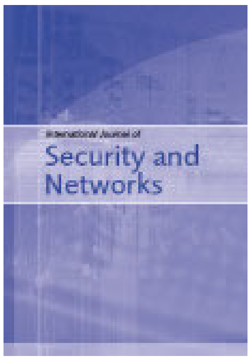


International Journal of Security and Networks

ISSN online: 1747-8413 - ISSN print: 1747-8405
https://www.inderscience.com/ijsn

OneR-DQN: a botnet traffic detection model based on deep Q network algorithm in deep reinforcement learning

Yutao Hu, Yuntao Zhao, Yongxin Feng, Xiangyu Ma

DOI: 10.1504/IJSN.2023.10062404

Article History:

Received:	20 March 2023
Last revised:	06 June 2023
Accepted:	08 June 2023
Published online:	12 March 2024

OneR-DQN: a botnet traffic detection model based on deep Q network algorithm in deep reinforcement learning

Yutao Hu, Yuntao Zhao*, Yongxin Feng and Xiangyu Ma

School of Information Science and Engineering,
Shenyang Ligong University,
Shengyang 110870, China
Email: huyutao_2023@163.com
Email: zhaoyuntao2014@163.com
Email: fengyongxin@263.net
Email: maxiangyu0729@126.com
*Corresponding author

Abstract: With the continuous progress of network technology, the rapid growth of botnets poses a significant challenge to network security. A universal detection model needs to be improved to cope with various datasets with variable feature states. This article constructs a detection model based on deep reinforcement learning (DRL) deep Q network (DQN). It uses a OneR classifier to select features from the dataset and hand them to the model for training. The unique experience pool mechanism of DQN is used to extract independent experience and training samples for cross-training continuously. The trained model compares with the other detection models in a new dataset for experimental comparison. The experiment shows that compared with the existing detection model, the improved DQN botnet detection model has higher accuracy rate and precision rate, which indicates that the detection model equipped with the DQN algorithm has more robust adaptability in the new dataset.

Keywords: botnet detection; deep reinforcement learning; DRL; DQN model; feature classification; OneR classifier.

Reference to this paper should be made as follows: Hu, Y., Zhao, Y., Feng, Y. and Ma, X. (2024) ‘OneR-DQN: a botnet traffic detection model based on deep Q network algorithm in deep reinforcement learning’, *Int. J. Security and Networks*, Vol. 19, No. 1, pp.31–42.

Biographical notes: Yutao Hu received his BS in Computer Science and Technology from the Shenyang University of Technology, China, in 2021. He is currently working toward the Master’s degree in Computer Technology with the School of Information Science and Engineering, Shenyang Ligong University, China. His research interests include network security and deep reinforcement learning (DRL).

Yuntao Zhao completed his PhD of Navigation, Guidance and Control from the Nanjing University of Science and Technology. He is currently a Professor and Doctoral Supervisor of the School of Information Science and Engineering at the Shenyang Ligong University. His main research area is in cyberspace security, machine learning and deep learning algorithms.

Yongxin Feng received her PhD in Computer Application Technology from the Northeastern University, China, in 2003. She is currently a Professor and Doctoral Supervisor of the School of Information Science and Engineering at the Shenyang Ligong University. Her research interests include mobile wireless network technology and network security.

Xiangyu Ma is currently pursuing the MS degree in Computer Technology with Shenyang Ligong University, Shenyang, China. During his pursuit of a Master’s degree, he participated in a network security innovation competition and won third prize at the national level. He research interests include network security, network engineering and deep learning.

1 Introduction

Nowadays, the methods of botnet network intrusion are constantly innovating. In 2020 alone, there were 35 million botnet network intrusion incidents in Shanghai, China (Heng’an Jiaxin Technology Co., 2020). Many hosts are unwittingly turned into ‘botnet’ hosts and become the host of botnet networks when they access phishing websites that have the same interface as legitimate sites, which are then used by the botnet network families to attack other websites, causing many sites to force to shut down due to DDoS attacks. Although experts have built many detection models for IRC botnet networks in the early days, the structure of botnet networks is also constantly evolving, gradually evolving from the early IRC mode to HTTP and peer-to-peer (P2P) modes (Dhayal et al., 2018). Hackers use DDoS, spam, brute-force cracking, penetration, and other attack methods to infect hosts in large quantities, forming large and small botnet network families. These families include massive botnet networks with hundreds of thousands of hosts and small botnet networks with only a hundred hosts (Huang, 2020). Many users may unknowingly utilise to engage in illegal activities, which not only brings adverse effects to significant operators' and merchants' websites but also affects the reputation and the user's computer. Due to the P2P mode, it is difficult to know the location of the botnet network controller's host in the trace, and it can only continuously improve its detection capability to avoid hosts infected by botnet networks, which poses a significant threat to the world's network security problem.

Due to the openness and vulnerability of the IRC protocol, detecting IRC botnets is relatively easy. The earliest detection method was based on rules such as port, protocol, and source address to filter traffic, but this method can only detect known attack traffic and not new attacks. With the development of P2P protocols, attackers began using P2P protocols to control botnets, which are more covert. The detection of P2P botnets needs to consider their distributed and decentralised characteristics. In P2P botnets, communication between botnet hosts is not through a centralised server but through a P2P network for connection and communication. Therefore, special techniques are needed to detect P2P botnet traffic. A standard method is based on statistical analysis and machine learning algorithms, using feature extraction and classification algorithms to analyse and identify traffic. This method can detect P2P botnet traffic and identify new unknown attacks.

Modern detection methods include feature analysis, behaviour analysis, machine learning, and deep learning. Botnet traffic can be detected by monitoring traffic in real-time, analysing traffic, and identifying features. This article proposes a botnet network detection method based on machine learning and deep reinforcement learning (DRL). By feature recognition and classification and training the detection model with features that have a more significant impact, it can recognise various attacks, thereby preventing botnet traffic from invading.

The main contributions of this study are as follows:

1 To address the issue of feature selection in reinforcement learning, a OneR classifier based on machine learning is first used to select individual features with significant impact. This section reduces the time and cost of manual operations. It ensures that the selected feature parameters are more conducive to model training, thus avoiding poor detection performance caused by feature selection problems.

2 Using DRL DQN algorithm to build a detection model makes the judgment of data flow more professional. In addition, the unique experience pooling mechanism of DQN can also eliminate correlations between data streams, ensuring the accuracy rate of judgments. Experimental comparisons have shown that compared to the already well-established LSTM and RNN+CNN models, the accuracy rate of the DQN model still improves by 1.4%.

3 Due to the singularity of the dataset itself, a definite conclusion that DQN is superior to other existing detection models cannot be obtained in comparative experiments. Therefore, this article used the initial dataset CIC-IDS2017, which also underwent data pre-processing, to compare the three types of models in the initial environment, which better reflects the adaptability of the models. The experimental comparison shows that compared to LSTM and RN + CNN models, The accuracy rate of the DQN detection model is 14% higher than theirs, with an accuracy rate increase of 12%, indicating a significant improvement and better adaptability to unfamiliar datasets.

2 Related work

In recent years, mainstream detection methods can divide into two categories based on traffic and behaviour. From a behavioural perspective, Lu (2020) ignores the sequential nature of data and uses graph centrality-based metrics to focus on communication graph structures. However, this method requires accessing all data at once to establish the graph model and is thus not very suitable as a handy tool. From a traffic perspective, Zeidanloo et al. (2010) constructed a general detection framework to determine whether the traffic form belongs to a botnet. Although this framework is practical for traditional IRC botnets, it cannot identify the new P2P botnets that emerged. To address this, Zhao et al. (2013) studied a real-time botnet detection method based on TCP/UDP flows, which classifies packets with the same five-tuple: source/destination IP address, source/destination port number, and protocol identifier into a data flow and uses short time windows to divide data flows to achieve real-time detection of botnets. Joshi et al. (2020) proposed using SVM and RFC random forest classifiers to conduct classification experiments on the N-BaIoT dataset. Alqatawna et al. (2021) conducted a feature importance analysis and detected the URL feature set in Android botnets. Some experts and professors used decision trees, K-NN, MLP, naive Bayes, random forests, SVM, and extreme gradient boosting based on each time window's dataset to detect mobile botnets and achieved high accuracy rate rates in specific datasets (Kirubavathi et al., 2018; Anwar et al., 2016; Abdullah et al., 2014; Girei et al., 2016;

Costa et al., 2019; Yerima et al., 2021; Fu et al., 2020; Xu, 2016).

With the rise of deep learning, Torres et al. proposed using RNN to detect a small amount of data through 5-fold cross-validation (Torres et al., 2016). Pektas et al. (2018) used deep neural networks to model network traffic to detect botnets. Hussain et al. (2021) used the residual network ResNet-18 to detect botnet network scanning activities and DDoS attacks during the zero-day attack phase. Chen used the residual spatiotemporal network to solve network degradation problems and combined the long short-term memory (LSTM) and convolutional neural network (CNN) pooling layer in deep learning to design the Res-1DCNN-LSTM model for botnet network traffic (Chen, 2022). Qi (2022), and Lu (2021), combined and improved deep learning algorithms such as GRU, LSTM, and CNN to achieve the detection of botnet network traffic. In addition, Xue and Wang (2022) combined deep learning with the internet of Things. They designed the bidirectional LSTM-RNN model to detect consumer IoT devices and botnet network activities in the network.

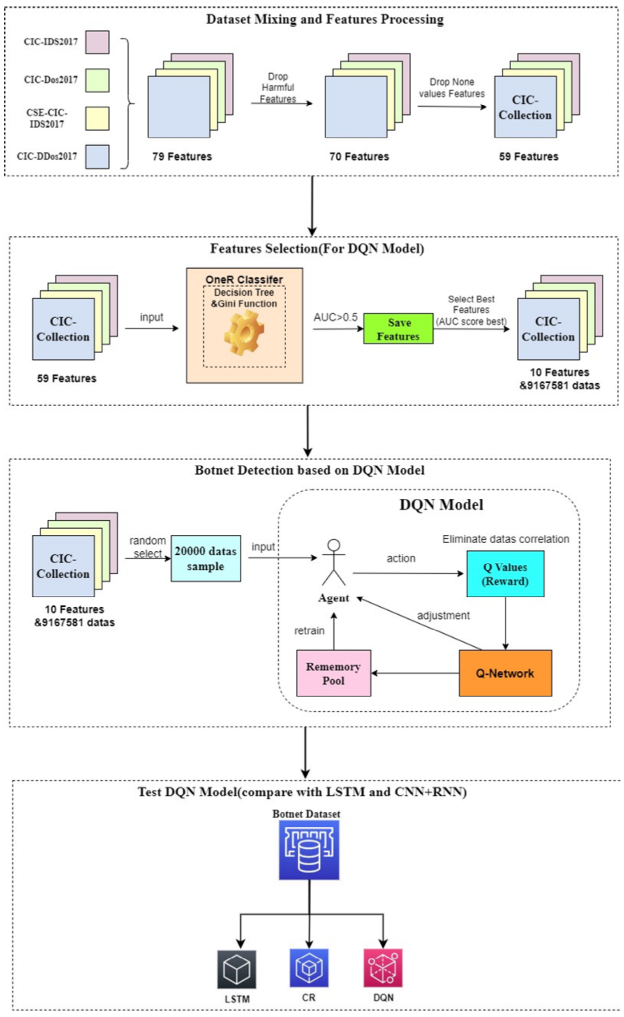
Christopher et al. started from the perspective of botnets. They used generative adversarial networks (GAN) to generate samples to make the detector more directional and improve the model's detection capability (Christopher et al., 2018). Not only this, some teams used DRL to generate adversarial samples that evade machine learning detection algorithms, thereby generating a large volume of botnet family networks and completing their invasion as botnets. They continuously improve the detection model's robustness (Wu et al., 2019; Apruzzese et al., 2020; Randhawa et al., 2022; Kadir et al., 2015). Venturi et al. (2020) created an adversarial dataset based on GAN to avoid traditional machine learning detectors.

With the development of DRL theory, combined with the robust learning mechanism of deep learning in high-dimensional and multi-feature scenarios (Huang, 2008; Volodymyr et al., 2015), it has begun to show outstanding performance in various fields (Zhao and Ding, 2018). This paper proposes a botnet network detection and classification method based on the deep Q-network (DQN). This method belongs to static analysis, builds a DQN model, inputs various features of botnet networks, and trains it using a botnet network traffic dataset. The model combines the trial-and-error mechanism and action optimisation strategy of the Q-learning algorithm in reinforcement learning with the high-dimensional feature input of deep learning to achieve detection and classification of seven types of botnet network traffic, including DDOS and Botnet, as well as regular traffic.

3 Methodology

This paper proposes a OneR-DQN model constructed using the machine learning OneR classifier and the DRL DQN algorithm. The aim is to maintain high predictive capabilities for different botnet datasets. The entire process is divided into four steps: merging and pre-processing the datasets, conducting feature engineering with machine learning to select relevant features, building the DQN model, comparison experiments with the built DQN model and the others botnet detection model to compare the performance of the two algorithms under different datasets, and using the existing deep learning LSTM detection model as an objective model judging reference. This paper also explores the impact of feature engineering without using the OneR classifier during model training. The experimental procedure and model framework are illustrated in Figure 1. And next, we will introduce each section one by one in the order shown in Figure 1.

Figure 1 Flow diagram of botnet traffic detection based on DDQN algorithm (see online version for colours)



3.1 *Data selection and processing*

The datasets used in this study are from the CIC-IDS series provided by the Canadian Institute for Cybersecurity. The datasets include CIC-IDS2017, CIC-Dos2017, CSE-CIC-IDS2018, and CIC-DDos2019. The NumPy library was employed in this paper to merge labels with the same label values and filter out certain data samples. The resulting merged dataset is referred to as the CIC-Collection, and the corresponding label values obtained are presented in Table 1.

Each label is associated with the features from the four datasets. By using the Pandas library for data reading, there are a total of 9,167,581 records with 79-dimensional features. However, not all features are relevant or informative. The dataset contains certain features completely irrelevant for prediction, referred to as pollution features, and need to be discarded. Following the ranking of feature pollution severity, this paper utilised the drop function to remove nine features, as shown in Table 2.

Table 1 'Label' labels in the CIC-collection dataset

Benign	DDos	Bruteforce	Infiltration
Botnet	Dos	Webattack	Portscan

Table 2 The deleted 9-dimensional feature severely contaminates the dataset

PSH flag count	*ECE flag count*	*RST flag count*
ACK flag count	Fwd packet length Min	Bwd packet length min
Packet length min	Protocol	Down/up ratio

There are also some features, such as active Std. and Idle Std., that can affect the dataset. However, after data batching on the smaller dataset, the degree of feature pollution is not significant or noticeable, so these features are retained.

During the process of identifying polluted features, this study discovered 11 features with a predictive capacity of zero. These features have no meaningful value in any classification model, and some even have None as their feature values. For these features, the drop function was also used to discard them, as shown in Table 3.

Table 3 Deleted 11 dimensional meaningless features

Bwd. avg. bulk rate	*Bwd. avg. bytes/Bulk*	*Bwd avg. packets/Bulk*
Bwd PSH flags	Bwd URG Flags	CWE flag count
FIN flag count	Fwd. avg. bulk rate	Fwd. avg. bytes/bulk
Fwd. avg. packets/bulk	Fwd. URG flags	

Table 4 Data sample of CIC collection botnet dataset

Botnet traffic class	*Number of samples*
Benign	7186189
DDos	1234729
Dos	397,344
Botnet	145,968
Bruteforce	103,244
Infiltration	94,857
Webattack	2,995
Portscan	2,255

Through the three steps described above, this study obtained a clean sample dataset (CIC-collection) of four datasets. The dataset contains detailed labels for personal attacks and corresponding broader attack categories. There are no none values, and no duplicate samples exist. The dataset comprises 9,167,581 records, with each record having 59-dimensional features. The dataset is stored in the Parquet file format. Compared to CSV files, Parquet files offer smaller compressed storage space and more efficient reading (Braams, 2018). The data can be read using the built-in Pyarrow library in Python. An example of the data samples in the dataset is presented in Table 4.

3.2 *OneR classifier*

OneR was first introduced by Holte in 1993, and it has since been used in a variety of domains, including medical diagnosis, credit scoring, and text classification.In recent years, more advanced machine learning algorithms, such as neural networks and decision trees, have gained popularity for classification tasks. However, OneR remains a useful tool for exploratory data analysis and as a baseline model for comparison with more complex methods.

The OneR classifier is a simple and effective algorithm that builds a rule-based model for classification tasks. It works by selecting a single feature from the dataset and then determining the class that most frequently occurs for each unique value of that feature. This process generates a set of rules, which can be used to predict the class of new instances based on their feature values.

The OneR algorithm is computationally efficient, making it well-suited for large datasets with many features. Additionally, it can provide insights into the most important features for a particular classification problem, aiding in feature selection and interpretation.

OneR classifier algorithm steps are as follows:

Algorithm 1 OneR classifier algorithm steps

```
1  Convert the feature values of the dataset into binary values
   (assign 1 to feature values that are greater than or equal to
   the mean, and 0 to those that are less than the mean)
2  Calculate the frequency of each feature value in each
   category and determine the misclassification rate of the
   most frequent feature value
3  Add up the misclassification rates of all feature values for
   a particular feature, and identify the combination of
   features with the lowest misclassification rate and the total
   number of errors
4  Compare the total number of errors for each feature and
   select the feature with the lowest number of errors as the
   best feature for classification
5  The classification rule for the best feature corresponds to
   the feature value classification
```

Before performing the classification operation with the OneR classifier, it is necessary to discard certain features that cannot be used for classification, such as features of 'type' type. After this pre-processing step, the dataset is reduced to 44 dimensions. Then, the remaining features are input into the OneR classifier, which utilises a machine-learning decision tree classifier as its core. The steps involved in this process are as follows:

$$DecisionTreeClassifier(\max\_depth = 1, critierion = 'gini')$$

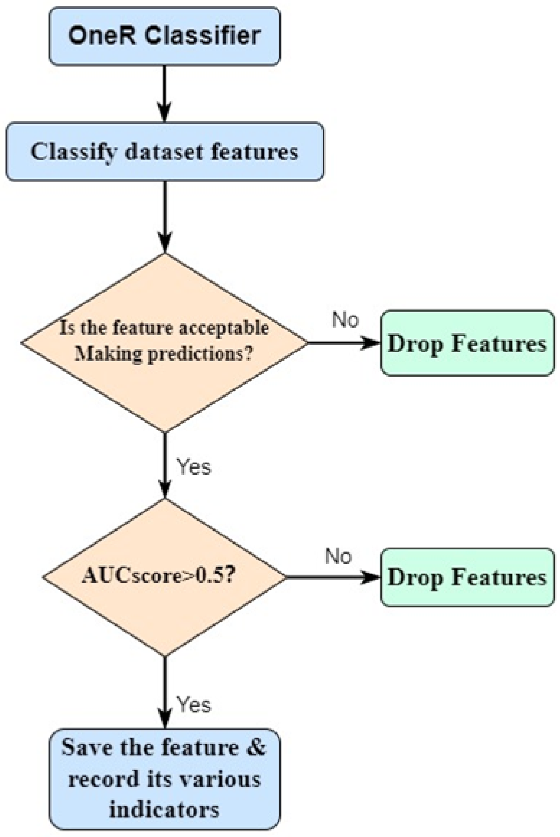
In this process, the decision tree has a depth of 1 layer and uses the Gini coefficient. After fitting the training set, predictions are made on both the testing and training sets using the prediction function:

$$preds = rootnode.predict(X\_test[feature].array.reshape(-1,1))$$
$$preds\_tr = rootnode.predict\left(X\_train[feature].array.reshape(-1,1)\right)$$

The evaluation metrics are then calculated by comparing the predicted results with the accurate labels. If the metric evaluation value is more significant than 0.5, the feature name, metric value, and prediction result of that feature are saved. These values are stored in a list and converted into a data frame. The metric evaluation value, specifically the roc_auc_score, is calculated directly using the metrics method from the sklearn library. The results are sorted in descending order:

$$pd.DataFrame\ (data = results,\ columns = ['feature',\ 'roc\_auc\_score','fitted\_models',\ 'predictions',\ 'preds\_train']\ ).sort\_values\ (\ by = 'roc\_auc\_score',\ ascending = False)$$

Figure 2 The working principle of feature selection for OnerR classifier (see online version for colours)


OneR Classifier
Classify dataset features
Is the feature acceptable Making predictions?
No
Drop Features
Yes
AUCscore>0.5?
No
Drop Features
Yes
Save the feature & record its various indicators

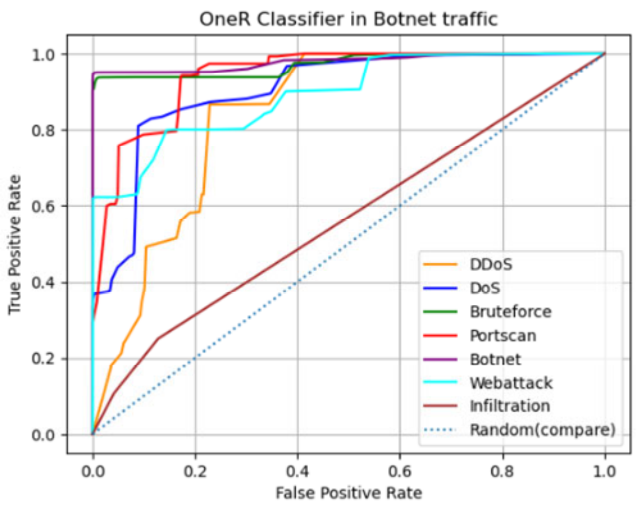
Based on the metric evaluation value (roc_auc_score) greater than 0.5, features with significant discriminatory power are selected. The average prediction results on the testing and training sets are computed for these valuable features, resulting in an overall prediction result vector. The ROC curve parameters are then calculated based on the overall prediction results and proper training set labels. The calculation method for the ROC curve is as follows:

$$ROC = TPR - FPR \tag{1}$$

True positive rate (TPR) represents the probability of correct predictions, and false positive rate (FPR) represents the probability of incorrect predictions. The area under the ROC curve (AUC score) is used as a widely recognised standard for evaluating the overall performance of classifiers. The AUC value ranges from 0 to 1, with higher values indicating better performance. The random selection benchmark is set at an AUC score of 0.5, with features having an AUC score greater than 0.5 considered more informative for classification. The flowchart of the entire OneR classifier process is depicted in Figure 2.

And after constructing the OneR classifier, the study proceeds by inputting the samples, excluding Benign, into the classifier for each of the seven attack methods. The ROC curves are then generated based on the best-performing feature for each attack, as shown in Figure 3.

Figure 3 Roc curve of OnerR classifier in botnet traffic (see online version for colours)



The OneR classifier performs perfect ROC curves in attacks such as Bruteforce and Botnet. It indicates its ability to accurately identify most samples in these attack categories without additional learning. However, its effectiveness diminishes when dealing with Infiltration attacks, as it exhibits no discerning capability. The study then proceeds to use the metrics class from the sklearn library to compute and obtain the top-ranked 2-dimensional features for each attack, resulting in 14 features, as shown in Table 5. From these features, the most effective ones are selected as inputs for the detection model.

In the subsequent detection model training, feature selection is a crucial step in achieving high performance. Among the features selected by the classifier, since features 17, 47, and 48 have appeared, and feature 8 (Bwd IAT Max) has shown poor performance, these four features are excluded. The remaining 10-dimensional features are chosen as the input for training the model.

Table 5 AUC score for filtering features of various attack categories

Attack means	*Number*	*Feature*	*AUC score*
DDoS	33	Fwd packet length max	0.7279
	34	Avg Fwd segment size	0.7257
DoS	17	Bwd packets/s	0.8171
	38	Fwd seg size min	0.7846
Bruteforce	7	Bwd header length	0.9638
	25	Fwd act data packets	0.9587
Portscan	47	Packet length Std.	0.8580
	48	Packet length variance	0.8580
Botnet	17	Bwd packets/s	0.8705
	14	Bwd packet length mean	0.8610
Webattack	48	Packet length variance	0.8074
	47	Packet length Std.	0.8065
Infiltration	2	Active min	0.5453
	8	Bwd IAT max	0.5324

3.3 *DQN detection model*

'DRL' was proposed by DeepMind in 2015. They combined the excellent performance of high-dimensional features in deep learning with the reward mechanism in reinforcement learning. They used the fused algorithm to complete the challenge of Atari games. According to their report, the performance of DRL after training is even stronger than some experts. The following will introduce the DRL algorithms used in this paper.

3.3.1 *Q-learning*

Q-learning is a value iteration-based reinforcement learning algorithm for solving Markov decision process (MDP) problems. The *Q*-learning algorithm aims to learn an optimal policy that maximises the long-term cumulative reward. In *Q*-learning, we learn a state-action value function (*Q*-function) representing the expected cumulative reward obtained by taking a particular action in a given state. The algorithm is shown as follows.

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r + \gamma \max_{a'} Q(s_{t'},a_{t'}) - Q(s_t,a_t)\right] \quad (2)$$

In equation (2), '*St*' represents the current state, '*at*' represents the action taken in the current state, '*r*' represents the immediate reward obtained after taking the action *at*, '*s*' represents the next state reached after taking the action, and 'α' is the learning rate parameter. This calculation formula uses temporal difference learning to update the *Q*-value. It selects the maximum *Q*(*St*', *at*') from the next state '*s*' and multiplies it by 'γ', then adds the actual reward value '*r*' to obtain the final true value of *Q*(*St*, *at*). *Q*-learning aims to learn an optimal state action-value function that maximises the cumulative return on taking actions in any state. During the learning process, we adopt a greedy strategy to select the optimal action in the current state:

$$at = \arg\max Q(St, at) \quad (3)$$

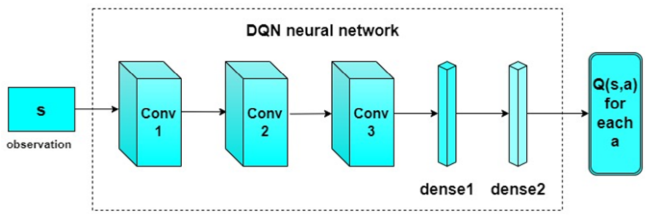
In equation (3), Update the *Q*(*St*, *at*) value function by continuously using the Bellman equation until the value function converges to the optimal value function $max_a Q(St, at)$. When the value function converges, the optimal strategy can obtain by selecting the action with the most significant value function in each state.

The advantage of the *Q*-learning algorithm is that it can handle continuous state and action space and does not need models (model-independent methods). Even if we do not know the model of the environment, we can use *Q*-learning to learn the optimal strategy.

3.3.2 *Deep Q network*

DQN is an algorithm based on DRL, an extension of the *Q*-learning algorithm in deep learning. DQN uses a neural network to approximate the *Q*-value function, which can learn in high-dimensional and complex state spaces and has a certain degree of generalisation ability.

Figure 4 DQN neural network structure (see online version for colours)



DQN consists of four main modules: the model, agent, memory, and DQN algorithm. Each agent's decision affects the environment's changes, and the model processes this feedback and stores the data in memory. The model provides the agent with a reward value based on the information, allowing the agent to update its decision continually. The memory releases data to allow the agent to undergo multiple training sessions, preventing problems with the expected *Q*-value being too large. Sample and memory data are passed back and forth to the agent for training until the model converges and completes the training.

The DQN algorithm starts by providing an environment states to the agent, who then utilises the value function network to obtain all $Q(s, a)$ values associated with states. The agent decides by employing the ε-greedy strategy to select an action a. The environment responds to the action by providing a reward value '*r*' and the following environment states'. This completes one epoch. The parameters of the value function network are then updated based on the reward value '*r*,' and the process proceeds to the next epoch. This cycle continues until a converged value function network is trained, concluding the loop. The workings of one epoch are depicted in Figure 4.

At the algorithm level, DQN extends the Q-learning algorithm. However, since DQN approximates the Q-values using a neural network, determining the 'θ' weights of the neural network becomes crucial in the entire process. In this study, the gradient descent algorithm is employed to minimise the loss function, continuously adjusting the θ weights. The algorithm for the loss function is depicted as equation (4).

$$L_i(\theta_i) = E_{(s,a,r,s')\sim U(D)}\left[y_i - Q(s,a;\theta_i))^2\right] \tag{4}$$

In equation (4), s represents the state, a represents the action, r represents the immediate reward, s' represents the next state, θ_i represents the parameters of the neural network in the *i*-th iteration, $Q(s, a, \theta_i)$ represents the Q-value function for taking action a in states with θ_i, U(D) represents the uniformly sampled experience set from the experience replay pool, and D represents the target Q-value, which is the representation of the experience pool itself. y_i calculated as follows:

$$y_i = r + \max_{a'} Q\left(s', a'; \theta_i^-\right) \tag{5}$$

In equation (5), θ_i^- represents the parameters of the target neural network at the *i*-th iteration. $Q\left(s', a'; \theta_i^-\right)$ represents the output of the target network. By introducing the target network, the target Q-values remain unchanged for a certain period of time, reducing the correlation between the current Q-values and the target Q-values to some extent, thereby improving the stability of the algorithm, which helps prevent overfitting.

Performing botnet detection using DQN mainly consists of three steps. Firstly, the Q-network is trained to identify the features of a large amount of network traffic and store multiple predicted results into the experience replay buffer. Then, the model is continuously trained through self-play. Finally, the trained model is used to detect anomalous traffic.

PARL is a classic framework based on DRL, which is widely used in various fields such as unmanned driving and game control using DRL algorithms. The DQN model architecture used in this article is built on the PARL framework and the PaddlePaddle library of PaddlePaddle, and has been specifically improved for traffic detection based on the general model.

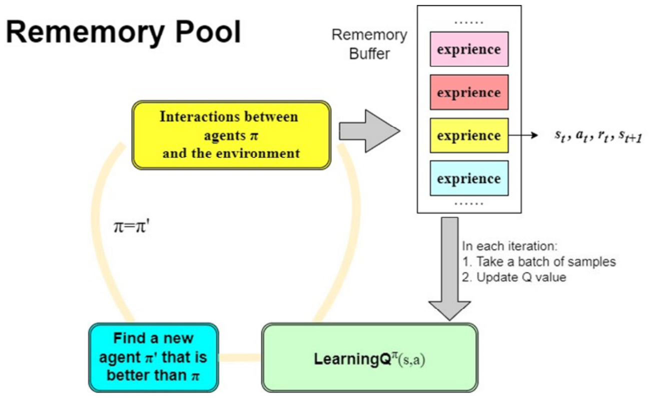
To achieve this, we first construct the framework and core algorithms of the model using Model and Agent classes. The Model class uses the PaddlePaddle layers module to create three fully connected layers, with ReLU activation functions in the first two layers and no activation function in the last layer, whose output dimension is the action space. The value function is then defined to output the state value function Q by processing the input state obs. through the fully connected layers. In the Agent class, we define the functions build_program, sample, predict, and learn to enable the agent to make decisions based on the maximum Q value as much as possible and to converge by slowly decreasing the ε-greedy value to prevent continuous exploration.

In the botnet traffic detection model, to facilitate agent learning, when the agent selects an action to classify the traffic type, it receives a reward value 'r'. The reward function is defined as follows:

$$r = \begin{cases} 5, & \mathit{True} \\ -5, & \mathit{False} \\ -0.1, & \mathit{Not\ judged\ every}\ 1\ \mathit{ms} \end{cases} \tag{6}$$

When the agent makes a correct selection, positive feedback of 5 scores is rewarded. On the other hand, when the agent makes an incorrect selection, negative feedback of –5 scores is given. Additionally, to prevent the agent from avoiding making decisions indefinitely, a penalty of 0.1 scores is deducted every 1 ms. This approach improves the model's accuracy rate and enhances its decision-making speed.

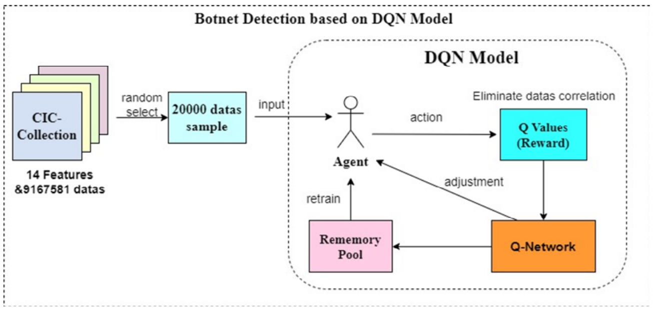
Figure 5 Working principle of re-memory pool (see online version for colours)



In addition, DQN introduces the replay memory mechanism. Traditional Q-learning algorithms interact and improve based on the current policy, discarding samples generated from each model's utilisation interaction after learning. This approach is inefficient in utilising data samples but also leads to correlated data samples. If the model assumes the presence of correlations between data samples that do not exist, its decision-making ability will be significantly compromised. Therefore, in DQN, a quadruple (*st*, *at*, *rt*, *st*′) is added to the experience replay memory as a data sample. During model learning, these samples are interleaved with new data samples to change the model's policy and update the Q-network parameters. Collected samples are sequentially stored in the experience replay memory according to their time order. If the memory is full, new samples will overwrite the oldest samples in the memory. The experience replay memory uniformly and randomly sample a batch of samples from the cached samples for the model to learn. This approach ensures that each training sample typically comes from multiple interaction sequences, reducing data correlations and improving sample utilisation. The working principle of the experience pool on the DQN model is shown in Figure 5.

After the model is constructed, 20,000 randomly selected samples from the pre-processed botnet dataset are inputted for training until convergence. The entire working principle of the DQN botnet traffic detection model is illustrated in Figure 6.

Figure 6 DQN botnet detection model (see online version for colours)



The overall Pseudocode of the DQN detection model is as follows:

Botnet detection based on DQN model

```
Input: state $s$, action $a$, discount factor $\gamma$, learning rate $\alpha$, greedy factor $\varepsilon$
1   Initialise replay memory $D$ to capacity $N$;
2   Random initialise $Q$ network weight $\theta$.
3   Initialise target $Q$ network weight $\theta^-$ so that $\theta^- = \theta$
4   for epoch = 1, M do:
5       Initialise start state ;
6         for $t$ =1, T do:
7             In state $s$,select action $at$ through the $\varepsilon$-greedy;
8                 Execute action $a$, observe the judgement results, and receive immediate rewards $r$ and a new state $s'$.
9             Push ($s$, $a$, $r$, $s'$) into the replay memory $D$;
10            sampling $st$, $at$, $rt$, $st'$ from $D$;
11            $Q = \begin{cases} r_t, & s_{t'} \text{ in a resting rate} \\ r_t + \max_{a'} Q_{\theta^-}(st, a'), & else \end{cases}$
12            Using $(y - Q_e(s, a))^2$ as loss function to train $Q$ network;
13            Update status, which is $s \leftarrow s'$;
14                Every C steps, Update target network weight, which is $\theta^- \leftarrow \theta$;
15            When $s$ is in the termination state, end for;
16      When $\forall s$, $a$, $Q_e(s, a)$ convergence, end for;
Output: DQN model training completed.
```

4 Experimental process and results

4.1 Experimental environment

This experiment using the computer operating system about Windows 10, with an i7-7200 CPU and 16G of memory. The development environment includes Python 3.6.5 and Pycharm, and the DQN detection model is built using the PaddlePaddle and PARL frameworks.

4.2 Data

In Section 3.1, the dataset has been processed in this study. A random sample of 20,000 data samples was selected according to the proportion for the experiments. The quantity and labels of the selected dataset are shown in Table 6.

Table 6 Randomly selected botnet dataset samples

Botnet traffic class	*Samples number*	*Select label*
Benign	13,998	1
DDos	2,928	2
Dos	1,104	3
Botnet	556	4
Bruteforce	460	5
Infiltration	444	6
Webattack	264	7
Portscan	264	8
Total	20,000	

4.3 Model assessment

In this study, two approaches were employed for feature classification: one using the OneR classifier and the other randomly selecting 10-dimensional features. Apply two sets of features to DQN and LSTM and CNN+RNN models compared to DQN, and compare their experimental results in the test set. Select accuracy rate and precision rate as the evaluation criteria. The formulas for accuracy rate and precision rate are shown in equations (7) and (8):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

$$Precision = \frac{TP}{TP + FP} \tag{8}$$

In equation (7) and (8), TP represents the number of positive samples predicted as positive, FP represents the number of negative samples predicted as positive, TN represents the number of positive samples predicted as negative, and FN represents the number of negative samples predicted as negative.

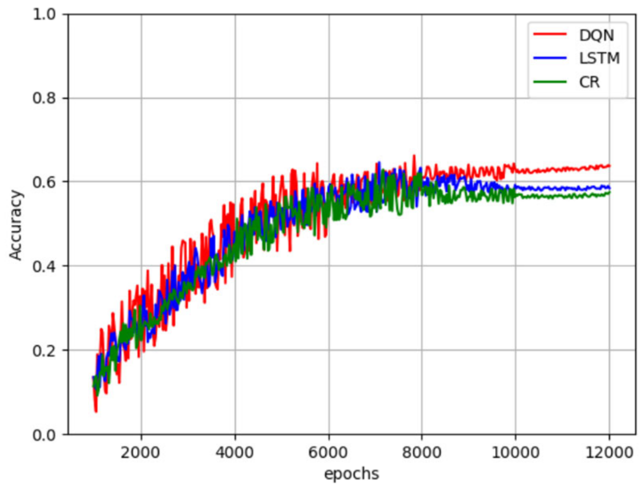
4.4 Analysis of experimental results

In this section, we will conduct three sets of comparative experiments: training the DQN detection model, LSTM detection model, and CNN + RNN(CR) detection model using the features filtered by the OneR classifier and randomly selected features, respectively, to verify the excellent ability of OneR classifier to filter features; Using the same pre-processed CIC-ISD2017 dataset instead of the CIC-collection test set, the detection ability of three types of models in unfamiliar datasets are tested to verify the high adaptability of DRL.

4.4.1 Model comparison with randomly selected features

In this experiment, 10-dimensional features were randomly selected using the random function and used for training in two models. The comparison of accuracy rate and precision rate obtained is shown in Figures 7 and Figure 8.

Figure 7 Accuracy rate comparison of three models (see online version for colours)



Based on Figures 4 and 5, it shows that the accuracy rate of the DQN detection model is very high in the early oscillation frequency, which is caused by the ε-greedy of the DQN detection model. This result is that the accuracy rate of the DQN continues to decrease in the early training. During the training process, as each of the three models converges, the e-discount factor decreases from the initial set of 0.1 to 0.01, ultimately allowing the model to complete the training. Both in terms of accuracy rate and precision rate, the DQN detection model is superior to the LSTM and CR detection models. The randomly selected features prevented all three models from achieving good detection performance in the test set, and even the best-performing DQN only achieved an accuracy rate of 63.74%, with an accuracy rate at most 80%. These results are not qualified for a detection model. Therefore, we will add a OneR classifier to filter features and complete the second comparative experiment.

Figure 8 Precision rate comparison of three models (see online version for colours)

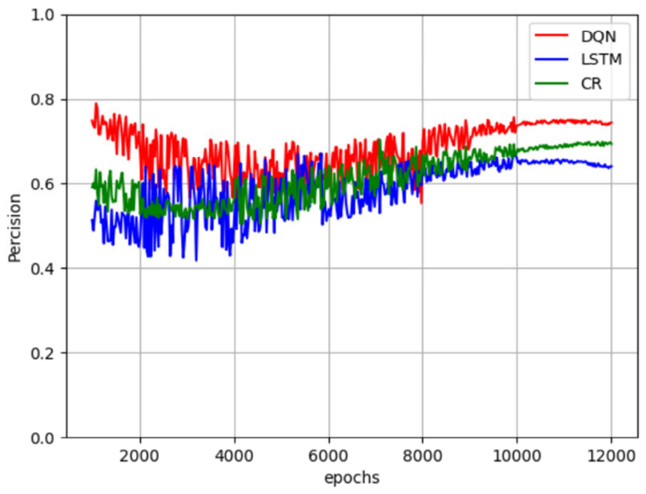
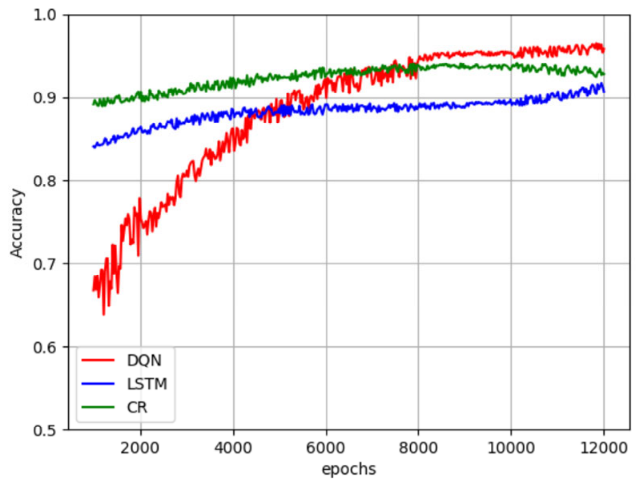


Figure 9 Accuracy rate comparison of three models (see online version for colours)



4.4.2 Model comparison with OneR Classifier

The 10 features selected by the OneR classifier, as mentioned in Section 3.2, are used for training. These features include 17th Bwd. Packets/s, 47th Packet Length Std., 48th Packet Length Variance, and others. The comparison of accuracy rate and precision rate obtained is shown in Figures 9 and Figure 10. Since all data of the three models are higher than 0.5, the ordinate range of the line chart in Figures 9 and Figure 10 is set between 0.5–1. After observing the Figures 9 and 10, it can be observed that using the features provided by the OneR classifier for training, the accuracy rate and precision rate of the three models have been greatly improved, indicating that the feature selection ability of the OneR classifier is excellent in completing the task. Unlike when LSTM and CR approaching converge at less than 500 epochs at the beginning, the DQN detection model slowly converges after

8,000 epochs, indicating that DQN places great emphasis on early exploration and sacrifices its accuracy rate. After the model slowly converged, the judgment ability of the DQN detection model began to show, ultimately surpassing the LSTM and CR detection models. In terms of precision rate, DQN inherits its performance in random feature selection experiments, and exploration not only slows down its precision rate improvement but also reduces its accuracy rate. However, as the model converges, the precision rate difference between the three models is slight, but DQN detection is still slightly better.

However, more is needed to improve the detection model, so the next experiment will use an unfamiliar dataset to test the testing ability of these three detection models that have already been trained through the OneR classifier and compare which model has more robust adaptability.

Figure 10 Precision rate comparison of three models (see online version for colours)

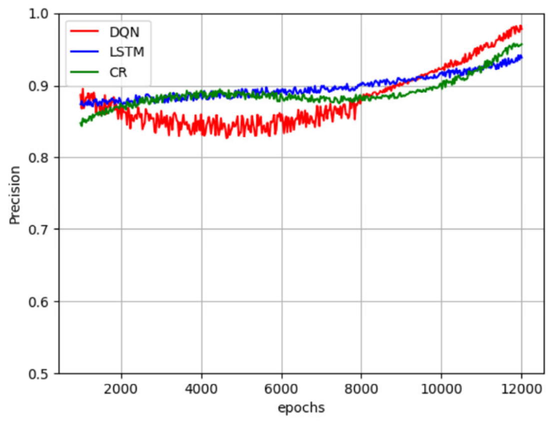
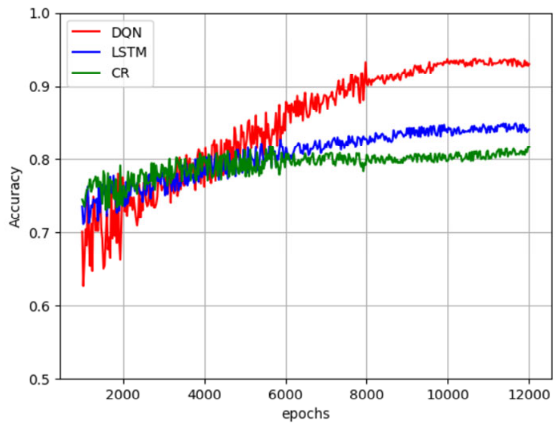


Figure 11 Accuracy rate comparison of three models (see online version for colours)

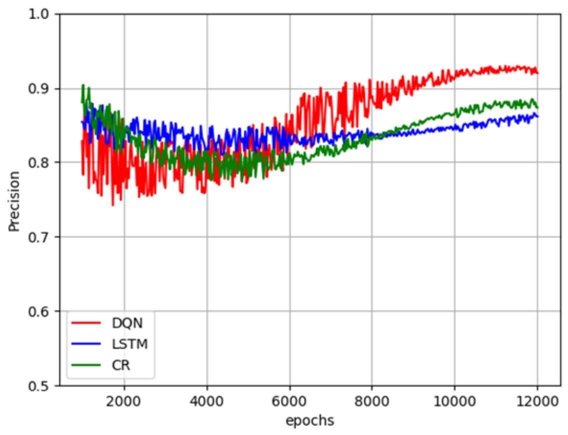


4.4.3 *Model comparison with other datasets*

In order to ensure the consistency of the labels, this experiment used CIC-ID2017, one of the original datasets of CIC-Collection, as the test set. After data pre-processing in Section 3.1, 20,000 data samples were randomly selected, and three detection models that had already been trained in the previous experiment were used to observe their accuracy and precision in this dataset. The comparison of accuracy rate and precision rate obtained is shown in Figures 11 and Figure 12. Since all data of the three models are higher than 0.5, the ordinate range of the line chart in Figures 11 and Figure 12 is set between 0.5–1.

From Figures 11 and 12, it can be observed that the detection ability of the DQN detection model in the new dataset is more robust than that of the LSTM detection model and the CR detection model. Although the DQN detection model converges last every time it is trained, and the accuracy and precision of the initial model are not as good as the other two models, with the increase of epochs, the accuracy and precision of the DQN detection model can still be improved, rather than stabilising like LSTM and CR. Whether in terms of accuracy or precision, the DQN detection model can still reach over 90% in the later stage, but the LSTM detection model and CR detection model fell by 90%. Although the detection ability of all three models in unfamiliar datasets has decreased, the decrease in the DQN detection model is acceptable, as it still has sufficient detection ability for unfamiliar datasets. This experiment demonstrates the strong adaptability of the DQN detection model.

Figure 12 Precision rate comparison of three models (see online version for colours)



However, the DQN detection model is not perfect either. In each comparative experiment, the training and judgment time of the DQN detection model is always the longest, with only 20,000 samples. The LSTM detection model and CR detection model only took 3–5 minutes to complete the training. However, for the DQN detection model, it takes 8 minutes to complete the training, which may vary due to differences in computer CPU and memory. However, it cannot be denied that the training time of the DQN detection model is longer than that of the LSTM and CR detection models.

5 Conclusions

This paper proposes a DQN algorithm detection model based on DRL by designing a botnet traffic detection model. Compared with existing LSTM and CR detection models, the DQN model improves judgment ability and reuses sample resources by accumulating Q-values and using experience pools. The experimental results show that the feature selection using OneR classifier improves the detection ability of the model. In addition, the stability of the DQN model enables it to better adapt to changes in different datasets. Regardless of whether the OneR classifier using for feature selection, the detection model constructed using DQN outperforms LSTM and CR detection models in terms of accuracy and accuracy. However, due to the presence of not only neural networks but also Q value prediction and experience pool storage in DQN, the model becomes more complex and requires more time and space costs than LSTM and CR algorithms. Moreover, the DQN model is sensitive to parameters, and unsuitable features can lead to poor performance.

Future research could explore other DRL algorithms that can improve over time and reduce the time cost of model construction. It may also be necessary to adopt other classifiers to analyse and train new models from the perspective of multiple features. Despite its limitations, our proposed method represents a significant step forward in botnet network detection. It can potentially contribute to developing more accurate and efficient detection methods.

Acknowledgements

This study was supported by the Liaoning Province Applied Basic Research Program, 2023JH2/101600038. The authors gratefully acknowledge the anonymous reviewers for their valuable comments.

References

Abdullah, Z., Saudi, M. and Anuar, B. (2014) 'Mobile botnet detection: proof of concept', *IEEE 5th Control and System Graduate Research Colloquium*, Vol. 5, No. 2, pp.257–262.

Alqatawna, J., Ala, M., Hassonah, M. et al. (2021) 'Android botnet detection using machine learning models based on a comprehensive static analysis approach', *Journal of Information Security and Applications*, Vol. 2021, No. 58, pp.1–14.

Anwar, S., Zain, J.M., Inayat, Z. et al. (2016) 'A static approach towards mobile botnet detection', *International Conference on Electronic Design (ICED)*, Vol. 3, No. 1, pp.563–567.

Apruzzese, G. et al. (2020) 'Deep reinforcement adversarial learning again-st botnet evasion attacks', *IEEE Transactions on Network and Service Management*, Vol. 17, No. 4, pp.1808–1821.

Braams, B. (2018) 'Predicate pushdown in parquet and databricks spark', *The Netherlands: Universiteit van Amsterdam*, Vol. 2018, No. 4, pp.849–911.

Chen, F. (2022) 'Research on botnet detection technology based on deep learning', *Guangdong University of Technology*, Vol. 12, No. 2, pp.1–56.

Christopher, D., Majdani, F. and Petrovski, V. (2018) 'Botnet detection in the internet of things using deep learning approaches', *2018 International Joint Conference on Neural Networks (IJCNN)*, Vol. 7, No. 1, pp.8–13.

Costa, V., Barbon, S., Miani, R. et al. (2019) 'Mobile botnets detection based on machine learning over system calls', *International Journal of Security and Networks*, Vol. 14, No. 2, pp.103–118.

Dhayal, H. and Kumar, J. (2018) 'Botnet and P2P botnet detection strategies: a review', *2018 International Conference on Communication and Signal Processing (ICCSP)*, Vol. 4, No.3, pp.1077–1082.

Fu, Z., Xu, Y. and Wu, Z. (2020) 'SVM-KNN network intrusion detection method based on incremental learning', *Computer Engineering*, Vol. 46, No. 4, pp.115–122.

Girei, A., Shah, A. and Shahid, B. (2016) 'An enhanced botnet detection technique for mobile devices using log analysis', *International Conference on Automation and Computing (ICAC)*, Vol. 2016, No. 22, pp.450–455.

Heng'an Jiaxin Technology Co., Ltd. (2020) '2020 network security situation report', *Information Security Research*, Vol. 7, No. 3, pp.198–206.

Huang, B. (2008) 'Research on intensive learning method and its application', *Shanghai Jiaotong University*, Vol. 2008, No. 8, pp.124–239.

Huang, Z. (2020) 'Focus on the global network security situation in the first half of 2020', *China Information Security*, Vol. 7, No. 15, p.68.

Hussain, F., Abbas, S.G., Pires, I.M. et al. (2021) 'A two-fold machine learning approach to prevent and detect IoT botnet attacks', *IEEE Access*, Vol. 2021, No. 9, pp.163412–163430.

Joshi, S. and Abdelfattah, E. (2020) 'Efficiency of different machine learning algorithms on the multivariate classification of IoT botnet attacks', *IEEE Annual Ubiquitous Computing, Electronics and Mobile Communication Conference (UEMCON)*, Vol. 11, No. 2, pp.517–521.

Kadir, A., Stakhanova, N. and Ghorbani, A. (2015) 'Android bot-nets: what urls are telling us', *International Conference on Network and System Security*, Vol. 12, No. 3, pp.78–91.

Kirubavathi, G. and Anitha, R. (2018) 'Structural analysis and detection of android botnets using machine learning techniques', *International Journal of Information Security*, Vol. 17, No. 2, pp.153–167.

Lu, F. (2021) 'Research on botnet detection technology based on deep learning', *Nanjing University of Posts and Telecommunications*, Vol. 2021, No. 1, pp.1–68.

Lu, H. (2020) 'Research on complex network representation learning based on graph convolution', *Nanjing University of Posts and Telecommunications*, Vol. 2020, No. 3, pp.1–44.

Pektas, A. and Acarman, T. (2018) 'Botnet detection based on network flow summary and deep learning', *Computer Science*, Vol. 4, No. 26, pp.1–15.

Qi, Z. (2022) 'Research on malicious network traffic detection technology based on neural network', *Guangdong University of Technology,* Vol. 2022, No. 1, pp.1–50.

Randhawa, R., Aslam, N., Alauthman, M. et al. (2021) 'Deep reinforcement learning based evasion generative adversarial network for botnet detection', arXiv preprint, arXiv:2210.02840, 2021.

Torres, P., Catania, C., Garcia, S. et al. (2016) 'An analysis of recurrent neural networks for botnet detection behavior', *2016 IEEE Biennial Congress of Argentina (ARGENCON)*, Vol. 2016, No. 6, pp.1–6.

Venturi, A. et al. (2021) 'DReLAB – deep reinforcement learning adversarial botnet: a benchmark dataset for adversarial attacks against botnet intrusion detection systems', *Data in Brief*, Vol. 2021, No. 34, pp.106631–106643.

Volodymyr, M., Koray, K., David, S. et al. (2015) ‘Human-level control through deep reinforcement learning’, *Nature*, Vol. 2015, No. 2, pp.529–533.

Wu, D., Fang, B., Wang, J., Liu, Q. and Cui, X. (2019) ‘Evading machine learning botnet detection models via deep reinforcement learning’. *IEEE International Conference on Communications (ICC)*, Vol. 2019, No. 3, pp.967–973.

Xu, J. (2016) ‘Research on mobile botnet detection methods’, *Computer Technology and Development*, Vol. 26, No. 12, pp.117–121.

Xue, H. and Wang, C. (2022) ‘Research on botnet traffic detection based on GAN’, *Electronic Design Engineering*, Vol. 30, No. 17, pp.146–149.

Yerima, Y. and Bashar, A. (2021) ‘Bot-IMG:A framework for image-based detection of Android botnets using machine learning’, *ACS/IEEE International Conference on Computer Systems and Applications (AICCSA)*, Vol. 2021, No. 18, pp.1–7.

Zeidanloo, R. and Manaf, B. (2010) ‘Botnet detection based on traffic monitoring’, *International Conference on Networking and Information Technolog.*, Vol. 7, No. 11, pp.97–101.

Zhao, D., Traore, I. and Ghorbani, A. (2013) ‘Botnet detection based on traffic behavior analysis and flow intervals’, *Computers and Security*, Vol. 39, No. 4, pp.2–16.

Zhao, X. and Ding, S. (2018) ‘Overview of deep intensive learning research’, *Computer Science*, Vol. 45, No. 7, pp.1–6.