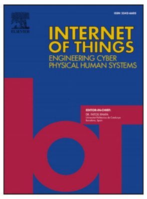
Internet of Things xxx (xxxx) xxx

Contents lists available at ScienceDirect

Internet of Things

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



Cost-sensitive stacked auto-encoders for intrusion detection in the Internet of Things

Akbar Telikani[a,*], Amir H. Gandomi[b]

[a] Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran
[b] Faculty of Engineering and IT, University of Technology Sydney, Ultimo, Australia

ARTICLE INFO

Article history:
Received 30 June 2019
Revised 30 September 2019
Accepted 30 September 2019
Available online xxx

Keywords:
Internet of Things
Security
Intrusion detection
Stacked auto-encoder
Class imbalance

ABSTRACT

Intrusion Detection System (IDS) is an important tool for protecting the Internet of Things (IoT) networks against cyber-attacks. Traditional IDSs can only distinguish between normal and abnormal behaviors. On the other hand, modern techniques can identify the kind of attack so that the appropriate reactions can be carried out against each type of attack. However, these techniques always suffer from the class-imbalance which affects the performance of IDS. In this paper, we propose a cost-sensitive stacked auto-encoder, CSSAE, to deal with class imbalance problem in IDS. CSSAE generates a cost matrix in which a unique cost is assigned to each class based on the distribution of different classes. This matrix is created in the first stage of CSSAE. In the second phase, a two-layer stacked auto-encoder is applied to learn features with better distinguish between the minority and the majority classes. These costs are used in the feature learning of deep learning, where the parameters of the neural network are modified by applying the corresponding costs in the cost function layer. The proposed method is able to perform on both binary-class data and multiclass data. Two well-known KDD CUP 99 and NSL-KDD datasets are used to evaluate the performance of CSSAE. Compared with other IDSs that have not considered class imbalance problem, CSSAE shows better performance in the detection of low-frequency attacks.

1. Introduction

The Internet of Things (IoT) connects all *things* in the world to the internet. This is the result of advancements in mobile communication, wireless sensor networks, and radio frequency identification so that things can collaborate with one another anytime, anywhere and in any form [1]. The IoT is an invisible but intelligent network that senses, controls and can be programmed [2]. It is predicted that over than 28 billion IoT devices can connect to the Internet by 2020 [3]. Therefore, each person will possess three devices with the ability to connect to the Internet. According to a report, the annual economic impact of IoT in 2025 will be between $2.7 and $6.2 trillion. These figures suggest the steep growth of IoT services [4]. Since the IoT was established based on the conventional unsecured Internet model, security and privacy are two critical challenges for IoT development [5]. Thus, questions regarding users, servers, and trusted third parties must be managed [6]. With the spread of network attacks either to access sensitive information without permission by external users or to raise access privileges by internal users, cyber-security recently has become one of the most critical issues. Although there are different ongoing security mechanisms for data confidentiality and authentication, networks are vulnerable to multiple

[*] Corresponding author.
E-mail addresses: akbar.telikani@gmail.com (A. Telikani), gandomi@uts.edu.au (A.H. Gandomi).

https://doi.org/10.1016/j.iot.2019.100122
2542-6605/© 2019 Elsevier B.V. All rights reserved.

disrupting attacks. Intrusion Detection System (IDS) is a defense mechanism to fulfill this purpose. IDS is required to be applied at the communication level to monitor network operation and communication links. When a pre-defined policy is ignored, the system alerts about anomaly activities [7].

Machine Learning (ML)-based intrusion detection mechanisms are a new trend. This is because cyber-security has become more complex than before and traditional signature-based mechanisms are no longer effective. The ML techniques are able to efficiently discover valuable knowledge and hidden information [8]. Deep Learning is a robust method based on artificial neural networks. According to the Gartner report,[1] deep learning and IoT are among the top strategic technology trends for 2019. This is because traditional ML approaches cannot address the emerging analytic needs of IoT systems. Reducing handcrafted dependence is the important improvement of deep learning over the traditional ML approaches. Indeed, engineered feature sets are to be used for the training; as a result, some features that may not be evident from an expert's perspective can easily be extracted in deep learning models. Although deep learning can harvest valuable knowledge from complex systems, its performance is affected when data is unbalanced. This is because of class imbalance in the training dataset, which means that IDS achieves poorer detection accuracy when dealing with these low-frequency attacks.

Because of the nature of the intrusion datasets, the class imbalance is a significant challenge which affects the attack identification. Cost-sensitive is an effective technique to deal with this problem. Although some deep learning-based intrusion detection mechanisms have been recently developed [9–11], to our best knowledge, there is no cost-sensitive feature learning for these approaches. While convolutional networks achieve better performance in applications related to vision, Stacked Auto-Encoder (SAE) performs very well with other areas such as intrusion detection [12]. Thus, we integrate a cost-sensitive function into SAE to address the class imbalance in intrusion detection for IoT. This is the first work proposing a solution for cost-sensitive deep learning for anomaly detection in IoT. In this paper, we propose a SAE-based system for intrusion detection to learn robust feature representations and classifier parameters under a cost-sensitive setting. We have called this method cost-sensitive stacked auto-encoder (CSSAE). Using this strategy, we are able to learn an improved classifier that deals with the class imbalance problem in IDS. The proposed system is extensively tested on two major intrusion datasets to show the superiority of CSSAE compared to the state-of-the-art methods. The results of evaluations are presented in terms of different classification measures such as accuracy, recall, precision, and skewed f-measure. The results show that CSSAE outperforms the other similar approaches in terms of effectiveness and efficiency.

The rest of the paper is organized as follows: Section 2 describes relevant background information related to this study, including intrusion detection problem, deep learning, and class imbalance problem in classification. Section 3 elaborates on some related works for intrusion detection problem in the IoT. In Section 4, a novel cost-sensitive deep learning is proposed to deal with class imbalance in intrusion detection. Section 5 details datasets as well as evaluation measures used for evaluating our proposed method. Section 6 presents the experimental results and discussions. Finally, Section 7 concludes the paper.

2. Background information

2.1. Stacked auto-encoders (SAEs)

An auto-encoder is an unsupervised back-propagation neural networks algorithm for feature extraction. It tries to learn the best parameters to reconstruct the output as closely to the input as possible. The auto-encoder compresses the input into a lower-dimensional code and then reconstructs the output from this representation. Indeed, an auto-encoder is a symmetrical structure of artificial neural networks [13]. Fig. 1 shows the basic structure of a single auto-encoder. As we can see, this auto-encoder consists of two stages: encoder and decoder. The aim of an encoder is to map an input vector $x \in [0, 1]^d$ into a hidden representation h. The input data is transformed into the hidden layer through the encoding function f:

$$h = f(W_1 x + b_1) \tag{1}$$

where f is a non-linear activation function. The widely used activation functions include sigmoid, relu, tanh, and softsign. A logistic sigmoid function $f(x) = 1/(1 + exp(-x))$ is often used as the activation function. The hidden representation h is then reconstructed using decoding function g in order to generate the output vector $x' \in [0, 1]^d$. The decoding function is a non-linear mapping function as shown below:

$$y = g(W_2 h + b_2) \tag{2}$$

The process of auto-encoding tries to minimize the difference between reconstructed input and the original input. This error value should be as small as possible. The network parameter set $\theta = \{W_1, b_1, W_2, b_2\}$ is fine-tuned to minimize the average reconstruction error. $W_1 \in \mathbb{R}^{z \times d}$ and $W_2 \in \mathbb{R}^{d \times z}$ represent the weight matrices of the hidden layer and output layer, respectively. The vectors $b_1 \in \mathbb{R}^z$ and $b_1 \in \mathbb{R}^d$ are the bias vectors of the hidden layer and output layer, respectively. The terms d and z are the numbers of input neurons and hidden neurons, respectively. The reconstruction error is defined by a likelihood function:

$$L(X, Y) = \frac{1}{2} \sum_{i=1}^{N} \| (y_i - x_i) \|_2^2 \tag{3}$$

[1] https://www.gartner.com/smarterwithgartner/gartner-top-10-strategic-technology-trends-for-2019/.

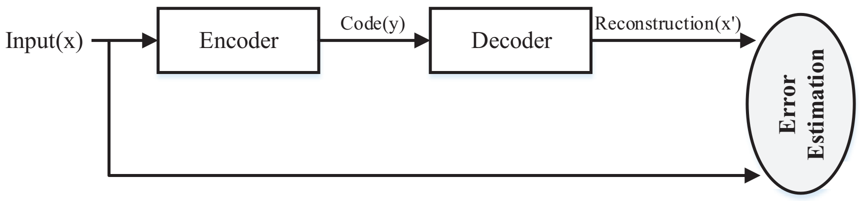


Fig. 1. The structure of a single auto-encoder.

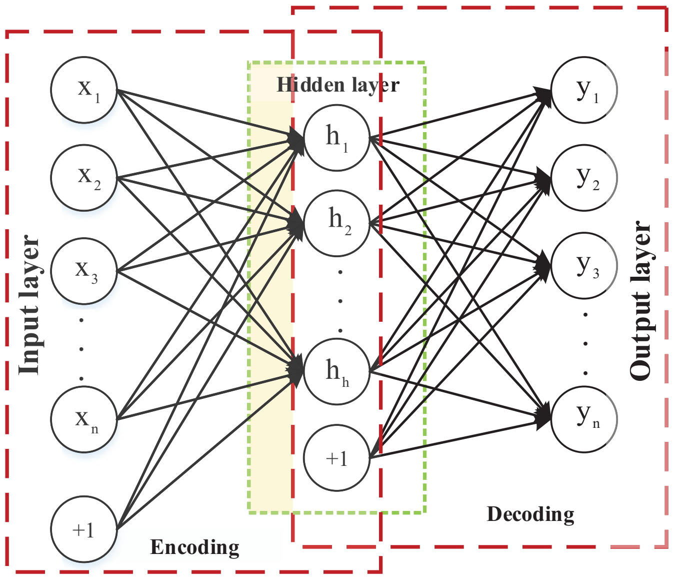


Fig. 2. Schematic diagram of single auto-encoder.

where N is the number of training samples, and x_i denotes the ith training sample.

The object function of sparse auto-encoder is computed by imposing a sparsity constrain on the target activation function to achieve nonlinear mapping of the input vector. This means that the size of hidden layers may be larger than the input size, which is referred to as the sparse auto-encoder. The reconstruction error of the sparse auto-encoder is calculated using the following equation:

$$L_{Sparse}(X, Y) = L(X, Y) + \beta \sum_{j=1}^{z} KL(\rho || \bar{\rho}) \tag{4}$$

where β represents the weight of sparsity penalty item, ρ is the sparsity parameter that is usually a small value close to zero. $\rho_j = \frac{1}{n} \sum_{i=1}^{N} h_i(x_i)$ is the average activation of $j$the hidden node. In order to enforce the constraint $\rho = \bar{\rho}$, an extra penalty term is added to the optimization objective that penalizes $\bar{\rho}$ by deviating significantly from ρ. The penalty term is based on Kullback–Leibler divergence that can be defined by Vincent et al. [14]:

$$KL(\rho || \bar{\rho}) = \rho \log \frac{\rho}{\rho_j} + (1 - \rho) \log \left(\frac{1 - \rho}{1 - \bar{\rho}} \right) \tag{5}$$

The ideal case is when only a few hidden neurons respond to specific classes and most of the neurons are suppressed [15]. It is clear that the parameters of a basic auto-encoder are trained in an unsupervised way. In a basic auto-encoder, when the number of hidden neurons (z) is smaller than that of d, the auto-encoder performs a feature reduction procedure. In contrast, when the size of hidden layer is larger than the size of the input layer, auto-encoder is called sparse auto-encoder [16]. Fig. 2 shows the schematic diagram of sparse auto-encoder with three layers, including the input layer, the hidden layer, and the output layer. In this network, *+1* is the bias neuron.

Single sparse auto-encoders can only learn low-level features. Several auto-encoders can be stacked to derive deeper and higher-level features in a hierarchical way [17]. This technique is called stacked auto-encoder. A SAE is trained using two phases: pre-training and fine-tuning. In the pre-training step, each auto-encoder reconstructs the input into the output using unsupervised learning without any labeling or prior processing [13]. The first auto-encoder takes the original data

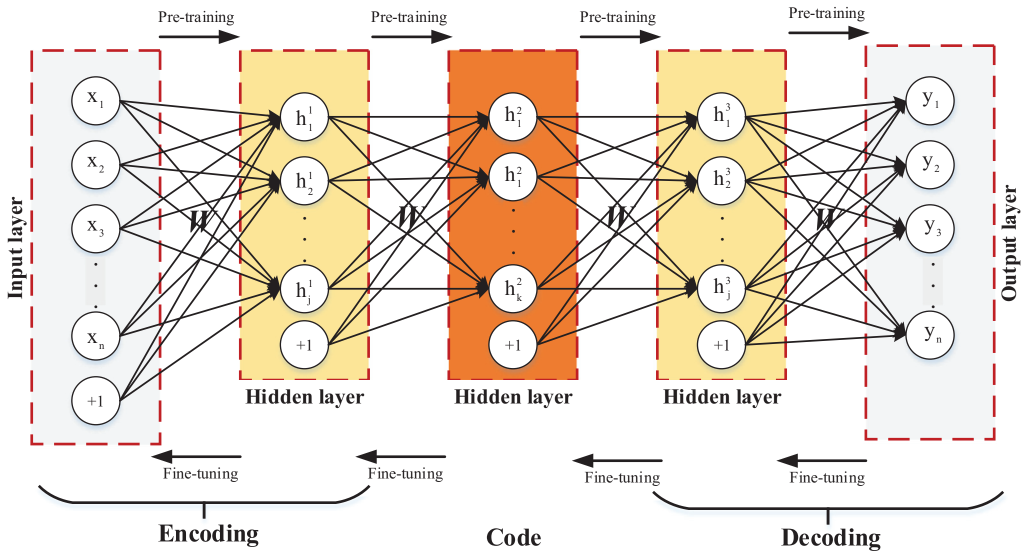


Fig. 3. Schematic diagram of stacked auto-encoder.

vector x as input to train the parameters of the first hidden layer. The output of the first auto-encoder is fed to the second hidden layer to train the parameters. These steps are repeated until the parameters of all hidden layers are trained. A SAE enlarges the parameter space, resulting in local optimum solution, particularly when the amount of input is limited. Therefore, greedy layer-wise training is used in a stacked network that can convey better initialized parameters [15]. After pre-training, the trained parameters are set to the initial parameters of the SAE. In the fine-tuning stage, the labeled samples are used as supervised instances [18]. The final layer can deal with traditional supervised classification and the pre-trained neural network can be fine-tuned using back-propagation. Fig. 3 shows an example of SAE with three hidden layers.

2.2. Class imbalance

Similar to most real-world problems, the collected data related to intrusion activities encountered the *class-imbalance problem*. When the instances of some classes in the training set outnumber the other instances, the multiclass dataset is said to be *imbalanced* or *skewed*. This skewed distribution of class instances affects the performance of machine learning algorithms. They are biased toward the majority classes [19]. Approaches that handle the unbalanced data problem can be grouped into two categories:

Data-level approaches: These approaches are performed in the pre-processing stage of the machine learning process to re-balance the class distributions. Since data-level approaches are independent of the classification phase, they can be employed flexibly. Under-sampling and over-sampling are two well-known strategies. Under-sampling removes examples from the majority class, while over-sampling adds examples to the minority class. These approaches have some drawbacks. Under-sampling can potentially remove useful knowledge about the majority class data. Over-sampling makes the training complex and burdensome by increasing the size of the training set. Besides, this technique is susceptible to over-fitting when the minority class samples are randomly duplicated [20]. The Random Over-Sampling (ROS) and the Random Under-Sampling (RUS) are two common data sampling methods. Chawla et al. [21] proposed Synthetic Minority Oversampling Technique (SMOTE) that generates new samples by linear interpolation between closely lying minority class instances. A combination of under-sampling and over-sampling has been presented in [22] to balance the training data. However, it increased the computational cost of a classification model.

Algorithm-level approaches: These methods modify the learning procedure by inserting an additional specialized mechanism into the original algorithm to prove the sensitivity of the classifier toward minority classes. Ng et al. [17] proposed a two-phase encoding to deal with class imbalance problem. In the first phase, an SAE with sigmoid function is used. On the other hand, the SAE of the second phase applies tanh as activation function. The features learned from the two SAEs are combined. Cost-sensitive is a technique to deal with class imbalance in classification, in which different costs are assigned to each class and the algorithm adapts these costs during the training phase. The aim is to assign a higher cost for minority class instances in order to reduce the overall learning cost. In the context of deep learning, some approaches have recently been proposed to integrate class-specific costs into deep networks. Chung et al. [23] designed a loss function that combined the cost information with the training stage of cost-sensitive deep learning. They also showed the effectiveness of integrating the loss function into the pre-training stage. This method was applied for Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN). The loss function proposed in [24] assigns equal importance to misclassifications in the minority and majority classes. Khan et al. [19] incorporated a cost-sensitive setting in CNN to learn feature representa-

Table 1
The distribution of different types of events in KDD 99 and NSL-KDD datasets.

Event	Dataset	
	NSL-KDD	KDD 99
DoS	0.356	0.781813
Probe	0.088	0.008246
R2L	0.022	0.009054
U2R	0.0000614	0.000116
Normal	0.532	0.200772

tions and classifier parameters for both the majority and minority classes. This approach can be used for both binary and multiclass problems. The cost-sensitive function was applied before the softmax and the loss layer to alter the result of the last layer of CNN. Since the proposed technique can only been developed for 2D data (e.g., computer vision) it is not appropriate for 1D data such as intrusion data.

2.3. Intrusion detection problem

Intrusion detection is the process of identifying activities that try to obtain unauthorized access to a computer system. An attacker may be internal or external. The former refers to the users who are inside the network with a degree of permissions. However, they attempt to misuse their legitimate access privileges to attain non-authorized privileges. The latter type of users tries to gain unauthorized access to system information [25].

An intrusion detection system collects and analyzes different types of network or host data such as packet headers, operating system calls, traffic statistics, and file-system changes. In the intrusion detection process, the data collected by the devices is received by the analysis engine. This engine aims to investigate the data to detect anomalies and intrusions. The IDS system generates an alert report for the network administrator if it diagnoses an intrusion [26].

There are two types of IDS: Network-based IDS (NIDS) and Host-based IDS (HIDS). In NIDS, network traffic of various network segments is monitored in order to identify malicious behaviors. HIDS is performed only for a network device and pursues malicious activities occurring within the system. HIDS investigates not only network traffic but other data such as running processes, file-system changes, inter-process communication, and application logs. On the other hand, the NIDS analyzes only network traffic.

2.4. Class imbalance problem in intrusion data

The imbalance of labels in the intrusion datasets is significant. In the case of cyber-security in IoT, the majority of machines in the network often behave normally while a few of devices produce malicious traffic. Moreover, compared to the number of normal transactions within a certain time range, some attacks are not extremely frequent [27]. KDD 99 and NSL-KDD are two well-known intrusion datasets which are unbalanced. The characteristics of these datasets are described in Section 5.1. However, the different types of behaviors are shown in Table 1. According to the analysis of these datasets, malicious activities in the IoT network can be grouped into four main categories:

- *Denial of Service (DoS)*: Attackers try to disrupt legitimate users' access to a particular service or device [28].
- *Probe*: An attacker seeks to attain information about the network through scanning activities (i.e. ports scanning).
- *User to Root (U2R)*: When an attacker tries to increase the privilege of a limited user to a superuser or root access (e.g. via malware infection or stolen credentials).
- *Remote to Local (R2L)*: when an attacker gains remote access to a victim machine imitating existing local users.

According to the presented ratios, including Dos and Normal events are included the majority of the number of samples. The distribution of both classes is 0.35 and 0.53 in NSL-KDD, respectively. These ratios are 0.78 and 0.2 in KDD 99. On the other hand, Probe, R2L, and U2R attacks have rarely happened. Furthermore, since U2R and R2L attacks mimic normal users' behavior, they are the most challenging attacks for detection [29]. Unbalanced distribution causes the misclassification of these attacks on DoS and Normal behaviors. Therefore, detection systems cannot correctly identify these attacks. These false detections affect the decision to deal with any type of attack. Thus, defense mechanisms are designed that cannot correctly detect some of the attacks. Therefore, the system will not work properly.

3. Related works

Intrusion detection techniques generally can be classified into supervised and unsupervised learning. In the former, the normal behavior of networks is modeled using a labeled dataset. The latter builds a model based on normal behaviors with this assumption that these behaviors frequently occur [29]. According to the placement strategy, intrusion detection approaches are classified into two categories: centralized and distributed. In the centralized-based, IDS is placed in the border router or a dedicated host. However, it makes communicational overhead between the nodes and IDS. Although distributed strategy can address this problem, it requires more processing and storage resources. In this paper, we categorize intrusion detection techniques into three groups: signature-based, anomaly-based, and specification-based.

In signature-based, a behavior is detected as an attack if the network activity matches an attack signature stored in the database. These approaches are very effective at detecting known threats, but are ineffective against new attacks and the variants of known attacks [30]. Anomaly-based approaches compare the behaviors of the network against a normal activity. If the deviation between normal and attack behaviors exceed a threshold, IDS generates an alert. The anomaly-based strategy is able to detect new attacks. Since this strategy considers anything that does not match normal activity as an intrusion, the false positive ratio is high. Statistical or machine learning techniques are usually used to model normal behavior. Specification-based approaches are similar to anomaly-based detection, i.e. detection of deviation from normal behavior. However, the rules of each specification are manually defined by a human expert. This is time-consuming and error-prone [31].

An IDS model has been presented by Pajouh et al. [29] based on a two-layer dimension reduction and two-tier classification module. Besides, the proposed model can be used to identify User to Root (U2R) and Remote to Local (R2L) attacks. Component analysis and linear discriminate analysis were applied for dimension reduction. On the other hand, Naive Bayes and K-Nearest Neighbor were used in the two-tier classification phase. Kozik et al. [27] proposed a distributed attack detection system for Cloud and Edge-assisted IoT applications. They used Extreme Learning Machines. Apache Spark cloud framework is employed on the artificial Netflow. Chen et al. [32] presented a model-based cyber security management approach for IoT with minimal impacts on the performance. Moreover, the approach can learn the signatures of an unknown attack in real-time. Pahl and Aubet [33] developed a machine learning-based approach for anomaly detection for IoT by only observing inter-service communication. They used two types of clustering algorithms: a grid-based algorithm and k-means. The clustering model is updated using an online learning technique. Hasan et al. [34] investigated the performance of five machine learning models for anomaly detection in the IoT systems. These algorithms included Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), and Artificial Neural Network (ANN). Experimental results showed that RF outperformed the other four models.

Deep learning-based intrusion detection has recently attracted more attention. A methodology for the online detection of network attacks against IoT gateways has been developed by Brun et al. [9], in which dense random neural network was used to predict Denial of Service (DoS) and Denial of Sleep attacks. Diro and Chilamkurti [10] designed a distributed deep learning-based attack detection mechanism using fog-to-things architecture. Feng et al. [16] applied an auto-encoder for anomaly detection in wireless communications. Lopez-Martin et al. [11] used conditional variational auto-encoder for IDS, in which the intrusion labels are integrated inside the decoder layers. Shone et al. [35] proposed a model using the combination of deep and shallow learning. In this model, Non-symmetric Deep Auto-Encoder (NDAE) is first used for unsupervised feature learning. A classification model is then constructed upon the model using stacked NDAEs and the RF classification algorithm. However, the critical issue in the afore-mentioned literature is that they have not considered the cost of different classes.

4. CSSAE: cost-sensitive stacked auto-encoders

4.1. Framework of the proposed method

The aim of this study is to modify the cost function of SAE to be more sensitive toward misclassification of minority class. Fig. 4 shows our proposed framework, which includes three main steps: *Cost Matrix Formation, Feature Learning using SAE*, and *Cost-Sensitive Cost Function*. Using this framework, we aim to reduce the impact of unbalanced data on the performance of IDS. At the first step of our schema, a cost matrix is generated by computing the distributions of samples of all classes. Then, the data is fed to a stacked sparse auto-encoder to classify instances. To avoid over-fitting, we use an early stopping technique. If the value of loss function on the validation set remains almost unchanged for several epochs, the training procedure stops. We also use the Batch Normalization technique to speed up the learning phase. In this study, Softmax

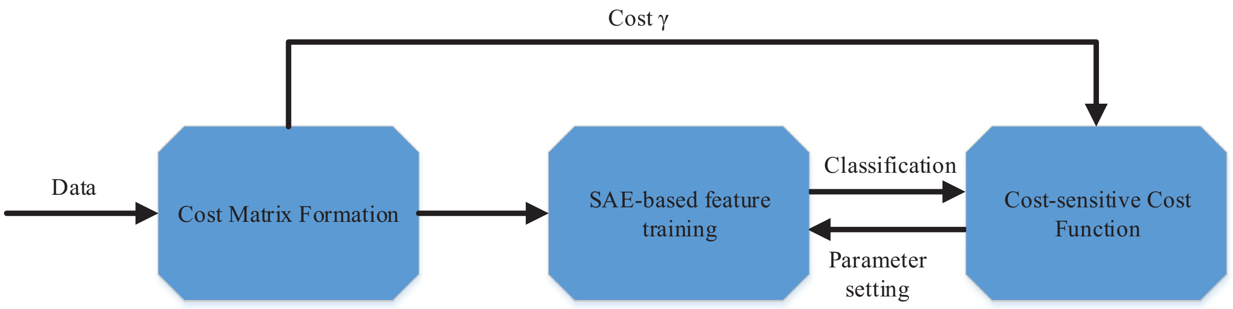


Fig. 4. Schematic diagram of our cost-sensitive SAE algorithm.

Table 2
An example of cost matrix for a four-class data.

		Classes			
		c_1	c_2	c_3	c_4
Classes	$\mathbf{c}_1$	0	$Cost_{c1\to c2}$	$Cost_{c1\to c3}$	$Cost_{c1\to c4}$
	$\mathbf{c}_2$	$Cost_{c2\to c1}$	0	$Cost_{c2\to c3}$	$Cost_{c2\to c4}$
	$\mathbf{c}_3$	$Cost_{c3\to c1}$	$Cost_{c3\to c2}$	0	$Cost_{c3\to c4}$
	$\mathbf{c}_4$	$Cost_{c4\to c1}$	$Cost_{c4\to c2}$	$Cost_{c4\to c3}$	0

classifier, which is a multi-class version of Logistic Regression, is used. Cross-entropy loss computes the probability of each class, instead of the hinge loss.

4.2. Cost matrix formulation

In this section, we propose a new cost matrix γ that is proper for SAE training. The output of the last layer of an SAE is modified using the cost matrix γ. In SAE, the classification decision is biased to the class with the maximum classification score. The classifier weights are altered during the training process. The aim is to assign the maximum score to the desired class, while the other classes have a lower score. In the case of class imbalance, the infrequent classes have less impact on the classification model than the others. Thus, we introduce new cost assignment to encourage the correct classification of the less frequent classes.

Let the cost $\gamma_{i,\,j}$ denote the cost of classifying a sample belonging to a class i into class j, $i \neq j$. Given x is an input sample and γ is the cost matrix, the classifier tries to minimize risk $\rho(c|x)$, where c is the class prediction made by the classifier. The risk can be computed as:

$$\rho(i|x) = P(j|x) \tag{6}$$

where P(j|x) is the posterior probability over all classes given in sample x. An effective classifier should minimize the risk.

In a cost matrix, the diagonal of the matrix is known as the utility vector. This vector shows the correct classification and is set to zero. Also, all costs are non-negative, i.e. $\gamma_{i,\,j} > 0$. Table 2 shows an example of a cost matrix for a dataset with four classes. A *4* × *4* matrix is created in such a way that all the matrices' cell values are greater zero except for those on the diagonal which are always zero. This means that there is no cost when the algorithm classifies a sample correctly. Otherwise, our algorithm assigns a misclassification cost depending on the corresponding cost in the cost matrix.

4.3. Cost-sensitive training

In order to deal with class imbalance during the process of feature learning, we propose a two-layer SAE to learn features. Fig. 5 shows our proposed SAE algorithm with two hidden layers and the Softmax classifier as the output layer for the classification algorithm. We improve this cost function based on a cost-sensitive strategy to deal with class imbalance. Although there is no limit to the number of AEs being stacked, a single auto-encoder is not powerful enough to learn useful features. On the other hand, SAE with more than two AEs may not achieve much better representation at the cost of slower training time.

Assume that the output of the last layer is $\{X, Y\} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_C)\}$, $x_i \in \mathbb{R}^{d\times\ 1}$, d is the number of encoding neurons in the last layer, $y_i \in \mathbb{R}^{C\times\ 1}$ is the labels. The activation function $f_\theta(x)$ of the Softmax classifier indicates in which class the instance x belongs. This is determined using maximum conditional probability $p(y = j,\ x; \theta),\ \ j \in \{1, 2, \ldots, C\}$. The value of $f_\theta(x)$ is computed as follows:

$$f_\theta(x) = \frac{1}{\sum_{j=1}^{C} e^{\theta_j^T x_i}} \begin{bmatrix} e^{\theta_1^T x_i} \\ e^{\theta_2^T x_i} \\ \cdots \\ e^{\theta_C^T x_i} \end{bmatrix} = \begin{bmatrix} p(y_i = 1|x_i; \theta_1) \\ p(y_i = 2|x_i; \theta_2) \\ \cdots \\ p(y_i = C|x_i; \theta_C) \end{bmatrix} \tag{7}$$

In this equation, $\theta_j$is the parameter mapping toward the $j$the class. The weight and bias parameters are learnt by the Softmax layer by minimizing the cost value. The cost function is calculated using the following equation:

$$J(W, b) = -\frac{1}{N}\left[\sum_{i=1}^{N}\sum_{j=1}^{C} I\{y_i = j\} \log \frac{e^{\theta_1^T x_i}}{\sum_{j=1}^{C} e^{\theta_1^T x_i}}\right] + \frac{\lambda}{2}\sum_{k=1}^{L}\sum_{i=1}^{nl}\sum_{j=1}^{nl+1} w_{ij}{}^2 \tag{8}$$

where N is the number of training samples, C is the number of classes, λ is a weight decay coefficient, L is number of hidden layers, nl is the number of neurons in the layer. $I\{y_i = j\}$ determines the membership of an element in a subset j of set y. If $y_i = j$, $I\{y_i = j\}$ is equal to *1*; otherwise, it is *0*. This means that when the sample y_i of class j is misclassified, $p(y_i = j|x_i; \theta_j)$ tends to be smaller and the result of Eq. (3) will be very large.

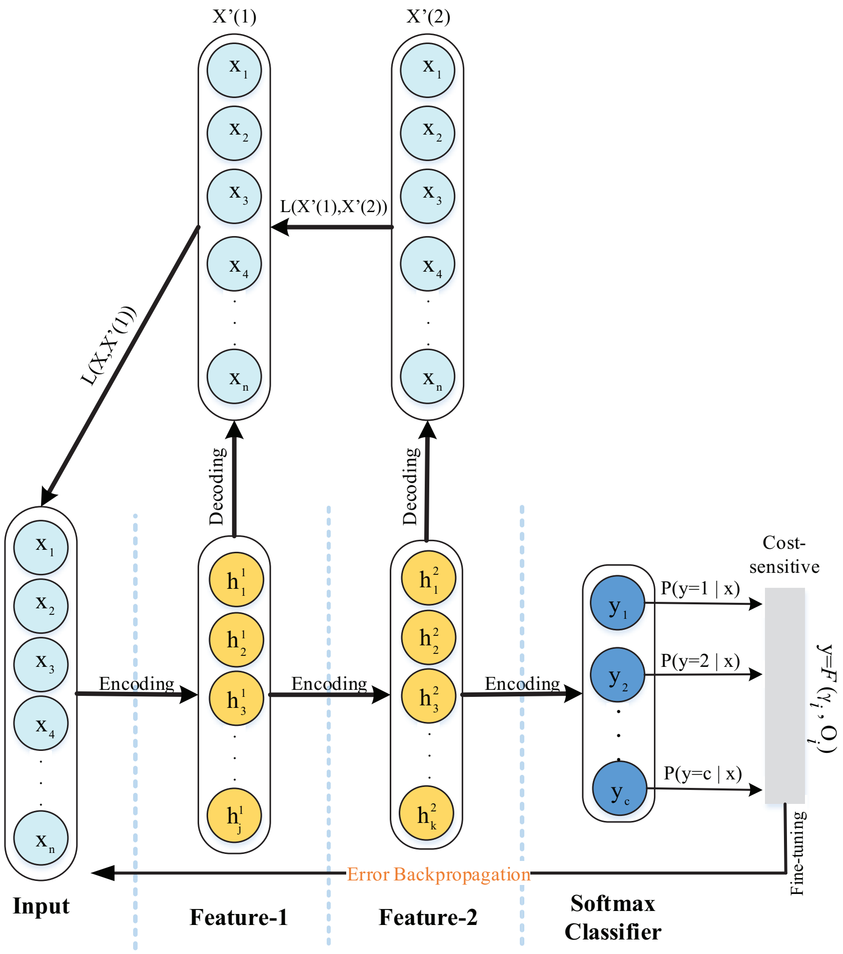


Fig. 5. An overview of our proposed method.

We aim to punish each type of misclassification error differently based on some given costs. This penalty value is greater when a minority sample is misclassified than when a majority sample is wrongly classified. As we mentioned in Section 4.2, minority and majority classes are determined and we only need to find the corresponding cost from the cost matrix. The superiority of our algorithm is that it is not necessary to determine the kind of classes in terms of minority or majority. Indeed, the costs are assigned to the classes solely based on the distributions. This characteristic helps algorithm could be applied on any datasets.

CSSAE algorithm aims to impose a penalty for each misclassification according to the corresponding cost. We punish both misclassified and actual classes in such a way that the output value of predicted class is reduced (Eq. (9)) while the output value of the actual class increases (Eq. (10)). The main idea behind this strategy is that the output with larger value has the more chance to get higher probability than smaller values. Therefore, CSSAE tries to increase the probability value of actual class by increasing the corresponding output value in the last layer. On the other hand, CSSAE decrease the output value of the misclassified class.

$$y_i = y_P - \gamma_{i,k} \times y_P \tag{9}$$

$$y_i = y_A + \gamma_{i,k} \times y_A \tag{10}$$

The terms y_P and y_A are the outputs of predicted class and actual class, respectively. The value of y_i is the new output of misclassified and actual class.

Once the weight and bias parameter learning is completed in the Softmax classifier, fine-tuning of all weight and bias parameters in the whole network is performed by the algorithm. The weights of all layers in the network are improved by using back-propagation technique, as follows:

$$w_{ij} = w_{ij} + \Delta w_{ij} \rightarrow w_{ij} + l\alpha y_i \tag{11}$$

$$\alpha = \begin{cases} y_P - y_i, & if\ i \in N_L\ and\ i\ \ is\ predicted\ class \\ y_i - y_A, & if\ i \in N_L\ and\ i\ is\ actual\ class \\ y_i(1-y_i)\sum_k \alpha_k w_{ik}, & if\ i\ N_L \\ 0, & otherwise \end{cases} \tag{12}$$

where N_L is the neurons in the last layer, α is the error of a neuron, and w_{ik} is the weight of the connection from unit i to a unit k in the next higher layer. According to Eq. (12), to compute the error of a unit in the last layer, we should calculate the different between the new cost-sensitive values and the previous values. On the other hand, the weighted sum of the errors of the units connected to unit i in the next layer is considered in order to compute the error of a hidden layer unit. Biases are updated by the following equation:

$$b_j = b_j + \Delta b_j \to b_j + l\alpha \tag{13}$$

5. Experimental setup

In our experiments, we used the KDD '99 and NSL-KDD datasets to evaluate the proposed method. These datasets are commonly used in the literature to assess the performance of IDS methods.

5.1. Datasets

- *KDD CUP 99:* The KDD CUP 99 dataset is a well-known intrusion evaluation dataset that was applied in DARPA's IDS evaluation program [36]. The original version of this dataset consists of seven million connection records including five million of training records and two million of test samples. These records were collected from 7 weeks of network traffic. In academic experiences, 10% of the full-size dataset is used because they aim to evaluate the performance of IDS, not efficiency. This reduced dataset is referred to as KDD CUP 99, which includes 494,021 training samples and 311,029 testing records.
- *NSL-KDD:* NSL-KDD benchmark dataset [37] is a new version of KDD99 dataset for network intrusion detection systems.

All attacks included in both datasets fell into four major groups: DoS, Probe, U2R, and R2L. The datasets have 22 attack patterns and 41 feature fields. These features are three types: basic features (e.g., protocol type, packet size), domain knowledge features (e.g., the number of failed logins), and timed observation features (e.g., the ratio of connections with SYN errors). Table 3 shows the distribution of train and test datasets for both benchmarks. All 22 attack types have been included in both train and test sets. These attacks are randomly selected in the pre-processing phase, described in Section 5.3.

Table 3
Description of datasets.

Category	Attack type	KDD 99		NSL-KDD	
		Train	Test	Train	Test
DoS	back	2970	331	1183	132
	land	27	3	22	3
	neptune	148,682	16,520	41,284	4587
	pod	316	35	218	24
	smurf	400,393	44,488	2980	331
	teardrop	892	99	813	91
Probe	ipsweep	1398	155	3366	374
	nmap	284	31	1409	157
	portsweep	1255	139	2779	309
	satan	2900	322	3931	437
R2L	ftp_write	10	1	10	1
	guess_password	3978	442	1155	129
	Imap	12	1	11	1
	multihop	23	2	22	3
	phf	6	0	5	1
	spy	2	0	2	0
	warezclient	918	102	801	89
	warezmaster	1460	162	867	97
U2R	loadmodule	10	1	10	1
	buffer_overflow	47	5	45	5
	rootkit	21	2	21	2
	perl	5	0	4	1
Normal		142,084	15,787	69,348	7706
Total		**707,693**	**78,628**	**130,286**	**14,481**

5.2. Evaluation measures

In order to evaluate the performance of our proposed method, different metrics are used, including accuracy, recall, precision, F-Measure, and false alarm rate. To measure the detection performance on the normal and attack classes, the probability of detection (recall) and the probability of a false alarm are commonly applied. Skewed *F*-measure is employed to measure how well IDS can balance the performance between the normal and attack classes. All these measures are computed using four predictions obtained from the application of the classification model on the test dataset [26]. These results are defined below:

- *True Positive (TP):* The number of attacks that are correctly identified.
- *False Positive (FP):* The number of normal samples classified as attacks.
- *True Negative (TN)*: The number of normal records that are correctly classified as normal ones.
- *False Negative (FN):* The number of intrusions that are mistakenly classified as normal.

Accuracy: The accuracy refers to the proportion of the total number of correct classifications (Eq. (14)).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

Recall: The recall indicates the proportion of the actual number of attacks which are correctly detected. Recall is very important for IDS because detection models intend to discover true attack instances as much as possible. The recall is defined as the following equation:

$$Recall = \frac{TP}{TP + FN} \tag{15}$$

Precision: The precision computes the proportion of correctly predicted attacks. This measure has a direct impact on the performance of the system. A higher value of precision indicates a lower false positive rate and vice versa. Precision is computed using the following equation:

$$Precision = \frac{TP}{TP + FP} \tag{16}$$

Skewed F-measure: Obviously, a good detection model tends to attain high value of recall and precision. However, there is a tradeoff between these two measures. The Skewed F-measure evaluates the harmonic mean of recall and precision. Indeed, it examines the accuracy of a system by considering both recall and precision (Eq. (17)).

$$Skewed\,F\text{-}measure = \frac{(1+\alpha)\text{Recall} * \text{Precision}}{\alpha \times \text{Recall} + \text{Precision}} \tag{17}$$

where α is a positive value to decide the relative significance of the precision over the recall. A greater value for α indicates the higher importance of recall over precision. From the aspect of cost in case of unbalanced data, the most important element of the confusion matrix is the *FN* value [38]. Therefore, we pay more attention to how many attack instances are correctly detected. In this case, improving recall is considered more than increasing precision. Similarly [38], we set α as 4 in our evaluations.

False Alarm Rate (FAR): The false alarm rate is the proportion of normal events that are incorrectly classified as malicious. Unlike recall, a low false alarm rate is a prerequisite for any intrusion detection system (Eq. (18)). It is noted that given FAR, a True Negative Rate (TNR) easily can be obtained, which is defined as $TNR = 1 - FAR$.

$$\text{FAR} = \frac{FP}{FP + TN} \tag{18}$$

5.3. Data pre-processing

Both KDD CUP 99 and NSL-KDD datasets require a pre-processing to be used for our proposed deep learning. This phase includes two phase sub-processes: *data separation, data transformation,* and *data normalization* (Fig. 6).

- *Data separation:* Before data modeling, it is necessary to divide data into two sets: a training set and test set. Both datasets should consist of all attack types. Since random sampling is performed, all instances of some attack classes may be selected as a training set. This is because some class distributions are unbalanced. Therefore, we separate all attack classes and choose 90% of samples as the training set and 10% as the test set.
- *Data transformation:* The trained deep learning model needs each tuple of data to be a vector of real numbers. Therefore, each nominal feature is converted into a numerical value using data transformation step. There are three nominal attributes: “protocol_type” (TCP, UDP, and ICMP), “service” (HTTP, FTP and so on), and “flag” (e.g. SF, SH and so on).
- *Data normalization:* In the data normalization step of neural network-based approaches, the value of each attribute is scaled into a well-proportioned range between 0 and 1. This process leads to eliminating the bias in favor of features with greater values. As with data transformation, data normalization is also been used for the test dataset.

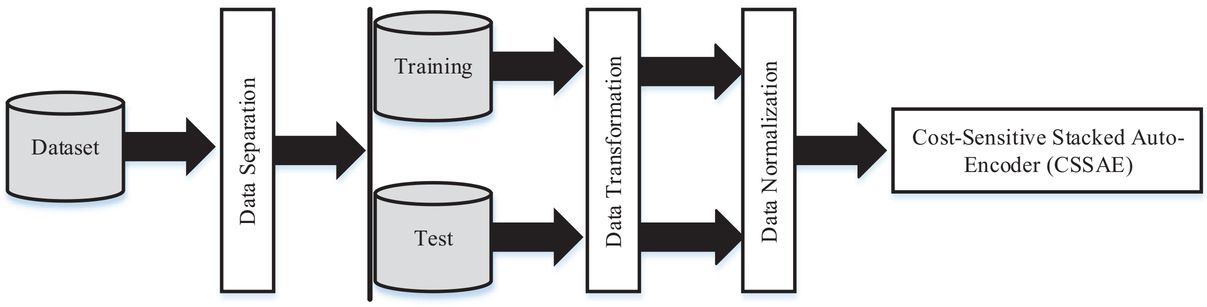


Fig. 6. Experimental setup steps.

6. Experimental results

This section provides the results obtained from experiments in terms of performance and efficiency. We have undertaken the evaluations based on a *5-class classification* and a *2-class classification* using KDD CUP 99 and NSL-KDD datasets. In 5-class classification, four types of attacks as well as normal behaviors are classified using three algorithms. CSSAE is compared with NDAE method [35] as well as applying SAE without the cost-sensitive approach. The results are presented using five measures: *accuracy, recall, precision, FAR*, and *F-Measure*. In addition, the efficiency is evaluated according to CPU time.

6.1. KDD CUP 99

Fig. 7 shows the confusion matrices of the models for KDD CUP 99 dataset in terms of two-class classification and five-class classification. In two-class classification, the algorithm builds a detection model based on normal and malicious events. Since all types of attacks are considered as malicious events, the anomaly detection rate is high for all three algorithms. This is because the attack instances greatly outnumber the normal behaviors, 62,841 (80%) against 15,787 (20%). Therefore, this kind of modeling does not reflect the impact of low-frequency attacks on the IDS. The five-class confusion matrix is a better way to show the effectiveness of anomaly detection systems. We presented these matrices on the left side of Fig. 7. Our model achieved a better anomaly detection rate for minority classes, 87% and 86% and 38% for Probe, R2L and U2R, respectively. These figures are under 0.85 for SAE and NDAE algorithms.

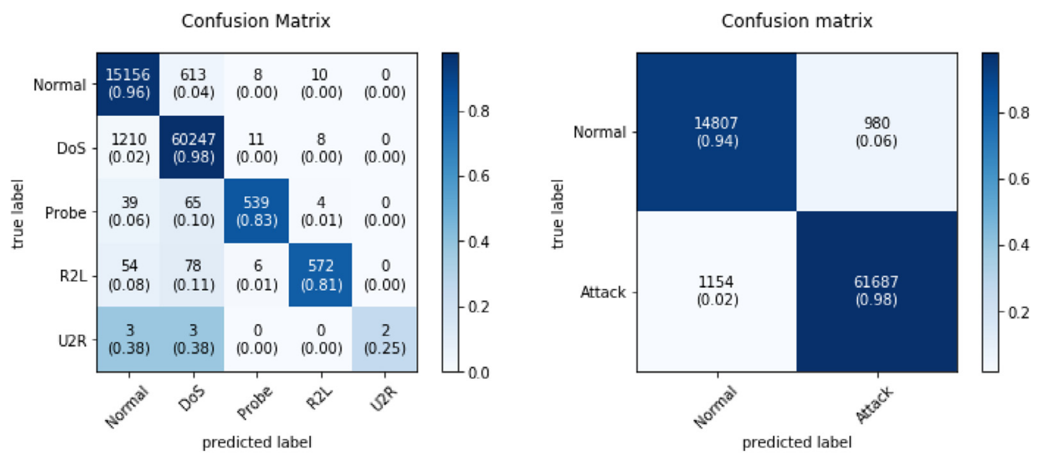
Fig. 8 presents the effectiveness of the CSSAE method against others. CSSAE outperforms the others, particularly for U2R and R2L attacks. This means that it can detect these types of attacks with more accuracy. All three methods have similar performance for identifying normal and DoS samples. The average FAR for CSSAE is 0.0057, while these figures are almost 0.016 and 0.011 for SAE and NDAE, respectively. Indeed, our proposed method can reduce the number of normal behaviors that are incorrectly detected as attacks. The main focus of our method is to reduce the impact of class imbalance on the detection rate for low-frequency attacks. According to Fig. 8, we can observe that the recall values of CSSAE for minority attacks (Probe, R2L, and U2R) are higher than that of other methods. This means that our method can correctly detect these attacks.

Fig. 9 shows the influence of epoch number on the training accuracy of models. As we can observe, the CSSAE algorithm reached the maximum accuracy at the iteration 20, 99.35%. In contrast, the training accuracies of NDAE and SAE have been maximized at iterations 30 and 35, respectively.

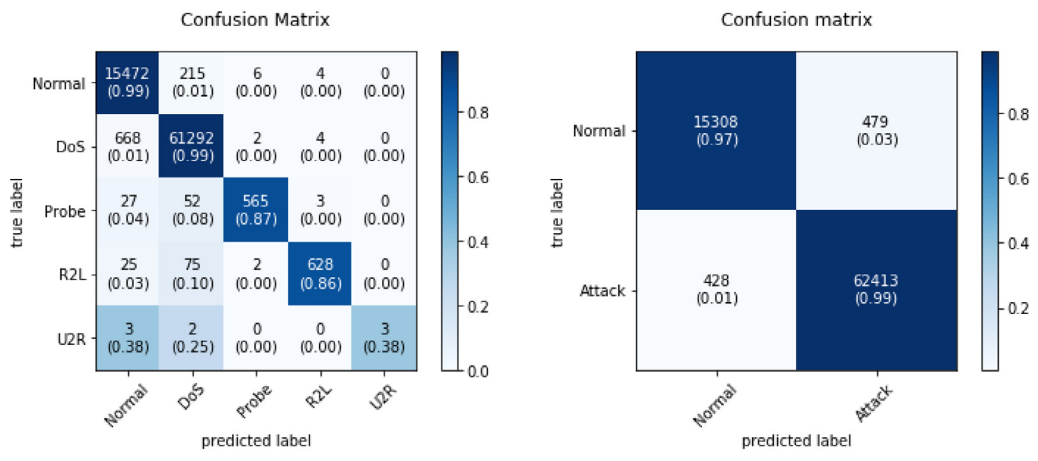
The training time required for generating an intrusion detection model is a critical issue in IDS, particularly when dealing with large scale data. Fig. 10 shows the results of the execution time comparison in terms of different neurons in hidden layers on the KDD CUP 99 dataset. The time needed for the learning model increased with the increase of neurons. However, our method can consistently reduce the required training time compared with SAE and S-NDAE. This is because CSSAE performs a data dimensionality reduction technique to reject redundant features. This leads to reducing training time. Overall, the CSSAE is up to 1.38 and 1.15 faster than the SAE and S-NDAE, respectively.

6.2. NSL-KDD

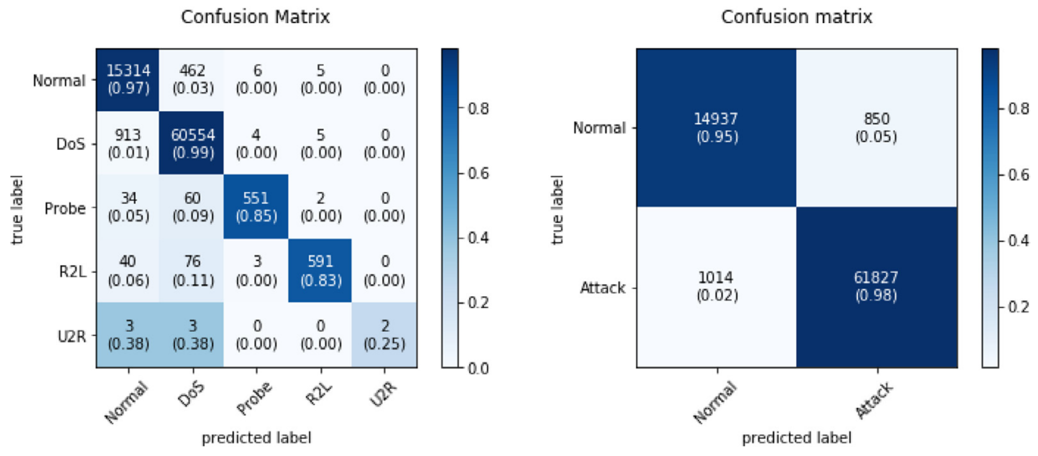
In this section, we show the performance of our model on the NSL-KDD dataset in comparison with other models. Fig. 11 shows the average results of the detailed classification for each model. The elements on the diagonal of the confusion matrix present accurately classified results. The results of the two-class classification show that our proposed algorithm can detect anomalies with an accuracy of 98%, while SAE and NDAE well-classified 97% of attacks. On the other hand, according to 5-class classification, CSSAE can detect correctly about 99% of normal and DoS events. It also achieves a good result in the classification of minority attacks, 96% for Probe and R2L attacks. The values are about 88% and 90% for SAE and NDAE, respectively. The most challenging problem of all three models is the detection of U2R attacks, which are 22%, 33%, and 56% for SAE, NDAE, and CSSAE, respectively.



(a) Confusion matrix of SAE

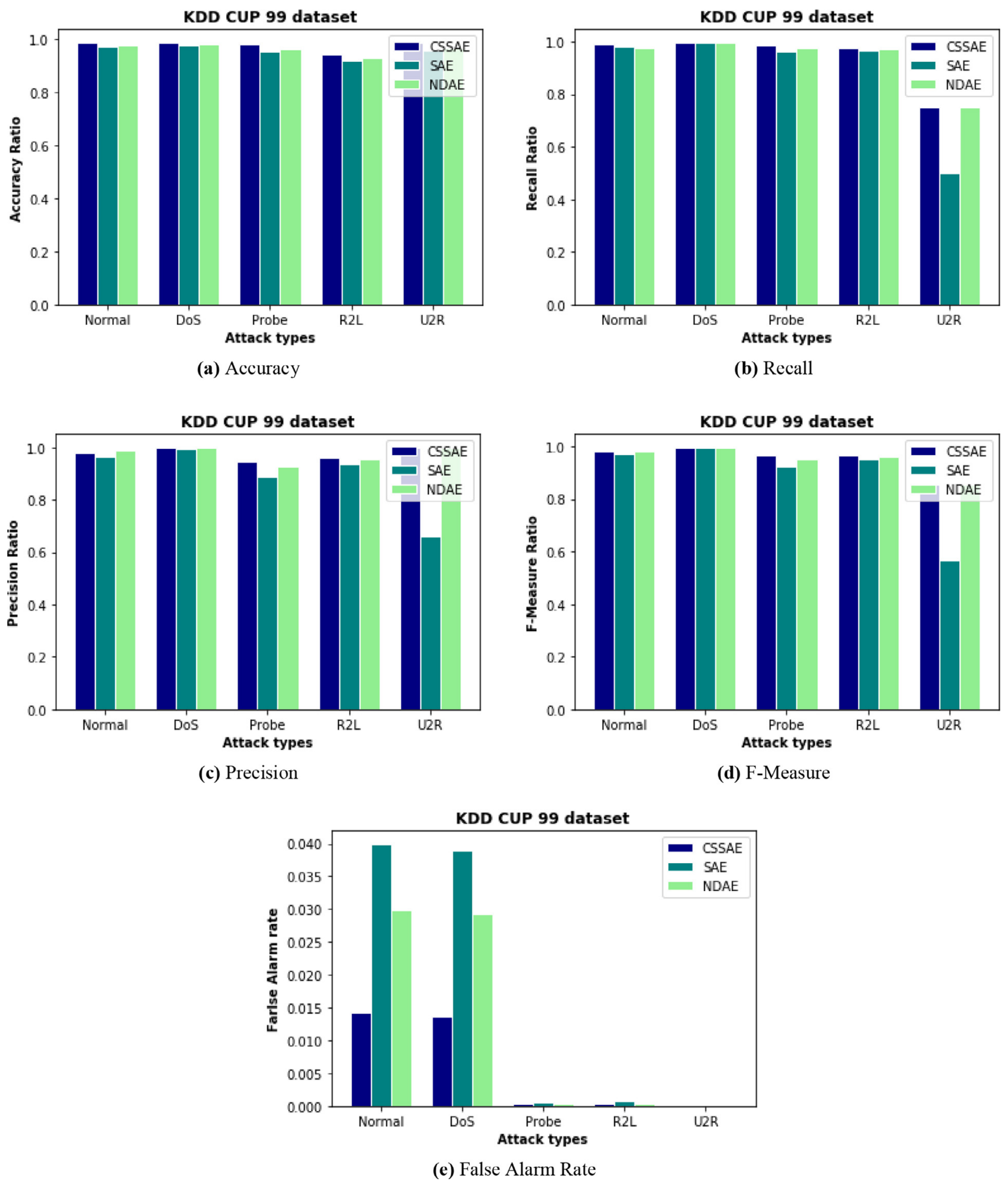


(b) Confusion matrix of CSSAE



(c) Confusion matrix of NDAE

Fig. 7. Confusion matrix of models for KDD CUP 99 dataset.



(a) Accuracy **(b)** Recall **(c)** Precision **(d)** F-Measure **(e)** False Alarm Rate

Fig. 8. The performance of CSSAE on KDD CUP 99 dataset.

Fig. 12 shows that CSSAE provided the best detection rate for all attacks. The values of this figure are derived from the confusion matrix shown in Fig. 11. Similar to KDD CUP 99 dataset, CSSAE outperforms the other approaches for detecting minority class instances (i.e. Prob, R2L, U2R attacks). The distribution ratio of U2R in NSL-KDD is much more than its ratio in the KDD CUP 99 dataset. However, our approach yielded a good detection ratio for the U2R attack compared to the two other methods. According to Fig. 12(a), the average accuracy of CSSAE is 0.996, against 0.97 and 0.98 for SAE and NDAE, respectively. As we observe in Fig. 12(b), the recall value for all approaches is lower than the other measures because it is

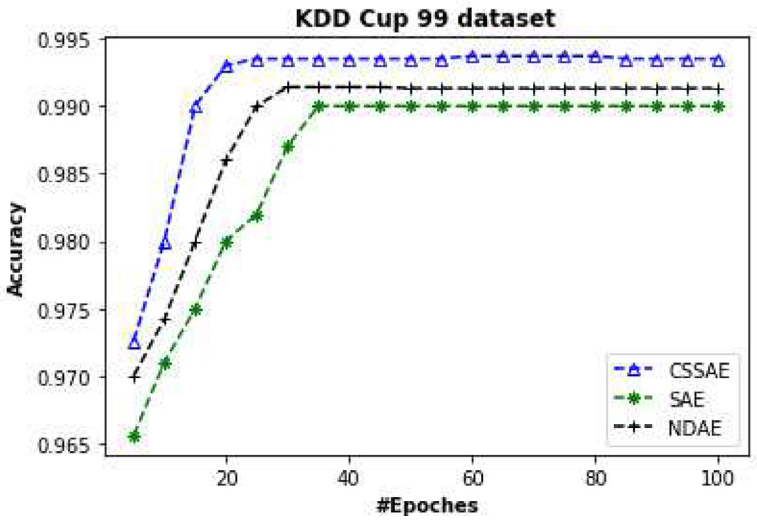


Fig. 9. The training accuracy for KDD CUP 99 dataset.

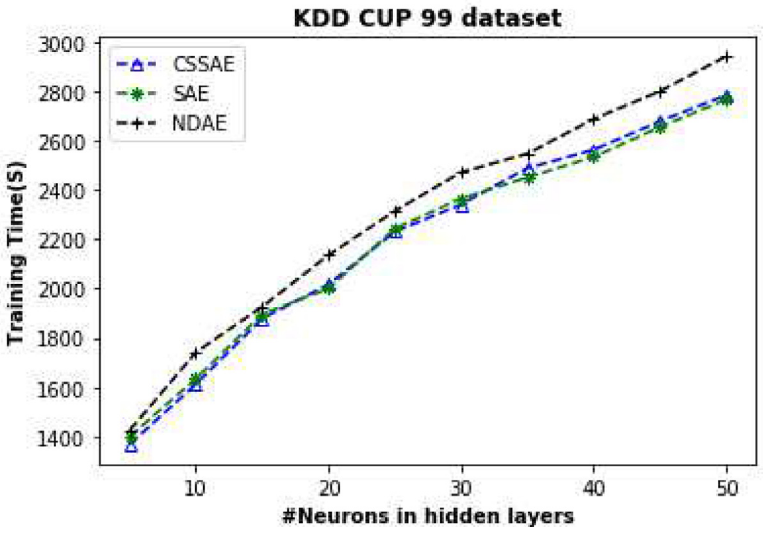
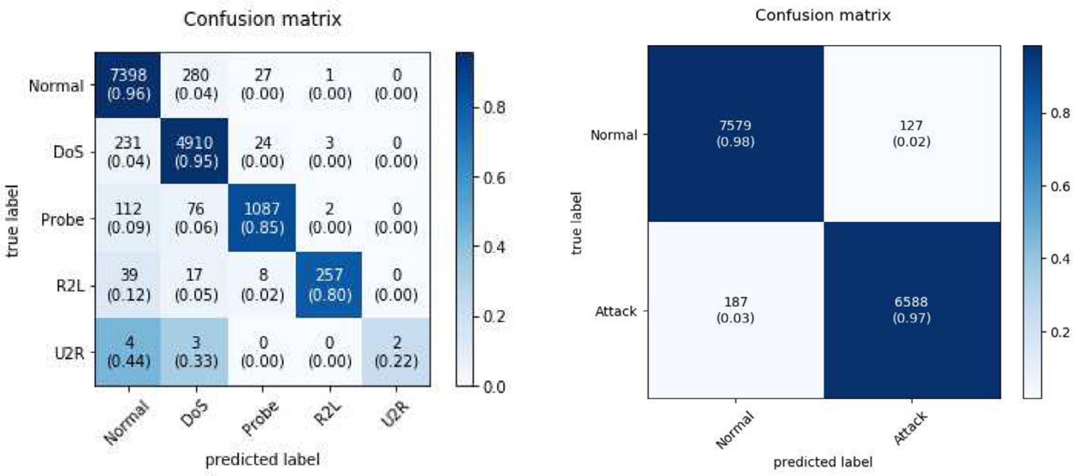


Fig. 10. The training time for KDD CUP 99 dataset.

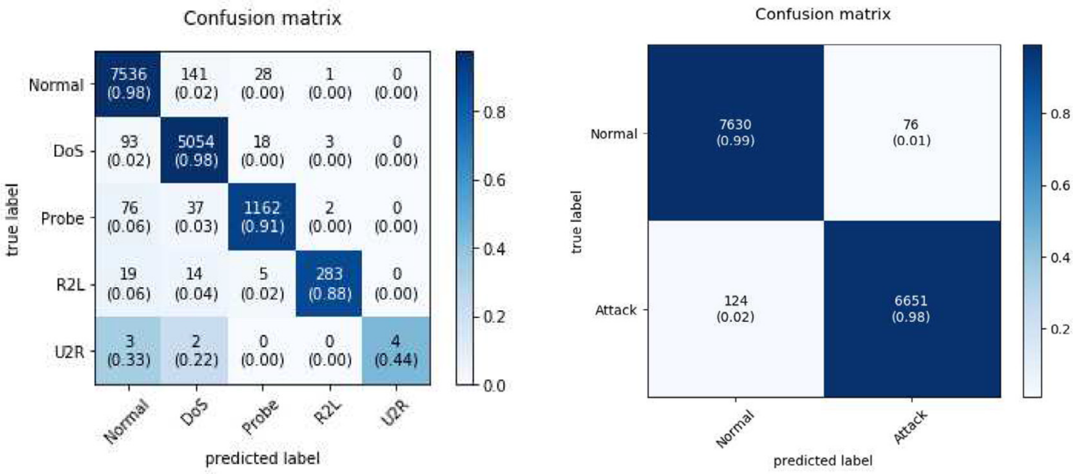
directly related to misclassifying the U2R attack as the high-frequency classes such as normal and DoS events. Figs. 12(c) and (d) prove that our proposed algorithm can identify attacks that occur much less frequently than commonly occurring intrusions. In particular, CSSAE can detect U2R attacks that are rare in the NSL-KDD dataset. Fig. 12(e) shows that the detection of minority attacks is an important challenge for IDS mechanisms. This is because the false alarm rates are high for rare attacks so that this value is above 0.4 for all three algorithms.

Fig. 13 compares the training accuracy of intrusion detection algorithms according to the increase of epochs. Unlike the KDD CUP 99 dataset (Fig. 9), algorithms achieved the maximum accuracy with more epochs: 35, 40, and 45 for CSSAE, NDAE, and SAE models, respectively. This is because the number of training records in the KDD CUP 99 dataset is greater than the NSL-KDD dataset. Therefore, detection models are trained with many more samples than the NSL-KDD.

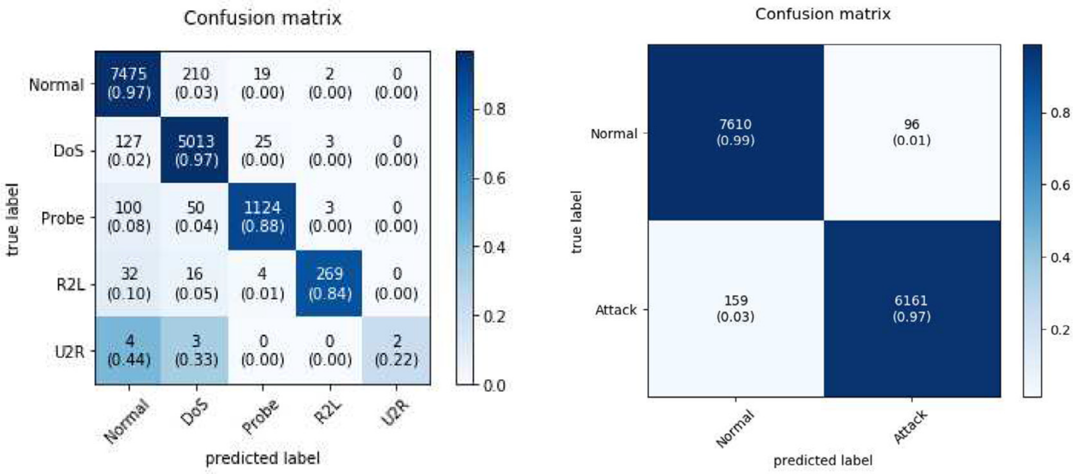
Fig. 12 shows the execution time of the training phase of the methods on the NSL-KDD dataset. The number of neurons in hidden layers varied from 5 to 50. Similar to the KDD CUP 99 dataset, the times for CSSAE and SAE are approximately similar. However, it seems that more training time seemed to be needed for CSSAE than SAE with the increasing the number of neurons. While the training time for the KDD CUP 99 dataset increased twofold when we increased the number of neurons from 5 to 50, the training time was threefold for the NSL-KDD dataset. Fig. 14.



(a) Confusion matrix of SAE

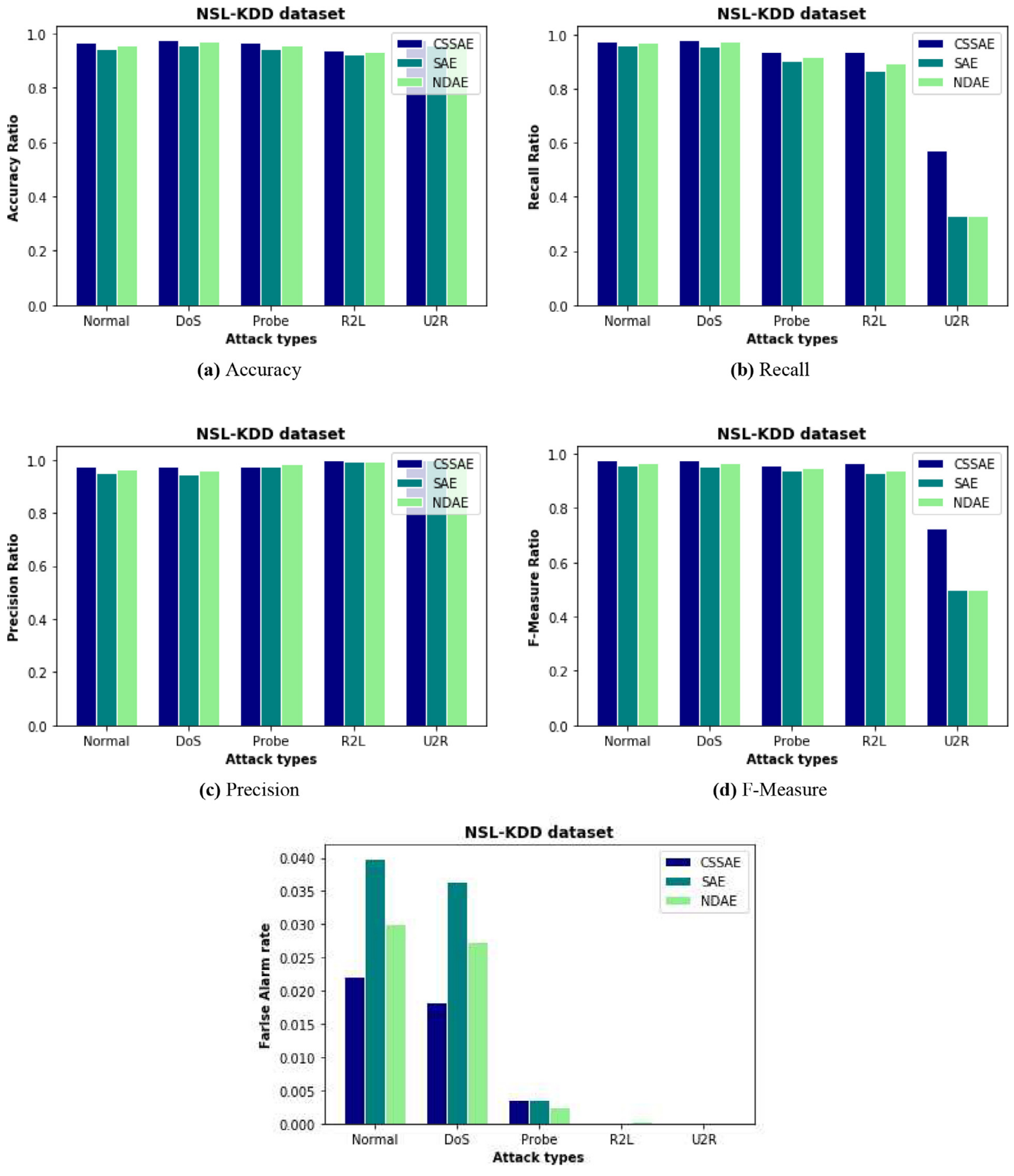


(b) Confusion matrix of CSSAE



(c) Confusion matrix of NDAE

Fig. 11. Confusion matrix of models for NSL-KDD dataset.



(a) Accuracy **(b)** Recall **(c)** Precision **(d)** F-Measure **(e)** False Alarm Rate

Fig. 12. The performance of CSSAE on NSL-KDD dataset.

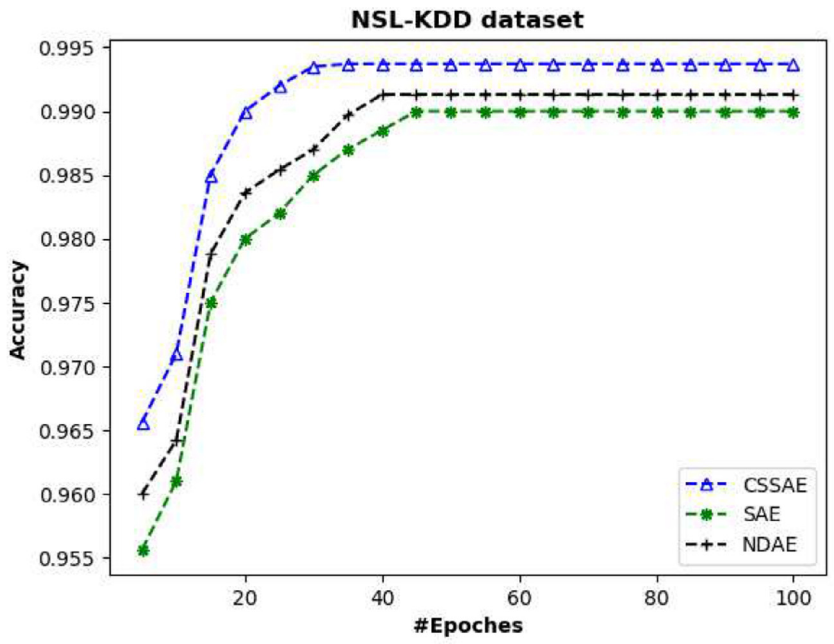


Fig. 13. The performance of CSSAE on NSL-KDD dataset.

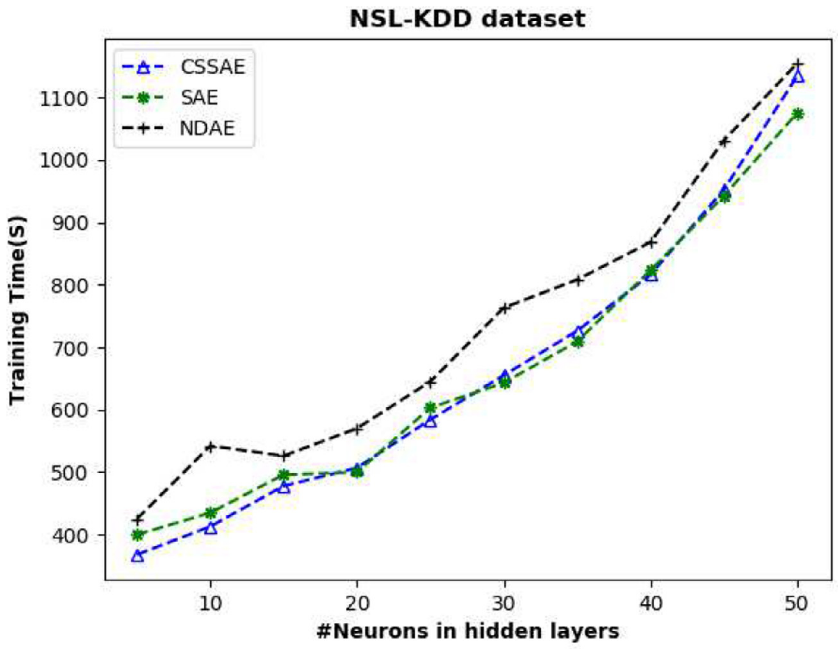


Fig. 14. The training time of algorithms for the NSL-KDD dataset.

7. Conclusion and future works

In this paper, we have confronted the problem of unbalanced data for intrusion identification in the Internet of Things. We proposed a cost-sensitive stacked auto-encoder for feature learning considering the cost of each class. Our proposed algorithm assigns a higher cost for misclassifying an abnormal example as a normal example than the cost of the reverse error. The costs are determined dynamically based on the distribution of class instances, instead of being determined by experts. Once each event is classified in the training phase, the corresponding cost is considered for parameters updating in the fine-tuning phase of algorithm. In order to evaluate the proposed method, we used the KDD CUP 99 and NSL-KDD datasets. The results demonstrated that our approach attains high levels of performance, particularly for low-frequency attacks such as Probe, U2R, and R2L.

On the one hand, the amount of traffic data generated by IoT devices is dramatically increasing. On the other hand, the traffic capacity of the Internet networks has sharply increased to facilitate the traffic passing through the networks. The modern backbone links are able to handle millions of packets per second. Therefore, IDS requires completing the analysis of a packet within a few nanoseconds. Developing a system that can investigate a large amount of data at high speed is an important requirement in the field of intrusion detection. Reducing the computational time of IDS for very large databases using parallel deep learning can be considered as a good opportunity for future work. Our proposed algorithm can be successfully applied in other areas in which class imbalance is a critical problem. Data with unbalanced distribution is a critical problem in network traffic classification that has not been considered in the learning phase of deep learning. We plan to apply this idea for encrypted traffic classification.

Declaration of Competing Interest

None.

References

[1] M. Abomhara, G.M. Køien, Security and privacy in the Internet of Things: current status and open issues, in: Proceedings of the IEEE International Conference on Privacy and Security in Mobile Systems, 2014, p. 1–8.
[2] T.M. Behera, S.K. Mohapatra, U.C. Samal, M.S. Khan, M. Daneshmand, A.H. Gandomi, Residual energy based cluster-head selection in WSNs for IoT application, IEEE Internet Things J. 6 (3) (2019) 5132–5139.
[3] J. Rivera, R. van der Meulen, Gartner says the internet of things installed base will grow to 26 billion units by 2020, 2013, (https://www.gartner.com/newsroom/id/2636073).
[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of Things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutor. 17 (4) (2015) 2347–2376.
[5] C.W. Tsai, C.F. Lai, M.C. Chiang, L.T. Yang, Data mining for internet of things: a survey, IEEE Commun. Surv. Tutor. 16 (1) (2013) 77–97.
[6] P.I.R. Grammatikis, P.G. Sarigiannidis, I.D. Moscholios, Securing the Internet of Things: challenges, threats and solutions, Internet Things 5 (2018) 41–70.
[7] M. Aly, F. Khomh, M. Haoues, A. Quintero, S. Yacout, Enforcing security in Internet of Things frameworks: a systematic literature review, Internet Things 6 (2019) 100050.
[8] J. Landford, R. Meier, R. Barella, X. Zhao, E. Cotilla-Sanchez, R.B. Bass, S. Wallace, Fast sequence component analysis for attack detection in synchrophasor networks, in: Proceedings of the Fifth International Conference on Smart Cities and Green ICT Systems, 2016, p. 1–8.
[9] O. Brun, Y. Yin, E. Gelenbe, Y.M. Kadioglu, J. Augusto-Gonzalez, M. Ramos, Deep learning with dense random neural networks for detecting attacks against IoT-connected home environments, Procedia Comput. Sci. 134 (2018) 458–463.
[10] A.A. Diro, N. Chilamkurti, Distributed attack detection scheme using deep learning approach for internet of things, Future Gen. Comput. Syst. 82 (2018) 761–768.
[11] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, J. Lloret, Conditional variational autoencoder for prediction and feature recovery applied to intrusion detection in IoT, Sensors 17 (9) (2017).
[12] M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: a survey, IEEE Commun. Surv. Tutor. 20 (4) (2018) 2923–2960.
[13] F.J. Pulgar, F. Charte, A.J. Rivera, M.J. del Jesus, Choosing the proper autoencoder for feature fusion based on data complexity and classifiers: analysis, tips and guidelines, Inf. Fus. 54 (2020) 44–60.
[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P. Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[15] Y. Fan, C. Zhang, Z. Liu, Z. Qiu, Y. He, Cost-sensitive stacked sparse auto-encoder models to detect striped stem borer infestation on rice based on hyperspectral imaging, Knowl. Based Syst. 168 (2019) 49–58.
[16] Q. Feng, Y. Zhang, C. Li, Z. Dou, J. Wang, Anomaly detection of spectrum in wireless communication via deep auto-encoders, J. Supercomput. 73 (7) (2017) 3161–3178.
[17] W.W. Ng, G. Zeng, J. Zhang, D.S. Yeung, W. Pedrycz, Dual autoencoders features for imbalance classification problem, Pattern Recognit. 60 (2016) 875–889.
[18] Q. Zhang, L.T. Yang, Z. Chen, P. Li, A survey on deep learning for big data, Inf. Fus. 42 (2018) 146–157.
[19] S.H. Khan, M. Hayat, M. Bennamoun, F.A. Sohel, R. Togneri, Cost-sensitive learning of deep feature representations from imbalanced data, IEEE Trans. Neural Netw. Learn. Syst. 29 (8) (2017) 3573–3587.
[20] B. Krawczyk, M. Woźniak, G. Schaefer, Cost-sensitive decision tree ensembles for effective imbalanced classification, Appl. Soft Comput. 14 (2014) 554–562.
[21] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Intell. Res. 16 (2002) 321–357.
[22] E. Ramentol, Y. Caballero, R. Bello, F. Herrera, SMOTE-RSB*: a hybrid preprocessing approach based on oversampling and undersampling for high imbalanced data-sets using smote and rough sets theory, Knowl. Inf. Syst. 33 (2) (2012) 245–265.
[23] Y.-A. Chung, H.-T. Lin, S.-W. Yang, Cost-aware pretraining for multiclass cost-sensitive deep learning, 2015, [Online]. Available: https://arxiv.org/abs/1511.09337.
[24] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, P.J. Kennedy, Training deep neural networks on imbalanced data sets, IEEE Int. Joint Conf. Neural Netw. (2016) 4368–4374.
[25] A. Patel, Q. Qassim, C. Wills, A survey of intrusion detection and prevention systems, Inf. Manag. Comput. Secur. 18 (4) (2010) 277–290.
[26] B.B. Zarpelao, R.S. Miani, C.T. Kawakani, S.C. de Alvarenga, A survey of intrusion detection in Internet of Things, J. Netw. Comput. Appl. 84 (2017) 25–37.
[27] R. Kozik, M. Choraś, M. Ficco, F. Palmieri, A scalable distributed machine learning approach for attack detection in edge computing environments, J Parallel Distrib Comput 119 (2018) 18–26.
[28] M. Nijim, H. Albataineh, M. Khan, D. Rao, FastDetict: a data mining engine for predecting and preventing DDoS attacks, in: Proceedings of the IEEE International Symposium on Technologies for Homeland Security, 2017, p. 1–5.
[29] H.H. Pajouh, R. Javidan, R. Khayami, D. Ali, K.-K.R. Choo, A two-layer dimension reduction and two-tier classification model for anomaly-based intrusion detection in IoT backbone networks, IEEE Trans. Emerg. Top. Comput. 7 (2) (2016) 314–323.
[30] H.-J. Liao, C.-H.R. Lin, Y.-C. Lin, K.-Y. Tung, Intrusion detection system: a comprehensive review, J. Netw. Comput. Appl. 36 (1) (2013) 16–24.
[31] R. Mitchell, I.R. Chen, A survey of intrusion detection techniques for cyber-physical systems, ACM Comput. Surv. 46 (4) (2014) 55.
[32] Q. Chen, S. Abdelwahed, A. Erradi, A model-based validated autonomic approach to self-protect computing systems, IEEE Internet Things 1 (5) (2014) 446–460.

[33] M.-O. Pahl, F.-X. Aubet, All eyes on you: distributed multi-dimensional IoT micro service anomaly detection, in: Proceedings of the Forth International Conference on Network and Service Management, 2018, p. 2018.
[34] M. Hasan, M.M. Islam, I. Islam, M.M.A. Hashem, Attack and anomaly detection in IoT sensors in IoT sites using machine learning approaches, Internet Things 7 (2019).
[35] 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) (2017) 41–50.
[36] S.J. Stolfo, W. Fan, W. Lee, A. Prodromidis, P.K. Chan, Cost-based modeling for fraud and intrusion detection: results from the JAMproject, in: Proceedings of the IEEE DARPA Information Survivability Conference and Exposition, 2000, pp. 130–144.
[37] M. Tavallaee, E. Bagheri, W. Lu, A.-A. Ghorbani, A detailed analysis of the KDD CUP 99 data set, in: Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications, 2009, pp. 53–58.
[38] A. Maratea, A. Petrosino, M. Manzo, Adjusted F-measure and kernel scaling for imbalanced data learning, Inf. Sci. 257 (2014) 331–341.