world data (PCAPs). It also includes the results of the network traffic analysis using CICFlowMeter-V3 with labeled flows based on the time stamp, source, and destination IPs, source and destination ports, protocols and attack (CSV files).

The authors of [19] have extracted more than 80 features for each flow, and include additional metadata on IP addresses and attacks. The extracted traffic features are explained in [12]. A part of these features are shown in Table 1. With regard to the mentioned features except the Flow ID (1), Time Stamp (7), Flow length (8) and Label (84), the total number of features we used in our analysis comprises 80.

Table 1. Listed features of network traffic in CICDDoS2019

No.	Feature	No.	Feature
1	Flow ID	**11**	Total Length of Fwd Pck
2	Source IP	**12**	Total Length of Bwd Pck
3	Source Port	**13**	Fwd Packet Length Max
4	Destination IP	**14**	Fwd Packet Length Min
5	Destination Port	**15**	Fwd Pck Length Mean
6	Protocol	**16**	Fwd Packet Length Std
7	Time Stamp	**.**	.
8	Flow duration	**.**	.
9	Total Fwd packets	**.**	.
10	Total Backward packets	**84**	Label

In this work, the first layer of the SAE included the original features represented by 80 neurons. With a reasonable error approximation, the first hidden layer of the sparse auto-encoder was able to successfully reduce the dimensions to 68 features. Additionally, in the second secret layer, the features were reduced to 61. The resulting sparse auto encoder can be used to perform the classification in the final stage once the weights are educated.

The sparse representation parameters are set as follows: $\lambda = 0.0006$. The parameter sparsity is $\beta = 0.04$, and the term sparsity penalty is $\beta = 7$. Sparsity and penalty parameters are built to limit the activation of the hidden units, thus reducing the dependence between the features. In Table 2, these parameters are presented.

Table 2. Simulation parameters

Parameters	Value
λ	0.0006
ρ	0.04
β	7

Firstly, a comparison study was carried out to check the efficiency of our approach that uses the sparse auto-encoder to learn features for the DNN and uses the "dropout" technique to resolve overfitting during the training process, where conventional SAE is used as the basis.

The performance classification under the different sizes of the labeled training data set was examined, where the size of the training data shifted from 120 to 600, with a phase size of 120. The outcomes are shown in Fig. 3.

It can be shown that the solution proposed has shown better performance than the SAE alone. Secondly, the purpose of the approach proposed is to avoid the expense of false alarms and to refuse access to legitimate users.

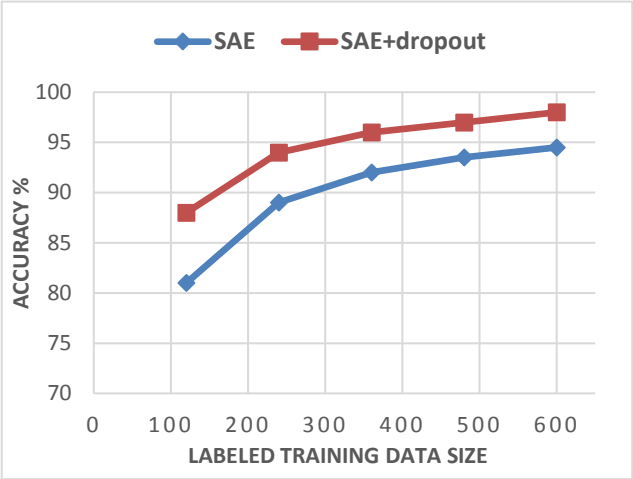

Fig. 3 Accuracy % using SAE with and without dropout

The detection accuracy of the system proposed is assessed using the following equations:

$$\text{Accuracy}: Acc = \frac{Tpos+Tneg}{Tpos+Tneg+Fpos+Fneg} \quad (13)$$

$$\text{Detection rate (DR)}: DR = \frac{T_{pos}}{T_{pos} + F_{neg}} \quad (14)$$

$$\text{Precision (PR)}: PR = \frac{T_{pos}}{T_{pos} + F_{pos}} \quad (15)$$

$$\text{Specificity (SP)}: SP = \frac{T_{neg}}{T_{neg} + F_{pos}} \quad (16)$$

The following parameters are used to test such equations:

- True positive (Tpos) = Number of samples correctly predicted as a class of attack.
- False positive (Fpos) = Number of wrongly predicted samples as the class of attack.
- True negative (Tneg) = Number of samples as a normal class correctly predicted.
- False negative (Fneg) = Number of wrongly predicted samples as a normal class

The experimental analysis of proposed EDSA was compared with a benchmark of 3 network models as presented in Table 3.

It was shown after studies that the accuracy of detection for the CICDDoS2019 dataset is 90 percent with 7.2 percent false positives for DNN, 92.9 percent with 5 percent false positives for DNN based on autoencoder with one hidden layer, 96.3 percent with 3.26 percent false positives for DNN based on autoencoder with one hidden layer and fine tuning and finally 98 percent with 1.4 percent false positives for the proposed EDSA with two-hidden layer SAE, fine tuning and cost minimization.

Table 3 Results for classification using DNN algorithm and DNN based on SAE

Algorithm	Acc (%)	DR (%)	PR (%)	SP (%)	False positive (%)
DNN	90	79.1	79.5	89	7.2
DNN based on autoencoder with one hidden layer	92.9	82.9	82.8	92	5
DNN based on autoencoder with one hidden layer and fine tuning	96.3	96.35	88.3	96.3	3.26
Proposed EDSA with two hidden layers and fine tuning	98	98.1	91	98	1.4

Fig. 4 demonstrates the normalized mean square error (NMSE) variation of the proposed EDSA method with the number of training epochs. The NMSE over time of the EDSA method is smaller than that of the DNN method. It converges after approximately 100 epochs to a value of 0.05, while the DNN approach converges to 0. In addition, the mean square error after 300 epochs is less than 0.01, which is a very satisfactory outcome. After the 350 epochs, the model converges to its steady-state location.

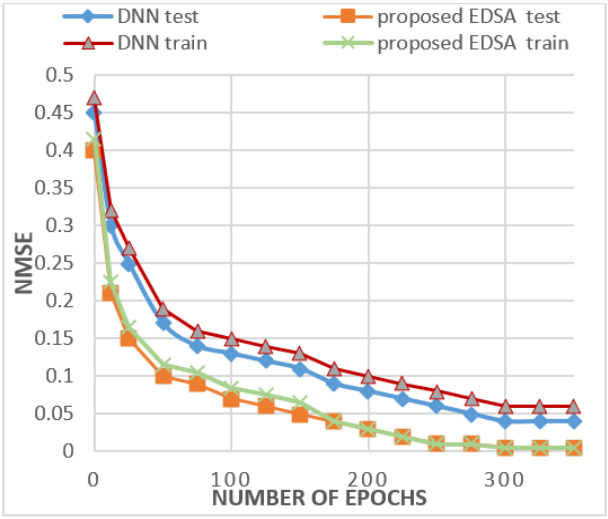

Fig. 4 NMSE of EDSA compared to DNN

6 Conclusion

For the detection of DDoS attacks, an Enhanced Deep Sparse Autoencoder-based Approach with two hidden layers is proposed in this paper. The main objective of this paper is to extract the representative features from CICDDoS2019 dataset using the autoencoder, minimize the classification error and correctly detect the DDoS attack.

The experimental analysis of proposed EDSA showed its high detection accuracy. A high percentage of enhancement is obtained in comparison with other network models for many performance indicators (accuracy, detection rate, precision, specificity) of the proposed technique. Whereas, the false positive percentage is much smaller. For the CICDDoS2019 dataset, the proposed technique obtained 98 percent of detection accuracy and 1.4 percent of false positive. In the future, it is possible to perform recent computer algorithms such as K-means clustering and introduce more layers on the level of the SAE in order to reduce more the features dimensions and apply classification algorithms other than the softmax function.

References:

[1] N. Shone, T.N. Ngoc, V.D. Phai, Q. Shi, "A deep learning approach to network intrusion

detection", IEEE Transactions on Emerging Topics in Computational Intelligence , 2018, 2, 41–50.
[2] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, J. Lloret, "Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in iot" , Sensors 2017, 17, 1967.
[3] R.K. Malaiya, D. Kwon, J. Kim, S.C. Suh, H. Kim, I. Kim, "An Empirical Evaluation of Deep Learning for Network Anomaly Detection", in Proceedings of the 2018 International Conference on Computing, Networking and Communications (ICNC), Maui, HI, USA, 5–8 , 2018; pp. 893–898.
[4] C. Li, J. Wang, X. Ye, "Using a Recurrent Neural Network and Restricted Boltzmann Machines for Malicious Traffic Detection", NeuroQuantology, 2018, 16.
[5] K. Saravanan, "Neuro-fuzzy-based clustering of DDoS attack detection", International Journal of Critical Infrastructure Protection, 2017, 13:46–56
[6] K. Grzegorczyk, M. Kurdziel, P. Wojcik , "Encouraging orthogonality between weight vectors in pretrained deep neural networks", Neurocomputing , 2016, 202:84–90
[7] V.L. Thing , "IEEE 802.11 network anomaly detection and attack classification: A deep learning approach", Proceedings of IEEE Wireless Communications and Networking Conference (WCNC),San Francisco, CA, USA, 19–22 , 2017; pp. 1–6.
[8] T. Ma, F. Wang, J. Cheng, Y. Yu, X. Chen, "A hybrid spectral clustering and deep neural network ensemble algorithm for intrusion detection in sensor networks", Sensors 2016, 16, 1701.
[9] T. Tang, S.A.R. Zaidi, D. McLernon, L. Mhamdi, M. Ghogho, "Deep Recurrent Neural Network for Intrusion Detection in SDN-based Networks", Proceedings of 4th IEEE Conference on Network Softwarization and Workshops (NetSoft), Montreal, QC, Canada, 2018.
[10] A.H. Muna, N. Moustafa, E. Sitnikova, "Identification of malicious activities in industrial internet of things based on deep learning models", Journal of Information Security and Applications, 2018, 41, 1–11.
[11] A. M. Cheriyadat, "Unsupervised feature learning for aerial scene classification", IEEE Trans. Geosci. Remote Sens. 52 (2014) 439-451.
[12] CIC. Canadian Institute of Cybersecurity. List of Extracted Traffic Features by CICFlowMeter-V3. 2017. Available online: https://www.unb.ca/cic/datasets/ids-2017.html (accessed on 23 January 2019).
[13] S. Kullback, R. A. Leibler, "On information and sufficiency", Annals of Mathematical Statistics 22, 1951, pp: 79-86.
[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", arXiv preprint arXiv:1207.0580, 2012.
[15] A. Coates, A. Y. Ng, H. Lee, "An analysis of single-layer networks in unsupervised feature learning", Jour. Mach. Learn. Res. 15, 2011 ,215-223.
[16] M. Tavallaee, E. Bagheri, W. Lu, A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set", IEEE symposium on computational intelligence for security and defense applications, 2009, pp 1-6
[17] I. Sharafaldin, A. H. Lashkari, A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization", International Conference on Information Systems Security and Privacy (ICISSP), 2018, pp. 108–116. doi:10.5220/ 0006639801080116.
[18] P. S. Foundation. Python Language Reference, Version 3.6. https://www. python.org/. 2016.
[19] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A.A. Ghorbani, "Developing Realistic Distributed Denial of Service (DDoS) Attack Dataset and Taxonomy", IEEE 53rd International Carnahan Conference on Security Technology, Chennai, India, 2019

Author Contributions: Please, indicate the role and the contribution of each author:

Samer sindian, carried out the simulation and the optimization.

Samar sindian, developed the model and wrote the paper.