Computer Networks 168 (2020) 107042

Contents lists available at ScienceDirect

Computer Networks

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

Evolving deep learning architectures for network intrusion detection using a double PSO metaheuristic

Wisam Elmasry[a], Akhan Akbulut[b,*], Abdul Halim Zaim[a]

[a] Department of Computer Engineering, Istanbul Commerce University, Istanbul, 34840, Turkey
[b] Department of Computer Engineering, Istanbul Kültür University, Istanbul, 34158, Turkey

ARTICLE INFO

Article history:
Received 21 June 2019
Revised 24 October 2019
Accepted 1 December 2019
Available online 10 December 2019

Keywords:
Cyber security
Deep learning
Feature selection
Hyperparameter selection
Network intrusion detection
Particle swarm optimization

ABSTRACT

The prevention of intrusion is deemed to be a cornerstone of network security. Although excessive work has been introduced on network intrusion detection in the last decade, finding an Intrusion Detection Systems (IDS) with potent intrusion detection mechanism is still highly desirable. One of the leading causes of the high number of false alarms and a low detection rate is the existence of redundant and irrelevant features of the datasets, which are used to train the IDSs. To cope with this problem, we proposed a double Particle Swarm Optimization (PSO)-based algorithm to select both feature subset and hyperparameters in one process. The aforementioned algorithm is exploited in the pre-training phase for selecting the optimized features and model's hyperparameters automatically. In order to investigate the performance differences, we utilized three deep learning models, namely, Deep Neural Networks (DNN), Long Short-Term Memory Recurrent Neural Networks (LSTM-RNN), and Deep Belief Networks (DBN). Furthermore, we used two common IDS datasets in our experiments to validate our approach and show the effectiveness of the developed models. Moreover, many evaluation metrics are used for both binary and multiclass classifications to assess the model's performance in each of the datasets. Finally, intensive quantitative, Friedman test, and ranking methods analyses of our results are provided at the end of this paper. Experimental results show a significant improvement in network intrusion detection when using our approach by increasing Detection Rate (DR) by 4% to 6% and reducing False Alarm Rate (FAR) by 1% to 5% from the corresponding values of same models without pre-training on the same dataset.

1. Introduction

In today's world, we are facing a big data era, where the Internet of Thing (IoT) devices are embedded, connected, and produce a big volume of data. Hence, they mount up challenges to the security in both academia and industry. As a result, a variety of malware variants and threats are newly emerging at a faster pace, but we cannot deal with them within due golden time with existing approaches [1].

In the open literature, the network intrusion can happen when an intruder launches one or more of potential attacks by utilizing system vulnerabilities to gain unauthorized access to user's information or to make the system down. Undeniably, there are many attacks can be initiated in computer networking such like Brute Force, Port Scanning, Denial of Service (DoS), Remote to Local (R2L), Probing (Probe), User to Root (U2R), etc. Notably, these attacks can be executed along with any application, transport, and network's protocols such as HTTP, TCP, SMTP, UDP, FTP, ICMP, etc. In order to cope with such serious threats, it is recommended to employ a Network-based Intrusion Detection System (NIDS). In general NIDS is responsible for monitoring the entire network infrastructure and detecting any malicious activities [2].

In network security, there are two common detection methods to NIDSs: signature-based detection and anomaly-based detection. Signature-based detection (or also known as misuse detection) is useful to use when only the attack signature (pattern) is known. In contrast, anomaly-based detection can be used for either known or unknown attacks. Moreover, NIDSs rely on the concept of "traffic identification", that is, extracting the useful features from the captured traffic flow, and then classifying the traffic record to either normal or attack by using one of previously trained machine learning algorithm [3].

Nowadays, due to the power of computing machines, a big advance, particularly in the Artificial Intelligence (AI) area, is occurred. Advanced technologies of machine learning, particularly deep learning, are being applied in the security area, and new

* Corresponding author.
E-mail addresses: wisam.elmasry@istanbulticaret.edu.tr (W. Elmasry), a.akbulut@iku.edu.tr (A. Akbulut), azaim@ticaret.edu.tr (A.H. Zaim).

https://doi.org/10.1016/j.comnet.2019.107042
1389-1286/

results and issues have been reported [4]. However, with deep learning, we can significantly increase the accuracy and robustness in the detection of attacks as well as operate detection systems without requiring deep security expert knowledge as before Du et al. [5].

The aims and contributions of this research are four-fold, as follows:

- We utilized a metaheuristic for selection of features and hyperparameters by employing a double PSO-based algorithm.
- We performed a comprehensive empirical study on network intrusion detection to investigate the effectiveness of three deep learning models with pre-training phase by leveraging a double PSO-based algorithm. Our approach enhanced deep learning models' detection rate by 4% to 6% as well as decreasing false alarm rate by %1 to 5% from the corresponding values of deep learning models without pre-training phase.
- We validated our approach by using NSL-KDD and CICIDS2017 datasets for both binary and multiclass classification tasks.
- We included three comparative analyses and compared our findings to the best results in the literature. In addition to that, we used various evaluation metrics in order to give further analysis and complete view of deep learning models' performance when using our approach.

The rest of this paper is organized as follows. A summary of literature review is introduced in Section 2. Section 3 presents the proposed double PSO-based algorithm. Then, Section 4 explains the methodology of our experiments and which models are used. In Section 5, we present a list of the used evaluation metrics and their formulas for both binary and multiclass classifications. Afterwards, Section 6 describes the used IDS datasets and their characteristics in detail. The experimental results are analyzed in Section 7. Finally, we draw conclusions in Section 8.

2. Literature review

Notably, dozens of previous work has been intensively researched on using deep learning in network intrusion detection in the last decade. Indeed, some of these articles have been used feature selection prior to intrusion detection. Whereas, the most previous works have been explored network intrusion detection on the full feature set of the dataset. To start with studies with feature selection, Tang et al. introduced a DNN model for network intrusion detection in software defined networking [6]. It trained on six selected features from NSL-KDD dataset and achieved a detection rate equal to 76%. The Principle Component Analysis (PCA) is used for feature transformation of NSL-KDD dataset [7]. Then, the feature subset obtained from PCA is optimized using Genetic Algorithm (GA) and PSO algorithms. The optimized features are used along with a Modular Neural Network (MNN) model for network intrusion detection. They obtained (DR=98.2%, FAR=1.8%) for GA and (DR=99.4%, FAR=0.6%) for PSO. In the study Chae et al. [8], they proposed a feature selection method using Attribute Ratio (AR). Then, they applied the proposed method on NSL-KDD dataset to select the feature subset and tested the selected features on a decision tree classifier.

Wahba et al. proposed a hybrid feature selection method based on Correlation based Feature Selection (CFS) and Information Gain (IG) [9]. The proposed method is applied on NSL-KDD dataset and a Naive Bayes classifier is trained on the selected features using the Adaptive Boosting (AdaBoost) technique. A misuse detection approach is presented using Classification And Regression Trees (CART) [10]. The proposed model is applied on 29 features of NSL-KDD dataset. In the study Eid et al. [11], the authors have proposed a hybrid Bi-Layer behavioral-based feature selection approach. The proposed approach is evaluated on 20 selected features of NSL-KDD dataset. A feature selection method based on mutual information is proposed and the optimal features are tested on a Least Square Support Vector Machine-based IDS (LSSVM-IDS) over NSL-KDD dataset [12]. Although, they had 18 optimal features, but they gained good results (DR=98.76%, FAR=0.28%). In the study [13], an IDS has been proposed to detect malicious in computer networks. The proposed IDS is validated on the CICIDS2017 dataset after a recursive feature elimination is performed via random forest. Then, a Deep Multilayer Perceptron (DMLP) model is applied on the selected features and they got Accuracy equal to 91%.

Naidoo et al. have introduced two-stage feature selection method called Cluster Validity Indices [14]. In the first stage a K-means cluster algorithm is applied to NSL-KDD dataset to select candidate feature subsets. Then, in the second stage a GA is utilized to identify the optimal feature subset. An approach of feature selection is employed using univariate features selection associated with a recursive feature elimination using a decision tree classifier [15]. It was tested on 12 selected features of NSL-KDD dataset. In the study [16], they employed a SVM classifier to select multiple feature subsets of NSL-KDD dataset. Then, they tested these subsets on a SVM classifier for multi-class classification and recorded the results (DR=82%, FAR=15%). Ganapathy et al. presented several feature selection and classification methods in network intrusion detection [17]. They also proposed their own feature selection approach and tested it along with multiclass SVM. In the study Wang et al. [18], they analyzed the problem of Gaussian-distributed Wireless Sensor Network (WSN) and discussed effects of various network parameters in intrusion detection. A network intrusion detection framework is presented in cluster-based WSN and a SVM classifier is used for classification [19].

Ahmad and Amin utilized PCA for feature transformation and PSO for feature selection [20]. Then, they used SVM for classification over KDD CUP 99 dataset. A monitoring technique is proposed for intrusion detection in Wireless Mesh Networks (WMN) [21]. They demonstrated optimal results in DR and resource consumption in WMN. In the study Staudemeyer and Omlin [22], they introduced a feature selection mechanism which was based on custom feature preprocessing. They reported that their mechanism may miss many important features. A features selection algorithm was proposed based on record to record travel and SVM is applied on KDD CUP 99 dataset for their experiments [23]. Feature selection method was proposed based on the cuttlefish optimization in network intrusion detection [24]. Decision tree (DT) was applied on the feature subset and they had improved performance in terms of DR and FAR. Alom et al. investigated the effectiveness of utilizing Extreme Learning Machine (ELM) and Regularized ELM (RELM) models in network intrusion detection on NSL-KDD dataset [25]. After reducing the data dimensions from 41 to 9 essential features with 40% training data, they had a testing accuracy of 98.2% and 98.26% for ELM and RELM, respectively.

Alternatively, there are many articles point out the success of using deep learning models in network intrusion detection without feature selection. Javaid et al. proposed a self-taught learning model in two stages, the first is sparse AutoEncoder (AE) for unsupervised feature learning and the second is softmax regression classifier trained on the derived training data [26]. Their model is applied on the NSL-KDD dataset, and they achieved accuracy greater than 98%. A novel stacked non-symmetric deep AE classifier was presented for network intrusion detection on NSL-KDD dataset (DR=85.42%, FAR=14.58%) [27]. In the study Potluri and Diedrich [28], an accelerated DNN model is employed along with AEs and softmax layer to perform the fine tuning with supervised learning. They evaluated their model over NSL-KDD dataset (DR=97.5%, FAR=3.5%). An RNN-based intrusion detection system is introduced and applied on NSL-KDD dataset (DR= 72.95%, FAR=

3.44%) [29]. Alom et al. examined the ability of DBN model to detect anomalies on only 40% of NSL-KDD dataset and achieved accuracy equal to 97.5% [30].

Liu and Zhang applied ELM into the learning process of DBN model, and evaluated DBN over NSL-KDD dataset (DR= 91.8%) [31]. An ensemble deep learning model is presented which comprises AE, DBN, DNN, and ELM methods and validated over NSL-KDD dataset (DR=97.95%, FAR= 14.72%) [32]. In the study Qu et al. [33], they proposed a DBN-based model for network intrusion detection over NSL-KDD dataset with accuracy equal to 95.25%. Tsiropoulou et al. have invistagated the problem of proactively protecting a passive RFID network from security threats imposed by intruders that introduce high interference to the system [34]. Moreover, they proposed a network control and management framework which can be used in Internet of Things (IoT) environment to react against malicious attacks, by minimizing if not totally eliminating the potential damage. A novel IDS for the IoT that called SVELTE is designed, implemented and evaluated [35]. The detection algorithms in SVELTE Target routing attacks such as spoofed or altered information, sinkhole, and selective-forwarding. They reported that SVELTE has a high performance in terms of DR and FAR as well as a small overhead. The above mentioned discussion highlights the importance of feature selection and classification in network intrusion detection. Accordingly, developing a deep learning approach for network intrusion detection that applying to the optimal feature subset with a high DR as well as a low FAR is still a big challenge.

3. Proposed approach

In this section, we introduce our metaheuristic-based intrusion detection approach. This technique extends our former work [36] by applying PSO-based algorithm in both feature and hyperparameter selection phases. The proposed algorithm will be used later in the pre-training phase to enhance the performance of deep learning models in network intrusion detection.

3.1. Feature selection

Principally, in any classification task, the feature space is a substantial factor that affects the performance of the classifier. Determining which features are significant to the classification task at hand is a hard process. In order to resolve this problem, a feature subset selection or sometimes called "dimensionality reduction" is useful to handle the process of removing unimportant features, e.g. redundant and irrelevant features. Whereas, redundant features refer to duplicate much or all of the information contained in one or more other attributes, irrelevant features contain no information that is useful for the particular data mining task. The benefits of feature selection are not limited to eliminating unimportant features, but also extend to avoid the curse of dimensionality, reduce noise, reduce the time and space required in data mining, and allow easier visualization. Notably, feature selection leads to significantly improve the performance of the classifier in intrusion detection, because the redundant and irrelevant features can confuse the classifier and increase the number of misclassifications. Also, it could improve the computation efficiency by shortening the running time as well as simplifying the model's structure [37].

One of the conventional feature selection methods is performing an exhaustive search to find the optimal feature subset which might take too long time [38]. Therefore, finding a good solution within a reasonable amount of time rather than the optimal solution is more interested in real-world applications. Recently, Evolutionary Computation (EC) techniques have been applied to obtain the optimal or near-optimal solution of feature selection problem. For instance, Genetic Algorithms (GAs) [39,40], PSO [41–43], Genetic Programming (GP) [44,45], and Ant Colony Optimization (ACO) [46,47]. Compared to other EC techniques, it has been shown that PSO is an effective algorithm for feature selection problems [41–43,48] because it is easier to implement and faster to converge [49]. After describing the background of PSO and its variations and usage in feature selection, we present in Section 3.1.4, the algorithm that we used in our experiments for feature selection.

3.1.1. Continuous PSO

PSO is a metaheuristic optimization algorithm for optimizing non-linear functions in continuous search space. It was firstly proposed by Eberhart and Kennedy in 1995 [50], and was inspired to mimic the social behavior of birds or fish. The swarm is made up of many particles, each of which is considered as a candidate solution. Every particle i at current iteration t has three vectors of length N, namely, position, velocity, and personal best, where N is the dimension of the problem. The position (P_t^i) identifies the current position of that particle in the search space of the problem, the velocity ($V_{(t+1)}^i$) determines both of direction and speed of that particle in the search space at next iteration, meanwhile, personal best (P_{best}^i) indicates the best position of that particle that has been found so far. Moreover, another important vector for the swarm called global best (*Gbest*) which stores the best position that has been explored over the swarm so far. The personal best vector for each particle and the global best vector for the swarm are updated at the end of each iteration. Indeed, the personal best vector is considered as the cognitive knowledge of the particle, whereas the global best vector represents the social knowledge of the swarm. Mathematically, the velocity and position vectors are updated to next iteration $t+1$ according to Eqs. (1) and (2), respectively.

$$V_{t+1}^i = W \times V_t^i + C_1 \times r_1(t) \times (P_{best}^i - P_t^i) + C_2 \times r_2(t) \times (G_{best} - P_t^i) \quad (1)$$

$$P_{t+1}^i = P_t^i + V_{t+1}^i \quad (2)$$

Where W is the Inertia weight constant which controls the impact of particle's velocity at the current iteration on the next iteration to not let the particle to get outside the search space. W constant is usually ranged in [0.4,0.9]. $C1$ and $C2$ are constants and known as acceleration coefficients. $C1$ and $C2$ constants are usually ranged in [1,5]. Meanwhile, $r1$ and $r2$ are random values uniformly distributed in [0,1]. The goal of using $C1$, $C2$, $r1$ and $r2$ constants is to scale both of cognitive knowledge and social knowledge on the velocity changes. Accordingly, all particles will approach to the optimal solution of the problem. Finally, PSO checks the stop criterion and if one satisfied, PSO will output the global best vector as the optimal solution and terminates. Otherwise, PSO will proceed to the next iteration and repeats the same procedure. The stop criterion is occurred when either the improvement of the global best is smaller than stopping value (ϵ) or the maximum number of iteration is reached.

3.1.2. Binary PSO

The traditional PSO works well for continuous domains, but it might bring negative effects on the results when dealing with discrete space. Therefore, Kennedy and Eberhart introduced Binary PSO (BPSO) algorithm in 1997 to overcome this problem [51]. Unlike PSO, in the BPSO, the position, personal best, and global best vectors are represented by binary strings, that is, all vector's elements are restricted to 0 or 1. Also, the velocity vector in BPSO shows the probability of the corresponding element in the position vector taking value 1. Mathematically, Eq. (1) is still applied to update the velocity vector at each iteration. Afterwards, the sigmoid function in Eq. (3) is employed to transform $V_{(t+1)}^i$ into the range of [0,1]. Then, BPSO updates the position vector for each particle

by using Eq. (4), where *rand*() is a random number selected from a uniform distribution in [0,1].

$$S(V_{t+1}^i) = \frac{1}{1 + e^{-V_{t+1}^i}} \tag{3}$$

$$P_{t+1}^i = \begin{cases} 1, & if rand() < S(V_{t+1}^i) \\ 0, & otherwise \end{cases} \tag{4}$$

It has been reported that the traditional BPSO algorithm suffers from two main drawbacks [52]. The first is the particle's position at the next iteration solely depends on the velocity vector. So, there is a need for a new way to compute the new particle's position taking into account the influence of current particle's position. The second is there is a big chance that BPSO has a premature convergence while maintaining the general diversity. Therefore, there is also a need to change the velocity updating formula to let the particle move constantly towards the best solution. As a result, Zhou et al. [52] proposed a new binary PSO algorithm named Fitness Proportionate Selection Binary Particle Swarm Optimization (FPSBPSO) to solve the two aforementioned drawbacks. FPSBPSO updates the particle's velocity and position at the next iteration according to Eqs. (5) and (6), respectively Zhou et al. [52].

$$V_{t+1}^i = \begin{cases} mr, & if n_0 = 0 \\ 1 - mr, & if n_1 = 0 \\ \frac{n_1}{n_0 + n_1}, & otherwise \end{cases} \tag{5}$$

$$P_{t+1}^i = \begin{cases} 1, & if rand() < V_{t+1}^i \\ 0, & otherwise \end{cases} \tag{6}$$

Where *mr* is the algorithm's free parameter, n_0 is the number of zero-valued of the corresponding bits of the particle's current position, the particle's personal best, and the global best vectors, and n_1 is the inverse of n_0 and can be calculated using (3- n_0). Rather than FPSBPSO algorithm resolved the drawbacks of BPSO, it also has been shown that FPSBPSO improved the results of optimization problems, especially in the case of feature selection process [52]. Furthermore, the FPSBPSO is easier to be tuned than BPSO, because it has only one parameter. They concluded that the value of 0.01 is a good choice for *mr* parameter in most cases. Finally, the binary PSO generally outperforms the continuous PSO in the feature selection problem due to the fact that the feature selection problem occurs in a discrete search space [53]. Therefore, we will exploit the binary PSO in our design of the feature selection method.

3.1.3. BPSO In feature selection

In general, any of feature selection methods needs an evaluation process to measure the goodness of the candidate feature subsets. Obviously, BPSO algorithm utilizes a predefined fitness function to handle this duty by computing the fitness score for every particle in the swarm. Thus, based on whether the fitness function involves a learning algorithm or not, Langley [54] grouped them into two broad categories: filter-based approaches and wrapper-based approaches. The filter-based approaches select features without using a learning algorithm as an evaluation criterion. On the other hand, the wrapper-based approaches construct a classifier to test the candidate feature subset on unseen data in the evaluation procedure. In spite of wrapper-based approaches usually achieve better results than filter-based approaches in feature selection problem [55], but they are computationally expensive especially when the number of feature are large [45]. However, the filter-based approaches are the most preferred in feature selection tasks because they argued to be computationally less expensive and more general [38,56]. For this reason, we focused in our study on filter-based approaches.

Further, the desired goal of feature selection methods is to discover the optimal feature subset that is the smallest one as well as it could achieve the highest classification accuracy. According to this perspective, the feature selection is basically a multi-objective optimization problem, and it produces several trade-off solutions (feature subsets) [57]. The single-objective feature selection method is with full contradictory with the multi-objective one where the former can solely generate one optimal feature subset [55]. In our study, we focused only on the single-objective feature selection methods because the nature of our approach which requires to produce one optimal feature subset without any interference of the user.

There are several single-objective filter-based feature selection methods have been proposed in the literature. It was reported that the two proposed methods in the study of Cervante et al. [53] proved their efficiency and superiority over others [55]. They developed two single-objective filter-based feature selection methods using BPSO and information theory. The first evaluates the relevance and redundancy of the selected feature subset by measuring the mutual information of each pair of features. Mathematically, the fitness function of the first method can be obtained using Eq. (7) [53].

$$Fitness_1 = \alpha_1 \times D_1 - (1 - \alpha_1) \times R_1 \tag{7}$$

where

$$D_1 = \sum_{x \in X, c \in C} I(x; c),$$

$$R_1 = \sum_{x_i, x_j \in X} I(x_i; x_j)$$

Where X is the set of the selected features and C is the set of class labels. D_1 calculates the relevance of the selected feature subset to the class labels by determining the mutual information between each feature and the class labels. On the other hand, R_1 evaluates the redundancy contained in the feature subset by indicating the mutual information shared by each pair of selected features. The goal of using $Fitness_1$ is to select a features subset with maximum relevance to the class labels and simultaneously with minimum redundancy to each others. Meanwhile, the second method determines the relevance and redundancy of the selected feature subset by measuring the entropy of each group of features. Mathematically, the fitness function of the second method can be obtained using Eq. (8) [53].

$$Fitness_2 = \alpha_2 \times D_2 - (1 - \alpha_2) \times R_2 \tag{8}$$

where

$$D_2 = \sum_{c \in C} IG(c|X),$$

$$R_2 = \frac{1}{|S|} \sum_{x \in X} IG(x|\{X/x\})$$

Where X and C is as same as defined in Eq. (7). D_2 indicates the relevance between the selected features and the class labels by calculating the information gain (entropy) in the class labels given information of the selected features. R_2 evaluates the redundancy contained in the selected feature subset by measuring the joint entropy of all the selected features. $Fitness_2$ is a maximization fitness function which minimizes the redundancy (R_2) and simultaneously maximizes the relevance (D_2). In addition to that, α_1 and α_2 are weight parameters that are constant values ranged in [0,1]. These parameters are used to control the importance of the relevance and redundancy of the selected features to improve the performance of the proposed methods. The experimental results showed that the proper value of these parameters is 0.8 or 0.9.

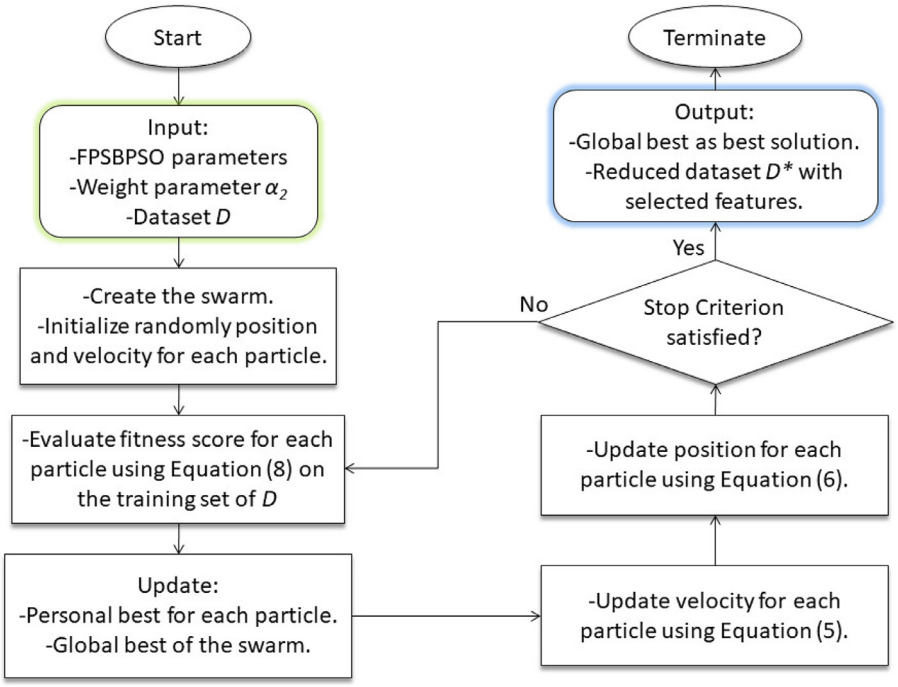

Fig. 1. The flowchart of FPSBPSO-E algorithm.

Although the first method produces a smaller feature subset, the second method reduces significantly the number of features and achieved higher classification accuracy [53]. Therefore, we have selected the second method as a basis of our feature selection algorithm in Section 3.1.4.

3.1.4. FPSBPSO-E Algorithm

We introduce a single-objective filter-based feature selection algorithm using FPSBPSO and Entropy (FPSBPSO-E). It works similar to the second method described in the previous subsection, but with one major difference. Instead of using the traditional BPSO, we employed the FPSBPSO algorithm to update the particle's position and velocity vectors. This pivotal modification will avoid premature convergence and enhance the overall performance in the feature selection process. Fig. 1 shows the flowchart of FPSBPSO-E algorithm.

Firstly, the algorithm makes up a swarm with a specified number of particles. Every particle is represented by a binary string which its length is equal to the size of all available features in the dataset. In the binary string, the value of each bit indicates whether the corresponding feature is selected or not, that is, value "1" means that the feature is selected and "0" otherwise. Next, the algorithm initializes the position and velocity of each particle randomly. After that, the algorithm executes the following steps: it computes the fitness score for each particle by using Eq. (8) on the training set of the given dataset. Moreover, it updates the personal best vector for each particle as well as updates the global best vector of the swarm. Then, for each particle, it updates each bit of the particle's velocity vector according to Eq. (5) along with updating each bit of the particle's position vector according to Eq. (6). Furthermore, it checks the stopping criterion whether is satisfied or not. We determined two different stooping criterion, the first is reaching the predefined number of iterations, whereas the second is the improvement of the global best vector's fitness score is less than the predetermined threshold (ϵ). If any of these conditions is met, then the algorithm returns the global best as the optimal solution, reduces the original dataset by keeping only the selected features and removing the others, and terminates. Else, the algorithm begins the next iteration and repeats the same steps until the stopping criterion is reached. The FPSBPSO-E algorithm have a complexity of $O(pnlog(n))$ where n is the initial population size and p is the number of iterations.

3.2. Hyperparameter selection

Although deep learning models may increase the training time due to their complexity, they outperform the traditional machine learning algorithms in terms of performance. Nevertheless, their performance relies extremely on the selected values of hyperparameters. Hence, the critical concern when using one of these deep learning models is how to adjust its hyperparameters properly. Basically, the hyperparameters of a deep learning model are the set of fundamental settings that control the behavior, architecture and performance of the model in the underlying task. The problem is that the values of these hyperparameters are varying from task to task, and prior knowledge of these values is required to set them just before the learning process begins. To overcome this problem, several exhaustive search techniques [58–60] are utilized to find the best values manually in a trail-and-error manner. In spite that the former techniques have been gained good results in many cases, but they take too long time to finish and are infeasible in large and complex search space. In contrast, using EC techniques such as GAs [61–65] and PSO [66–69] to select the best values of hyperparameters automatically is recently widely explored. It has been shown that PSO is superior to other EC techniques for hyperparameter selection problem due to its simplicity and generality [70].

Recently, Elmasry et al. [71] proposed a novel PSO-based algorithm for hyperparameter selection. This study has attracted significant attention, because the proposed algorithm is consistent and flexible to any given deep learning model. The former PSO-based algorithm discovers the optimal hyperparameter vector that maximizes the accuracy over the given training set. In addition to that, it sustains the generality where the user is in charge of identifying the desired hyperparameters. Regarding the functionality, it consists of four sequential phases which are preprocessing, initializa-

Table 1
The defined hyperparameters and their domains.

Hyperparameter	Domain	Type
Learning rate	[0.01,0.9]	Continuous
Momentum	[0.1,0.9]	Continuous
Decay	[0.001, 0.01]	Continuous
Dropout rate	[0.1,0.9]	Continuous
Number of hidden layers	[1,10]	Discrete with step=1
Numbers of neurons of hidden layers	[1,100]	Discrete with step=1
Number of epochs	[5,100]	Discrete with step=5
Batch size	[100,1000]	Discrete with step=50
Optimizer	Adagrad, Nadam, Adam, Adamax, RMSprop, SGD	Discrete with step=1
Initialization function	Zero, Normal, Lecun uniform, Uniform, Glorot uniform, Glorot normal, He uniform, He normal	Discrete with step=1
Layer type	Dropout, Dense	Discrete with step=1
Activation function	Linear, Softmax, Relu, Sigmoid, Tanh, Hard sigmoid, Softsign, Softplus	Discrete with step=1

tion, evolution, and finishing phases. It starts with the preprocessing phase where the user sets the main parameters of PSO as well as defines a list of the desired hyperparameters and their default domains. They defined twelve hyperparameters, namely, learning rate, decay, momentum, number of epochs, batch size, optimizer, initialization function, number of hidden layers, layer type, dropout rate, activation function, and number of neurons of the hidden layer. Equally important, the first eight of them are global parameters that are fixed in the model. Meanwhile, the last four parameters are layer-based which vary from layer to layer. Table 1 shows the defined hyperparameters and their default domains [71]. The training set of the particular dataset is split into two separate parts using a hold-out sampling technique with 66% for training only and 34% for validation. Also, the user specify the type of the learning model he wants to use.

Next, in the initialization phase, the algorithm initializes the position and velocity vectors for each particle in the swarm randomly. After that, the algorithm enters the evolution phase by executing the following steps: it computes the fitness score for each particle by constructing the deep learning model that tuned by the selected hyperparmeters, training the model on the training only set, and computing the accuracy of the trained model over the validation set. Further, it updates the personal best vector for each particle as well as updates the global best vector of the swarm. Then, for each particle, it updates the velocity vector according to Eq. (1) along with updating the position vector according to Eq. (2). Furthermore, it checks the stopping criterion whether is satisfied or not. If the stopping criterion is met, the algorithm outputs the optimized hyperparameters vector, and terminates in the finishing phase. Otherwise, it begins the next iteration and repeats the same previous steps till converging. Fig. 2 depicts the flowchart of the PSO-based algorithm for hyperparameter selection.

Since the PSO-based hyperparameter selection algorithm computes fitness function for each particle, updates each particleâs personal best vector and finally updates the global best vector for the swarm regarding all iterations, the complexity become $O(n^4)$. To emphasize the computational overhead, we benefited from Rosenbrock function to reveal the performance of the FPSBPSO-E algorithm, and we observed that it was more consistent but less sensitive to the choice of hyperparameters where PSO-based hyperparameter selection algorithm produced better results than any other alternative. But it has a slower convergence on locating the global minimizer. When comparing both approaches PSO-based hyperparameter selection algorithm requires almost double resource and execution time against FPSBPSO-E algorithm.

3.3. Double PSO-based algorithm

The double PSO-based algorithm is a hierarchical multipurpose optimization algorithm. It is a top-down algorithm consists of two levels. The upper level for feature selection that utilizes the FPSBPSO-E algorithm, whereas the lower level for hyperparameter selection that exploits PSO-based algorithm explained in Section 3.2. The user is responsible to enter all required operating parameters in the two levels separately. Afterwards, the double PSO-based begins in the upper level and receives the original dataset which has the complete set of features (D) as an input. The FPSBPSO-E algorithm produces the reduced dataset which has only the selected feature subset (D^*) as an output. Then, the double PSO-based algorithm moves down to the lower level and receives the type of deep learning model (M) along with a copy of D^* as inputs. The PSO-based algorithm finds out the optimal hyperparameter vector (H^*) for M as an output. Finally, the double PSO-based algorithm outputs both D^* and H^* then terminates. Fig. 3 shows the mechanism of the proposed algorithm.

4. Experimental setup

We carried out two main empirical experiments, each of them on one of datasets described in Section 6. Indeed, each experiment is done twice, firstly for a binary classification, then for a multiclass classification task. Moreover, in each experiment, The performance of three deep learning models in network intrusion detection is validated on the particular dataset. In the next subsections, we explain the methodology of executing our empirical experiments as well as the description of each model.

4.1. Methodology

Our methodology is designed to be obvious and straightforward. It consists of four consecutive phases, namely, preprocessing, pre-training, training, and testing phases. The details of these phases are presented in the following subsections. Fig. 4 depicts the diagram of our methodology.

4.1.1. Data preprocessing

Initially, we performed data preprocessing on the particular dataset by applying data numericalization and normalization processes. The first constraint of our experiments; most of the machine learning models can only work with numerical values for training and testing. Therefore, it is necessary to convert all non-numerical values to numerical values by performing a data numericalization. Indeed, in the literature, there are two methods to perform data numericalization. The first is called "one-hot encoding" which gives each type of the nominal attribute a different binary vector. For instance, in NSL-KDD dataset, there are three nominal features, namely, 'protocol_type', 'service', and 'flag' features. For 'protocol_type' feature, there are three types of attributes, 'tcp', "udp', and "icmp', and its numeric values are encoded as binary vectors (1,0,0), (0,1,0) and (0,0,1). Similarly, the feature "service"

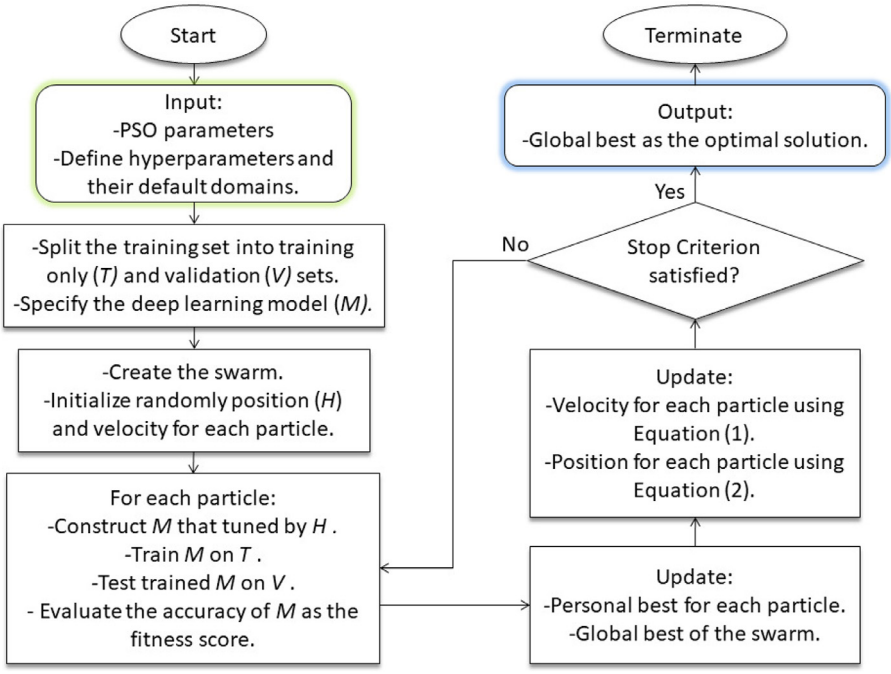

Fig. 2. The flowchart of the PSO-based algorithm for hyperparameter selection.

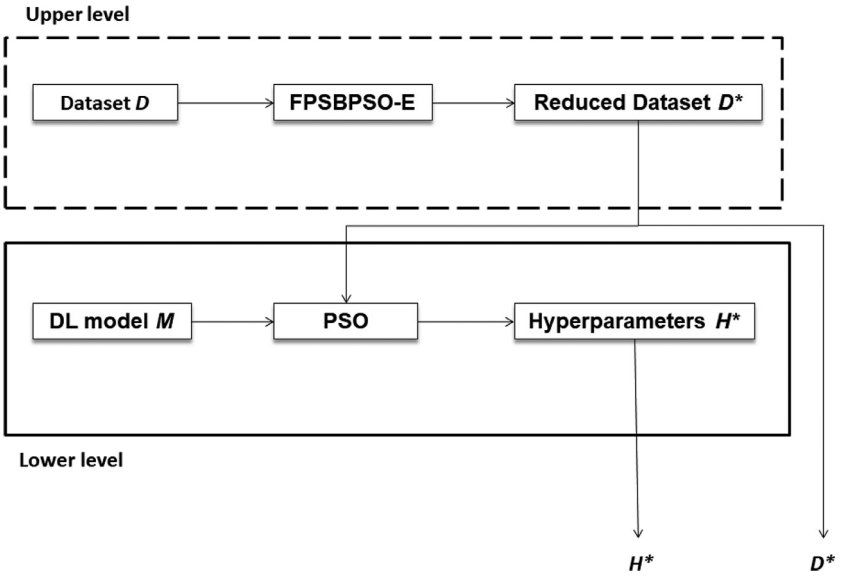

Fig. 3. The mechanism of the double PSO-based algorithm.

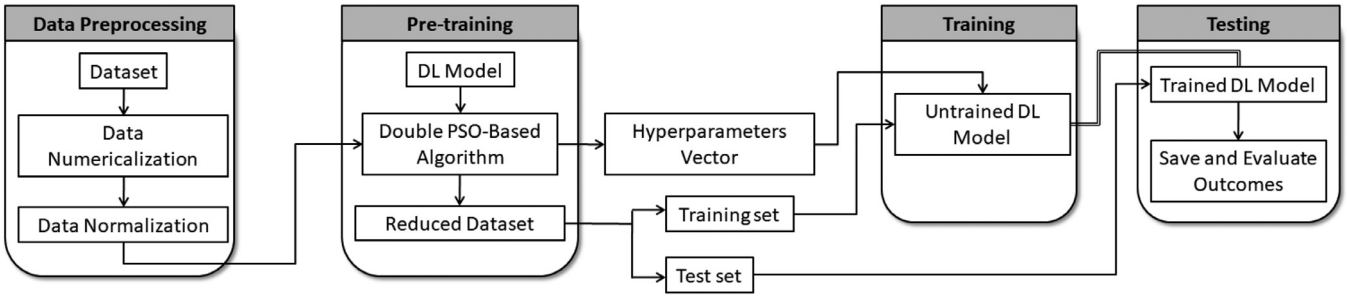

Fig. 4. The methodology diagram of the experiments.

Table 2
The main operating parameters of the double PSO-based algorithm.

Parameter	Domain	Selected value	
		Feature selection FPSBPSO-E	Hyperparameter selection Continuous PSO
Swarm size	[5,40]	20	40
Minimum velocity (V_{min})	{0,1}	–	0
Maximum velocity (V_{max})	{0,1}	–	1
Acceleration coefficients (C_1, C_2)	[1,5]	–	1.43
Inertia weight constant (W)	[0.4,0.9]	–	0.69
Maximum number of iterations	[30,50]	30	50
Stopping threshold (ϵ)	[0.001,0.0001]	0.0001	0.001
Free parameter (mr)	[0,1]	0.01	–
Weight parameter (α_2)	[0,1]	0.9	–

has 70 types of attributes, and the feature "flag" has 11 types of attributes. Continuing in this way, 41-dimensional features map into 122-dimensional features after transformation. The second method is what we used in this paper in such manner that for each nominal feature, its values are ordered alphabetically. After that, the ordered nominal values are converted to numerical values by assigning specific values to each variable ranged in [1, length of the list] (e.g. 'icmp'=1, 'tcp'=2 and 'udp'=3).

Compared with one-hot encoding method, we prefer to use the second method because it has many advantages. The second method does not increase the number of features because every transformed nominal feature is represented by one value. In contrast, one-hot encoding method increases the number of features because every transformed nominal feature is represented by a binary vector which its length depends on number of the nominal feature's values. As a result, the architecture of the models when using the second method will be simpler than when using one-hot encoding method that because the model's inputs will be less. Thus, the second method will decrease the time needed to train and test the model.

Then, data normalization process is taken place when all numeric features in the dataset (including the transformed nominal features) are mapped into [0,1] linearly by using Min-Max transformation formula in (9).

$$x_i = \frac{x_i - Min}{Max - Min} \tag{9}$$

Where x_i is the numeric feature value of the i^{th} sample, Min and Max are the minimum and maximum values of every numeric feature, respectively.

4.1.2. Pre-training

The pre-training phase is imperative to employ the proposed double PSO-based algorithm. Hence, the preprocessed dataset obtained from the previous phase is passed along with the type of deep learning model as inputs to the proposed algorithm. Once the double PSO-based algorithm finished, it outputs the optimal hyperparameter values of the used model as well as the reduced version of the particular dataset. These outputs are delivered to the next phases for further processing. Table 2 shows the values of main operating parameters of the proposed algorithm. The selected values are gained by performing a grid search for each parameter in its predefined domain. In addition to that, the domains are recommended in many theoretical and empirical previous studies [72,73].

Regarding the feature selection process using FPSBPSO-E algorithm in the upper level of the proposed algorithm, we listed in Table 3 the selected feature subset and feature reduction rate for each dataset. The Feature Reduction Rate (FRR) gives information about the fraction of number of features which is removed from the complete set of features. It can be calculated according to Eq. (10).

$$FRR = 1 - \frac{NumberOfSelectedFeatures}{NumberOfAllFeatures} \tag{10}$$

Table 4 shows the values of the global hyperparameters associated for deep learning models on the corresponding datasets. These findings are obtained after finishing of the hyperparameter selection process using PSO-based algorithm in the lower level of the proposed algorithm. The layer-based hyperparameters as well as a brief explanation of each deep learning models are presented in Section 4.2.

4.1.3. Training and testing

We construct the deep learning model and tune it by the optimal hyperparameters. The resulting model is trained on the full training set of the reduced dataset. Subsequently, the trained model is tested on the test set of the reduced dataset. Finally, the classification outcomes are stored for further processing later.

4.2. Models

To accomplish the network intrusion detection on each of the datasets, we utilized three well-known deep learning models, namely, DNN, LSTM-RNN, and DBN. Indeed, there are many reasons for choosing these models rather than other deep learning techniques. Firstly, many review articles point out their success in solving the network intrusion detection problem [74,75]. Further, they are common in the static classification tasks such like intrusion detection. Finally, the aforementioned models are widely used in the literature, so our results can be compared easily.

4.2.1. DNN

DNN merely models the Artificial Neural Network (ANN) but with many deep hidden layers [76]. Basically, DNN typically consists of an input layer, one or more hidden layers, and an output layer. The hidden layers deemed to do the most important work in DNN. Every layer in DNN consists of one or more artificial neuron in such a way that theses neurons are fully-connected from layer to layer. Moreover, the information is processed and propagated through DNN in feed-forward manner, i.e., from the input layer to the output layer via the hidden layers. Fig. 5 presents the resulting architectures of DNN for each dataset regarding binary and multiclass classifications.

4.2.2. LSTM-RNN

Unlike traditional ANN, each neuron in any of the hidden layers of the RNN has additional connections from its output to itself which so-called self-recurrent as well as to its adjacent neuron at the same hidden layer. Therefore, the information circulates in the network which practically make the hidden layers as a storage

Table 3
The feature selection of datasets.

Dataset	No. of features	Selected features	No. of selected features	Selected features (%)	FRR (%)
NSL-KDD	41	1,2,3,5,6,23,25,30,32,37	10	24.39	75.61
CICIDS2017	80	8,11,15,18,20,23,24,26,28,31,33,37, 43,44,51,53,54,59,70,73,74,77,80	23	28.75	71.25

Table 4
The resulting global hyperparameter values.

Hyperparameter	NSL-KDD			CICIDS2017		
	DNN	LSTM-RNN	DBN	DNN	LSTM-RNN	DBN
Learning rate	0.4	0.2	0.09	0.1	0.06	0.01
Decay	0.01	0.01	0.008	0.005	0.003	0.001
Momentum	0.71	0.55	0.2	0.3	0.14	0.1
Number of epochs	55	30	20	15	10	5
Batch size	200	350	400	450	550	550
Optimizer	SGD	Adagrad	Adamax	RMSprop	Nadam	Adam
Initialization function	Normal	Lecun Uniform	He normal	Uniform	Zero	He uniform
Dropout rate	0.5	0.4	–	0.25	0.1	–

unit of the whole network. However, the traditional RNN's structure suffers from inherent drawback known as gradient vanishing and exploding [77]. This serious problem often appears when RNN is trained using the back-propagation method. It limits the use of RNN to be only in short-term memory tasks. In order to resolve this problem, a Long Short-Term Memory (LSTM) structure is introduced by Hochreiter and Schmidhuber [78]. LSTM uses a memory cell that is composed of four parts, namely, input gate, neuron with a self-recurrent connection, forget gate, and output gate. While, the main objective of using a neuron with a self-recurrent is to record information, the goal of using three gates is to control the flow of information from or into the memory cell. It has been reported that the LSTM-RNN model can be obtained easily by replacing every neuron in the RNN's hidden layers to an LSTM memory cell [79].

Alternatively, to overcome the gradient vanishing and exploding in the conventional RNN, Cho et al. [80] proposed a new sophisticated gating mechanism which called Gated Recurrent Unit (GRU). The main distinction between LSTM and GRU is that GRU has only two gates (update and reset) instead of three in LSTM. Thus, GRU only exposes the full hidden content without any control. Intuitively, the function of the update gate in the GRU is to determine how much of the previous information needs to be kept around. On the other hand, the reset gate is responsible for deciding how much of the previous information to forget. Indeed, GRU's performance generally is on par with LSTM, but LSTM's tend to do better than GRU in large datasets [81]. Fig. 6 depicts the resulting architectures of LSTM-RNN for each dataset regarding binary and multiclass classifications.

4.2.3. DBN

The Boltzmann Machine (BM) is an energy-based neural network model which consists of only two cascaded layers, namely, the visible and hidden layers [74]. BM is a particular form of log-linear Markov Random Field (MRF), and all its neurons are binary, that is, outputs either 0 or 1. Regarding BM's structure, the neurons are connected by undirected connections between the visible and hidden layers as well as between neurons at the same layer. Recently, a customized version of BM is introduced and known as Restricted Boltzmann Machine (RBM) [82]. RBM is deemed to be a kind of the stochastic generative learning model. It is nothing more than a BM without visible-to-visible and hidden-to-hidden connections, that is, the whole connection is only between the visible layer and the hidden layer.

Hinton et al. [83] proposed in 2006 a generative probabilistic neural network called DBN. From a structural point of view, DBN is a deep classifier that combines several stacked RBMs to a layer of Back Propagation (BP) [84] neural network. Meanwhile, the stacked RBMs is considered as the hidden layers of DBN, the BP layer is the visible layer. In addition to that, the connectivity between all hidden layers in DBN is undirected connections. In contrast, the last RBM is connected to the visible layer by directed weights. In the open literature, the conventional DBN has two sequential procedures which are pre-training and fine-tuning. The pre-training procedure takes place by training all the hidden layers (RBMs) in layer-by-layer manner, i.e., it trains only one layer at a time with the output of the higher layer being used as the input of the lower layer. In order to achieve that, a greedy layer-wise unsupervised training algorithm [85] is used along with unlabeled training data. Afterwards, the parameters of whole DBN are fine-tuned by using BP learning algorithm along with the labeled training data. Recently, DBN has attracted much attention and used in many data mining applications. Some of these applications uses DBN as an unsupervised feature extraction and feature reduction model. In this case, DBN has only the stacked RBMs without any BP layer. On the other hand, some applications uses DBN for classification tasks, and in that case, DBN consists of several stacked RBMs along with a BP layer [86]. In this study, we used the DBN as a classifier on each dataset. Fig. 7 shows the resulting architectures of DBN for each dataset regarding binary and multiclass classifications.

Moreover, the term DBN^k, where k denotes the number of RBM layers, is using to explain the structure of a DBN model. According to our results, we got DBN^3 (10-5-3-2), DBN^2 (10-8-5), DBN^2 (23-14-2), and DBN^4 (23-17-16-10-8) for the NSL-KDD (binary), NSL-KDD (multiclass), CICIDS2017 (binary), and CICIDS2017 (multiclass), respectively. Regarding the iterations number, we also got 250, 350, 450, and 500 iteration numbers for the DBN models of the NSL-KDD (binary), NSL-KDD (multiclass), CICIDS2017 (binary), and CICIDS2017 (multiclass), respectively.

5. Evaluation metrics

In this paper, we have performed our experiments for both binary and multiclass classification tasks. As mentioned in Section 6, each dataset has a mixture of normal (negatives) and various attacks (positives) samples. In binary classification, there are only two labeled classes, namely, normal and attack. Regardless to the attack type, the attack class contains the samples of all attacks in one class. On the other hand, the multiclass classification seeks to

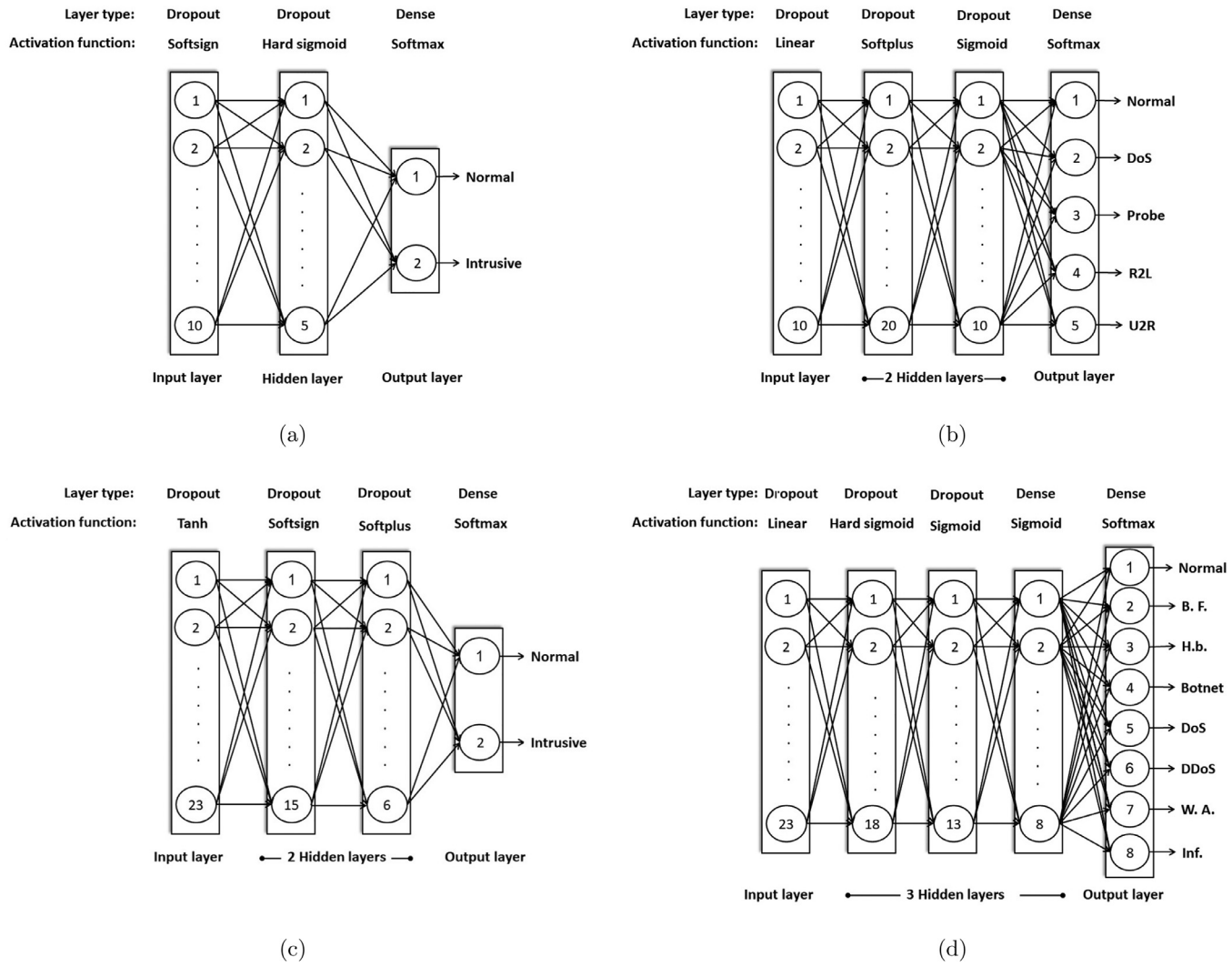

Fig. 5. DNN architectures (a) NSL-KDD (binary) (b) NSL-KDD (multiclass) (c) CICIDS2017 (binary) (d) CICIDS2017 (multi-class).

Table 5
The confusion matrix of binary classification of network intrusion detection.

Actual class	Predicted class	
	Normal	Attack
Normal	TN	FP
Attack	FN	TP

not just detect a malicious connection but also to assign the correct type as well. As a result, the number of labeled classes varies from dataset to another (normal class + n attack classes). In the following subsections, we introduce the binary and multiclass classifications, and which evaluation metrics are used along with them.

5.1. Binary classification

Notably, there are four major outcomes can be gained from any classification task, namely, True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). Table 5 shows the possible confusion matrix when applying the network intrusion detection as a binary classification.

Then, the four outcomes are utilized to compute 15 well-known evaluation metrics to assess the performance of the deep learning model on the particular dataset. Some of the used metrics are widely used in the previous studies that focused on network intrusion detection subject, so it would be easy for readers to compare results. The list of the evaluation metrics" definition and their corresponding equations are as follows:

- Accuracy is the rate of true classifications for all test set.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{11}$$

- Precision shows the classifier's exactness, i.e., the rate of samples that correctly labeled as an attack from all samples in the test set that were classified as an attack.

$$Precision = \frac{TP}{TP + FP} \tag{12}$$

- Recall shows the classifier's completeness, i.e., the rate of samples that correctly classified as an attack for all attack samples that existed in the test set. It is also called Hit, True Positive Rate (TPR), Detection Rate (DR) or Sensitivity.

$$Recall = \frac{TP}{TP + FN} \tag{13}$$

- F1-Score is deemed to be the harmonic mean of both Precision (P) and Recall (R) metrics. It is also known as F1 metric.

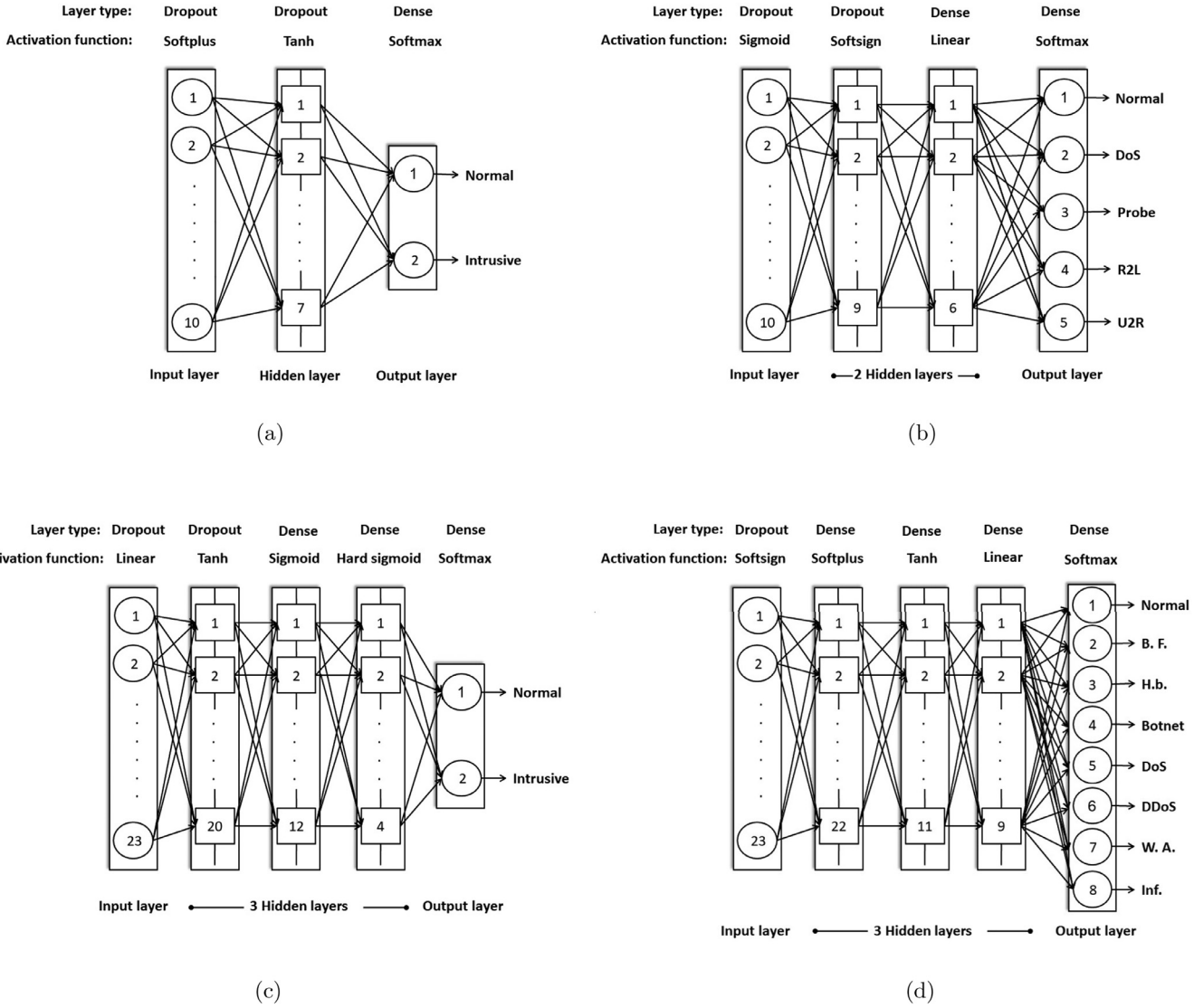

Fig. 6. LSTM-RNN architectures (a) NSL-KDD (binary) (b) NSL-KDD (multiclass) (c) CICIDS2017 (binary) (d) CICIDS2017 (multi-class).

$$F1\_Score = \frac{2 \times P \times R}{P + R} \tag{14}$$

- False Alarm Rate (FAR) is the rate of normal samples that were misclassified to an attack for all normal samples that existed in the test set. It is also called False Positive Rate (FPR).

$$FAR = \frac{FP}{TN + FP} \tag{15}$$

- Specificity shows the rate of normal samples that were correctly predicted for all normal samples that existed in the test set. It is also known as True Negative Rate (TNR).

$$Specificity = \frac{TN}{TN + FP} \tag{16}$$

- False Negative Rate (FNR) is the complement of recall, i.e., gives information about the rate of attack samples that were incorrectly classified as a normal class from all attack samples in the test set. It is also called Miss.

$$FNR = \frac{FN}{TP + FN} \tag{17}$$

- Negative Precision shows the rate of correctly classified normal samples over all samples in the test set that were classified as a normal class.

$$NegativePrecision = \frac{TN}{TN + FN} \tag{18}$$

- Error Rate gives information about the rate of false predictions for all test set.

$$ErrorRate = \frac{FP + FN}{TP + TN + FP + FN} \tag{19}$$

- Bayesian Detection Rate (BDR) is based on Base-Rate Fallacy problem which is firstly addressed by Axelsson [87]. Base-Rate Fallacy is one of the basis of the Bayesian statistics. It occurs when people do not take the basic rate of incidence (Base-Rate) into their account when solving problems in probabilities. Unlike recall metric, BDR indicates the rate of correctly classified attack samples for all test set taking into consideration the base-rate of attack class. Mathematically, let I and I^* denote an intrusive and a normal behavior, respectively. Furthermore, let A and A^* denote the predicted attack and normal behavior, respectively. Then, BDR can be computed as the probability $P(I|A)$ according to formula (20)

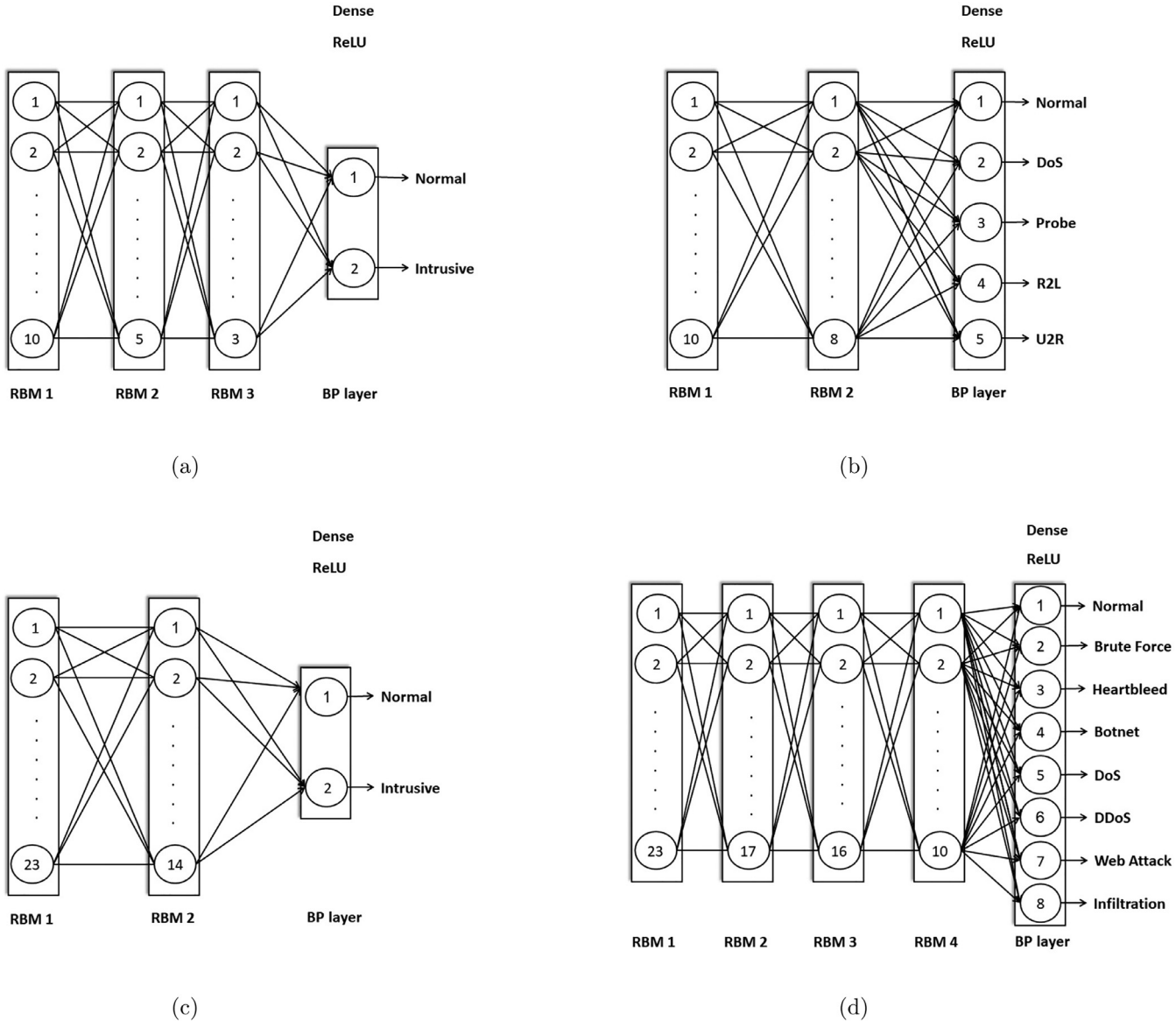

Fig. 7. DBN architectures (a) NSL-KDD (binary) (b) NSL-KDD (multiclass) (c) CICIDS2017 (binary) (d) CICIDS2017 (multi-class).

[87].

$$BDR = P(I|A) = \frac{P(I) \times P(A|I)}{P(I) \times P(A|I) + P(I^*) \times P(A|I^*)} \quad (20)$$

where $P(I)$ is the rate of the attack samples in the test set, $P(A|I)$ is the Recall, $P(I^*)$ is the rate of the normal samples in the test set, and $P(A|I^*)$ is the FAR.

- Bayesian True Negative Rate (BTNR) is also based on the problem of Base-Rate Fallacy. It gives information about the rate of correctly classified normal samples for all test set such that the predicted normal behavior indicates really a normal connection [87]. Mathematically, let I and I^* denote an intrusive and a normal behavior, respectively. Moreover, let A and A^* denote the predicted attack and normal behavior, respectively. Then, BTNR can be computed as the probability $P(I^*|A^*)$ according to formula (21) [87].

$$BTNR = P(I^*|A^*) = \frac{P(I^*) \times P(A^*|I^*)}{P(I^*) \times P(A^*|I^*) + P(I) \times P(A^*|I)} \quad (21)$$

where $P(I^*)$ is the rate of the normal samples in the test set, $P(A^*|I^*)$ is the Specificity, $P(I)$ is the rate of the attack samples in the test set, and $P(A^*|I)$ is the FNR.

- Geometric Mean (g-mean) combines the Specificity and Recall metrics at one specific threshold where both the errors are considered equal. It has been extremely used for evaluating the performance of classifiers on imbalance dataset [88]. Indeed, it has two different formulas. While, g_mean_1 focuses on both the positive and the negative classes [89], g_mean_2 focuses solely on the positive class [90].

$$g\_mean_1 = \sqrt{Recall \times Specificity} \quad (22)$$

$$g\_mean_2 = \sqrt{Recall \times Precision} \quad (23)$$

- Matthews Correlation Coefficient (MCC) is a metric that takes into account all the cells of the confusion matrix in its equation. It is considered as a balanced measure which can be used with imbalance datasets, i.e., even if the classes are of very different sizes [91]. MCC has a range of -1 to 1 where -1 refers to a completely wrong classifier while 1

refers to a completely correct classifier. As calculated using formula (24) [92].

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP + FN) \times (TP + FP) \times (TN + FP) \times (TN + FN)}} \tag{24}$$

- Training time is the time that elapsed for completing the training phase of the model.
- Testing time is the time that elapsed for accomplishing the testing phase of the model.

5.2. Multiclass classification

The confusion matrix of a multiclass classification task is built from the list of predicted classes versus the true classes [93]. It is very useful and intuitive measure where the usefulness of this measure lies in its interpretability. Unlike binary classification, the four major outcomes have a slightly different meaning in multiclass classification. To start with, TN is the number of proper classification of normal samples. In contrast, FP is the sum of all benign traffic instances that misclassified to any of the attack classes. FP can be calculated according to Eq. (25), where n is the number of attack classes, and FP_i is the number of benign traffic instances incorrectly classified to the i^{th} attack class. TP is the sum of all attack samples that truly classified to their proper attack class, as calculated using Eq. (26), where TP_i is the number of accurate predictions of the i^{th} attack class. Finally, FN is the sum of all attack samples that wrongly classified to the normal class. FN can be calculated according to Eq. (27), where FN_i is the number of the i^{th} attack class samples misclassified to the normal class.

$$FP = \sum_{i=1}^{n} FP_i \tag{25}$$

$$TP = \sum_{i=1}^{n} TP_i \tag{26}$$

$$FN = \sum_{i=1}^{n} FN_i \tag{27}$$

Then, the four outcomes are exploited to compute 15 evaluation metrics which are common measures when performing a multiclass classification task. It is worthy to mention that some equations are modulated to adapt with the terminology definition of multiclass classification described above. The list of the evaluation metrics" definition and their corresponding equations is as follows:

- Overall Accuracy is the rate of overall true predictions for all test set.

$$OverallAccuracy = \frac{TP + TN}{TestSetSize} \tag{28}$$

- Average Accuracy is the average per-class effectiveness of a classifier [94].

$$AverageAccuracy = \frac{\sum_{i=1}^{l} \frac{tp_i + tn_i}{tp_i + tn_i + fp_i + fn_i}}{l} \tag{29}$$

where tp_i, fp_i, tn_i, and fn_i are the true positive, false positive, true negative, and false negative for the i^{th} class, respectively, and l is the number of available classes in the dataset.

- Overall Error Rate gives information about the rate of overall false predictions for all test set.

$$OverallErrorRate = \frac{FP + FN}{TestSetSize} \tag{30}$$

- Average Error Rate is the average per-class classification error [94].

$$AverageErrorRate = \frac{\sum_{i=1}^{l} \frac{fp_i + fn_i}{tp_i + tn_i + fp_i + fn_i}}{l} \tag{31}$$

- Macro-Averaged precision is simply the mean of the precision of each individual class.

$$Precision_M = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l} \tag{32}$$

where M index indicates Macro-Averaging.

- Macro-Averaged Recall is the average of the recall of each individual class.

$$Recall_M = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fn_i}}{l} \tag{33}$$

- Macro-Averaged F1-Score is the average of per-class F1-measure [95].

$$F1\_Score_M = \frac{2 \times Precision_M \times Recall_M}{Precision_M + Recall_M} \tag{34}$$

- Micro-Averaged Precision is the precision that computed from the grand total of the numerator and denominator.

$$Precision_\mu = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fp_i)} \tag{35}$$

where μ index presents Micro-Averaging.

- Micro-Averaged Recall is the summation of the dividends and divisors that make up the per-class recall metric to calculate an overall quotient.

$$Recall_\mu = \frac{\sum_{i=1}^{l} tp_i}{\sum_{i=1}^{l} (tp_i + fn_i)} \tag{36}$$

- Micro-Averaged F1-Score is the F1 measure that computed from the individual micro-averaged precision and recall of each class.

$$F1\_Score_\mu = \frac{2 \times Precision_\mu \times Recall_\mu}{Precision_\mu + Recall_\mu} \tag{37}$$

- Missed Rate (MR) is a performance metric for a multiclass classifier that was proposed by Elhamahmy et al. [96]. They defined a new outcome that can be extracted from a multiclass confusion matrix, namely, Misclassification of attack Class (MC). MC determines the number of the particular attack class samples that incorrectly classified to any another attack class. In this case, these wrongly labeled samples could not belong to any of the four main outcomes. MR can be computed using formula (38) [96].

$$MR = \frac{FN + \sum_{i=1}^{l} MC_i}{ActualAttacksSize} \tag{38}$$

where MC_i is the MC of the i^{th} attack class.

- Wrong Rate (WR) is also a performance metric for a multiclass classifier, and based on MC outcome [96]. It is the proportional fraction of incorrectly labeled attack samples to the all samples in the test set that were classified as attacks, and can be computed according to formula (39) [96].

$$WR = \frac{FP + \sum_{i=1}^{l} MC_i}{TP + FP + \sum_{i=1}^{l} MC_i} \tag{39}$$

- F-score Per Cost (FPC) is a new metric for a multiclass classifier, and based on F1-Score, MR, and WR metrics [96]. FPC value varies from 0 to 1, where 0 refers to a completely

Table 6
The main characteristics of the used datasets.

Characteristics		NSL-KDD	CICIDS2017
Year		2009	2017
Audit format		tcpdump	pcap
Number of features		41	80
Number of protocols		6	5
Number of attacks		38	20
Number of Attack categories		4	7
Number of labeled classes		5	8
	Normal	67,343	18,750
	Attacks	58,630	131,250
Distribution of the	Total	125,973	150,000
training set	Normal	9711	18,750
	Attacks	12,833	131,250
Distribution of the test	Total	22,544	150,000
set			

wrong classifier. Otherwise, when it equals 1, it refers to an ideal classifier. As calculated using formula (40) [96].

$$FPC = \frac{F1\_Score}{\sqrt{(F1\_Score)^2 + (Cost)^2}} \tag{40}$$

where

$$Cost = \sqrt{(MR)^2 + (WR)^2}$$

- Training Time and Testing Time metrics are as same as defined in Section 5.1.

6. Datasets

Of importance, there is a need for an effective dataset to measure the effectiveness of a NIDS. An effective dataset means a repository consists of enough amount of reliable data, which reflects the real-world networks. Thus, a quite number of IDS datasets have been created since 1998. In this paper, we selected two different IDS datasets, namely, NSL-KDD and CICIDS2017. Although NSL-KDD dataset is relatively old, it is still deemed to be a well-known benchmark and it is widely used in network intrusion detection field. For the sake of diversity and exploring up-to-date IDS dataset, CICIDS2017 dataset is involved in this study. The upcoming subsections describe the structure of each dataset. Table 6 presents the main characteristics of the used datasets. It is worthy to say that the number of samples in both training and test sets of CICIDS2017 dataset in Table 6 is only 10%.

6.1. NSL-KDD

It is firstly started when MIT Lincoln Laboratory had constructed DARPA dataset [97] in 1998. In short time, it was stated that DARPA is insufficient in network intrusion detection due to its limitation to represent real-world network traffic [98]. Therefore, an updated version of DARPA, namely, KDD CUP 99 is created in 1999 by processing tcpdump portion [99]. Despite 10% of KDD CUP 99 has been widely used in the field of network intrusion detection, it suffers seriously from inherent problems which necessitated the need for a new IDS dataset. Hence, Tavallaee et al. have been proposed in 2009 an improved and reduced version of the 10% of KDD CUP 99 dataset, namely, NSL-KDD [100]. They solved all drawbacks of the 10% of KDD CUP 99 in two ways. First, they eliminated all the redundant records from both training and test sets. Second, they partitioned the records into various difficulty level, then they selected records from each difficulty level inversely proportional to the percentage of records in the original 10% of KDD CUP 99 dataset. As a result, NSL-KDD has a reasonable number of records in both training and test sets, and makes it affordable to run the experiments on the complete set. Further, NSL-KDD dataset is publicly available on the Internet [101].

Regarding the structure of NSL-KDD, every sample is labeled to either normal or an attack record. Basically, there are a total of 38 types of attacks are included in NSL-KDD. In the training set of NSL-KDD, only samples from 24 types of attacks are appeared. In contrast, samples from all types of attacks are appeared in the test set. This to evaluate the effectiveness of the tested NIDS to detect novel attacks which are not appearing in the training set. In addition to that, in order to improve detection rate, similar attacks are combined together into a single category which leads to form four major attacks categories, namely, DoS, Probe, R2L, and U2R. According to that, there are five classes available in NSL-KDD dataset, which are Normal, DoS, Probe, R2L, and U2R. Whereas, NSL-KDD has a total of 125,973 traffic samples in the training set, it has 22,544 samples in its test set. The distribution of samples into each class in its training set is as follows: Normal (67343), DoS (45927), Probe (11656), R2L (995), and U2R (52). On the other hand, the distribution of samples in the test set is as follows: Normal (9711), DoS (7458), Probe (2421), R2L (2887), and U2R (67).

The NSL-KDD has six different network protocols and services such as SMTP, HTTP, FTP, Telnet, ICMP, and SNMP. Finally, it contains 41 features which can be divided into three types, namely, basic features, traffic-based features, and content-based features. The data types of these features are nominal (3), binary (4), and continuous (34).

6.2. CICIDS2017

Canadian Institute for Cybersecurity Intrusion Detection System (CICIDS2017) is modern anomaly-based IDS dataset proposed in 2017 and publicly available on the Internet upon a request from its owners [102]. The purpose behind the creation of this dataset is to release a reliable and up-to-date dataset which contains a realistic data to help researchers to evaluate their models properly. Moreover, it is reported that this dataset overcomes all shortcomings of other existed IDS datasets [103]. They used an environment infrastructure consists of two separate networks, which are Attack-Network and Victim-Network. The Victim-Network is used to provide the benign behavior by using B-Profile system [104]. On the other hand, the Attack-Network is exploited for executing the attack flows. Both of them are equipped with the necessary network devices and PCs running different operating systems. Furthermore, they used CICFlowMeter [102] to analysis the captured pcap data over five working days. The network connection records in this dataset are based on HTTP, HTTPS, FTP, SSH, and email protocols. In addition to that, the attack flows consist of a total of 20 attacks and are grouped into seven major categories, namely, Brute Force, Heartbleed, Botnet, DoS, DDoS, Web attack, and Infiltration. Finally, CICIDS2017 contains 80 different features as well as a class label to identify the particular traffic record to one of eight possible classes.

Unlike NSL-KDD dataset in which there is a specified number of samples for both training and testing, CICIDS2017 is a very huge dataset which has approximately 3 million network flows in different files [102]. Further, in CICIDS2017 dataset, there is no specified training or test sets to be used in the experiments. Therefore, we have selected 10% of CICIDS2017 for training and testing in order to reduce the training and testing time reasonably. Because when using the full size of CICIDS2017, the training and testing times would be too long. The 10% of CICIDS2017 is selected randomly by using the sampling without replacement technique to ensure that once an object is selected, it removed from the population. For the sake of ensuring the diversity of traffic records and avoiding overfitting, we have implemented balanced training and test sets, that is, they are equivalent in size (150 thousand samples in each). The samples in the training set are evenly distributed as follows: Normal (18750), Brute Force (18750), Heartbleed (18750), Botnet (18750), DoS (18750), DDoS (18750), Web attack (18750), and In-

Table 7
The binary classification results of NSL-KDD-experiment.

Metric (%)	DNN		LSTM-RNN		DBN	
	−	+	−	+	−	+
Accuracy	92.18	97.72	94.07	98.8	95.72	**99.79**
Precision	95.77	99.6	97.23	99.7	98.37	**99.83**
Recall	90.25	96.38	92.21	98.18	94.04	**99.81**
F1-Score	92.93	97.96	94.65	98.94	96.16	**99.82**
FAR	5.26	0.51	3.47	0.39	2.06	**0.23**
Specificity	94.74	99.49	96.53	99.61	97.94	**99.77**
FNR	9.75	3.62	7.79	1.82	5.96	**0.19**
Negative Precision	88.03	95.41	90.36	97.65	92.56	**99.74**
Error Rate	7.82	2.28	5.93	1.2	4.28	**0.21**
BDR	95.77	99.6	97.23	99.7	98.37	**99.83**
BTNR	88.03	95.41	90.36	97.65	92.56	**99.74**
g_mean_1	92.47	97.92	94.34	98.89	95.97	**99.79**
g_mean_2	92.97	97.97	94.69	98.94	96.18	**99.82**
MCC	84.39	95.43	88.16	97.57	91.45	**99.57**
Training time (sec)	7938	**740**	8241	936	9059	1552
Testing time (sec)	1421	**132**	1475	168	1621	278

Table 8
The binary classification results of CICIDS2017-experiment.

Metric (%)	DNN		LSTM-RNN		DBN	
	−	+	−	+	−	+
Accuracy	92.92	97.85	94.41	98.83	95.82	**99.91**
Precision	99.5	99.96	99.69	99.98	99.83	**99.99**
Recall	92.38	97.58	93.9	98.68	95.38	**99.92**
F1-Score	95.81	98.76	96.71	99.33	97.56	**99.95**
FAR	3.24	0.28	2.02	0.16	1.11	**0.1**
Specificity	96.76	99.72	97.98	99.84	98.89	**99.9**
FNR	7.62	2.42	6.1	1.32	4.62	**0.08**
Negative Precision	64.45	85.5	69.65	91.55	75.37	**99.41**
Error Rate	7.08	2.15	5.59	1.17	4.18	**0.09**
BDR	99.5	99.96	99.69	99.98	99.83	**99.99**
BTNR	64.45	85.5	69.65	91.55	75.37	**99.41**
g_mean_1	94.54	98.64	95.92	99.26	97.12	**99.91**
g_mean_2	95.87	98.76	96.75	99.33	97.58	**99.95**
MCC	75.5	91.19	79.82	94.96	84.2	**99.61**
Training time (sec)	9310	**1274**	10,043	1492	10,688	1617
Testing time (sec)	4665	**637**	5029	846	5439	915

filtration (18750). Then, the same process is repeated again when creating the test set, but with samples which do not exist in the training data, and are equally distributed as same as in the training set. Furthermore, some types of attacks are included only in the test set rather than in the training set to examine the ability of the used NIDS to classify them correctly. According to this procedure, we believe that both training and test data are balanced and the used models will not bias to any class during the training phase.

7. Results and discussion

The double PSO-based algorithm was implemented using Python programming language version 3.6.4 [105] with NumPy library [106]. Further, we deployed all deep learning models by using Keras open source neural network library [107,108] over TensorFlow machine learning framework version 1.6 [109,110] with CUDA integrated backend version 9.0 [111] and cuDNN version 7.0 [112]. The hardware environment is as follows: Intel Core i7 CPU (3.8 GHz, 16 MB Cache), 16 GB of RAM, and the Windows 10 operating system (64-bit mode). In order to accelerate the computations and improve efficiency, the GPU-accelerated computing with NVIDIA Tesla K20 GPU 5 GB GDDR5 is also utilized. In the following subsections, we present our experimental results and discuss them via three different analyses such as performance analysis, Friedman statistical test, and ranking methods.

7.1. Performance analysis

The key component of the effectiveness of a model in intrusion detection based on its score of evaluation metrics. The higher values in Accuracy, Precision, Recall, and F1-Score, as well as the lower values of FNR, Error Rate, and FAR indicate an efficient classifier. The ideal classifier has Accuracy and Recall values reach 1, as well as FNR and FAR values reach 0.

7.1.1. Binary classification

Tables 7 and 8 present the values of the evaluation metrics for NSL-KDD-experiment and CICIDS2017-experiment, respectively. In fact, all values are presented in the percentage format except Training time and Testing time in seconds. Moreover, the bold values represent the best results among the same dataset. To show performance differences, the columns with minus sign (-) in Tables 7 and 8 refer to the results of the model without using our approach (full feature set), meanwhile the columns with plus sign (+) refer to the results of the model when applying a pre-training phase using the proposed double PSO-based algorithm.

Regarding datasets, all used models gained higher performance on CICIDS2017 than on NSL-KDD. This due to the fact that CICIDS2017 dataset is modern form of IDS datasets which is more reliable than NSL-KDD dataset. However, the training and testing times of all models recorded for NSL-KDD is less than the corresponding values for CICIDS2017. This can be explained by the fact that NSL-KDD dataset has a small number of features (41 for full-featured, 10 for feature subset) compared to CICIDS2017 (80 for full-featured, 23 for feature subset). Also, the number of samples in the training and test sets of CICIDS2017 is larger than those in NSL-KDD dataset.

As we expected, the deep learning models with pre-training phase outperformed the same models without pre-training in terms of all metrics and for all datasets. This was possible due to the impact of using the double PSO-based algorithm in the pre-training phase of deep learning models. As described in Section 3, the double PSO-based algorithm generates the reduced dataset with optimal feature subset which simplifies the structure of the model. In addition to that, the double PSO-based algorithm attempts to select the optimal hyperparameters of the model that maximizes accuracy on the given dataset. According to Eq. (11), as the accuracy value increases, the sum of TP and TN in the numerator will be increased significantly since the denominator is constant. Therefore, this yields to definitely increase the value of classification metrics such as recall, as well as to decrease the value of misclassification metrics such as false alarm rate. According to the findings, the deep learning models with pre-training increased the recall metric by 4% to 6% and decreased the false alarm rate metric by 1% to 5% from the corresponding values of same models without pre-training on the same dataset.

At a glance, LSTM-RNN models performed superior than DNN models in terms of all evaluation metrics regarding all datasets. The fact behind this culminate of the results that instead of using the traditional artificial neurons in all hidden layers, the LSTM-RNN model uses LSTM memory cells. Moreover, the LSTM-RNN models have extra hidden-to-hidden connections, that is, among the processing elements in the same hidden layer. Thus, these built-in characteristics of the LSTM-RNN models which do not exist in DNN models, enable the classifier to memorize the previous states, explore the dependencies among them, and finally use them along with current inputs to predict the output. Meanwhile, the difference between the performance of LSTM-RNN and DNN models on all datasets is significant for recall metric which is between 1% and

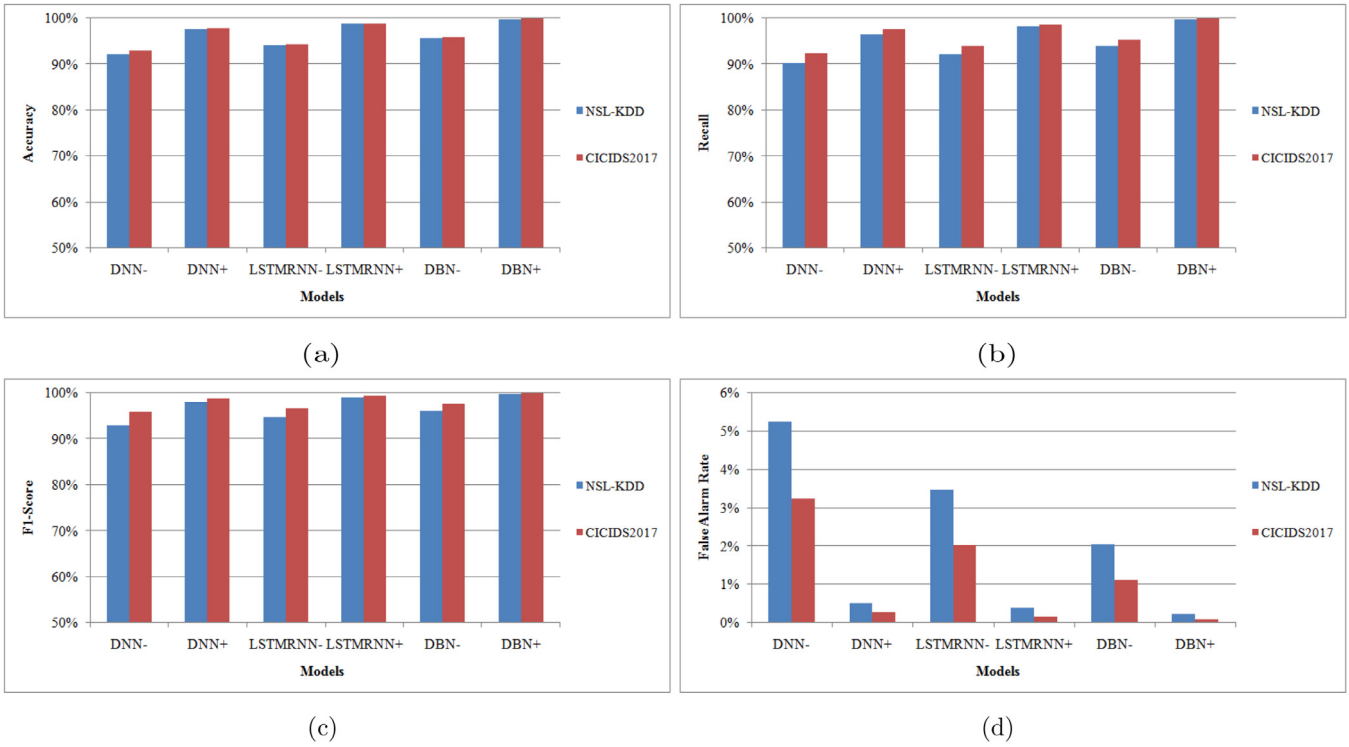

Fig. 8. The standard metrics comparison between models on each dataset. (a) Accuracy (b) Recall (c) F1-Score (d) FAR.

2%, but it is very small for false alarm rate metric which is between 0.1% and 0.2% in all cases.

Although DNN models scored the best training and testing times, the experimental results demonstrated the superiority of DBN models over other deep learning models on all datasets. This due to the nature of DBN's deep structure where consists of several RBMs followed by a fully-connected BP layer. The RBMs try to learn the useful features from the inputs by using their hidden and visible layers. During the learning process, the RBMs are pre-trained in an unsupervised manner. Then, the shorten extracted features are fed to a BP layer which is fine-tuned with the whole network by using back-propagation in a supervised manner. Hence, this functionality makes DBN as a robust classifier that able to deeply understand the underlying task. The DBN models gained excellent results on all datasets such that recall between 99.81% and 99.92% and false alarm rate between 0.1% and 0.23%. In spite of that, DNN and LSTM-RNN models also performed very well in network intrusion detection on the used datasets.

For the sake of brevity and space limitation, we selected only the standard classification metrics to be presented visually in Fig. 8. Fig. 8(a), 8(b), 8(c), and 8(d) show Accuracy, Recall, F1-Score, and FAR percentages of all models in each dataset, respectively. Whereas in Fig. 8, the notation $(M-)$ denotes a deep learning model M without pre-training phase applied on the full-featured dataset, $(M+)$ denotes a deep learning model M with pre-training using the proposed algorithm. Fig. 8 can give us a visual comparison of the performance of all models on each dataset.

7.1.2. Multiclass classification

Tables 9 and 10 present the values of the evaluation metrics for NSL-KDD-experiment and CICIDS2017-experiment, respectively. All values are presented in the percentage format except Training time and Testing time in seconds. The minus and plus signs in Tables 9 and 10 have the same meaning as in Tables 7 and 8.

In like manner, NSL-KDD dataset has an imbalanced structure. The imbalanced dataset typically refers to a problem with classifi-

Table 9
The multiclass classification results of NSL-KDD-experiment.

Metric (%)	DNN		LSTM-RNN		DBN	
	−	+	−	+	−	+
Overall Accuracy	73.35	90.63	74.68	93.6	86.53	**96.91**
Average Accuracy	89.34	96.25	89.87	97.44	94.61	**98.77**
Overall Error Rate	25.09	7.84	23.09	5.49	12.03	**2.45**
Average Error Rate	10.66	3.75	10.13	2.56	5.39	**1.23**
$Precision_M$	83.37	93.86	83.3	95.85	91.39	**98.1**
$Recall_M$	47.61	80.61	50.4	86.19	69.82	**92.29**
$F1\_Score_M$	60.61	86.73	62.81	90.76	79.16	**95.11**
$Precision_\mu$	73.35	90.63	74.68	93.6	86.53	**96.91**
$Recall_\mu$	73.35	90.63	74.68	93.6	86.53	**96.91**
$F1\_Score_\mu$	73.35	90.63	74.68	93.6	86.53	**96.91**
MR	43.53	14.81	42.37	10.38	21.85	**5.07**
WR	9.62	4.83	9.47	2.68	5.27	**1.53**
FPC	84.59	98.57	85.69	99.36	96.81	**99.85**
Training time (sec)	9452	**881**	9813	1115	10,787	1848
Testing time (sec)	1692	**157**	1756	200	1930	331

Table 10
The multiclass classification results of CICIDS2017-experiment.

Metric (%)	DNN		LSTM-RNN		DBN	
	−	+	−	+	−	+
Overall Accuracy	76.19	88.04	78.81	92.41	82	**95.81**
Average Accuracy	94.05	97.01	94.7	98.1	95.5	**98.95**
Overall Error Rate	6.29	2.34	5.25	1.23	4.03	**0.58**
Average Error Rate	5.95	2.99	5.3	1.9	4.5	**1.05**
$Precision_M$	76.35	88.08	78.92	92.44	82.08	**95.82**
$Recall_M$	76.19	88.04	78.81	92.41	82	**95.81**
$F1\_Score_M$	76.27	88.06	78.87	92.43	82.04	**95.81**
$Precision_\mu$	76.19	88.04	78.81	92.41	82	**95.81**
$Recall_\mu$	76.19	88.04	78.81	92.41	82	**95.81**
$F1\_Score_\mu$	76.19	88.04	78.81	92.41	82	**95.81**
MR	24.33	12.76	21.9	8.21	18.89	**4.63**
WR	23.24	12.02	20.81	7.77	17.88	**4.3**
FPC	92.94	98.26	94.42	99.31	95.96	**99.79**
Training time (sec)	13,950	**1911**	150,645	2238	16,032	2426
Testing time (sec)	6998	**956**	7544	1269	8156	1373

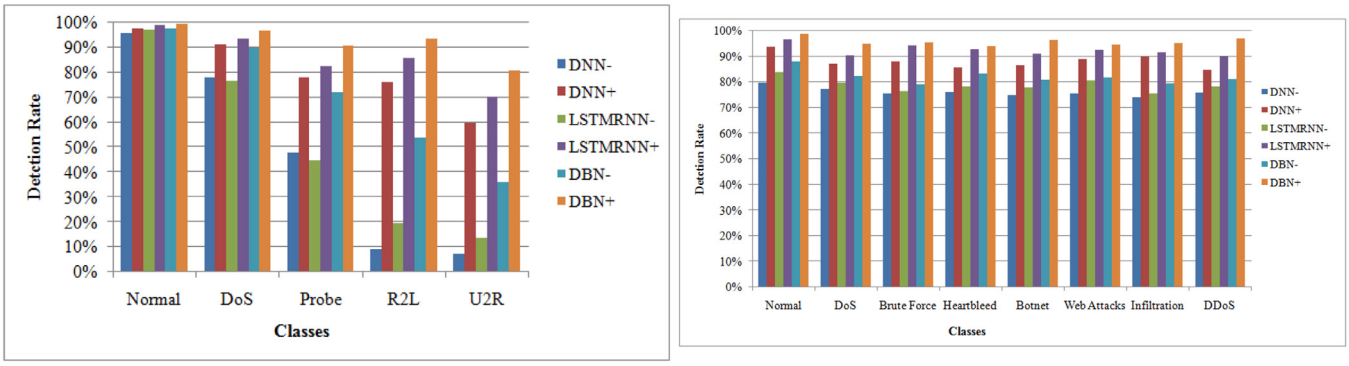

Fig. 9. The detection rate of models for each class in the dataset. (a) NSL-KDD (b) CICIDS2017.

cation tasks where the entire classes are not represented equally. In the case of NSL-KDD dataset, the test set distribution is as follows 43%, 33%, 10.7%, 13%, and 0.3% for Normal, DoS, Probe, R2L, and U2R classes, respectively. Therefore, the learning models tend to classes with the vast majority of samples rather than with ones of the small minority. This leads to, NSL-KDD dataset is more suitable to binary classification, where there is a small difference between Normal (43%) and Attack (57%) classes, rather than to multiclass classification. On the other hand, we have used a balanced structure of CICIDS2017 dataset which enhanced the results in most cases.

As a matter of fact, the accuracy value is only reflecting the underlying class distribution in the dataset. This condition is also known as the accuracy paradox where the accuracy value does not reflect the exact performance of the model. In general, micro-averaged metrics are more preferable to be used with imbalanced datasets, that is, when we want to bias all the classes towards the frequent class. In contrast, the macro-averaged metrics are useful with balanced datasets and we want to put the same emphasis on all the classes by giving equal weight to each individual class. However, the training and test times' values in the results of multiclass classification are higher than the corresponding values in binary classification. This due to the architecture of the models in multiclass classification is more complex than those in binary classification.

Obliviously, our findings confirmed that the models with pre-training phase outperformed the models without pre-training in all metrics and for all datasets. Figs. 9(a) and 9(b) depict the detection rate for each entire class in NSL-KDD and CICIDS2017 datasets, respectively. According to Fig. 9, our proposed approach not only enhanced the performance of the models compared to the models without using our approach, but also increased the detection rate of each class significantly.

7.1.3. Double PSO vs. PSO

In this subsection, we present an empirical comparison between double PSO and PSO to justify the reason behind using double PSO rather than one PSO in our experiments. Indeed, our proposed approach utilizes a double PSO-based algorithm in the pre-training phase of the deep learning model to execute two main processes: feature and hyperparameter selection. Thus, it is worthy to show performance differences between using double PSO and using one PSO for either feature selection or hyperparameter selection in the pre-training phase. Firstly, Table 11 compares between our results and the results of some related works in Section 2 that adopted optimization techniques for only feature selection.

We can notice easily that our deep learning models especially DBN outperformed all the listed models in Table 11. Despite the studies of Tang et al.[6] and Ahmad [7] had generated smaller feature subsets than our proposed double PSO-based algorithm, but the selected features in our study improved the overall performance of the models in both binary and multiclass classifications. Secondly, Table 12 presents a comparison between the results of double PSO and the results of PSO when the latter is used only for hyperparameter selection in the pre-training phase. The bold values in Table 12 refer to the best score among the same dataset.

As we can notice from Table 12, the performance of double PSO outperformed PSO in terms of all metrics and for all datasets. Finally, based on our findings, we can highlight the importance of using the proposed double PSO-based algorithm in the pre-training phase of the model which will lead to improve the performance of the model significantly in network intrusion detection.

7.1.4. Comparison to other previous works

In this subsection, we provide an indicative comparative analysis to other previous studies in Section 2 which are not used pre-training phase for feature or hyperparameter selection. Table 13 shows the performance differences between our models with proposed double PSO-based algorithm and some previous studies without pre-training phase.

From Table 13, we can demonstrate the superiority of our proposed approach over all listed previous studies in terms of performance in intrusion detection. This due to the existence of the pre-training phase that helps to select the optimal features and hyperparameters just before the training phase. However, our proposed approach has a drawback that it increases the time needed for training and testing, but according to findings in Tables 7–10 the recorded training and testing times of our models are still acceptable (not too long).

7.2. Friedman test

The Friedman test is a well-known non-parametric statistical test for discovering the differences between several repeated treatments [113]. Non-parametric means the test does not assume your data comes from a particular distribution. In this study, we have two related treatments ($k = 2$) each for one of the datasets, and six subjects ($Z = 6$) in every treatment in such a way that each subject is related to one of the models. The null hypothesis for the Friedman test is that all the treatments have identical effects, that is, there are no differences between the treatments. Mathematically, we can reject the null hypothesis if and only if the calculated Friedman test statistic (FS) is larger than the critical Friedman test value (FC), and the calculated probability ($P - value$) is less than the selected significance level (α). Table 14 shows the results of the Friedman test for TP, TN, FP, and FN measurements. In all tests,

Table 11
Comparison between double PSO and some related works for feature selection of NSL-KDD dataset .

Study	Optimization algorithm	Model	No. of features	DR (%)	FAR (%)
[6]	Manual	DNN	6	76	–
[7]	PCA	MNN	20	96.6	3.4
	GA	MNN	10	98.2	1.8
	PSO	MNN	8	99.4	0.6
[8]	AR	DT	22	–	–
[9]	CFS+IG	Naive Bayes	13	–	4.2
[10]	Manual	CART	29	88.23	–
[11]	hybrid Bi-Layer behavioral-based method	–	20	–	–
[12]	Mutual information	LSSVM	18	98.76	0.28
[14]	K-means + GA	–	19	–	–
[15]	Univariate method with a recursive feature elimination	DT	12	99.79	–
[16]	SVM	SVM	36	82	15
Our	Double	DNN	10	96.38	0.51
study	PSO	LSTM-RNN	10	98.18	0.39
		DBN	10	**99.81**	**0.23**

Table 12
Comparison between double PSO and PSO when it is used only for hyperparameter selection.

Dataset	Optimization algorithm	Model	DR (%)	FAR (%)
NSL-KDD	PSO	DNN	92.22	2.4
		LSTM-RNN	95.36	1.66
		DBN	98.8	1.55
	Double	DNN	96.38	0.51
	PSO	LSTM-RNN	98.18	0.39
		DBN	**99.81**	**0.23**
CICIDS2017	PSO	DNN	94	1.9
		LSTM-RNN	95.38	1.3
		DBN	96.68	0.8
	Double	DNN	97.58	0.28
	PSO	LSTM-RNN	98.68	0.16
		DBN	**99.92**	**0.1**

Table 13
Comparison between our results and previous studies without pre-training phase.

Study	Technique	DR (%)	FAR (%)
[27]	Non-symmetric AE	85.42	14.58
[28]	Accelerated DNN+AEs+Softmax layer	97.5	3.5
[29]	RNN	72.95	3.44
[31]	DBN+ELM	91.8	–
[32]	AE+DBN+DNN+ELM	97.95	14.72
Our	Double PSO+DNN	96.38	0.51
study	Double PSO+LSTM-RNN	98.18	0.39
	Double PSO+DBN	**99.81**	**0.23**

Table 14
The results of the Friedman test for each outcome ($\alpha = 0.05$).

Measurement	FS	FC	P-value
TP	12	7	0.01431
TN	12	7	0.01431
FP	9.33	7	0.0094
FN	12	7	0.01431

we have selected the significance level equals 0.05 because it is fairly common.

Accordingly, we rejected the null hypothesis of the Friedman test, because in all cases ($FS > FC$) and ($P - value < \alpha$). Hence, it can be concluded that the scores of the models for each measurement are significantly different from each other. In order to interpret the results of the Friedman test visually, we also plotted the Critical Difference Diagram [114]. Fig. 10 presents the Critical Difference Diagram of the used models on all datasets. Finally, we obtained the Critical Difference (CD) value which is shown as a bar above the figure equals 5.3319.

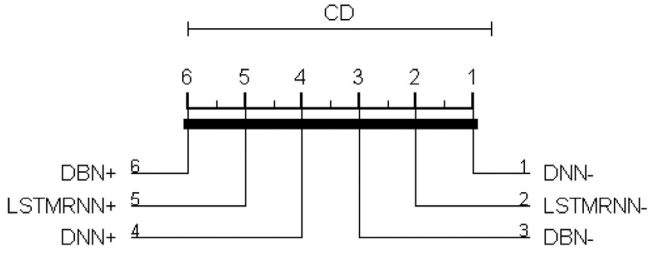

Fig. 10. The Critical Difference Diagram of the used models on all datasets.

7.3. Ranking methods

The ranking methods are usually utilized to show the trade-off between different machine learning algorithms as well as to evaluate the performance of them in order to choose the optimal classifier. The main advantages of the ranking methods are enabling visualization or summarization performance over the classifier's full operating range, and highlighting the information related to skew of the data [90]. In this paper, we have selected two popular ranking methods, namely, Receiver Operating Characteristic (ROC) curves and Precision-Recall (PR) curves. The major difference between them is PR curves give a more informative picture of the model's performance when dealing with imbalanced datasets [115]. In other words, PR curves are better for a class imbalance problem. Otherwise, ROC curves are better for balanced datasets.

At the outset, a ROC curve is a plot of the true positive rate (or Recall) as a function of the false positive rate (or FAR) of a classifier [116]. The diagonal line of ROC is deemed to be a reference line which indicates 50% of performance is achieved. The top-left corner of ROC refers to the best performance with 100%. Fig. 11 shows ROC curves of the tested models on CICIDS2017 dataset.

The Area Under the ROC Curve (AUC-ROC or simply AUROC) is a widely used measure to compare quantitatively between various ROC curves [117]. AUROC value summarizes the corresponding ROC curve into a single value between 0 and 1. The optimal classifier will have AUROC value equals 1. Table 15 presents AUROC values of ROC curves which are plotted in Fig. 11.

On the other hand, an PR curve is a plot of the precision on Y-axis versus the recall on X-axis. The ideal model gives an PR curve at the upper-right corner, which means that this model got only the true positives with no false positives and no false negatives [115]. Fig. 12 depicts PR curves of the tested models on NSL-KDD dataset.

Finally, the Area Under the PR Curve (AUC-PR or simply AUPR) is the area under the PR curve of a model [118]. Basically, the AUPR value is in the range of [0,1] and the perfect classifier has an AUPR

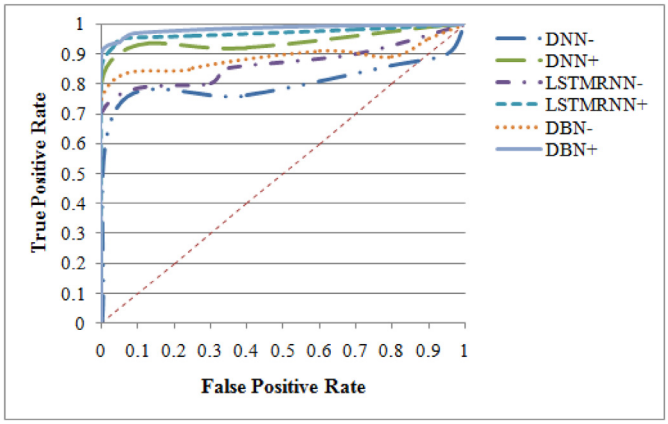

Fig. 11. ROC curves of the models over CICIDS2017 dataset.

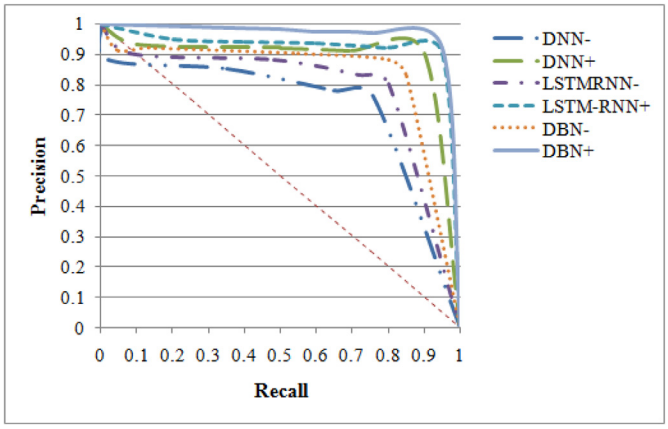

Fig. 12. PR curves of the models over NSL-KDD dataset.

Table 15
AUROC values of ROC curves.

Model	AUROC
DNN−	0.9457
DNN+	0.9865
LSTM_RNN−	0.9594
LSTM_RNN+	0.9926
DBN−	0.9713
DBN+	0.9991

Table 16
AUPR values of PR curves.

Model	AUPR
DNN−	0.9301
DNN+	0.9799
LSTM_RNN−	0.9472
LSTM_RNN+	0.9894
DBN−	0.962
DBN+	0.9982

value equals 1. Table 16 presents AUPR values of PR curves which are plotted in Fig. 12. ROC and PR curves show that models in the order: DBN, LSTM-RNN, and DNN have the effective performance in network intrusion detection over all datasets. However, all the used models still have a pretty good fit.

8. Conclusion

In this paper, we presented a comprehensive empirical study on network intrusion detection using three deep learning models: DNN, LSTM-RNN, and DBN. These models are pre-trained by leveraging a double PSO-based metaheuristic algorithm. The revitalized algorithm consists of two levels: the upper level where the optimal feature subset is selected for the given dataset, following that in the lower level, the vector of optimized hyperparameters is determined automatically to maximize accuracy over the reduced dataset. We utilized two common well-known IDS datasets in our experiments, namely, NSL-KDD and CICIDS2017. The first one is considered as a benchmark and widely used in the literature, whereas the latter is the most up-to-date reliable NIDS dataset. Further, we carried out our experiments in two different perspectives, the binary, and multiclass classifications. In order to give a deep view of models' performance, we have evaluated the experimental results using performance analysis, Friedman statistical test, and two ranking methods. The experimental results reveal that our approach performed an achievement in network intrusion detection compared to the models without having the pre-training phase. Moreover, our deep learning approach outperformed the performance of deep learning models in the previous studies as well as proved its ability to enhance the accuracy and detection rate by 4% to 6% as well as decrease the false alarm rate by 1% to 5% in most cases.

Funding statement

The authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speakersâ bureaus; membership, employment, consultancies, stock ownership, or other equity interest; and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Declaration of Competing Interest

None.

References

[1] T. Mahmood, U. Afzal, Security analytics: Big data analytics for cybersecurity: A review of trends, techniques and tools, in: 2013 2nd National Conference on Information Assurance (NCIA), IEEE, 2013, pp. 129–134.
[2] N. Marir, H. Wang, G. Feng, B. Li, M. Jia, Distributed abnormal behavior detection approach based on deep belief network and ensemble svm using spark, IEEE Access 6 (2018) 59657–59671.
[3] B. Dong, X. Wang, Comparison deep learning method to traditional methods using for network intrusion detection, in: 2016 8th IEEE International Conference on Communication Software and Networks (ICCSN), IEEE, 2016, pp. 581–585.
[4] L. Deng, A tutorial survey of architectures, algorithms, and applications for deep learning, APSIPA Trans. Signal Inform. Process. 3 (2014).
[5] X. Du, Y. Cai, S. Wang, L. Zhang, Overview of deep learning, in: 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), IEEE, 2016, pp. 159–164.
[6] T.A. Tang, L. Mhamdi, D. McLernon, S.A.R. Zaidi, M. Ghogho, Deep learning approach for network intrusion detection in software defined networking, in: 2016 International Conference on Wireless Networks and Mobile Communications (WINCOM), IEEE, 2016, pp. 258–263.
[7] I. Ahmad, Feature selection using particle swarm optimization in intrusion detection, Int. J. Distrib. Sens. Netw. 11 (10) (2015) 806954.
[8] H. Chae, B. Jo, S.-H. Choi, T. Park, Feature selection for intrusion detection using nsl-kdd, Recent Adva. Comput. Sci. (2013) 184–187.
[9] Y. Wahba, E. ElSalamouny, G. ElTaweel, Improving the performance of multiclass intrusion detection systems using feature reduction, Int. Jour. of Comp. Sci. 12 (3) (2015) 255–262.
[10] H.P.S. Sasan, M. Sharma, Intrusion detection using feature selection and machine learning algorithm with misuse detection, Int. J. Comput. Sci. Inform. Technol. (IJCSIT) 8 (1) (2016) 17–25.
[11] H.F. Eid, M.A. Salama, A.E. Hassanien, A feature selection approach for network intrusion classification: the bi-layer behavioral-based, Int. J. Comput. Vis. Image Proces. (IJCVIP) 3 (4) (2013) 51–59.

[12] M.A. Ambusaidi, X. He, P. Nanda, Z. Tan, Building an intrusion detection system using a filter-based feature selection algorithm, IEEE Trans. Comput. 65 (10) (2016) 2986–2998.
[13] S. Ustebay, Z. Turgut, M.A. Aydin, Intrusion detection system with recursive feature elimination by using random forest and deep learning classifier, in: 2018 International Congress on Big Data, Deep Learning and Fighting Cyber Terrorism (IBIGDELFT), IEEE, 2018, pp. 71–76.
[14] T. Naidoo, A. Mcdonald, J.-R. Tapamo, Feature selection for anomaly–based network intrusion detection using cluster validity indices, in: http://www.satnac. org. za, 2015, pp. 1–6.
[15] H. Nkiama, S.Z.M. Said, M. Saidu, A subset feature elimination mechanism for intrusion detection system, Int. J. Adv. Comput. Sci. Appl. 7 (4) (2016) 148–157.
[16] M.S. Pervez, D.M. Farid, Feature selection and intrusion classification in nsl-kdd cup 99 dataset employing svms, in: The 8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA 2014), IEEE, 2014, pp. 1–6.
[17] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, P. Yogesh, A. Kannan, Intelligent feature selection and classification techniques for intrusion detection in networks: a survey, EURASIP J. Wirel. Commun. Netw. 2013 (1) (2013) 271.
[18] Y. Wang, W. Fu, D.P. Agrawal, Gaussian versus uniform distribution for intrusion detection in wireless sensor networks, IEEE Trans. Parallel Distrib. Syst. 24 (2) (2012) 342–355.
[19] H. Sedjelmaci, S.M. Senouci, M. Feham, An efficient intrusion detection framework in cluster-based wireless sensor networks, Secur. Commun. Netw. 6 (10) (2013) 1211–1224.
[20] I. Ahmad, F. e Amin, Towards feature subset selection in intrusion detection, in: 2014 IEEE 7th Joint International Information Technology and Artificial Intelligence Conference, IEEE, 2014, pp. 68–73.
[21] A. Hassanzadeh, A. Altaweel, R. Stoleru, Traffic-and-resource-aware intrusion detection in wireless mesh networks, Ad Hoc Netw. 21 (2014) 18–41.
[22] R.C. Staudemeyer, C.W. Omlin, Extracting salient features for network intrusion detection using machine learning methods, S. Afr. Comput. J. 52 (1) (2014) 82–96.
[23] Z.A. Othman, Z. Muda, L.M. Theng, M.R. Othman, Record to record feature selection algorithm for network intrusion detection, Int. J. Adv. Comput. Technol. 6 (2) (2014) 163.
[24] A.S. Eesa, Z. Orman, A.M.A. Brifcani, A novel feature-selection approach based on the cuttlefish optimization algorithm for intrusion detection systems, Expert Syst. Appl. 42 (5) (2015) 2670–2679.
[25] Z. Alom, V.R. Bontupalli, T.M. Taha, Intrusion detection using deep belief network and extreme learning machine, Int. J. Monitor. Surveill. Technol. Res. (IJMSTR) 3 (2) (2015) 35–56.
[26] A. Javaid, Q. Niyaz, W. Sun, 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), ICST (Institute for Computer Sciences, Social-Informatics and â Q, 2016, pp. 21–26.
[27] N. Shone, T.N. Ngoc, V.D. Phai, Q. Shi, A deep learning approach to network intrusion detection, IEEE Trans. Emerg. Top. Comput. Intell. 2 (1) (2018) 41–50.
[28] S. Potluri, C. Diedrich, Accelerated deep neural networks for enhanced intrusion detection system, in: 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA), IEEE, 2016, pp. 1–8.
[29] C. Yin, Y. Zhu, J. Fei, X. He, A deep learning approach for intrusion detection using recurrent neural networks, IEEE Access 5 (2017) 21954–21961.
[30] M.Z. Alom, V. Bontupalli, T.M. Taha, Intrusion detection using deep belief networks, in: 2015 National Aerospace and Electronics Conference (NAECON), IEEE, 2015, pp. 339–344.
[31] Y. Liu, X. Zhang, Intrusion detection based on idbm, in: 2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), IEEE, 2016, pp. 173–177.
[32] S.A. Ludwig, Intrusion detection of multiple attack classes using a deep neural net ensemble, in: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2017, pp. 1–7.
[33] F. Qu, J. Zhang, Z. Shao, S. Qi, An intrusion detection model based on deep belief network, in: Proceedings of the 2017 VI International Conference on Network, Communication and Computing, ACM, 2017, pp. 97–101.
[34] E.E. Tsiropoulou, J.S. Baras, S. Papavassiliou, G. Qu, On the mitigation of interference imposed by intruders in passive rfid networks, in: International Conference on Decision and Game Theory for Security, Springer, 2016, pp. 62–80.
[35] S. Raza, L. Wallgren, T. Voigt, Svelte: real-time intrusion detection in the internet of things, Ad Hoc Netw. 11 (8) (2013) 2661–2674.
[36] W. Elmasry, A. Akbulut, A.H. Zaim, Empirical study on multiclass classification-based network intrusion detection, Comput. Intell. (2019) 1–36, doi:10.1111/coin.12220.
[37] M. Dash, H. Liu, Feature selection for classification, Intell. Data Analy. 1 (1–4) (1997) 131–156.
[38] C.-S. Yang, L.-Y. Chuang, J.-C. Li, C.-H. Yang, Chaotic maps in binary particle swarm optimization for feature selection, in: 2008 IEEE Conference on Soft Computing in Industrial Applications, IEEE, 2008, pp. 107–112.
[39] L. Oliveira, R. Sabourin, F. Bortolozzi, C. Suen, Feature selection using multiobjective genetic algorithms for handwritten digit recognition, in: International Conference on Pattern Recognition, 16, 2002, pp. 568–571.
[40] K. Waqas, R. Baig, S. Ali, Feature subset selection using multi-objective genetic algorithms, in: 2009 IEEE 13th International Multitopic Conference, IEEE, 2009, pp. 1–6.
[41] G.L. Azevedo, G.D. Cavalcanti, E.C. Carvalho Filho, An approach to feature selection for keystroke dynamics systems based on pso and feature weighting, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 3577–3584.
[42] S.-W. Lin, K.-C. Ying, S.-C. Chen, Z.-J. Lee, Particle swarm optimization for parameter determination and feature selection of support vector machines, Expert Syst. Appl. 35 (4) (2008) 1817–1824.
[43] A.W. Mohemmed, M. Zhang, M. Johnston, Particle swarm optimization based adaboost for face detection, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 2494–2501.
[44] K. Neshatian, M. Zhang, P. Andreae, A filter approach to multiple feature construction for symbolic learning classifiers using genetic programming, IEEE Trans. Evolut. Comput. 16 (5) (2012) 645–661.
[45] A. Purohit, N.S. Chaudhari, A. Tiwari, Construction of classifier with feature selection based on genetic programming, in: IEEE Congress on Evolutionary Computation, IEEE, 2010, pp. 1–5.
[46] L. Ke, Z. Feng, Z. Xu, K. Shang, Y. Wang, A multiobjective aco algorithm for rough feature selection, in: 2010 Second Pacific-Asia Conference on Circuits, Communications and System, 1, IEEE, 2010, pp. 207–210.
[47] R.K. Sivagaminathan, S. Ramakrishnan, A hybrid approach for feature subset selection using neural networks and ant colony optimization, Expert Syst. Appl. 33 (1) (2007) 49–60.
[48] C.-L. Huang, J.-F. Dun, A distributed pso–svm hybrid system with feature selection and parameter optimization, Appl. Soft Comput. 8 (4) (2008) 1381–1391.
[49] J. Kennedy, W.M. Spears, Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator, in: 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360), IEEE, 1998, pp. 78–83.
[50] R. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science, Ieee, 1995, pp. 39–43.
[51] J. Kenedy, R. Eberhart, A discrete binary version of the particle swarm optimization, Comput. Cybern. Simul. 5 (1) (1997) 4104–4108.
[52] Z. Zhou, X. Liu, P. Li, L. Shang, Feature selection method with proportionate fitness based binary particle swarm optimization, in: Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2014, pp. 582–592.
[53] L. Cervante, B. Xue, M. Zhang, L. Shang, Binary particle swarm optimisation for feature selection: A filter based approach, in: 2012 IEEE Congress on Evolutionary Computation, IEEE, 2012, pp. 1–8.
[54] P. Langley, Selection of relevant features in machine learning, in: Proceedings of the AAAI Fall symposium on relevance, 1994, pp. 1–5.
[55] B. Tran, B. Xue, M. Zhang, Overview of particle swarm optimisation for feature selection in classification, in: Asia-Pacific conference on simulated evolution and learning, Springer, 2014, pp. 605–617.
[56] B. Chakraborty, Feature subset selection by particle swarm optimization with fuzzy fitness function, in: 2008 3rd international conference on intelligent system and knowledge engineering, 1, IEEE, 2008, pp. 1038–1042.
[57] S. Subbotin, A. Oleynik, The multi objective evolutionary feature selection, in: 2008 International Conference on" Modern Problems of Radio Engineering, Telecommunications and Computer Science"(TCSET), IEEE, 2008, pp. 115–116.
[58] J.S. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: Advances in neural information processing systems, 2011, pp. 2546–2554.
[59] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (Feb) (2012) 281–305.
[60] J. Snoek, H. Larochelle, R.P. Adams, Practical bayesian optimization of machine learning algorithms, in: Advances in neural information processing systems, 2012, pp. 2951–2959.
[61] O.A. Abdalla, A.O. Elfaki, Y.M. AlMurtadha, Optimizing the multilayer feed–forward artificial neural networks architecture and training parameters using genetic algorithm, Int. J. Comput. Appl. 96 (10) (2014) 42–48.
[62] S. Belharbi, R. Hérault, C. Chatelain, S. Adam, Deep multi-task learning with evolving weights, in: European Symposium on Artificial Neural Networks (ESANN), 2016, 2016, pp. 1–6.
[63] S.S. Tirumala, S. Ali, C.P. Ramesh, Evolving deep neural networks: A new prospect, in: 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), IEEE, 2016, pp. 69–74.
[64] O.E. David, I. Greental, Genetic algorithms for evolving deep neural networks, in: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, 2014, pp. 1451–1452.
[65] A. Martín, F. Fuentes-Hurtado, V. Naranjo, D. Camacho, Evolving deep neural networks architectures for android malware classification, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1659–1666.
[66] P.R. Lorenzo, J. Nalepa, M. Kawulok, L.S. Ramos, J.R. Pastor, Particle swarm optimization for hyper-parameter selection in deep neural networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2017, pp. 481–488.
[67] P.R. Lorenzo, J. Nalepa, L.S. Ramos, J.R. Pastor, Hyper-parameter selection in deep neural networks using parallel particle swarm optimization, in: Pro-

ceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, 2017, pp. 1864–1871.
[68] J. Nalepa, P.R. Lorenzo, Convergence analysis of pso for hyper-parameter selection in deep neural networks, in: International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, Springer, 2017, pp. 284–295.
[69] F. Ye, Particle swarm optimization-based automatic parameter selection for deep neural networks and its applications in large-scale and high-dimensional data, PloS one 12 (12) (2017) e0188746.
[70] H.J. Escalante, M. Montes, L.E. Sucar, Particle swarm model selection, J. Mach. Learn. Res. 10 (Feb) (2009) 405–440.
[71] W. Elmasry, A. Akbulut, A.H. Zaim, Deep learning approaches for predictive masquerade detection, Secur. Commun. Netw. 2018 (2018).
[72] Y. Shi, R.C. Eberhart, Parameter selection in particle swarm optimization, in: International conference on evolutionary programming, Springer, 1998, pp. 591–600.
[73] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, IEEE Trans. Evolut. Comput. 6 (1) (2002) 58–73.
[74] E. Aminanto, K. Kim, Deep learning in intrusion detection system: an overview, in: 2016 International Research Conference on Engineering and Technology (2016 IRCET), Higher Education Forum, 2016, pp. 1–12.
[75] R. Vani, Towards efficient intrusion detection using deep learning techniques: a review, Int. J. Adv. Res. Comput. Commun. Eng. (IJARCCE) 6 (10) (2017).
[76] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al., Deep neural networks for acoustic modeling in speech recognition, IEEE Signal Process. Mag. 29 (2012).
[77] Y. Bengio, P. Simard, P. Frasconi, et al., Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neur. Netw. 5 (2) (1994) 157–166.
[78] S. Hochreiter, J. Schmidhuber, Long short-term memory, Neur. Comput. 9 (8) (1997) 1735–1780.
[79] J. Kim, J. Kim, H.L.T. Thu, H. Kim, Long short term memory recurrent neural network classifier for intrusion detection, in: 2016 International Conference on Platform Technology and Service (PlatCon), IEEE, 2016, pp. 1–5.
[80] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, Y. Bengio, Learning phrase representations using rnn encoder-decoder for statistical machine translation, arXiv preprint arXiv:1406.1078 (2014).
[81] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, Empirical evaluation of gated recurrent neural networks on sequence modeling, arXiv preprint arXiv:1412.3555 (2014).
[82] R. Salakhutdinov, G. Hinton, Deep boltzmann machines, in: Artificial intelligence and statistics, 2009, pp. 448–455.
[83] G.E. Hinton, S. Osindero, Y.-W. Teh, A fast learning algorithm for deep belief nets, Neur. Comput. 18 (7) (2006) 1527–1554.
[84] D.E. Rumelhart, G.E. Hinton, R.J. Williams, et al., Learning representations by back-propagating errors, Cognit. Model. 5 (3) (1988) 1.
[85] G.E. Hinton, Training products of experts by minimizing contrastive divergence, Neur. Comput. 14 (8) (2002) 1771–1800.
[86] M.A. Salama, H.F. Eid, R.A. Ramadan, A. Darwish, A.E. Hassanien, Hybrid intelligent intrusion detection scheme, in: Soft computing in industrial applications, Springer, 2011, pp. 293–303.
[87] S. Axelsson, The base-rate fallacy and its implications for the difficulty of intrusion detection, in: Proceedings of the 6th ACM Conference on Computer and Communications Security, ACM, 1999, pp. 1–7.
[88] Z.-Q. Zeng, J. Gao, Improving svm classification with imbalance data set, in: International Conference on Neural Information Processing, Springer, 2009, pp. 389–398.
[89] M. Kubat, S. Matwin, et al., Addressing the curse of imbalanced training sets: one-sided selection, in: Icml, 97, Nashville, USA, 1997, pp. 179–186.
[90] H. He, Y. Ma, Imbalanced Learning: Foundations, Algorithms, and Applications, John Wiley & Sons, 2013.
[91] S. Boughorbel, F. Jarray, M. El-Anbari, Optimal classifier for imbalanced data using matthews correlation coefficient metric, PloS one 12 (6) (2017) e0177678.
[92] B.W. Matthews, Comparison of the predicted and observed secondary structure of t4 phage lysozyme, Biochimica et Biophysica Acta (BBA)-Protein Struct. 405 (2) (1975) 442–451.
[93] J.A. Swets, Measuring the accuracy of diagnostic systems, Science 240 (4857) (1988) 1285–1293.
[94] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inform. Process. Manag. 45 (4) (2009) 427–437.
[95] M. Hossin, M. Sulaiman, A review on evaluation metrics for data classification evaluations, Int. J. Data Min. Knowl.Management Process 5 (2) (2015) 1.
[96] M. Elhamahmy, H.N. Elmahdy, I.A. Saroit, A new approach for evaluating intrusion detection system, CiiT International Journal of Artificial Intelligent Systems and Machine Learning 2 (11) (2010) 290–298.
[97] MIT Lincoln Laboratory, DARPA Intrusion Detection Evaluation Data Set, 1998,. https://www.ll.mit.edu/r-d/datasets/1999-darpa-intrusion-detection-evaluation-dataset/
[98] J. McHugh, Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory, ACM Trans. Inform. Syst. Secur. (TISSEC) 3 (4) (2000) 262–294.
[99] S.J. Stolfo, W. Fan, W. Lee, A. Prodromidis, P.K. Chan, Cost-based modeling for fraud and intrusion detection: Results from the jam project, in: Proceedings DARPA Information Survivability Conference and Exposition. DISCEX'00, 2, IEEE, 2000, pp. 130–144.
[100] M. Tavallaee, E. Bagheri, W. Lu, A.A. Ghorbani, A detailed analysis of the kdd cup 99 data set, in: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, IEEE, 2009, pp. 1–6.
[101] NSL-KDD Dataset, 2009, https://github.com/defcom17/NSL_KDD/
[102] Canadian Institute for Cybersecurity - IDS 2017, 2017, https://www.unb.ca/cic/datasets/ids-2017.html
[103] I. Sharafaldin, A.H. Lashkari, A.A. Ghorbani, Toward generating a new intrusion detection dataset and intrusion traffic characterization., in: ICISSP, 2018, pp. 108–116.
[104] I. Sharafaldin, A. Gharib, A.H. Lashkari, A.A. Ghorbani, Towards a reliable intrusion detection benchmark dataset, Softw. Netw. 2018 (1) (2018) 177–200.
[105] Python (2019) URL https://www.python.org.
[106] NumPy (2019) URL https://www.numpy.org.
[107] Keras (2019) https://keras.io.
[108] F. Chollet, Keras (2015) URL https://github.com/fchollet/keras.
[109] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467.
[110] TensorFlow, (2019) URL https://www.tensorflow.org.
[111] CUDA- Compute Unified Device Architecture (2019) URL https://developer.nvidia.com/about-cuda.
[112] cuDNN- The NVIDIA CUDA Deep Neural Network library (2019) URL https://developer.nvidia.com/cudnn.
[113] W.W. Daniel, Friedman two-way analysis of variance by ranks, Applied Nonparametric Stat. (1990) 262–274.
[114] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.
[115] J. Davis, M. Goadrich, The relationship between precision-recall and roc curves, in: Proceedings of the 23rd international conference on Machine learning, ACM, 2006, pp. 233–240.
[116] T. Fawcett, An introduction to roc analysis, Patt. Recogn. Lett. 27 (8) (2006) 861–874.
[117] A.P. Bradley, The use of the area under the roc curve in the evaluation of machine learning algorithms, Patt. Recogn. 30 (7) (1997) 1145–1159.
[118] T.C. Landgrebe, P. Paclik, R.P. Duin, Precision-recall operating characteristic (p-roc) curves in imprecise environments, in: 18th International Conference on Pattern Recognition (ICPR'06), IEEE, 2006, pp. 123–127.

Wisam Elmasry received his B.Sc. and M.Sc. degrees in computer engineering from The Islamic University of Gaza (IUG), Gaza, Palestine in 2004 and 2010 respectively. He is currently a PhD student of computer engineering in Istanbul Commerce University in Turkey. His areas of research interest include Image Steganography, Cryptography, Cyber Security, Deep Learning Models, Evolutionary Algorithms, and MANET Routing Protocols Security.

Akhan Akbulut is an Assistant Professor of Computer Engineering Department at Istanbul Kültür University where he has been a faculty member since 2004. He worked as a Postdoctoral Researcher in the Computer Science Department at North Carolina State University between 2017 and 2019. His research interests focus on the design and performance optimization of software intensive systems, Internet architectures, and network security. Dr. Akbulut received a PhD in Computer Engineering from the Istanbul University, Turkey.

Abdül Halim Zaim obtain Bachelor's Degree in 1993 from Yildiz Technical University Department of Computer Science Engineering Department of Electrical and Electronics Engineering Faculty. He completed his master's degree in Computer Engineering at Boğaziçi University in 1996 and completed his doctorate in Electrical and Computer Engineering at North Carolina State University (NCSU) in 2001. During his Ph.D., he served as a senior researcher at Alcatel USA for one and a half year. Dr. Zaim received his assistant professor title in 2003, Associate Professor in 2004, Professor in 2010 from Istanbul University, respectively. His main academic research interests are Telecommunication, Computer Networks, Performance Modeling, and Software Development.