Computers & Security 135 (2023) 103502

Contents lists available at ScienceDirect

Computers & Security

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

A soft actor-critic reinforcement learning algorithm for network intrusion detection

Zhengfa Li [a,b], Chuanhe Huang [a,b,*], Shuhua Deng [c], Wanyu Qiu [a,b], Xieping Gao [d]

[a] School of Computer Science, Wuhan University, Wuhan, China
[b] Hubei LuoJia Laboratory, Wuhan, China
[c] Key Laboratory of Intelligent Computing and Information Processing of Ministry of Education, Xiangtan University, Xiangtan, China
[d] College of Information Science and Engineering, Hunan Normal University, Changsha, China

ARTICLE INFO

Keywords:
Network security
Anomaly detection
Network intrusion detection
Deep reinforcement learning
Soft actor-critic

ABSTRACT

Network intrusion detection plays a very important role in network security. Although current deep learning-based intrusion detection algorithms have achieved good detection performance, there are still limitations in dealing with unbalanced datasets and identifying minority attacks and unknown attacks. In this paper, we propose an intrusion detection model AE-SAC based on adversarial environment learning and soft actor-critic reinforcement learning algorithm. First, this paper introduces an environmental agent for training data resampling to solve the imbalance problem of the original data. Second, rewards are redefined in reinforcement learning. In order to improve the recognition rate of few categories of network attacks, we set different reward values for different categories of attacks. The environment agent and classifier agent are trained adversarially around maximizing their respective reward values. Finally, a multi-classification experiment is conducted on the NSL-KDD and AWID datasets to compare with the existed excellent intrusion detection algorithms. AE-SAC achieves excellent classification performance with an accuracy of 84.15% and a f1-score of 83.97% on the NSL-KDD dataset, and an accuracy and a f1-score over 98.9% on the AWID dataset.

1. Introduction

Intrusion detection technology employs an active defense strategy that enables timely and accurate warning before intrusions have a bad impact on computer systems, and builds a resilient defense system in real time to avoid, transfer, and reduce the risks faced by information systems. Intruders are detected and responded to in a timely manner through continuous monitoring and analysis of network activity. Without a doubt, intrusion detection plays a critical role in network security. Based on intrusive behaviors, intrusion detection systems (IDS) can be divided into two types: host-based intrusion detection systems (HIDS) and network-based intrusion detection systems (NIDS) (Mishra et al., 2018). HIDS monitor individual hosts or devices and send alerts to users when suspicious activity is detected. Unlike HIDS, NIDS are usually placed at network points such as gateways or routers to check for intrusions in network traffic. In this paper, we will focus on network intrusion detection algorithms.

Machine learning methods for intrusion detection have been studied by researchers for over 20 years. The large amount of network telemetry and other types of security data makes the intrusion detection problem solvable by machine learning methods. These methods can be broadly classified into two categories: shallow learning or classical models and deep learning models (Gamage and Samarabandu, 2020). The structure of a shallow learning model can basically be viewed as having a layer of hidden layer nodes (e.g., SVM, Boosting), or no hidden layer nodes (e.g., Logistic Regression). The intrusion detection model based on shallow machine learning algorithm has problems such as difficulty in processing large-scale network intrusion data (Shone et al., 2018; Zhang et al., 2020a), poor ability to identify various new attacks, high false positive rate, and over-reliance on researchers for feature design and feature selection (Thaseen and Kumar, 2017). Compared to shallow learning, currently used deep learning models are neural network models with a large number of hidden layers. These models are capable of learning highly complex nonlinear functions, and the hierarchical arrangement of layers enables them to learn useful feature representations from the input data (Dong and Wang, 2016; Pingale and Sutar, 2022; Hou et al., 2022), which can overcome the limitations of classical models. In fact, deep learning has yielded good results in the field of intrusion detection.

* Corresponding author at: School of Computer Science, Wuhan University, Wuhan, China.
E-mail addresses: zhengfali@whu.edu.cn (Z. Li), huangch@whu.edu.cn (C. Huang), shuhuadeng@xtu.edu.cn (S. Deng), wanyu3135@gmail.com (W. Qiu), xpgao@hunnu.edu.cn (X. Gao).

https://doi.org/10.1016/j.cose.2023.103502
Received 25 November 2022; Received in revised form 15 March 2023; Accepted 20 September 2023
Available online 26 September 2023

For example, Cil et al. (2021) proposed the deep neural network (DNN) model that has an attack detection success rate (the attack detection success rate can be understood as the precision of the intrusion detection model) of 99.99% on the CICIDS2019 (Sharafaldin et al., 2019) dataset for network traffic and an accuracy of 94.57% for classification of attack types.

In addition to DNN models, Convolutional Neural Network (CNN), Gated Recurrent Units (GRU), Long-Term Short-Term Memory (LSTM), Deep Belief Network (DBN), Stacked Autoencoder (SAE) and hybrid neural network models are also widely used for intrusion detection. Although the application of deep learning in the field of intrusion detection has achieved expected research results, there are still many problems to be solved. On one hand, deep learning models are very sensitive to the dataset used for training. However, the network intrusion detection data obtained in the real network environment often contains a large amount of normal behavior data and a small number of attack behavior data, resulting in an extremely unbalanced dataset. This leads to poor recognition of a few intrusions by deep learning models. On the other hand, the building of good models is extremely difficult. As we all know, the number of parameters for deep learning models is huge and tuning them is time consuming.

The emergence of deep reinforcement learning (DRL) has given a new approach to solve the intrusion detection problem. In fact, the intrusion detection problem can be viewed as an optimal decision problem; that is, given any network traffic, the IDS must determine whether it is a normal or anomalous flow, or whether it falls into the category of attack traffic. Lopez-Martin et al. (2020) used four deep reinforcement learning algorithms Deep Q-Network (DQN), Double Deep Q-Network (DDQN), Policy Gradient (PG), Actor-Critic (AC) for intrusion detection. It is very noteworthy that, in contrast to deep learning models, deep reinforcement learning models do not require complex neural network model[1] and often require only a few simple layers of neural networks. Although these reinforcement learning algorithms have achieved good results in intrusion detection, there are still challenges in dealing with data imbalance and multiple classifications of cyber attacks. In order to solve the network intrusion detection problem using reinforcement learning, it is necessary to map the label space of intrusion detection (the number of categories of network attacks) to the action space of reinforcement learning. This implies that if there are more categories of network attacks, then reinforcement learning algorithms are required to have more action space exploration capabilities.

In this paper, we propose a deep reinforcement learning model based on adversarial learning for network intrusion detection. The Soft Actor-Critic (SAC) (Haarnoja et al., 2018a,b; Christodoulou, 2019) model with the best exploration ability as the base model for this paper. First, we modify the original reinforcement learning framework by adding an additional environment agent. The role of the environment agent is to resample the original training dataset and change its imbalance. The resampled dataset is used to train the main agent (classifier agent). Second, the reward function is redefined and the rewards of the classifier agent and the environment agent form an adversarial relationship. Different reward values are set for different classes of network attacks, giving larger rewards for the minority attacks and smaller rewards for the majority attacks. Then, the classifier agent and the environment agent maximize their respective reward rewards through an adversarial learning model. Finally, the policy learned by the classifier is used to detect anomalous traffic. The model of this paper was evaluated on the public dataset NSL-KDD (Tavallaee et al., 2009) and Aegean WiFi Intrusion Dataset (AWID) (Kolias et al., 2015), and our algorithm achieved excellent results in terms of accuracy, precision, and recall and f1-score when compared with the current state-of-the-art algorithms in the multi-classification case.

The main contributions of the work are as follows:

- We formalize the intrusion detection problem as a Markov decision problem and propose an intrusion detection algorithm based on SAC deep reinforcement learning. On top of the original reinforcement learning framework, additional environmental agents are introduced for resampling the original training dataset.
- We redefine the reward function in reinforcement learning to better handle the imbalance of the original dataset. Different reward values are set for different classes of network attacks. For a few categories of attacks, the classifier agent receives a higher reward value if it identifies successfully, otherwise the environment agent receives a higher reward value. The classifier agent and the environment agent are trained in parallel in an adversarial manner.
- We choose excellent deep learning models and reinforcement learning models as comparison algorithms and conduct multi-classification experiments on the publicly available datasets NSL-KDD[2] and AWID.[3] The code of AE-SAC can be found in the GitHub repository.[4]

The rest of this paper is organized as follows. Section 2 presents the current state of network intrusion detection research. Section 3 describes the proposed intrusion detection framework. Section 4 discusses the experimental results, and section 5 summarizes the study's findings and looks ahead to future research.

2. Related work

In this section, we summarize the research progress related to intrusion detection from the perspective of deep learning and reinforcement learning.

2.1. Deep learning-based intrusion detection

Shone et al. (2018) proposed an intrusion detection model based on the unsupervised feature learning method of nonsymmetric deep autoencoder, and evaluated in the KDDCup 99[5] and NSL-KDD datasets have yielded promising results.

Muna et al. (2018) proposed an intrusion detection model for Internet industrial control systems based on deep autoencoders and deep feedforward neural networks. According to the experimental results, the model has a higher detection rate and a lower false positive rate.

Zhang et al. (2020b) proposed a hybrid intrusion detection model based on multiscale CNN (MSCNN) and LSTM. The model first analyzes the spatial features of the dataset with a MSCNN, and then processes the temporal features with a LSTM. Finally, the model classifies using spatiotemporal features.

Popoola et al. (2020) proposed a hybrid LSTM-Autoencoder and Bidirectional LSTM (LAE-BLSTM) intrusion detection method. LAE reduces the dimensionality of network traffic features, freeing up memory space and increasing computational speed. BLSTM, on the other hand, learns the long-term temporal relationships between low-dimensional features in order to correctly distinguish benign traffic from various types of botnet attack traffic.

To detect network intrusions, Hassan et al. (2020) proposed a hybrid deep learning model based on CNN and weight-dropped long short-term

[1] The complexity of neural networks consists of spatial complexity and temporal complexity. The spatial complexity is represented by the number of layers of the neural network and the number of parameters to be optimized in the neural network. The time complexity can be measured by the number of floating point operations. Undoubtedly, both spatial and temporal complexity are closely related to the number of hidden layers and the number of neurons per layer.

[2] https://www.unb.ca/cic/datasets/nsl.html.

[3] https://icsdweb.aegean.gr/awid/.

[4] https://github.com/ZhengfaLi1992/AE-SAC/tree/master.

[5] https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

memory networks (WDLSTM). To avoid overfitting, the model employs CNN to extract meaningful features from IDS big data, while WDLSTM preserves long-term dependencies among the extracted features.

Wang and Li (2021) created a hybrid neural network architecture that combines transformers and CNN to detect DDoS attacks (DDosTC) on Software-defined networking, and tested it on the most recent dataset CICDDoS2019 (Sharafaldin et al., 2019). The experimental results show that DDosTC outperforms the current optimal model.

Han et al. (2021) proposed a network attack detection model combining sparse autoencoder and kernel, and used an iterative method of adaptive genetic algorithm to optimize the objective function of the combined kernel with sparse autoencoder. The model was also trained and evaluated using a dataset based on IoT botnet attacks.

Imran et al. (2022) proposed an intelligent and efficient network intrusion detection system based on deep learning. This paper uses a novel stacked asymmetric deep auto-encoder for unsupervised feature learning combined with a support vector machine classifier for network intrusion detection. The experimental evaluation was performed on KDDCup 99 with an accuracy of 99.65%.

Lan et al. (2022) proposed a multi-task learning model with hybrid deep features. Based on a CNN with embedded space and a channel attention mechanism, two auxiliary tasks (an auto-encoder enhanced with a memory module and a distance-based prototype network) are introduced in an innovative way to improve the model's generalization ability and mitigate performance degradation in an unbalanced network environment.

2.2. Reinforcement learning-based intrusion detection

DRL has sparked a flurry of research and applications since AlphaGo defeated human Go masters (Demis, 2016). DRL is currently producing promising research results in the field of intrusion detection.

Sethi et al. (2020b) created a context-adaptive intrusion detection model that detects and classifies new and complex attacks using multiple independent deep reinforcement learning agents distributed throughout the network. The model was extensively tested on the NSL-KDD, UNSW-NB15 (Moustafa and Slay, 2015),[6] and AWID datasets and demonstrated superior accuracy and lower false positive rate when compared to state-of-the-art systems.

Caminero et al. (2019) presented the first application of adversarial reinforcement learning in intrusion detection, as well as a new technique for incorporating environmental behavior into the learning process of improved reinforcement learning algorithms. This model incorporates a reinforcement and supervisory framework to generate environments that interact with pre-recorded sample data sets formed by network features and relevant intrusion labels, and it chooses samples with optimal policies to achieve the best classification results.

Ma and Shi (2020) proposed an intrusion detection model combining reinforcement learning and class imbalance techniques based on the literature (Caminero et al., 2019). This model addressed the class imbalance issue by introducing an adaptive SMOTE (Synthetic Minority Over-Sampling Technique)[7] and re-modeling the behavior of the environment agents to improve performance.

Sethi et al. (2020a) investigated intrusion detection in cloud platforms, taking into account their vulnerability to novel attacks and the inability of existing detection models to strike a balance between high accuracy and low false positive rate. The authors proposed an adaptive cloud IDS architecture based on DRL that addresses the limitations mentioned above while also providing accurate detection and fine-grained classification of novel and complex attacks.

Lopez-Martin et al. (2020) designed the improved the classical DRL model to allow it to be used for intrusion detection, replacing the environment in the traditional DRL with a sampling function for training intrusion data that generates rewards based on error detection during the training phase. The literature presents the experimental results of four deep learning algorithms, DQN, DDQN, PG, and AC, on NSL-KDD and AWID datasets, with DDQN outperforming the others.

Dong et al. (2021) proposed a semi-supervised double-depth Q network-based network anomaly traffic detection method (SSDDQN). SSDDQN employs a auto-encoder to reconstruct network traffic features, while the K-means algorithm is used to improve the model's ability to detect unknown attacks. The model is tested on the NSL-KDD and AWID datasets and performs well in several metrics.

Zhou et al. (2021) designed an adaptable asynchronous advantage actor-critic reinforcement learning model for intrusion detection. For both sequence and image anomalies, the model employs an attention mechanism neural network and a convolutional neural network. The model was tested on the NSL-KDD, AWID, and CICIDS 2017[8] datasets (Jazi et al., 2017), and it outperformed or achieved comparable results to other anomaly detection models.

Sethi et al. (2021) developed a distributed attack detection method based on DRL that employs DQN across multiple distributed agents while efficiently detecting and classifying advanced network attacks through the use of attention mechanisms. The model was evaluated on the NSL-KDD and CICIDS 2017 datasets and achieved excellent results in terms of accuracy, precision, and recall compared to the state-of-the-art methods.

Alavizadeh et al. (2022) proposed a network intrusion detection method based on Q Learning (QL) and deep feed forward neural networks. The method uses DQL to provide continuous automatic learning for the network environment, while detecting various types of network intrusions using automatic trial-and-error methods and continuously improving its detection capability.

In conclusion, researchers in the field of intrusion detection, whether deep learning or reinforcement learning, have focused on the design of detection models while ignoring the imbalance of the original dataset. Caminero et al. (2019) employs adversarial learning to sample balanced training data with environmental agents, while Ma and Shi (2020) introduces the SMOTE technique to address training data imbalance. Despite the fact that these two models produced good results, there is still much room for improvement. We address the imbalance of the original training data in this paper from two perspectives: the selection of the reinforcement learning model and the design of the reward function. On the other hand, the existing intrusion detection model (Cil et al., 2021; Imran et al., 2022), although achieving excellent classification performance, however, focuses more on binary classification and neglects multi-categorization of anomalous traffic. In fact, in the field of network intrusion, if we can classify the attack categories in more detail, it plays a crucial role in the subsequent defense. The focus of this paper is on the multi-categorization of network intrusions.

3. Proposed method for network intrusion detection

We propose the AE-SAC method for network intrusion detection, which is based on adversarial learning and the SAC reinforcement learning model. As illustrated in Fig. 1, the AE-SAC model includes an environmental agent for data sampling. In fact, both environment agent and classifier agent are implemented with SAC model. Therefore, we proceed to introduce the SAC model in detail before introducing the AE-SAC model proposed in this paper. The meanings of the symbols used in the paper are listed in Table 1.

3.1. Soft actor-critic

Unlike general reinforcement learning, which maximizes the cumulative reward expectation by learning a policy, the SAC algorithm

[6] https://research.unsw.edu.au/projects/unsw-nb15-dataset.
[7] https://imbalanced-learn.org/stable/.
[8] https://www.unb.ca/cic/datasets/dos-dataset.html.

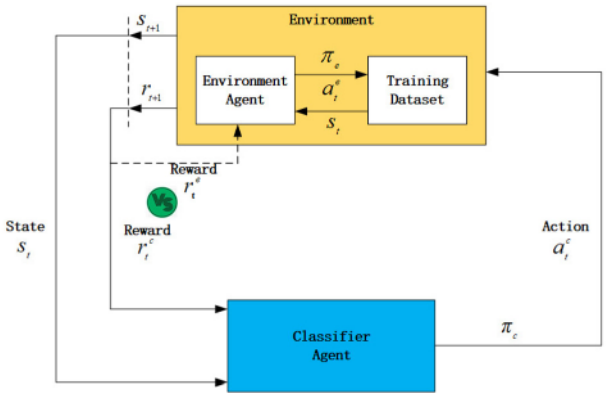

Fig. 1. AE-SAC Framework, Environment Agent: Resampling of the training set, Classifier Agent: Learning strategies for network traffic identification.

Table 1
Summary of symbols and their meanings.

Symbols	Description
s_t	the state at moment t
a_t	the action at moment t
r_t	the reward value obtained at moment t
$R(a,s)$	the reward value obtained by performing action a in state s
π	policy function, it represents the decision component, that is, what action a should be taken in state s
$\pi(a\|s)$	a conditional probability distribution function, that is, the probability of taking action a state s
$H(\pi(\cdot\|s_t)$	the entropy of the policy π at the state s_t
Q(a,s)	the action state value function, that is, the value obtained by performing action a in state s
V(s)	the value function of the state s
γ	the discount rate of reward
α	the temperature parameter that determines the importance of entropy
M	a batch of historical data taken from the experience replay memory

requires that the action entropy of each policy output be maximized in addition to the basic goal stated above. This means that SAC must learn an optimal policy ($\pi^\star$) that maximizes the sum of the cumulative reward value and the policy's entropy value, as shown in equation (1).

$$\pi^\star = \arg\max_{\pi} \mathbb{E} \sum_t (R(a_t, s_t) + \alpha H(\pi(\cdot|s_t))), \tag{1}$$

where α determines the relative importance of the entropy term with respect to the reward and is referred to as the temperature parameter, while $H(\pi(\cdot|s_t)$ is the entropy of the policy π at the state s_t. It is calculated as:

$$H(\pi(\cdot|s_t)) = -\log \pi(\cdot|s_t). \tag{2}$$

For more theoretical derivations and details, please refer to references (Haarnoja et al., 2018a,b; Christodoulou, 2019). In this paper, we will focus more on applying the discrete SAC algorithm to solve the intrusion detection problem.

3.1.1. Overview of SAC

As shown in Fig. 2, the SAC architecture contains five neural networks, including an Actor network, two V Critic networks (evaluation network and target network), and two Q Critic networks (Q_0 and Q_1 network). The Actor network is used to obtain the probability distribution of the actions, the V network is used to estimate the state values, and the Q network is used to estimate the state action values.

In the SAC algorithm, we introduce an experience replay (Mnih et al., 2013), which is similar to reinforcement learning algorithms such as DDQN. Assume the state at time t is s_t, and the probability $\pi(a|s_t)$ of all actions is obtained via the Actor network, and then the action a_t is sampled based on the probability $\pi(a|s_t)$, and then a_t is input to the environment to obtain r_{t+1} and s_{t+1}, resulting in an experience

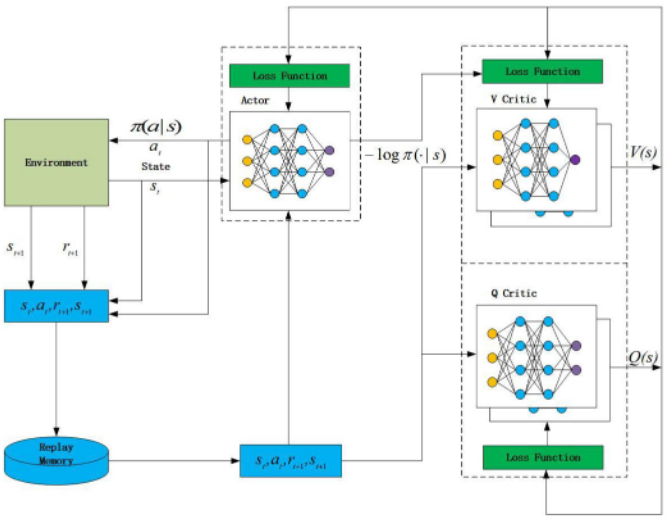

Fig. 2. SAC Framework, Actor Network: action probability distribution function, V Critic Network: estimate the state value, Q Critic Network: estimate the state action values, Replay Memory: store experience data.

$(s_t, a_t, r_{t+1}, s_{t+1})$, which is then added to the experience replay memory. In reinforcement learning, the current state s_t of the environment is frequently transferred from the previous state s_{t-1}, and the future state s_t of the environment is transferred from the current state. Therefore, the experience data has a temporal dependency (Mnih et al., 2013). The purpose of the experience replay is to break up the correlation of experiences (Schaul et al., 2015; Zhang and Sutton, 2017), and then randomly select a batch of experiences from the experience replay memory when training the neural network, so that the neural network can be trained better.

Table 2 lists the inputs, outputs, and loss functions of the Actor, Q Critic, and V Critic networks. a'_t is not the action a_t in an experience drawn from the experience replay memory, but rather all possible actions predicted by reusing the Actor network. The action state value function Q_0 can be replaced by Q_1. M represents a batch of historical data taken from the experience replay memory. $\mathbb{E}$ denotes expectation. Two Q networks are trained alone using the same update method and MSE (Mean Square Error) as the loss function. One of the two V networks is the evaluation network and the other network is the target network. The target network is updated using soft update.

3.1.2. RL paradigm for intrusion detection

In fact, the intrusion detection problem can be viewed as a reinforcement learning problem with a discrete action space. The current DRL input has two parts: state and action. Furthermore, the dataset we use includes two components: network traffic features and feature labels. To reconcile network traffic elements with the DRL concept, we consider network traffic features to be states and feature labels to be actions. As shown in the upper part of Fig. 3, $S = \{s_0, s_1, ..., s_n, s_{n+1}\}$ is network traffic feature and $A = \{a_0, a_1, ..., a_n, a_{n+1}\}$ is the feature label.

General reinforcement learning uses random sampling to read the original dataset and begins sampling at time t. Each sample is made up of three parts: (1) network traffic characteristics s_t at time t. (2) network traffic real labels a_t at time t (3) network traffic characteristics s_{t+1} at time $t+1$. Before random sampling, the original data is divided into $N+1$ random samples and randomly arranged. In this paper, we use an environment agent for random sampling, please refer to subsection 3.2.3 for details.

3.2. AE-SAC intrusion detection method

In the previous section, we introduced the network structure and update method of the SAC model. In this section, we will describe in detail how to use the SAC model for intrusion detection.

Table 2
Inputs, outputs and loss functions of each network (Actor, Q Critic and V Critic) in the SAC network architecture.

Network	Input	Output	Loss Function
Actor	state s_t	action probability $\pi(a\|s)$	$L_A = -\frac{1}{\|M\|}\sum_{e\in M}\mathbb{E}_{a'_t\sim\pi(\cdot\|s)}[Q_0(a'_t,s_t)-\alpha\log(\pi(a'_t\|s_t))], e=(s_t,a_t,r_{t+1},s_{t+1})$
Q Critic	state s_t	action state value $Q(a_t,s_t)$	$L_Q = \frac{1}{\|M\|}\sum_{e\in M}[Q(s_t,a_t)-U_t^q]^2,\ U_t^q = r_t+\gamma V(s_{t+1})$
V Critic	state s_t	state value $V(s_t)$	$L_V = \frac{1}{\|M\|}\sum_{e\in M}[V(s_t)-U_t^v]^2,\ U_t^v = \mathbb{E}_{a'_t\sim\pi(\cdot\|s_t)}[\min_{i=0,1}Q_i(a'_t,s_t)-\alpha\log(\pi(a'_t\|s_t))]$

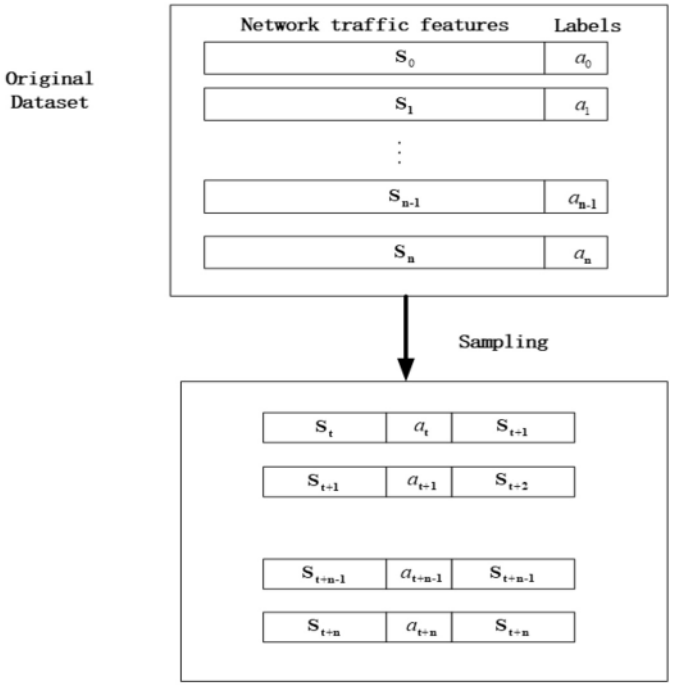

Fig. 3. Random sampling from the original dataset, network traffic features are mapped to the state space and traffic labels are mapped to the action space.

3.2.1. Overview of AE-SAC

Network intrusion data sets are typically uneven, with a high proportion of normal traffic and very little abnormal traffic. Even if the random sampling method (Lopez-Martin et al., 2020) is used, the sampled data still contains a significant amount of normal traffic, making it impossible to solve the unbalanced data problem. Therefore, as shown in Fig. 1, we modified the original reinforcement learning framework to include an environmental agent that provides intelligent behavior for the environment beyond random dataset sampling. This modification enables us to address the issue of dataset imbalance by employing an environment agent that performs dynamic and intelligent resampling of the dataset during training.

The environment agent and the classifier agent learn adversarially iteratively, with the ultimate goal of teaching the classifier agent a near-optimal policy. The environment agent is in responsible of sampling from the training set to provide training data for the classifier, and the classifier agent trains and learns using the data provided by the environment agent. If the classifier successfully classifies the batch of training data collected by the environment agent, the classifier is rewarded; otherwise, the environment agent is rewarded.

Following that, we will go over the design of the reward function, the sampling of the environment agents, and the training of the two agents.

3.2.2. Design of the reward function

The environment agent and classifier agent perform adversarial learning around obtaining the maximum reward. The environment agent gets the maximum reward by sampling more data that the classifier cannot recognize or classifies incorrectly. Conversely, classifiers receive a larger reward by recognizing or correctly classifying more data. As a result, the design of the reward function is critical.

Unlike the simple 0/1 reward function taken in the literature (Lopez-Martin et al., 2020; Caminero et al., 2019), this paper modifies the 0/1 reward function on this basis. The goal of introducing environmental agents in this paper is to sample more balanced data; however, because the proportion of normal traffic in the training dataset is much larger, there is a different proportion of traffic in different categories even in abnormal traffic. In order to enable the classifier to identify anomalous traffic in a few categories, this paper assigns a higher reward value to the anomalous traffic in a few categories. Equations (3) and (4) describe the settings of the classifier agent and environment agent reward values, respectively.

$$r_{t+1}^c = \begin{cases} 0, & a_t^c != a_t, \\ 1, & a_t^c = a_t, a_t \in A_L, \\ 2, & a_t^c = a_t, a_t \in A_M, \end{cases} \tag{3}$$

$$r_{t+1}^e = \begin{cases} 0, & a_t^c = a_t, \\ 1, & a_t^c != a_t, a_t \in A_L, \\ 2, & a_t^c != a_t, a_t \in A_M, \end{cases} \tag{4}$$

where a_t^c is the label chosen by the classifier agent for the current flow (s_t represents the network traffic characteristics of the current data flow). a_t is the actual label for that flow, A_L indicates that the flow belongs to the category with a high percentage and A_M indicates that the flow belongs to the category with a low percentage. The design of r_{t+1}^c and r_{t+1}^e is consistent with the pattern of environment agents and classifier agents confronting each other around reward values. In fact, this method of designing reward values may not be optimal, but it has been proven to be more effective through extensive experiments.

3.2.3. Sampling of training data

As shown in Fig. 4, this paper uses an environmental agent to sample the training dataset. The environment agent uses the policy π_e to select an action label a_t^e for state s_t. This action label a_t^e is then used to select the next state s_{t+1} (the actual action label a_t is also obtained at this point) in the training set. Next, the action label a_t^c of state s_{t+1} is obtained using the classifier agent's policy π_c and the reward r_{t+1}^e for the environment agent and the reward r_{t+1}^c for the classifier agent are calculated. At this point, we obtain a pair of experience data $(s_t, a_t^e, r_{t+1}^e, s_{t+1})$ and $(s_t, a_t^c, r_{t+1}^c, s_{t+1})$, and store them in the respective experience replay memory. Subsequently, when we have obtained a certain amount of empirical data, we update the two agents to obtain new policies using the update method of the SAC model introduced above. Finally the environment agent continues the data sampling using the new policy.

It is worth noting that the initial state s_0 is randomly selected from the training set. The output of policies π_e and π_c have probability values for each action, which are selected according to the probability of the action when selecting a_t^e and a_t^c. Consistent with literature (Caminero et al., 2019), the spaces of action A_e and action A_c can be different on the NSL-KDD dataset. The classifier agent has a set of actions ($A_c \in [0,4]$) corresponding to the number of classes of the classifier. On the other hand, the environment agent has a set of actions ($A_e \in [0,22]$) corresponding to each of the possible attacks in the training data set.

Fig. 5 depicts the distribution of the training data after sampling through the environmental agent. Initially (epoch 0), the environment

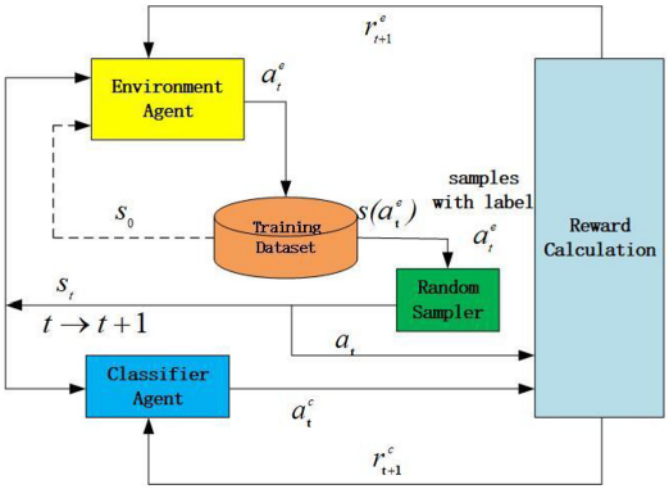

Fig. 4. Sampling with environmental agent, Environment Agent: output the action label for the next sample state s_{t+1}, Classifier Agent: predicted action labels for output state s_t, Random Sampler: randomly select the next state from the data with the same label, Reward Calculation: calculate the reward value obtained by the classifier agent and the environment agent.

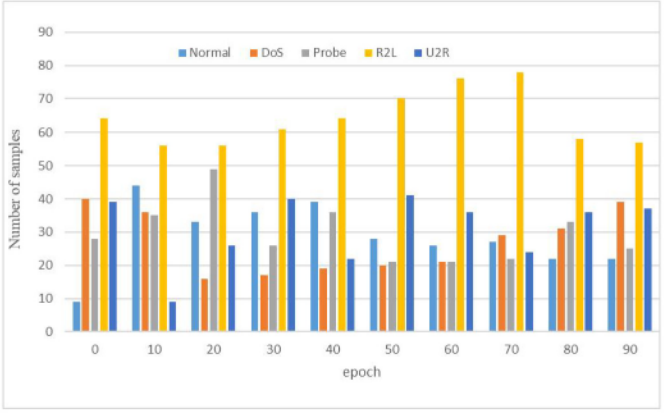

Fig. 5. The distribution of attacks and normal data generated by the environment agent during training (NSL-KDD dataset).

agent is randomly provided with training data. As training proceeds, the environment agent learns and samples the training data that maximizes its reward. We clearly observe that the data volume of the R2L attack category is much higher than that of the other categories, mainly because we set its reward value high. The main reason why the U2R category is not high may be that the number of U2Rs in the training set is too small, and the classifier agents are better at classifying and giving less reward to the environment agents. Nevertheless, the data volume of U2R is still slightly higher than that of the other categories. The unevenness of the dataset is largely reduced by resampling of the environment agents.

3.2.4. Training process of AE-SAC model

The environment agent and the classifier agent can be trained in a parallel manner. During the training process, a certain amount of empirical data is taken from the respective experience replay memory for network training. Algorithm 1 describes the training process for both agents.

In Algorithm 1, the $Reward(a_t^c, a_t)$ function calculates the reward values r_{t+1}^e and r_{t+1}^c for the environment and classifier agents using formulas (3) and (4). a_t^c is the action label obtained by the classifier using the policy to predict state s_t, and a_t is the true action label of state s_t. Lines 4 to 18 of Algorithm 1 describe the training process for

Algorithm 1: AE-SAC.

Input: train data set D_{train}, test data set D_{test}
Output: detection results on the test data set
1 Initialize experience replay Memory: $EM_e \longleftarrow 0$, $EM_c \longleftarrow 0$;
2 Initialize $\pi_c(s_t)$ arbitrarily;
3 Initialize $\pi_e(s_t)$ arbitrarily;
4 **for** $i = 1$ *to epochs* **do**
5 Initialize the state: $s_0 \longleftarrow$ random sampling from D_{train};
6 Choose initial action: $a_t^e \longleftarrow \pi_e(s_0)$;
7 $s_t \longleftarrow$ random sampling from $s(a_t^e)$, $s(a_t^e)$ are all sample whose label is a_t^e;
8 **for** $t = 0$ *to* N **do**
9 $a_t^c \longleftarrow \pi_c(s_t)$;
10 $r_{t+1}^c, r_{t+1}^e \longleftarrow$ Reward(a_t^c, a_t);
11 $a_{t+1}^e \longleftarrow \pi_e(s_t)$;
12 $s_{t+1} \longleftarrow$ random sampling from $s(a_{t+1}^e)$, $s(a_{t+1}^e)$ are all sample whose label is a_{t+1}^e;
13 EM_e.add($(s_t, a_t^e, r_{t+1}^e, s_{t+1})$);
14 EM_c.add($(s_t, a_t^c, r_{t+1}^e, s_{t+1})$);
15 **end**
16 Update environment agent by randomly sampling M experience data from EM_e;
17 Update classifier agent by randomly sampling M experience data from EM_c;
18 **end**
19 Result $\longleftarrow \pi_c(D_{test})$;
20 return Result;

both agents, including data sampling and network updates. N denotes the number of environmental agents sampling the training set, and M denotes the number of empirical data involved in the update of both agents. After the training is completed, we predict the test set D_{test} (line 19 of the Algorithm 1) using the policy π_c learned by the classifier and return the classification results.

4. Experiments and results

In this section, we will introduce the data set, data pre-processing, evaluation metrics, experimental environment, and experimental results, respectively.

4.1. Dataset

The dataset used in this paper must satisfy the following aspects: (1) A public and well known dataset that is used by many researchers for experiments. (2) A highly unbalanced dataset with varying degrees of imbalance closer to the real network environment. (3) A dataset contains sufficient samples for training and testing.

Therefore, the NSL-KDD (Tavallaee et al., 2009) dataset and AWID (Kolias et al., 2015) dataset are selected as the primary dataset for the experiment. The CICIDS2017 and CICDDoS2019 datasets were obtained by parsing Packet Capture (PCAP) files with the CICFlowMeter[9] tool. However, there are many bugs in the CICFlowMeter tool (Engelen et al., 2021; Liu et al., 2022; Lanvin et al., 2023), which makes us doubt the authenticity and reliability of these two datasets. There are still many researchers using the CICIDS2017 dataset, and this paper also performs a simple algorithm validation on the CICIDS2017 dataset.

4.1.1. NSL-KDD

NSL-KDD is an improved version of KDDCup 99 that removes redundant records from the original data set and produces more realistic simulation results. The NSL-KDD dataset includes 41 features and a label. Furthermore, the NSL-KDD contains a training set (KDDTrain+) and two test sets (KDDTest+ and KDDTest21-). These 41 features included 38 continuous and 3 categorical variables, all of which were transformed by a max-min normalization method to narrow the range between $[0, 1]$ for continuous variables and one-hot coding for categor-

[9] https://gitlab.com/hieulw/cicflowmeter.

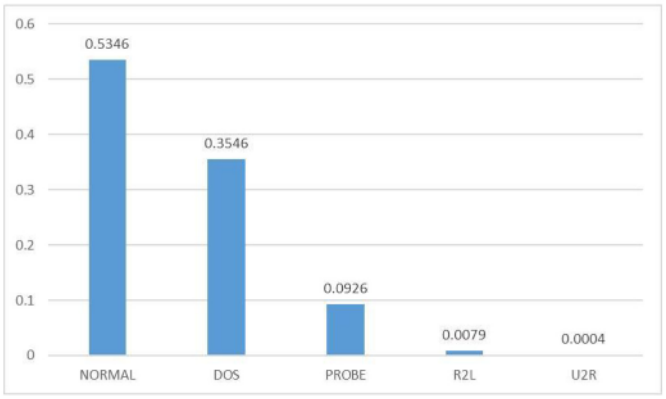

Fig. 6. Distribution frequency of each class of NSL-KDD.

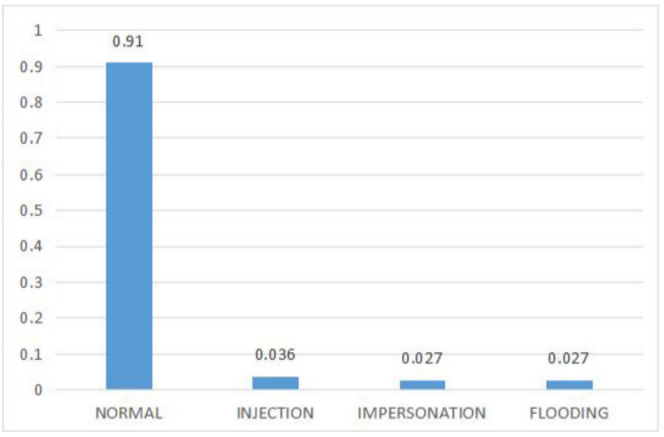

Fig. 7. Distribution frequency of each class of AWID.

Table 3
Attack Categories, including 4 types of attacks: DoS, Probe, R2L and U2R.

DoS	back, land, neptune, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm
PROBE	ipsweep, nmap, portsweep, satan, mscan, saint
R2L	ftp write, guess passwd, imap, multihop, phf, spy, warezclient, warezmaster, sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, worm
U2R	buffer overflow, loadmodule, perl, rootkit, httptunnel, ps, sqlattack, xterm

Table 4
Data distribution of 4 types of network attacks and normal data after resampling using environmental agent and original data distribution (NSL-KDD).

	Normal	DoS	Probe	R2L	U2R
Resampling	4689	2789	2187	5795	3971
Original	67343	45927	11656	995	52

ical variables. After data transformation, the dataset consisted of 122 features.

Furthermore, the attack type distribution in the dataset is severely unbalanced. The training set contains one normal type and 22 attack types, while the test set contains one normal type and 37 attack types. The training and test sets share 21 common types, with 2 unique to the training set and 17 unique to the test set. The presence of unknown types of attacks in the test set makes learning the DRL model difficult. According to the attack type, the NSL-KDD dataset can be divided into five categories: NORMAL, DoS, PROBE, R2L and U2R. The proportion of each traffic shows in Fig. 6. Without a doubt, the number of U2R and R2L attacks is extremely low. Table 3 lists the classification in detail.

The purpose of introducing environmental agent resampling in this paper is to solve the problem of unbalanced original training set. The data distribution of the four types of network attacks and normal data on the NSL-KDD dataset after using environmental proxy sampling is shown in Table 4. Without a doubt, the sampled data fraction is more balanced than the original data distribution. In particular, the two types of network attack data, R2L and U2R, have been resampled by environmental agents with data volumes of over 5000 and 3000 respectively.

4.1.2. AWID

The AWID (Kolias et al., 2015) dataset is the largest and most comprehensive dataset collected in a real-world WiFi network environment. The dataset can be divided into two data subsets based on attack class level: the ATK dataset with 16 sub-attack classes and the CLS dataset with four large attack classes. The AWID-CLS-R dataset, which contains 154 features divided into continuous and categorical features, is preferred by the majority of researchers. Furthermore, the training and test sets contain 1795,474 and 675,642 samples, respectively. In the AWID dataset, some features have null values, constant values, timestamp and network addresses. The 154-dimensional features were reduced to 46 dimensions after they were removed. In particular, the timestamp, which might affect the performance of the algorithm, we have removed it in the data preprocessing phase (Chatzoglou et al., 2022). Because the range of values varies greatly across features, continuous variables are also scaled to the interval [0–1] to eliminate the effect of this variation on the model. Furthermore, one-hot coding was used to convert categorical variables into dummy variables.

Table 5
Data distribution of 3 types of network attacks and normal data after resampling using environmental agent and original data distribution (AWID).

	Normal	Flooding	Injection	Impersonation
Resampling	46891	13946	12770	26393
Original	1633190	48484	65379	48522

The dataset includes one normal traffic category and three abnormal traffic categories. The three types of abnormal flows are injection, simulation, and flooding. Fig. 7 depicts the sample distribution frequency for each traffic class. Undoubtedly, the AWID dataset, like the NSL-KDD dataset, is extremely unbalanced and well suited for the study presented in this paper. Table 5 shows the distribution of the AWID dataset after the resampling of environmental agents, and we can observe that the resampled dataset is more balanced, which solves the category imbalance of the original training dataset to some extent. Compared with the original data distribution, the proportion of normal data is significantly reduced and the proportion of data from the other three types of cyber attacks is increased.

4.2. Data pre-processing

Network intrusion detection datasets contain a wide range of data types, including character, boolean, numeric, and even null values. Therefore, before training the intrusion detection model, we must preprocess the dataset. The pre-processing of the dataset includes data cleaning, data conversion, and data normalization.

- Data cleaning. The infinity value, which has misled the model's training, is replaced with -1, and data rows containing NaN values or NULL values are removed.
- Data conversion. The NSL-KDD dataset contains three types of characteristics: nominal, binary, and numeric. Nominal data cannot be dealt with directly by machine learning or deep learning algorithms. As a result, the dataset's non-numeric data features must be converted to numeric data (especially protocol types, services, and flag features). One-hot encoding (Potdar et al., 2017) is used to map non-numeric features to numeric features in this paper.

- Data normalization. The numerical data values vary greatly. In the NSL-KDD dataset, for example, feature attribute durations (connection durations) range from 0 to 58329. The large range of values can cause issues such as slow network convergence and neuron output saturation. Therefore, data normalization is essential. The data in this paper is restricted to $[0, 1]$ using the max-min normalization method, as shown in equation (5).

$$f_{new} = \frac{f_{old} - f_{min}}{f_{max} - f_{min}}, \tag{5}$$

where f_{old} is a network traffic feature vector and f_{min} and f_{max} are the minimum and maximum values of f_{old}. The original distribution of network traffic characteristics is preserved using this method.

The DRL input is divided into two parts: state and action. Furthermore, the dataset we use includes two additional components: network traffic features and feature labels. To incorporate network traffic elements into the DRL concept, we consider network traffic features to be states and feature labels to be actions.

4.3. Evaluation metrics

The generic ML-based approach evaluates its performance using four metrics: accuracy, precision, detection rate (DR, also known as recall), and f1-score. Accuracy is used to assess the method's overall effectiveness. Precision and DR were used to evaluate each category's efficiency. F1-score is a comprehensive overall indicator that takes into account both precision and recall. It is the weighted average of precision and recall. As a result, when evaluating models, the f1-score is more representative. The calculation formula of each evaluation indicator is as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6}$$

$$Precision = \frac{TP}{TP + FP} \tag{7}$$

$$DR = Recall = \frac{TP}{TP + FN} \tag{8}$$

$$F1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{9}$$

The number of correctly labeled samples in the target sample is referred to as the true positive (TP). The number of mislabeled samples in a non-target sample is referred to as false positive (FP). The number of samples in the target sample that were incorrectly labeled as other categories is referred to as the true Negative (TN). The number of non-samples marked in non-target samples is referred to as false negative (FN).

4.4. Experiment environment

The experiment is running on a PC with Windows 10, an Intel(R) Core(TM) I7-9750H CPU running at 2.60 GHz, and 16 GB of RAM. The experiment made use of the Scikit-Learn library, the Tensorflow2.0 library, and Keras.

To implement the environment agent and classifier agent, the SAC algorithm is used. The Actor network, Q Critic network, and V Critic network in the SAC model all use simple neural network structure in this paper. The network structure of the environment agent and classifier agent is shown in Table 6 and Table 7. The data in parentheses represent the number of neurons in the input and output layers on the AWID dataset. All the networks in the table use fully connected networks. All hidden layers in the Actor network and Critic network use ReLu as the activation function. The output layer of the Actor network uses Softmax as the activation function, while the output layer of the Critic network does not use the activation function. Adam algorithm is used to optimize the parameters of all networks. Table 8 lists the other parameters in the training process of the AE-SAC model, where γ is the discount rate, α is the temperature parameter that determines the importance of entropy.

Table 6
Network structure of the environment agent.

Name	Network Struct
Actor	122(46)-100-100-100-23(4)
Q Critic	122(46)-100-100-100-23(4)
V Critic	122(46)-100-100-100-1

Table 7
Network structure of the classifier agent.

Name	Network Struct
Actor	122(46)-100-100-100-5(4)
Q Critic	122(46)-100-100-100-5(4)
V Critic	122(46)-100-100-100-1

Table 8
Other training parameters of AE-SAC model.

Parameter	Value
epoch	100
batch_size	180
sample_size	180
learning rate	0.1
γ	0.001
α	0.2
experience replay memory	1000

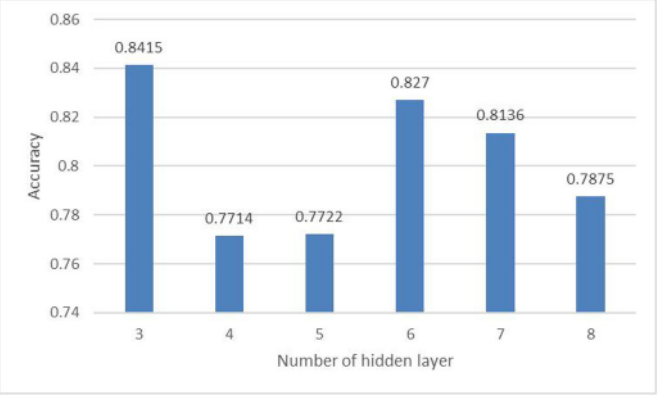

Fig. 8. Trend of accuracy with the number of hidden layers.

Grid search was used to find the best values for the SAC model's hyperparameters. It works by performing an exhaustive search for specific hyperparameter values automatically, saving time and resources. The number of layers of hidden layers and the number of neurons in each hidden layer were determined using grid search. The optimal value chosen is the one with the highest accuracy across all parameters. Figs. 8 and 9 illustrate the results of the grid search. We can see that the SAC model only needs 3 hidden layers, and the number of neurons in each hidden layer is 100, which can achieve good classification results. Of course, more hidden layers and more neurons may result in better classification results, but it will require more training time because SAC model is more complex than other reinforcement learning models. As described in section 3.1.1, the three network actors, Q Critic, and V Critic in the SAC model are only used to approximate the probability distribution function, the state action value function, and the state value function. Deep learning networks usually learn a classifier directly, which undoubtedly requires more hidden layers and more neurons.

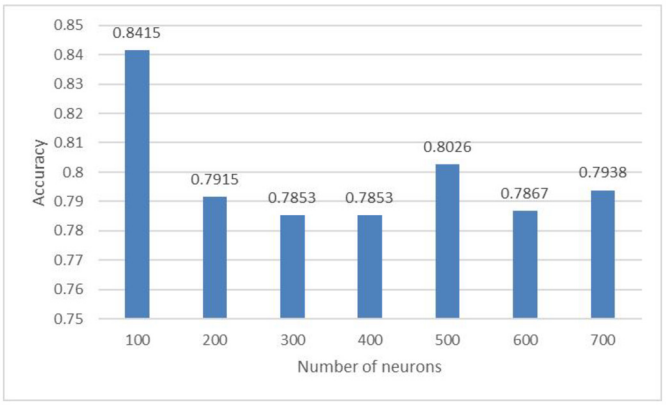

Fig. 9. Trend of accuracy with the number of neurons.

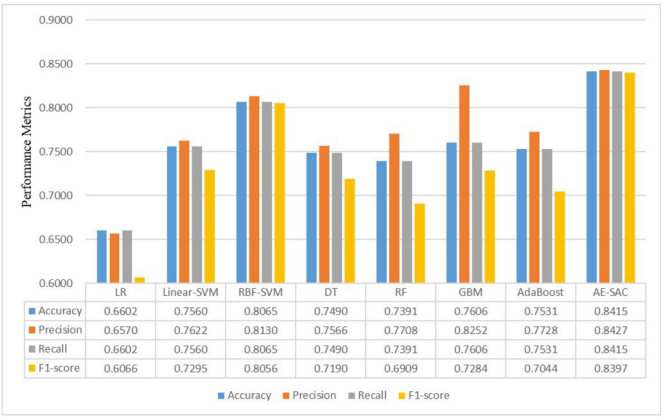

Fig. 10. Performance scores of AE-SAC compared to other ML algorithms (KDDTest+).

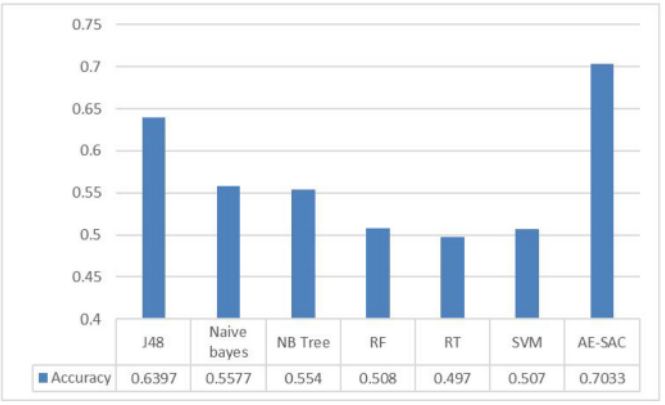

Fig. 11. Accuracy comparison of AE-SAC method and other ML methods on the KDDTest21 dataset.

4.5. Results

Early intrusion detection mostly used misuse based techniques to detect intrusions. However, misuse-based intrusion detection systems are highly dependent on the existing signature knowledge base, which makes it difficult to detect unknown attacks and cannot adapt to new intelligent attack behaviors. Therefore, anomaly-based intrusion detection is the focus of current research and development. From the perspective of model learning, anomaly-based intrusion detection also includes traditional machine learning-based intrusion detection models, deep learning-based intrusion detection models, and reinforcement learning-based intrusion detection models. Machine learning algorithms, deep learning networks belong to supervised learning and fully exploit the data labels for model learning. Although reinforcement learning is not supervised learning, it also uses the real labels of the data to obtain reward values and thus to perform policy learning (see subsection 3.2.3 for the design of the reward function). Therefore, from the data point of view, supervised learning and the reinforcement learning designed in this paper use the same data, with the difference that the AE-SAC model resamples the data set to change the imbalance of the original training set.

In order to verify the effectiveness of the method in this paper, we selected the existed network intrusion detection models (Caminero et al., 2019; Ma and Shi, 2020; Dong et al., 2021; Vinayakumar et al., 2019) as the comparison algorithm for multi-classification experiments on NSL-KDD and AWID. These methods involved in the comparison can be divided into three main categories: ML methods, DL methods, and DRL methods. In the remainder of this section, we describe in detail the performance comparison of the AE-SAC algorithm with other algorithms and the shortcomings of the AE-SAC algorithm.

4.5.1. Compared to ML methods

For the machine learning algorithms involved in the comparison, we choose Logistic Regression (LR), Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Gradient Boosting Machine (GBM) and Adaboost (AB) algorithms.

Fig. 10 compares the experimental results of this AE-SAC algorithm to machine learning algorithms in terms of accuracy, precision, recall, and f1-score in the multi-classification case. Only RBF-SVM achieves an accuracy of 80% among the machine learning algorithms in the comparison, while the other machine learning algorithms achieve a maximum accuracy of 76.06%. The accuracy of our AE-SAC algorithm has exceeded 84%, which is higher than the accuracy of all machine learning algorithms used for the comparison. Another important phenomenon is that the accuracy of RBF-SVM is 5.05% higher than that of Linear-SVM, which is actually expected. The kernel function of SVM is optional, and linear functions are often less effective than Gaussian kernel functions.

In terms of precision, recall, and f1-score, the AE-SAC algorithm still has a clear advantage. The AE-SAC algorithm has a precision and recall of more than 84%. In fact, precision and recall affect each other, and while we would like both to be high, they are mutually constraining in practice. As a result, we will use f1-score to assess both precision and recall simultaneously. There is no doubt that the f1-score of AE-SAC algorithm is still higher than other machine learning algorithms. AE-SAC is 3.41% higher than the best RBF-SVM algorithm and 23.31% higher than the worst LR algorithm.

The most likely reason for the poor classification effect of the ML algorithm on NSL-KDD is that the ML algorithm is more dependent on feature engineering. In general, as the number of features increases, the performance of machine learning classifiers tends to rise and then fall. This means that having too many or too few features can seriously reduce the classifier's effectiveness. When there aren't enough features, it's easy for data to overlap; when the feature dimension is too high, it's easy for the same category to become more distant and sparse in space, causing many classification algorithms to fail. The feature dimension of NSL-KDD is as high as 122 dimensions after processing, which will seriously affect the performance of the machine learning algorithm.

The Fig. 11 shows the accuracy of the AE-SAC algorithm compared with other machine learning algorithms on the KDDTest21 dataset. In fact, the recognition difficulty of KDDTest21 is higher than that of KDDTest+, but the accuracy of AE-SAC algorithm reaches 70.33% far higher than other algorithms compared to the commonly used machine learning algorithms.

On the AWID dataset, Fig. 12 compares the performance of the AE-SAC algorithm to that of other machine learning algorithms. In addition to the NSL-KDD dataset, we compare experimental results of algorithms such as Hyperpipes (HP), J48, Naive Bayes (NB), OneR, Random Tree (RT) and ZeroR provided in (Kolias et al., 2015). The experimental re-

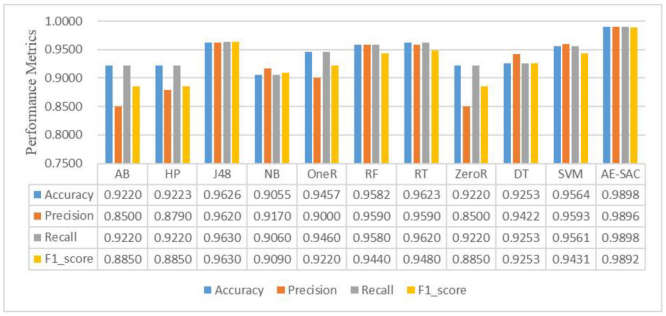

Fig. 12. Performance scores of AE-SAC compared to other ML algorithm (AWID).

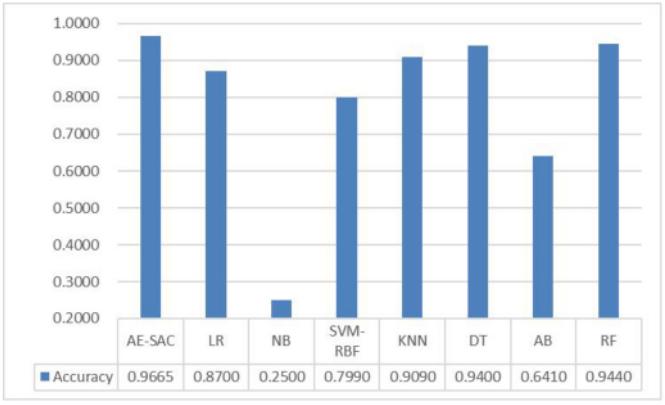

Fig. 13. Accuracy comparison of AE-SAC method and other ML algorithm on the CICIDS2017.

sults are similar to the NSL-KDD dataset, the AE-SAC algorithm still outperformed the other algorithms, with all metrics exceeding 98.9% and only J48 having accuracy and f1-score exceeding 96%. Although the processed AWID dataset only has 46 feature dimensions, feature selection is still an important factor for machine learning algorithms to affect their performance.

Fig. 13 depicts the experimental results of the accuracy of AE-SAC with other machine learning algorithms on the CICIDS2017 dataset. The accuracy of AE-SAC reached 96.65%, which is higher than other algorithms.

4.5.2. Compared to DL methods

Compared with machine learning algorithms, deep learning networks can automatically extract complex network features without relying on feature engineering. In this paper, the commonly used DL methods MLP, CNN, GRU, DNN are selected as comparison algorithms.

Fig. 14 shows the experimental results of the AE-SAC algorithm compared with the deep learning algorithm on NSL-KDD in the multi-classification case. Combining Fig. 10 and Fig. 14, we can know that except for the RBF-SVM algorithm, the classification performance of the DL algorithm is better than the performance of the ML algorithm. Among these deep learning algorithms, the accuracy of CNN, DNN, and MLP all exceeded 77%, with the highest accuracy of CNN reaching 78.75%. The performance of GRU is the worst, with an accuracy of only 75.39%, which may be mainly due to the fact that GRU is better at processing data with sequence characteristics. AE-SAC algorithm is actually also composed of several simple DNN networks, but its accuracy is better than DNN. The experimental results of the DNN series of algorithms illustrate that the performance of classification does not improve with the increase of the number of DNN layers. Among the deep learning algorithms, the MLP algorithm has the highest precision of 82.62%, and the DNN 5 layer has the highest recall and f1-score of 78.5% and 76.5%, respectively. There is no doubt that the AE-SAC algorithm has

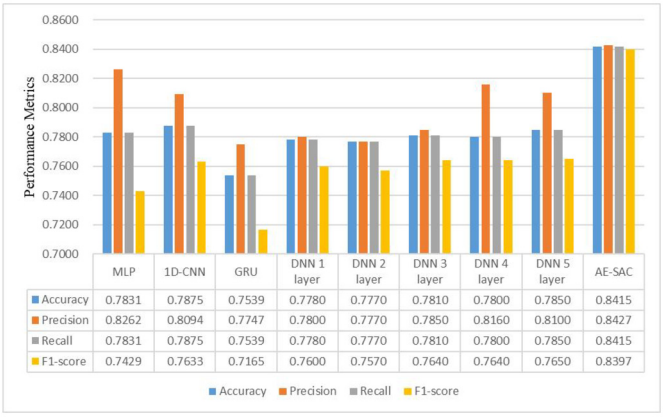

Fig. 14. Performance scores of AE-SAC compared to other DL algorithm (KDDTest+).

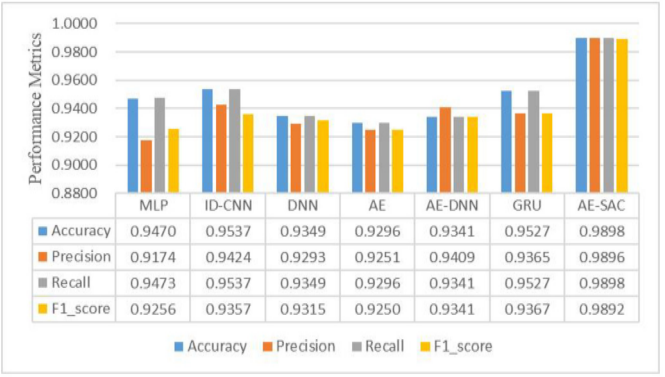

Fig. 15. Performance scores of AE-SAC compared to other DL algorithm (AWID).

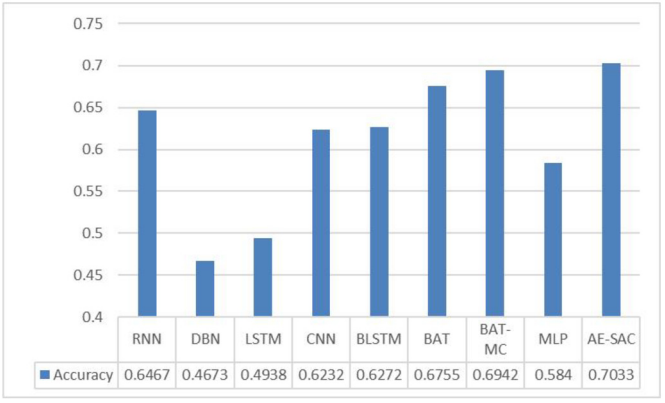

Fig. 16. Accuracy comparison of AE-SAC method and other DL methods on the KDDTest21 dataset.

a significant advantage over the deep learning algorithm in terms of precision, recall and f1-score.

The Fig. 16 shows the accuracy of the AE-SAC algorithm compared with other deep learning models on the KDDTest21 dataset, where the algorithms BAT (Bidirectional Long Short-term memory and ATtention mechanism) and BAT-MC (BAT and Multiple Convolutional layers) are proposed by Su et al. (2020). The BAT-MC algorithm processes the input data using multiple convolutional layers, and the attention mechanism is used to filter the network flow vectors composed of the grouping vectors generated by the BLSTM model to obtain the key features for network traffic classification. It achieves 69.42% accuracy, second only

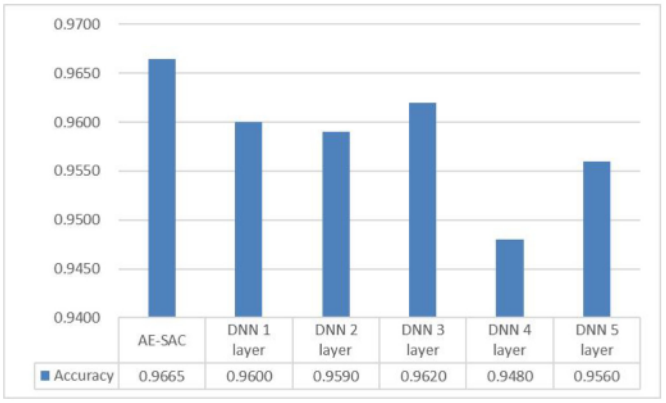

Fig. 17. Accuracy comparison of AE-SAC method and other DL networks on the CICIDS2017 dataset.

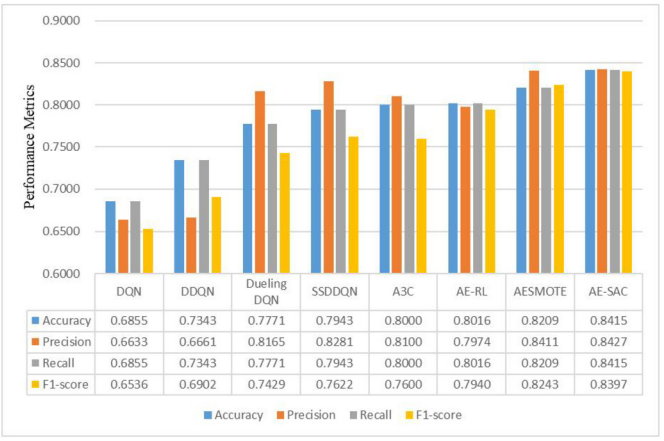

Fig. 18. Performance scores of AE-SAC compared to other DRL algorithm (KDDTest+).

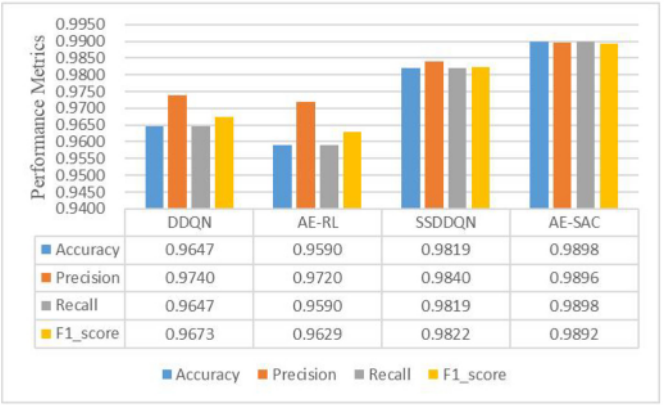

Fig. 19. Performance scores of AE-SAC compared to other DRL algorithm (AWID).

to the AE-SAC algorithm designed in this paper. Other deep learning models do not perform well for classification on KDDTest21.

Consistent with the experimental results on the NSL-KDD dataset, the AE-SAC algorithm achieves a very clear advantage on the AWID dataset as shown in Fig. 15. the accuracy of AE-SAC is higher than the 95.37% of the CNN algorithm and the 95.27% of the GRU algorithm, while none of the other algorithms achieves 95% accuracy. In terms of f1-score, the deep learning algorithms involved in the comparison have not yet reached 94%, while the AE-SAC algorithm has an f1-score close to 99%.

Fig. 17 shows the accuracy of AE-SAC compared with different hidden layer DNN networks on the CICIDS2017 dataset. The accuracy of AE-SAC algorithm is slightly higher than that of DNN algorithm. At the same time, we can also see that the classification performance of DNN will not improve with the increase of layers. A DNN network with one hidden layer is more accurate than a DNN network with 4 hidden layers. However, the accuracy of DNN with 3 hidden layers is higher than that of DNN with 1 hidden layer.

The main reason for the significant advantage of AE-SAC algorithm over the deep learning algorithm on NSL-KDD and AWID may be that AE-SAC changes the imbalance of the original dataset by introducing environmental agents for training data resampling.

4.5.3. Compared to DRL methods

DRL algorithms have also achieved good research results in intrusion detection. In this paper, DQN, DDQN, Dueling DQN, SSDDQN, A3C, AE-RL, AESMOTE algorithms are selected as comparison algorithms.

The experimental results are shown in Fig. 18. The DQN algorithm has the worst classification performance, and the AESMOTE algorithm obtains better classification results, which is only worse than the performance of the AE-SAC algorithm proposed in this paper. The accuracy of DQN is only 68.55%, which is worse than the accuracy of machine learning and deep learning algorithms. The main reason may be that the DQN algorithm suffers from a serious overestimation problem. DDQN introduces the target network, Dueling DQN changes the network structure of the original DDQN. The SSDDQN (Dong et al., 2021) algorithm introduces the K-means algorithm to improve the model's ability to detect unknown attacks. The K-mean algorithm is used to predict the action label of the next state s_{t+1}. The obtained action label and the next state s_{t+1} are input to the target network to obtain the Q value. Therefore, all three algorithms have higher accuracy than the DQN algorithm.

Precision, recall, and f1-score are similar to accuracy, and the AE-SAC algorithm is in the overall leading position. The f1-score of AE-SAC algorithm is 18.61% higher than DQN and 1.54% higher than AESMOTE.

Fig. 19 depicts the experimental results of AE-SAC on the AWID dataset. There is no doubt that the overall performance of the AE-SAC algorithm is still higher than that of the DDQN, AE-RL, and SSDDQN algorithms. It is worth noting that the performance of SSDDQN is very close to that of the AE SAC algorithm, with only 0.79% less accuracy and 0.7% less f1-score.

In fact, the AE-RL, AESOMTE and AE-SAC algorithms have similarities. The difference between AE-RL and AE-SAC is that the environment agent and the classifier agent use different reinforcement learning algorithms; AE-RL uses the DDQN algorithm, while this paper uses the SAC algorithm which has better action space exploration capability. On the other hand, SAC uses fewer hyperparameters compared to the DDQN algorithm. AESOMTE is more complex than the other two algorithms, introducing the SMOTE (Synthetic Minority Over-Sampling Technique) technique to solve the class imbalance problem by generating additional data. Experiments show that the AE-SAC algorithm has better classification performance.

4.5.4. Time performance analysis

The AE-SAC algorithm achieves a significant advantage over the commonly used machine learning algorithms, deep learning algorithms, and deep reinforcement learning algorithms. In fact, we may be more interested in the time performance of the AE-SAC algorithm when it is actually trained and deployed. Table 9 lists the training times for various algorithms, but is not sufficiently informative, with different parameter settings for each algorithm. Compared to other algorithms, the AE-SAC training time is longer, which is to be expected. The main reason is that the AE-SAC model samples more data in each training round and has to update multiple networks. In fact, the real training time of an algorithm can be affected by numerous factors, such as sample size,

Table 9
Training time of various algorithms and models on the NSL-KDD dataset.

Algorithm	LR	Linear SVM	RBF SVM	RF	GBM	AdaBoost	MLP
Train time (sec)	97.37	65.06	1696.16	97.31	2242.14	201.40	314.74
Algorithm	1D-CNN	DDQN	Dueling DQN	A3C	AE-RL	AESOMTE	AE-SAC
Train time (sec)	590.58	228.391	454.477	218.14	1090.13	2000	9099

Table 10
Number of network parameters on the AWID dataset.

Name	Classifier Parameters	Environment Parameters
Actor	25304	25304
Q Critic	25304	25304
V Critic	25001	25001

Table 11
Number of network parameters on the NSL-KDD dataset.

Name	Classifier Parameters	Environment Params
Actor	33005	34823
Q Critic	33005	34823
V Critic	32601	32601

Table 12
FLOPs on the AWID dataset.

Name	Classifier FLOPs	Environment FLOPs
Actor	50324	50324
Q Critic	50304	50304
V Critic	49701	49701

Table 13
FLOPs on the NSL-KDD dataset.

Name	Classifier FLOPs	Environment FLOPs
Actor	65730	69438
Q Critic	65705	69323
V Critic	64901	64901

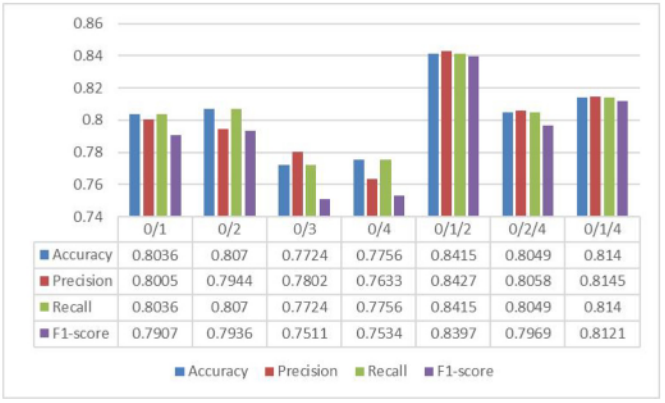

Fig. 20. Performance scores of AE-SAC algorithm with different reward functions.

GPU, etc., which is difficult to evaluate accurately. Therefore, we evaluate the performance of AE-SAC by the number of network parameter updates and floating point computations (FLOPs) during training.

The number of parameters included in each network for the AE-SAC algorithm on the AWID and NSL-KDD datasets is shown in Tables 10 and 11, respectively. We can see that the number of parameters that must be trained for AE-SAC is small. On the NSL-KDD dataset, the number of parameters for the Actor and Q Critic networks is 33005 and the number of parameters for the V Critic network is 32601. FLOPs are the number of multiplicative and additive operations in the model and are used to measure the computational complexity of the model. Tables 12 and 13 show how many floating-point operations are performed in the AE-SAC algorithm when each network is updated once. Both Actor and Critic networks only require tens of thousands of floating point operations to be updated once. This means that no high-performance machine is needed to complete the training. Although the SAC model's overall structure is complex, it does not necessitate a complex neural network. We achieved excellent classification performance in this paper by using only a fully connected network with three hidden layers.

Tables 14 and 15 show the total number of parameters and FLOPs for the other models on the NSL-KDD. Although the AE-SAC model has a complex structure, the fact is that in a real deployment we only need to deploy the Actor network. Its number of parameters and FLOPs are very small compared to other models.

For predictive classification of network traffic, we only need to use the Actor network, a fully connected network with three hidden layers, which has a very simple network structure. According to our experimental results, the prediction time on the NSL-KDD is only 0.57 s with a sample size of 22544, implying that the prediction of a single sample takes only about 25 microseconds (Table 16). Similarly, the prediction time on the AWID dataset is 6.91 s, and the sample size is 675642, which means that the prediction of a single sample takes only 10 microseconds. This prediction time can be deployed in a real environment according to the conclusions obtained in Lopez-Martin et al. (2020); Caminero et al. (2019); Dong et al. (2021); Paleyes et al. (2022). In a real environment, we can consider distributed deployment of detection models to improve detection efficiency even if the network flow generated at a certain moment is large.

4.5.5. Impact of reward function on the performance of AE-SAC

For reinforcement learning, the design of the reward function is crucial. However, it is difficult to have a direct way to obtain the optimal reward function. In this paper, seven reward functions are designed, and then a relatively optimal reward function is obtained through a grid search.

The reward functions involved in the comparison can be divided into two categories: 0/N and 0/M/N. The former indicates that if the classifier agent correctly identifies the classification to which the current network traffic belongs, it receives a reward value of N; otherwise, it is 0. The latter indicates that the classifier identifies successfully and gives different reward values according to the different classes of network traffic. The reward value is M when the network traffic is correctly classified as a majority network attack, and N when it is correctly classified as a minority network attack.

Fig. 20 depicts the performance scores of the AE-SAC algorithm on the KDDTest+ test set with different reward functions. As shown in the Fig. 20, the best classification performance of AE-SAC is obtained when the reward function is set to 0/1/2. This result is also expected to be predictable compared to other reward functions. When the reward function is set to 0/N, it means that the reward value is the same for all network traffic identified successfully, however, this setting is unfair for

Table 14
Number of parameters and FLOPs for other DL models on the NSL-KDD dataset.

Model	MLP	1D-CNN	GRU	DNN 1 layer	DNN 2 layer	DNN 3 layer	DNN 4 layer	DNN 5 layer
Parameters	12805	90373	28837	52229	841221	1235717	1366789	1399557
FLOPs	25530	6886280	19678	103454	1680670	2469150	2731038	2796446

Table 15
Number of parameters and FLOPs for other DRL models on the NSL-KDD dataset.

Model	AE-RL		AESOMTE		SSDDQN
	Environment Agent	Classifier Agent	Environment Agent	Classifier Agent	
Parameters	14623	33005	34823	53205	61307
FLOPs	29238	65730	69438	105930	121952

Table 16
Prediction time of AE-SAC on different data sets.

Dataset	Number of Samples	Predict Time
NSL-KDD	22544	0.57 s
AWID	675642	6.91 s

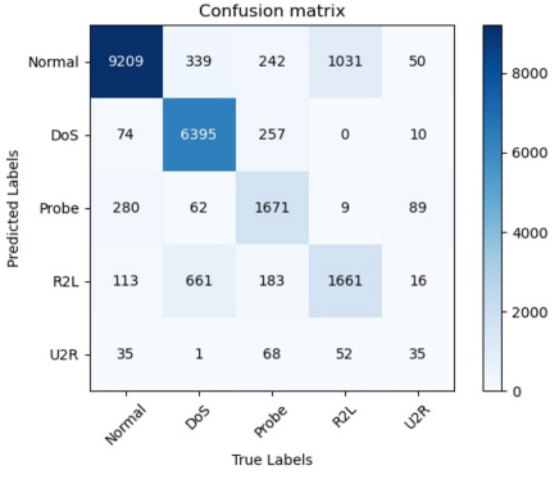

Fig. 21. Confusion matrix of AE-SAC model (KDDTest+).

a minority network attack, and the classifier learning strategy is more biased to identify majority network attacks. When the reward function is set to 0/M/N, if N is set to a value greater than M, the classifier agent learns a strategy that is more biased towards identifying a few classes of network attacks. The reward functions 0/1/2, 0/1/4 and 0/2/4 outperform 0/N. However, the value of N should not be too large, and if it is set too large, the learning strategy is completely biased towards the minority class of network attacks and unfair to other classes of network attacks. How the optimal M and N can be set is still a difficult problem left for future research.

4.5.6. Discussion of AE-SAC limitations

Although the AE-SAC algorithm achieves excellent classification results compared to other algorithms, it still has shortcomings. The confusion matrix of NSL-KDD in the AE-SAC algorithm is presented in Fig. 21. As shown in Fig. 21, the precision and recall of U2R are very low, only 18.32% and 17.5%. The main reason is that the sample size of the U2R category in the training set of NSL-KDD is only 52, and both DL and DRL require a large number of samples for training. Maybe we can introduce SMOTE technique to generate samples of U2R category like AESOMTE to solve the problem of insufficient samples.

Fig. 22 validates our conclusion that although the AWID dataset also has a class hate imbalance, flooding and injection attacks still have

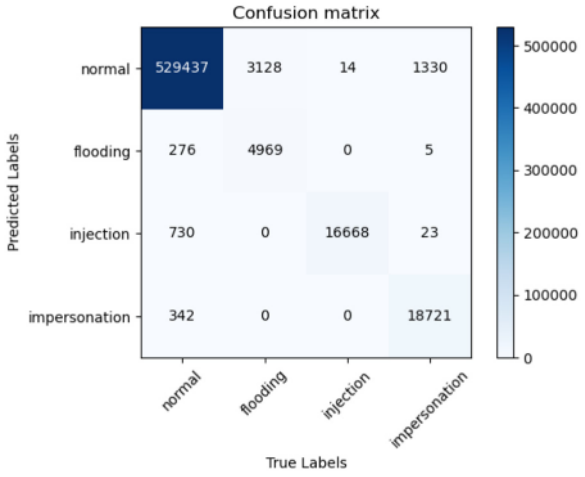

Fig. 22. Confusion matrix of AE-SAC model (AWID).

484,778 sample, which is good enough for deep learning and deep reinforcement learning. The precision of flooding is 95.47%, the recall is 61.37%, and the f1-score is 74.46%. There is no doubt that this result still seems to have much room for improvement. In fact, the AE-SAC algorithm can only solve the unbalanced situation of the data set to some extent.

Another significant issue is the model's complex structure, which results in an excessively long training time. The environment agent and the classifier agent must update at least three networks during each training round.

5. Conclusion

In this paper, the original reinforcement learning framework is modified by introducing an environmental agent to propose the AE-SAC model for solving the intrusion detection problem. In the SAC algorithm, the role of the environment agent is to resample the training data to solve the unbalanced problem of the original training set. The classifier agent uses the sampled data for training. The two agents are trained against each other in order to maximize their respective rewards. Finally, we performed a multiclassification evaluation on the NSL-KDD and AWID test set using a classifier learning strategy. On the NSL-KDD dataset, AE-SAC achieved an accuracy of 84.15% and an f1-score of 83.97%, while on the AWID dataset, all performance metrics of AE-SAC exceeded 98.9%.

However, the sample size of the U2R category in the original training set is too small, resulting in a relatively low precision and recall of AE-SAC on the U2R category. In the future, we will introduce SMOTE technique or generative adversarial network to generate the number of samples of U2R category to solve the above problem.

Another critical issue is the vulnerability of neural network models to adversarial attacks (Merzouk et al., 2022), which is not currently addressed in this paper. In the future, we will consider using adversarial training to improve the robustness of the model.

CRediT authorship contribution statement

Zhengfa Li: Conceptualization, Formal analysis, Methodology, Validation, Writing – original draft, Writing – review & editing. **Chuanhe Huang:** Funding acquisition, Project administration, Supervision, Writing – review & editing. **Shuhua Deng:** Conceptualization, Formal analysis, Methodology. **Wanyu Qiu:** Formal analysis, Resources. **Xieping Gao:** Conceptualization, Formal analysis, Methodology.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work is supported by the National Natural Science Foundation of China (No. 61772385).

References

Alavizadeh, Hooman, Alavizadeh, Hootan, Jang-Jaccard, Julian, 2022. Deep q-learning based reinforcement learning approach for network intrusion detection. Computers 11 (3), 41.

Caminero, Guillermo, Lopez-Martin, Manuel, Carro, Belen, 2019. Adversarial environment reinforcement learning algorithm for intrusion detection. Comput. Netw. 159, 96–109.

Chatzoglou, Efstratios, Kambourakis, Georgios, Kolias, Constantinos, Smiliotopoulos, Christos, 2022. Pick quality over quantity: expert feature selection and data preprocessing for 802.11 intrusion detection systems. IEEE Access 10, 64761–64784.

Christodoulou, Petros, 2019. Soft actor-critic for discrete action settings. arXiv preprint. arXiv:1910.07207.

Cil, Abdullah Emir, Yildiz, Kazim, Buldu, Ali, 2021. Detection of ddos attacks with feed forward based deep neural network model. Expert Syst. Appl. 169, 114520.

Demis, Hassabis, 2016. AlphaGo: using machine learning to master the ancient game of Go. Google Blog 27.

Dong, Bo, Wang, Xue, 2016. 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, pp. 581–585.

Dong, Shi, Xia, Yuanjun, Peng, Tao, 2021. Network abnormal traffic detection model based on semi-supervised deep reinforcement learning. IEEE Trans. Netw. Serv. Manag. 18 (4), 4197–4212.

Engelen, Gints, Rimmer, Vera, Joosen, Wouter, 2021. Troubleshooting an intrusion detection dataset: the CICIDS2017 case study. In: 2021 IEEE Security and Privacy Workshops. SPW. IEEE, pp. 7–12.

Gamage, Sunanda, Samarabandu, Jagath, 2020. Deep learning methods in network intrusion detection: a survey and an objective comparison. J. Netw. Comput. Appl. 169, 102767.

Haarnoja, Tuomas, Zhou, Aurick, Abbeel, Pieter, Levine, Sergey, 2018a. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International Conference on Machine Learning. PMLR, pp. 1861–1870.

Haarnoja, Tuomas, Zhou, Aurick, Hartikainen, Kristian, Tucker, George, Ha, Sehoon, Tan, Jie, Kumar, Vikash, Zhu, Henry, Gupta, Abhishek, Abbeel, Pieter, et al., 2018b. Soft actor-critic algorithms and applications. arXiv preprint. arXiv:1812.05905.

Han, Xiaolu, Liu, Yun, Zhang, Zhenjiang, Lü, Xin, Li, Yang, 2021. Sparse auto-encoder combined with kernel for network attack detection. Comput. Commun. 173, 14–20.

Hassan, Mohammad Mehedi, Gumaei, Abdu, Alsanad, Ahmed, Alrubaian, Majed, Fortino, Giancarlo, 2020. A hybrid deep learning model for efficient intrusion detection in big data environment. Inf. Sci. 513, 386–396.

Hou, Tianhao, Xing, Hongyan, Liang, Xinyi, Su, Xin, Wang, Zenghui, 2022. Network intrusion detection based on DNA spatial information. Comput. Netw. 217, 109318.

Imran, Muhammad, Haider, Noman, Shoaib, Muhammad, Razzak, Imran, et al., 2022. An intelligent and efficient network intrusion detection system using deep learning. Comput. Electr. Eng. 99, 107764.

Jazi, Hossein Hadian, Gonzalez, Hugo, Stakhanova, Natalia, Ghorbani, Ali A., 2017. Detecting HTTP-based application layer DoS attacks on web servers in the presence of sampling. Comput. Netw. 121, 25–36.

Kolias, Constantinos, Kambourakis, Georgios, Stavrou, Angelos, Gritzalis, Stefanos, 2015. Intrusion detection in 802.11 networks: empirical evaluation of threats and a public dataset. IEEE Commun. Surv. Tutor. 18 (1), 184–208.

Lan, Jinghong, Liu, Xudong, Li, Bo, Sun, Jie, Li, Beibei, Zhao, Jun, 2022. Member: a multi-task learning model with hybrid deep features for network intrusion detection. Comput. Secur. 123, 102919.

Lanvin, Maxime, Gimenez, Pierre-François, Han, Yufei, Majorczyk, Frédéric, Mé, Ludovic, Totel, Eric, 2023. Errors in the CICIDS2017 dataset and the significant differences in detection performances it makes. In: Risks and Security of Internet and Systems.

Liu, Lisa, Engelen, Gints, Lynar, Timothy, Essam, Daryl, Joosen, Wouter, 2022. Error prevalence in NIDS datasets: a case study on CIC-IDS-2017 and CSE-CIC-IDS-2018. In: 2022 IEEE Conference on Communications and Network Security. CNS. IEEE, pp. 254–262.

Lopez-Martin, Manuel, Carro, Belen, Sanchez-Esguevillas, Antonio, 2020. Application of deep reinforcement learning to intrusion detection for supervised problems. Expert Syst. Appl. 141, 112963.

Ma, Xiangyu, Shi, Wei, 2020. AESMOTE: adversarial reinforcement learning with SMOTE for anomaly detection. IEEE Trans. Netw. Sci. Eng. 8 (2), 943–956.

Merzouk, Mohamed Amine, Delas, Joséphine, Neal, Christopher, Cuppens, Frédéric, Boulahia-Cuppens, Nora, Yaich, Reda, 2022. Evading deep reinforcement learning-based network intrusion detection with adversarial attacks. In: Proceedings of the 17th International Conference on Availability, Reliability and Security, pp. 1–6.

Mishra, Preeti, Varadharajan, Vijay, Tupakula, Uday, Pilli, Emmanuel S., 2018. A detailed investigation and analysis of using machine learning techniques for intrusion detection. IEEE Commun. Surv. Tutor. 21 (1), 686–728.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, Riedmiller, Martin, 2013. Playing Atari with deep reinforcement learning. arXiv preprint. arXiv:1312.5602.

Moustafa, Nour, Slay, Jill, 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In: 2015 Military Communications and Information Systems Conference. MilCIS. IEEE, pp. 1–6.

Muna, AL-Hawawreh, Moustafa, Nour, Sitnikova, Elena, 2018. Identification of malicious activities in industrial internet of things based on deep learning models. J. Inf. Secur. Appl. 41, 1–11.

Paleyes, Andrei, Urma, Raoul-Gabriel, Lawrence, Neil D., 2022. Challenges in deploying machine learning: a survey of case studies. ACM Comput. Surv. 55 (6), 1–29.

Pingale, Subhash V., Sutar, Sanjay R., 2022. Remora whale optimization-based hybrid deep learning for network intrusion detection using CNN features. Expert Syst. Appl. 210, 118476.

Popoola, Segun I., Adebisi, Bamidele, Hammoudeh, Mohammad, Gui, Guan, Gacanin, Haris, 2020. Hybrid deep learning for botnet attack detection in the internet-of-things networks. IEEE Int. Things J. 8 (6), 4944–4956.

Potdar, Kedar, Pardawala, Taher S., Pai, Chinmay D., 2017. A comparative study of categorical variable encoding techniques for neural network classifiers. Int. J. Comput. Appl. 175 (4), 7–9.

Schaul, Tom, Quan, John, Antonoglou, Ioannis, Silver, David, 2015. Prioritized experience replay. arXiv preprint. arXiv:1511.05952.

Sethi, Kamalakanta, Kumar, Rahul, Prajapati, Nishant, Bera, Padmalochan, 2020a. Deep reinforcement learning based intrusion detection system for cloud infrastructure. In: 2020 International Conference on COMmunication Systems & NETworkS. COMSNETS. IEEE, pp. 1–6.

Sethi, Kamalakanta, Rupesh, E. Sai, Kumar, Rahul, Bera, Padmalochan, Madhav, Y. Venu, 2020b. A context-aware robust intrusion detection system: a reinforcement learning-based approach. Int. J. Inf. Secur. 19 (6), 657–678.

Sethi, Kamalakanta, Madhav, Y. Venu, Kumar, Rahul, Bera, Padmalochan, 2021. Attention based multi-agent intrusion detection systems using reinforcement learning. J. Inf. Secur. Appl. 61, 102923.

Sharafaldin, Iman, Lashkari, Arash Habibi, Hakak, Saqib, Ghorbani, Ali A., 2019. Developing realistic distributed denial of service (DDoS) attack dataset and taxonomy. In: 2019 International Carnahan Conference on Security Technology. ICCST. IEEE, pp. 1–8.

Shone, Nathan, Ngoc, Tran Nguyen, Phai, Vu Dinh, Shi, Qi, 2018. A deep learning approach to network intrusion detection. IEEE Trans. Emerg. Top. Comput. Intell. 2 (1), 41–50.

Su, Tongtong, Sun, Huazhi, Zhu, Jinqi, Wang, Sheng, Li, Yabo, 2020. BAT: deep learning methods on network intrusion detection using NSL-KDD dataset. IEEE Access 8, 29575–29585.

Tavallaee, Mahbod, Bagheri, Ebrahim, Lu, Wei, Ghorbani, Ali A., 2009. A detailed analysis of the KDD CUP 99 data set. In: 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications. IEEE, pp. 1–6.

Thaseen, Ikram Sumaiya, Kumar, Cherukuri Aswani, 2017. Intrusion detection model using fusion of chi-square feature selection and multi class SVM. J. King Saud Univ. Comput. Inf. Sci. 29 (4), 462–472.

Vinayakumar, Ravi, Alazab, Mamoun, Soman, K.P., Poornachandran, Prabaharan, Al-Nemrat, Ameer, Venkatraman, Sitalakshmi, 2019. Deep learning approach for intelligent intrusion detection system. IEEE Access 7, 41525–41550.

Wang, Haomin, Li, Wei, 2021. DDosTC: a transformer-based network attack detection hybrid mechanism in SDN. Sensors 21 (15), 5047.
Zhang, Guoling, Wang, Xiaodan, Li, Rui, Song, Yafei, He, Jiaxing, Lai, Jie, 2020a. Network intrusion detection based on conditional Wasserstein generative adversarial network and cost-sensitive stacked autoencoder. IEEE Access 8, 190431–190447.
Zhang, Jianwu, Ling, Yu, Fu, Xingbing, Yang, Xiongkun, Xiong, Gang, Zhang, Rui, 2020b. Model of the intrusion detection system based on the integration of spatial-temporal features. Comput. Secur. 89, 101681.
Zhang, Shangtong, Sutton, Richard S., 2017. A deeper look at experience replay. arXiv preprint. arXiv:1712.01275.
Zhou, Kun, Wang, Wenyong, Hu, Teng, Deng, Kai, 2021. Application of improved asynchronous advantage actor critic reinforcement learning model on anomaly detection. Entropy 23 (3), 274.

Zhengfa Li received the B.S. and M.S. degrees in computer science and technology from Xiangtan University, China, in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree in computer science and technology at Wuhan University, Wuhan, China. His research interests are in the area of computer networks and network security.

Chuanhe Huang received the B.Sc., M.Sc., and Ph.D. degrees in computer science from Wuhan University, Wuhan, China, in 1985, 1988, and 2002, respectively. He is currently a Professor with the School of Computer Science, Wuhan University. His research interests include computer networks, VANETs, the Internet of Things, and distributed computing.

Shuhua Deng received the B.S. degree in computer science and the Ph.D. degree in computational mathematics from Xiangtan University, Hunan, China, in 2013 and 2018, respectively. His current research interests include software-defined networks, network security, and system security.

Wanyu Qiu received her B.E. degree in the School of Information Engineering from Hubei University of Economics, China in 2017 and M.E. degree in the School of Computer Science from Wuhan University, China in 2019. She is currently a Ph.D. student in the School of Computer Science, Wuhan University, China. Her research interest is computer network, game theory.

Xieping Gao was born in 1965. He received the B.S. and M.S. degrees from Xiangtan University, China, in 1985 and 1988, respectively, and the Ph.D. degree from Hunan University, China, in 2003. He is a Professor in the Hunan Provincial Key Laboratory of Intelligent Computing and Language Information Processing, Hunan Normal University, Changsha, China. He is also with the Key Laboratory of Intelligent Computing and Information Processing of Ministry of Education, Xiangtan University, China. He was a visiting scholar at the National Key Laboratory of Intelligent Technology and Systems, Tsinghua University, China, from 1995 to 1996, and at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, from 2002 to 2003. He is a regular reviewer for several journals and he has been a member of the technical committees of several scientific conferences. He has authored and co-authored over 110 journal papers, conference papers, and book chapters. His current research interests are in the areas of wavelet analysis, neural network, bioinformatics, image processing, and computer network.