In Search of Distance Functions That Improve Autoencoder Performance for Intrusion Detection

Rémi Bouchayer
ECE Paris
Institut Polytechnique de Paris
Télécom SudParis
Nexter Systems
Paris, France
0000-0002-0523-8873

Jae-Yun Jun
ECE Paris
Paris, France
jaeyunjk@gmail.com

Hakima Chaouchi
Institut Polytechnique de Paris
Télécom SudParis
Palaiseau, France
hakima.chaouchi@telecom-sudparis.eu

Philippe Millet
Nexter Systems
Versailles, France
p.millet@nexter-group.fr

***Abstract*—Expectations of detection systems have risen with the increase in cyber-attacks. In order to detect the latest and future attacks, systems capable of detecting unknown attacks are needed. Among the various approaches offered by machine learning models, anomaly detection methods can address this need. It is possible to use the autoencoder to detect anomalies and therefore attacks. An autoencoder trained on data from normal use is able to detect attacks, unknown to the model. The attack detection is possible by observing the reconstruction error, which is the distance between the input and the reconstructed input resulting from the model. We considered different distance functions to improve the separation between attacks and normal events, and thus, to improve the performance of the autoencoder. We propose to use the cosine function of the angle formed between the actual input vector and the reconstructed input vector, as a distance function to address the problem of overlapping between normal events and attacks. In addition, we used Tree-structured Parzen Estimator algorithm for the optimization of the hyperparameters of the model. We ran our method on the NSL-KDD dataset and compared the obtained results to those of other methods that exist in the literature.**

***Index Terms*—Information security, intrusion detection, anomaly detection, deep learning, autoencoder**

I. Introduction

Computer systems are vulnerable [1]. This idea is captured in the quote "it's not a question of if, but when", indicating that any system can be compromised. Vulnerabilities in these systems are being exploited by an increasing number of threat actors. There are several approaches to protecting these systems, this work focuses on intrusion detection (ID). The purpose of ID is to detect a compromise as early as possible so that incident response actions can be quickly initiated to limit the severity of the incident.

Securing systems is an ongoing challenge. Attackers try to break into systems, which are protected by defenders. Attackers look for new vulnerabilities and procedures to exploit them. When defenders are made aware of a vulnerability, they may implement patch or detection measures. In turn, attackers look for new vulnerabilities and exploit techniques that fly under the radar of defenders' tools. In this context, intrusion detection system (IDS) are deployed and updated to keep up with the latest attacks.

Keeping an IDS up and running it is expensive. To be effective, the system must detect the latest attacks and even unknown attacks. Developing systems to detect unknown attacks is an active research problem [2] [3]. Recently, various machine learning (ML) algorithms have been studied to solve this problem, especially using autoencoder (AE) [4], a deep learning (DL) method [5]. AE can be trained with normal data only and is relatively simple compared to other more complex DL models.

We formulate our research question as how to further improve the detection of unknown attacks by using AE (**RQ**). Three main problems arise from our research question. The first is choosing a detection approach (**PB1**). There is no standard approach to detecting unknown attacks. Different pattern recognition or anomaly detection methods can be used. We have chosen to use AE to perform anomaly detection. Accordingly, we used only normal data to train our model. In order to detect attacks, the model should be able to differentiate normal events from attacks. The classification is done according to the reconstruction error (RE) of the input passed to the model. The second problem is to define a threshold for classification based on the RE (**PB2**). Events whose RE is below the threshold are classified as normal, and those whose RE is above the threshold are classified as attacks. If the model were perfect, there would be a threshold that would separate the two classes without classification error, but this is not the case. The RE distributions of normal events and attacks overlap. Limiting this overlap is the third problem (**PB3**). To reduce this overlap and improve the accuracy of our model, we studied two different distance functions : the Euclidean and cosine distance between the actual input vector and the reconstructed input vector.

This paper is structured as follows. A theoretical background is provided in Section II, the notation used is presented in the Section III and related works are described in Section IV. Our proposed method is described in Section V. The results achieved from this proposed method are presented in Section VI and discussed in Section VII. Finally, we provide concluding remarks and some possible future works in Section VIII.

II. Theoretical framework

In this section, we present an overview and comparison of ID approaches to provide a framework for this study. There are two main approaches to ID: misuse and anomaly detection.

A. Misuse Detection

In order to detect traces of attacks, misuse detection uses pattern-mining methods. The detection system uses data that characterize attacks. These data are called rules or signatures. Identifying these characteristics within the elements analyzed by the detection system allows attacks to be detected. The performance of these systems is highly dependent on the degree of specificity of the signatures. A very specific signature will produce low false positives. In the case of a low specificity signature, unknown attacks may be detected as well as many false positives. To detect the latest attacks, signatures must be continually updated to include variants and new attacks.

B. Anomaly Detection

Anomaly detection is an independent discipline rather than an ID-specific science. The necessary assumption for anomaly detection methods to work is that anomalies are significantly different from normal data. By "significant" we mean that it is possible to characterize this difference. When applying anomaly detection methods to ID, the following assumption is made : the anomalous data represent attacks. The detection system uses normal data to establish a norm. Once the norm is established, new events are compared to the norm. The deviation can be used to define a probability that the event is an attack. It is possible to obtain a binary output by classifying events as normal or attack based on a defined threshold. In practice, this approach generates many false positives. This is because not only attacks deviate from the norm. A failure or even an unusual normal event can also trigger an alert. Since the generated alerts only indicate a deviation from the norm, it may be necessary to analyze them in order to obtain security alerts. Furthermore, the hypothesis that attacks are different from normal events is not always verified. Thus, some attacks that are too close to normal events may go undetected, and therefore a special attention needs to be paid for them.

C. Issues

Misuse detection and anomaly detection have different strengths and weaknesses. Therefore, a third approach, called the hybrid approach, is recommended [6]. By combining the two approaches, the hybrid system may be more efficient, combining normal data and known attack knowledge.

III. Notation

In the sequel, we refer to the following notation. $X \in \mathbb{R}^{I \times N}$ represents the input matrix, and $Y \in \mathbb{R}^{I \times 1}$ represents the labels. The total number of events is represented by I, and each event $x^{(i)}$ have N features. The value of a label $y^{(i)}$ is 0 for a normal event, and 1 for an abnormal one.

$$X = \begin{bmatrix} x_1^{(1)} & \cdots & x_N^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(I)} & \cdots & x_N^{(I)} \end{bmatrix}, \quad Y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(I)} \end{bmatrix} \tag{1}$$

IV. Related work

Intrusion detection problem is typically addressed with models having many parameters. A large number of events are recorded on the hosts every second, and, for each event, the number and size of the fields vary depending on the data source. The use of ML methods to generate new signatures or to define a norm can be beneficial. The data that can be used to train a ML model vary depending on the attack detection approach. In the case of misuse detection, the model can be trained on both normal and abnormal data (X, Y). In the case of anomaly detection, the model is only trained on normal data $(X, Y = 0)$.

Problems similar to those of ID can be found in fields like knowledge discovery, data mining and anomaly detection. Some methods of these fields have been used as shown in the survey [7]. Using deep learning methods is also studied in [8].

A. Autoencoder

Kramer presented an autoencoder as a nonlinear principal component analysis model [4], and the use of AE for ID has often been studied [5] [9]–[11]. As an AE is trained without using Y, it is considered as an unsupervised learning algorithm. The RE can be calculated as the distance between the input and the reconstructed input. During the training, the model is optimized to minimize the RE for normal events. As attacks may behave differently from normal events, their RE can be expected to be higher than for normal events. Thus, it is possible to use a threshold to discriminate between the normal and attack classes based on the RE.

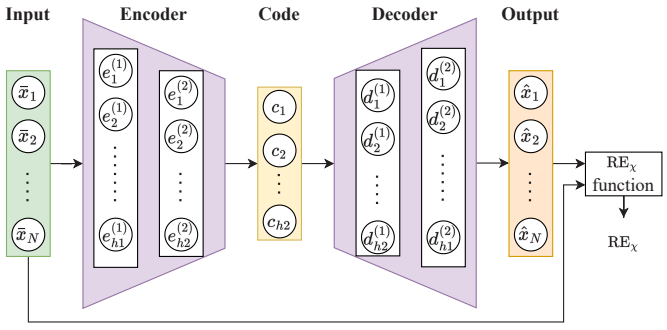


Fig. 1. Autoencoder architecture.

An AE is composed of two symmetric neural networks, presented in Fig. 1. The first network is called the encoder. It encodes the input vector $\bar{x}$ into a latent vector c. The vector $\bar{x}$ is the preprocessed output using x in input. The dimension of c $(h2)$ is smaller than the one of $\bar{x}$, which forces the AE to extract the main components of $\bar{x}$. The latent vector c is

then processed by the second neural network, the decoder, to reconstruct the input. The reconstructed vector is $\hat{x}$.

The RE between $\bar{x}$ and $\hat{x}$ can be measured with different loss functions [10]. Model parameters are adjusted during training to minimize RE for normal data. The majority of normal test data are reconstructed with low RE, although for some unusual normal events, the RE can be large. However, attacks may also be reconstructed with a low RE. The characteristics of these attacks are usually close to normal events, and therefore, in some sense, they are considered as to be known by the model. This overlap between the RE distribution of normal events and attacks makes the separation difficult (PB3). To solve this problem, [12] suggests that adding a memory to the AE can increase the RE for attacks. For this same limitation of the AE, [13] suggests adding a different module (an isolation forest) after the AE.

B. Datasets

In the literature, we can observe that different datasets are used for ID problems. The oldest major dataset we can find is the DARPA Intrusion Detection datasets [14]. There exist three versions: 1998, 1999 and 2000. In these datasets, a first attack database was proposed [15]. The next major dataset is KDD-99, which was used for the Third International Knowledge Discovery and Data Mining Tools Competition. The KDD-99 dataset has been criticized because of the redundancy of the records [16]. To resolve some of the statistical issues, the NSL-KDD dataset was published [16]. Later, the Canadian Institute for Cybersecurity published numerous datasets like ISCX IDS dataset 2012 [17], and CIC-IDS2017 [18]. The Cyber Range Lab of UNSW Canberra published a new network dataset UNSW-NB15 [19]. NSL-KDD, CIC-IDS2017 and UNSW-NB15 are the most used datasets for ID today.

V. Method

In this section we present how we trained and evaluated different AEs on the the NSL-KDD dataset, by varying the distance function and hyperparameters. Only the normal data of the training dataset "KDDTrain+" are used to train the AE. These data are first encoded and normalized. We investigated Euclidean and cosine distance functions to compute the RE (PB3). AE parameters are optimized to minimize the training RE, and hyperparameters are optimized to maximize the F1-score. We defined a threshold from training RE (PB2). Finally, the data from the test dataset "KDDTest+" are classified using the previously defined threshold.

A. Environment

The experiments carried out for this work were executed on the Lambda Quad Max Deep Learning server, equipped with an Intel I9 CPU, 256GB of RAM and 4 Nvidia Titan V GPU cards. The proposed models have been implemented in Python (version 3.8.0) with the PyTorch library (version 1.13.1).

TABLE I
NSL-KDD features

Index	Feature name	Index	Feature name
1	duration	22	is_guest_login
2	protocol_type	23	count
3	service	24	srv_count
4	flag	25	serror_rate
5	src_bytes	26	srv_serror_rate
6	dst_bytes	27	rerror_rate
7	land	28	srv_rerror_rate
8	wrong_fragment	29	same_srv_rate
9	urgent	30	diff_srv_rate
10	hot	31	srv_diff_host_rate
11	num_failed_logins	32	dst_host_count
12	logged_in	33	dst_host_srv_count
13	num_compromised	34	dst_host_same_srv_rate
14	root_shell	35	dst_host_diff_srv_rate
15	su_attempted	36	dst_host_same_src_port_rate
16	num_root	37	dst_host_srv_diff_host_rate
17	num_file_creations	38	dst_host_serror_rate
18	num_shells	39	dst_host_srv_serror_rate
19	num_access_files	10	dst_host_rerror_rate
20	num_outbound_cmds	41	dst_host_srv_rerror_rate
21	is_host_login		

B. Dataset

The NSL-KDD dataset is used to evaluate the proposed AE model. This dataset is used by many studies as a reference and is small enough to be trained on reasonable hardware, like a laptop CPU. The dataset contains a total of 125 973 records for training and 22 544 records for testing. Each record has 41 features presented in Table I and is labeled with one of the 41 classes. Out of these 41 classes, the normal class represents normal events, and the other 40 represent different specific types of attacks. The approach used in this work allows us to distinguish attacks from normal events but not to classify the type of attack. In the rest of this work, we refer to the normal class, and the attack class, the latter including all the attacks.

C. Pre-processing

The preprocessing phase consists of two stages: encoding and normalizing. To enhance the performance of the model, all textual data are numerically encoded. This is done by associating a different number to each label. After encoding the records, a normalization function is applied. The normalization method is done with feature scaling using min-max method.

D. Training the autoencoder

The training objective is to find the AE parameters that minimize the loss (RE) for the normal data. The two networks of the AE, the encoder and decoder are trained at the same time. The activation function used is the hyperbolic tangent. The encoder is composed of two layers, the second being smaller than the first. The decoder is also composed of two layers of symmetrical size to those of the encoder, the first layer being smaller than the last. Hereafter, the architecture is indicated as *h1-h2*, with *h1* being the size of the first layer of the encoder and the last of the decoder and *h2*, the size of the last layer of the encoder and the first of the decoder.

The parameters of the model are optimized using Adam algorithm. Regularization is applied with the weight decay (WD) hyperparameter for "AdamW" supported by PyTorch [20]. The loss is computed with different distances functions, which are presented afterwards. The loss for each batch is summed and then averaged. The model is trained while the epoch loss has improved by at least 1e-06 over the last *patience* number of epochs. To compute the loss in (2), we compared two distance functions RE_χ, where $\chi = \{cosine, euclidean\}$.

$$\text{Loss} = \frac{1}{I}\sum_{i=1}^{I}(\text{RE}_\chi(\bar{x}^{(i)}, \hat{x}^{(i)})), \quad \text{where } \bar{x}^{(i)}, \hat{x}^{(i)} \in \mathbb{R}^{N\times 1}, \tag{2}$$

$$\text{RE}_{euclidean}(\bar{x}^{(i)}, \hat{x}^{(i)}) = \|\bar{x}^{(i)} - \hat{x}^{(i)}\|_2^2, \tag{3}$$

$$\text{RE}_{cosine}(\bar{x}^{(i)}, \hat{x}^{(i)}) = 1 - \frac{(\bar{x}^{(i)})^T \cdot \hat{x}^{(i)}}{\max(\|\bar{x}^{(i)}\|_2 \cdot \|\hat{x}^{(i)}\|_2, \epsilon)}. \tag{4}$$

E. Threshold definition

Ideally, it is possible to determine a threshold that allows normal events and attacks to be classified without any classification error. However, the RE obtained for normal events can overlap with the RE obtained for attacks (PB3). Thus, the choice of the threshold must be made as best as possible but will often lead to classification errors (PB2). We computed the threshold as a quantile of the training RE.

F. Hyperparameters optimization

To maximize the performance of our model, we executed an algorithmic search for the best parameters. We have used the implementation of the Tree-structured Parzen Estimator (TPE) algorithm [21] proposed by the Optuna library [22]. Hyperparameters are optimized to maximize the F1-score. The hyperparameters to be optimized are presented in Table II.

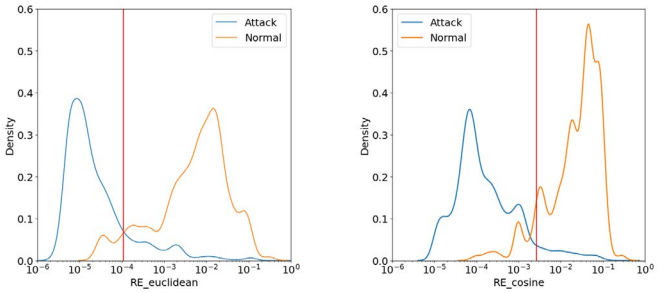
G. Model evaluation

To evaluate the proposed model (i.e., the classification phase) the "KDDTest+" dataset was used. It contains 22 544 records distributed between 12 833 attacks and 9 711 normal records. We observed the presence of 610 duplicates between the training and testing dataset. The number of duplicated normal records is 87. This represents 0.13% of normal records used in training and 0.90% of normal records used in testing. Due to the small percentage of duplicates, and in order to compare with the results obtained on the standard NSL-KDD dataset, we did not remove the duplicates.

To evaluate the performance of our approach we used accuracy, precision, recall, F1-score and the Matthews Correlation Coefficient (MCC) [23]. The accuracy metric is not

TABLE II
HYPERPARAMETERS TO OPTIMIZE.

Hyperparameter	h1	h2	LR	WD	patience	Batch size
Min	4	2	1e-05	1e-05	1	2^5
Max	128	40	1e-03	1e-03	50	2^8



(a) Euclidean distance. (b) Cosine distance.

Fig. 2. Kernel Density Estimation of RE for attack and normal class.

significant of the performance when classes are imbalanced. As in general, ID is an imbalanced problem, using metrics like MCC are encouraged. Obtaining normal records is easier than obtaining attack records, in particular for unknown attacks.

VI. Results

The best hyperparameters for each distance function have been searched with the TPE algorithm. The objective of the hyperparameters optimizer was to maximize the average F1-score over 10 runs. The three best combination of hyperparameters for each distance function are presented in Table III. The mean training time using the Euclidean and cosine distance are 22 and 19 minutes, respectively. The average MCC obtained with the best hyperparameters for the Euclidean and cosine distance are 0.7909 and 0.8301, respectively.

TABLE III
BEST HYPERPARAMETERS FOR DIFFERENT DISTANCE FUNCTIONS

h1	**h2**	**LR**	**WD**	**Patience**	**Batch size**	**Quantile**	**F1-score**
Euclidean distance							
64	34	1.2e-04	7e-05	7	256	0.83	**0.9129**
99	37	6e-05	1.5e-04	16	256	0.83	0.9128
88	40	9e-05	1.7e-07	13	256	0.85	0.9122
Cosine distance							
68	8	6e-05	7.6e-04	30	256	0.87	**0.92490**
71	8	6e-05	6.6e-04	20	256	0.87	0.92420
66	7	1.1e-04	6e-04	6	128	0.86	0.92417

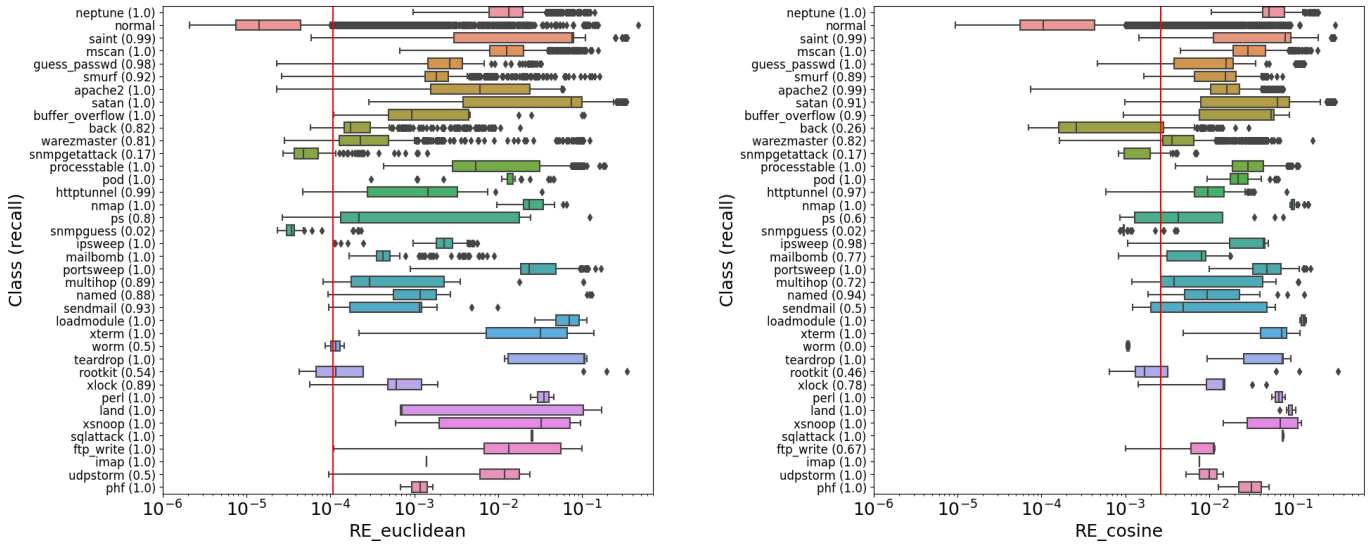
VII. Discussion

A. Intrusion detection approach (PB1)

We followed an anomaly detection approach using the RE to detect attacks. The classification performed is binary, but the attacks are defined in fact with their classes. The results show that the model does not perform well for some classes, like *back*, as shown in Fig. 3b. To efficiently improve the detection performance of the system, AE can be complemented by an other IDS which handles the problematic classes.

B. Training the autoencoder

AEs presented in the literature are often trained over a small number of epochs: 20 [24], 50 [10] and 100 [9] [12]. Limiting



(a) Euclidean distance.

(b) Cosine distance.

Fig. 3. Boxplot of RE_χ distribution per specific attack types.

the number of epochs is often done to prevent overfitting or overgeneralization [25]. However, in our experiments, the trend curve of the accuracy and training error suggested a possible improvement beyond 100 epochs. We therefore trained our models for longer (5000 and 10 000 epochs). We observed a gain, but it was small compared to the increase in training time. Therefore, we decided to use a stopping condition based on the evolution of the training error. The definition of the stopping condition to obtain optimal performance remains a problem to be studied.

C. Threshold definition (PB2)

The classification of events between normal and attack is based on an error threshold. This threshold should not be set with the sole aim of maximizing a performance criterion such as the F1-score. A false positive does not have the same cost as a false negative. In the former case, an alert is raised by mistake. This behavior can lead to fatigue in the teams responsible for handling alerts. In the second case, an attack is not detected, which can have severe consequences. Several criteria should be taken into account when defining the strategy for setting the threshold. A strategy can take into account the number of events to be analyzed, the number of processed alerts, the size of the team processing the alerts and the criticality of the system to protect.

D. Limit overlapping (PB3)

The choice of the distance function leads to different overlapping ranges of reconstruction errors for the two classes. The accuracy metric identifies the distance function that best separates the distributions. The distribution of the RE for each class in the dataset can be seen in Fig. 3a and Fig. 3b along with the appropriate classification threshold (the vertical red line). In Fig. 2a and Fig. 2b, we can see that the overlap of the two distributions is lower for the cosine distance function compared to the Euclidean distance. In Fig. 3b, it can be seen that most types of attacks have an RE far from the normal distribution. While for some types of attacks, the overlap is greater with the Euclidean distance function. For example, the RE for the *snmpguess* class is confused with the distribution of RE for *normal* class in Fig. 3a. Despite local improvements, none of the distance functions presented in this work reliably separates all attacks from normal events.

E. Impact of the distance function

The choice of the distance function has an impact on the model performance and efficiency. The cosine distance function performs better on the metrics presented in the Table III. Using the Euclidean distance takes about 10% more time during training, and it requires a higher number of neurons, increasing the number of model parameters. Overall, the use of the cosine distance seems to be the better choice.

F. Comparison with previous works in the literature

Contributions can be divided into three levels: preprocessing, model and postprocessing. As our contribution is at the model level, pre- or post-processing methods could also be applied to our model in order to obtain better performance. We therefore do not compare ourselves with the results obtained in [10], [13]. We compare our model to other AEs, presented in [12], [13] and [26], in Table IV. The use of the cosine distance provides high performance, which may be further improved with other advanced methods (pre- and/or post-processing).

VIII. Conclusion and Future Work

The detection of unknown attacks can be addressed with anomaly detection methods. Autoencoder is a DL model that can be considered for this purpose, requiring only normal data

TABLE IV
COMPARISON WITH SIMILAR WORKS IN THE LITERATURE

Model	Accuracy	Precision	Recall	F1-score
Contribution : pre-processing				
AE + outlier [10]	0.9061	0.8683	0.9843	0.9226
Contribution : model				
AE + memory [12]	0,8951	0,9062	0,8951	0,8993
AE [13]	0,8898	0,8792	0,9348	0,9061
AE [26]	0,9170	0,9768	0,8468	0,9071
AE + $RE_{euclidean}$	0.8973	0.8819	0.9462	0.9129
AE + RE_{cosine}	0.9159	0.9410	0.9093	0.9249
Contribution : post-processing				
AE + IF [13]	0,954	0,9481	0,9725	0,9601

to be trained. We compared different distance functions to improve the separation between normal events and attacks. To compare the best performance for each distance function, we used TPE to optimize the hyperparameters. Our results show that the cosine distance between the input and the reconstructed input provides the maximal performance. The analysis of the distribution of reconstruction errors as a function of attack classes allows us to identify problematic attacks types where future improvements should be made. In order to further improve the performance of the autoencoder, we suggest different directions. Reducing the overlapping between normal events and attacks remains a major problem. Finding a better loss function may be beneficial. The selection of the data used in training, as done by [10], is another axis of improvement. Combining the use of a new loss function with data selection may lead to better performance. As the performance of the autoencoder is not the same for all attack classes, developing a hybrid approach could lead to an improved solution. This may be done by using the autoencoder followed by a dedicated model for difficult classes for the autoencoder.

ACKNOWLEDGMENT

The authors are grateful to Assia Soukane and Olivier Chesnais who made this collaboration possible. The authors also thank Phong Cao, Erwan Gautron and Elodie Plassa for the discussions. The authors appreciate the ECE for financing the purchase of the Lambda Quad Max Deep Learning server, which is employed to obtain the results illustrated in the present work.

REFERENCES

[1] D. Denning, "An Intrusion-Detection Model," *IEEE Transactions on Software Engineering*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987.

[2] M. Almansor and K. B. Gan, "Intrusion Detection Systems: Principles And Perspectives," *Journal of Multidisciplinary Engineering Science Studies*, pp. 2458–925, Nov. 2018.

[3] L. N. Tidjon, "Formal modeling of intrusion detection systems," PhD Thesis, Institut Polytechnique de Paris ; Université de Sherbrooke (Québec, Canada), Nov. 2020.

[4] M. A. Kramer, "Nonlinear Principal Component Analysis Using Autoassociative Neural Networks," *AIChE Journal*, vol. 37, no. 2, 1991.

[5] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "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)*. New York City, United States: ACM, 2016.

[6] H. Hanafi, A. Pranolo, Y. Mao, T. Hariguna, L. Hernandez, and N. F. Kurniawan, "IDSX-Attention: Intrusion detection system (IDS) based hybrid MADE-SDAE and LSTM-Attention mechanism," *International Journal of Advances in Intelligent Informatics*, vol. 9, pp. 121–135, 2023.

[7] A. L. Buczak and E. Guven, "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection," *IEEE Commun. Surv. Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[8] D. Kwon, H. Kim, J. Kim, S. C. Suh, I. Kim, and K. J. Kim, "A survey of deep learning-based network anomaly detection," *Cluster Comput*, vol. 22, no. S1, pp. 949–961, Jan. 2019.

[9] C. Ieracitano, A. Adeel, F. C. Morabito, and A. Hussain, "A novel statistical analysis and autoencoder driven intelligent intrusion detection approach," *Neurocomputing*, vol. 387, pp. 51–62, Apr. 2020.

[10] W. Xu, J. Jang-Jaccard, A. Singh, Y. Wei, and F. Sabrina, "Improving Performance of Autoencoder-Based Network Anomaly Detection on NSL-KDD Dataset," *IEEE Access*, vol. 9, pp. 140 136–140 146, 2021.

[11] R. C. Aygun and A. G. Yavuz, "Network Anomaly Detection with Stochastically Improved Autoencoder Based Models," in *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, Jun. 2017, pp. 193–198.

[12] B. Min, J. Yoo, S. Kim, D. Shin, and D. Shin, "Network Anomaly Detection Using Memory-Augmented Deep Autoencoder," *IEEE Access*, vol. 9, pp. 104 695–104 706, 2021.

[13] K. Sadaf and J. Sultana, "Intrusion Detection Based on Autoencoder and Isolation Forest in Fog Computing," *IEEE Access*, vol. 8, pp. 167 059–167 068, 2020.

[14] DARPA, "MIT Lincoln Laboratory: DARPA Intrusion Detection Evaluation," 1999.

[15] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," Ph.D. dissertation, Massachusetts Institute of Technology, 1999.

[16] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the KDD CUP 99 data set," in *IEEE Symposium on Computational Intelligence for Security and Defense Applications*, Jul. 2009, pp. 1–6, iSSN: 2329-6275.

[17] 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, May 2012.

[18] I. Sharafaldin, A. Habibi Lashkari, and A. A. Ghorbani, "Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization:," in *Proceedings of the 4th International Conference on Information Systems Security and Privacy*. Funchal, Madeira, Portugal: SCITEPRESS - Science and Technology Publications, 2018, pp. 108–116.

[19] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," in *2015 Military Communications and Information Systems Conference (MilCIS)*, Nov. 2015, pp. 1–6.

[20] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," Jan. 2019, arXiv:1711.05101 [cs, math].

[21] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for Hyper-Parameter Optimization," *Advances in neural information processing systems*, 2011.

[22] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," Jul. 2019, arXiv:1907.10902 [cs, stat].

[23] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442–451, Oct. 1975.

[24] K. Narayana Rao, K. Venkata Rao, and P. R. P.V.G.D., "A hybrid Intrusion Detection System based on Sparse autoencoder and Deep Neural Network," *Computer Communications*, vol. 180, pp. 77–88, 2021.

[25] G. Spigler, "Denoising Autoencoders for Overgeneralization in Neural Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 998–1004, Apr. 2020, arXiv:1709.04762 [cs].

[26] H. Choi, M. Kim, G. Lee, and W. Kim, "Unsupervised learning approach for network intrusion detection system using autoencoders," *J Supercomput*, vol. 75, no. 9, pp. 5597–5621, Sep. 2019.