Intrusion Detection Based on Reinforcement Learning

by

Bin Yang

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Control Systems

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Powered by advancements of information and Internet technologies, there has been a rapid development in network based applications in recent years. Meanwhile, it is recognized that more attentions need to be paid to the issue of cybersecurity. The security of the network environment plays a vital role in stable functioning of the society.

Research on cybersecurity has become more active lately. Researchers have proposed a number of approaches to protect the network. Among them, a broadly practiced approach is the intrusion detection system (IDS). Building a powerful and robust intrusion detection system is long-established as it can provide effective protection to prevent Internet from intrusions and attacks. Through pattern or rules matching, the intrusion detection system can filter out harmful traffics. However, traditional rule-based intrusion detection systems are unqualified to acclimate to the ever-changing network environments because rules are drawn up manually. Thus, a significant number of research works are focused on the development of novel methods to handle the new challenges. Benefiting from the vigorous development of machine learning and artificial intelligence, researchers have been actively deploying these new technologies to handle network traffic analytics, data processing and feature engineering, which are important modules in building the intrusion detection system. Machine learning techniques have already achieved substantial success in the area of cybersecurity.

Reinforcement learning (RL) is one of the most significant and compelling methodologies of machine learning. It is used to describe and solve the problem of the agent in the process of interaction with the environment through learning the strategies to

maximize returns or achieve specific goals. RL has achieved considerable accomplishments in a multitude of fields, such as games, robotics and autonomous systems. For example, RL has been shown as the most promising method in designing game AI agents. In some games, RL agents have outcompeted top professional players. Inspired by the success of RL in other areas, in this work, we intend to study how it can be utilized in designing the intrusion detection system to improve the cybersecurity.

This thesis is mainly divided into two parts. Chapter 3 includes an empirical study of Proximal Policy Optimization Algorithm (PPO), one of the most well-known reinforcement learning algorithms. During this empirical study, we can further understand how RL algorithms work in game AI, and hope to find common ideas between game AI and the cybersecurity research. This way, we can apply the reinforcement learning framework to solve the intrusion detection tasks. Hence, the second part is focused on designing intrusion detection systems (IDS) based on reinforcement learning. The approaches are categorized into packet-based and flow-based, which are separately addressed in Chapter 4 and Chapter 5. In this part, network traffic data are firstly processed using different methods, and subsequently, reinforcement learning algorithms are developed as the overall framework of the intrusion detection system.

Acknowledgements

There are many people I felt so grateful to. I could not have completed my MSc study without the help of these people.

At first, I would like to thank my supervisor, Dr. Qing Zhao. Without her careful cultivation and guidance in the past two years, I could not complete my research work and this thesis.

Next, I would like to express my great gratitude towards my family, my father, mother and sister. With their help, I had the opportunity to come to University of Alberta to learn the most advanced artificial intelligence knowledge.

Table of Contents

List of Tables

List of Figures

List of Symbols

Latin

a	action
G	cumulative reward
p	transition function
Q	state value function
r	reward
s	state
V	value function

Greek

α	learning step
χ	label space of DDoS2019
Γ	GAN exploration level
γ	discount value
λ	anomaly threshold
π	policy
ε	exploration level

Abbreviations

AI stands for artificial intelligence.

CGAN stands for conditional generative adversarial networks.

CNNs stands for convolutional neural networks.

CV stands for computer vision.

DNNs stands for deep neural networks.

DQN stands for Deep Q-Learning.

FCN stands for fully connected neural networks.

GAN stands for generative adversarial networks.

HTTP stands for HyperText Transfer Protocol.

IDS stands for intrusion detection system.

IP stands for Internet Protocol Address.

ML stands for machine learning.

NLP stands for natural language processing.

PPO stands for Proximal Policy Optimization.

RF stands for random forest.

RL stands for reinforcement learning.

SAE stands for stacked autoencoder.

SVM stands for support vector machine.

TCP stands for Transmission Control Protocol.

UDP stands for User Datagram Protocol.

Chapter 1

Introduction

1.1 Background

Network is the substantial underpinning for developing a modernized economy. The progression of all professions and trades, including business, scientific research, entertainment as well as education, requires the assistance of a reliable and secure network system. However, numerous network security problems have also emerged, causing tremendous damages to individuals or the collective. For example, in April 2020, World Health Organization (WHO) announced that it had encountered an increased number of cyber attacks, with about 450 email addresses and passwords of WHO and thousands of related staff exposed. If the network infrastructure is damaged and destroyed, not only will personal information be leaked, but the whole society will be thrown into chaos. Therefore, the research of cybersecurity is crucial for the successful and stable development of the whole society.

To protect the network infrastructure and confidentiality of data, a series of approaches are proposed by researchers. For example, most computers are equipped with firewalls that can impede some malicious traffics. An intrusion detection system (IDS) is a type of network security equipment that monitors the network transmission in real-time and gives an alarm or takes active reaction measures when the suspicious transmission is found. IDS is a proactive security technique to protect IT infrastructure from being destroyed by malicious attacks. Conventional intrusion detection

systems depend on pattern matching, for which specific filtering rules are determined in advance manually [1].

Seeing the rapid development and the success of artificial intelligence (AI) attained in research areas such as computer vision, natural language processing and robotics, an increasing number of network engineers and researchers start exploring the applications of artificial intelligence techniques in cybersecurity. As mentioned above, conventional intrusion detection systems rely on user-defined filtering rules, under which the matched network traffic will be filtered. However, this filtering method has severe defects [2]. Firstly, it is difficult to set up faultless filtering rules, even if the rules are specified by experts. In the face of tremendous network traffic, along with a huge variety of attacks, limited artificial rules are unable to cover the characteristics of these different types of network traffic. Secondly, from hackers' perspective, rules are formulated by people, so hackers can easily figure out the pattern of those rules by certain tests. Hence, it is straightforward for them to launch attacks that can bypass those rules. Thirdly, the update of rules is inefficient. Because the malicious network attacks are evolving quickly, intrusion detection systems also require frequent updates and upgrades to detect these attacks. However, it is time-consuming and laborious to determine what kind of outdated rules should be discarded and to lay down pivotal regulations.

Employing artificial intelligence approaches in intrusion detection systems can mitigate the problems mentioned earlier [3–6]. When using machine learning based model as the IDS to detect malicious network attacks, the filter rules are rendered automatically. Machine learning models, such as deep neural networks (DNNs) [7], can learn representative features of different network attacks efficiently and then implement classification or detection, hence manual rules are not required anymore. Because neural networks are a type of black box to everyone, hackers can not acquire what kind of patterns and relationships are captured by the neural networks. Therefore, it is much harder for hackers to launch novel attacks that can bypass the detection.

Reinforcement learning research [8] has been booming during recent years. Incorporating with deep neural networks (DNNs), reinforcement learning algorithms outperform in many fields in comparison to traditional deep learning approaches, such as games, recommendation systems and many scientific fields. Some well-known reinforcement learning algorithms including Deep Q-Learning (DQN) [9] and Proximal Policy Optimization (PPO) [10], have already been widely used in many areas. AlphaGo [11] is one of the most representative achievements of RL in games. In addition, many enterprises, such as YouTube [12], and Alibaba [13], etc., are exploring novel RL based recommendation techniques. Also, RL has found many great applications and research value in various engineering fields. Studies [14] use deep reinforcement learning algorithms to solve the autonomous underwater vehicles (AUVs) low-level control problem. Many researchers [15, 16] apply reinforcement leaning algorithms, especially PPO for unmanned aerial vehicles (UAVs) control. These successful applications demonstrate that reinforcement learning is at the stage of rapid growth.

1.2 Motivation

Motivated by the importance of the cybersecurity issue and the promising results of RL in many applications, we conduct research mainly on these two topics in this thesis.

Policy gradient methods, such as Asynchronous Advantage Actor-Critic (A3C) [17], Trust Region Optimization (TRPO) [18], and Proximal Policy Optimization (PPO), are widely applied in RL in devising game agents, including both discrete-action and continuous-action games. Among these methods, PPO stands out by achieving high scores in many test beds, such as Atari games and MuJoCo [19]. The core innovation of PPO is the clipped surrogate objective. However, studies [20–23] indicate that the high achievements of PPO result from some additional code-level optimizations, instead of its key idea in clipped surrogate objective. Thus, further understanding of these code-level optimizations is essential for applying PPO to tackle more problems,

such as the intrusion detection.

Reinforcement learning and intrusion detection are remarkably compatible in many aspects. Reinforcement learning algorithms are used for solving Markov problems [24]. Essentially, network flow is a special type of dynamic process, which can be modelled by a Markov process (See Chapter 4 and 5). Besides, when making a decision related to current state, the RL agent can consider foresighted states by introducing discount factors when calculating the cumulative reward (See Chapter 2). This is also conducive to intrusion detection. The network flow consists of a sequence of different packets. When the current packet is examined, those packets behind it in the same flow can also provide useful information. Treat a packet as a state, by using the cumulative reward with the discount factor, reinforcement learning algorithms can be employed to take the features of future packets into account. Furthermore, intrusion detection can be considered as a special game (See Chapter 4 and 5). Hence, it is feasible to formulate and solve the intrusion detection problem in the reinforcement learning framework. Motivated by these pertinent aspects, we attempt to study how to improve the performance of intrusion detection by using the RL methods.

1.3 Thesis Organization and Contribution

The thesis is organized as follows with five chapters and three main research directions.

Chapter 2 is a general introduction of reinforcement learning (RL) algorithms and intrusion detection systems (IDS) research. RL and IDS are two essential topics in this thesis.

Chapter 3 conducts a thorough study of a representative RL algorithm, namely the Proximal Policy Optimization (PPO). We study the properties of PPO in this chapter with Atari games. PPO is well-known for its core idea (clipped surrogate objective) and code-level optimization skills. It has already achieved high performance in many scenarios, including playing Atari games. Through a series of experiments, we hope to figure out what factors inside the algorithm contribute to the high performance of

PPO on playing Atari games.

Chapter 4 investigates the formulation and application of reinforcement learning in building intrusion detection systems at packet-level. In this chapter, we propose a novel embedding approach to encode the network traffic. In this way, we can integrate flow statistics with packet information and convert intrusion detection tasks to image-associated tasks. In addition, we design a reinforcement learning module, incorporating deep neural networks (DNNs) to train the intrusion detection system. The experiments are conducted on an up-to-date dataset DDoS2019 [25].

Chapter 5 conducts improved experiments on the basis of chapter 4 but from another angle. It investigates how to incorporate reinforcement learning in intrusion detection systems at the flow-level. We devise a novel RL algorithm. A feature learning module using Stacked autoencoder (SAE) [26] is designed to reduce the dimension of network data. More importantly, an exploration module is designed to facilitate training. Conditional GAN (CGAN) [27] and ε-greedy policy are employed for the exploration purpose. Meanwhile, CGAN can generate new traffics to help simulate a more realistic network environment. The validation experiments are conducted on both the NSL-KDD [28] and the DDoS2019 dataset.

Finally the thesis is summarized in Chapter 6 with some concluding remarks.

Chapter 2

Review of Preliminaries

2.1 Reinforcement Learning

2.1.1 Introduction

As an important method of machine learning, reinforcement Learning (RL) is based on agents learning strategies to maximize returns or achieve specific goals while exploring and interacting with the environment. The basic framework of reinforcement learning is shown in Figure 2.1. In general, a reinforcement learning framework consists of an agent and the interaction environment. The agent sends actions to the environment, and then the environment moves to a new state and generates rewards, which are sent back to the agent for the next step. The state represents the observation of the environment, and the rewards evaluate the outcome of the current action.

Similar to the controller in a control system, the agent is responsible for taking actions and interacting with the environment. An agent is controlled by its policy. As shown in Table 2.1, the policy function takes the states/observations as the input and generates the action for the agent at the current step. In the actual implementation, deep neural networks (DNNs) are commonly employed to realize the policy function.

Input	**Output**	**Transformation**	**Function**
State	Action	Action = Function (State)	Deep Neural Networks

Table 2.1: The explanation of policy.

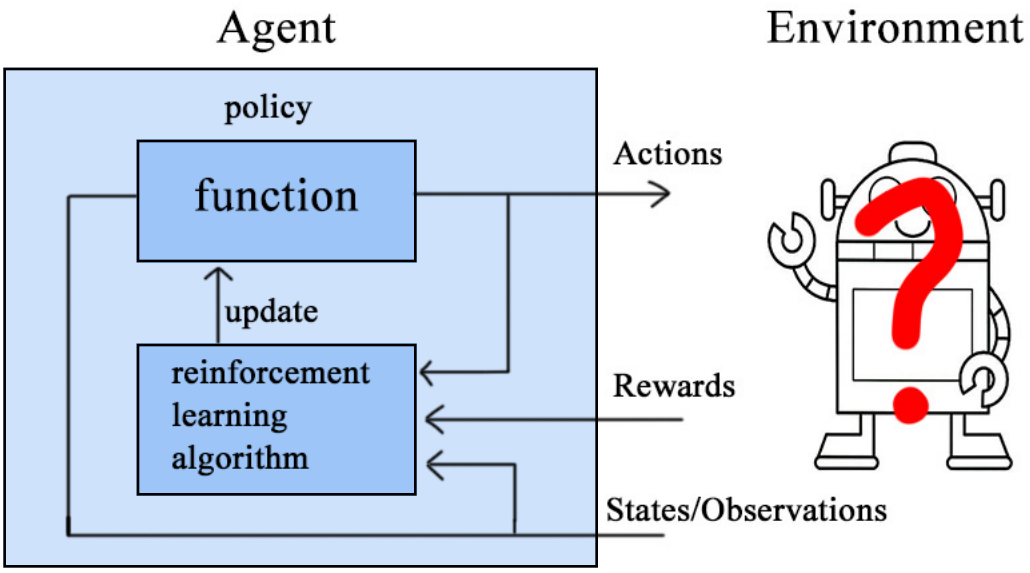


Figure 2.1: Reinforcement Learning Framework. The agent interacts with the environment to collect training data. The environment can be real or simulated. For example, in the unmanned aerial vehicle (UAV) control, an UAV (agent) flies in a simulation environment to collect data.

In many cases, we need to define the reward feedback rules manually. The rewards are used for updating the policy (See section 2.1.5 and 2.1.4). After the policy is updated, it then takes the current state as the input to issue new actions. In short the policy can be considered as a function that maps states to actions, and is subject to changes according to rewards, as shown in Table 2.1. The objective of RL is to learn the optimal policy for the agent to take the best action at each step.

2.1.2 Formulation

When interacting with the environment, an agent always follows a certain behavior pattern during the entire procedure. For example, when running a maze, an agent always goes left, which is a kind of behavior pattern. This kind of behavioral choice is defined by the policy, denoted by π. When the agent takes an action a, the

environment provides the agent with a reward feedback r and moves to a new state s. It is always desirable for the agent to get maximum reward at every step or over the long term. This interaction can be modelled by a Markov process.

In general, reinforcement learning algorithms are applied for solving problems which can be described by the Markov process. Its objective is to learn a policy that the agent can receive maximum reward by following this policy throughout the whole process. Given a Markov process starting from s_0:

$$
\begin{aligned}
& s_0 \xrightarrow{\pi} a_0, \;\; s_0, a_0 \xrightarrow{p} r_0, s_1 \\
& s_1 \xrightarrow{\pi} a_1, \;\; s_1, a_1 \xrightarrow{p} r_1, s_2 \\
& \dots \\
& s_{i-1} \xrightarrow{\pi} a_{i-1}, \;\; s_{i-1}, a_{i-1} \xrightarrow{p} r_{i-1}, s_i \\
& s_i \xrightarrow{\pi} a_i, \;\; s_i, a_i \xrightarrow{p} r_i, s_{i+1} \\
& \dots
\end{aligned}
$$

where s denotes the state, a the action, p the transition function, π the policy, r the reward, and i is the step. The transition (probability) function can be further written as $p(r_i, s_{i+1}|s_i, a_i)$, which is the conditional probability that the environment returns reward r_i and reach the new state s_{i+1} under the current state s_i, after action a_i is taken. However, in real world, the actual transition function is always unknown. We can use sample-based approaches, such as Monte Carlo methods (MC) and Temporal Difference (TD) Learning [8], to approximate the transition function.

As mentioned above, we define the behaviour pattern as the policy, also named the actor $\pi(a|s)$. $\pi(a|s)$ is the conditional probability of taking a under current s. The actor can directly guide the agent to take the optimal action at each step. In addition to the actor, the critic is another mechanism which guides the agent to make the optimal decision. Define the value function $V(s)$ at step i as:

$$V(s_i) = E[G_i|s_i]$$

and the state value function (i.e. Q function) $Q(s, a)$ at step i as:

$$Q(s_i, a_i) = E[G_i | s = s_i, a = a_i]$$

where G_i is the expected discounted accumulative reward at step i, defined as:

$$G_i = r_{i+1} + \gamma r_{i+2} + \gamma^2 r_{i+3} + ... = \sum_{t=0}^{\infty} \gamma^i r_{i+t+1}$$

The value function and the state value function are two types of dominating critics. They can evaluate the quality of the action taken in current state. The value function represents the expected reward that can be obtained beyond the current state. The state value function represents the expected reward that can be obtained after action a is taken in the current state. The discount factor is denoted by γ, and $\gamma \in [0,\ 1]$. By increasing the value of γ, we can force the agent to consider more farsighted states. In continuous tasks which have no terminal states, γ can not be 1. Otherwise, G_t does not converge. Only episodic task is considered in this thesis, which means that there is a terminal state in each episode.

2.1.3 RL Algorithms

In Figure 2.2 some frequently-used reinforcement learning algorithms are shown. As mentioned earlier, we can optimize a policy by directly learning an actor or indirectly learning a critic instead. According to different learning targets, reinforcement learning algorithms can be divided into three main categories: policy-based algorithms, such as Proximal Policy Optimization (PPO) [10] and Trust Region Policy Optimization (TRPO) [18]; value-based algorithms, such as Q-Learning [29]; policy and value based algorithms, such as Actor-Critic (A2C) [30] and Asynchronous Advantage Actor-Critic (A3C) [31]. Policy-based algorithms train an actor which makes decisions directly while value-based algorithms train a critic which can evaluate current decisions. Policy and value based algorithms utilize both the critic and the actor to train the agent.

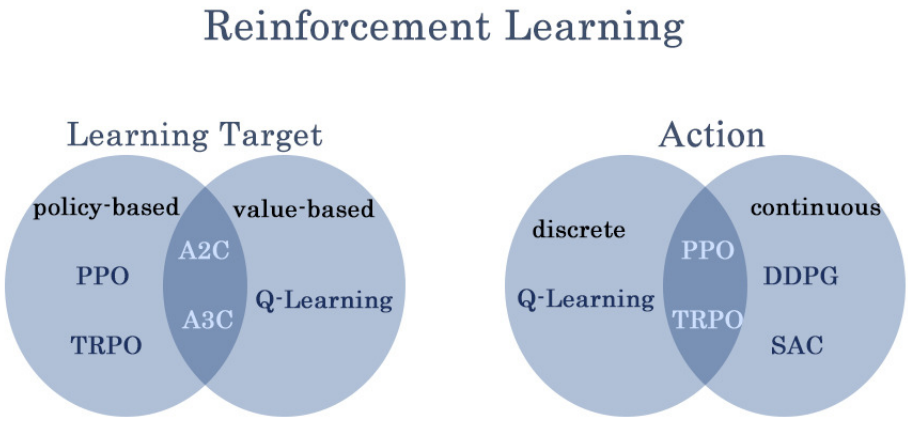


Figure 2.2: Classification of Reinforcement Learning Algorithms

Another important classification basis is the type of actions taken by the agent. In real implementation, actions can be either discrete or continuous. For example, the actions taken in Atari games are transferred to discrete values, but in MuJoCo games [19] actions are transferred to continuous values or vectors. In addition, Q-Learning is employed in the discrete domain. PPO and TRPO are suitable for both discrete domain and continuous domain. Deep Deterministic Policy Gradient (DDPG) [32] and Soft Actor-Critic (SAC) [33] are specifically devised for the continuous domain.

2.1.4 Q-Learning

Q-Learning [29] is a widely used model free[1] value-based reinforcement learning algorithm, and it always works in the discrete domain. In this algorithm, the Q function $Q(s, a)$ works as the critic, which is devised to evaluate the quality of the action taken in the current state. Its value, also called the Q-value, is updated by the following

[1]Model free indicates that model planning is not considered.

equation:

$$Q^{new}(s_i, a_i) \leftarrow \underbrace{Q(s_i, a_i)}_{Current\ Value} + \underbrace{\alpha}_{Learning\ Step} \underbrace{(r_i + \gamma \max_a Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i))}_{TD\ Learning} \quad (2.1)$$

This equation demonstrates the idea of the temporal difference (TD) learning, which relies on the one-step bootstrapping technique [34] to update the values. Once updated, Q values are stored in a tabular form. The Q value is being updated until it stops changing or only changes slightly. After learning the Q values of all existing state and action pairs, we can generate the optimal policy under the following rule:

$$a = \arg\max Q(s, a) \quad (2.2)$$

Deep Q-Learning (DQN) [9] is an advanced version of Q-Learning, where deep neural networks (DNNs) are introduced to estimate Q values. Although Q-Learning is powerful and widely adopted, it can only estimate the Q values of those existing state and action pairs, which have appeared during training. For a problem with a large state space, it is almost impossible to store each value into a tabular form. Incorporating Q-Learning with DNNs can solve the problem. Given a state-action pair as the input, the deep neural network can estimate the Q values. Even if the state-action pair has never appeared in the training stage, the deep neural network can still output an appropriate value. Compared with traditional tabular Q-Learning method, the generality of DQN is superior. DQN has several improved versions, such as Double DQN [35] and Dueling DQN [36].

DQN and its improved versions have been successfully applied in the game control. Researchers [9] have employed a variant of DQN to play seven Atari 2600 from the Arcade Learning Environment. The state is the game screen in the image form and thus convolutional neural networks (CNNs) are adopted to estimate the Q values. Experience replay is proposed to reduce the correlation among the training data. Results show that DQN outperforms almost all previous approaches when playing Atari games, including Sarsa [8] and human experts. However, in many cases, Q

values tend to be overestimated by DQN. To handle this problem, Double DQN [35] is proposed. Studies [35] compare the DQN with the Double DQN on Atari games and find that the Double DQN can generate much more appropriate Q values while DQN always overestimats them, and this is the reason why Double DQN outperforms DQN in most Atari games.

2.1.5 Policy Gradient

Policy gradient method [37] is a type of policy-based reinforcement learning technique, which can be applied in both discrete and continuous domains. Policy gradient methods can directly optimize the parameterized policies π_θ[2] by maximizing the long-term cumulative reward $E_\pi[G_i]$ based on the gradient ascent approach. The policy gradient method has many advantages over other reinforcement learning algorithms. It can make the polices' convergence more greedy over time autonomously. Furthermore, in comparison with Q-Learning and its variants, the policy gradient method can work effectively in continuous control problems. The policy gradient estimator can be expressed in the following form:

$$\hat{g} = \hat{\mathbb{E}}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta \left(a_i \mid s_i \right) \hat{\Psi}_i \right]$$

where π_θ is a parameterized policy with respect to parameter θ, $\hat{\Psi}_i$ is a function related to rewards and $\hat{\mathbb{E}}_{\pi_\theta}$ is the mean over a finite batch of samples collected under π_θ. As shown in Table 2.2, there are several expressions of Ψ_i adopted by different policy-based algorithms. Specially, the advantage function and the TD-error are widely applied. The average function measures how much reward can be gained by taking the action a_i in the current state s_i, compared to the average reward. As shown in Table 2.3, the TD-error calculates the difference between the estimated accumulative reward to be received starting at given time step i and the actual accumulative reward received starting at step i, and the algorithm acts to reduce such error.

[2] π denotes the policy and θ denotes the parameter of the policy. For example, if the policy is deep neural networks (DNNs), θ denotes the parameters of DNNs.

Expression	Explanation
$\sum_{i=0}^{\infty} r_i$	total trajectory reward
$\sum_{i'=i}^{\infty} r_{i'}$	total reward after taking a_i
$\sum_{i'=i}^{\infty} r_{i'} - b$	add a baseline (constant)
$Q^{\pi}(s_i, a_i)$	state-action value function
$Q^{\pi}(s_i, a_i) - V^{\pi}(s_i)$	advantage function
$r_i + V^{\pi}(s_{i+1}) - V^{\pi}_{s_i}$	TD error

Table 2.2: Multiple Expressions of $\hat{\Psi}$

Expression	Explanation
r_i	instant reward at step i
$V^{\pi}(s_{i+1})$	estimated accumulative reward starting at step $i+1$
$r_i + V^{\pi}(s_{i+1})$	actual accumulative reward starting at step i
$V^{\pi}(s_i)$	estimated accumulative reward starting at step i

Table 2.3: TD-error

Several RL algorithms are developed from the policy gradient method, including the well-known Trust Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO), which have already been successfully implemented in many areas. Particularly, PPO has been extensively applied to game AI research and solving control problems. PPO is a type of off-policy[3] reinforcement learning algorithm applicable for both continuous and discrete environments. The core idea of PPO is the clipped surrogate objective. PPO not only attains high scores in test-beds like Atari and MuJoCo [19], but also solves complex problems such as DOTA [38, 39] and Glory of Kings [40]. The well-known StarCraft agent AlphaStar [41] and Mahjong agent Suphx [42] are also trained with PPO. Additionally PPO has also be exploited to solve control problems, such as unmanned aerial vehicles (UAVs) control [15, 16] and

[3]Off policy means that the agent used for interaction and the agent used to update the policy are different. On-policy means that the agent used for interaction and the agent used to update the policy is the same one.

autonomous underwater vehicles (AUVs) control [14].

2.2 Packet-level and Flow-level Intrusion Detection

Packet-level and flow-level analyses are two fundamental methods for designing intrusion detection systems. In the packet-level intrusion detection, messages and headers transported by network packets are primarily extracted for detecting malignant traffics [3, 4]. In the flow-level intrusion detection, the characteristics of the traffic flow, which usually contains numerous packets, are extracted for detecting attacks [5, 6].

SOURCE	**PACKET**	**DESTINATION**
IP 1	Packet 4, Packet 3, Packet 2, Packet 1	IP 2
Port 1	$\longrightarrow$ (one direction)	Port 2
	TCP or UDP Connection	

Table 2.4: An example of a flow. It consists of four packets, which travel from the source IP and port to the destination IP and port. The transportation protocol is TCP or UDP.

Intrusion detection systems at flow-level extract and analyze flow knowledge. Flow knowledge includes the statistics of a flow, such as the number of packets, the duration, the average packet size, etc. Table 2.4 shows an example flow, which contains four packets traveling from the source to the destination. A flow is defined by a 5-tuple knowledge [43]: (source IP, source port, transportation protocol, destination IP and destination port). Many important features can be extracted from a flow. For example, the number of packets in the flow is a useful feature. The work in [5] contains the evidence that some attacks, such as Denial of Service (DoS) and distributed denial of service (DDoS) attacks tend to transmit a large number of packets in a short time. The duration of the flow, the average packet size in the flow and the transportation protocol can also be considered as important features. Recently a great number of datasets have been collected and published for the purpose of flow-level intrusion de-

tection research. NSL-KDD [28] is one of the most well-known datasets in this area. In the NSL-KDD dataset, 41 flow-level features are extracted from TCP or UDP connections. Some of those features are time-based traffic features, and this type of features are usually analysed within a time interval (i.e. a window). For example, by fixing the destination and source host, we test the flow data using a window of 2 seconds' length and store statistical information associated with the protocol and service, etc.

Packet-level intrusion detection systems analyze and extract features from the network packet data through transmissions for intrusion detection. As shown in Table 2.5, a network packet travels across five layers, including an application layer, a transportation layer, a network layer, an Ethernet layer and a physical layer (not considered in the thesis) [2]. A packet structure consists of messages generated in the application layer and three headers generated in the remaining three layers. The content of messages generated in the application layer is different with respect to different protocols. For example, HTTP generates special HTTP request messages. Different headers carry different knowledge. In the transportation layer, TCP and UDP are two most common protocols with TCP and UDP headers as the most common ones in this layer. In the network layer, IP is the most common protocol with the IP header. Similarly, in the Ethernet layer, an Ethernet header is generated. The knowledge carried inside headers is called field (See Chapter 4).

Packet-level and flow-level techniques approach the intrusion detection problem from two different directions. Packet-level research studies the data knowledge carried inside packets but ignores the relationship among different packets. On the contrary, flow-level research focuses on flow knowledge and attempts to capture the relationship among packets, but ignores data transported inside a packet. Both approaches have their own advantages and disadvantages [44], so in our study (in Chapter 4), we attempt to combine the packet-level approach with the flow-level one, and build a more robust intrusion detection system.

Layer	Main Protocol	Packet Structure
Application Layer	FTP HTTP	Messages
Transportation Layer	TCP UDP	TCP/UDP Header+Messages
Network Layer	IP	IP Header+TCP/UDP Header +Messages
Ethernet Layer (Data Link Layer)	Ethernet	Ethernet Header + IP Header + TCP/UDP Header + Messages
Physical Layer	Not Considered	Not Considered

Table 2.5: Network Layers and Packet Structure

Step	Training	Detection
0	Collect Network for Training	Collect Network for Detection
1	Feature Engineering	Feature Extraction
2	Data Preprocessing	Data Preprocessing
3	Train AI Models	Monitored by IDS
4	Apply AI Model as IDS	Evaluate Detection Results

Table 2.6: Intrusion Detection Framework

2.3 ML-Based Intrusion Detection

In recent years, machine learning algorithms are broadly adopted in the studies of cybersecurity [3–6]. Some well-known machine learning algorithms, such as random forest (RF) [45], support vector machine (SVM) [46], and deep neural networks (DNNs), have shown promising results in the area of cybersecurity. In the thesis, we mainly focus on supervised learning based approaches, such as deep neural networks.

Table 2.6 shows the fundamental supervised learning based intrusion detection framework [2]. The entire procedure of intrusion detection can be divided into two stages: training stage and detection stage.

For the training stage, in step 1, feature engineering is first conducted on the network traffics. It is important to design appropriate features of traffics to improve

the performances of intrusion detection systems. In flow-based research, for most datasets available online, certain standard flow-level features have already been given, so the feature engineering step is not mandatory. However for the packet-based research, it is often necessary to conduct feature engineering and extract features from raw network traffics. In this case, the quality of features is the key to the successful application of the intrusion detection system.

In step 2, data preprocessing is necessary for both flow-level and packet-level research. The features always contain three different value types: continuous value, categorical value, and discrete value. In order for the features to be processed by machine learning models, we need to transform the categorical and discrete values into certain suitable formats.

In step 3, we need to select an appropriate machine learning model. This step is of great importance to the final performance of the IDS [2]. With respect to different tasks, we select the most suitable model according to different rules (See Chapter 4). For example, if the input data is image, convolutional neural networks (CNNs) [47] are often the optimal choice. If it is time series data, we can choose the 1D-CNN [48] and the long-Short Term Memory (LSTM) network [49]. Afterward we perform the training of the model and tuning of hyperparameters. Finally in step 4, we apply the optimal model as the intrusion detection system.

For the detection stage, we are able to extract features engineered at the training stage from raw network traffics and conduct data preprocessing on these features. Subsequently, we feed the preprocessed data into the trained intrusion detection system. In order to evaluate the quality of the IDS, we need to adopt proper metrics for the machine learning model, such as the accuracy, precision, recall and F1 measures, etc.

Chapter 3

What Matters in PPO[1]

3.1 Introduction

Reinforcement learning (RL) has made considerable advancements in numerous research and application areas recently. Policy gradient methods, such as REINFORCE [8], Asynchronous Advantage Actor-Critic (A3C), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO) have been widely applied in both discrete and continuous domains. Especially PPO has been developed as one of the most popular algorithms due to its excellent performance and simple implementation.

The core idea of PPO is the clipped surrogate objective. Studies [20–23] indicate that PPO is a fragile algorithm and its high performance actually comes from several auxiliary code-level optimizations, rather than the core idea. These studies are all conducted in the continuous domain. Considering that many complex problems are essentially discrete, a further understanding of these optimizations in the discrete domain would be crucial for applying PPO to more practical problems, such as the intrusion detection.

Moreover, studies [20–23] indicate that code-level optimizations have significant impacts on the final performance of PPO, thus it is natural to figure out what is the

[1]The results in this Chapter are also included in the project report "An Empirical Study of PPO", co-authored with Hongming Zhang, Yan Wang and Xueying Zhang for CMPUT 655 at Univ. of Alberta.

real functionality of the clipped surrogate objective. To our best knowledge, there is insufficient information in the literature studying this problem. Studies [21] investigate this problem in the continuous domain, but the authors draw the conclusion based on the best learning rate. Since the clipped surrogate objective is used to limit update steps, the performance at the best learning rate may not reflect its effects. To further analyse the property of the clipped surrogate objective, we evaluate the performances of PPO over a wide range of learning rates.

In this chapter, we focus on both the code-level optimizations and the clipped surrogate objective. We develop our own PPO implementation using the OpenAI source code [50] to identify all the code-level optimizations. The optimizations are mainly classified into two groups: environmental optimizations and algorithmic optimizations. At first, we conduct comparative experiments to study the overall effects of the two groups. Then we conduct comparative experiments on commonly adopted optimizations to study their effects individually in the Atari environments. Our results indicate that among different optimizations, reward clipping has the most significant effect. We further investigate into the reward clipping and have discovered that its effect is influenced by the reward clipping scale (see Section 3.4.3). To comprehensively understand the core idea of PPO, we compare the performances of PPO with and without the clipped surrogate objective at different learning rates. We find that the clipped surrogate objective makes PPO less sensitive to learning rates.

3.2 Background

Policy gradient becomes a popular optimization method in recent years. It parameterizes the policy π and uses the gradient ascent method to maximize the expected return $\mathbb{E}_{\pi}[R_t]$ under policy π. In general, the policy gradient estimator is given as,

$$\hat{g} = \hat{\mathbb{E}}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta\left(a_i \mid s_i\right) \hat{A}_i\right]$$

where π_θ is a parameterized policy with regard to θ, $\hat{A}_t$ is an estimator of the advantage function and $\hat{\mathbb{E}}_{\pi_\theta}$ is the mean over a finite batch of samples collected under π_θ. $\hat{g}$ can be obtained by differentiating the following objective function:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_{\pi_\theta}[\log \pi_\theta(a_i|s_i)\hat{A}_i].$$

In order to improve the sample efficiency and make the policy update more numerically stable, Trust Region Policy Optimization(TRPO) [18] and Proximal Policy Optimization (PPO) [10] use importance sampling to achieve off-policy learning and use surrogate objectives to obtain monotonic improvement.

More specifically, the importance sampling calculates the average rewards under the target policy π_θ from trajectories generated by following a different policy called the behavior policy $\pi_{\theta_{old}}$. Let $r_i(\theta) = \frac{\pi_\theta(a_i|s_i)}{\pi_{\theta_{old}}(a_i|s_i)}$ denote the importance sampling ratio, we have

$$\hat{\mathbb{E}}_{\pi_\theta}\left[A_i\right] = \hat{\mathbb{E}}_{\pi_{\theta\,old}}\left[\frac{\pi_\theta\left(a_i \mid s_i\right)}{\pi_{\theta_{\text{old}}}\left(a_i \mid s_i\right)}\hat{A}_i\right] = \hat{\mathbb{E}}_{\pi_{\theta\,old}}\left[r_i(\theta)\hat{A}_i\right].$$

Without any constraints, the maximization of $\hat{\mathbb{E}}_{\pi_\theta}\left[A_t\right]$ would lead to exceedingly large update steps, which will jeopardize the monotonic improvement of the target policy when using function approximation. To tackle this problem, Trust Region Policy Optimization(TRPO) [18] maximizes the objective function with a trust region constraint

$$L^{TRPO}(\theta) = \max_\theta \hat{\mathbb{E}}_{\pi_{\theta\,old}}[r_i(\theta)\hat{A}_i - \beta \mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_i), \pi_\theta(\cdot|s_i)]],$$

for some coefficient β. Here, $\pi_{\theta_{old}}$ is the old policy and π_θ is the new policy. The Kullback-Leibler divergence term $\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_i), \pi_\theta(\cdot|s_i)]$ is called the trust region constraint between the new policy and the old policy.

PPO inherits the same idea of TRPO but simplifies the trust region constraint with a clip operator. The clipped surrogate objective introduced by PPO is

$$L^{PPO}(\theta) = \hat{\mathbb{E}}_{\pi_{\theta\,old}}[\min(r_i(\theta)\hat{A}_i, \mathrm{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i)].$$

The clip operator removes the value that is outside the interval $[1-\epsilon, 1+\epsilon]$, and is defined as

$$\text{clip}(r_i(\theta), 1-\epsilon, 1+\epsilon) = \begin{cases} 1-\epsilon, & r_i(\theta) \leq 1-\epsilon \\ 1+\epsilon, & r_i(\theta) \geq 1+\epsilon \\ r_i(\theta), & \text{else} \end{cases},$$

Here $0 < \epsilon < 1$ is the hyperparameter controlling the extent of clipping.

3.3 Optimization Details

Table 3.1 lists the terms of reinforcement learning corresponding to the actual significance of the Atari game. The RL agent aims to receive maximum episode reward. Through the transformation, we can use PPO to train our game agent. In addition to the basic transformation, some optimizations are also applied in PPO for facilitating training. Table 3.2 lists all the Atari environments related optimizations applied in OpenAI implementation [50]. The optimizations which modify the conditions of environments are classified as the environmental optimizations, and the others are classified as the algorithmic optimizations.

RL	**Atari Game**	**Implementation**
State (s)	Game Screen	Image[1]
Action (a)	Game Operation	Discrete Numbers[2]
Reward (r)	Bonus System	Discrete Numbers[3]
Episode	Game Round	$s_0, s_1, ..., s_i, s_{i+1}, ...s_t$ [4]
Agent	Game User	Convolutional Neural Networks

[1] Images are acquired by screenshot.
[2] The operations in the game are discretized in real implementation.
[3] Bonus system is designed by game developers. The reward is represented by discrete numbers.
[4] An episode starts with s_0 (initial game screen) and ends with s_t (terminated game screen).

Table 3.1: Reinforcement Learning for Atari Game Control

Algorithmic	Environmental
value loss clip(×)	No-op Reset(×)
global gradient clip(×)	Max And Skip(×)
orthogonal initialization(×)	Reward Clip(×)
normalized advantage(×)	Episodic Life Reset(×)
Adam learning rate annealing(✓)	Stack Frames(×)
generalized advantage estimation (✓)	Resize Frames(×)
loss function with entropy(✓)	Observation Normalization(×)

[1] Optimizations not mentioned in paper [10] are denoted by (×), otherwise denoted by (✓)

Table 3.2: Optimizations[1] in PPO

3.3.1 Algorithmic Optimizations

The following shows the main elements of the algorithmic optimization (for the OpenAI implementation of PPO):

- **Value Loss Clip:** Original PPO [10] treats the Mean Squared Error (MSE) shown in Eqn. (3.1) as the value loss function.

$$L^V = \left(V_\theta - V_{target}\right)^2. \tag{3.1}$$

  However, the OpenAI implementation [50] uses a clipped MSE:

$$L^V_{\text{clip}} = \max\left[\left(V_\theta - V_{target}\right)^2, \left(\text{clip}\left(V_\theta, V_{\theta_{old}} - \varepsilon, V_{\theta_{old}} + \varepsilon\right) - V_{target}\right)^2\right],$$

  where V_θ is the new value estimate, $V_{\theta_{old}}$ is the old value estimate and V_{target} is target value estimate.

- **Global Gradient Clip:** The maximum of gradients are bounded within 0.5 under l_2-norm for each update to prevent gradient explosion.

- **Orthogonal Initialization:** The network is initialized with the orthogonal initializer [51] instead of the default uniform initializer.

- **Normalized Advantage:** The advantage vector is normalized by subtracting the mean and being divided by the standard deviation of each mini-batch.
- **Adam Learning Rate Annealing:** The learning rate of Adam [52] is linearly annealed during the training process.
- **Generalized Advantage Estimation (GAE):** Generalized advantage estimation [53] is an effective variance reduction method for policy gradient methods. It has been a default method to estimate advantages.
- **Loss Function With Entropy:** An entropy loss term is added to the loss function to keep the agent exploring while learning.

3.3.2 Environmental Optimizations

The following shows the main elements of the environmental optimization (for the Atari game):

- **No-op Reset:** This wrapper[2] takes no actions for a random number of steps in the beginning of each game, producing a randomized initial state.
- **Max And Skip:** This wrapper skips several steps and returns the larger value of the latest two states for each pixel.
- **Reward Clip:** This wrapper returns +1 for positive rewards, -1 for negative rewards and leaves 0 unchanged.
- **Episodic Life Reset:** This wrapper ends the current episode when the agent loses a life.
- **Fire Reset:** This wrapper resets the environment by firstly firing a couple of bullets.

[2]The wrapper denotes the encapsulated programming module which can realize the corresponding functionality.

- **Stack Frames:** This wrapper stacks the latest k frames such that the agent can infer the velocities and directions of the moving objects.
- **Resize Frames:** This wrapper resizes the observations to 84×84.
- **Observation Normalization:** This wrapper normalizes the observations to $[0, 1]$ by dividing 255 for each pixel.

3.4 Experiments on Code-level Optimizations

Studies [20–23] indicate that the great performances of PPO in the Mujoco environments is the consequence of several code-level optimizations. Inspired by their work, we hypothesize that code-level optimizations are essential for PPO to attain high performances in discrete environments as well. We use the OpenAI's PPO code bases [50, 54] to develop our own implementation to reproduce the algorithm of PPO, and we select Alien, a classic Atari video maze game, as our experiment environment.

In order to draw statistically valid conclusions, each experiment is run with 30 different random seeds to reduce random errors. We use the mean score[3] as the performance metric. For accurate comparisons, the 95% confidence interval of each experiment is evaluated on the last-iteration mean scores[4]. Our default hyperparameter settings are consistent with the original PPO paper [10] and given in Table **??**. We define the fully optimized PPO with default hyperparameters as the BASELINE configuration.

[3]Whenever an episode ends, we save the score of this episode into a queue of size 100. When the queue is full, old scores will be replaced. Before each training iteration, we calculate the mean score of the episodes in the queue as the current performance.

[4]Last-iteration mean score is the mean score of the last 100 episode of each run. We have 30 runs for each experiment, so there are 30 last-iteration mean scores for each experiment.

Hyperparameter	Value
Horizon	128
Adam stepsize	2.5e-4 $\times\alpha$
Num. epochs	3
Minibatch size	256
Discount (γ)	0.99
GAE parameter (λ)	0.95
Number of actors	8
Clipping parameter ϵ	0.1 $\times\alpha$
Value coefficient	1
Entropy coefficient	0.01

Table 3.3: PPO hyperparameters used in Atari experiments. α is linearly annealed from 1 to 0 over the course of learning.

3.4.1 Overall Study on Environmental and Algorithmic Optimizations

We first run the BASELINE configuration with 30 random seeds on Alien to reproduce the results presented by the PPO paper [10]. Then we perform comparative studies by 1) running PPO-M: removing both the environmental and the algorithmic optimizations[5], 2) running PPO-M-W: removing only the algorithmic optimizations, to compare the effects of the two groups.

As seen in Figure 3.1(a), the gap between the three configurations are significant. Table 3.4 lists the proportions of performance decline of PPO-M and PPO-M-W compared with BASELINE, which is 70.13% and 26.13% respectively. The results indicate that code-level optimizations, including algorithmic and environmental optimizations, contribute remarkably to the performance of PPO in the discrete domain.

[5]Only optimizations not mentioned in the PPO paper [10] are removed, same below.

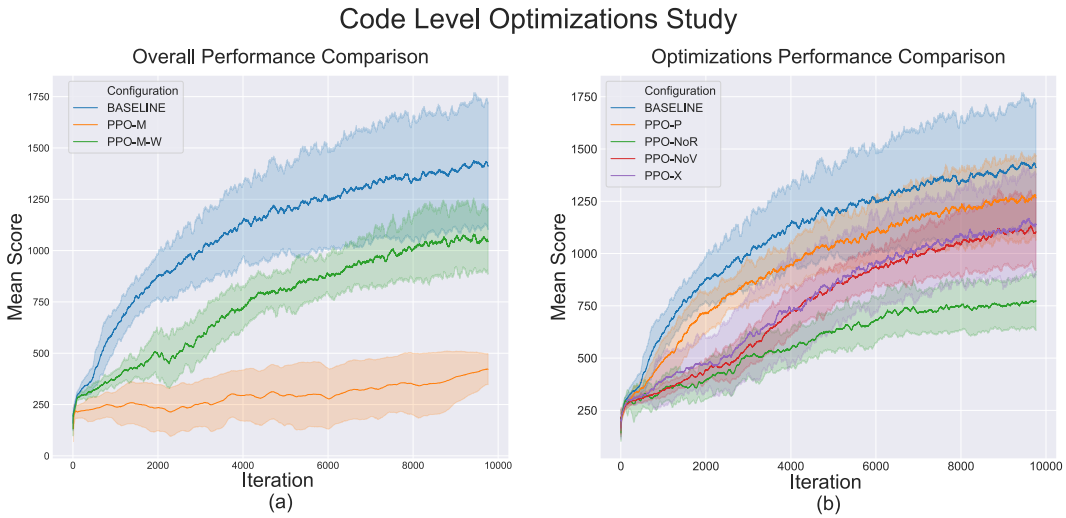


Figure 3.1: All results are averaged over 30 runs with different random seeds. **(a)**: Overall performance comparison among BASELINE, PPO-M, PPO-M-W. **(b)**: Performance comparison among different configurations.

3.4.2 Preliminary Study on Selected Optimizations

Recent studies [20, 21, 55] show that rewarding clipping, GAE [53], value loss clip and orthogonal initialization [51] have significant impacts on the performance of PPO in continuous control problems. To investigate whether these optimizations matters or not in the discrete domain, we perform experiments with the following configurations: **PPO-NoV**: remove value loss clip; **PPO-P**: replace GAE [53] with plain advantage function [17]; **PPO-NoR**: remove reward clipping; **PPO-X**: replace orthogonal initialization [51] with Xavier initialization [56].

Experimental results in Figure 3.1(b) show that PPO-NoV, PPO-P, PPO-NoR and PPO-X all experience severe performance degradation, especially PPO-NoR. Thus, reward clipping, value loss clip, orthogonal initialization and GAE can significantly improve the performance of PPO in Alien. Studies [21] indicate that the value loss clip does not have substantial influence in continuous domain. The discrepancy between our results and that in paper [21] demonstrates that a specific optimization may have different effects in discrete and continuous domains.

As seen in Table 3.4, the performance of PPO-NoR is 45.14% worse than BASE-

Configuration	Last-iteration Mean Score	Difference with BASELINE
BASELINE	**1411.917** [1296.935, 1526.898]	-
PPO-M	**421.79** [393.561, 450.019]	**-70.13%**
PPO-M-W	**1042.968** [966.957, 1118.980]	**-26.13%**
PPO-NoV	**1103.503** [1037.946,1169.061]	**-21.84%**
PPO-NoR	**774.5233** [721.3818, 827.6649]	**-45.14%**
PPO-X	**1137.52** [1004.209,1270.831]	**-19.43%**
PPO-P	**1266.7** [1191.503,1341.897]	**-10.29%**

Table 3.4: Comparison among different configurations at the default learning rate. All results are averaged over 30 runs with different random seeds. The last-iteration mean score with 95% confidence interval is shown in table.

LINE, which reveals that reward clipping is the most significant one among our selected optimizations. In the next section, we perform in-depth studies on the reward clipping.

3.4.3 Study on Reward Clip

Our case studies show that PPO-NoR experiences severe performance degradation compared to BASELINE. Since the score settings of different Atari games are quite different, studies[19] use reward clipping to normalize all positive rewards to 1, all negative rewards to -1, and 0 remains unchanged. However, there is no evidence supporting that '1' is the best reward clipping scale[6]. We assume that the performance gain from reward clipping changes with the clipping scale. To further improve the performance of PPO, and figure out whether the reward clipping scale matters or not, we conduct a series of experiments.

Starting from BASELINE ($x = 1$), the clipping scale is increased and decreased by a factor of 2. We observe that the most frequent positive score returned in Alien is 10. Since it is unreasonable to enlarge the clipped scores beyond the original scores,

[6]We define reward clipping scale to be x ($x > 0$), that is, positive rewards are clipped to be x and negative rewards are clipped to be $-x$, 0 remains unchanged.

Configuration	Learning Rate	Last-iteration Mean Score	Difference
BASELINE	2.5×10^{-4}	**1411.917** [1296.935, 1526.898]	
PPO-0.5R	5.0×10^{-4}	**1630.890** [1517.629, 1744.151]	**+13.42**%
PPO-1.0R	10×10^{-4}	**1437.933** [1304.700, 1571.167]	**+1.84**%
PPO-0.2R	5.0×10^{-4}	**1426.590** [1308.464, 1544.716]	**+1.04%**
PPO-2.0R	10×10^{-4}	**1321.837** [1208.267, 1435.404]	**-6.38%**
PPO-0.1R	2.5×10^{-4}	**1290.427** [1187.946, 1392.907]	**-8.63%**
PPO-4.0R	5.0×10^{-4}	**1077.013** [970.775, 1183.250]	**-23.72%**
PPO-NoR	2.5×10^{-4}	**774.523** [721.382, 827.665]	**-45.14%**

Table 3.5: Last-iteration mean scores are averaged over 30 runs with different random seeds with 95% confidence interval. Difference is compared with BASELINE.

we stop increasing the clipping scales at 10. Correspondingly, we stop decreasing the clipping scales at 0.1. For in-depth studies, a learning rate sweep from 6.25×10^{-5} to 4×10^{-3} is also performed. Starting from the learning rate 2.5×10^{-4} proposed in the original PPO paper [10], we increase and decrease the learning rate by a factor of 2. Other hyperparameters are given in Table **??**.

In Figure 3.2, we plot the mean scores generated from 30 runs at the best learning rate during the learning rate sweep. The best learning rate selection is shown in Figure A.1 in the Appendix. Figure 3.2(a) shows that PPO-0.5R generates the best performance among all the tested scales in our experiments, including PPO-1R whose scale is the same as BASELINE. Figure 3.2(b) suggests that the performance changes with the clipping scale. An inappropriate scale can diminish the effect of reward clipping. In addition, as shown by the mean score curves in Figure 3.2(c), the performance of PPO-0.5R is consistently better than BASELINE and PPO-1R over the entire training iteration. According to Table 3.5, the last-iteration mean score of PPO-0.5R is improved by 13.42% compared to BASELINE and by 11.83% compared to PPO-1R. Our experimental results confirm that the reward clipping scale plays an important role in PPO. It is also shown that the default reward clipping scale is not

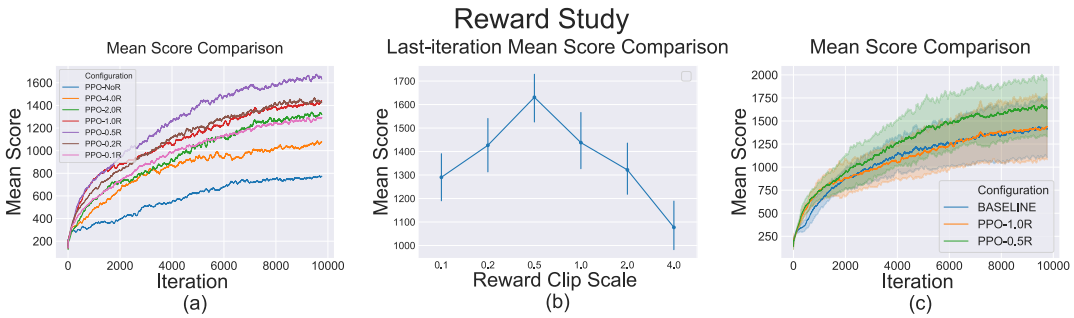


Figure 3.2: All results are averaged over 30 runs with different random seeds. PPO-xR is defined as the configuration with the reward clipping scale x at the best learning rate. **(a)**: Mean score curve comparison among configurations. PPO-0.5R outperforms all other configurations. **(b)**: Bell-shaped curve of the last-iteration mean scores. 0.5 is the best reward clipping scale among our selected scales. **(c)**: Mean score curves of BASELINE, PPO-1R and PPO-0.5R.

optimal. As a result, the performance of PPO can be further improved.

3.5 Experiments on Clipped Surrogate Objective

To explore the properties of the clipped surrogate objective, we run the following experiments: 1) run PPO without the clipped surrogate objective (PPO-NoClip) in the Alien environment with 30 random seeds. 2) run the original fully optimized PPO (PPO) in the Alien environment with 30 random seeds. The 95% confidence intervals are evaluated for the last-iteration mean scores.

This problem is also studied in continuous control environments but their conclusion is drawn at the best learning rate[21]. Since the clipped surrogate objective is used to constrain the size of policy updates [10], we hypothesize that its effect will be more significant at higher learning rates. Therefore, a learning rate sweep is conducted for in-depth comparisons. The sweep range for this section is extended to 1.6×10^{-2}.

Our results are listed in Table 3.6 and Figure 3.4. We find that with relatively small learning rates, the clipped surrogate objective does not provide too much performance gain. The large overlapping confidence band in Figure 3.4 between PPO and PPO-NoClip implies that the PPO does not have statistical advantages over the PPO-

LR	PPO	PPO-NoClip
0.625×10^{-4}	949.283 [907.806, 990.760]	**1009.337** [960.743, 1057.930]
1.25×10^{-4}	1185.063 [1109.453, 1260.674]	**1309.510** [1197.965, 1421.055]
2.5×10^{-4}	1411.917 [1296.935, 1526.898]	**1523.153** [1416.938, 1629.369]
5×10^{-4}	1395.097 [1280.438, 1509.756]	**1578.670** [1421.375, 1735.965]
10×10^{-4}	**1437.933** [1304.700, 1571.167]	994.863 [914.478, 1075.249]
20×10^{-4}	**1403.713** [1295.376, 1512.050]	641.357 [555.116, 727.598]
40×10^{-4}	**947.483** [828.336, 1066.631]	280.247 [209.095, 351.398]
80×10^{-4}	**362.067** [272.284, 451.849]	260.167 [250.221, 270.113]
160×10^{-4}	**262.607** [249.198, 276.016]	250.640 [244.696, 256.584]

Table 3.6: Comparison of last-iteration mean scores with 95% confidence intervals at different learning rates. Results are averaged over 30 runs with different random seeds.

NoClip. This is because, at relatively small learning rates, the policy π_θ is less likely to be updated with large steps astray from the vicinity of the current policy. As demonstrated by Figure 3.3(b), the clip fractions[7] at learning rate 2.5×10^{-4} are more skewed to the left compared to those at learning rate 20×10^{-4}, which indicates that there are fewer large update steps at relatively small learning rates.

On the other hand, as the learning rate increases, the performance gain from the clipped surrogate objective becomes more and more significant, which agrees with our hypothesis. The plateau in Figure 3.3(a) spanning from 2.5×10^{-4} to 20×10^{-4} implies that the clipped surrogate objective plays an important role in stabilizing the performance of PPO over a wide range of learning rates.

However, we also observe that if the learning rate grows beyond 20×10^{-4}, the performance of PPO starts to degrade quickly until all performance gain from the clipped surrogate objective is offset. According to Figure 3.3(a), at learning rate 160×10^{-4}, both the original PPO and the PPO without clipped surrogate objective

[7]Clip fractions are calculated by dividing the total number of steps with the number of clipped steps.

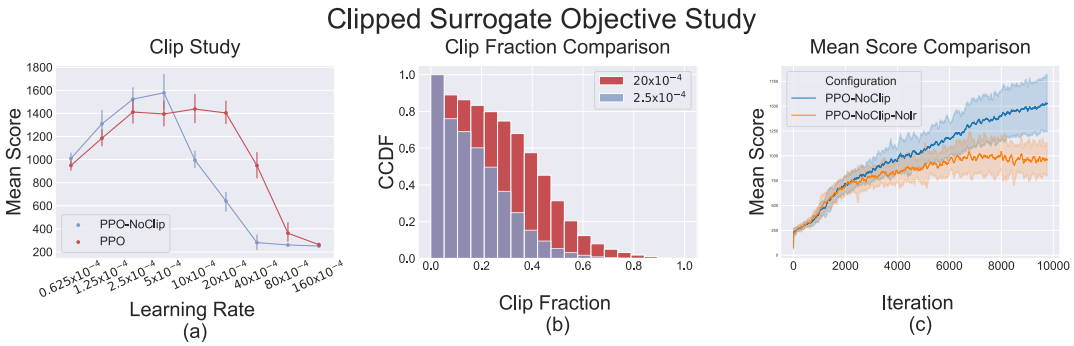


Figure 3.3: All results are averaged over 30 runs with different random seeds. **(a)** The last-iteration mean scores of PPO and PPO-NoClip at different learning rates. **(b)** The Complementary Cumulative Density Functions (CCDF) of clip fractions at learning rates 20×10^{-4} and 2.5×10^{-4}. It shows the distribution of clip fractions of PPO. **(c)** The mean score curves of PPO-NoClip and PPO-NoClip-NoLr at learning rate 2.5×10^{-4}.

do not work properly. This performance degradation can be explained by PPO's defect in limiting the first update step of each training iteration. Studies [21] briefly discuss this defect. Herein we provide a more detailed discussion of the problem.

In PPO, we use the sample average to represent the expectation in the clipped surrogate objective. In general, we compute the gradient of the clipped surrogate objective as:

$$\begin{aligned}\nabla_\theta L^{PPO}(\theta) &= \hat{\mathbb{E}}_{\pi_{\theta_{old}}}[\nabla_\theta \min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \\ &= \begin{cases} \hat{\mathbb{E}}_{\pi_{\theta old}}[\nabla_\theta r_t(\theta)\hat{A}_t], & \text{if } L1 \text{ or } L2, \\ 0, & \text{Otherwise} \end{cases}\end{aligned}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, $L1 : \{r_t(\theta) \leq 1+\epsilon \text{ and } \hat{A}_t > 0\}$ and $L2 : \{r_t(\theta) \geq 1-\epsilon \text{ and } \hat{A}_t < 0\}$. π_θ is the new policy and $\pi_{\theta old}$ is the old policy.

By the clipping steps with $r_t(\theta) \geq 1+\epsilon$ for $\hat{A}_t > 0$ or $r_t(\theta) \leq 1-\epsilon$ for $\hat{A}_t < 0$, the current policy is protected from significant changes in order to guarantee the monotonic improvement [18]. But the above gradient formula tells us that PPO will not limit the update steps for the first update of each iteration. It is because we initialize π_θ to $\pi_{\theta_{old}}$ before the first update so that the ratio $r_t(\theta)$ will be 1 in this case, which may result in an arbitrary large update to the current policy [21]

and disrupt the monotonic improvement of the current policy, especially at relatively large learning rates.

In addition, Figure 3.4 also reveals that with relatively small learning rates, PPO significantly outperforms PPO-NoClip in early iterations but PPO-NoClip catches up and exceeds PPO quickly in late iterations. Considering that the Adam learning rate linearly decreases as the training iteration increases, we hypothesize that the Adam learning rate annealing is one of the reasons that facilitate the faster training of PPO-NoClip in the late iterations.

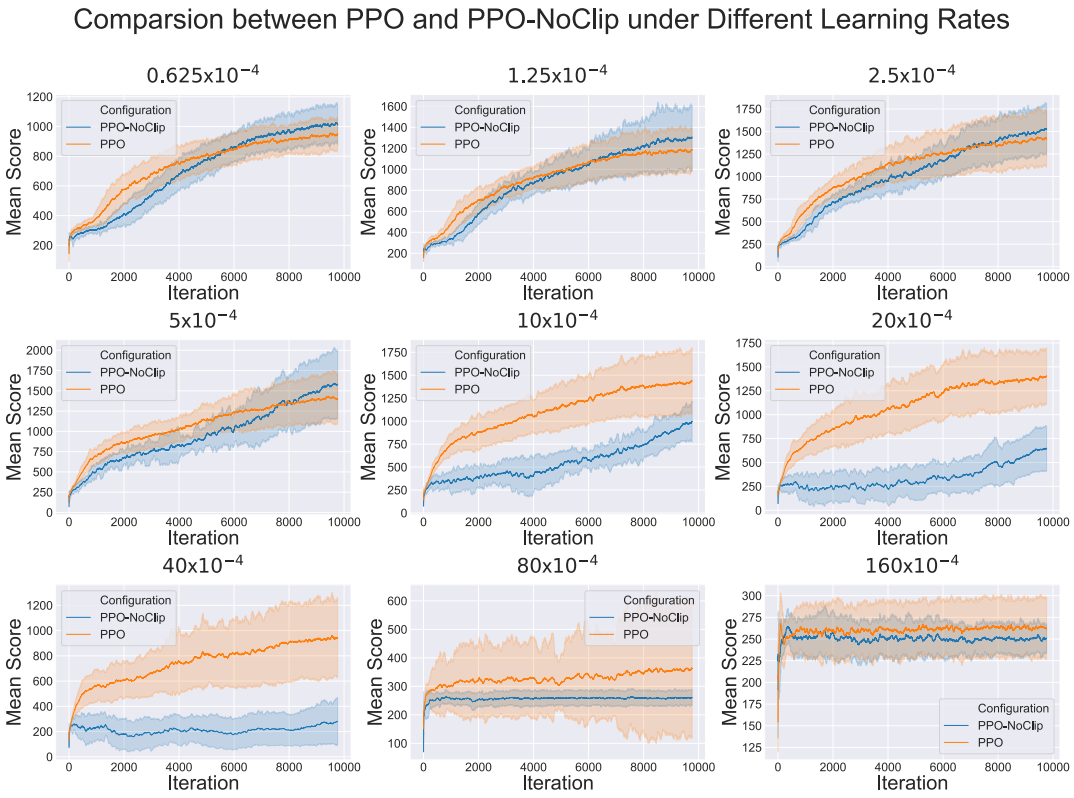


Figure 3.4: All results are averaged over 30 runs with different random seeds. The shade of the curve is the confidence band with 1 std.

To verify our hypothesis, we run PPO-NoClip without Adam learning rate annealing (PPO-NoClip-Nolr) with 30 random seeds in the Alien environment. The mean score curves of PPO-NoClip and PPO-NoClip-Nolr plotted in Figure 3.3(c) confirm our hypothesis. In early iterations, the confidence bands of PPO-NoClip and

PPO-NoClip-Nolr are overlapped so they have similar performance. As the iteration increases, the performance of PPO-NoClip grows steadily while that of PPO-NoClip-Nolr barely grows in late iterations, causing their performance to be statistically different. Our experimental results illustrate that PPO-NoClip is sensitive to learning rates, and Adam learning rate annealing is a significant factor to facilitate its training.

3.6 Conclusion

In this chapter, we first study the effects of code-level optimizations for the PPO algorithm in the discrete domain. Based on the results of our case studies, both the environmental and algorithmic optimizations improve the performance of PPO. Then, we delve into four code-level optimizations: reward clipping, value loss clip, orthogonal initialization [51] and GAE [53]. Our results show that they all play important roles in attaining high scores, especially the reward clipping which gives the best outcome. Our detailed studies on reward clipping demonstrate that its effect changes remarkably with the reward clipping scale.

In addition, we study the properties of the clipped surrogate objective. Results indicates that at relatively small learning rates, the clipped surrogate objective does not have significant impacts on PPO's performance. However, as the learning rate increases, the performance of PPO without clipped surrogate objective declines dramatically while that of the original PPO still remains high. Thus, the clipped surrogate objective stabilizes the performance of PPO over a wider range of learning rates. We also observe that PPO's performance degrades substantially at larger learning rates. This degradation can be explained by PPO's defect in limiting the first update step of each training iteration.

Most importantly, the research conducted in this chapter provides many valuable guiding ideas for the following chapter 4 and 5, where we conduct research on intrusion detection with reinforcement learning. For example, we will attempt to treat the

intrusion detection task as a special game, and adopt the same transformation method to solve the intrusion detection problem with reinforcement learning approaches. In addition, the studies of reward clip are also beneficial to the intrusion detection research, providing insights for designing the reward system. The importance of the reward scale to the reinforcement learning framework will be further studied in devising a proper reward system to facilitate training for the RL.

Chapter 4

Packet-Level Intrusion Detection Based on Reinforcement Learning

4.1 Introduction

In this chapter, we conduct research on intrusion detection at packet-level with reinforcement learning approaches. Packet headers transported by network packets though the transmission will be extracted for training and evaluating the intrusion detection system.

4.1.1 TCP/IP Model and Packet Switches

In this section, we introduce how a packet traverses from the source host to the destination host. Furthermore, important knowledge carried by the packets through the entire transmission trip will be introduced in details.

Figure 4.1 [57] shows a classical TCP/IP network model[1] and the path of a packet traveling from the source host to the destination host. The TCP/IP model consists of an application layer, a transportation layer, a network layer, a data link layer (Ethernet layer) and a physical layer. The physical layer is not considered in the thesis. We explain the procedure of packet transmission [57] from a client side such as a browser.

[1]TCP/IP is a set of communication protocols used to realize network interconnection. Internet network architecture is based on TCP/IP.

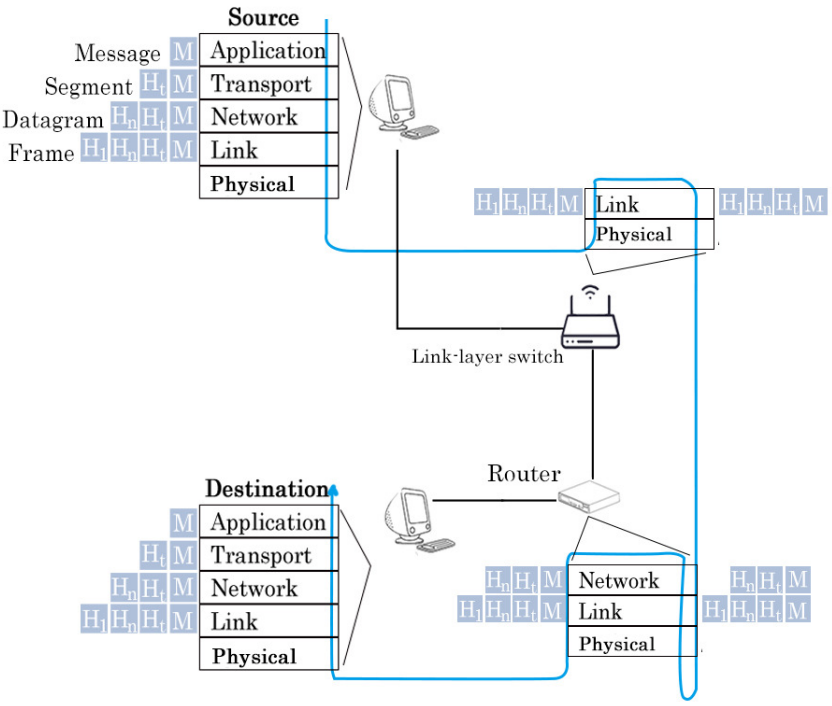


Figure 4.1: TCP/IP Model and Packet Switches. M represents the application messages generated in the application layer. H_t, H_n, H_l represent transportation layer header, network layer header and data link layer header, respectively.

First of all, the client launches an HTTP[2] request in the application layer. The client creates a socket and sends the HTTP request message to the transportation layer through the socket[3].

Both TCP and UDP protocols operate in the transportation layer, and HTTP can launch the TCP connection for transportation. After receiving HTTP messages from the application layer, the transportation layer creates a logic TCP connection with the transportation layer of the server through three-way handshaking[4]. Then, the transportation layer generates a TCP header (H_t), and adds it to the message, composing a TCP segment. The most important knowledge carried in the TCP

[2]Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, and hypermedia information systems. HTTP is the basis of data communication on the World Wide Web.

[3]The socket is an endpoint for two-way communication between application processes on different hosts in the network. It provides a mechanism for application layer processes to exchange data using network protocols.

[4]Three-way handshaking is not required in UDP connections.

header is the source port and destination port. The port identifies the application process, and different ports are bound with different application protocols. In our case, HTTP is bound with the port 80. Subsequently, the TCP segment is sent to the network layer for routing and forwarding.

After receiving the TCP segment from the transportation layer, the network layer generates and adds an IP header (H_n) to the TCP segment, composing the IP datagram. The most important knowledge transported in the IP header is the source IP address and destination IP address. IP address indicates the start and the end network address of a packet trip. Then, the IP datagram is sent to the data link layer. In addition to IP protocol, ICMP and ARP protocols also operate in the network layer.

In most cases, Ethernet protocol is used in the data link layer, so the data link layer is also named the Ethernet layer. After receiving the IP datagram, the data link layer generates and adds an Ethernet header (H_l) to the IP segment, composing the Ethernet frame. Finally the frame is sent to the server-side. Similarly, the procedure of receiving a packet on the server-side is the reverse of sending a packet as described in the above.

In general, we take advantage of the quintuple, i.e. source IP address, destination IP address, source port, destination port, and the transportation protocol, to distinguish between multiple packet transmissions. In the following experiments, we attempt to adopt these properties to split network traffics.

4.1.2 Packet Headers and Fields

As stated previously, in this chapter, the knowledge transported by packet headers will be extracted for the purpose of packet-level intrusion detection.

As shown in Table 2.5 in Chapter 2, the application layer generates application messages, and the remaining three layers generate special headers. A packet header consists of different type of fields to store the most representative knowledge of each

layer. Table 4.1 lists some common headers and fields. Taking the TCP header as an example, a TCP header consists of various types of fields, including source and destination port, sequence number, acknowledgement number, review window, header length, options and flag. The successful operation of TCP protocol requires the support of these filed knowledge [57]. Since fields store the most essential knowledge of headers, it is natural to adopt fields knowledge to establish a stable intrusion detection system. Fields are stored in the binary form in bytes, and in the following study, we use this property to transfer fields into pixels.

Header	**Field**
IP Header	Version Number, Header Length, Type of Service, Datagram Length, Identifier, Flags, Fragmentation Offset, Time-to-Live, Protocol, Header Checksum, Source and Destination IP, Options
TCP Header	Source and Destination Port, Sequence Number, Acknowledgement Number, Receive Window, Header Length, Options, Flag
UDP Header	Source and Destination Port, Length, Checksum
Ethernet Header	Source and Destination Address, Type, CRC, Preamble

Table 4.1: Headers and Fields

Packet headers and application messages are recorded in 'pcap' (packet capture) files, which can be acquired by 'wireshark' [58], a commonly used network analysis tool. In the experiment, after selecting the network card of the network device on 'wireshark', it can automatically capture all the packets (Ethernet frames) traveling through the network card in chronological order and record these packets in 'pcap' files in a hexadecimal format. We can then use 'pyshark' [59] to read packets from 'pcap' files. In our case studies, we adopt the DDoS2019 package [25], which contains raw 'pcap' files for the packet-level IDS research.

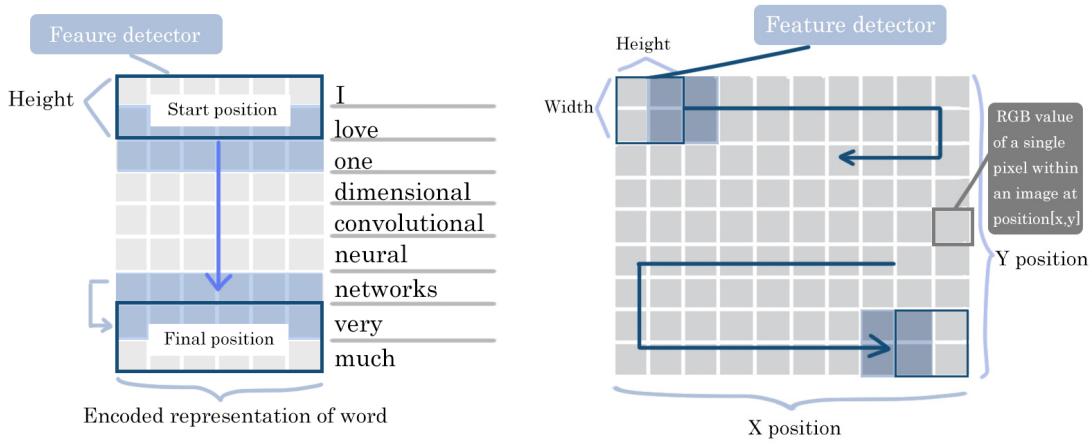
4.1.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) [47] are a type of feed forward neural networks with a deep structure facilitating convolutional computations. It is one of the most representative algorithms of deep learning. CNN is widely applied in the area of computer vision, such as image classification [60, 61]. The name of CNN indicates that the network utilizes the convolution operation extensively. A basic CNN consists of convolutional layers, pooling layers and fully connected layers. CNN gains in popularity because of its affine invariance property, which is achieved by receptive field, shared weights and pooling.

A convolutional layer conducts feature extraction on images with convolution kernels, through which the receptive field and shared weights can be employed. The receptive field signifies that a neuron in the CNN is required to sense information of a part of the image, which can considerably reduces the number of parameters connected to a neuron. This is beneficial considering that the local correlation of an image is relatively stronger. By shared weights, different neurons can share the same convolution kernels, which can further reduce the number of parameters required for training. As shown in Figure 4.2b, a kernel (feature detector) is performing the convolution operation on the image to extract image features. Different kernels can extract different type of features. The output of the convolutional layer is called the feature map. Notably, one kernel generates one feature map.

A pooling layer can reduce the dimensionality of the data. As a general rule, the pooling layer is sandwiched between continuous convolutional layers, and it can compress the amount of data/parameters and prevent over-fitting. In this work, we adopt two pooling methods, the maximum pooling and the average pooling, to perform down-sampling on each feature map.

The other commonly used CNN is 1D-CNN [48]. 1D-CNN is especially efficient when extracting essential features from shorter and fixed-length segments of the over-



(a) 1D-CNN (b) CNN

Figure 4.2: Convolutional Operation Comparison. **(a)**: 1D-CNN convolutional operation example. **(b)**: CNN convolutional operation example.

all data set, and when the feature within the segment is not of high relevance. Accordingly, 1D-CNN is a good choice for analyzing sensor time-series data [62]. There are two main differences between the 1D-CNN and the CNN. First, the input dimensions are different, and second, the convolution kernel traverses the input data in different ways, as shown in Figure 4.2. As displayed in Figure 4.2a, assuming that the input is a sentence, each word can be transferred into a vector by word embedding approaches. We can treat the convolution kernel as the feature detector. The feature detector in 1D-CNN always covers the whole word, and scans from the start position to the final position. For example, in Figure 4.2a, the height of the detector is 2, and it moves from the top to the bottom with the step of 1. With respect to CNN, as shown in Figure 4.2b, the input is a batch of 2D images. Using a square window, the convolution kernel slides both horizontally and vertically across the image. As shown in Figure 4.2b, the feature detector operates in a 2x2 window.

4.1.4 Related Works

Packet-level approaches, combined with machine learning algorithms and deep neural networks, have already been broadly applied in various areas of cybersecurity,

Module	Sub-Module	Function
Data Preprocessing	None	Data Transformation and Feature Engineering
Reinforcement Learning	Interaction	Collect batch data and store in replay buffer[1]
	Training	Train the RL agent with batch data
Anomaly Detection	None	Detect attacks blind to the training module

[1] Replay buffer is used to facilitate training. It stores batch data when agent is interacting with the environment. At training stage, training data is sampled from replay buffer.

Table 4.2: Intrusion Detection Framework at Packet-level

including intrusion detection, and traffic classification, etc.

In studies [3], an LSTM-based IDS is designed for detecting malicious traffics in raw network traffics at packet-level. The fields carried by packet headers are extracted and processed for experiments. They treat a field, such as IP version in the IP header, and TCP source port in the TCP header, as a word. A packet represents a sentence which consists of these fields (words). A novel word embedding approach has been given by researchers in this field. They have produced a dictionary to map these words into integers. In this manner, one can transfer fields knowledge into 64-dimension vectors in the integer format. After implementing data extraction and word embedding, a 3-layer LSTM model with dropout layers is established for detection.

Studies [4] also employ packet-level knowledge for detecting malicious traffics in IoT environment. Similarly, they conduct feature extraction on raw traffic flow. Field information is extracted from separated packets and each field represents a feature. The authors concentrate on header fields, including frame, IP and TCP/UDP associated information and exploit one-hot encoding to encode those categorical features. After implementing feature extraction and data preprocessing, they design a three-layer fully connected neural network with the sparse cross-entropy loss function for multi-class classification.

4.2 Methods and Procedures

In this section, we elaborate the novel intrusion detection framework proposed at packet-level. The whole framework is shown in Table 4.2. The framework consists of two major modules and a number of sub-modules. The preprocessing module is devised for data transformation and feature engineering. The reinforcement learning module is devised for training the intrusion detection system with RL approaches. This module further comprises a training module and an interaction module. An additional anomaly detection module is deployed to detect those attacks which are blind to the training module. We introduce each module in the following sections in details.

Step	**Process (Input $\longrightarrow$ Method $\longrightarrow$ Output)**
1	Raw Network Traffic $\longrightarrow$ Session-based Rule $\longrightarrow$ Separated Sessions
2	Separated Sessions $\longrightarrow$ Image Embedding $\longrightarrow$ Session Images
3	Session Images $\longrightarrow$ Labeling with Log Files $\longrightarrow$ Images and Label
4	Images and Label $\longrightarrow$ Normalization $\longrightarrow$ Applicable Dataset

Table 4.3: The Data Preprocessing Module at Packet-level

4.2.1 The Data Preprocessing Module

Motivated by the extensively used embedding approaches such as word2vec [63], we propose a novel embedding method, namely the image embedding for the RL based IDS developed in this chapter. In this method, several designated transformations are performed to convert the network packets into images.

Table 4.3 show the main steps of the data preprocessing module. In the following, we explain these steps in details.

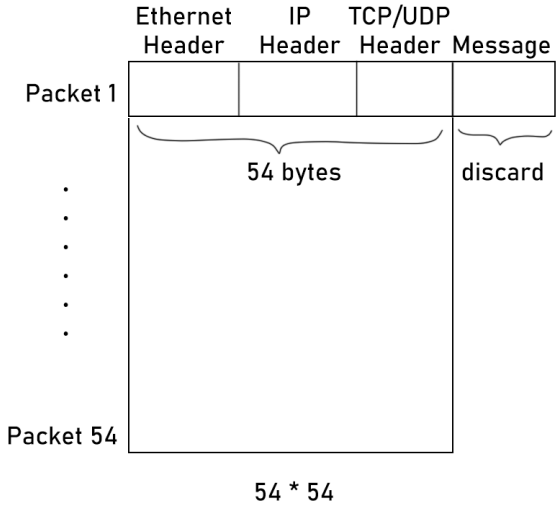
In step 1, we split the vast network traffic recorded in the 'pcap' files into separated and small traffic files according to specific rules, such as the session-based and flow-based partition rules. The difference between a session and a flow is shown in

Source	**Flow I**	**Destination**
IP 1	Packet 4, Packet 3, Packet 2, Packet 1	IP 2
Port 1	$\longrightarrow$ (one direction)	Port 2
TCP or UDP Connection		
Destination	**Flow II**	**Source**
IP 1	Packet 4, Packet 3, Packet 2, Packet 1	IP 2
Port 1	$\longleftarrow$ (one direction)	Port 2
TCP or UDP Connection		
Source/Des.	**Session I**	**Des./Source**
IP 1	Packet 4, Packet 3, Packet 2, Packet 1	IP 2
Port 1	$\Longleftrightarrow$ (bi-direction)	Port 2
TCP or UDP Connection		

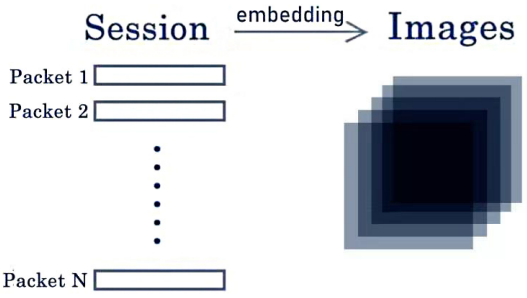
Table 4.4: Session and Flow. Flow I and Flow II both belong to Session I.

Table 4.4. The packet transmission directions in the same session can be opposite, but the directions must be the same in the same flow. The common rule of session-based and flow-based partition is that if two packets share the same 5-tuple knowledge (source IP, source port, destination IP, destination port, transportation protocol), then these two packets will be categorized in the same session or flow. As explained previously, such 5-tuple knowledge determines the travel route of a packet, thus, we can use this rule for flow or session categorization. A session or a flow always contains numerous packets. In our experiments, we use the session-based rule for partition. After partition, we can obtain several separated sessions stored in 'pcap' files; each session contains various packets recorded in the order of capture time.

In step 2, we conduct image embedding on these separated sessions. The structure of a network packet can be found in Figure 2.5. Generally, a network packet consists of an Ethernet header, a TCP or UDP header, an IP header and application messages. We only extract fields stored in packet headers in this work. We discard



(a) Image Embedding I



(b) Image Embedding II. The number of images M is calculated by: $[(N-1)/54]+1$. Operation [] represents rounding down.

Figure 4.3: Procedure of Image Embedding. **(a)**: Transfer packets to an image. **(b)**: Transfer a session to a batch of images.

application messages because the length of a packet header is fixed, while the length of the application messages is not. The fixed embedding length is conducive to image embedding since we can fix the size of the input image. The whole procedure of image embedding is shown in Figure 4.3. The fields transported by the packets are stored in bytes, so they can be converted to base 10 form with a range of 0 to 255, and one byte represents one pixel. Hence one packet represents a line of an image. With simple calculations, we find that the total length of a packet header is 54 bytes. Thus,

we choose the standard image format with a fixed size of 54×54, meaning that an image consists of only 54 packets. Some sessions may contain more than 54 packets, in this case, we can use more images to embed the extra packets. If the number of remaining packets is less than 54, we fill in the missing parts with zeros. Packets are embedded in the order in which they are captured. This way, we can also attach session knowledge into the packet-level experiments.

In step 3, after conducting image embedding, we label each session by matching the time stamp given in the log file provided by the raw dataset.

In step 4, we perform the normalization on these images by dividing 255. All pixels of images are then normalized into $[0, 1]$ from $[0, 255]$.

4.2.2 The Reinforcement Learning Module

Before developing our reinforcement learning module, it is necessary to first transfer the intrusion detection problem to a RL-based problem. For this purpose, we compare the intrusion detection problem to the Atari games studied in chapter 3, by considering the intrusion detection as a special game. The comparison is shown in Table 4.5.

RL	**Atari**	**IDS**	**IDS Space**
State	Image(s)	Image	Images in dataset
Action	Game Operation	Prediction	Label Space {0, 1, 2...}
Reward	Game Feedback	Reward Mechanism	{+1, -1}
Episode	Game Round	Session	-
Agent	Game User	Classifier	-

Table 4.5: Comparison between Atari game and the intrusion detection game.

Sessions identified in an intrusion detection game correspond to game rounds in an Atari Game. When the agent is playing the game, it takes the following trajectory:

$$s_0,\ a_1,\ r_1,\ s_1,\ a_2,\ r_2\ ...\ s_t$$

where s represents the game screen, a represents the game operation and r represents the game reward. The game agent repeats until it reaches the terminated state s_t (game over) of an episode. Similarly, the intrusion detection agent can also take the above trajectory until it reaches the end (last image) of a session.

The state is the game screen (in image format) in the Atari games, and in intrusion detection, the state is also an image after we conduct image embedding on the sessions. The action space of Atari games contains game operations, which can be expressed as discrete numbers, e.g. 0 (one step left), 1 (one step right), 2 (one step up), etc. The action space of the intrusion detection game contains the types of the traffic class, which can also be expressed as discrete numbers like 0 (normal), 1 (attack 1), 2 (attack 2), etc.

However, the reward system is different for the above two problems. The reward mechanism of Atari games has been devised by game developers. For the intrusion detection problem, we need to carefully design the reward system. Referring to the research conducted in Chapter 3, we have found that the reward scale has a significant impact on the performance of reinforcement learning algorithms. Thus, inspired by reward clip for the code-level optimization implemented in the Atari game agent, we design the following reward feedback rule: if the prediction made by the agent is correct, the reward is 1; otherwise, the reward is -1.

In Chapter 3, PPO is the main algorithm used for solving the discrete problem. In this chapter, we apply Deep Q-Learning algorithm, rather than PPO. Deep Q-Learning is also an excellent choice for solving discrete problems, and it is much easier to implement compared to PPO. Later in this chapter, we show in a comparison case study that the Deep Q-Learning outperforms policy gradient methods in the intrusion detection task.

After identifying these essential reinforcement learning ingredients, we start designing our reinforcement learning module. Convolutional neural networks (CNNs) are chosen as the network structure of the intrusion detection agent since the input

	IDS	**Language Model**
Input	Session	Sentence
Element	Packet	Word
Embedding	Image Embedding	Word2vec
Input Format	[batch, len, dim]	[batch, Len, Dim]

Table 4.6: Comparison between the intrusion detection system and the language model. len and Len represent the length of the session and the sentence, respectively; dim and Dim represent the embedding dimension of the packet and word, respectively.

states are images. It should be noted that, contrasting to Atari games and other computer vision tasks, 1D-CNN is also appropriate in our intrusion detection system. One reason is that we treat session images as a type of time-series data since packets embedded in the image are arranged in chronological order according to the capture time. Most importantly, as shown in Table 4.6, the intrusion detection problem also has similarities with the language model, where 1D-CNN is widely applied. Generally, sentences consist of different words and have different lengths. Similarly, a packet can be treated as a word, and a session can be treated as a sentence. When dealing with language tasks, we take advantage of word2vec approaches to conduct data preprocessing, while in our IDS we use the proposed image embedding to conduct data preprocessing for packets. Due to the above, we are motivated to adopt the 1D-CNN in our experiments for the proposed IDS.

The main structure of the two RL agents, i.e. DQN-CNN and DQN-1D-CNN, is shown in Figure 4.4. The input states (s) of the RL agents are a finite batch of images. CNN and 1D-CNN are applied for feature extraction on DQN-CNN and DQN-1D-CNN, respectively. An additional fully connected layer is deployed for final classification and detection. The output is Q-values of the current state s, and the prediction/action (a) is decided based on Eqn. (2.2) with the Q-values.

As discussed in the above, the RL module in the IDS has two sub-modules, the interaction module and the training module. The complete procedure of the interac-

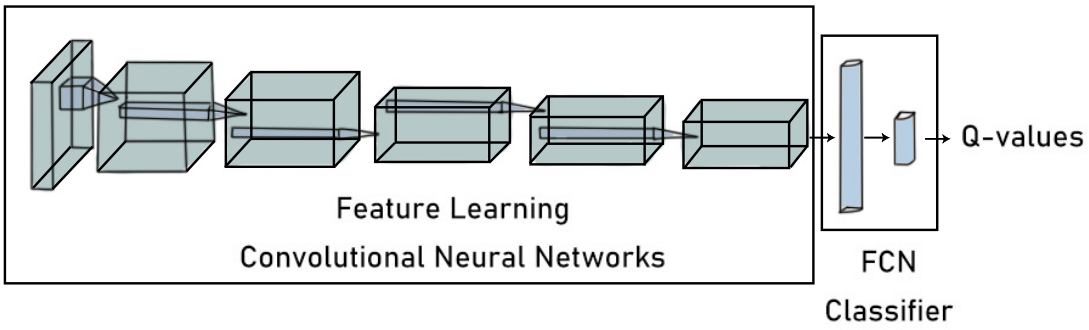


Figure 4.4: Main structure of two reinforcement learning agents DQN-CNN and DQN-1D-CNN. For DQN-CNN, the structure of the feature learning part is CNN. For DQN-1D-CNN, the structure of the feature learning part is 1D-CNN.

tion module is shown in Algorithm 1. In the beginning, we randomly sample a session from the dataset. A session contains numerous images and each image represents a state s. If the current image is not the last image in the session, we store s, r, $s\_$ and a into the replay buffer and continue. If the current image is the last image, we store s, r, and a into the replay buffer and randomly sample another session from the dataset.

The complete procedure of the training module is shown in Figure 4.5. As mentioned above, Deep Q-Learning algorithm is employed in our experiments. Two networks (the Update and the Target) operate together to achieve the approximate regression. The functionality of the target network is to improve the training stability[64] by fixing the regression target in N steps. The structure of the update network is the same as the agent we have introduced above (See Figure 4.4). The target network copies the structure from the update network and is initialized with the parameters of the update network. The update network is updated through back propagation, and the target network is updated by copying the parameters from the update network every N times.

The training module and the interaction module work alternately. For example, once the replay buffer is full, we can start training for several iterations. After training,

Algorithm 1 Interaction Module

```
Start Interacting
for interaction process do
  Randomly sample a session and get its label
  Take first image in the session as current state s
  Feed s into agent and obtain action (prediction)
  Feed a and label into reward mechanism, get reward r
  for each episode do
    if Last image in the session then
      Store (s, r, a, None) into replay buffer
      break
    else
      Take next image in the session as next state s_
      Store (s, r, a, s_) into replay buffer
      Set s = s_
    end if
  end for
end for
```

we can use the new agent to interact with the environment and store the new data into the replay buffer and remove the old data. The complete pack-level IDS algorithm including both the interaction module and training module is shown in Algorithm 2.

4.2.3 The additional Anomaly Detection Module

We deploy an anomaly detection model to detect attacks which are blind to the training set, by considering them as an anomaly class. This is important to a robust intrusion detection system because it is impossible to include all types of attacks in the training set.

As shown in Figure 4.6, we output a confidence score and set a threshold λ manually. In the experiment, we add a Softmax layer at the end of the agent and view the output of the Softmax layer (Q-values) as the confidence score of each class. As shown in Eqn. (4.1), if all confidence scores are smaller than λ, the input will be determined as the 'anomaly' attack. Conversely, the class that belongs to the max

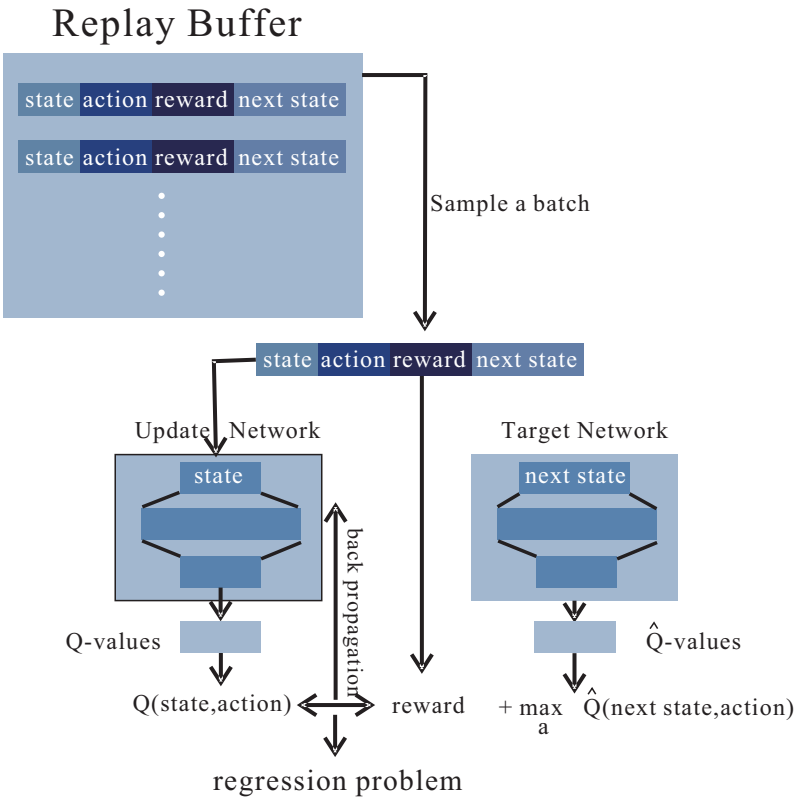


Figure 4.5: Deep Q-Learning Training Module. Update network is updated through back propagation. Target network is initialized with the parameters of update network, and updates are made every N times through copying the parameters from update network. The mean squared error (MSE) is used as the objective function for the approximate regression problem.

Figure 4.6: The work flow of anomaly detection model.

confidence score is the expected detection result.

$$f(s) = \begin{cases} \arg\max_a Q(s,a) \text{ if } \max_{a \in \chi} Q(s,a) > \lambda \\ \text{anomaly if } \max_{a \in \chi} Q(s,a) \leq \lambda \end{cases} \tag{4.1}$$

Algorithm 2 Deep Q-Learning Framework for Detection at Packet-level

Extract sessions from raw traffic file
Split sessions based on session-based rule
Conduct image embedding on each session
Initialize Q-function Q for agent, target Q-function $\hat{Q} = Q$
for each episode: **do**
  Randomly choose a session from the dataset.
  for each image within session, $t \in [0, N]$ **do**
    Given image s_t, take action a_t based on Q
    Compare a_t with true label, obtain reward r_t and $\bar{r}_t$
    Derive next state s_{t+1}: the image behind current state
    Store s_t, a_t, r_t, s_{t+1} into agent replay buffer
  end for
  Sample a batch s_t, a_t, r_t, s_{t+1} from agent replay buffer
  Target $y = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$
  Update Q through back propagation to make $Q(s_t, a_t)$ close to y
  Every C steps reset $\hat{Q} = Q$
end for

4.2.4 Dataset

We select the published DDoS2019 as our major dataset for packet-level experiments. DDoS2019 is a relatively new dataset [25] which is collected for DDoS and packet-level research. The distributed denial of service (DDoS) attack denotes that multiple attackers in different locations simultaneously attack one or more targets, or one attacker takes control of multiple machines in different locations and uses them to attack victims simultaneously. The originating points of the attack are distributed in different places in DDoS attacks, and there can be multiple attackers. DDoS attacks can cause great damages to the society. There are various types of DDoS attacks, posing huge challenges for intrusion detection systems. In such circumstances, DDoS2019 was collected to facilitate the DDoS research.

There are eight traffic types collected in the training set: Normal, PortMap, NetBIOS, LDAP, MSSQL, UDP, UDP-Lag, SYN. These are multiple types of DDoS attacks which have come up frequently in recent years. DDoS2019 is a balanced and sufficient dataset. Each attack contains at least 10000 samples. The test set con-

tains 12 types of traffic. In addition to the eight types collected in the training set, other four types have been included: NTP, DNS, WebDDoS, TFTP. These additional attacks put forward higher requirements to the generality of the intrusion detection system because they are blind to our intrusion detection system at the training stage. For this problem, we deploy an anomaly detection model to detect these four additional attacks, by considering them as an anomaly class.

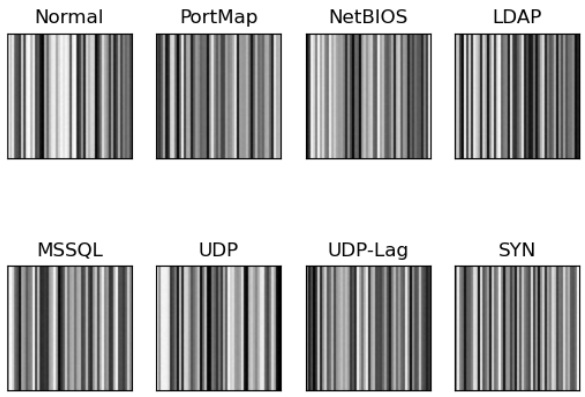


Figure 4.7: Images after image embedding for each class.

Traffic Class	Session	Max Session Len	Average Session Len	Image
Normal (0)	20000	786	71	31256
PortMap (1)	20000	687	54	25258
NetBIOS (2)	20000	556	69	29564
LDAP (3)	20000	456	34	23102
MSSQL (4)	20000	632	56	27695
UDP (5)	20000	346	53	25210
UDP-Lag (6)	20000	452	43	24750
SYN (7)	20000	563	60	27610

Table 4.7: Dataset Details

4.3 Results and Discussions

4.3.1 Evaluation Metrics

We adopt four metrics, including the accuracy, precision, recall and F1 score to evaluate the intrusion detection system.

Predicted	**Actual**	
	Positive	**Negative**
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)

Table 4.8: Confusion Matrix

Metric	**Calculation**
Accuracy	(TP+TN)/(TP+FP+FN+TN)
Precision	TP/(TP+FP)
Recall	TP/(TP+FN)
F1	(2*Recall*Precision)/(Recall+Precision)

Table 4.9: Evaluation Metrics

The definition of these four metrics are shown in Table 4.8 and Table 4.9. These evaluation metrics are regularly used in two-class classification problems. As mentioned above, we have more than two categories in our experiments. When calculating TP, FP, FN and TN with respect to each traffic class, we consider the current class as the positive one, and the remaining classes as the negative one. With the assistance of these machine learning metrics, we can comprehensively evaluate our intrusion detection system.

4.3.2 Experiment Results

We use Deep Q-Learning algorithm combined with CNN and 1D-CNN for our IDS experiments. In our IDS, we devise two agents based on DQN-CNN and DQN-1D-

CNN. As a comparison, we devise another two agents PG-CNN and PG-1D-CNN, which are trained with policy gradient methods. The sessions are extracted from raw network traffic data in 'pcap' files and split according to the session-based rule. The dataset is split in a ratio of 4:1 (16000 sessions for training and 4000 sessions for validation). During the partition, we ensure that the session length distribution of the training set and validation set is consistent, thus the image ratio can also reach approximately 4:1. After partition, the statistics of the dataset is shown in Table 4.10.

Traffic Class	**Training Images**	**Validation Images**
Normal	25012	6256
PortMap	20200	5058
NetBIOS	23652	5912
LDAP	18019	5083
MSSQL	22986	4709
UDP	19159	6051
UDP-Lag	19552	5198
SYN	22640	4970

Table 4.10: Training and Validation Set Details

Experiments are conducted on the computer configured as RTX2070, i5-9600k and 32GB Memory. We compare three discount values γ, including 0.1, 0.5 and 0.9 on four agents. We use the value of accuracy averaged over each class on validation set to evaluate the performance and select the optimal discount value. The experiment results are shown in Table 4.11. We notice that when γ equals to 0.1, all of the four agents attains the highest performance. With regard to reinforcement learning algorithms, DQN-1D-CNN slightly outperforms PG-1D-CNN over all discount factors, and DQN-CNN also marginally outperforms PG-CNN over all discount factors. Meanwhile, it is observed that PG-based agents are more sensitive to discount factors.

Overall results indicate that Deep Q-Learning is a relatively better algorithm than policy gradient methods in our experiments. Hence in the following case studies, we assume that the discount value is fixed as 0.1, and mainly adopt Deep Q-Learning algorithms for the agent.

Discount Factor	**Val Acc (DQN-1D-CNN)**	**Val Acc (DQN-CNN)**
0.1*	**98.78%**	**96.07%**
0.5	95.21%	93.22%
0.9	87.12%	86.24%
Discount Factor	**Val Acc (PG-1D-CNN)**	**Val Acc (PG-CNN)**
0.1	96.17%	96.03%
0.5	90.15%	87.12%
0.9	79.68%	70.69%

Table 4.11: Performances of different discount values on four agents

Detailed experiment results of DQN-1D-CNN with discount value γ equal to 0.1 are shown in Table 4.12. As shown in the table, our detection system can reach high accuracy at 98.78%. With respect to each class, 100% of normal traffic can be detected. The detection rate (Recall) of MSSQL is the least among all eight types, but still reaching 97.30%. Table 4.13 lists the validation results of DQN-CNN. It is seen that the performance of DQN-CNN is slightly worse than that of DQN-1D-CNN, but still reach a high accuracy at 96.07%. The experimental results prove that both DQN-1D-CNN and DQN-CNN can achieve high performances when dealing with image tasks. In addition, in our experiments, the image can also be readily treated as the time series data, and DQN-1D-CNN can perform much better in this case.

Afterwards we evaluate our DQN-1D-CNN agent on the test set. In this stage, we are required to set another important hyperparameter: λ. λ controls the detection performance of (unknown) anomaly traffic. We treat the known eight types of attacks as the negative class, and the anomaly traffic as the positive class. We use precision

and recall to evaluate the performances and select the optimal λ. Results are shown in Table 4.15. As λ is increasing, the precision is decreasing, which means that an increasing volume of network traffic is classified as the anomaly type. Conversely, as λ is increasing, the recall is also increasing, meaning that an increasing amount of anomaly traffic is detected. We can determine the value of λ based on actual requirements of the intrusion detection system. In our experiments, we fix the λ as 0.7, which is a trade-off between 0.5 and 0.9.

Finally, we evaluate our intrusion detection system on the test set with $\gamma = 0.1$ and $\lambda = 0.7$. Results are shown in Table 4.16. Due to the existence of anomaly traffic, the general accuracy has declined greatly to 84.27% compared with the validation accuracy at 96.07%. With respect to all classes, the DQN-1D-CNN agent can attain relatively high performances on those known types. With regard to anomaly types, although these anomaly types are completely blind to our system, 71.05% of them can still be detected.

Traffic Class	**Accuracy**[1]	**Precision**	**Recall**	**F1 Score**
Normal	98.78%	99.32%	100%	99.66%
PortMap	98.78%	98.04%	98.95%	98.49%
NetBIOS	98.78%	98.87%	99.26%	99.06%
LDAP	98.78%	98.85%	98.37%	98.61%
MSSQL	98.78%	98.47%	97.30%	97.86%
UDP	98.78%	98.95%	99.62%	99.28%
UDP-Lag	98.78%	98.02%	98.31%	98.61%
SYN	98.78%	98.64%	97.83%	98.23%
Average	**98.78%**	**98.65%**	**98.71%**	**98.73%**

[1] According to the calculation formula displayed in Table 4.9, accuracy is the general metric of a intrusion detection system, so the accuracy value of each category is the same, and it represents the overall accuracy of the system.

Table 4.12: Validation results of DQN-1D-CNN with discount value $\gamma = 0.1$. The computation time for detection is **0.112s**.

Traffic Class	Accuracy	Precision	Recall	F1 Score
Normal	96.07%	96.91%	99.68%	98.27%
PortMap	96.07%	95.67%	93.53%	94.59%
NetBIOS	96.07%	97.35%	95.75%	96.55%
LDAP	96.07%	95.76%	96.95%	96.35%
MSSQL	96.07%	95.06%	96.03%	95.54%
UDP	96.07%	95.32%	96.00%	95.66%
UDP-Lag	96.07%	95.05%	94.92%	94.99%
SYN	96.07%	97.14%	94.87%	95.99%
Average	**96.07%**	**96.03%**	**95.97%**	**95.99%**

Table 4.13: Validation results of DQN-CNN with discount value $\gamma = 0.1$. The computation time for detection is **0.071s**.

4.3.3 Comparison with Deep Learning Approaches

In this section, we compare the performance of our approach with traditional 1D-CNN and CNN approaches (without the reinforcement learning framework). We evaluate the performances using two metrics. The first one is the detection rate (Recall), which measures how many attacks are correctly detected. The other one is the false alarm rate, which measures how many normal traffics are classified as malignant types. Anomaly attacks are excluded in this experiment.

Results are shown in Table 4.17. It can be seen that traditional deep learning methods can also achieve relatively high performances on the test set, which supports that image embedding is a useful data preprocessing approach for network traffic analysis. Most importantly, the proposed RL-based approach still outperforms these traditional approaches in both detection rate and false alarm rate Without increasing the computation time.

Traffic Class	Session	Max Session Len	Average Session Len	Image
Normal	5000	691	65	9514
Portmap	5000	578	59	8768
NetBIOS	5000	612	54	6978
LADP	5000	531	41	6013
MSSQL	5000	598	61	9143
UDP	5000	408	46	6784
UDP-Lag	5000	425	39	6102
SYN	5000	601	55	7035
Anomaly	5000*4	752	57	29260

Table 4.14: Test Set Details

λ_1	Precision	Recall	λ_2	Precision	Recall	λ_3	Precision	Recall
0.5	93.94%	19.17%	**0.7***	85.13%	71.05%	0.9	43.21%	86.86%

Table 4.15: Anomaly Detection λ Selection

Traffic Class	Accuracy	Precision	Recall	F1 Score
Normal	84.27%	88.81%	93.64%	91.16%
PortMap	84.27%	86.83%	91.38%	89.05%
NetBIOS	84.27%	84.62%	90.69%	87.55%
LDAP	84.27%	82.36%	88.31%	85.23%
MSSQL	84.27%	88.24%	92.14%	90.14%
UDP	84.27%	83.55%	89.40%	86.38%
UDP-Lag	84.27%	82.07%	87.92%	84.90%
SYN	84.27%	84.67%	89.57%	87.05%
Anomaly	84.27%	81.12%	71.05%	75.75%
Average	**84.27%**	**84.70%**	**88.23%**	**86.36%**

Table 4.16: Test results of DQN-1D-CNN with discount value $\gamma = 0.1$ and $\lambda = 0.7$. The computation time for detection is **0.143s**.

Approach	Detection Rate	False Alarm	Detection Time
1D-CNN	92.12%	5.12%	0.096s
CNN	90.68%	6.44%	0.084s
DQN-1D-CNN*	**97.69%**	**0.12%**	0.101s
DQN-CNN	95.14%	3.50%	**0.079s**

Table 4.17: Comparison with deep learning approaches (anomaly types in the test set are excluded)

4.4 Conclusion

In this chapter, we conduct an investigation on intrusion detection at packet-level with raw traffic files provided by DDoS2019, and in devising the intrusion detection system, we also incorporate some ideas from the flow-based research, which will be discussed in details in Chapter 5.

First of all, we design a novel image embedding approach. By means of image embedding, we can transfer raw traffic files, which are generally difficult to process by artificial intelligence techniques, to images. This is significant because there are many artificial intelligence techniques which can achieve high performance on image tasks. In addition, those packets classified in the same session are arranged in the chronic order of the capture time. This way we can extend the the image-based task to the time-series based task, where supplementary AI techniques such as 1D-CNN and LSTM, can also be applied.

Secondly, we use Markov process to model the dynamic process of network session. Therefore, we can devise a reinforcement learning framework to train the system. By introducing discount value γ, we can make the agent to consider more farsighted information. This property is also important to the intrusion detection problem. Assuming that we encounter a malignant flow, if we force agent to consider more farsighted, the agent can detect the intrusion earlier.

Thirdly, we design an additional anomaly detection system to detect those un-

known attacks, and propose the idea of the confidence score. Owing to this anomaly detection module, we are able to detect some unknown and novel attacks. It is crucial because the real network environment is complicated and ever-changed, and hackers will continue to launch novel attacks which are blind to intrusion detection systems.

Our experimental results show that, with the assistance of CNN, 1D-CNN and reinforcement learning algorithms, the intrusion detection system can attain high detection rate and maintain low false alarm rate.

4.5 Future Work

More research works can be done in the future on the basis of our reinforcement learning framework.

Firstly, we can introduce GAN into the reinforcement learning framework. Since we have already transformed the traffic data into images, we can use GAN to generate some novel flows or sessions. Furthermore, we can use GAN to simulate a dynamic network environment for interaction.

Secondly, we can devise a more robust and high-accuracy anomaly detection system. In our experiments, we treat the Q-value as the confidence score. Nevertheless, it is difficult to adjust the value of λ. One possibility is to train the agent to learn the confidence score by itself.

Thirdly, we can introduce the idea of exploration into our reinforcement learning framework. Some exploration approaches, such as ε-greedy policy, are widely adopted in RL. Incorporating a simulation interaction environment, we can attempt to employ the exploration policy to capture more states in the interaction space.

Last but not the least, we may consider locating the malignant packet. In our experiments, the intrusion detection system can detect the malignant image, but can not identify its specific position. One possible option is to employ a N-to-N LSTM model to solve the problem.

It should be noted that several of the aforementioned research ideas are investigated

in Chapter 5, the RL based flow-level intrusion detection.

Chapter 5

Flow-Level Intrusion Detection Based on Reinforcement Learning

5.1 Introduction and Preliminaries

In this chapter, we conduct the research on flow-level intrusion detection with reinforcement learning approaches. We design a novel RL based flow-level intrusion detection framework. In this framework, a sample agent is specially devised for the purpose of adversarial training, and the conditional GAN (CGAN) and ε-greedy policy are employed for data augmentation and state exploration. Then we evaluate our framework on two data sets, the NSL-KDD and DDoS2019.

5.1.1 Generative Adversarial Networks

The generative adversarial network (GAN) is a novel deep learning model [65], and one of the most promising unsupervised learning methods for data with complex distributions. GAN has achieved great success in data augmentation, image generation and feature extraction.

In the framework of GAN, outputs are generated through the mutual game learning of at least two modules: the generator (G) and the discriminator (D). In the original GAN theory, G and D are not necessarily built by neural networks. Functions that can fit the corresponding generation and discrimination rules are also applicable. But in practice, deep neural networks are generally used as G and D. Thus, in this work,

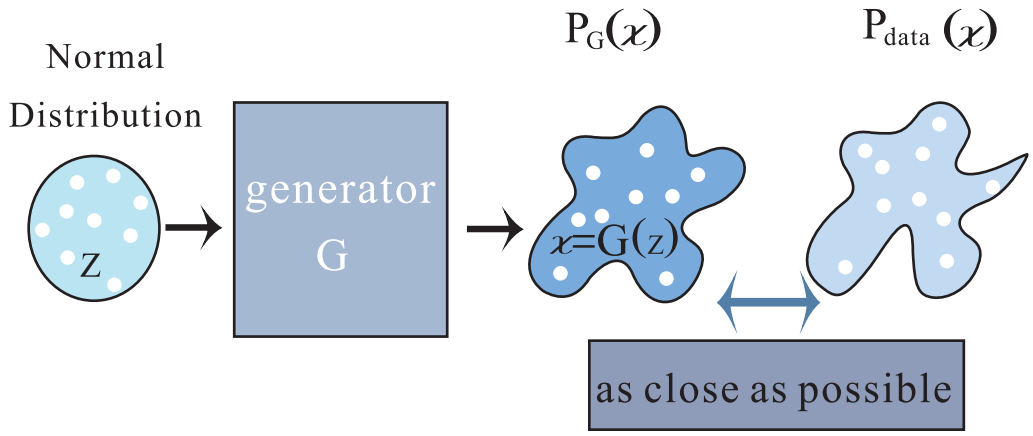


Figure 5.1: The Generator Work Flow. Normal distribution is denoted by P_z.

we consider G and D as two neural networks.

$$\underset{G}{Min}\underset{D}{Max}\ V(D,G) = E_{x\sim P_{data}(x)}[logD(x)] + E_{z\sim P_z(z)}[log(1-D(G(z)))] \tag{5.1}$$

The objective function of GAN is shown in Eqn. 5.1. The generator G and the discriminator D are trained alternately via the above min-max problem. The generator defines a probability distribution P_G, as shown in Figure 5.1. It generates fake samples with the input sampled from the normal distribution or other simple and known distributions. The goal of the generator is to transfer the normal distribution (or other known distributions) to P_G, which is as close as possible to the distribution of data, P_{data}. In this fashion, we can sample data from known distributions to implement the desired data augmentation. The generator aims to solve the following optimization problem:

$$G^* = \arg\min_G Div(P_G, P_{data})$$

where $Div(P_G, P_{data})$ stands for the divergence between distributions P_G and P_{data}, which can be measured by different methods.

From the discriminator's perspective, its function is to distinguish between true samples from the real distribution and fake samples produced by the generator. Essentially, the discriminator is a binary classifier. The objective function for the dis-

criminator is:

$$V(G,D) = E_{x\sim P_{data}}[logD(x)] + E_{x\sim P_G}[log(1-D(x))]$$

where G is fixed. The discriminator aims to solve the optimization problem:

$$D^* = \arg\max_D V(D,G)$$

In fact, the maximum objective value of $V(D,G)$ is related to Jensen-Shannon (JS) divergence, thus we can use it to replace $Div(P_G, P_{data})$ and obtain:

$$G^* = \underset{G}{Min}\underset{D}{Max}\ V(D,G)$$

Thus, by training G and D alternately, we can obtain a robust generative model.

The original GAN is known to have several disadvantages. It is unstable and difficult to converge through training [66]. On the basis of the original GAN, WGAN [66] and WGAN-GP [67] are proposed. These two versions of GAN employ Wasserstein divergence [68] to replace Jensen-Shannon divergence [69]. Further, for the stability through training, on the basis of WGAN, WGAN-GP utilizes gradient clip to limit the update. In comparison to GAN, WGAN and WGAN-GP tend to be more stable during the training process. Thus, WGAN and WGAN-GP are frequently used substitutes for GAN.

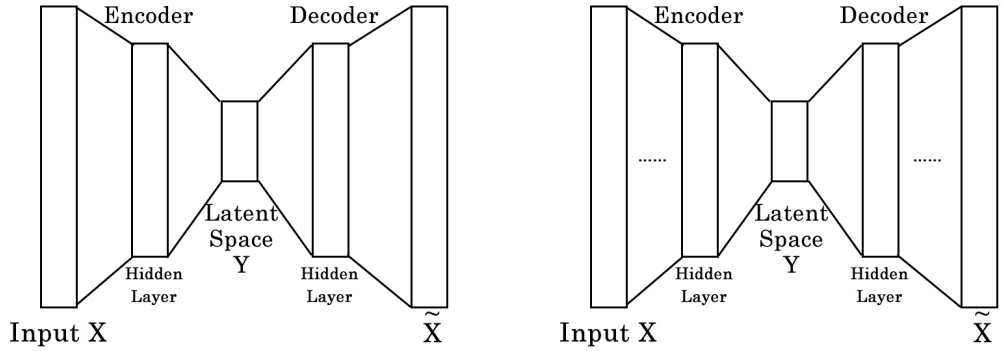
The impact of GAN in the area of computer vision has already been discovered. StyleGAN [70] has been developed as the best face image generator. In this work, a style-based generator is proposed to generate face images, and it is able to control the high-level attributes of the generated image, such as hairstyles, freckles, etc. StyleGAN has following innovations. Firstly, the style-based generator utilizes a non-linear mapping network to replace the traditional input layer. Secondly, researchers adopt mixing regularization to control the style generation. Thirdly, researchers use Gaussian noise as the input to implement stochastic variation to mimic randomly different properties of human faces. The generated images attain high scores on several evaluation criteria.

In addition to the well-known applications in the area of computer vision, many researchers have been exploring the potential of GAN in the area of cybersecurity. Researchers in [71] apply GAN to generate flow-based network traffic. The generated traffic contains all typical attributes of network traffic. They also propose a novel embedding approach, namely IP2Vec to assist GAN in solving other tasks than just the continuous values problem. The IP2Vec approach can learn good representations of categorical attributes. By using IP2Vec embedding, those IP addresses that occur frequently in similar contexts are mapped close to one another in the feature space. Basically, IP2Vec is a fully connected neural network with a single hidden layer. They use all IP addresses, destination ports and transport protocol that appear in the dataset to define an input vocabulary. One-hot encoding is applied to represent these values. After training the IP2Vec, the weights of the m-dimensional hidden layer can be extracted. These one-hot vectors can be transferred into continuous m-dimension vectors. After transformation, they use WGAN-GP to generate the network traffic.

5.1.2 Stacked Autoencoder

Stacked autoencoder (SAE) is a stacked-version of auto encoders [26]. An autoencoder (AE) is an unsupervised artificial neural network. As shown in Figure 5.2a, its functionality is to perform representation learning on the input by taking the input itself as the learning target. Autoencoder has already gained enormous interests in the areas of dimension reduction and anomaly detection.

A typical AE consists of an encoder and a decoder, which are symmetrical. As shown in Figure 5.2a, the encoder compresses the input X to the latent space Y, and the decoder then attempts to reconstruct $\widetilde{X}$ from Y. The objective of an AE is to minimize the reconstruction error $||X - \widetilde{X}||$ through backward propagation (BP) [72]. Through the iterative training, $\widetilde{X}$ is forced to eventually converge to X. This process ensures that Y retains the most significant information of X, thus one can



(a) Auto Encoder (b) Stacked Autoencoder (AE)

Figure 5.2: Autoencoder (AE) versus Stacked Autoencoder (SAE) **(a)**: Auto encoder only has one hidden layer. **(b)**: Stacked autoencoder can have many hidden layers (denoted by ..... in the figure).

use Y to replace the raw input X, which not only reduces the burden of the neural network, but also achieves dimension reduction.

The stacked autoencoder adds a number of hidden layers into the encoder and decoder, as shown in 5.2b. By adding these hidden layers, SAE can handle more complex tasks and learn better representations.

AE and SAE have also been applied to build intrusion detection systems. In [73], the sparse AE, where sparsity constraints are applied to the weight matrix, is incorporated with the support vector machine (SVM) for intrusion detection. The authors establish a single-layer SAE for feature learning and add sparsity constraints on the SAE model. Subsequently, the features extracted from the SAE model are sent to the SVM classifier with Radial Basis Function (RBF) kernel. Another SAE-based research [74] proposes a semi-supervised and unsupervised framework SU-IDS for flow level intrusion detection. The core innovation of SU-IDS is to incorporate clustering and classification approaches with the AE. Likewise, they make use of AE for learning the representative features. At first they pre-train an AE with unlabeled data, then

the features extracted by the AE are fed into the clustering or classification module. The parameters of the clustering/classification networks are shared with the encoder layer in the pre-trained AE.

5.1.3 Bayesian Search

Bayesian search [75] adopts Bayesian Optimization [75] to address the automatic hyperparameter search problem. It gains popularity in exploring optimal hyperparameters because of its high efficiency.

The hyperparameter search problem can be expressed as an optimization problem. Given a function $f : x \to R$, the following optimization problem needs to be solved:

$$x^* = \arg\min_{x \in X} f(x)$$

When f is a convex function and the domain X is also convex, convex optimization solutions can be obtained. However, f may not be convex in complex problems, and it is often a black-box function, such as neural networks. Bayesian optimization can solve this type of problems in an efficient way. Sequential Model-based Optimization (SMBO) [76] is the simplest form of Bayesian optimization. The complete algorithm is shown in algorithm 3.

Algorithm 3 Sequential Model-Based Optimization

Input: f, X, S, M
$D \leftarrow \text{InitSamples}(f, X)$
for $i \leftarrow |D|$ to T **do**
  $p(y|X, D) \leftarrow \text{FitModel}(M, D)$
  $x_i \leftarrow \arg\max_{x \in X} S(x, p(y|x, D))$
  $y_i \leftarrow f(x_i)$
  $D \leftarrow D \cup (x_i, y_i)$
end for

Since Bayesian search can facilitate searching for optimal parameters, in our experiments, we create a search space for the Bayesian search program to identify the optimal parameters automatically. It is much more accurate and efficient than setting the values of hyperparameters by hand.

5.1.4 Related Work

In view of the fact that there are a larger number of datasets accessible for flow-level intrusion detection studies, the flow-based IDS research are more predominant than the packet-based IDS research.

Studies [5] compare the performances of several traditional machine learning algorithms, such as random forest (RF), K-nearest neighbors (KNN) and Support vector machine (SVM) in DDoS detection for Internet of Things (IoT) devices. To understand the difference between DDoS traffic and normal traffic, the authors devise several user-defined features, instead of directly using features provided by available datasets. Features are classified into two main categories: namely, the stateful features and stateless features. Stateless features represent flow-independent characteristics of individual packets, such as the packet size and protocol. Stateful features describe how traffic evolves over time, such as the bandwidth and IP destination address cardinality and novelty. These features are fed into five machine learning models: RF, decision tree (DT), SVM, DNN and KNN. They have launched an experiment with Raspberry Pi to collect DDoS traffic and then created a dataset for testing. All of the five algorithms obtain high detection accuracy, higher than 99%.

Studies [6] use two stacked bidirectional LSTM with a 20% dropout rate applied between each layer for anomaly detection. Bi-LSTM is a type of black-box time-series model which works well with natural language processing (NLP) problems. For this reason, the authors transfer the intrusion detection problem to a type of NLP problems. By treating the flow as a sentence and the packet as a word. They apply Bi-LSTM to capture the relationships among different words (packets), and predict the communications between two IP addresses. Experiment results show great performance on the ISCX IDS [77] dataset.

Module	Sub-Module	Function
Data Preprocessing	None	Data transformation and Feature Extraction
Reinforcement Learning	Interaction	Store batch data into replay buffer
	Training	Train the agent and the sample agent
Exploration	None	A sub-module of Interaction Module

Table 5.1: Functions of each module defined in Figure 5.3.

5.2 Methods and Procedures

In this section, we elaborate the novel intrusion detection framework proposed at flow-level. The whole framework is shown in Figure 5.3 and Table 5.1. The framework consists of two major modules and a number of sub-modules. The data preprocessing module is devised for data transformation and dimension reduction. The reinforcement learning module is devised for training the intrusion detection system with RL approaches. This module further comprises a training module and an interaction module. Additionally, an agent and a sample agent are designed in the training module. Finally an exploration module is designed for the interaction process. We introduce each module in the following sections in details.

Step	Process (Input $\longrightarrow$ Method $\longrightarrow$ Output)
1	Raw Dataset $\longrightarrow$ Encoding $\longrightarrow$ Encoded Dataset
2	Encoded Dataset $\longrightarrow$ Normalization $\longrightarrow$ Applicable Dataset
3	Applicable Dataset $\longrightarrow$ Feature Learning (PreTrain) with SAE

Table 5.2: Data Preprocessing Module at Flow-level

5.2.1 The Data Preprocessing Module

The complete data preprocessing module is shown in Table 5.2. The dataset collected for flow-level research always involves some discrete and categorical features, such as protocols and packet size. **In step 1**, we need to convert discrete and categor-

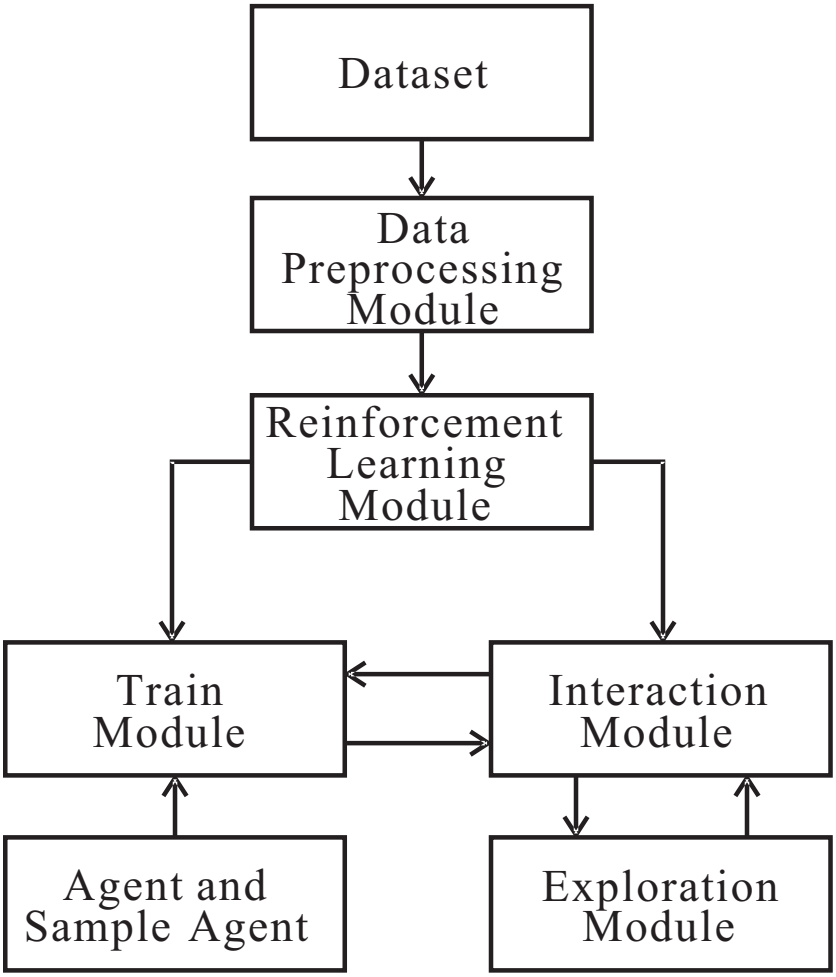


Figure 5.3: Intrusion Detection Framework at Flow-level

ical features into continuous features that can be processed by deep neural networks (DNNs).

With regard to the transformation of categorical features, the most simple and common approach is one hot encoding. As shown in Figure 5.3, one hot encoding is a type of positional encoding. If a categorical variable takes K values, we encode it with a K-dimension vector. For example, the protocol has three values: TCP, UDP and others, we can use positional coding on this attribute, as shown in Table 5.3. One hot encoding has many advantages over other encoding approaches, one of which being the easy implementation.

With regard to the transformation of discrete features, we propose a N-bit binary

Protocol	Encoding Position	Encoding Vector
TCP	0	[1, 0, 0]
UDP	1	[0, 1, 0]
OTHERS	2	[0, 0, 1]

Table 5.3: One Hot Encoding for Feature Protocol

encoding approach. A discrete feature has its maximum value. Based on the maximum value, we can determine the value of N, which ensures that N-bit binaries can encode all values of this attribute. For example, if the maximum value of a discrete feature is 120, we can use 7-bit binary to encode this feature.

In step 2, after we obtain the encoded dataset, we carry out max-min normalization on it, converting all values to $[0,\ 1]$.

Notably, after implementing one hot encoding and binary encoding, the dimension of data considerably increases. **In step 3**, we propose an SAE-based approach to conduct dimension reduction, as well as feature extraction. As previously noted, SAE is efficient in feature extraction and dimension reduction. We pre-train the SAE model and then extract the encoder as the primary structure of the RL agent.

5.2.2 The Reinforcement Learning Module

Similarly as in Chapter 4, we describe the important elements before developing the flow-level intrusion detection systems. Likewise, in the analysis we consider the flow-level intrusion detection as a special game. The comparison between packet-level intrusion detection game and flow-level intrusion detection game is shown in Table 5.4.

As shown in Table 5.4, there are both similarities and differences between the packet-level and the flow-level intrusion detection problems. In both cases, the action represents the prediction performed by the agent. Furthermore, the flow-level reward mechanism is the same with that of packet-level. However, the state and episode are

RL	Packet-level	Flow-level	Flow-level Space
State	Image	Feature Vector	Traffic Features
Action	Prediction	Prediction	Label Space {0, 1, 2}
Reward	Reward Mechanism	Reward Mechanism	{+1, -1}
Episode	Session	User Defined	None
Agent	Classifier	Classifier	None

Table 5.4: Comparison between Packet-level and Flow-level Intrusion Detection

defined differently. At packet-level, we split raw 'pcap' files into separated sessions, then through image embedding we transform these sessions into images, which are the states. However, at flow-level, we directly use traffic features provided by the dataset as the states, and the main structure of the update module is fully connected neural networks. Moreover, at packet-level, packets embedded in the image and images in a session are both arranged in chronological order (capture time), thus we can view a session as an episode. Nonetheless, at flow-level, all the data collected in the dataset has been shuffled, so there is no obvious chronological order. The length of the episode needs to be determined, and it is assumed to be fixed in the interaction process.

In the proposed flow-level intrusion detection system, in addition to the normal agent, we also design a sample agent to facilitate the adversarial training. As shown in Table 5.5, there are some major differences between the agent and the sample agent. The agent performs the correct prediction (the action) by achieving maximum rewards. The sample agent provides guidance (the action) for the next class to be sampled from. To improve the variability, the sample agent tends to counteract the agent. It chooses a class that is most likely to be misclassified and suggests it as the class to be sampled from for the next state. For this reason, the reward feedback of the agent and the sample agent are opposite. If the prediction of the agent is wrong, the reward feedback of the sample agent is 1; Otherwise, if the prediction of agent is correct, the reward feedback of sample agent is -1. This way, the sample

agent functions as the adversarial training agent. The objective of the sample agent is to ensure that those state-action pairs with high classification errors rates can be adequately trained.

	State	**Action**	**Reward**
Agent	Current Features	Prediction	Reward Mechanism
Sample Agent	Current Features	Next Sample Class	Contrary Reward

Table 5.5: Agent and Sample Agent

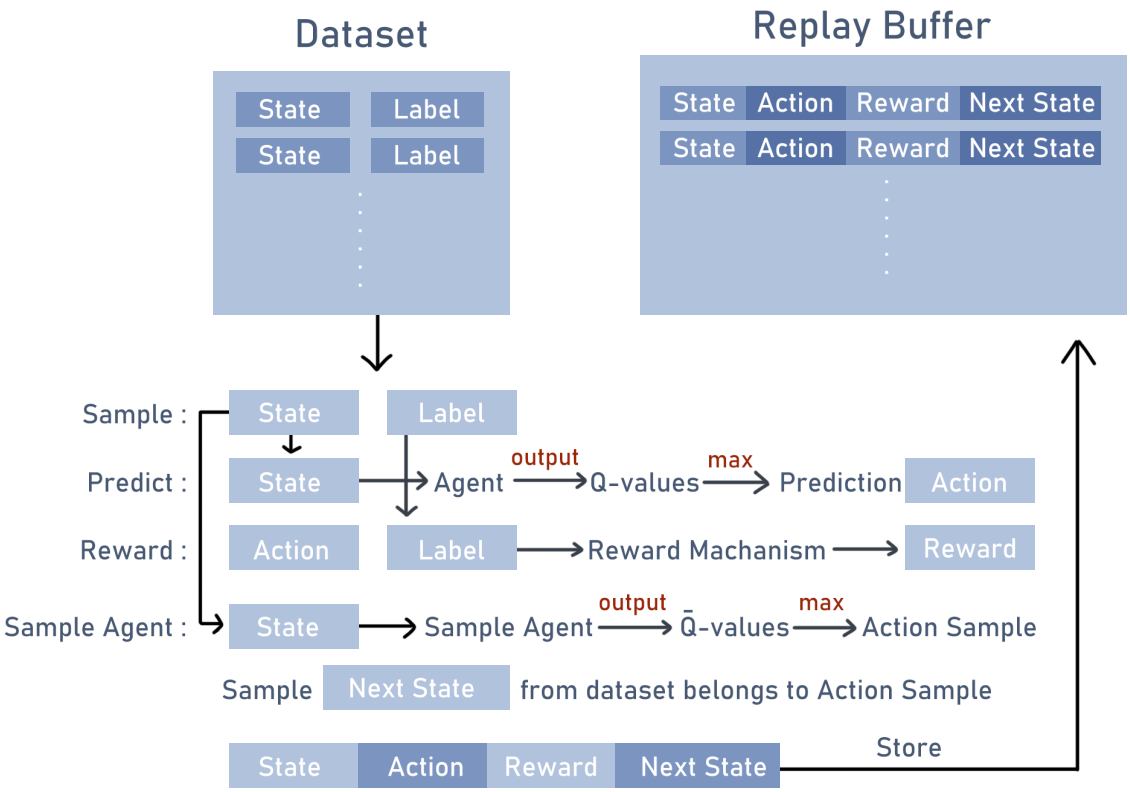


Figure 5.4: Interaction Module

In the interaction stage, we should also create a simulation environment. We take the preprocessed dataset as the simulation environment. In our experiments, we focus on episodic tasks, hence we fix the length of an episode in advance. The complete interaction module is shown in Figure 5.4. All of the traffic features collected in the

dataset can be considered as the states. In the beginning, a state is randomly sampled from dataset. Feeding the state into the agent (classifier), the agent then outputs the prediction (action) of the current state. Feeding the action-label pair into the reward mechanism, we can obtain the reward. If the current state is not reaching the end of the present episode, we also feed the current state to the sample agent and obtain the next sample class. Afterwards, we sample the next state which belongs to this class from the dataset. Next, we store state, action, reward and next state into the replay buffer. Subsequently, treat the next state as the current state and repeat the above process. It should be noted that if the current state is reaching the end of the episode, we store state, action, reward into the replay buffer and then randomly sample a state from the dataset, which indicates that a new episode is launched.

The whole training module is shown in Figure 5.5. The main structure of the update network and target network in the training module is shown in Table 5.6. The agent and the sample agent are trained concurrently, as shown in Algorithm 4.

Encoder			**Decoder**	
InputLayer→	HiddenLayer→	LatentSpace→	HiddenLayer→	OutputLayer
	Agent			
InputLayer→	HiddenLayer→	LatentSpace→	FCN$\xrightarrow{Softmax}$	Q-values
	SampleAgent			
InputLayer→	HiddenLayer→	LatentSpace→	FCN$\xrightarrow{Softmax}$	$\bar{Q}$-values

Table 5.6: Stacked Autoencoder (Encoder and Decoder), Agent and Sample Agent. The encoder is shared with agent and sample agent.

5.2.3 The Exploration Module

In our flow-level IDS, we attempt to employ both the ε-greedy policy and conditional GAN (CGAN) to execute the exploration.

Eqn. (5.2) displays ε-greedy policy which is applied to the sample agent, and ε controls the exploration degree. ε-greedy policy is a commonly used exploration

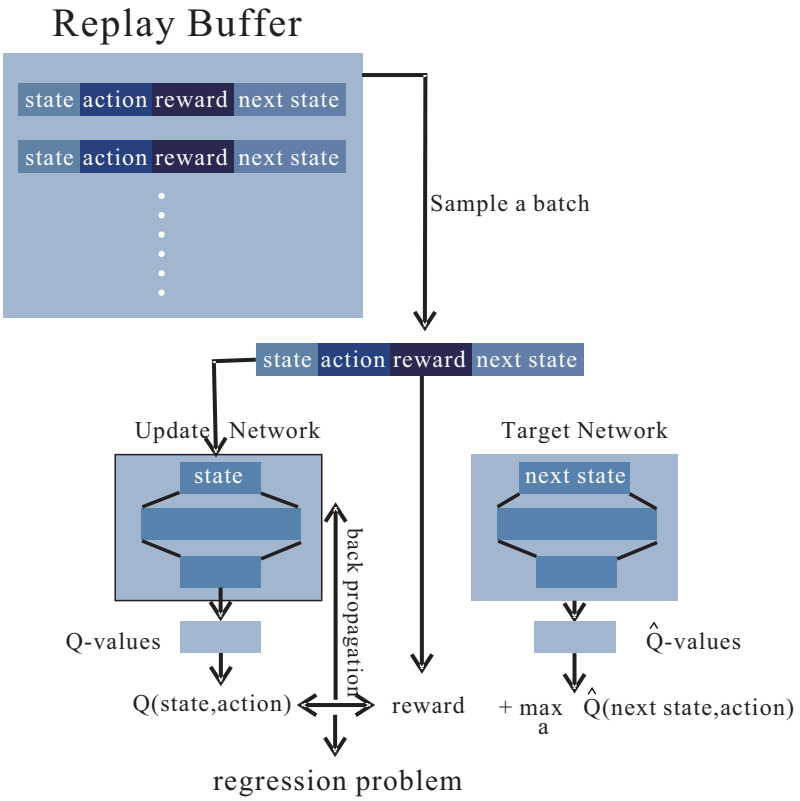


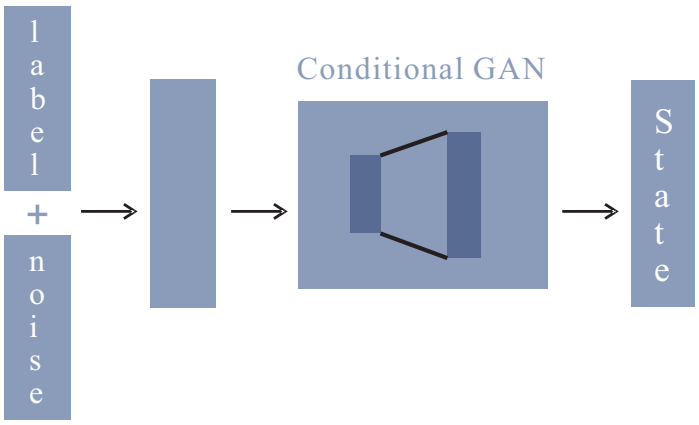
Figure 5.5: Deep Q-Learning Training Module. Update network is updated through back propagation. Target network is initialized with the parameters of update network, and updates are made every N times through copying the parameters from the update network. The mean squared error (MSE) is used as the objective function for the approximate regression problem.

approach in Deep Q-Learning, which encourages the agent to explore novel states through interacting. The action based on ϵ-greedy policy is determined as follows,

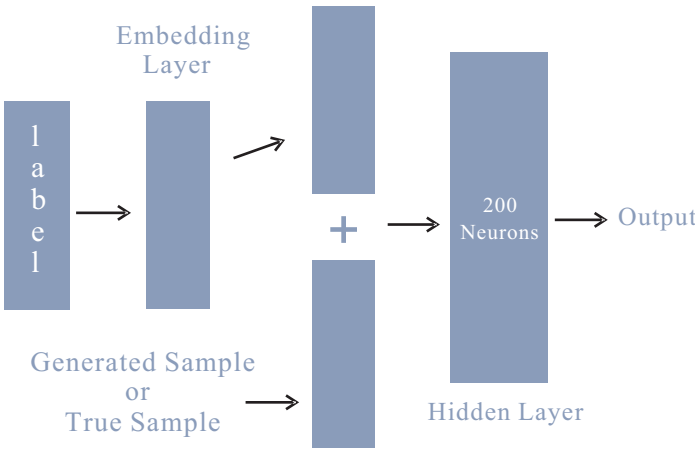
$$a = \begin{cases} \arg\max_a \bar{Q}(s,a) \text{ with probability of 1-}\varepsilon \\ \text{any action with probability of } \varepsilon \end{cases} \tag{5.2}$$

The other exploration approach is conducted by CGAN. A fixed dataset is actually not appropriate for simulating a complicated and ever-changing network environment. Thus, by means of conditional GAN, we attempt to generate some novel attack flows for each class. As shown in Figure 5.6a, the conditional GAN takes the label and noise as the input and outputs a state which belongs to the class.

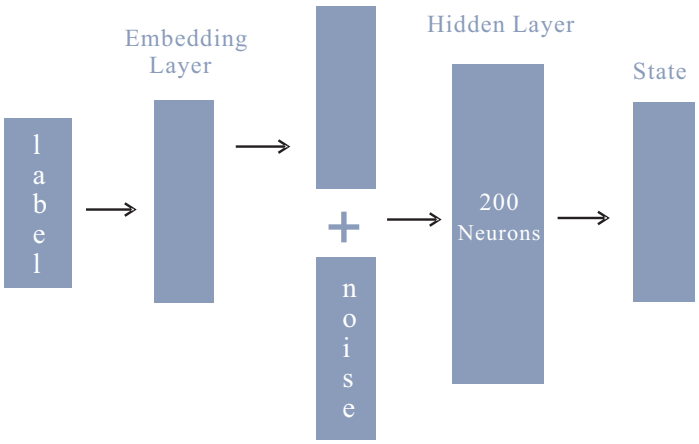
Exploration is conductive to solve imbalance problems. Because the vast majority



(a) Conditional GAN



(b) Discriminator



(c) Generator

Figure 5.6: **(a)**: Conditional GAN **(b)**: Discriminator **(c)**: Generator

of traffic in real-world is normal traffic, it is easier to collect normal traffic than malignant traffic. This results in a bad consequence that most datasets collected for

intrusion detection are unbalanced. Conditional GAN can help mitigate the problem by generating some attack traffic for training. We employ the CGAN exploration rate Γ to control the extent of the exploration. It means that with the probability of Γ, the state comes from the dataset; with the probability of $1-\Gamma$, the state is generated from CGAN.

Conditional GAN comprises a discriminator and a generator. The discriminator is a two-binary classifier, as shown in Figure 5.6b. An embedding layer is deployed to expand the label and the embed label is concatenated with a generated or true sample, composing the complete input which is fed into the classifier. The output is a numerical value ranging in $[0,\ 1]$. If the output is larger than 0.5, the prediction is a real sample; otherwise, the prediction is a generated sample. The cross entropy loss function is applied for this purpose. The generator is revealed in Figure 5.6c, and the functionality of the generator is to generate simulated states which can deceive the discriminator. The label is expanded by the embedding layer and concatenated with noises. The output is the simulated state.

The evaluation of generated samples is a challenging problem in our experiments because we can not judge the quality of generated samples manually. For tackling this problem, we train an additional simple classifier for the dataset. If most of the generated samples can be correctly classified by the classifier, we consider the generator as a good model. We use WGAN-GP in our experiments. The complete algorithm of Deep Q-Learning with an exploration module is shown in Algorithm 4.

5.2.4 Dataset

NSL-KDD [28] and DDoS2019 [25] are adopted in our flow-level experiments in this chapter. These two datasets are briefly introduced in the following.

NSL-KDD has been upgraded and improved based on KDD-99, in which superfluous records of KDD-99 are eliminated. This way the bias of frequent records in the dataset can be significantly reduced. However, as shown in Table 5.7, NSL-KDD is an

Algorithm 4 Deep Q-Learning for Detection with CGAN at Flow-level (RL-CGAN)

Conduct preprocessing on dataset
Pretrain a stacked auto encoder
Train a conditional GAN
Initialize Q-function Q for agent, Q-function $\bar{Q}$ for sample agent
Initialize the agent replay buffer and the sample agent replay buffer for training the agent and the sample agent, respectively.
Copy parameters to Q and $\bar{Q}$ from encoder
Set target Q-function $\hat{Q} = Q$, $\hat{\bar{Q}} = \bar{Q}$
for each episode: **do**
  Randomly choose s_0 from the dataset.
  for each sample within episode, $t \in [0, N]$ **do**
    Given state s_t, take action a_t based on Q
    Given state s_t, take action $\bar{a}_t$ based on $\bar{Q}$ (ε-greedy)
    Compare a_t with true label, obtain reward r_t and $\bar{r}_t$
    Derive next state: choose s_{t+1} whose true label is $\bar{a}_t$ from dataset or CGAN, with probability of $1 - \Gamma$ and Γ, respectively
    Store s_t, a_t, r_t, s_{t+1} into agent replay buffer
    Store $s_t, \bar{a}_t, \bar{r}_t, s_{t+1}$ into sample agent replay buffer
    Sample a batch s_t, a_t, r_t, s_{t+1} from agent replay buffer
    Target $y = r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$
    Update Q through back propagation to make $Q(s_t, a_t)$ close to y
    Sample a batch $s_t, \bar{a}_t, \bar{r}_t, s_{t+1}$ from sample agent replay buffer
    Target $\bar{y} = \bar{r}_t + \gamma \max_a \hat{\bar{Q}}(s_{t+1}, a)$
    Update $\bar{Q}$ through back propagation to make $\bar{Q}(s_t, \bar{a}_t)$ close to $\bar{y}$
    Every C steps reset $\hat{Q} = Q, \hat{\bar{Q}} = \bar{Q}$
  end for
end for

unbalanced dataset, in which some classes may not have sufficient data for training. Taking the class U2R as an example, there are only 22 U2R samples collected in the training set and 17 U2R samples collected in the testing set.

NSL-KDD gathers 41 features, including 38 continuous variables and three categorical variables. In the preprocessing stage, we normalize continuous features to the range of [0, 1] and apply one-hot encoding on the categorical features. After transformation, the dataset consists of totally 122 features, including 38 continuous and 84 binary variables. The dataset has 23 different labels, which can be grouped into four main attack types, as seen in Table 5.8. We use the main class as the label

Main Class	Train Sample	Test Sample	Class
Normal	67434	11449	0
DoS	45927	7456	1
Probe	11656	2102	2
R2L	934	1520	3
U2R	22	17	4

Table 5.7: Train set and Test set. We treat Normal as 0, DoS as 1, Probe as 2, R2L as 3 and U2R as 4.

Main Class	Sub Class
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 5.8: Attack Sub Categories of NSL-KDD

in our experiments, but not the subclass.

The other dataset used for the experiment is DDoS2019. We have adopted this dataset in Chapter 4. In packet-level research, we extract features from raw network traffics recorded in 'pcap' files. DDoS2019 also provides researchers with the flow-level dataset. Entirely 79 features are extracted in the dataset, and some of the features are listed in Table 5.9.

There are totally three types of features collected in DDoS2019, including continuous, discrete and categorical features. We still conduct one-hot encoding on categorical features. With regard to the transformation of discrete features, we adopt

Type	Feature (Part)	Encoding
Continuous	Flow Duration, Average Packet Size	None
Discrete	Total Fwd Packets, Total Backward Packets, Total Length of Fwd Packets, Fwd Packet Length Max	Binary Encoding
Categorical	Protocol	One-hot Encoding

Table 5.9: DDoS2019 Features and transformation

the binary transformation approach, with which we can transfer discrete features to a 9-bit binary to encode the value from 0 to 1023, which is sufficient for the dataset. Continuous features are not required to be transformed.

The statistics of DDoS2019 are listed in Table 5.10. DDoS2019 is a huge dataset with millions of flows stored for each type. We extract 25000 flows for each traffic type. In this flow-based IDS experiment, we exclude anomaly attacks considered in chapter 4.

Traffic Class	Train Set	Validation Set	Test Set	Class
Normal	20000	5000	5000	0
Portmap	20000	5000	5000	1
NetBIOS	20000	5000	5000	2
LADP	20000	5000	5000	3
MSSQL	20000	5000	5000	4
UDP	20000	5000	5000	5
UDP-Lag	20000	5000	5000	6
SYN	20000	5000	5000	7

Table 5.10: DDoS2019 Description

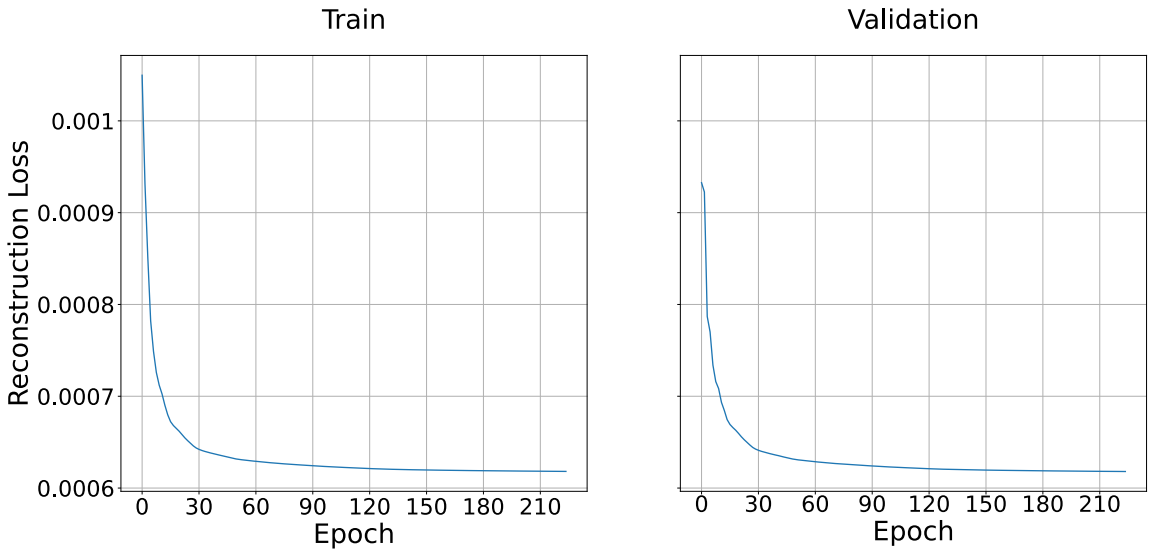


Figure 5.7: Stacked autoencoder performances on NSL-KDD

5.3 Results and Discussion

5.3.1 Evaluation Metrics

In our experiments, we utilize accuracy, precision, recall, and F1 score to evaluate the performance. The definitions of these metrics are shown in Table 4.8 and Table 4.9. Accuracy can evaluate the general quality of the system. Recall can evaluate how many attacks are detected. Precision expresses how many of the traffic judged to be in a certain category are correct. The F1 score is a trade-off between the two metrics, recall and precision.

5.3.2 NSL-KDD Experiment Results

In our experiments, we discard U2R traffics because 22 samples can not provide any helpful information. We pretrain a stacked autoencoder for the remaining data. The structure of the stacked autoencoder is shown in Table 5.6. The dimension of the input layer, hidden layer and latent space is 122, 60, 15, respectively. The dimension of the fully connected layer is four, equal to the number of classes. We split the training set in the ratio of 8:2, 80% for train and 20% for validation.

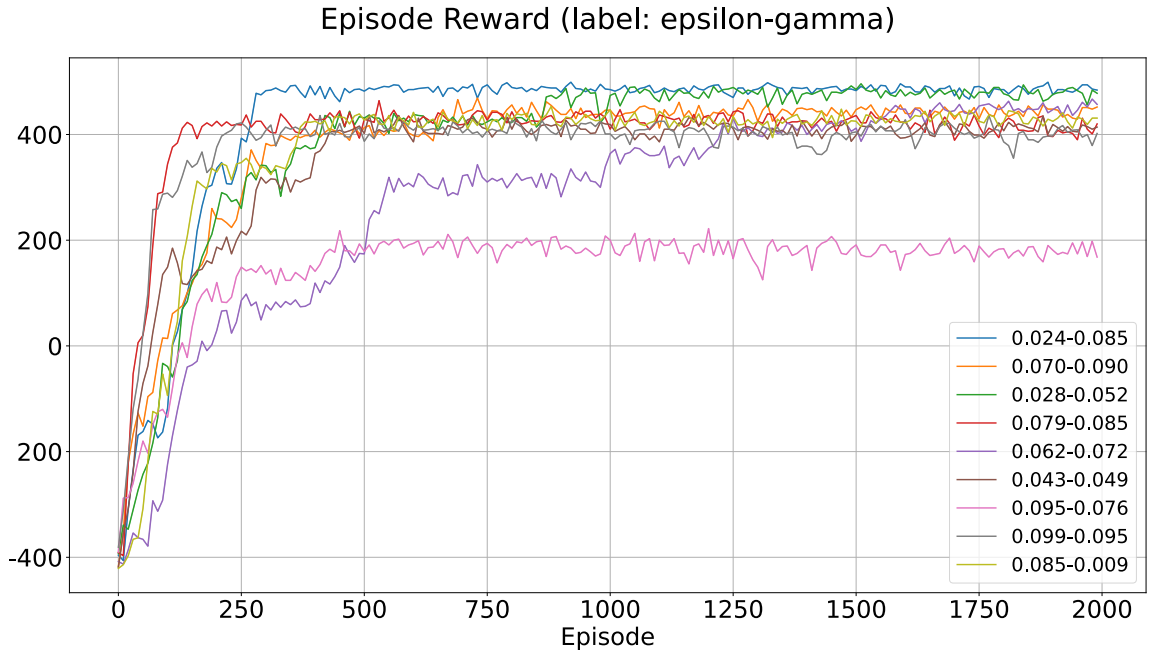
Experiments are conducted on the computer configured as RTX2070, i5-9600k

and 32GB Memory. As shown in Table 5.6, the encoder is extracted as the main structure of the RL agent. Meanwhile, the parameters are also migrated to the agent. An additional Softmax layer is added to implement classification. Furthermore, the sample agent takes the same structure as the agent, and the first three layers are shared with the agent at the beginning. We define the length of the episode as 512. The maximum reward of an episode is 512 (all states are correctly predicted); the minimum reward of an episode is -512 (all states are wrongly predicted).

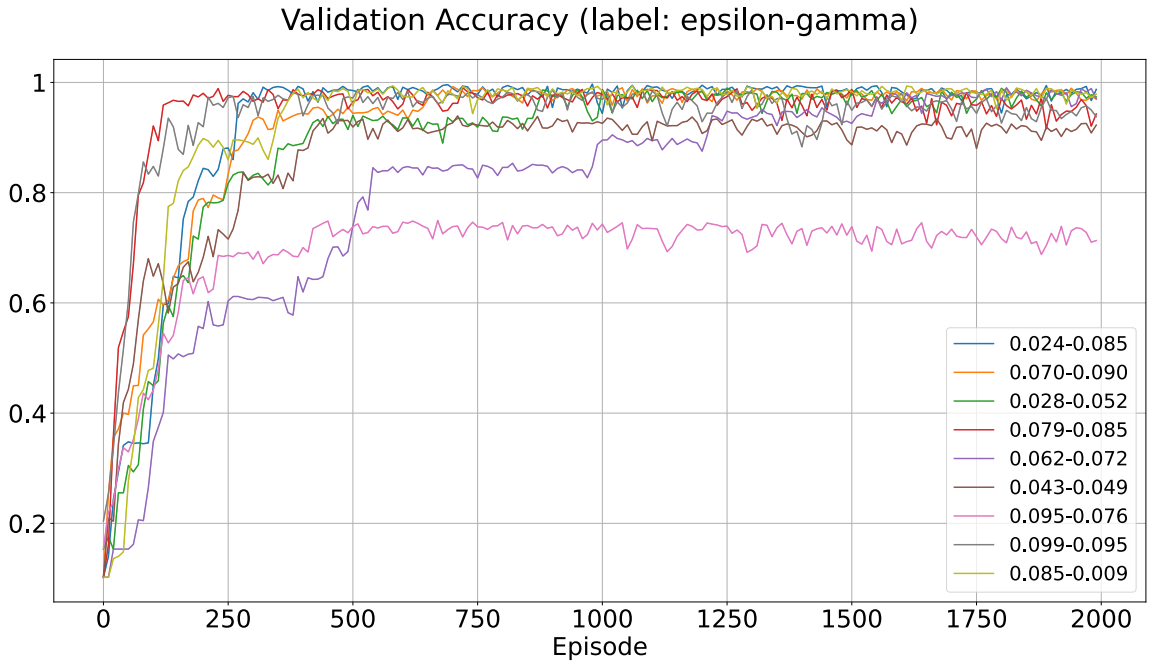
In the first stage, we explore the influence of epsilon ε and gamma γ. The experiments results are shown in Figure 5.8 and Table 5.13. Figure 5.8 shows great performances with small discount value γ and small exploration value ε. The epsilon-gamma pair is obtained by Bayesian search program. We can discover that the intrusion detection system can achieve excellent performance with small value of γ (smaller than 0.1) and ε (smaller than 0.1) in most cases. As shown in Figure 5.8, ruling out an extreme case, all of the epsilon-gamma pairs can receive more than 400 points in an episode. The optimal epsilon-gamma pair can reach more than 500 points. From the perspective of accuracy, most agents can reach the classification accuracy of 90%. Some agents can reach the peak of the accuracy and episode reward at around 250 episodes. Table 5.13 extracts some important information from the figure. The mean episode reward and validation accuracy are the mean value of last 10 episodes. The optimal mean validation accuracy can reach 98.25%, with the γ of 0.085 and ε of 0.024.

We create three search spaces for the Bayesian search program and compare the performances on different values of γ and ε. We record the optimal parameters of each search round in Table 5.12. It is shown that the discount value has a significant impact on the performance while the influence of the exploration rate is relatively trivial. In addition, a smaller γ is more suitable for the intrusion detection system.

In the second stage, we strengthen the intrusion detection system with CGAN to simulate a more realistic network environment with the optimal parameter discovered



(a)



(b)

Figure 5.8: Performances on validation set on NSL-KDD. The averaged computation time for training on each episode is **8.9s**.

in the first stage. We test the final intrusion detection system on the test set, as shown in Figure 5.13. The experiment results are shown in Table 5.13. We use accuracy,

γ	ε	**Mean Episode Reward**	**Mean Val Accuracy**
0.085	0.024	485.0	98.25%
0.090	0.071	442.0	97.82%
0.009	0.085	427.6	97.98%
0.096	0.099	402.0	94.60%
0.076	0.096	180.6	72.33%

Table 5.11: Experiments results (averaged over last 10 episodes) without CGAN

γ	ε	**MeanEpisodeReward**	**MeanValAccuracy**
0.085 (0, 0.1)	0.024 (0, 0.1)	485.0	98.25%
0.485 (0.4, 0.6)	0.098 (0, 0.1)	387.0	89.84%
0.912 (0.9, 1)	0.065 (0, 0.1)	175.0	68.97%
0.094 (0, 0.1)	0.567 (0.4, 0.6)	425.6	96.43%

Table 5.12: Performances (averaged over last 10 episodes) on validation set. (a, b) denotes the search space for Bayesian search program.

precision, recall and F1 score for evaluation. The definitions of these metrics are shown in Table 4.8 and Table 4.9.

Results are shown in Table 5.13. Obviously, CGAN can contribute to the robustness of the intrusion detection system. In general, CGAN can improve the overall accuracy of the intrusion detection system, improving from 85.98% to 88.50%. Concerning different traffic types, CGAN contributes a lot to the detection of normal and Dos traffics, but not works in the remaining two types. There could be many reasons resulting in the outcome. Firstly, the training samples of Probe and R2L are not sufficient for training a great generator to generate high-quality Probe and R2L traffics. Secondly, the test set is limited, which may not reflect the advantages of CGAN.

Exploration Rate Γ: 0.0		**Overall Accuracy: 85.98%**	
Time[1]: 0.087s			
Class	Precision	Recall[2]	F1
Normal	89.08%	90.01%	89.54%
Dos	88.79%	82.15%	85.34%
Probe	73.93%	80.40%	77.03%
R2L	70.38%	82.24%	75.85%
Exploration Rate Γ: 0.5		**Overall Accuracy: 88.50%**	
Time: 0.131s			
Class	Precision	Recall	F1
Normal	89.90%	95.14%	92.45%
Dos	96.08%	84.92%	90.16%
Probe	75.09%	78.45%	76.73%
R2L	71.04%	76.18%	73.52%

[1] The computation time for detection.
[2] Precision and recall are used in two-class classification, one class is positive and the other is negative. In our experiments we view current class as the positive and other three classes as negative.

Table 5.13: Performances on test set with RL-CGAN

5.3.3 DDoS2019 Experiment Results

The same experiments are conducted on DDoS2019. The complete experiment procedure is the same as what we have done on NSL-KDD. After data preprocessing, the dimension of the state is 97. The structure of the stacked autoencoder is shown in Table 5.6. The dimension of the input layer, hidden layer and latent space is 97, 60, 25, respectively. The dimension of the fully connected layer is eight, equal to the number of classes. Figure 5.9 shows the performance of the stacked autoencoder. As shown in Figure 5.9, the performance of the stacked autoencoder is also excellent on DDoS2019.

In the first stage, we explore the influence of epsilon ε and gamma γ. We also fix

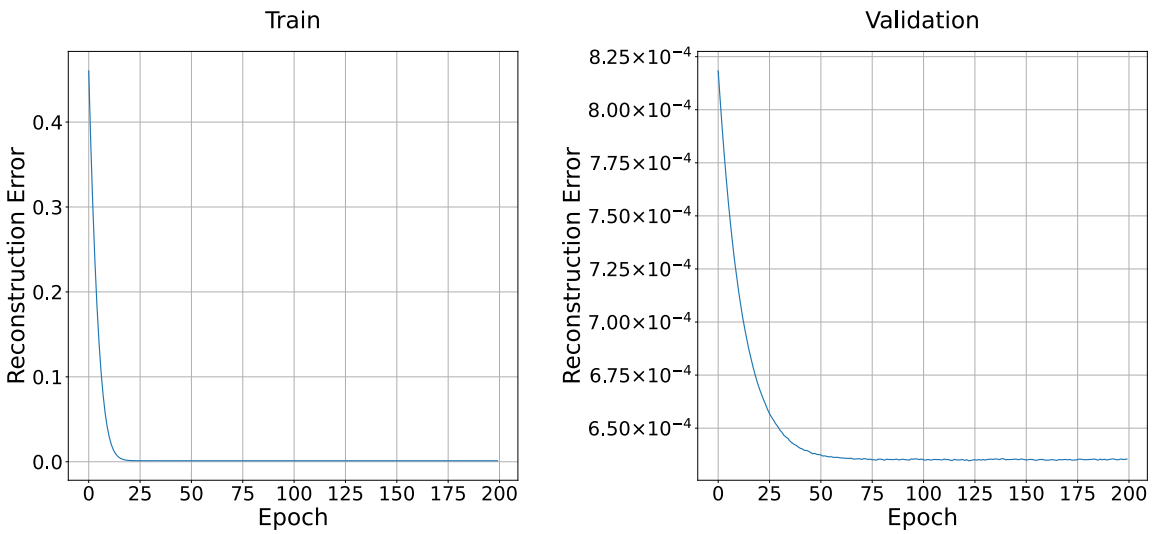


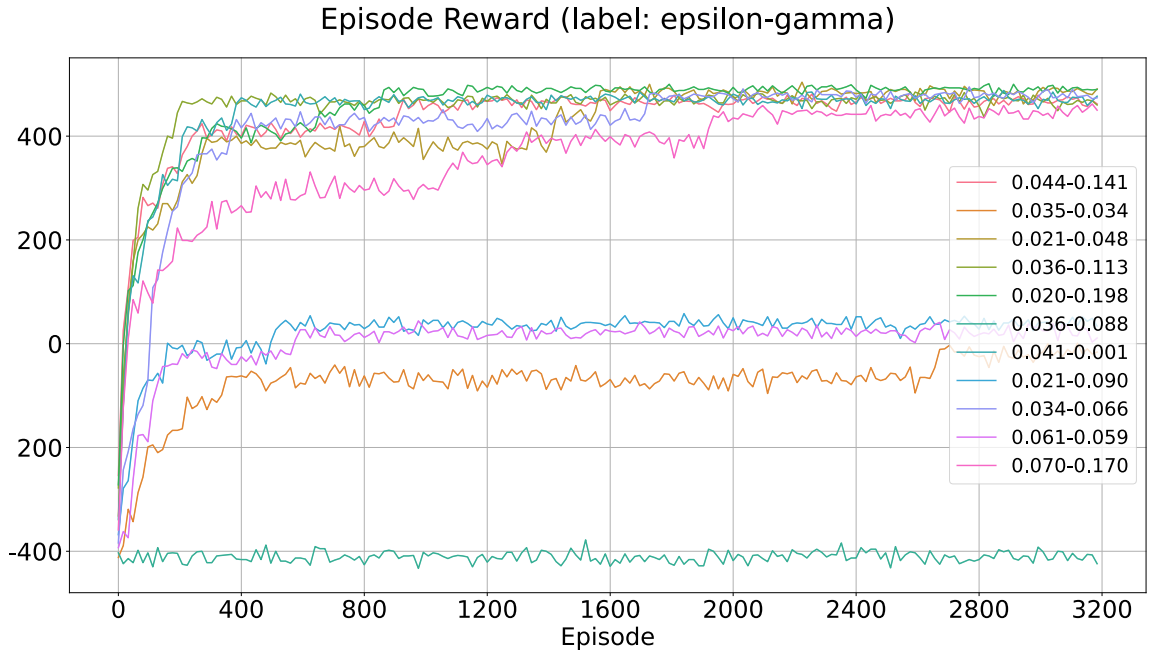
Figure 5.9: Performances of stacked autoencoder on DDoS2019

γ	ε	**Mean Episode Reward**	**Mean Val Accuracy**
0.034	0.066	477.3	98.86%
0.041	0.001	471.3	98.48%
0.070	0.170	449.8	97.73%
0.061	0.059	22.2	54.47%
0.036	0.088	-410.6	10.10%

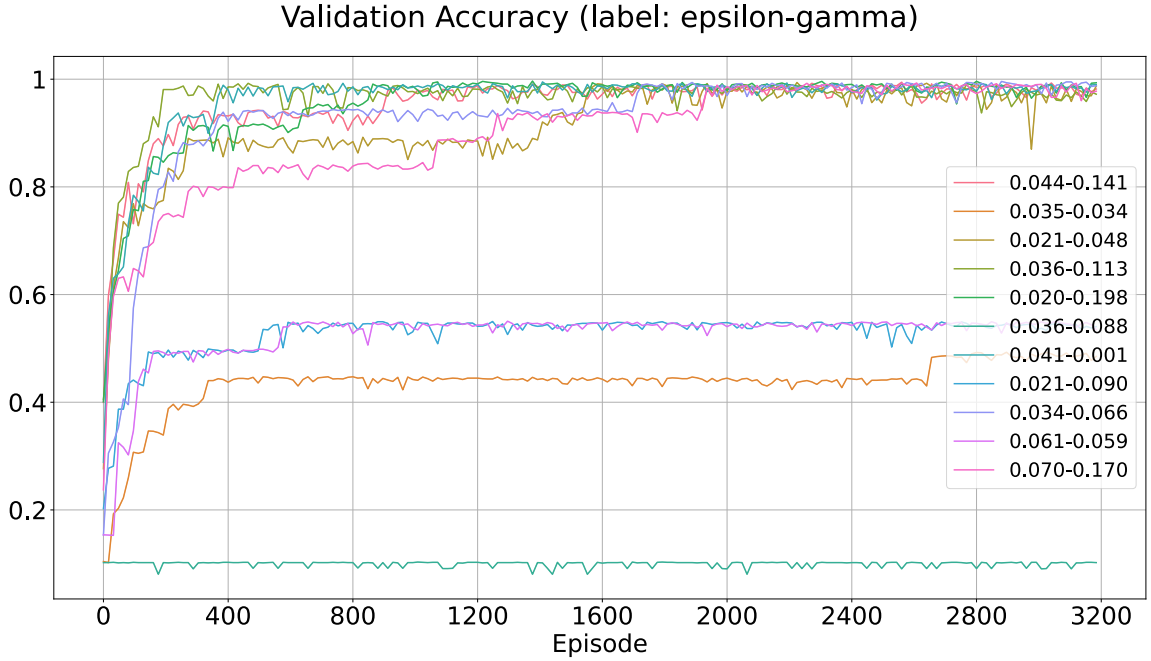
Table 5.14: Experiments results (averaged over last 10 episodes) without CGAN

the length of the episode as 512. The experiment results are shown in Figure 5.10. We create a search space (0, 0.2) for both ε and γ. From the plots we can find that despite some anomaly pairs generate worse performances, most pairs associated with relatively small ε and γ can reach high accuracy and obtain high episode reward. We extract some statistics from Figure 5.10 and store in Table 5.14. The optimal pair is γ-0.034, ε-0.066, reaching 98.86% on validation set. The worst pair is γ-0.036, ε-0.088. We can discover that a small variance of parameters can have a significant impact on performance. Thus, the Bayesian search program is necessary for our experiments.

Afterward, we create another two search spaces for the pair γ and ε and record the optimal results in Table 5.15. The results inform that a large gamma γ is not



(a)



(b)

Figure 5.10: Performances on validation set on DDoS2019. The averaged computation time for training on each episode is **6.9s**.

appropriate for our intrusion detection system. If we adjust γ to a relatively small value, we can ensure that the intrusion detection system can always achieve good

performances.

In the second stage, we strengthen the intrusion detection system with CGAN for exploration and data augmentation. Experiments results on the test set are shown in Table 5.16. Using CGAN, the intrusion detection system attains better performance, improving from 93.67% to 96.43%. The enhancement achieved by CGAN prove that we can simulate a more realistic network environment through CGAN.

γ	ε	**MeanEpisodeReward**	**MeanValAccuracy**
0.034 (0, 0.2)	0.066 (0, 0.2)	477.3	98.86%
0.529 (0.4, 0.6)	0.062 (0, 0.2)	462.9	96.82%
0.804 (0.8, 1)	0.102 (0, 0.2)	40.9	54.22%
0.015 (0, 0.2)	0.596 (0.4, 0.6)	472.6	98.49%

Table 5.15: Performances (averaged over last 10 episodes) on validation set. (a, b) denotes the search space for Bayesian search program.

5.3.4 Comparison with Machine Learning Approaches

In this section, we compare our RL-CGAN-based approach to some other frequently used machine learning algorithms, including random forest, support vector machine, adaboost [78], FCN and LSTM on NSL-KDD and DDoS2019. We use one-hot encoding to encode the NSL-KDD dataset and conduct max-min normalization on it. Similarly, we use one-hot encoding and binary transformation to encode the DDoS2019 dataset, and then scale all values to $[0, 1]$. We compare the performances of different algorithms on the test set, and the results are shown in Table 5.17 and Table 5.18.

With respect to NSL-KDD, as shown in Table 5.17, our approach outperforms all other algorithms. Ensemble approaches may undertake overfitting on the training set. Support vector machine performs slightly better than ensemble approaches, but it consumes more time for training. Neural Networks perform better than traditional machine learning approaches. 3-Layer LSTM performs marginally better than 3-layer

Exploration Rate Γ: 0.0		**Overall Accuracy: 93.67%**	
Time[1]: 0.059s			
Class	**Precision**	**Recall**	**F1**
Normal	96.23%	95.52%	95.87%
PortMap	93.79%	93.96%	93.88%
NetBIOS	92.14%	96.12%	94.09%
LDAP	92.53%	92.72%	92.63%
MSSQL	91.48%	95.50%	93.44%
UDP	93.67%	89.98%	91.79%
UDP-Lag	93.92%	91.70%	92.79%
SYN	95.79%	93.82%	94.80%
Exploration Rate Γ: 0.5		**Overall Accuracy: 96.42%**	
Time: 0.083s			
Class	**Precision**	**Recall**	**F1**
Normal	98.11%	97.54%	97.82%
PortMap	96.45%	97.14%	96.79%
NetBIOS	95.45%	98.20%	96.81%
LDAP	95.98%	95.44%	95.71%
MSSQL	94.86%	96.76%	95.80%
UDP	96.52%	93.68%	95.08%
UDP-Lag	96.75%	95.76%	96.25%
SYN	97.29%	96.80%	97.04%

[1] The computation time for detection.

Table 5.16: Performances on test set with RL-CGAN

FCN, but LSTM has much more parameters to train than fully connected neural networks.

We exclude anomaly types in the test set of DDoS2019 as we have done earlier. As shown in Table 5.18, in comparison to NSL-KDD, DDoS2019 is a much more balanced and sufficient dataset, so the general performances on DDoS2019 are relatively

better. Ensemble methods, including random forest and adaboost, achieve great performances on DDoS2019. Support vector machine is the worst one among all models, and it takes the longest time to train and detectio. Neural networks, including fully connected neural networks and LSTM, also attain great performances. Our approach still outperforms all other models listed in Table 5.18 without increasing much computation time.

In conclusion, our proposed RL-CGAN-based approach, which incorporates with an exploration module and a sample agent, can improve the performances of intrusion detection systems.

Approach	**Accuracy**	**Precision**	**Recall**	**F1**[1]	**Time**[2]
Random Forest	79.56%	83.80%	58.55%	60.87%	1.695s
Adaboost	61.36%	72.19%	48.94%	58.33%	2.012s
SVM	79.71%	82.35%	61.98%	70.73%	9.211s
3-Layer FCN	82.09%	**85.14%**	70.03%	76.85%	**0.058s**
3-Layer LSTM	84.34%	84.91%	72.45%	78.19%	0.349s
RL-CGAN	**88.50%**	83.02%	**83.67%**	**83.22%**	0.102s

[1] Precision, recall and F1 are the averaged value of each class.
[2] The computation time for detection.

Table 5.17: Comparison among different approaches on NSL-KDD

Approach	**Accuracy**	**Precision**	**Recall**	**F1**	**Time**
Random Forest	95.42%	**96.72%**	94.12%	95.40%	1.751s
Adaboost	94.79%	93.98%	95.11%	94.54%	2.610s
SVM	90.12%	92.15%	88.98%	90.54%	10.019s
3-Layer FCN	94.12%	93.09%	95.24%	94.15%	**0.034s**
3-Layer LSTM	93.90%	94.01%	91.99%	92.99%	0.192s
RL-CGAN	**96.42%**	96.43%	**96.42%**	**96.41%**	0.092s

Table 5.18: Comparison among different approaches on DDoS2019

5.4 Conclusions

In this chapter, we conduct comprehensive research on intrusion detection at the flow-level with two datasets. Our approach has numerous advantages and innovations over other traditional machine learning methods.

In the first place, we incorporate stacked autoencoder into our reinforcement learning module. This feature learning step is crucial for the success of detection. We can learn a more simple but purposeful vector for final detection or classification. As mentioned previously, stacked autoencoder has already been widely applied to cybersecurity and achieved accomplishments. Our experiments strongly support this conclusion.

Secondly, we take full advantage of reinforcement learning algorithms. We introduce discount factor γ into our intrusion detection systems. In such a way, we can take back states into account when examining the current state. We also create a simulation environment for interaction. For adapting to complicated real network environments, we make use of conditional GAN to strengthen the simulation environments.

Thirdly, we design a sample agent in our experiments as the adversarial training agent. Through the sample agent, we can force our classifier (agent) to give greater attention to those incorrectly predicted action-state pairs. Cooperating with ε-greedy algorithm, Q values of all states can be appropriately learned.

In addition, we adopt the Bayesian search program in our experiments. Reinforcement learning is sensitive to hyperparameters, which indicates that setting these hyperparameters manually may lead to incorrect solutions. Besides, the Bayesian search program also helps save time on finding optimal hyperparameters.

5.5 Future Work

In future research, we wish to focus more on the application of GAN.

Firstly, we attempt to apply GAN to generate more realistic traffic. In chapter 4, we transfer sessions into images. It is known to all that GAN performs greatly on generating images [70]. Generating novel session images for traffic generation is a potential research direction. Moreover, we can add some constraints to the generator. Attack traffics may have some specific and fixed features, which can be learnt through training. In addition, some features may only take integer values. We can try to use reinforcement learning approaches to train GAN [79].

Secondly, we can make further developments on the reinforcement learning module. For example, the reward mechanism can be detailed devised. When playing Atari games, game developers set different scores for the different operations of game users. We can also design a comprehensive reward feedback system. For example, we can give a larger penalty point to the intrusion detection system when some malignant traffics are not detected by the system because this may cause severe loss.

Thirdly, we can use some other reinforcement learning algorithms to solve the discrete problem. We use Deep Q-Learning in our experiments because it is simple to implement and appropriate for solving discrete problems. However, there are a great number of algorithms that can also be applied into the discrete domain, such as Policy Gradient methods and Double Deep Q-Learning algorithms. It is reasonable to compare the performances of these algorithms.

Chapter 6

Conclusions

The thesis mainly focuses on two popular and promising areas, one is reinforcement learning algorithms, and the other is cybersecurity. Over a series of experiments, we successfully extend the reinforcement learning application to cybersecurity.

In chapter 3, we study the properties of PPO on the discrete domain. Since PPO has already achieved great performances on controlling Atari games, we study further on the core idea (clipped surrogate objective) and some code level skills applied on PPO to figure out whether they have significant impacts on the final performances of PPO. We firstly conduct a series of comparative experiments on code level skills. Experiments results indicate that most code level skills, including value loss clip, orthogonal initialization, GAE and reward clip can significantly improve the performances of PPO on selected Atari games. Most importantly, reward clip has a most considerable impact on the performances, thus we conduct in-depth experiments on reward clip and find that a small reward scale is conducive to the final performances. In addition, we study the functionality of clipped surrogate objective in a large range of learning rates. Experiment results indicate that the clipped surrogate objective can stabilize the performances of PPO over a relatively larger range of learning rates.

In the following two chapters, the main applied reinforcement learning algorithm is Deep Q-Learning. The Deep Q-Learning algorithm is also one of the most accepted reinforcement learning algorithms. Different from PPO, Deep Q-Learning is not suit-

able for continuous tasks, but it is an excellent tool for solving discrete problems. We explore the value of Deep Q-Learning in the area of cybersecurity in the following two chapters.

In chapter 4, we explore the potential value of reinforcement learning on building intrusion detection systems at the packet-level.We propose a novel embedding approach, namely image embedding, to encode the network traffics. Utilizing image encoding, raw network traffics, which are difficult to tackle by machine learning models, can be converted to images. Thus, convolutional neural networks can be applied in the experiments. In addition, packets embed in images are arranged in time order. In this way, we can engineer some flow-level features into packet-level experiments. Meanwhile, 1D-CNN can also be applied in the experiments. With respect to the reinforcement learning framework, we compare the packet-level intrusion detection game to Atari games and transform the cybersecurity problem to a reinforcement learning based problem. We select the Deep Q-Learning algorithm in our experiments, and design a training module and an interaction module. Experiments results indicate that our RL-image-based approach can attain high performance on raw DDoS traffics provided by DDoS2019 and outperforms other traditional deep learning approaches.

In chapter 5, we study the potential value of reinforcement learning on building intrusion detection systems at the flow-level. On the basis of packet-level intrusion detection framework, we establish the intrusion detection system from the flow angle and make some enhancements. We use two datasets NSL-KDD and DDoS2019 for our experiments. We pretrain a stacked autoencoder for a first-step feature learning and dimension reduction. Besides, we devise a sample agent as the adversarial training agent to ensure that all states can be learned adequately. We also implement an exploration policy in our experiments. The first exploration policy is ε-greedy policy. This is a widely used exploration policy of Deep Q-Learning, which can ensure that most states that occurred in the environment can learn their values. The other exploration approach is conducted by conditional GAN, which can help stim-

ulate a more realistic interaction environment by generating novel traffics. We also employ a Bayesian search program to facilitate finding the optimal hyperparameters automatically. Experiment results show that our RL-CGAN-based approach with exploration can attain great scores on NSL-KDD and DDoS2019 and outperforms other traditional machine learning approaches.

Bibliography

[1] S. E. Smaha *et al.*, "Haystack: An intrusion detection system," in *Fourth Aerospace Computer Security Applications Conference*, Orlando, FL, USA, vol. 44, 1988.

[2] S. Dua and X. Du, *Data Mining and Machine Learning in Cybersecurity*, 1st. USA: Auerbach Publications, 2011, ISBN: 1439839425.

[3] R.-H. Hwang, M.-C. Peng, V.-L. Nguyen, and Y.-L. Chang, "An lstm-based deep learning approach for classifying malicious traffic at the packet level," *Applied Sciences*, vol. 9, no. 16, 2019, ISSN: 2076-3417. DOI: 10.3390/app9163414. [Online]. Available: https://www.mdpi.com/2076-3417/9/16/3414.

[4] M. Ge, X. Fu, N. Syed, Z. Baig, G. Teo, and A. Robles-Kelly, "Deep learning-based intrusion detection for iot networks," in *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, IEEE, 2019, pp. 256–25 609.

[5] R. Doshi, N. Apthorpe, and N. Feamster, "Machine learning ddos detection for consumer internet of things devices," in *2018 IEEE Security and Privacy Workshops (SPW)*, IEEE, 2018, pp. 29–35.

[6] B. J. Radford, L. M. Apolonio, A. J. Trias, and J. A. Simpson, "Network traffic anomaly detection using recurrent neural networks," *arXiv e-prints*, arXiv–1803, 2018.

[7] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1.

[8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An introduction*. MIT press, 2018.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[11] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[12] M. Chen, A. Beutel, P. Covington, S. Jain, F. Belletti, and E. H. Chi, "Top-k off-policy correction for a reinforce recommender system," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019, pp. 456–464.

[13] Y. Hu, Q. Da, A. Zeng, Y. Yu, and Y. Xu, "Reinforcement learning to rank in e-commerce search engine: Formalization, analysis, and application," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 368–377.

[14] I. Carlucho, M. De Paula, S. Wang, Y. Petillot, and G. G. Acosta, "Adaptive low-level control of autonomous underwater vehicles using deep reinforcement learning," *Robotics and Autonomous Systems*, vol. 107, pp. 71–86, 2018.

[15] E. Bøhn, E. M. Coates, S. Moe, and T. A. Johansen, "Deep reinforcement learning attitude control of fixed-wing uavs using proximal policy optimization," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2019, pp. 523–533.

[16] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia, "Learning swing-free trajectories for uavs with a suspended load," in *2013 IEEE International Conference on Robotics and Automation*, IEEE, 2013, pp. 4902–4909.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2016, pp. 1928–1937.

[18] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International Conference on Machine Learning (ICML)*, 2015, pp. 1889–1897.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI gym," *arXiv:1606.01540*, 2016.

[20] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[21] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep policy gradients: A case study on ppo and trpo," in *International Conference on Learning Representations*, 2020.

[22] P. Henderson, J. Romoff, and J. Pineau, "Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods," *ArXiv*, vol. abs/1810.02525, 2018.

[23] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, *et al.*, "What matters in on-policy reinforcement learning? a large-scale empirical study," in *ICLR 2021-Ninth International Conference on Learning Representations*, 2021.

[24] M. J. Beal, Z. Ghahramani, and C. E. Rasmussen, "The infinite hidden markov model," *Advances in Neural Information Processing Systems*, vol. 1, pp. 577–584, 2002.

[25] I. Sharafaldin, A. H. Lashkari, S. Hakak, and A. A. Ghorbani, "Developing realistic distributed denial of service (ddos) attack dataset and taxonomy," in *2019 International Carnahan Conference on Security Technology (ICCST)*, IEEE, 2019, pp. 1–8.

[26] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, 2021. arXiv: 2003.05991 [cs.LG].

[27] M. Mirza and S. Osindero, *Conditional generative adversarial nets*, 2014. arXiv: 1411.1784 [cs.LG].

[28] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, "A detailed analysis of the kdd cup 99 data set," in *2009 IEEE symposium on computational intelligence for security and defense applications*, IEEE, 2009, pp. 1–6.

[29] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[30] V. R. Konda and J. N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, Citeseer, 2000, pp. 1008–1014.

[31] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," *arXiv preprint arXiv:1611.06256*, 2016.

[32] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[33] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, PMLR, 2018, pp. 1861–1870.

[34] R. Wehrens, H. Putter, and L. M. Buydens, "The bootstrap: A tutorial," *Chemometrics and intelligent laboratory systems*, vol. 54, no. 1, pp. 35–52, 2000.

[35] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, 2016.

[36] Y. Huang, G. Wei, and Y. Wang, "Vd d3qn: The variant of double deep q-learning network with dueling architecture," in *2018 37th Chinese Control Conference (CCC)*, IEEE, 2018, pp. 9130–9135.

[37] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063.

[38] OpenAI, *Openai five*, https://blog.openai.com/openai-five/, 2018.

[39] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[40] D. Ye, Z. Liu, M. Sun, B. Shi, P. Zhao, H. Wu, H. Yu, S. Yang, X. Wu, Q. Guo, Q. Chen, Y. Yin, H. Zhang, T. Shi, L. Wang, Q. Fu, W. Yang, and L. Huang, "Mastering complex control in moba games with deep reinforcement learning," in *AAAI*, 2020.

[41] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[42] J. jie Li, S. Koyamada, Q. Ye, G. Liu, C. Wang, R. Yang, L. Zhao, T. Qin, T. Liu, and H. Hon, "Suphx: Mastering mahjong with deep reinforcement learning," *ArXiv*, vol. abs/2003.13590, 2020.

[43] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, IEEE, 2017, pp. 43–48.

[44] S. A. Salloum, M. Alshurideh, A. Elnagar, and K. Shaalan, "Machine learning and deep learning techniques for cybersecurity: A review.," in *AICV*, 2020, pp. 50–57.

[45] G. Biau, "Analysis of a random forests model," *The Journal of Machine Learning Research*, vol. 13, pp. 1063–1095, 2012.

[46] W. S. Noble, "What is a support vector machine?" *Nature Biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006.

[47] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Shape, Contour and Grouping in Computer Vision*, Springer, 1999, pp. 319–345.

[48] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107 398, 2021.

[49] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning precise timing with lstm recurrent networks," *Journal of machine learning research*, vol. 3, no. Aug, pp. 115–143, 2002.

[50] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, *Openai baselines*, https://github.com/openai/baselines, 2017.

[51] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *CoRR*, vol. abs/1312.6120, 2014.

[52] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[53] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[54] J. Achiam, "Spinning Up in Deep Reinforcement Learning," 2018.

[55] M. Tomar, L. Shani, Y. Efroni, and M. Ghavamzadeh, "Mirror descent policy optimization," *arXiv preprint arXiv:2005.09814*, 2020.

[56] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *Journal of Machine Learning Research - Proceedings Track*, vol. 9, pp. 249–256, Jan. 2010.

[57] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach (6th Edition)*, 6th. Pearson, 2012, ISBN: 0132856204.

[58] U. Lamping and E. Warnicke, "Wireshark user's guide," *Interface*, vol. 4, no. 6, p. 1, 2004.

[59] H. Li and S. Qin, "Optimization and implementation of industrial control system network intrusion detection by telemetry analysis," in *2017 3rd IEEE International Conference on Computer and Communications (ICCC)*, IEEE, 2017, pp. 1251–1254.

[60] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[61] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[62] W. Tang, G. Long, L. Liu, T. Zhou, J. Jiang, and M. Blumenstein, "Rethinking 1d-cnn for time series classification: A stronger baseline," *arXiv preprint arXiv:2002.10061*, 2020.

[63] Y. Goldberg and O. Levy, "Word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method," *arXiv preprint arXiv:1402.3722*, 2014.

[64] H. Hasselt, "Double q-learning," *Advances in Neural Information Processing Systems*, vol. 23, pp. 2613–2621, 2010.

[65] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[66] M. Arjovsky, S. Chintala, and L. Bottou, *Wasserstein gan*, 2017. arXiv: 1701.07875 [stat.ML].

[67] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[68] J. Wu, Z. Huang, J. Thoma, D. Acharya, and L. Van Gool, "Wasserstein divergence for gans," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 653–668.

[69] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, IEEE, 2004, p. 31.

[70] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," *CoRR*, vol. abs/1812.04948, 2018. arXiv: 1812.04948. [Online]. Available: http://arxiv.org/abs/1812.04948.

[71] M. Ring, D. Schlör, D. Landes, and A. Hotho, "Flow-based network traffic generation using generative adversarial networks," *Computers & Security*, vol. 82, pp. 156–172, 2019.

[72] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural networks for perception*, Elsevier, 1992, pp. 65–93.

[73] M. Al-Qatf, Y. Lasheng, M. Al-Habib, and K. Al-Sabahi, "Deep learning approach combining sparse autoencoder with svm for network intrusion detection," *IEEE Access*, vol. 6, pp. 52 843–52 856, 2018.

[74] E. Min, J. Long, Q. Liu, J. Cui, Z. Cai, and J. Ma, "Su-ids: A semi-supervised and unsupervised framework for network intrusion detection," in *International Conference on Cloud Computing and Security*, Springer, 2018, pp. 322–334.

[75] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *arXiv preprint arXiv:1206.2944*, 2012.

[76] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, Springer, 2011, pp. 507–523.

[77] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark datasets for intrusion detection," *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.

[78] R. E. Schapire, "Explaining adaboost," in *Empirical inference*, Springer, 2013, pp. 37–52.

[79] M. Sarmad, H. J. Lee, and Y. M. Kim, "Rl-gan-net: A reinforcement learning agent controlled gan network for real-time point cloud shape completion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5898–5907.

Appendix A:

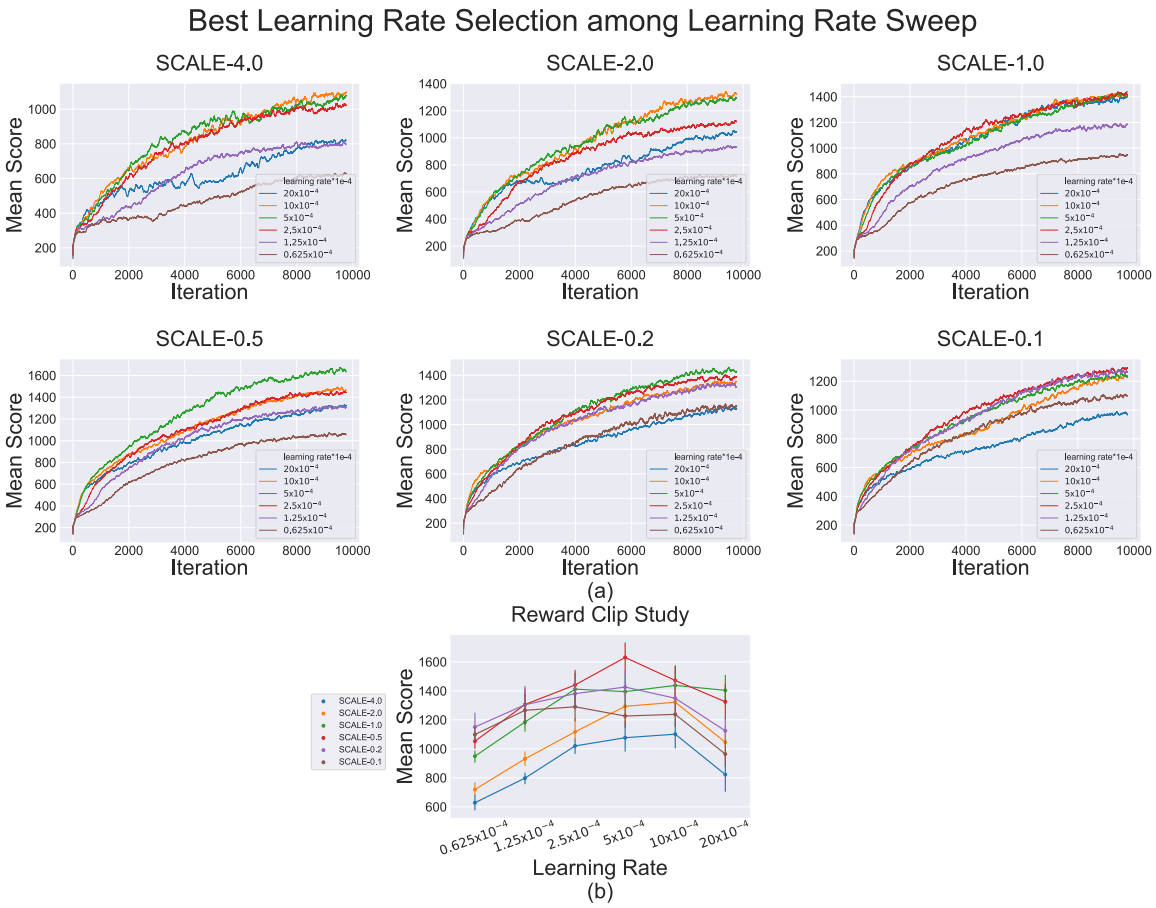


Figure A.1: All results are averaged over 30 runs with different random seeds. **(a)**: Mean score curves at different learning rates and clipping scales. **(b)**: Bell-shaped curves of the last-iteration mean scores for each scale. Our best learning rate selection is based on **(a)** and **(b)**.