Robust and Adaptive Decision-Making: A Reinforcement Learning Perspective

Wanqi Xue

School of Computer Science and Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

16/06/2023

Date

Wanqi Xue

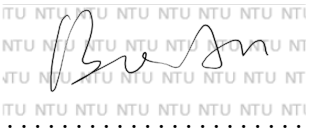
Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

16/06/2023

.....

Date



.....

Prof. Bo An

Authorship Attribution Statement

This thesis contains material from 5 paper(s) published in the following peer-reviewed conferences in which I am the first author.

Chapter 3 is published as **Wanqi Xue**, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, Chai Kiat Yeo. Solving large-scale extensive-form network security games via neural fictitious self-play. *Proceedings of the 30th International Joint Conference on Artificial Intelligence* (**IJCAI**), 2021.

The contributions of the co-authors are as follows:

- Prof. Bo An provided the initial project direction.
- Youzhi Zhang and I proposed the formulation of the problem.
- I proposed the algorithms, performed the experimental work, and wrote the manuscript.
- The manuscript was revised by Prof. Bo An and Prof. Chai Kiat Yeo.
- Shuxin Li, Xinrun Wang and Youzhi Zhang discussed with me about the details of the implementation.

Chapter 4 is published as **Wanqi Xue**, Bo An, Chai Kiat Yeo. NSGZero: Efficiently learning non-exploitable policy in large-scale network security games with neural monte carlo tree search. *Proceedings of the 36th AAAI Conference on Artificial Intelligence* (**AAAI**), 2022.

The contributions of the co-authors are as follows:

- Prof. Bo An proposed the problem.
- I proposed the algorithms, performed the experimental work, and wrote the manuscript.
- Prof. Bo An provided critical comments to the manuscript.
- The manuscript was revised by Prof. Bo An and Prof. Chai Kiat Yeo.

Chapter 5 is published as **Wanqi Xue**, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, Chai Kiat Yeo. Mis-spoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. *Proceedings of the 21th International Joint Conference on Autonomous Agents and Multi-Agent Systems* (**AAMAS**), 2022.

The contributions of the co-authors are as follows:

- Wei Qiu and I proposed the problem.
- I proposed the algorithms. Wei Qiu and I performed the experimental work. I wrote the manuscript.

- Prof. Bo An and Prof. Zinovi Rabinovich provided insightful comments to the manuscript.
- The manuscript was revised together by Prof. Bo An, Prof. Zinovi Rabinovich, Prof. Svetlana Obraztsov, and Prof. Chai Kiat Yeo.

Chapter 6 is published as **Wanqi Xue**, Qingpeng Cai, Ruohan Zhan, Dong Zheng, Peng Jiang, Kun Gai, Bo An. ResAct: Reinforcing long-term engagement in sequential recommendation with residual actor. *Proceedings of the 11th International Conference on Learning Representations* (**ICLR**), 2023.

The contributions of the co-authors are as follows:

- Prof. Bo An and Peng Jiang proposed the problem.
- I proposed the algorithms, performed the experimental work, and wrote the manuscript.
- Prof. Bo An and Qingpeng Cai provided insightful comments to the manuscript.
- The manuscript was revised together by Qingpeng Cai, Ruohan Zhan, Dong Zheng, Peng Jiang, Kun Gai, and Prof. Bo An.
- Qingpeng Cai and Ruohan Zhan discussed with me about the details of the implementation.

Chapter 7 is published as **Wanqi Xue**, Qingpeng Cai, Zhenghai Xue, Shuo Sun, Shuchang Liu, Dong Zheng, Peng Jiang, Kun Gai, Bo An. PrefRec: Recommender systems with human preferences for reinforcing long-term user engagement *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (**KDD**), 2023.

The contributions of the co-authors are as follows:

- Qingpeng Cai and I proposed the problem.
- I proposed the algorithms, performed the experimental work, and wrote the manuscript.
- The manuscript was revised together by Qingpeng Cai, Shuchang Liu, Dong Zheng, Peng Jiang, Kun Gai, and Prof. Bo An.
- Zhenghai Xue, and Shuo Sun helped me in writing.

16/06/2023

Date

Wanqi Xue

Acknowledgements

As I reach the end of the long journey as a Ph.D. student, I am filled with gratitude for the people who have taught, helped and inspired me over the past four years.

First and foremost, I express my sincere gratitude to my supervisors, Prof. Bo An and Prof Chai Kiat Yeo, without whose immense help and guidance I would not have reached this point. I am truly thankful for the incredible opportunity to pursue my Ph.D. under your supervision. From the beginning of this journey, your unwavering enthusiasm for research has left a lasting impression on me, and I will strive to maintain the same level of dedication. I deeply appreciate your patience and invaluable guidance throughout this process. Your insightful comments have consistently deepened my understanding of the research problem, and your tremendous assistance has enhanced my writing style. You have transformed my perception of research, emphasizing the importance of making fundamental contributions. I am also inspired by your dedication to addressing real-world issues that benefit society. You have not only served as a research role model, but also instilled in me a sense of responsibility, professionalism, and much more. I will forever be grateful for your supervision. My hope is that the relationships and connections fostered within your group will continue to grow stronger over time and endure indefinitely.

I would like to extend my gratitude to my Thesis Advisory Committee (TAC) members, Prof. Xiaohui Bei and Prof. Zinovi Rabinovich. The discussions I had with you have greatly contributed to the improvement of my research. I will always remember and value your valuable guidance and suggestions. Additionally,

I am also grateful to the many professors who have taught me courses during this journey, as their teachings have been immensely beneficial.

Throughout this lengthy journey, I have had the privilege of collaborating with exceptional researchers: Dr. Qingpeng Cai, Dr. Zhongwen Xu, Prof. Ruohan Zhan, Dr. Shuchang Liu, Dr. Dong Zheng, Dr. Peng Jiang, Dr. Kun Gai and Prof. Shuicheng Yan. I express my appreciation for your guidance, insights, and patient mentoring during our projects.

I consider myself fortunate to have shared my Ph.D. experience with a group of talented individuals: Xinrun Wang, Youzhi Zhang, Xu He, Lei Feng, Aye Phyu, Hongxin Wei, Renchunzi Xie, Rundong Wang, Wei Qiu, Yakub Cerny, Runsheng Yu, Yanchen Deng, Shuxin Li, Shuo Sun, Pengdeng Li, Shufeng Kong, Chaojie Wang, Pengjie Gu, Xinyu Cai, Yewen Li, Zhenghai Xue, Molei Qin, Longtao Zheng, Sheng Zang. I express my gratitude to each and every one of you for the moments of laughter, joy, and excitement we shared, as well as your generous assistance whenever I needed it. I will forever cherish the memories of working together to meet deadlines, conducting internal group meetings, going hiking, and enjoying group dinners.

Lastly, but most importantly, I want to express my deepest gratitude to my parents for their unwavering love and support, which have been invaluable throughout the journey.

Abstract

How to make decisions in complex and uncertain environments is a challenging and crucial task. Adversaries and perturbations in these environments disrupt existing policies, while the dynamic nature of the environments renders policies obsolete. Therefore, it is vital to learn robust policies capable of maintaining optimal performance in extreme scenarios and quickly adapting to changes. In this thesis, we focus on three real-world problem domains: network security games (NSGs), inter-agent communication and sequential recommendation. All of these domains necessitates robust and adaptive decision-making.

Our first emphasis is to learn robust defending policies in NSGs. In this domain, two algorithms are designed to improve scalability and data efficiency, respectively. First, we propose NSG-NFSP, a novel approach that aims to find Nash equilibria in NSGs on a large scale. NSG-NFSP employs deep neural networks to learn mappings from state-action pairs to values, representing either Q-values or probabilities. NSG-NFSP surpasses state-of-the-art algorithms in terms of scalability and solution quality. Second, we introduce NSGZero, a data-efficient learning method for acquiring non-exploitable policies in NSGs. NSGZero incorporates three neural networks, i.e., the dynamics network, the value network, and the prior network, to facilitate efficient Monte Carlo tree search (MCTS) in NSGs. Furthermore, we integrate decentralized control into neural MCTS, enabling NSGZero to handle NSGs with a large number of security resources. Extensive experiments on diverse NSGs with various graph structures and scales demonstrate the superior performance of NSGZero, even with limited training experiences.

The next focus of this thesis is on addressing the problem of robust communication in multi-agent communicative reinforcement learning (MACRL), which is a

topic that has been largely neglected before. We provide a formal definition of adversarial communication and propose an effective method for modeling message attacks in MACRL. We design a two-stage message filter to defend against message attacks. To enhance robustness, we formulate the adversarial communication problem as a two-player zero-sum game and design the algorithm, $\mathfrak{R}$-MACRL, to solve the game. Extensive experiments across different algorithms and tasks reveal the vulnerability of state-of-the-art MACRL methods to message attacks, while our proposed algorithm consistently restores the multi-agent cooperation and improves the robustness of MACRL algorithms under message attacks.

Furthermore, we investigate the problem of adapting recommendation policies to newly collected data for optimizing long-term user engagement in sequential recommendation. Two reinforcement learning algorithms are developed to learn policies with and without explicitly designed rewards. First, we introduce ResAct, an algorithm that improves the performance of recommender systems with pre-defined rewards. ResAct reconstructs the behavior of the online-serving policy and enhances it by adding a residual to the actions, resulting in a policy that closely aligns with the original policy but performs better. To improve the expressiveness and conciseness of state representations, we design two information-theoretical regularizers. Empirical evaluation demonstrates that ResAct outperforms previous state-of-the-art algorithms across all tasks. Additionally, we propose PrefRec which learns recommendation policies from preferences between users' historical behaviors rather than pre-defined rewards. This approach leverages the strengths of RL, such as optimizing long-term goals, while avoiding the complexities of reward engineering. PrefRec automatically learns a reward function from preferences and uses it to generate reinforcement signals for training the recommendation policy. We design an effective optimization method for PrefRec, utilizing an additional value function, expectile regression, and reward function pre-training to enhance performance. Experimental results highlight the significant performance improvements of PrefRec over the current state-of-the-art across various long-term user engagement optimization tasks.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

Making decisions in complex and uncertain environments is a challenging and crucial task. In reality, these environments can introduce adversaries or perturbations that disrupt our policies. Furthermore, the dynamic nature of environments leads to the obsolescence of existing policies. Therefore, it is vital to learn robust policies capable of maintaining optimal performance in extreme scenarios and quickly adapting to new changes.

This thesis systematically examines the problem of learning robust and adaptive policies from a reinforcement learning perspective. We specifically focus on three targeted problem domains: i) optimal defending policy in network security games (NSGs); ii) robust inter-agent communication in cooperative games; and iii) improving long-term user engagement in sequential recommendation systems. To address these challenges, we propose several novel reinforcement learning algorithms.

First, we introduce NSG-NFSP, which aims to find Nash equilibria in extensive-form NSGs on a large scale. The learned policy is robust against adversarial attacks and ensures good performance even in the worst-case scenarios. Second, to enhance data efficiency, we propose NSGZero, an algorithm that utilizes neural

Monte Carlo tree search to solve NSGs. NSGZero surpasses state-of-the-art algorithms in terms of data efficiency, achieving superior performance even with limited training experiences. Third, we investigate the problem of robust communication in multi-agent communicative reinforcement learning (MACRL). Our research reveals that many existing MACRL methods are vulnerable to message attacks. To address this vulnerability, we present $\mathfrak{R}$-MACRL, an algorithm that consistently restores multi-agent cooperation and enhances the robustness of MACRL algorithms against message attacks. Fourth, we explore the adaptation of recommendation policies to newly collected data for optimizing long-term user engagement in sequential recommendation systems. We propose ResAct, a method that efficiently improves local recommendation policies using a residual actor, thereby enhancing long-term user engagement. Lastly, we introduce PrefRec, a novel approach that enables reinforcement learning (RL) recommender systems to learn from preferences between users' historical behaviors, rather than explicitly defined rewards. This approach leverages the strengths of RL, such as optimizing long-term goals, while circumventing the complexities associated with reward engineering.

Decision-making in complex and uncertain environments presents significant challenges, especially in today's fast-paced and ever-changing world. In this thesis, we undertakes a systematic exploration of the challenges, employing a reinforcement learning perspective to develop effective solutions. The proposed algorithms contribute to the advancement of machine learning techniques and have the potential to drive improvements in network security games, inter-agent communication, and sequential recommender systems.

1.2 Targeted Problem Domains

In this thesis, we focus on designing efficient and effective algorithms to make decisions in highly uncertain real-world environments. The learned policy should be robust to perturbations or adversaries, and could be adaptive to continuously

evolving environments. Our research spans over a number of domains, as described below.

- **Optimal Defending Policy in Network Security Games:** How resources are deployed to secure critical targets in networks can be modelled by Network Security Games (NSGs). Recently, NSGs have attracted extensive attention because many real-world security problems, such as infrastructure protection, wildlife conservation and traffic enforcement, can be boiled down to NSGs. In NSGs, a defender, controlling several security resources, aims to interact with an attacker. Since the attacker can dynamically and adversarially adjust its escaping strategy to avoid being caught, the defending policy should be robust to ensure good performance under the worst situation. If we formulate a NSG as a two-player zero-sum game, the learning objective is equivalent to approximate a Nash equilibrium policy for the defender.

- **Robust Inter-agent Communication in Cooperative Games:** Apart from competitive game like NSGs, cooperative game is another type of games in which agents behave cooperatively to achieve common goals. Recent studies in multi-agent reinforcement learning (MARL) have demonstrated that inter-agent communication can greatly improve cooperation between agents. However, the spreading messages sent by agents cannot be always reliable. There could be one or more malicious agents that attempt to disrupt multi-agent cooperation by manipulating messages. How to effectively learn a robust communication policy with high tolerance to adversaries is an importance problem.

- **Improving Long-term User Engagement in Sequential Recommendation:** In reality, the environments could be always evolving over time. How to learn a policy which is adaptive to newly collected data is important in real-world applications. We study sequential recommendation for this problem. In sequential recommendation, the recommender system keeps feeding items to users until they leave the platform. Since the interests, preferences and

concerns of users always change with time, a recommendation agent should be able to quick adapt to the changes. We focus on improving long-term user engagement rather than immediate engagement because long-term user engagement is a more desired metric in sequential recommendation.

1.3 Main Contributions

In this thesis, we made several significant contributions for robust and adaptive decision-making problem. We study the problem from a reinforcement learning perspective. The major contributions are summarized as follows.

First, we propose a novel learning paradigm, NSG-NFSP, to learn robust defending policy in large-scale extensive-form NSGs based on Neural Fictitious Self-Play (NFSP). The novelties of NSG-NFSP include: i) reformulating the best response (BR) policy network in NFSP to be a mapping from action-state pair to action-value, to make the calculation of BR possible in NSGs; ii) converting the average policy network of an NFSP agent into a metric-based classifier, helping the agent to assign distributions only on legal actions rather than all actions; iii) enabling NFSP with high-level actions, which can benefit training efficiency and stability in NSGs; and iv) leveraging information contained in graphs of NSGs by learning efficient graph node embeddings. The algorithm significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

Second, we design a novel model-based reinforcement learning method, NSGZero, to learn a non-exploitable policy in NSGs. NSGZero is able to significantly improve data efficiency in NSGs. It works by performing planning with neural Monte Carlo Tree Search (MCTS). There are mainly three key contributions in this work. First, we design deep neural networks (DNNs) to perform neural MCTS in NSGs. Second, we enable neural MCTS with decentralized control, making NSGZero applicable to NSGs with many resources. Third, we provide an efficient learning paradigm, to achieve joint training of the DNNs in NSGZero. Compared to state-of-the-art

algorithms, the proposed method achieves significantly better data efficiency and scalability.

Third, we systematically explore the problem of adversarial communication in multi-agent systems. We propose $\mathfrak{R}$-MACRL for improving robustness in multi-agent communication. Several contributions are made in this work. First, we propose an effective method to perform attacks in MACRL, by learning a model to generate optimal malicious messages. Second, we develop a defence method based on message reconstruction, to maintain multi-agent coordination under message attacks. Third, we formulate the adversarial communication problem as a two-player zero-sum game and propose a game-theoretical method to improve the worst-case defending performance. Empirical results demonstrate that many state-of-the-art multi-agent communication methods are vulnerable to message attacks, and the proposed method can significantly improve their robustness.

Fourth, we propose ResAct which seeks to optimize long-term engagement in sequential recommendation. ResAct learns a policy that is close to, but better than, existing recommendation policy. In this way, we can collect sufficient data near the learned policy so that state-action values can be properly estimated, and there is no need to perform online interaction. ResAct optimizes the policy by first reconstructing the online behaviors and then improving it via a **Res**idual **Act**or. To extract long-term information, ResAct utilizes two information-theoretical regularizers to confirm the expressiveness and conciseness of the features. We conduct experiments on a benchmark dataset and a large-scale industrial dataset which consists of tens of millions of recommendation requests. Experimental results show that our method significantly outperforms the state-of-the-art baselines in various long-term engagement optimization tasks.

Fifth, we propose a novel paradigm, recommender systems with human preferences (or **Pre**ference-based **Rec**ommender systems), which allows RL recommender systems to learn from preferences about users' historical behaviors rather than explicitly defined rewards. With PrefRec, we can fully exploit the advantages of RL in

optimizing long-term goals, while avoiding complex reward engineering. PrefRec uses the preferences to automatically train a reward function in an end-to-end manner. The reward function is then used to generate learning signals to train the recommendation policy. Furthermore, we design an effective optimization method for PrefRec, which uses an additional value function, expectile regression and reward model pre-training to improve the performance. We conduct experiments on a variety of long-term user engagement optimization tasks. The results show that PrefRec significantly outperforms previous state-of-the-art methods in all the tasks.

1.4 Outline of the Thesis

The organization of this thesis is as follows. Chapter 2 provides background information and discusses related work relevant to this thesis. Chapter 3 considers the problem of large-scale extensive-form NSGs where a defending policy should be learned to play against an adversarial attacker. Chapter 4 focuses on improving the data efficiency when learning a defending policy in NSGs. Chapter 5 focuses on achieving robust communication in multi-agent reinforcement learning. Chapter 6 focuses on optimizing long-term user engagement from online-logged data in sequential recommendation. Chapter 7 focuses on optimizing long-term user engagement from human preferences rather than delicately designed rewards. Chapter 8 summarise this thesis and provides possible future directions.

Chapter 2

Literature Review

In this chapter, we introduce the previous works that are relevant to our methods. Our research span over a number of domains, so we review the literatures in each targeted problem domain separately.

2.1 Deep Reinforcement Learning for Security Games

The objective of security games is to find a Nash Equilibrium (NE) policy for the defender. Traditionally, the defender's policy is computed by programming-based NE-solving techniques, e.g., the incremental strategy generation algorithms, which start from a restricted game and iteratively expand it until convergence. One important requirement of these approaches is that all of the attacking paths are enumerable, which is to ensure that there is at least a terminal state in the restricted game for each attacking path to make the incremental strategy generation algorithm converge. In large-scale NSGs, e.g., real-world road networks, where the number of attacking paths are prohibitively large, programming-based NE-solving approaches tend to lose effectiveness. Recently, applying deep learning, especially

deep reinforcement learning (DRL), to solve security games has received extensive attention. In common game-theoretical frameworks such as NFSP and PSRO, they typically involve the computation of best response to the opponent. DRL formulates the computation of best response as a Markov Decision Process (MDP), where an MDP is defined by a 5-tuple $\langle \mathcal{S}, \mathcal{A}, r, P, \gamma \rangle$. $\mathcal{S}$ is a set of states, $\mathcal{A}$ is a set of actions, $r(s, a)$ is a reward function, $P(s'|s, a)$ is a transition function, and γ is a discount factor. The goal of DRL is to find the policy π, mapping states to actions that maximize the expected discounted reward. Integrating DRL into those game-theoretical frameworks has significantly improve the scalability of the methods. However, most applications of them are limited in domains with small discrete action spaces. Applying game-theoretical frameworks like NFSP to other types of action spaces has received extensive attention. OptGradFP [1] firstly introduces fictitious play to continuous action spaces. It applies policy gradient algorithm over policy network which predicts parameters of continuous action distributions. DeepFP [2] generalizes OptGradFP by using flexible implicit density approximators. Currently, how to solve games like NSGs, whose action spaces are large and legal actions vary significantly with states, remains unexplored. The main challenge is that the output of the two DNNs in NFSP cannot cover all actions, and if just covering legal actions, they will lack consistent semantics.

Another problem of DRL methods is data inefficiency. Exiting model-free RL methods rely on a large number of sampling for learning a good best response policy. However, in large-scale extensive-form security games, each sampling could be very time consuming, leading to infeasible approximation of the best response policy. We consider to apply model-based RL methods, especially Monte Carlo Tree Search (MCTS), to address the problem of data inefficiency. Monte Carlo Tree Search [3] is a planning method which explores possible future states and actions by querying a simulator or model of the environment. At each decision point, MCTS repeatedly performs multiple simulations, to evaluate the probability of choosing each available action. There have been many attempts to combine

MCTS with neural network function approximations to solve complex games, notably the AlphaGo series. AlphaGo [4], the first algorithm that defeats human professional players in the full-sized game of Go, conducts lookahead searches by using a policy network to narrow down the decision to high-probability moves and using a value network to evaluate state values in the search tree. AlphaGo Zero [5] achieves superhuman performance purely from random initialization, without any supervision or use of expert data. AlphaZero [6] generalizes its predecessor into a single algorithm which can master many challenging domains, including chess, shogi and Go. For the aforementioned algorithms, they assume complete access to the rules of the game. MuZero [7] lifts this restriction and uses neural networks to approximate the transition function and the reward function of the environment. Recently, Sampled MuZero [8] extends MuZero to complex action space by performing policy improvement and evaluation over small subsets of sampled actions. Despite the great breakthroughs, all the previously discussed approaches have focused on controlling a single agent, while whether and how neural MCTS can be applied to multi-agent scenarios like NSGs remain unexamined.

2.2 Multi-Agent Communicative Reinforcement Learning

There has been extensive research on encouraging communication between agents to improve performance on cooperative or competitive tasks. Among the recent advances, some design communication mechanisms to address the problem of when to communicate [9–11]; other lines of works, e.g., TarMac [12], focus on who to communicate. These works determine the two fundamental elements in communication, i.e., the message sender and the receiver. Apart from the two elements, the message itself is another element which is crucial in communication, i.e., what to communicate: Jaques et al. propose to maximize the social influence of messages [13]. Kim et al. encode messages such that they contain the intention of agents [14].

Some other works learn to send succinct messages to meet the limitations of communication bandwidth [15–17]. Despite significant progress in MACRL, if some agents are adversarial and send maliciously designed messages, multi-agent coordination will disintegrate rapidly as these messages propagate. Such adversarial problems in RL have also been widely studied recently. SA-MDP [18] characterizes the problem of decision making under adversarial attacks on state observations. Lin et al. propose two tactics of attacks, i.e., the strategically-timed attack and the enchanting attack, which attack by injecting noise to states and luring the agent to a designated target [19]. Gleave et al. consider the problem of taking adversarial actions that change the environment and consequentially change the observation of agents [20]. ATLA [21] propose to train the optimal adversary to perturb state observations and improve the worst-case agent reward. The settings of these works are different from the proposed method as we consider the multi-agent scenario and restrict the attacking approach to adversarial messages, which makes the detection of anomalies difficult. Tu et al. propose to attack multi-agent communication [22]. However, their focus is on the representation-level, whereas we focus on the policy-level. Recently, there are some works considering a similar setting as ours [23, 24]. However, they either focus on random attacks in specific competitive games or the defence of specific communication methods.

2.3 Reinforcement Learning for Sequential Recommendation

Sequential recommendation has been used to model real-world recommendation problems where the browse length is not fixed [25]. Many existing works focused on encoding users' previous records with various neural network architectures. For example, GRU4Rec [26] utilizes Gated Recurrent Unit to exploit users' interaction histories; BERT4Rec [27] employs a deep bidirectional self-attention structure to learn sequential patterns. However, these works focus on optimizing immediate engagement like click-through rates. In fact, long-term user engagement is

a more important metric because it directly affects some operational indicators such as DAU and dwell time. One major difficulty in promoting long-term user engagement is the lack of consistent and informative reward signals. Reward signals can be sparse and change over time, making it difficult to accurately predict and respond to user needs. Zhang et al. [28] augment the sparse rewards by re-weighting the original feedbacks with counterfactual importance sampling. Wu et al. [29] consider user click as immediate reward and user's return time as a hint on long-term engagement. It also assumes a stationary distribution of candidates for recommendation. In contrast, Zhao et al. [30] empirically identify the problem of non-stationary users' taste distribution and propose distribution-aware recommendation methods. Other researches focus on maximizing the diversity of recommendations as an indirect approach to optimize long-term engagement. For example, Adomavicius et al. [31] propose a graph-based approach to maximize diversity is based on maximum flow. Ashton et al. [32] propose that high recommendation diversities is related to long-term user engagement. Apart from diversities, Cai et al. [33] conduct large-scale empirical studies and propose several surrogate criteria for optimizing long-term user engagements, including high-quality consumption, repeated consumption, etc.

Reinforcement learning allows an autonomous agent to interact with the environment and optimizes long-term goals from experiences, which is particularly suitable for tasks in recommender systems [34–39]. Shani et al. [36] first proposed to employ Markov Decision Process (MDP) to model the recommendation behavior. The subsequent researches focus on standard RL problems, e.g., exploration and exploitation trade-off in the context of recommendation systems [40, 41], together with model-based or simulator-based approaches to improve sample efficiency [37, 42]. Recently, there has been works on designing proper reward functions for efficient training of recommender systems. For example, Zou et al. [34] use a Q-network with hierachical LSTM to model both the instant and long-term reward. Ji et al. [43] model the recommendation process as a Partially Observable MDP and estimate the lifetime values of the recommended items. Zheng et al. [44] explicitly

model the future returns by introducing the user activeness score. Ie et al. [45] decompose the Q-function for tractable and more efficient optimization. There are also researches focusing on behavior diversity [46, 47] as a surrogate for the reward function.

Chapter 3

Learning Robust Policy against Adaptive Adversaries via Neural Fictitious Self-Play

Recently, how to secure networked infrastructures, e.g., urban city networks, transportation networks, and web networks, has received extensive attention [49–52]. In this chatper[1], we analyze the problem of deploying security resources to protect against an adaptive attacker in networked domains, which can be modeled as Network Security Games (NSGs). We focus on how to learn a robust policy against adaptive adversaries to maximize the worst case utility at a large scale. A realistic game setting is considered where the players interact sequentially (extensive-form) and the defender makes decisions based on real-time information about the attacker [51]. The objective of NSGs is to find a Nash Equilibrium (NE) policy for the defender. Traditionally, the defender's policy is computed by programming-based NE-solving techniques, e.g., the incremental strategy generation algorithms [51, 53], which start from a restricted game and iteratively expand it until convergence. One important requirement of these approaches is that all of the attacking

[1]The work in this chapter has been published as **Wanqi Xue**, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, Chai Kiat Yeo. Solving large-scale extensive-form network security games via neural fictitious self-play [48]. *Proceedings of the 30th International Joint Conference on Artificial Intelligence* (**IJCAI**), pp.3713-3720, 2021.

paths are enumerable, which is to ensure that there is at least a terminal state in the restricted game for each attacking path to make the incremental strategy generation algorithm converge. However, in large-scale NSGs, e.g., real-world road networks, where the number of attacking paths are prohibitively large, programming-based NE-solving approaches tend to lose effectiveness. For example, in the real world, the number of possible attacking paths could be more than 6.6^{18} [49], which will make it impossible to enumerate all of them due to the limited memory.

Recently, there has been an increasing interest in combining Deep Learning (DL) with game theory for finding NE [54–56]. DL-based NE-solving algorithms use Deep Neural Networks (DNNs) to learn states-to-actions mappings for approximating strategies, counterfactual regrets, etc. They usually execute in a sampling style and are able to capture the structure of underlying enormous state spaces by leveraging the strong representation ability of DNNs, making them potential for solving large-scale and complex real-life problems.

Unfortunately, existing DL-based NE-solving algorithms are unable to solve large NSGs. In NSGs, players occupy nodes of graphs, e.g., road networks, and can only move to their adjacency nodes (legal actions) at each step. A consequence is that legal actions change with players' current positions or states. To approximate states-to-actions mappings, a naive implementation is to set the output dimension of DNNs equal to the maximum number of legal actions, with each output corresponding to one legal action though the action changes with states. However, this naive setting yields poor results in practice because each output of DNNs has no consistent semantics [57]. On the other hand, it is infeasible to set the output dimension of DNNs equal to the number of all actions and use masks to filter out all illegal actions at each state. The reason is that, in NSGs, the defender's action space is prohibitively large because it is a combination of all sub-action spaces of the defender's security resources. For example, when there are four security resources deployed on a road network with one hundred nodes, the defender's action set has 100^4 elements. Obviously, we cannot define the output of DNNs at such a scale.

In this chapter, we propose a novel learning paradigm, NSG-NFSP, for approximating an NE policy in large-scale extensive-form NSGs. The method is based on Neural Fictitious Self-Play (NFSP), which intrinsically ensures its convergence. Our main contributions are fourfold. Firstly, we propose to train the best response (BR) policy network in NFSP to be a mapping from action-state pair to action-value, which avoids the aforementioned unachievable requirement where the output of DNN must cover the overall action spaces of NSGs. Secondly, we convert the average policy network into a metric-based classifier, helping an NFSP agent to assign distributions only on legal actions rather than all actions. Thirdly, we propose a framework to enable NFSP with high-level actions, which can enhance training efficiency and stability in NSGs. Finally, we propose to learn efficient graph node embeddings by *node2vec*, to leverage information contained in the graphs of NSGs. We conduct experiments in NSGs played on synthetic networks and real-world road networks. Our algorithm significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

3.1 Network Security Games

Network Security Games (NSGs) are proposed to model the problem of deploying a limited number of security resources to protect against an adaptive attacker in networked domains [49]. For example, the police department dispatches collaborating police officers to prevent a criminal from escaping or attacking in urban cities [49, 50]. An NSG is played on a graph $G = (V, E)$ which consists of a set of edges E and a set of nodes V. There are two players, the defender and the attacker. The attacker, starting from one of the source nodes $v_0^{att} \in V_s \subset V$, tries to reach one of the target nodes $\chi \in V_t \subset V$ within a fixed time horizon T[2]. The defender controls m security resources and dynamically allocates them to catch the attacker before the later reaches any of the targets. We assume that the defender can observe the real-time location of the attacker, with the help of advanced tracking

[2]The target nodes represent destinations to be attacked or exits to escape.

technologies such as GPS, but the attacker can only see the initial locations of the security resources [51]. We model NSGs as extensive-form games, where players make decisions sequentially.

At time step t, the attacker's state s_t^{att} is a sequence of nodes visited, i.e., $s_t^{att} = \langle v_0^{att}, v_1^{att}, \ldots, v_t^{att} \rangle$. $\mathcal{A}_{att} = V$ is the set of attacker actions and $\mathcal{A}_{att}(s_t^{att}) = \{v_{t+1}^{att} | (v_t^{att}, v_{t+1}^{att}) \in E\}$ is the set of legal attacker actions at s_t^{att}. For the defender, its state s_t^{def} consists of the attacker's action history (state) and its resources' current locations, i.e., $s_t^{def} = \langle s_t^{att}, l_t^{def} \rangle$ where $l_t^{def} = \langle v_t^0, \ldots, v_t^{m-1} \rangle$. This design is based on the assumption that security resources able to track the real-time location of the attacker to improve interdiction efficiency. l_t^{def} is adjacent to resource location $\langle v_{t+1}^0, \ldots, v_{t+1}^{m-1} \rangle$ if $(v_t^R, v_{t+1}^R) \in E, \forall R \in \{0, \ldots, m-1\}$. We denote $Adj(l_t^{def})$ as the set of all resource locations which are adjacent to l_t^{def}. With this concept, we can define the set of legal defender actions at s_t^{def} as $\mathcal{A}_{def}(s_t^{def}) = Adj(l_t^{def})$ and $\mathcal{A}_{def} = V^m$ is the set of defender actions. For both the attacker and the defender, illegal actions at state s are those actions in $\mathcal{A}$ but not in $\mathcal{A}(s)$. A policy $\pi(s) = \Delta(\mathcal{A}(s))$ describes a player's behavior, where $\Delta(\cdot)$ represents a probability distribution. Both players act simultaneously at each step by sampling actions from their policies: $a \sim \pi(s)$. The attacker is caught if he and at least one of the security resources are in the same node at the same time. A game ends when the attacker either reaches any of the targets within the maximum allowed time or is caught. If the defender successfully protects the targets within the time horzion T, she will be awarded with a positive unit utility (an end-game reward) $u_{def} = 1$. Otherwise, no award will be given to the defender. The game is zero-sum, so $u_{att} = -u_{def}$. The worst-case defender utility $\mathcal{E}_{def}(\pi_{def})$ is the expected payoff for the defender (with policy π_{def}) given that the attacker best responds to it. Formally, $\mathcal{E}_{def}(\pi_{def}) = \min_{\pi_{att}} \mathbb{E}\big[u_{def} | \pi_{def}, \pi_{att}\big]$. π_{def}^* is optimal if $\pi_{def}^* \in \arg\max_{\pi_{def}} \mathcal{E}_{def}(\pi_{def})$. The optimality for the attacker is defined similarly. A Nash Equilibrium (NE) is reached if and only if both the defender and the attacker perform the optimal policy. In NSGs, the optimization objective is to learn an NE policy for the defender.

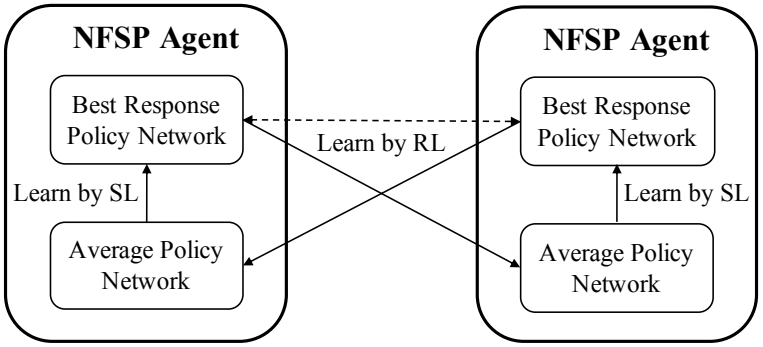


FIGURE 3.1: The NFSP framework.

3.2 Solution Algorithm: NSG-NFSP

In this section, we introduce a novel learning paradigm, NSG-NFSP, for solving large-scale NSGs. Note that despite the method being proposed for solving NSGs, the basic ideas can be easily applied to other games, especially those whose legal action spaces vary significantly with states.

3.2.1 Game-theoretical Framework: Neural Fictitious Self-Play

Fictitious play (FP) [58] is a widely used game-theoretic algorithm for learning NE from self-play. In FP, each agent repeatedly plays with its opponent's past average policy and best responds against it. **Fictitious Self-Play (FSP)** [59] extends FP from normal form to extensive form and realizes it in a sampling and machine learning style. In our method, we adopt **Neural Fictitious Self-Play (NFSP)** [54] which combines FSP with neural network function approximation. As in Figure 3.1, each NFSP agent consists of two neural networks, i.e., the best response (BR) policy network and the average policy network. The BR policy network is trained by reinforcement learning (RL) algorithms, e.g., DQN [60], to maximize the expected total rewards. It considers the opponent as part of the environment. The average policy network is trained to approximate the past average behaviours of the BR policy network by supervised learning (SL). It outputs the

probabilities of actions chosen, historically, by the BR policy network. Each NFSP agent behaves according to a mixture of its BR policy and average policy (with a mixing constant η which is called anticipatory parameter). Most applications of NFSP are limited in domains with small discrete action spaces. Applying NFSP to games like NSGs, whose action spaces are large and legal actions vary significantly with states, remains unexplored. The main challenge is that the output of the two DNNs in NFSP cannot cover all actions, and if just covering legal actions, they will lack consistent semantics. With many recent works focusing on adapting deep RL to large discrete action spaces [61, 62], we propose our solution for the BR policy network based on DRRN [61]. It models Q-values as an inner product of state-action representation pairs. DRRN is designed for natural languages domain. We adapt it to make it suitable for NSGs whose actions are defined on graph nodes. For the average policy network, our solution is inspired by metric-based few-shot learning [63]. We propose to address the problem by transforming actions and states to a space where the probabilities of actions can be determined via comparing some metrics, e.g., cosine similarity.

3.2.2 Approximating Best Response Policy

It is essential to properly approximate the BR policy in NFSP because the final (average) policy is supervisedly trained from the behavior of the BR policy network. The BR policy network in the vanilla NFSP algorithm learns a mapping from states to action-values, which internally requires the DNN's outputs to cover all possible actions. However, for games like NSGs, it is impossible to meet this requirement because the overall action space is enormous. To address the problem, we propose to convert the BR policy network to be a mapping from state-action pairs to Q-values. Concretely, we use an action representation network and a state representation network, parameterized by θ^Q_α and θ^Q_β, to extract features from each legal action and state respectively, generating feature vectors h_a and h_s. The extracted features h_a and h_s are concatenated and sent to a fully connected network $f(h_a, h_s; \theta^Q_\gamma)$ with parameters θ^Q_γ to predict the action-value. We denote

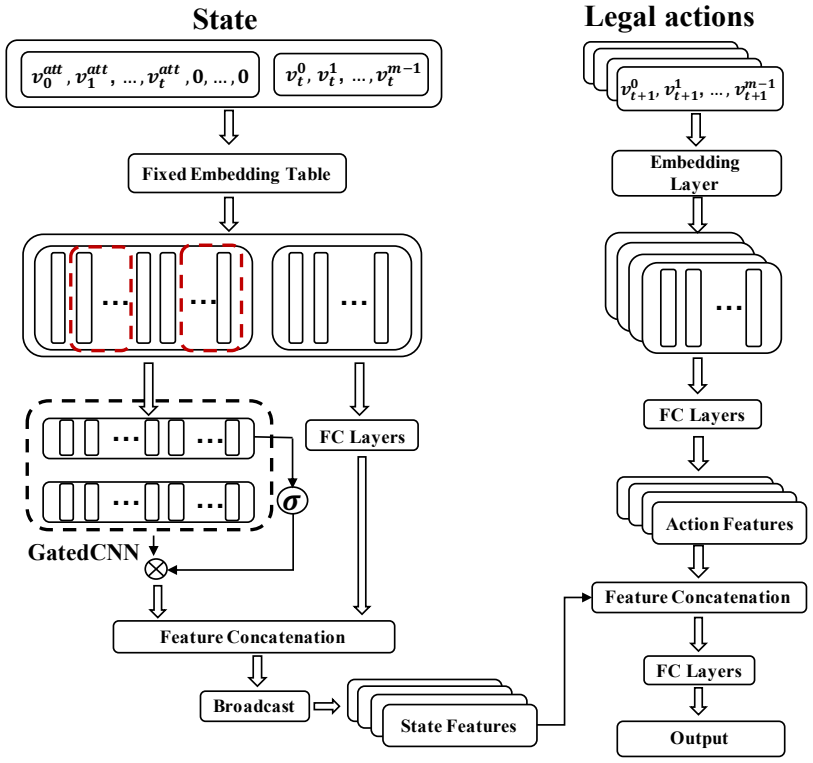


FIGURE 3.2: The network structure of the defender (the two red boxes indicate the convolutional blocks in GatedCNN).

$\theta^Q = \{\theta^Q_\alpha, \theta^Q_\beta, \theta^Q_\gamma\}$ as the parameters of the BR policy network. During training, an agent stores its experienced transition tuples, $(s, a, r, s', \mathcal{A}(s'))$, in a replay buffer $\mathcal{M}_{RL}$, where $r \sim R(\cdot|s, a)$ (reward function) is the immediate reward and $s' \sim P(\cdot|s, a)$ (transition function) is the next state. The BR network parameters θ^Q are optimized by minimizing the loss:

$$\mathcal{L}(\theta^Q) = \mathbb{E}_{s,a,r,s'}\Big[(r + \max_{a' \in \mathcal{A}(s')} Q(s', a'; \theta^{Q'}) - Q(s, a; \theta^Q))^2\Big] \tag{3.1}$$

where $\theta^{Q'}$ denotes the parameters of the target network. $\theta^{Q'}$ is periodically copied from θ^Q, while in other cases it is frozen to improve the stability of training.

Figure 3.2 presents the overall network architecture, NSG-BR, for the defender. We can design the network for the attacker similarly. Since the elements of states and actions are graph nodes, we firstly embed those graph nodes before they can be fed into neural networks. For the action representation network, we use a learnable embedding layer. For the state representation network, we pre-compute the

embeddings (the approach is introduced in Section 3.2.5). After embedding graph nodes, we need to extract features from the attacker's history. Taking into account the speed and effectiveness, we apply a structure similar to GatedCNN [64] to process these sequential data. Specifically, the sequential data padded to the maximum length is fed into two separate convolutional blocks which have identical structures. The output of one block is activated by the sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$, and the result serves as the gate to control the output of the other convolutional block. After extracting features for a state, the state feature vector is duplicated (broadcast) $|\mathcal{A}(s)|$ times (number of legal actions at the state) and concatenated with legal action features. Then the state-action pairs' features are passed to several fully connected layers to predict the final state-action values.

3.2.3 Approximating Average Policy

The average policy network in the vanilla NFSP algorithm is a classifier-like network whose output scales linearly with the cardinality of action set $\mathcal{A}$. In NSGs, assigning distributions over legal action set $\mathcal{A}(s)$, rather than the whole action set, is preferred. We propose to convert the average policy network into a metric-based classifier, transforming states and legal actions to a space where they can be compared by some metrics. We use fully-connected layers to learn the metric. The reason is that i) learnable metric is more representative compared to heuristic metric; and ii) we can reuse the network architecture as depicted in Figure 3.2. Similar to NSG-BR, the average policy network, NSG-AVG, has three parts, i.e., the action representation network, the state representation network and the metric network. We denote $\theta^{\Pi} = \{\theta^{\Pi}_{\alpha}, \theta^{\Pi}_{\beta}, \theta^{\Pi}_{\gamma}\}$ to be parameters of NSG-AVG, where $\theta^{\Pi}_{\alpha}, \theta^{\Pi}_{\beta}, \theta^{\Pi}_{\gamma}$ are the parameters of its three sub-networks respectively. NSG-AVG works similarly to NSG-BR, the main difference being that NSG-AVG assigns probabilities to legal actions rather than Q-values. Let $\Pi(\mathcal{A}(s)|s)$ be the output of NSG-AVG and $\hat{\Pi}(\mathcal{A}(s)|s)$ be the ground truth average BR behaviours. We measure the difference between these two distributions, $\Pi(\mathcal{A}(s)|s)$ and $\hat{\Pi}(\mathcal{A}(s)|s)$, by the expected Kullback-Leibler (KL) divergence. Then the parameters θ^{Π} of NSG-AVG can be

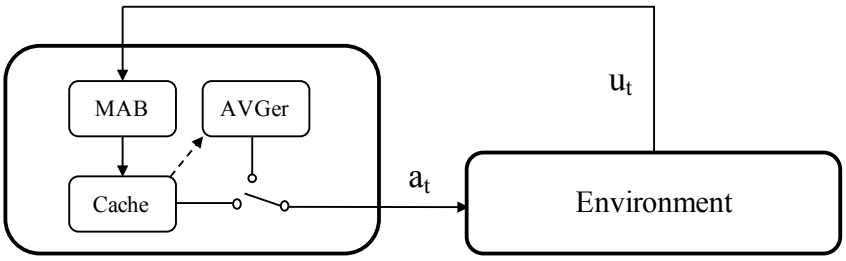


FIGURE 3.3: The architecture of NFSP with high-level actions.

optimized by minimizing the difference:

$$\begin{aligned}\mathcal{L}(\theta^{\Pi}) &= -\mathbb{E}_s\Big[\sum_{a\in\mathcal{A}(s)}\hat{\Pi}(a|s)\log\Big(\frac{\Pi(a|s)}{\hat{\Pi}(a|s)}\Big)\Big] \\ &= -\mathbb{E}_{s,a}\Big[\log\Big(\frac{\Pi(a|s)}{\hat{\Pi}(a|s)}\Big)\Big]\end{aligned} \tag{3.2}$$

The denominator in Eq.(3.2) can be omitted since it does not depend on θ^{Π}. Then the optimization objective becomes minimizing the loss function:

$$\mathcal{L}(\theta^{\Pi}) = -\mathbb{E}_{s,a}\Big[\log\Big(\Pi(a|s)\Big)\Big] \tag{3.3}$$

An agent records its BR behaviours, i.e., (s, a), in a reservoir buffer $\mathcal{M}_{SL}$ which serves as an expanding dataset. It fits the dataset by applying gradient descent. According to the theoretical convergence of NFSP [54], the average policy network approximates an NE policy.

3.2.4 Enabling NFSP with High-Level Actions

During training, it usually takes many episodes for the BR policy network to approximate the BR policy. For example, in NSGs, the NFSP attacker will do a lot of unnecessary explorations in invalid paths, i.e., the paths which cannot reach any of the targets, before finding the optimal path. This will lead to training inefficiency and instability because i) other agents will play against this weak opponent for a long period; and ii) behaviours of non-BR policy will be used for training the average policy network. The problems can be mitigated if an NFSP agent makes

decisions on high-level actions (HLA). For instance, we can force the NFSP attacker in NSGs to make decisions on valid paths or source-target pairs (high-level actions) rather than the next-step location. The idea of HLA is similar to action abstractions [65, 66] and options in hierarchical RL [67].

To extend the NFSP framework so that an agent can decide on High-Level Actions (NFSP-HLA), we propose to use Multi-Armed Bandit (MAB), a widely used approach to optimize decisions between multiple options (actions), to model the BR policy for an NFSP-HLA agent. Each option of the MAB corresponds to a high-level action. We use a first-in-first-out (FIFO) buffer with length k to record the most recent k game results (utilities). The estimated action value for a high-level action ζ after n episodes becomes:

$$\hat{Q}_n(\zeta) = \frac{\sum_{j=\tau}^{n} u_j \mathbb{I}[\zeta_j = \zeta]}{\sum_{j=\tau}^{n} \mathbb{I}[\zeta_j = \zeta]} \tag{3.4}$$

where $\tau = \max(n - k, 1)$, u_j is the player's utility for the j-th episode, and $\mathbb{I}$ is binary indicator function. We design two auxiliary modules to fit the MAB best responsor into the framework of NFSP: i) the Averager (AVGer) module, which is used to measure the average policy, by counting the frequency of each high-level action; and ii) the Cache module, which is to temporarily store behaviours of the MAB. Data stored in the Cache is used to update the AVGer.

Learning Process As in Figure 3.3, before each episode, an NFSP-HLA agent samples its behaviour pattern, acting as either the MAB (with probability η) or the AVGer (with probability $1 - \eta$). If acting as the MAB, the agent chooses the high-level action with the largest estimated value and stores the high-level action in the Cache. Otherwise, the agent samples an high-level action according to the distribution in the AVGer. After confirming the high-level action, the agent interacts with the environment (containing the opponents) and receives an utility at the end of a game. Then the utility is used for training the MAB in accordance with Eq. (3.4). The Cache pours its records into the AVGer in a fixed frequency,

after which it clears itself. By using the Cache, we can avoid rapid changes in the AVGer, thus reducing instability.

Additional Exploration An NFSP-HLA agent does exploration by acting as the AVGer (not the MAB). Such mechanism may lead to suboptimality when some actions are dominated. For example, if the AVGer explores some good actions, the MAB is likely to choose them because of their high estimated values. Behaviours of the MAB will be recorded by the AVGer, further increasing these actions' occurrence frequency (selected by the AVGer). This may result in some actions appearing rarely, and the MAB cannot precisely estimate those actions' values. To overcome this, we design additional exploration for NFSP-HLA agent: if the agent does not act as the MAB, it will perform additional sampling to decide whether to explore or not. If the sampling result indicates exploration, the agent will act randomly and the transitions for this episode will not be recorded by the opponent; Otherwise, the agents will interact normally. Additional exploration confirms that each high-level action occurs enough times that the MAB can conduct action-value estimation.

3.2.5 Efficient Graph Node Embeddings

To leverage information, e.g., adjacency and connectivity, contained in graphs of NSGs, we propose to use *node2vec* [68], a semi-supervised learning algorithm, to learn representations for nodes. Concretely, we learn a mapping function to transform nodes to low-dimensional space of features which maximizes the likelihood of preserving graph structures. Let $g : V \to \mathbb{R}^D$ be the mapping function, we need to optimize:

$$\max_{g} \sum_{v \in V} \log Prob(N_O(v)|g(v)) \tag{3.5}$$

where $N_O(\cdot)$ is a randomized procedure that samples many different neighborhoods of a given node. Concretely, the procedure $N_O(\cdot)$ is parameterized by a sampling

strategy O:

$$O(v_t = x | v_{t-1} = u) = \begin{cases} \frac{U(u,x)}{Z} & \text{if } (u,x) \in E \\ 0 & \text{else} \end{cases} \tag{3.6}$$

where $U(\cdot, \cdot)$ is the unnormalized transition probability, and Z is the normalizing constant. The sampling strategy O could be, for example, Breadth-First Sampling (BFS), Depth-First Sampling (DFS), etc., generating different transition probabilities. We follow *node2vec* to use a biased 2^{nd} order random walk as the sampling strategy. In our unweighted road network graph,

$$U(u,x) = \alpha_{pq}(u,x) = \begin{cases} \frac{1}{p} & \text{if } d_{wx} = 0 \\ 1 & \text{if } d_{wx} = 1 \\ \frac{1}{q} & \text{if } d_{wx} = 2 \end{cases} \tag{3.7}$$

where p, q are two parameters of the random walk which control how fast the walk explores, w is the predecessor of u, and d_{wx} is the distance between nodes w and x.

After obtaining the mapping function g, we use it to embed nodes when extracting features from a state. For action representation, we use a learnable embedding layer. We apply this setting because it can keep flexibility in learning while leveraging graph information.

3.2.6 Overall Algorithm

We present the overall algorithm in Algorithm 1. In line 1, we calculate node embeddings by applying *node2vec* (introduced in Section 3.2.5). From lines 2 to 3, we do initialization for the defender and the attacker. In line 5, both the defender and the attacker sample to determine the behaviour for the current episode. From lines 6 to 8, the attacker samples to decide whether to explore in the current episode or not. From lines 9 to 10, the attacker determines the escaping path for the current episode. From lines 11 to 19, the defender collects transitions for

Algorithm 1: NSG-NFSP

Create *node2vec* embeddings $\mathcal{G}$ for nodes in the road network;
Initialize modules of the defender, including the BR policy network $Q(s, a|\theta^Q)$, the average policy network $\Pi(s, a|\theta^\Pi)$, and the corresponding replay memories, $\mathcal{M}_{RL}$ (circular buffer) and $\mathcal{M}_{SL}$ (reservoir buffer);
Initialize modules of the attacker, including the MAB, the Cache, and the AVGer;
for *episode* $\tau = 1$ *to* M **do**
    Both players sample to set behaviour for the current episode, either as the BR policy ($Q(s, a|\theta^Q)$, the MAB) or the average policy ($\Pi(s, a|\theta^\Pi)$, the AVGer);
    if *the attacker behaves as the AVGer* **then**
        The attacker samples to decide whether to explore;
    end
    The attacker samples an exit ζ_τ by following the distribution of its current behaviour (the MAB, the AVGer or Exploration);
    The attacker samples an escaping path to the selected exit;
    The defender observes initial state s_0^{def} and legal actions $\mathcal{A}_{def}(s_0^{def})$;
    for *time step* $t = 0$ *to* T **do**
        The defender inputs the state s_t^{def} and legal actions $\mathcal{G}(\mathcal{A}_{def}(s_t^{def}))$ into the network corresponding to its behaviour and samples an action a_t^{def};
        Execute action a_t^{def} in game and the defender observes reward r_{t+1}^{def}, state s_{t+1}^{def} and legal actions $\mathcal{A}_{def}(s_{t+1}^{def})$;
        The defender stores transition $(s_t^{def}, a_t^{def}, r_{t+1}^{def}, s_{t+1}^{def}, \mathcal{A}_{def}(s_{t+1}^{def}))$ in $\mathcal{M}_{RL}$;
        if *the defender is in the BR mode and the attacker is not in exploration* **then**
            The defender stores tuple (s_t^{def}, a_t^{def}) in $\mathcal{M}_{SL}$;
        end
    end
    Calculate attacker utility u_τ^{att} and the attacker stores tuple $(\zeta_\tau, u_\tau^{att})$ in the MAB;
    if *the attacker in the MAB mode* **then**
        The attacker stores ζ_τ in the Cache;
    end
    The defender periodically updates θ^Q with SGD on the loss described in Eq. (3.1);
    The defender periodically updates target network parameters $\theta^{Q'} \leftarrow \theta^Q$;
    The defender periodically updates θ^Π with SGD on the loss described in Eq. (3.3);
    The attacker periodically updates the MAB by following Eq. (3.4);
    The attacker periodically updates the AVGer using data stored in the Cache and clear the Cache;
end
Output: The defender's average policy network $\Pi(s, a|\theta^\Pi)$

training its policy networks. From lines 20 to 23, the attacker collects data for learning its policy. From lines 24 to 26, the defender updates its policy networks (introduced in Section 3.2.2 and Section 3.2.3). From lines 27 to 28, the attacker updates its policy (introduced in Section 3.2.4). The defender's average policy network is the output of the algorithm.

3.2.7 Discussion about Other Applicable Games

The proposed method is suitable for solving games whose legal actions vary significantly with states. Since the legal actions at different states can be very different, the overall action spaces of these games are usually enormous, leading to inefficiency of existing approaches. Our method is a mitigation for the problem. Apart from NSGs, our method can be applied to games that involve high-dimensional control. In such games, the overall legal action space is a combination of each dimension's legal action space, changes of legal actions in each dimension will accumulate and may lead to a significant change in the overall legal action space. Team-Goofspiel (Section 3.3.4) is an example for that. Another potential application scenario of our method is text-based games whose actions are defined based on natural languages, i.e., the action spaces are composed of sequences of words from a fixed size and large dictionary. There are constraints on actions at each state to generate a meaningful command (sequence of words), e.g., "open door", "turn left". For different states, e.g., "open" or "turn", the corresponding legal action spaces are very different ("door, box, book, ..." or "left, right, ..."). Our method can be applied to them by first learning states and legal action representations and then mapping state-action pair representations to values which represent Q-values or probabilities. High-level actions can be defined, for example, as phrases or sentences, and learning embeddings for each word is also reasonable.

3.3 Experimental Evaluation

We firstly evaluate our algorithm on large-scale NSGs. Then, we perform ablation studies to understand how each component of NSG-NFSP affects the results. Finally, we justify the adaptability of our method to games in other domains.

3.3.1 Experimental Settings

In our implementation, graph nodes are embedded into vectors of length 32. For the state representation network, we use a two-layer MLP with 64 hidden units and 64 outputs to extract features for resource locations. The GatedCNN module is composed of two identical 1D-convolutional layers with 64 kernels of size 3 and stride 1. One of the convolutional layers is activated by sigmoid function, and the result is point-wisely multiplied with the output of the other convolutional layer. The GatedCNN module is followed by a global max-pooling layer. The action representation network is a two-layer MLP with 64 hidden units and 64 outputs. State and action features are concatenated and sent to a two-layer MLP with 64 hidden units to predict Q-values or probabilities. Unless otherwise specified, all layers are separated with ReLU nonlinearities.

We use the Adam optimizer with a learning rate of 1×10^{-4} to optimize the average policy network. The RMSprop optimizer with the same learning rate is used for optimizing the BR policy network. Gradients are clipped according to the 2-norm to a threshold of 1. We update the BR policy network every 4 episodes with batch size at 128 and update the average policy network every 32 episodes with batch size at 256. The capacities of $\mathcal{M}_{RL}$ and $\mathcal{M}_{SL}$ are 5×10^5 and 1×10^7, respectively. The parameters of the target network are copied from the BR policy network every 1×10^3 episodes. The attacker updates the MAB at the end of each episode, and the MAB keeps a record of the latest 1×10^4 plays. The AVGer is updated every 1×10^3 episodes. We set the anticipatory parameter η as 0.1 and the attacker exploration

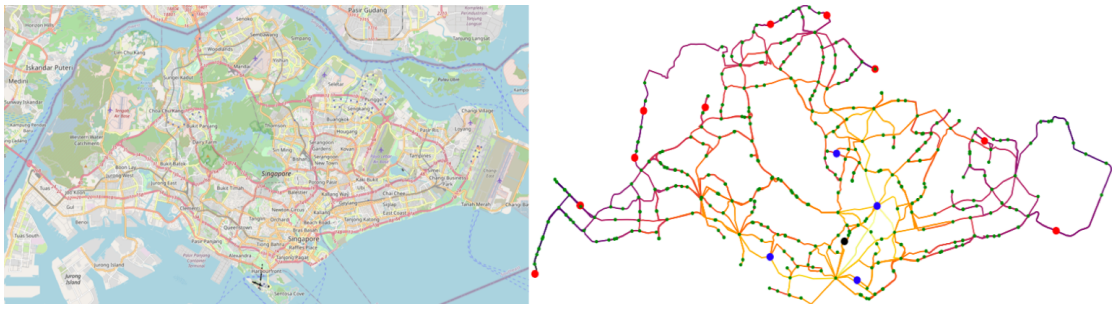


FIGURE 3.4: Singapore map and the extracted road network.

probability as 0.1. Experiments are performed on a server with a 10-core 3.3GHz Intel i9-9820X CPU and an NVIDIA RTX 2080 Ti GPU.

3.3.2 Large-Scale NSGs

We evaluate our algorithm in NSGs played on both artificially generated networks and real-world road networks.

Artificially Generated Networks We generate the evaluation network by the grid model with random edges [69]. Concretely, we sample a 15×15 grid whose horizontal/vertical edges appear with probability 0.4 and diagonal edges appear with probability 0.1. We set the initial location of the attacker at the center of the grid, and let the defender controls 4 security resources which are distributed uniformly on the network at the beginning[3]. There are 10 target nodes located randomly at the border. We set the time horizon T as 70, 90, and 300, to create NSGs with different scales.

We compare our algorithm with two heuristic defender policies, namely uniform policy and greedy policy, as well as a state-of-the-art algorithm, IGRS++ [51]. For the uniform policy, the defender assigns equal probability to each legal action at a state. For the greedy policy, all of the defender's security resources always move along the shortest path to the attacker's current location. When evaluating

[3]NSG-NFSP allows the players to do stochastic initialization, by taking the initialization as the first-step action.

	NSG-NFSP	Uniform policy	Greedy Policy	IGRS++
T=70	**0.1770 ± 0.0168**	0.0730 ± 0.0114	0 ± 0	OOM
T=90	**0.1205 ± 0.0143**	0.0715 ± 0.0111	0 ± 0	OOM
T=300	**0.0825 ± 0.0120**	0.0675 ± 0.0110	0 ± 0	OOM

(A) Synthetic network

	NSG-NFSP	Uniform Policy	Greedy Policy	IGRS++
T=30	**0.1225 ± 0.0140**	0.0230 ± 0.0066	0 ± 0	OOM
T=300	**0.0720 ± 0.0113**	0.0055 ± 0.0032	0 ± 0	OOM

(B) Real-world road network

TABLE 3.1: Approximate worst-case defender utilities in NSGs with different time horizons. The "±" indicates 95% confidence intervals over the 2000 testing episodes. OOM stands for Out of Memory.

the performance, because the game sizes are very large, it is intractable to calculate exact worst-case defender utility. We overcome this by using approximate worst-case defender utility. Specifically, we use a DQN attacker to best respond to the defender and calculate defender utility under this scenario to approximate the worst-case defender utility. We train the DQN attacker for 2×10^5 episodes, store the best model, and load the best model to play with the defender for another 2000 episodes to obtain the final results. We run the NSG-NFSP for 5×10^6 episodes, which takes around 3-4 days. Despite that the method consumes non-trivial resources when training, it can realize real-time inference, making its deployment possible. As in Table 3.1a, our method, NSG-NFSP, outperforms the baselines in all three large-scale NSGs. The state-of-the-art algorithm, IGRS++, cannot execute. The reason is that, as an incremental strategy generation algorithm, IGRS++ requires the attacking paths to be enumerable. However, for all of the three settings, it is impossible to enumerate attacking paths. We try to enumerate attacking paths for the case $T = 70$ on a machine with 32G RAM, but after all the memory is occupied, the enumeration does not end. Note that the approximate worst-case defender utilities for the greedy policy are always 0, which means that the DQN attacker can always find at least one path to a target node such that the defender with greedy policy cannot prevent him.

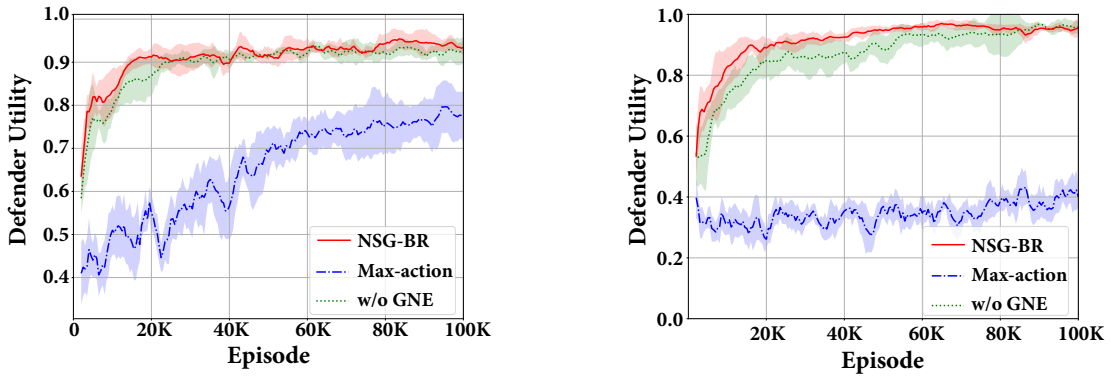
Real-World Road Networks As in Figure 3.4, we extract highways, primary roads and the corresponding intersections from Singapore map via OSMnx [70].

There are 372 nodes and 1470 edges. Edges are colored according to closeness centrality. The brighter the color, the closer the edges are to the center. The initial position of the attacker, marked in the dark point, locates near the center. There are 4 security resources, marked in blue points. Those target nodes, denoting exits of the map, are marked in red points. We test out that $T = 30$ will lead to attacking/escaping paths unenumerable, and we set the time horizon at 30 and 300. The NFSP defender is trained for 5×10^6 episodes which takes around a week to finish. We keep the evaluation settings the same as in synthetic networks for $T = 30$ because we find the settings work well. For $T = 300$, where training a DQN attacker is more difficult, we fine-tune hyperparameters to enhance the DQN attacker's performance. As presented in Table 3.1b, our method significantly outperforms the baselines in both settings.

3.3.3 Ablation Studies

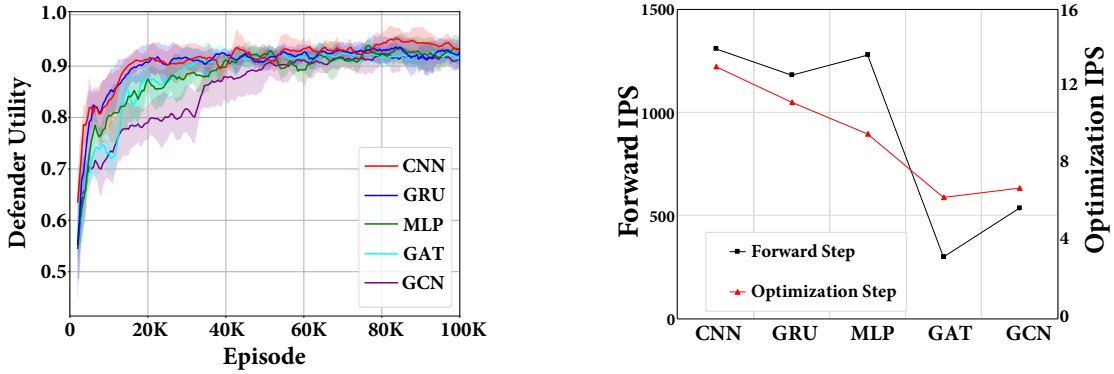
We perform ablation studies on 7×7 and 15×15 randomly generated grids, with T at 7 and 15, respectively.

Best Response Approximation We try to evaluate whether the proposed network architecture, NSG-BR, is able to enhance best response approximation. We set the policy of the attacker to be uniform, and let the defender best respond to him. We compare NSG-BR with a naive implementation, max-action DQN, which fixes the output dimension of BR policy network equal to the maximum number of legal actions. As in Figure 3.5, the performance of max-action DQN is significantly worse than NSG-BR in both settings. We further explore the effect of pre-defined graph node embedding (GNE). We replace GNE with a learnable embedding layer (w/o GNE). Results show that, in simple graph (the 7×7 grid), the learning curves have no obvious difference. However, in complex graph (the 15×15 grid), GNE does benefit the training. If created properly, GNE can not only speed up the training but also make the process more stable.



(A) 7×7 grid (T=7) (B) 15×15 grid (T=15)

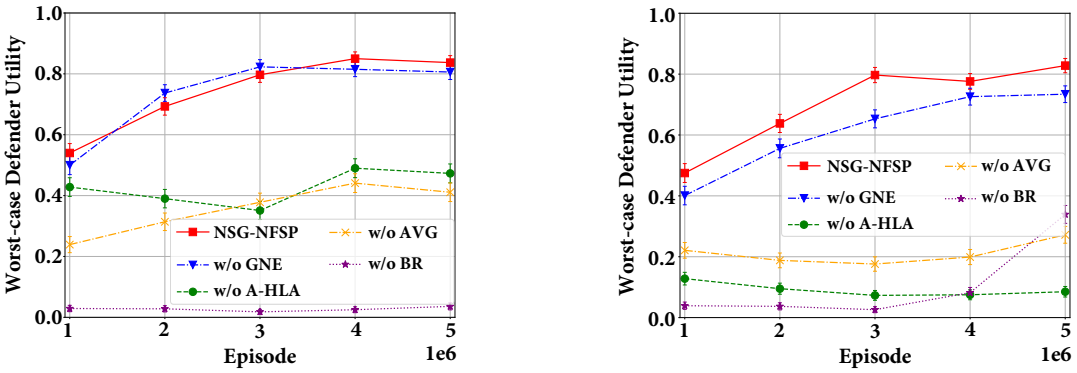
FIGURE 3.5: The learning curves of the defender against an attacker with uniform policy on synthetic networks, averaged across 5 runs.



(A) The learning curves (B) Iterations per second

FIGURE 3.6: Performance of sequence encoders on the 7×7 grid.

Comparisons of Different Sequence Encoders We compare the performance of different types of sequence encoders for extracting state features. The encoders include: i) GatedCNN (CNN), a CNN-based approach (what we use in NSG-NFSP); ii) Gated Recurrent Unit (GRU) [71]; iii) Graph Convolution Network (GCN) [72]; iv) Graph Attention Network (GAT) [73]; and v) Multi-Layer Perceptron (MLP). We evaluate the average defender utility on the synthetic 7×7 grid, setting the policy of the attacker to be uniform. The learning curves of the defender are presented in Figure 3.6a. We can find that GatedCNN obtains superior performance over other sequence encoders. We further test the running speed by measuring number of iterations per second (IPS) for both the forward process and the optimization process (batchsize=256). The forward process affects the speed to



(A) 7×7 grid (T=7) (B) 15×15 grid (T=15)

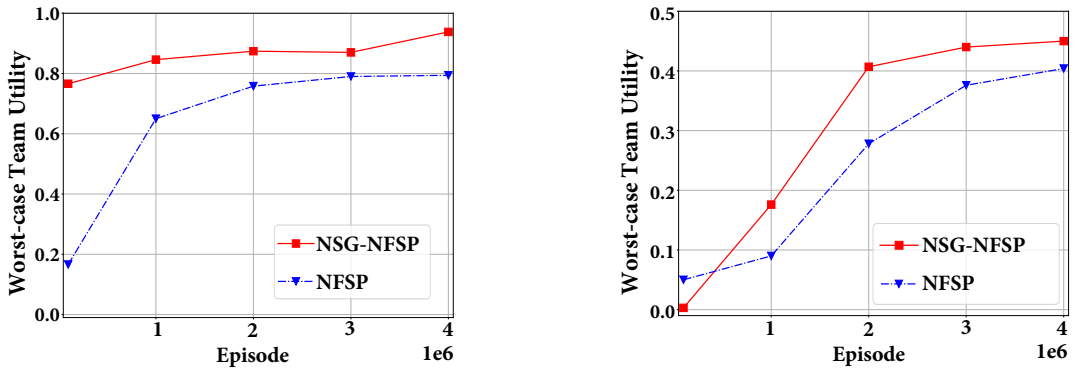
FIGURE 3.7: Ablation studies regarding each component of NSG-NFSP.

collect experiences, and the optimization process affects the speed to optimize parameters of DNNs. As in Figure 3.6b, GatedCNN is the fastest of all the sequence encoders.

Worst-Case Defender Utility To show how each component of NSG-NFSP affects the performance, we replace i) best response policy approximation module (Section 3.2.2) with max-action DQN (w/o BR); ii) average policy approximation module (Section 3.2.3) with max-action network (w/o AVG); iii) the attacker in high-level control (Section 3.2.4) with the original low-level implementation (w/o A-HLA); and iv) graph node embeddings (Section 3.2.5) with a learnable embedding layer (w/o GNE). As in Figure 3.7, replacing any of BR, AVG and A-HLA will cause a dramatic drop in performance. As for GNE, it shows a slight performance improvement in simple networks (the 7×7 grid), but in complex networks (the 15×15 grids), the enhancement is significant. NSG-NFSP achieves a performance of around 0.8 for the both games. Since the worst-case defender utility is upper-bounded by 1 (the defender wins the game definitely), the results demonstrate near-optimal solution quality of our method.

3.3.4 Adaptability

Our method is suitable for games whose legal action spaces or their sizes vary significantly with states, including but not limited to NSGs. To justify this, we



(A) MAX mode (B) AVERAGE mode

FIGURE 3.8: The worst-case team utility in 4-rank 2-member Team-Goofspiel under MAX and AVERAGE game modes.

conduct experiments on an extended version of Goofspiel [74], a popular poker game. Specifically, we modify Goofspiel to be played between a team with several members and a single player, as opposed to two single players. We name the modified game as Team-Goofspiel. In k-rank n-member Team-Goofspiel, there are k rounds and the team consists of n members. At each round t ($t = 1, \ldots, k$), n team members and the single player place bids for a prize of value t. The possible bids are $1, \ldots, k$ and each bid can be placed exactly once (the action space of the team player decays rapidly with respect to t, containing $(k + 1 - t)^n$ legal actions in total). The player with higher bid wins the prize of the current round; if the bids are equal, no player wins the prize. Both players can only observe the outcome of each round but not the bids. The single player will win the game if its total prize is greater than the team's prize at the end of the game, otherwise the team wins. The winner will obtain a utility of 1. We design two game modes for Team-Goofspiel, namely the MAX mode and the AVERAGE mode, where the bid of the team is determined by taking the maximum or the average of all team members' bids, respectively. We set $k = 4$ and $n = 2$ in the experiments. As in Figure 3.8, our method obtains superior performance compared to the vanilla NFSP algorithm in both the MAX mode and the AVERAGE mode, demonstrating

its good adaptability[4].

3.4 Chapter Summary

In this chapter, we propose a novel learning paradigm, NSG-NFSP, for finding NE in large-scale extensive-form NSGs. The algorithm trains DNNs to map state-action pairs to values, which may represent Q-values or probabilities. It enhances the performance by enabling the NFSP attacker with high-level actions and learning efficient graph node embeddings. Our method significantly outperforms state-of-the-art algorithms in both scalability and solution quality.

[4]Despite that it is easy to define high-level actions, e.g., bids sequences, and learning bids embeddings to reflect their numerical relationships is reasonable, we omit these in our implementation because they are domain-specific.

Chapter 4

Efficiently Learning Non-Exploitable Policy with Neural Monte Carlo Tree Search

In chapter 3, we propose NSG-NFSP, which is able to learn a robust policy when playing against an adaptive adversarial in large-scale NSG. However, NSG-NFSP suffers from the problem of data efficiency, which requires to sample a large number of experiences form the environment to learn a reasonable policy. NSG-NFSP has largely neglected intrinsic properties of the environment's dynamics. Specifically, when a player in NSGs selects an action (the node it will move to), the next state of the game can be partially determined, because the next location of the player can be inferred from the chosen action. However, in NSG-NFSP, it samples many rounds of the game and uses Monte Carlo method to estimate the distribution of the next state from scratch, leading to poor data efficiency. Another problem is that the centralized control of security resources in NSG-NFSP will inevitably result in combinatorial explosion in action space [75], making the algorithm unsuitable for handling scenarios where there are many resources.

In this chapter[1], we propose a DL-based method, NSGZero, for efficiently approaching a non-exploitable defender policy in NSGs. The motivation is that the state transition in NSGs is easy to model and we can leverage these information to perform effective planning before make the decisions. Specially, NSGZero improves data efficiency by modeling the dynamics of NSGs and performing planning with neural Monte Carlo Tree Search (MCTS) [3]. Our key contributions are in three aspects. First, we design three DNNs, namely the dynamics network, the value network and the prior network, to model environment dynamics, predict state values, and guide exploration, respectively, which unlock the use of efficient MCTS in NSGs. Second, we enable decentralized control in neural MCTS, to improve the scalability of NSGZero and make it applicable to NSGs with a large number of security resources. Third, we design an effective learning paradigm to jointly train the DNNs in NSGZero. Experimental results show that, compared to state-of-the-art algorithms, NSGZero achieves significantly better data efficiency and scalability.

4.1 Problem Formulation

A Network Security Game (NSG) can be formulated by a tuple $\langle G, V_s, V_t, V_m, T\rangle$, where $G = (V, E)$, consisting of a set of nodes V and a set of edges E, is the graph on which the NSG is played. $V_s \subset V$ is the set of possible starting nodes of the attacker. The target nodes, which represent destinations to be attacked or exits to escape, are denoted by $V_t \subset V$. The defender controls $m = |V_m|$ resources and the resources start from the nodes in $V_m \subset V$. T is the time horizon. The attacker and resources move on graph nodes, and a move is valid if and only if $(v_t, v_{t+1}) \in E$. Let v_t^{att} and $l_t^{def} = \langle v_t^0, \ldots, v_t^{m-1}\rangle$ denote the positions of the attacker and the m resources at step t respectively, following former works [48, 51]. The state of the attacker s_t^{att} is the sequence of the nodes he has visited, i.e.,

[1]The work in this chapter has been published as **Wanqi Xue**, Bo An, Chai Kiat Yeo. NSGZero: Efficiently learning non-exploitable policy in large-scale network security games with neural monte carlo tree search [76]. *Proceedings of the 36th AAAI Conference on Artificial Intelligence* (**AAAI**), pp.4646-4653, 2022.

$s_t^{att} = \langle v_0^{att}, v_1^{att}, \ldots, v_t^{att} \rangle$, and the state of the defender s_t^{def} consists of s_t^{att} and the resources' current locations, i.e., $s_t^{def} = \langle s_t^{att}, l_t^{def} \rangle$. The defender and the attacker interact sequentially, and their policies $\pi(s_t) = \Delta(\mathcal{A}(s_t))$ are mappings from state to a distribution Δ over valid moves (legal actions)[2]. Here, $\mathcal{A}(s_t)$ is a function which returns the set of legal actions at s_t, e.g., $\mathcal{A}(s_t^{att}) = \{v_{t+1}^{att} | (v_t^{att}, v_{t+1}^{att}) \in E\}$. For both players, they have free access to $\mathcal{A}(s_t)$. An NSG ends when the attacker reaches any of the target nodes within T or is caught[3]. If the attacker is caught, the defender will receive an end-game reward $r^{def} = 1$, otherwise, no award will be awarded to her. The game is zero-sum, so $r^{att} = -r^{def}$. A policy is said to be non-exploitable if it achieves the best performance in the worst-case scenario. Therefore, the optimization objective in NSGs is to maximize the worst-case defender reward $\max_{\pi^{def}} \min_{\pi^{att}} \mathbb{E}\left[r^{def} | \pi^{def}, \pi^{att}\right]$.

4.2 Efficiently Learning Non-Exploitable Policy

In this section, we introduce our approach, NSGZero, for efficiently learning a non-exploitable policy in NSGs. The algorithm improves data efficiency by performing planning with neural MCTS and improves scalability by enabling neural MCTS with decentralized execution. We begin by introducing the DNNs required to perform neural MCTS in NSGs. Next, we introduce how NSGZero enables neural MCTS with decentralized execution. Finally, we present how to effectively train the DNNs in NSGZero.

4.2.1 Designing the DNNs Required by Neural MCTS

To perform neural MCTS in NSGs, we should i) model the dynamics of the environment to perform state transition within the search tree; ii) predict the win rate

[2]We omit the superscripts for π and s_t because the formulate applies to both the defender and the attacker.

[3]The attacker is caught if he and at least one of the resources are in the same node at the same time or the time is up.

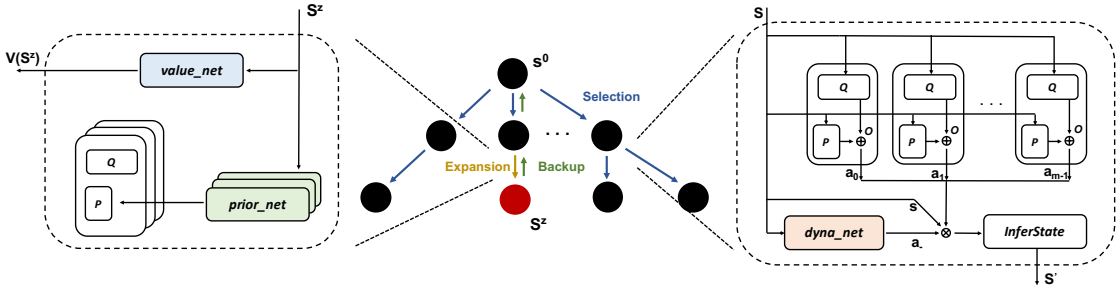


FIGURE 4.1: MCTS with the DNNs in NSGZero. Expansion: When the search tree reaches a new state s^z, the prior network are invoked to predict prior policies for the resources, and the predicted polices are stored in $P(i, s, a)$. Meanwhile, the value network is applied to predict the state value $V(s^z)$, which is used to update $Q(i, s, a)$ in the backup phase. Selection: For each resource i, a hypothetical action a_i is selected by comparing a score which is a weighted sum of $Q(i, s, a)$ and $P(i, s, a)$, and the weight is a function of $O(i, s, a)$. The dynamics network is used to predict the opponent's action a_-. With the hypothetical state s, the hypothetical actions for the m resources $\boldsymbol{a} = \langle a_0, \ldots, a_{m-1} \rangle$ and the predicted action for the opponent a_-, the next hypothetical state s' can be inferred because the environment of NSGs is deterministic.

(value) of a state to replace the expensive Monte Carlo rollout at a new state; and iii) leverage prior knowledge to guide exploration and narrow down the decision to high-probability actions. To achieve these purposes, we design three modules:

Dynamics network. In NSGs, the next state of a player depends heavily on what action is taken, because the action directly forms part of the state. Concretely, when the defender is in $s_t^{def} = \langle s_t^{att}, l_t^{def} \rangle$ and takes action $a_t^{def} = \langle v_{t+1}^0, \ldots, v_{t+1}^{m-1} \rangle$ (the movement of each resource), we can substitute l_t^{def} with a_t^{def} for constructing the next state $s_{t+1}^{def} = \langle s_{t+1}^{att}, a_t^{def} \rangle$. The only unknown part in s_{t+1}^{def} is $s_{t+1}^{att} = \langle s_t^{att}, v_{t+1}^{att} \rangle$, more specifically v_{t+1}^{att} (s_t^{att} is already known). Therefore, the dynamics network is designed to be a mapping from s_t^{def} to v_{t+1}^{att}, to model the behavior of the opponent. Considering that legal actions change with states, we adopt a structure similiar to DRRN [61] which learns representations for state and legal actions separately and outputs a likelihood by comparing the inner product of the representations. Formally, $dyna\_net(s, \mathcal{A}_-(s)) = SoftMax(f(s) \odot g(\mathcal{A}_-(s)))$, where $\mathcal{A}_-(s)$ are the legal actions of the opponent at state s, f and g are feature extractors for state and action respectively, $\odot$ denotes inner product.

Value network. This module is designed for substituting the inefficient Monte Carlo rollout when estimating the value of a new state. Given a state, $value\_net : \mathcal{S} \rightarrow \mathbb{R}$ is able to predict the expected future return, which is equivalent to the expected end-game reward in NSGs.

Prior network. This module is used to incorporate prior knowledge into the selection phase of MCTS. Specifically, the prior network takes state s and legal actions $\mathcal{A}_i(s)$ (for resource i) as input and outputs a distribution over the actions, indicating the preferences for selecting each available action. Unlike the dynamics network and the value network that are shared among resources, the prior network is held by each resource individually. For each resource, the structure of its prior network is the same as that of the dynamics network, because these two types of DNNs are both used to predict the behavior of an agent in graphs. In practice, we apply parameter sharing among the prior networks, to transfer knowledge between the resources and accelerate training [77].

Equipped with the three DNNs, NSGZero has the ability to simulate the future and plan based on the simulated state. At a decision point, NSGZero performs multiple simulations and generates policies based on the simulation results.

4.2.2 Neural MCTS with Decentralized Execution

Previous neural MCTS approaches, such as MuZero, focus on controlling a single agent, while in NSGs where the defender controls several resources, we need to enable neural MCTS with decentralized execution so that it can scale to the scenario with many resources. In this part, we introduce how to realize decentralized execution in NSGZero. At each decision point, neural MCTS iterates N simulations and outputs policies for resources based on the simulation results. We begin by discussing the simulation process. Then we introduce how to generate policies based on the simulation results.

Simulation Process. During the simulation process, for each resource i and for each edge (s, a) in the search tree Ψ, a set of statistics $\{Q(i, s, a), O(i, s, a), P(i, s, a)\}$ are stored, representing the state-action value, visit counts and prior policy respectively. Among the three types of statistics, $P(i, s, a)$ is calculated only once, at the step where the search tree reaches a new state. $Q(i, s, a)$ and $O(i, s, a)$ are continuously updated throughout the overall simulation process. As in Fig. 4.1, each simulation consists of three phases: selection, expansion and backup:

- **Selection:** In the selection phase, the search starts from the current hypothetical state s^0 and finishes when the simulation reaches a leaf node of the tree. To traverse within the search tree, NSGZero iteratively performs the selection operation. Assuming that it takes z steps for NSGZero to reach a leaf node (in a simulation), for each hypothetical step $k = 1, \ldots, z$, each resource i takes a hypothetical action a_i^k according to the search policy:

$$a_i^k = \arg\max_{a_i} \left[\mathrm{C_{PUCT}} \cdot \frac{\sqrt{\sum_{b_i} O(i, s^{k-1}, b_i)}}{1 + O(i, s^{k-1}, a_i)} \cdot P(i, s^{k-1}, a_i) + Q(i, s^{k-1}, a_i) \right] \tag{4.1}$$

  where $O(i, s^{k-1}, a_i)$ records the number of times resource i takes action a_i at hypothetical state s^{k-1}, $Q(i, s^{k-1}, a_i)$ is the estimated value for action a_i at hypothetical state s^{k-1}, $P(i, s^{k-1}, a_i)$ is the prior probability of taking a_i at hypothetical state s^{k-1} which is calculated when the simulation first encounters hypothetical state s^{k-1}. Here, we use polynomial upper confidence trees (PUCT) [6, 78], an adaptation of the standard MCTS, to combine value estimates with prior probabilities. A constant C_{PUCT} is used to control the trade-off between $Q(i, s^{k-1}, a_i)$ and $P(i, s^{k-1}, a_i)$, i.e., exploitation and exploration. After selecting hypothetical actions $\boldsymbol{a}^k = \langle a_0^k, \ldots, a_{m-1}^k \rangle$ for all resources, NSGZero predicts the behavior a_-^k of the opponent by invoking the dynamics network. With the hypothetical state s^{k-1}, the hypothetical actions for the m resources $\boldsymbol{a}^k$ and the predicted action for the opponent a_-^k, we can infer the next hypothetical state s^k and perform the next search process from s^k.

- **Expansion:** The expansion phase starts when the transition $(s^{k-1}, \boldsymbol{a}^k, a_-^k) \to s^k$ leads to a state s^z which is not in the search tree. In the expansion phase, NSGZero predicts the prior policy for each resource using the prior network and stores the probabilities in $P(i, s^k, a_i)$ correspondingly. In the meantime, since the search process has reached a node (state) which has never been visited before, NSGZero predicts the expected value $V(s^z)$ for the new state by calling the value network. The predicted state value is used to update $Q(i, s, a)$ along the trajectory from the root node to the parent of the leaf node.

- **Backup:** At the end of a simulation, the statistics, i.e., $Q(i, s^{k-1}, a_i^k)$ and $O(i, s^{k-1}, a_i^k)$, along the trajectory from the root node s^0 to the node s^{z-1} are updated. For $k = z, \ldots, 1$, we estimate the cumulative discounted return R^k by bootstrapping from $V(s^z)$. Formally, $R^k = \sum_{\tau=0}^{z-k-1} \gamma^\tau \cdot r(s^{k+\tau}, \boldsymbol{a}^{k+1+\tau}, a_-^{k+1+\tau}) + \gamma^{z-k} \cdot V(s^z)$, where $r(s^{k+\tau}, \boldsymbol{a}^{k+1+\tau}, a_-^{k+1+\tau})$ is the immediate reward for the transition $(s^{k-1}, \boldsymbol{a}^k, a_-^k) \to s^k$, $V(s^z)$ is the predicted state value for the hypothetical state s^z, and γ is the discount factor. The estimated value $Q(i, s^{k-1}, a_i^k)$ and the visit counts $O(i, s^{k-1}, a_i^k)$ are updated as follows,

$$\begin{aligned} Q\left(i, s^{k-1}, a_i^k\right) &\leftarrow \frac{O\left(i, s^{k-1}, a_i^k\right) \cdot Q\left(i, s^{k-1}, a_i^k\right) + R^k}{O\left(i, s^{k-1}, a_i^k\right) + 1} \\ O\left(i, s^{k-1}, a_i^k\right) &\leftarrow O\left(i, s^{k-1}, a_i^k\right) + 1 \end{aligned} \tag{4.2}$$

An overview about how to perform search in a simulation is presented in Algo. 2.

Generating the Policies. At each decision point s_t, NSGZero first initializes the search tree with s_t as the unique node and all former statistics being cleared. Then it repeatedly performs N simulations, gradually growing the search tree and tracking the statistics. Note that s_t denotes a real state, as contrast to s^k which is the k-th hypothetical state. A simulation starts by performing the selection operation from s_t and it takes s_t as the 0-th hypothetical state ($s^0 \leftarrow s_t$). After the N simulations, NSGZero generates a policy for each resource by querying the visit frequency of legal actions, with a temperature parameter $\mathcal{T}$ to control the randomness of the distribution. Formally, the policy for resource i is $\pi_i(s_t, a_i) \propto$

Algorithm 2: NGSZero-Ψ.SEARCH

Input: A hypothetical state s.

if *s is an ending state* **then**
  return the ending-game reward $r(s)$;
else if *s is not in the search tree* Ψ **then**
  Add s to the search tree Ψ;
  for *resources* $i = 0, \ldots, m-1$ **do**
    $\Psi.P(i, s) \leftarrow \Psi.prior\_net(s, \mathcal{A}_i(s))$;
  end
  return the predicted reward $\Psi.value\_net(s)$;
else
  for *resources* $i = 0, \ldots, m-1$ **do**
    Select hypothetical action a_i (Eq. 4.1);
  end
  $a_- \sim \Psi.dyna\_net(s, \mathcal{A}_-(s))$ \\ the opponent;
  $s' \leftarrow$ InferState(s, $\boldsymbol{a}$, a_-), $\boldsymbol{a} = \langle a_0, \ldots, a_{m-1} \rangle$;
  $r(s') \leftarrow \Psi.search(s')$;
  for *resources* $i = 0, \ldots, m-1$ **do**
    Update $\Psi.Q(i, s, a_i)$ and $\Psi.O(i, s, a_i)$;
  end
  return the searched reward $r(s')$.
end

Algorithm 3: NGSZero-EXECUTION

Input: The current state s_t, the search tree Ψ.

$\Psi.clear()$ \\ clear statistics stored in the search tree;
for *N simulations* **do**
  $\Psi.search(s_t)$ \\ perform lookahead search;
end
for *resources* $i = 0, \ldots, m-1$ **do**
  $\pi_i(s_t, a_i) \propto \frac{O(i, s_t, a_i)^{1/\mathcal{T}}}{\sum_{b_i} O(i, s_t, b_i)^{1/\mathcal{T}}}$; $\quad a_{i,t} \sim \pi_i(s_t)$;
end

Output: Joint action $\boldsymbol{a}_t = \langle a_{0,t}, \ldots, a_{m-1,t} \rangle$.

$\frac{O(i, s_t, a_i)^{1/\mathcal{T}}}{\sum_{b_i} O(i, s_t, b_i)^{1/\mathcal{T}}}$. Each resource i samples its action $a_{i,t}$ from $\pi_i(s_t)$ respectively. The pseudo-code of the overall execution process is in Algo. 3.

4.2.3 Training in NSGZero

All parameters of the DNNs in NSGZero are trained jointly to match corresponding targets collected by repeatedly playing NSGs. For brevity, we use Ψ_p, Ψ_v, Ψ_d to denote the prior network, the value network and the dynamics network respectively. Our first objective is to minimize the error between the prior policies predicted by Ψ_p and the improved policies $\boldsymbol{\pi}(s) = \langle \pi_0(s), \ldots, \pi_{m-1}(s) \rangle$ searched by NSGZero. To share experiences between resources and simplify the networks to optimize, we propose to let all resources share the parameters of their prior networks. As in Algo. 3 (lines 5-7), NSGZero generates a policy $\pi_i(s)$ and samples an action a_i for each resource i. Therefore, we update Ψ_p by minimizing $-\frac{1}{m}\sum_{i=0}^{m-1} a_i \cdot \log(\Psi_p(s, \mathcal{A}_i(s)))$, here a_i is in one-hot form. Next, the value network Ψ_v needs to be optimized to match discounted return. Let h be the length (total time steps) of an episode and r be the end-game reward, for $1 \leq t \leq h$, the discounted return is $\gamma^{h-t} \cdot r$. The conventional optimization method for value function is to minimize the mean squared error (MSE) [60], i.e., minimizing $(\Psi_v(s) - \gamma^{h-t} \cdot r)^2$. However, in NSGs, the target is bounded within the $[0, 1]$ interval, so we propose to convert the optimization of Ψ_v to a classification problem, i.e., optimize it by minimizing the binary cross entropy (CE) loss, with $\gamma^{h-t} \cdot r$ as soft label. Formally, $l_v = -(\gamma^{h-t} \cdot r) \cdot \log(\Psi_v(s)) - (1 - \gamma^{h-t} \cdot r) \cdot \log(1 - \Psi_v(s))$. The optimization of the dynamics network Ψ_d is similar to that of Ψ_p, with the objective as $-a_- \cdot \log(\Psi_d(s, \mathcal{A}_-(s)))$, and a_- is a one-hot label, indicating the opponent's action. The overall loss is

$$\mathcal{L}_{p,v,d} = -\frac{1}{B}\sum_{b=0}^{B-1}\frac{1}{h_b}\sum_{t=1}^{h_b}\Big[\frac{1}{m}\sum_{i=0}^{m-1} a_{i,t} \cdot \log(\Psi_p(s_t, \mathcal{A}_i(s_t)) + (1 - \gamma^{h_b - t} \cdot r_b) \cdot \log(1 - \Psi_v(s_t)) + (\gamma^{h_b - t} \cdot r_b) \cdot \log(\Psi_v(s_t)) + a_{-,t} \cdot \log(\Psi_d(s_t, \mathcal{A}_-(s_t)))\Big] \quad (4.3)$$

where B is the batch size, h_b and r_b denote the length and the end-game reward of an episode b.

Implementation. Optimizing $\mathcal{L}_{p,v,d}$ with batch training is non-trivial because

the length of each episode varies. In practice, for each episode b, we store the trajectories of the defender and the attacker, then we can infer all the variables required to calculate the loss for this episode, i.e., r_b, h_b, and s_t. For a batch of episodes B, we pad all the trajectories within the batch to the same length as the time horizon T (pad with 0). Then we calculate the loss in a batch for the B episodes by iterating over the time steps. For the padded entries, they do not contribute to the overall loss, so the calculated values for them should be filtered out. We generate a mask with h_b for each episode b, to indicate the padding items, then use the mask to filter out the values of padded items. Finally, the overall loss $\mathcal{L}_{p,v,d}$ is calculated by averaging all valid entries within the batch.

Modeling the Attacker. Despite self-play MCTS having been widely used in symmetric games [4], in NSGs which are asymmetric, we need to model the defender and the attacker differently. We apply high-level actions to the attacker to achieve efficient training [48]. Specifically, the attacker makes decisions on which target to be attacked, rather than deciding where to go in the next time step. At the beginning of each episode, the attacker selects a target and samples a path to the chosen target, then he moves along this path at each step. We use Multi-Armed Bandit (MAB), an algorithm to optimize the decision of multiple actions, to estimate the value of each target according to the latest J plays. Formally, the estimated value for a target is $Q(\zeta) = \frac{\sum_{j=1}^{J} r_j \cdot \mathbb{I}[\zeta_j=\zeta]}{\sum_{j=1}^{J} \mathbb{I}[\zeta_j=\zeta]}$, where ζ denotes the target, r_j is the player's reward for the j-th episode, and $\mathbb{I}$ is the binary indicator function. The MAB acts by selecting ζ with the largest estimated value. There is an Averager (AVGer) that tracks the historical behaviour of the MAB, by counting the number of times each target has been selected by the MAB. The AVGer generates a categorical distribution according to the counts and makes decision by sampling from the distribution. The attacker behaves as a mixture of the MAB and the AVGer, with an anticipate parameter η indicating the probability of him following the MAB.

4.3 Experimental Evaluation

We evaluated the performance of NSGZero on a variety of NSGs with different scales. We use these experiments to answer three questions. First, whether NSGZero is sufficiently data efficient compared to state-of-the-art algorithms, i.e., whether the algorithms can achieve comparable or even better performance when learning from fewer experiences. Second, whether the algorithm has better scalability and is applicable to large-scale NSGs. Third, how components of NSGZero affect the performance and whether the DNNs are trained as we expect. Experiments are conducted on a server with a 20-core 2.10GHz Intel Xeon Gold 5218R CPU and an NVIDIA RTX 3090 GPU.

4.3.1 Data Efficiency

To answer the first question, i.e., whether NSGZero has better data efficiency compared to state-of-the-art learning approaches, we evaluate the performance of NSGZero on two NSGs with synthetic graphs. For the first NSG, its graph is a 7×7 grid, with vertical/horizontal edges appearing with probability 0.5 and diagonal edges appearing with probability 0.1. We initialize the location of the attacker at the center of the grid and let resources of $m = 4$ be located uniformly around the attacker. There are 10 target nodes distributed randomly at the boundary of the grid. The time horizon is set to be equal to the length of the grid, i.e., 7. For the second NSG, it is generated similarly except that the graph is a randomly sampled 15×15 grid (vertical/horizontal edges appear with probability 0.4 and diagonal edges appear with probability 0.1) and the time horizon is 15. We find the best response attacker by enumerating all attack paths and choosing the path which leads to the lowest defender reward. The worst-case defender reward is the defender's payoff when she plays against the best response attacker.

We train NSGZero for 100,000 episodes and plot the worst-case defender reward. As in Fig. 4.2, the worst-case defender reward has seen a stable increase as training

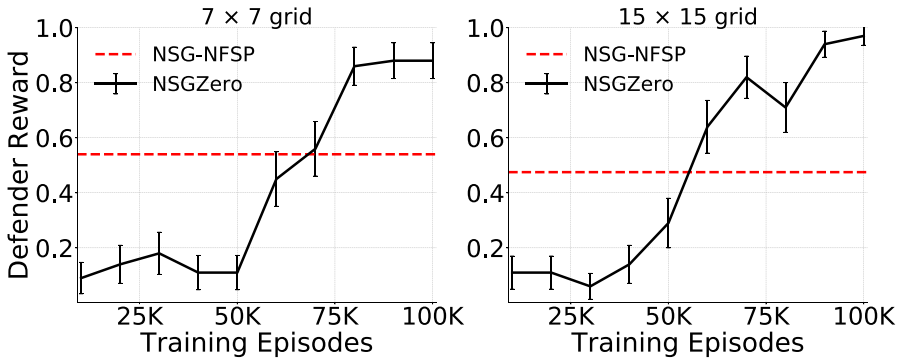
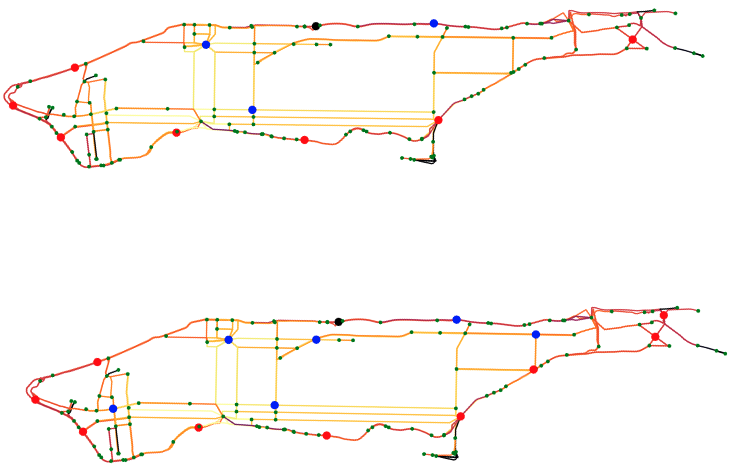


FIGURE 4.2: The worst-case defender reward. Error bars indicate 95% confidence intervals over the 100 testing episodes. The red dashed line indicates the performance of NSG-NFSP after training 1 million episodes.

proceeds and it can reach values at around 0.88 and 0.95 for the 7×7 grid and the 15×15 grid, respectively. Considering that the worst-case defender reward is upper-bounded by 1, NSGZero has found a near-optimal solution for both NSGs. Comparisons are made with a state-of-the-art learning method NSG-NFSP. We first train NSG-NFSP for 100,000 episodes, but find that there is no obvious learning effect and the worst-case defender reward remains close 0. Therefore, we increase its training episodes by 10 times to 1 million. Then NSG-NFSP reaches values of 0.54 and 0.48 for the two NSGs, still significantly lower than that of NSGZero. The experiments show that NSGZero is significantly more data efficient than NSG-NFSP, and it is able to achieve better performance even when learning from fewer experiences.

4.3.2 Scalability of NSGZero

To examine whether NSGZero is more scalable compared to existing methods, we extract two real-world maps from Manhattan and Singapore vis OSMnx [70] and create two large-scale NSGs for each map (As depicted in Fig. 4.3). For the Manhattan map, the first NSG has 3 resources and 7 targets. In the second NSG, we increase the scale to 6 resources and 9 targets to evaluate the performance of NSGZero in an NSG with many secure resources. For the Singapore map, the setup is similar, with the first NSG having 4 resources and 10 targets and the second



(A) Manhattan

(B) Singapore

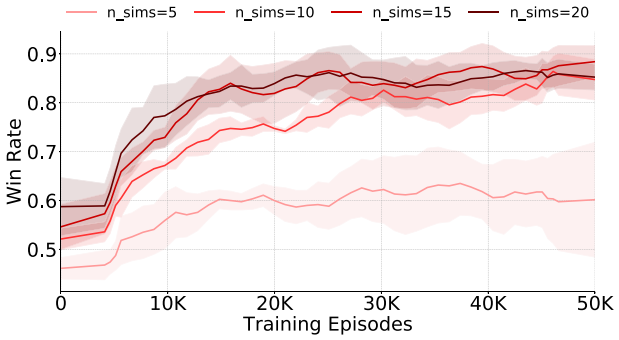
FIGURE 4.3: The extracted real-world maps. The attacker, the resources and the targets are marked by dark point, blue points and red points, respectively.

	Manhattan: $m = 3$	Manhattan: $m = 6$	Singapore: $m = 4$	Singapore: $m = 8$
NSGZero (Ours)	**0.1670 ± 0.0231**	**0.3660 ± 0.0299**	**0.1810 ± 0.0239**	**0.4140± 0.0306**
NSG-NFSP [48]	0.0390 ± 0.0120	–	0.0130 ± 0.0070	–
IGRS++ [51]	OOM	OOM	OOM	OOM
Uniform Policy	0.0120 ± 0.0096	0.1020 ± 0.0188	0.0240 ±0.0095	0.2590 ± 0.0272

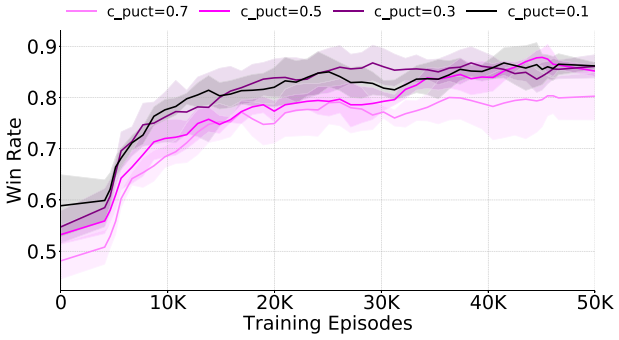
TABLE 4.1: Approximate worst-case defender rewards, averaged over 1000 test episodes. OOM stands for Out of Memory. The "±" indicates 95% confidence intervals over the 1000 plays.

NSG having 8 resources and 14 targets. Time horizon T is set as 30 for all NSGs to ensure that the policy space is sufficiently large. We manually set the initial location of the attacker, the resources and the targets to make the NSGs more realistic. When performing evaluation, considering that we cannot enumerate all attack paths as is done in small NSGs, as a mitigation, we use all the shortest paths and a best response DQN attacker to approximate the worst case. The smallest defender reward for playing against the worst-case approximators is reported.

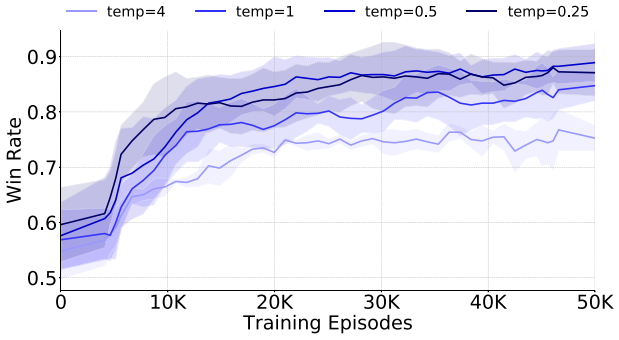
We make comparisons with two state-of-art-algorithms, i.e., NSG-NFSP [48] and IGRS++ [51], and a heuristic uniform policy. For the two learning-based approaches, i.e., NSGZero and NSG-NFSP, we train the model for 100,000 episodes. As shown in Table 4.1, our algorithm significantly outperforms the baselines. IGRS++, as an incremental strategy generation algorithm, requires that all attack paths to be enumerable, otherwise it runs out of memory due to lack of a terminal state. However, this requirement is not satisfied in the created games. IGRS++ fails to solve the games. For NSG-NFSP, when m is small, it can learn a policy, but the performance is not satisfactory, just comparable to uniform policy. We infer the reason is because of data inefficiency, i.e., NSG-NFSP is unable to learn a good policy from just 100,000 episodes. When m of the two maps is increased to 6 and 8, respectively, the training of NSG-NFSP becomes infeasible due to the huge action space. NSGZero performs well in all four NSGs, and it can scale to NSGs with many security resources, which are unsolvable by existing methods.



(A) Number of simulations



(B) Exploration constant



(C) Temperature parameter

FIGURE 4.4: Ablation studies for hyper-parameters in the execution process. The learning curves are for the NSGZero defender which plays against a heuristic uniform attacker, averaged over 5 runs.

4.3.3 Ablation Study and Analysis

To investigate how each component affects NSGZero, we evaluate the performance of NSGZero under different hyper-parameters, which leads to different structures of search trees or different policy generating strategies. We create an NSG by randomly sampling a 7×7 grid, and the setup is similar as before, i.e., vertical/horizontal edges appear with probability 0.5 and diagonal edges appear with

probability 0.1. There are $m = 4$ resources and 10 targets. The time horizon is 7. For the attacker, he uses a fixed strategy: at the beginning of each episode, he draws a target as a destination and randomly samples a path to the chosen target. At each step of the episode, the attacker moves according to the path. We use NSGZero to model the defender.

Number of Simulations. NSGZero generates policies according to the search tree, which starts from a single root node and gradually expands by performing N simulations. Therefore, we first examine how the number of simulations N affects the performance of NSGZero. We set N to 5, 10, 15, and 20, respectively, and test the average win rate of the defender throughout the training process. As in Fig. 4.4a, the performance of NSGZero becomes better in general as N increases. When $N = 5$, the performance is unsatisfactory and the win rate increases from 0.47 to around 0.6. As we increase the number of simulations, obvious learning effect can be observed, with the win rate reaching around 0.9. We also find that $N = 15$ is sufficient for good performance, after which increasing the number of simulations is no longer beneficial.

Exploration Constant. NSGZero uses an exploration constant C_{PUCT} to control the extent of exploration when performing searches. To study how this parameter affects the performance, we set $C_{PUCT} \in \{0.7, 0.5, 0.3, 0.1\}$. Fig. 4.4b shows that a large C_{PUCT} could hurt the performance since it weights more on the prior knowledge and the exploration term may overwhelm the value estimation. We can also find that a small C_{PUCT} is sufficient to narrow down the selection to high-probability actions.

Temperature Parameter. The number of simulations N and the exploration constant C_{PUCT} influence the simulation process, while after the simulations, NSGZero generates policies by counting the visit frequency and this procedure is controlled by the temperature parameter. To investigate the effect of the temperature parameter on NSGZero, we set the temperature to 4, 1, 0.5 and 0.25 to introduce different degrees of randomness when generating the policies. As in Fig. 4.4c, high

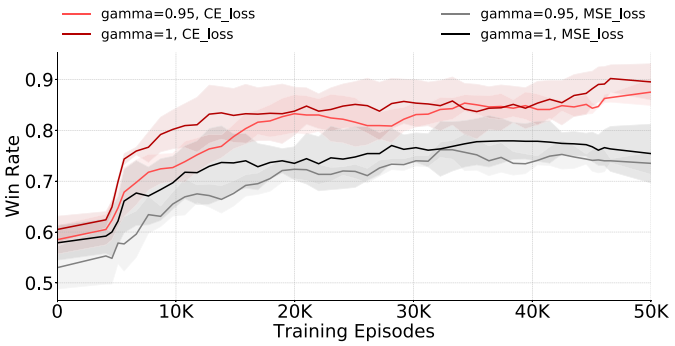


FIGURE 4.5: Ablation studies for the training process. The learning curves are for the NSGZero defender which plays against a heuristic uniform attacker, averaged over 5 runs.

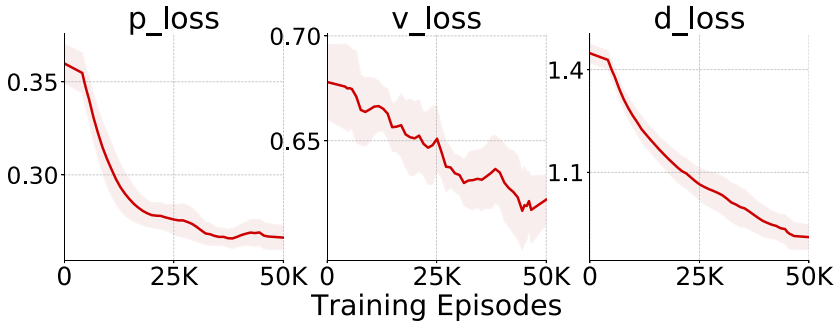


FIGURE 4.6: The loss curves of the prior network, the value network, and the dynamics network, averaged over 5 runs.

temperature causes a decrease in performance. When the temperature equals to 4, the final win rate is about 0.75, significantly lower than the win rate of 0.9 at lower temperatures. Meanwhile, low temperature can also lead to a slight decrease in win rate, in which case NSGZero is too confident of the generated policies and does not introduce enough randomness.

Previous experiments investigated three factors affecting the execution phase of NGSZero. Next, we examine those factors that influence the training process. In NGSZero, when training the value network, we convert its optimization objective from Mean Square Error (MSE) loss to binary cross entropy (CE) loss. To examine whether this change is beneficial, we plot the learning curves for these two optimization objectives respectively. As shown in Fig. 4.5, CE loss significantly outperforms MSE loss throughout the training, with a final win rate of 0.9 compared to a value of only about 0.75 for the MSE loss. We further investigate the effect of

the discount factor γ, which influences the optimization target of the value network ($\gamma^{h-t} \cdot r$). We find that, in NSGs where only end-game rewards are available, not applying a discount to rewards can result in consistently better performance. As in Fig. 4.5, $\gamma = 1$ outperforms $\gamma = 0.95$ for both MSE loss and CE loss. The reason may be that the distribution of the target is easier to learn when the rewards are not discounted. To examine whether the three DNNs of NSGZero are trained to match their corresponding targets, we plot the loss curves for them respectively. As shown in Fig. 4.6, significant reduction in loss can be found in all the DNNs, indicating that the parameters of the DNNs are optimized properly.

4.4 Chapter Summary

In this chapter, we introduce NSGZero as a learning method to efficiently approach a non-exploitable policy in NSGs. We design three DNNs, i.e., the dynamics network, the value network and the prior network, to unlock the use to efficient MCTS in NSGs. To improve scalability, we enable decentralized control in neural MCTS, which makes NSGZero applicable to NSGs with a large number of resources. To optimize the parameters of the DNNs, we provide a learning paradigm to achieve effective joint training in NSGZero. Experiments are conducted on a variety of NSGs with different graphs and scales. Empirical results show that, compared to state-of-the-art algorithms, NSGZero is significantly more data efficient and it is able to achieve much better performance even when learning from fewer experiences.

Chapter 5

Achieving Robust Communicative Policy in Multi-Agent Reinforcement Learning

Cooperative multi-agent reinforcement learning (MARL) has achieved remarkable success in a variety of challenging problems. For example, in NSGs, cooperative security resources are trained to perform a task with the same goal. To tackle the problems of scalability and non-stationarity in MARL, the framework of centralized training with decentralized execution (CTDE) is proposed [75, 79], where decentralized policies are learned in a centralized manner so that they can share experiences, parameters, etc., during training. Despite the advantages, CTDE-based methods still perform unsatisfactorily in scenarios where multi-agent coordination is necessary, mainly due to partial observability in decentralized execution. To mitigate partial observability, many multi-agent communicative reinforcement learning (MACRL) methods have been proposed, which allow agents to exchange information such as private observations and intentions during the execution phase. MACRL greatly improves multi-agent coordination in a wide range of tasks [11, 12, 14, 15, 17, 80, 81].

Meanwhile, adversarial machine learning has received extensive attention [82, 83]. Adversarial machine learning demonstrates that machine learning (ML) models are vulnerable to manipulation [20, 84]. As a result, ML models often suffer from performance degradation when under attack. For the same reason, many practical applications of ML models are at high risk. For instance, researchers have shown that, by placing a few small stickers on the ground at an intersection, self-driving cars can be tricked into making abnormal judgements and driving into the opposite lane [85]. Maliciously designed attacks on ML models can have serious consequences.

Unfortunately, despite its great importance, adversarial problems, especially adversarial inter-agent communication problems, remain largely uninvestigated in MACRL. Blumenkamp et al. show that, by introducing random noise in communication, agents are able to deceive their opponents in competitive games [23]. Moreover, the attacks are not artificially designed and therefore inefficient. Besides, cooperative cases, where communication is more crucial, are neglected. They also fail to propose an effective defence, but merely retrain the models to adapt to the attacks. Mitchell et al. propose to generate weights of Attention models [86] through Gaussian process for defending against random attacks in attention-based MACRL [24]. However, the applicability of this approach is unsatisfactory, being limited to attention-based MACRL, and its performance on maliciously designed attacks is unclear.

In this chapter[1], we systematically explore the problem of adversarial communication in MACRL, where there are malicious agents that attempt to disrupt multi-agent cooperation by manipulating messages. We aim to learn a robust communication policy for cooperative agents. Our contributions are in three aspects. First, we propose an effective learning approach to model the optimal attacking scheme in MACRL. Second, to defend against the adversary, we propose a two-stage message

[1]The work in this chapter has been published as **Wanqi Xue**, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, Chai Kiat Yeo. Mis-spoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning [87]. *Proceedings of the 21th International Joint Conference on Autonomous Agents and Multi-Agent Systems* (**AAMAS**), pp.1418-1426, 2022.

filter which works by first detecting the malicious message and then recovering the message. The defending scheme can greatly maintain the coordination of agents under message attacks. Third, to address the problem that the message filter can be exploited by learnable attackers, we formulate the attack-defense problem as a two-player zero-sum game and propose $\mathfrak{R}$-MACRL, based on a game-theoretical framework Policy-Space Response Oracle (PSRO) [88, 89], to approximate a Nash equilibrium policy in the adversarial communication game. $\mathfrak{R}$-MACRL improves the defensive performance under the worst case and thus improves the robustness. Empirical experiments demonstrate that many state-of-the-art MACRL algorithms are vulnerable to attacks and our method can significantly improve their robustness.

5.1 Achieving Robustness in MACRL

In this section, we propose our method for achieving robustness in MACRL. We begin by proposing the problem formulation for adversarial communication in MACRL. Then, we introduce the method to build the optimal attacking scheme in MACRL. Next, we propose a learnable two-stage message filter to defend against possible attacks. Finally, we propose to formulate the attack-defense problem as a two-player zero-sum game, and design a game-theoretic method $\mathfrak{R}$-MACRL to approximate a Nash equilibrium policy for the defender. In this way, we can improve the worst-case performance of the defender and thus enhance robustness.

5.1.1 Problem Formulation: Adversarial Communication in MACRL

An MACRL problem can be modeled by *Decentralised Partially Observable Markov Decision Process with Communication* (*Dec-POMDP-Com*) [90], which is defined by a tuple $\langle \mathcal{S}, \mathcal{M}, \mathcal{A}, \mathcal{P}, R, \Upsilon, O, C, \mathcal{N}, \gamma \rangle$. $\mathcal{S}$ denotes the state of the environment and $\mathcal{M}$ is the message space. Each agent $i \in \mathcal{N} := \{1, ..., N\}$ chooses an action

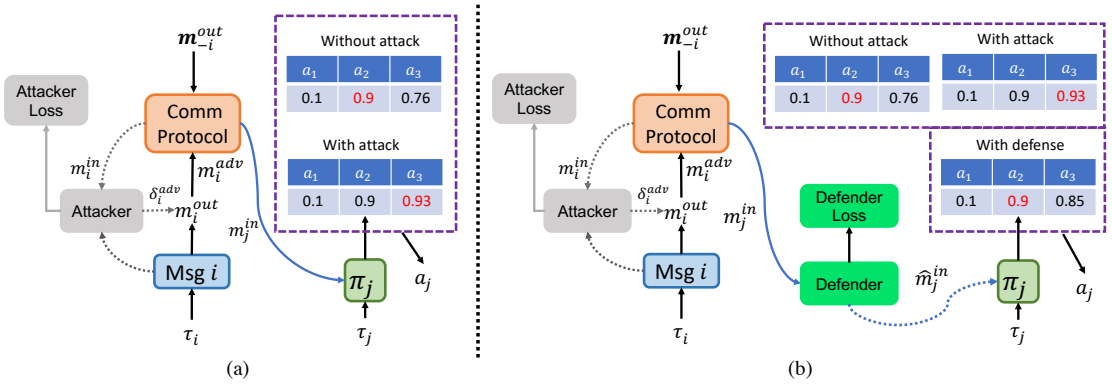


FIGURE 5.1: General framework of attack (left) and defence (right) in MACRL. *Attack:* The agent i, taken over by the attacker, generates malicious message m_i^{adv} to disrupt multi-agent cooperation. The incoming message m_j^{in} and estimated Q-values of the benign agent j will be affected due to m_i^{adv}, which may lead to incorrect decisions. *Defence:* A learnable defender is cascaded to the communication protocol module, which is used to reconstruct the contaminated message (m_j^{in} to $\hat{m}_j^{in}$). The benign agent j estimates Q-values based on $\hat{m}_j^{in}$, which can reduce the probability of selecting sub-optimal actions.

$a_i \in \mathcal{A}$ at a state $\boldsymbol{s} \in \mathcal{S}$, giving rise to a joint action vector, $\boldsymbol{a} := [a_i]_{i=1}^N \in \mathcal{A}^N$. $\mathcal{P}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A}^N \times \mathcal{S} \mapsto \mathcal{P}(\mathcal{S})$ is a Markovian transition function. Every agent shares the same joint reward function $R(\boldsymbol{s}, \boldsymbol{a}) : \mathcal{S} \times \mathcal{A}^N \mapsto \mathbb{R}$, and $\gamma \in [0, 1)$ is the discount factor. Due to partial observability, each agent has individual partial observation $v \in \Upsilon$, according to the observation function $O(\boldsymbol{s}, i) : \mathcal{S} \times \mathcal{N} \mapsto \Upsilon$. Each agent holds two policies, i.e., action policy $p_i(a_i|\tau_i, m_i^{in}) : \mathcal{T} \times \mathcal{M} \times \mathcal{A} \mapsto [0, 1]$ and message policy $v_i(m_i^{out}|\tau_i, m_i^{in}) : \mathcal{T} \times \mathcal{M} \times \mathcal{M} \mapsto [0, 1]$, both of which are conditioned on the action-observation history $\tau_i \in \mathcal{T} := (\Upsilon \times \mathcal{A})$ and incoming messages m_i^{in} aggregated by the communication protocol $C(m_i^{in}|\boldsymbol{m}^{out}, i) : \mathcal{M}^{|\mathcal{N}|} \times \mathcal{N} \times \mathcal{M} \mapsto [0, 1]$.

In adversarial communication where there are N_{adv} malicious agents, we assume that each malicious agent holds the third private adversarial policy, $\xi(\delta_i^{adv}|\tau_i, m_i^{in}, m_i^{out}) : \mathcal{T} \times \mathcal{M} \times \mathcal{M} \times \mathcal{M} \mapsto [0, 1]$, which generates adversarial message δ_i^{adv} based on its action-observation history τ_i, received messages m_i^{in} and the message m_i^{out} intended to be sent. Malicious agents could send messages by convexly combining their original messages with adversarial messages, i.e., $m_i^{adv} = (1 - \omega) \times m_i^{out} + \omega \times \delta_i^{adv}$, or simply summing up the messages, i.e., $m_i^{adv} = m_i^{out} + \delta_i^{adv}$. To reduce the likelihood of being detected, apart from the adversarial policy ξ, malicious agents

strictly follow their former action policy and message policy, trying to behave like benign agents. Fig. 5.1(a) presents the overall attacking procedure. The agent i (attacker) is malicious and tries to generate adversarial message m_i^{adv} to disrupt cooperation. The adversarial message sent by agent i together with normal messages sent by other agents (denoted by $\boldsymbol{m}_{-i}^{out}$) are processed by the communication protocol (algorithm-related), generating a contaminated incoming message m_j^{in} for a benign agent j. From agent j's perspective, under such attack, the estimated Q-values will change. If the action with the highest Q-value shifts, agent j will make incorrect decisions, leading to suboptimality. To perform an effective attack in MACRL, we propose to optimize the adversarial policy ξ by minimizing the joint accumulated rewards, i.e., $\min_\xi \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$. We make two assumptions to make the adversarial communication problem both practical and tractable.

Assumption 5.1. (Byzantine Failure [91]) Agents have imperfect information on who are malicious.

Assumption 5.2. (Concealment)Malicious agents do not communicate or cooperate with each other when performing attacks in order to be covert.

To defend against the attacker, as in Fig. 5.1(b), we propose to cascade a learnable defender to the communication protocol module. The defender performs a transformation from m_j^{in} to $\hat{m}_j^{in}$ to reconstruct the contaminated messages, to avoid distributing the contaminated message m_j^{in} directly to agent j. With such transformation, the benign agent j can estimate the Q-value for each action more properly and reduce the probability of selecting sub-optimal actions.

5.1.2 Learning the Attacking Scheme

To model the attack scheme in adversarial communication, we use a deep neural network (DNN) f_μ, parameterized by θ_μ, to generate adversarial messages for a malicious agent. The adversarial policy ξ is a multivariate Gaussian distribution whose parameters are determined by the DNN f_μ, i.e., $\xi = \mathcal{N}(f_\mu(\cdot|\tau, m^{in}, m^{out}; \theta_d), \Lambda)$

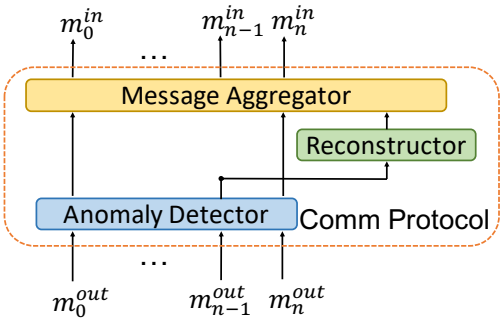


FIGURE 5.2: Structure of the communication protocol with the message filter (defender). The anomaly detector and the reconstructor are designed to determine and recover the potential malicious messages, respectively.

where Λ is a fixed covariance matrix. The reason of using Gaussian distribution as the prior is that it is the maximum entropy distribution under constraints of mean and variance. Each malicious agent generates adversarial messages by sampling from its adversarial policy. The optimization objective of the adversarial policy is to minimize the accumulated team rewards subject to a constraint on the distance between m^{out} and m^{adv}. We utilize Proximal Policy Optimization (PPO) [92] to optimize the adversarial policy by maximizing the following objective:

$$\mathcal{J}_\xi(\theta_\mu) = \mathbb{E}_{(\tau, m^{in}, m^{out}, \delta^{adv})}[\min(\rho \cdot A^\xi, \text{clip}\,(\rho, 1 \pm \epsilon) \cdot A^\xi)] \\ - \alpha \cdot \mathbb{E}_{(m^{out}, \delta^{adv})}[(m^{out} - m^{adv})^2] \quad (5.1)$$

where $\rho = \xi(\delta^{adv}|\tau, m^{in}, m^{out})/\xi^{old}(\delta^{adv}|\tau, m^{in}, m^{out})$, ξ^{old} is the policy in the previous learning step, ϵ is the clipping ratio, and $A^\xi(\delta^{adv}, \tau, m^{in}, m^{out})$ is the advantage function. Let $e = \langle \tau, m^{in}, m^{out} \rangle$ denotes the input of the value function and we define the reward of the attacker to be negative team reward, then $A^\xi(\delta^{adv}, e) = -r + V(e') - V(e)$ where V is the value function, e' denotes the next step state, and r is the immediate reward. We learn the value function $V(e)$ by minimizing $\mathbb{E}[(V(e) + \sum_t \gamma^t r_t)^2]$.

5.1.3 Defending against Adversarial Communication

We can train a DNN to model the attack scheme in adversarial communication. However, the defence of that is non-trivial because i) benign agents have no prior knowledge on which agents are malicious; ii) the malicious agents can inconsistently pretend to be cooperative or non-cooperative to avoid being detected; and iii) recovering useful information from the contaminated messages is difficult. To address these challenges, we design a two-stage message filter for the communication protocol to perform defence. The message filter works by first determining the messages that are likely to be malicious and then recovering the potential malicious messages before distributing them to the corresponding agents. As in Fig. 5.2, the message filter $\zeta(h_d, g_r)$ consists of an anomaly detector h_d and a message reconstructor g_r. The anomaly detector, parameterized by θ_d, outputs the probability that each incoming message needs to be recovered, i.e., $h_d(\cdot|m_i^{out};\theta_d) : \mathcal{M} \mapsto \Psi$, where Ψ denotes a binomial distribution, m_i^{out} denotes the message sent by agent i. We perform sampling from the generated distributions to determine the anomaly messages $\boldsymbol{x} \sim h_d(\cdot|\boldsymbol{m}^{out};\theta_d)$ [2]. Here, $\boldsymbol{x}$ is an indicator that records whether a message is predicted to be malicious or not. The predicted malicious messages $\dot{\boldsymbol{m}}$ are recovered by the reconstructor $g_r(\cdot|\dot{m}_i;\theta_r) : \mathcal{M} \mapsto \mathcal{M}$ separately. The recovered message and other normal messages are aggregated to determine the messages that each agent will receive.

The optimization objective of the message filter is to maximize the joint accumulated rewards under attack, i.e., $\max_\zeta \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \tilde{r}_t|\xi\right]$, where $\tilde{r}_t$ is the team reward after performing the defending strategy. We utilize PPO to optimize the strategy. To mitigate the local optima induced by unguided exploration in large message space, we introduce a regularizer which is optimized under the guidance of the ground-truth labels of messages (whether they are malicious). For the reconstructor, we train it by minimizing the distance between the messages before and after being attacked. To improve the tolerance of the reconstructor to errors made by

[2] We abuse $h_d(\cdot|\boldsymbol{m}^{out};\theta_d)$ to represent that messages $m_i^{out} \in \boldsymbol{m}^{out}$ are fed into h_d in batch.

the detector, apart from malicious messages, we also use benign messages as training data, in which case the reconstructor is an identical mapping. Formally, the two-stage message filter is optimized by maximizing the following function:

$$\begin{aligned}\mathcal{J}_\zeta\left(\theta_d, \theta_r\right) = \mathbb{E}_{(\boldsymbol{m},\boldsymbol{x})}\left[\sum_i^{|\boldsymbol{m}|} \min\left(\rho_i \cdot A_i^{h_d}, \operatorname{clip}\left(\rho_i, 1 \pm \epsilon\right) \cdot A_i^{h_d}\right)\right] \\ + \beta_1 \cdot \mathbb{E}_{\boldsymbol{m}}\left[\hat{\boldsymbol{y}} \cdot \log(h_d(\cdot\,|\boldsymbol{m}; \theta_d))\right] \\ - \beta_2 \cdot \mathbb{E}_{\boldsymbol{m}}\left[(\hat{\boldsymbol{m}} - g_r(\cdot\,|\boldsymbol{m}; \theta_r))^2\right] \quad (5.2)\end{aligned}$$

where $\rho_i = h_d(x_i|m_i;\theta_d)/h_d^{old}(x_i|m_i;\theta_d)$, $A_i^{h_d}(x_i, m_i)$ is the advantages which are estimated similarly as in the attack, $\hat{\boldsymbol{y}}$ is the one-hot ground-truth labels of messages, $\hat{\boldsymbol{m}}$ is the messages that have not been attacked. β_1, β_2 and ϵ are hyperparameters.

5.1.4 Achieving Robust MACRL

Despite the defensive message filter, the effectiveness of the defence system can rapidly decrease if malicious agents are aware of the defender and adjust their attack schemes. To mitigate this, we formulate the attack-defence problem as a two-player zero-sum game (malicious agents are controlled by the attacker) and improve the performance of the defender in the worst case. We define the adversarial communication game by a tuple $\langle \Pi,\, U\rangle$, where $\Pi = \langle \Pi_\xi, \Pi_\zeta\rangle$ is the joint policy space of the players (Π_ξ and Π_ζ denote the policy space of the defender and the attacker respectively), $U : \Pi \mapsto \mathbb{R}^2$ is the utility function which is used to calculate utilities for the attacker and the defender given their joint policy $\pi = \langle \pi_\xi, \pi_\zeta\rangle \in \Pi$. The utility of the defender is defined as the expected team return. The adversarial communication game is zero-sum, i.e., the utility of the defender $U_\zeta(\pi)$ must be the negative of the utility of the attacker $U_\xi(\pi)$ for $\forall \pi \in \Pi$. The solution concept of the adversarial communication game is Nash equilibrium (NE) and we can approach an NE by optimizing the following MaxMin objective:

$$\mathcal{J}_{\xi,\zeta} = \max_{\pi_\zeta \in \Pi_\zeta} U_\zeta(\mathrm{Br}(\pi_\zeta), \pi_\zeta) = \max_{\pi_\zeta \in \Pi_\zeta} \min_{\pi_\xi \in \Pi_\xi} \mathbb{E}\left[\sum\nolimits_{t=0}^{\infty} \gamma^t \tilde{r}_t | \pi_\xi, \pi_\zeta\right] \quad (5.3)$$

where $\mathrm{Br}(\pi_\zeta)$ is the best response policy of the defender, i.e., $\mathrm{Br}(\pi_\zeta) = \arg\min_{\pi_\xi} U_\zeta(\pi_\xi, \pi_\zeta)$.

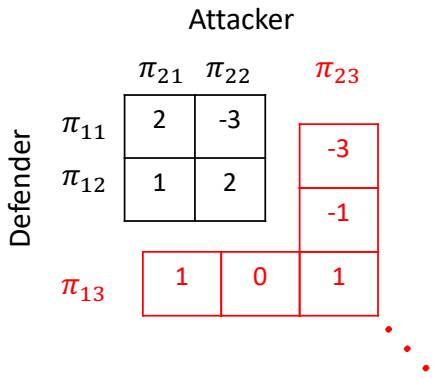


FIGURE 5.3: Workflow of $\mathfrak{R}$-MACRL.

We propose $\mathfrak{R}$-MACRL to optimize the objective in the adversarial communication game. $\mathfrak{R}$-MACRL is developed based on Policy-Space Response Oracle (PSRO) [88], a deep learning-based extension of the classic Double Oracle algorithm [93]. The workflow of $\mathfrak{R}$-MACRL is depicted in Fig. 5.3. At each iteration, $\mathfrak{R}$-MACRL keeps a population (set) of policies $\Pi^t \subset \Pi$, e.g., $\Pi^t_\zeta = (\pi_{11}, \pi_{12})$ and $\Pi^t_\xi = (\pi_{21}, \pi_{22})$. We can evaluate the utility table $U(\Pi^t)$ for the current iteration and calculate a Nash equilibrium $\pi^t_* = \langle \Delta(\Pi^t_\xi), \Delta(\Pi^t_\zeta) \rangle$ for the game restricted to policies in Π^t by, for example, linear programming, replicator dynamics, etc. Δ denotes the meta-strategy which is an arbitrary categorical distribution. Next, we calculate the best response policy $\mathrm{Br}(\pi^t_{*,-i})$ to the NE in this restricted game for each player i (player $-i$ denotes the opponent of player i, $i \in \{\xi, \zeta\}$), e.g., $\pi_{13} = \mathrm{Br}(\pi^t_{*,\xi})$ and $\pi_{23} = \mathrm{Br}(\pi^t_{*,\zeta})$. We extend the population by adding the best response policies to the policy set: $\Pi^{t+1}_i = \Pi^t_i \cup \mathrm{Br}(\pi^t_{*,-i})$ for $i \in \{\xi, \zeta\}$. After extending the population, we complete the utility table $U(\Pi^{t+1})$ and perform the next iteration. The algorithm converges if the best response policies are already in the population. Practically, $\mathfrak{R}$-MACRL approximates the utility function $U(\,\cdot\,)$ by having each policy in the population Π^t_i playing with each other policy in Π^t_{-i} and tracking the average utilities. The approximation of the best response policies is performed by maximizing $\mathcal{J}_\xi(\theta_\mu | \pi^t_{*,\zeta})$ and $\mathcal{J}_\zeta(\theta_d, \theta_r | \pi^t_{*,\xi})$ for the attacker and the defender, where the full expressions of the two objectives are described in Eq. 5.1 and Eq. 5.2, respectively.

Algorithm 4: $\mathfrak{R}$-MACRL

1 **Input:** initial policy sets for both players Π^0, maximum number of iterations T;
2 Empirically estimate utilities $U(\Pi^0)$ for each joint policy $\pi \in \Pi^0$;
3 Initialize mixed strategies for both players $\pi^0_{*,i} = \textsc{Uniform}(\Pi^0_i)$ for $i \in \{\xi, \zeta\}$;
4 Initialize number of iterations $t = 0$;
5 **while** *not converge and* $t \leq T$ **do**
6 **for** *player* $i \in \{\xi, \zeta\}$ **do**
7 Train π_i^{t+1} by letting it exploit $\pi^t_{*,-i}$;
8 Extend policy set $\Pi_i^{t+1} = \Pi_i^t \cup \{\pi_i^{t+1}\}$;
9 **end**
10 Check convergence and $t = t + 1$;
11 Estimate missing entries in $U(\Pi^{t+1})$;
12 Compute mixed strategies π_*^{t+1} by solving the matrix game defined by $U(\Pi^{t+1})$;
13 **end**
14 **Output:** mixed strategies π_* for both players ;

The overall algorithm of $\mathfrak{R}$-MACRL is presented in Algorithm 4. We first initialize the meta-game and the mixed strategies (line 2 and line 3). Then, at each step, we train the attacker and the defender alternately by letting them best respond to their corresponding opponent (line 7). Next, the learned new policy π_i^{t+1} is added to the policy set Π_i^{t+1} for the two players respectively (line 8). After extending the policy set, we can check the convergence (line 10) and calculate the missing entries in $U(\Pi^{t+1})$ (line 11). Finally, we solve the new meta-game (line 12) and repeat the iteration.

5.2 Experimental Evaluation

We conduct extensive experiments on a variety of state-of-the-art MACRL algorithms to answer: **Q1:** *Are MACRL methods vulnerable to message attacks and whether the two-stage message filter is able to consistently recover multi-agent coordination?* **Q2:** *Whether* $\mathfrak{R}$*-MACRL is able to improve the robustness of MACRL algorithms under message attacks?* **Q3:** *Whether our method is able to scale to scenarios where there are multiple attackers?* **Q4:** *Which components contribute*

to the performance of the method and how does the proposed method work? We first categorize existing MACRL algorithms and then select representative algorithms to perform the evaluation. All experiments are conducted on a server with 8 NVIDIA Tesla V100 GPUs and a 44-core 2.20GHz Intel(R) Xeon(R) E5-2699 v4 CPU.

5.2.1 Experimental Setting

MACRL methods are commonly categorized by whether they are implicit or explicit [90, 94]. In implicit communication, the actions of agents influence the observations of the other agents. Whereas in explicit communication, agents have a separate set of communication actions and exchange information via communication actions. In this chapter, we focus on explicit communication because in implicit communication, to carry out an attack, the attacker's behaviour is often bizarre, making the attack trivial and easily detectable. We consider the following three realistic types of explicit communication:

- **Communication with delay (CD):** Communication in real world is usually not real-time. We can model this by assuming that it takes one or a few time steps for the messages to be received by the targeted agents [81].
- **Local communication (LC):** Messages sometimes cannot be broadcast to all agents due to communication distance limitations or privacy concerns. Therefore, agents need to learn to communicate locally, affecting only those agents within the same communication group [12, 95].
- **Communication with cost or limited bandwidth (CC):** Agents should avoid sending redundant messages because communication in real world is costly and communication channels often have a limited bandwidth [15, 16].

Categories	Methods	Environments
CD	CommNet [81]	Predator Prey (PP) [95]
LC	TarMAC [12]	Traffic Junction (TJ) [81]
CC	NDQ [15]	StarCraft II (SCII) [96]

TABLE 5.1: The chosen algorithms and environments.

Following the above taxonomy, we select some representative state-of-the-art algorithms to perform evaluation[3]. As in Table 5.1, we select CommNet [81], TarMAC [12] and NDQ [15] for CD, LC and CC respectively. CommNet [81] is a simple yet effective multi-agent communicative algorithm where incoming messages of each agent are generated by averaging the messages sent by all agents in the last time step. TarMAC [12] extends CommNet by allowing agents to pay attention to important parts of the incoming messages via Attention [86] network. Concretely, each agent generates signature and query vectors when sending message. In the message receiving phase, attention weights of incoming messages are calculated by comparing the similarity between the query vector and the signature vector of each incoming message. Then a weighted sum over all incoming messages is performed to determine the message an agent will receive for decentralized execution. NDQ [15] achieves nearly decomposable Q-functions via communication minimization. Specifically, it trains the communication model by maximizing mutual information between agents' action selection and communication messages while minimizing the entropy of messages between agents. Each agent broadcasts messages to all other agents. The sent messages are sampled from learned distributions which are optimized by mutual information. We evaluate in the following environments, which are similar to those in the works that first introduced the algorithms:

Predator Prey (PP). There are 3 predators, trying to catch 6 preys in a 7×7 grid. Each predator has local observation of a 3×3 sub-grid centered around it and can move in one of the 4 compass directions, remain still, or perform catch

[3]Sweeping the whole list of algorithms for each category is impossible and unnecessary since there have been a lot of algorithms proposed and algorithms in each category usually share common characteristics.

action. The prey moves randomly and is caught if at least two nearby predators try to catch it simultaneously. The predator will obtain a team reward at 10 for each successful catch and will be punished p for each losing catch action. There are two tasks with $p = 0$ and $p = -0.5$, respectively. A prey will be removed from the grid after it being caught. An episode ends after 60 steps or all preys have been caught.

Traffic Junction (TJ). In TJ, there are N_{max} cars and the aim of each car is to complete its pre-assigned route. Each car can observe of a 3×3 region around it, but is free to communicate with other cars. The action space for each car at every time step is {`gas`, `brake`}. The reward function is $-0.01\tau + r_{\text{collision}}$, where τ is the number of time steps since the activation of the car, and $r_{\text{collision}} = -10$ is a collision penalty. Cars become available to be sampled and put back to the environment with new assigned routes once they complete their routes.

StarCraft II (SCII). We consider SMAC [96] combat scenarios where the enemy units are controlled by StarCraft II built-in AI (difficulty level is set to hard), and each of the ally units is controlled by a learning agent. The units of the two groups can be asymmetric. The action space contains `no-op`, `move[direction]`, `attack[enemy id]`, and `stop`. Agents receive a globally shared reward which calculates the total damage made to the enemy units at each time step. For the SCII environments, we conduct experiments on the following four maps to examine the attack and defence on NDQ: *3bane_vs_hM:* 3 Banelings try to kill a Hydralisk assisted by a Medivac. 3 Banelings together can just blow up the Hydralisk. Therefore, to win the game, 3 Banelings ally units should not give the Hydralisk changes for rest time when the Medivac can restore its health. *4bane_vs_hM:* Similar to 3bane_vs_hM, the main difference in 3bane_vs_hM is there are 4 Banelings. *1o_2r_vs_4r:* Ally units consist of 1 Overseer and 2 Roaches. The Overseer can find 4 Reapers and then notify its teammates to kill the invading enemy units, the Reapers, in order to win the game. At the start of an episode, ally units was spawn at a random point on the map while enemy units are initialized at another random point. Given that only the Overseer knows the position of the enemy unit,

the ability to learn to deliver this message to its teammates is vital for effectively winning the combat. *1o_3r_vs_4r:* Similar to 1o_2r_vs_4r, the main difference in 1o_3r_vs_4r is there are 1 Overseer and 3 Roaches. Fig. 5.4 shows some snapshots of the testing environments.

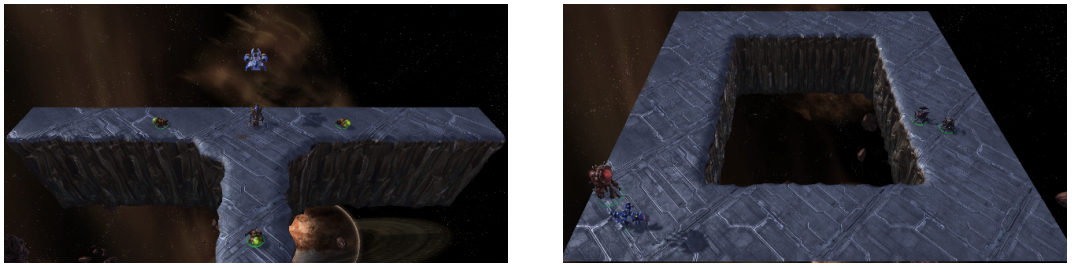


FIGURE 5.4: Examples of the StarCraft II scenarios. Left: 3bane_vs_hM. Right: 1o_2r_vs_4r.

For each of the tasks, we first train the chosen MACRL methods to obtain the action policy and the message policy for each agent. After that, we randomly select an agent to be malicious and train its adversarial policy to examine whether MACRL methods are vulnerable to message attacks. Then we perform defence by training the message filter, during which the adversarial policy of the malicious agent is fixed but keeps working. To show that a single message filter is brittle and can be easily exploited if the attacker adapts to it, we freeze the learned message filter and retrain the adversarial policy. Finally, we integrate the message filter into the framework of $\mathfrak{R}$-MACRL and justify if $\mathfrak{R}$-MACRL is helpful to improve the robustness. We train all the selected MACRL methods with the same hyperparameters as used in the corresponding works, to reproduce the performances. We use Adam [97] optimizer with a learning rate of 0.001 to train the attacker model as well as the anomaly detector and the message reconstructor. We use the default hyperparameters of Adam optimizer. α, β_1 and β_2 are all 0.001. The clip ratio in PPO is 0.5. The attacker model contains 3 linear layers and each layer has 64 neurons with ReLU activation. The same deep neural networks are used in the anomaly detector and the message reconstructor. We use PyTorch [98] to implement all the deep neural network models. All experiments are carried out with five random seeds and results are shown with a 95% confidence interval.

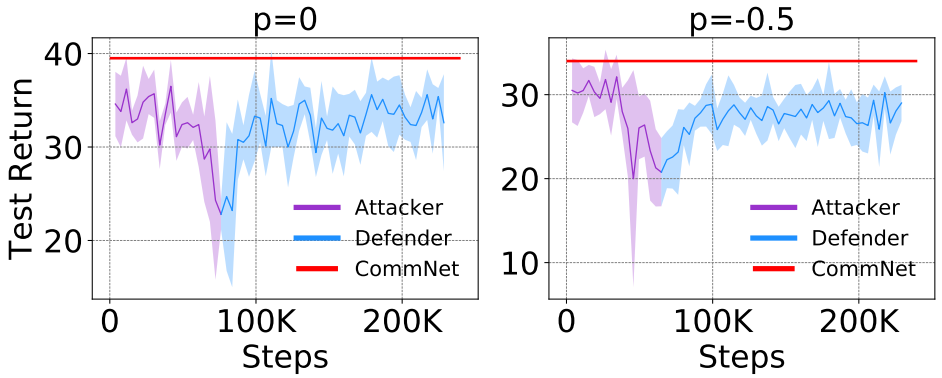


FIGURE 5.5: Attack and defence on CommNet.

5.2.2 Recovering Multi-Agent Coordination

We first evaluate the performance of our attacking method on the three selected MACRL algorithms. Then we try to recover multi-agent coordination for the attacked algorithms by applying the message filter.

CommNet. The experiments for CommNet are conducted on predator prey (PP) [95]. We set the punishment as $p = 0$ and $p = -0.5$ to create two PP tasks with different difficulties. As in Fig. 5.5, at the beginning of the attack, the performance of CommNet does not have obvious decrease, indicating that injecting random noise into the message is hard to disrupt agent coordination. As we gradually train the adversarial policy, there is a significant drop in the test return, with 40% and 33% decreases in the task of $p = 0$ and $p = -0.5$, respectively. Multi-agent cooperation has been severely affected due to the malicious messages. When the test return decreases to preset thresholds, i.e., 23 for $p = 0$ and 20 for $p = -0.5$, we freeze the adversarial policy network and start to train the message filter. As shown in the blue curves in Fig 5.5, with the message filter, test return steadily approaches the converged value of CommNet (red line), indicating that the message filter can effectively recover multi-agent coordination under attack.

TarMAC. We conduct message attack and defence on the Traffic Junction (TJ) environment [81]. There are two modes in TJ, i.e., `easy` and `hard`. The `easy` task has one junction of two one-way roads on a 7×7 grid with $N_{max} = 5$. In the `hard`

	Easy	Hard
TarMAC	$99.9 \pm 0.1\%$	$94.9 \pm 0.2\%$
TarMAC w/ π_ξ	$\mathbf{87.2 \pm 4.68\%}$	$\mathbf{88.75 \pm 7.29\%}$
TarMAC w/ π_ζ	$\mathbf{96.41 \pm 1.38\%}$	$\mathbf{93.23 \pm 8.11\%}$

TABLE 5.2: Attack and defence on TarMAC.

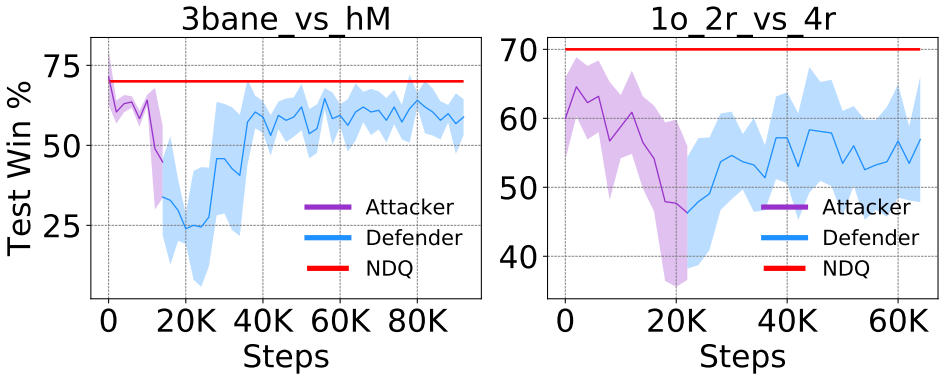


FIGURE 5.6: Attack and defence on NDQ.

task, its map has four junctions of two-way roads on a 18×18 grid and $N_{max} = 20$. As shown in Table 5.2, under attack, the success rate of TarMAC decreases in both the `easy` and the `hard` scenarios, demonstrating the vulnerability of TarMAC under malicious messages. After that, we equip TarMAC with the defender, then its performance improves considerably, demonstrating the merit of the message filter.

NDQ. We further examine the adversarial communication problem in NDQ. We perform evaluation on two StarCraft II Multi-Agent Challenge (SMAC) [96] scenarios, i.e., 3bane_vs_hM and 1o_2r_vs_4r, in which the communication between cooperative agents is necessary [15]. As presented in Fig. 5.6, under attack, the test win rate of NDQ decreases dramatically, demonstrating that NDQ is also vulnerable to message attacks. After applying the message filter as the defender, we can find the test win rate quickly reaches to around 60% in 3bane_vs_hM and 55% in 1o_2r_vs_4r, demonstrating that, once again, our defence methods succeed in restoring multi-agent coordination under message attacks.

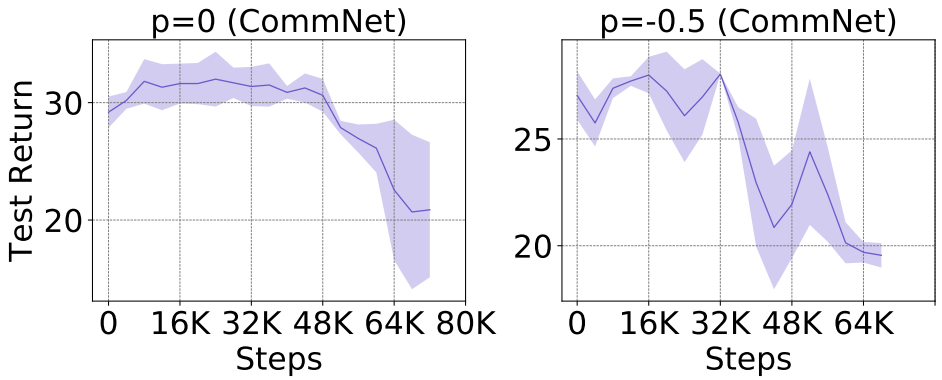


FIGURE 5.7: Exploiting the message filter in CommNet.

5.2.3 Improving Robustness with R-MACRL

We have demonstrated that many state-of-the-art MACRL algorithms are vulnerable to message attacks, and after applying the message filter, multi-agent coordination can be recovered. In this part, we integrate the message filter into the framework of $\mathfrak{R}$-MACRL and show that the robustness of MACRL algorithms can be significantly improved with $\mathfrak{R}$-MACRL.

Exploiting the message filter. We first show that a single message filter is brittle and can be easily exploited if the attacker adapts to it. We perform experiments on CommNet by freezing the message filter and retraining the adversarial policy. As depicted in Fig. 5.7, the test return of the team gradually decreases as the training proceeds, with around 30% and 20% decreases in the task of $p = 0$ and $p = -0.5$ respectively. We also conduct experiments on NDQ and similar phenomenon is observed. We conclude that even though the designed message filter is able to recover multi-agent cooperation under message attacks, its performance can degrade if faced with an adaptive attacker.

Improving robustness. To illustrate the improvement in robustness, we make comparisons between the defender trained by $\mathfrak{R}$-MACRL and the vanilla defending method. For the defender trained with $\mathfrak{R}$-MACRL, a population of attack policies are learned together with the defender, whose policy is also a mixture of sub-policies. In the vanilla training method, only a single defending policy is trained

Methods	Scenarios	$u_\zeta^{\mathfrak{R}}$	u_ζ^{vn}
CommNet	$p = 0$	$\mathbf{41.75 \pm 0.00}$	40.74 ± 0.23
	$p = -0.5$	$\mathbf{35.38 \pm 1.28}$	31.91 ± 0.94
TarMAC	Easy	$\mathbf{-4.08 \pm 0.04}$	-4.24 ± 0.08
	Hard	$\mathbf{-9.33 \pm 0.29}$	-9.80 ± 0.08
NDQ	3bane_vs_hM	$\mathbf{12.36 \pm 0.26}$	11.32 ± 0.15
	1o_2r_vs_4r	$\mathbf{17.70 \pm 0.16}$	16.33 ± 0.54

TABLE 5.3: Expected utilities for the defender trained with $\mathfrak{R}$-MACRL and the vanilla approach.

for the defender. We use the expected utility value (the accumulated team return) as the metric to compare the performance of the defender. Specifically, the larger the expected utility, the better the robustness. We denote the expected utility of the defender trained by $\mathfrak{R}$-MACRL as $u_\zeta^{\mathfrak{R}}$ and the result for the vanilla one as u_ζ^{vn}. As shown in Table 5.3, $\mathfrak{R}$-MACRL consistently outperforms the vanilla method over all the algorithms and environments. The improvement of expected utilities indicates that the defender trained by $\mathfrak{R}$-MACRL is more robust. Intuitively, the defender benefits from exploring a wider range of the policy space with $\mathfrak{R}$-MACRL, which enables the defender to maintain multi-agent coordination when faced with attacks.

5.2.4 Scaling to Multiple Attackers

To examine the performance of the method in scenarios where there is more than one attacker, we conduct experiments on NDQ in two new challenge tasks: 4bane_vs_hM and 1o_3r_vs_4r. In the former maps, i.e., 3bane_vs_hM and 1o_2r_vs_4r, there are only three agents in the team, making multiple attackers unreasonable. To mitigate this, we create 4bane_vs_hM and 1o_3r_vs_4r based on 3bane_vs_hM and 1o_2r_vs_4r by increasing one more agent to the team. We randomly sample two agents to be malicious. According to our assumptions (agents have no knowledge about who are malicious and malicious agents cannot communicate with each other), we can directly apply the attacking method to each of the malicious

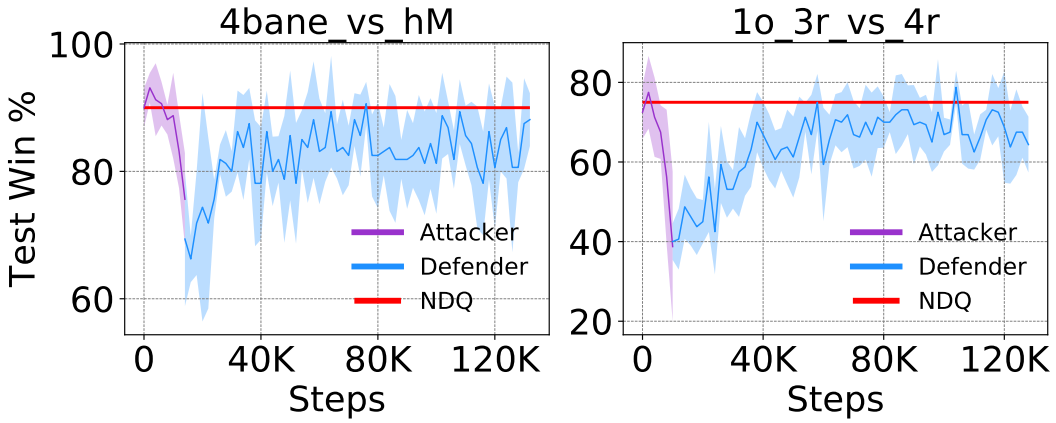


FIGURE 5.8: *Multiple attackers:* Attack and defence on NDQ.

Methods	Scenarios	$u_\zeta^{\mathfrak{R}}$	u_ζ^{vn}
NDQ	4bane_vs_hM	$\mathbf{15.86 \pm 0.08}$	15.50 ± 0.29
	1o_3r_vs_4r	$\mathbf{18.39 \pm 0.80}$	17.03 ± 0.53

TABLE 5.4: *Multiple attackers:* Expected utilities for the defender trained with $\mathfrak{R}$-MACRL and the vanilla approach.

agents. As shown in Fig. 5.8, the attackers can quickly learn effective adversarial policies with less learning steps. On the other hand, the message filter can still successfully recover multi-agent cooperation under multiple attackers, though it takes much more learning steps to converge. We further evaluate the effectiveness of $\mathfrak{R}$-MACRL on the two tasks and consistent results are observed: trained with $\mathfrak{R}$-MACRL, the agents can obtain larger expected utilities.

5.2.5 Ablation Study and Analysis

We have shown that by integrating the message filter into the framework of $\mathfrak{R}$-MACRL, the robustness of MACRL algorithms can be improved. However, the performance of $\mathfrak{R}$-MACRL is directly affected by the message filter. In this part, we will take a deeper look at how the message filter works.

Which components contribute to the performance? The message filter consists of two components: the anomaly detector and the message reconstructor. Here, we alternatively disable these two components in the message filter to study

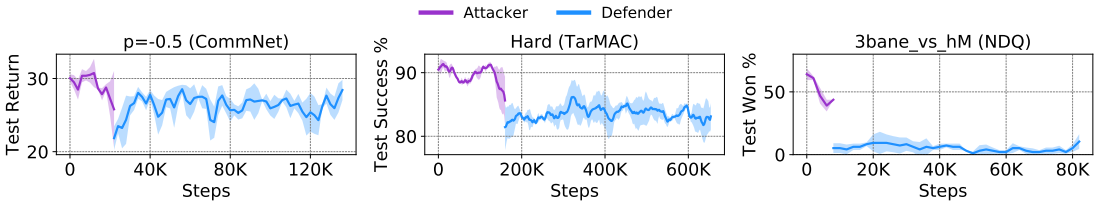


FIGURE 5.9: Defending by the message filter with the disabled message reconstructor.

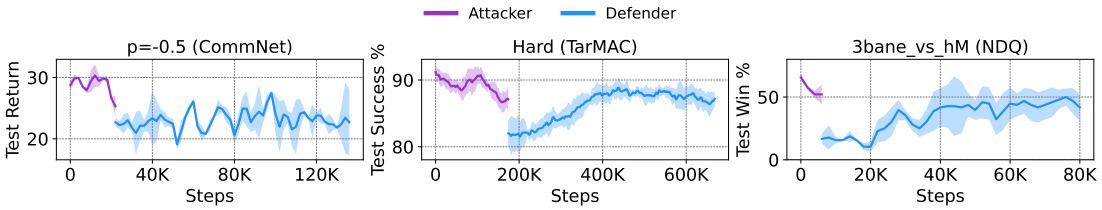


FIGURE 5.10: Defending by the message filter with the disabled anomaly detector.

how they contribute to the performance. We run experiments on three scenarios, with each scenario corresponding to an MACRL algorithm. Following the former training procedure, we first train the original MACRL algorithm to obtain the action policy and the message policy; then we sample an attacker and train its adversarial policy by PPO; next we freeze the policy of the attacker and train the message filter (with the anomaly detector or the reconstructor disabled). As in Fig. 5.9, if we disable the message reconstructor in the message filter and replace it with a random message generator, after being attacked, multi-agent coordination is hard to be recovered, demonstrating the importance of the reconstructor. We further disable the anomaly detector and randomly choose an agent to reconstruct its message, as in Fig. 5.10, the defending performance is also poor. The ablation study illustrates that both the anomaly detector and the reconstructor are critical in the message filter.

How does the proposed method work? Now we take a deeper look at how the message filter works. We conduct experiments on NDQ in the 3bane_vs_hM task. There are three agents in the team, of which agent 0 is the attacker and the others, namely agent 1 and agent 2, are benign agents. As shown in Fig. 5.11 (a)-(c), the action space of each agent contains eight elements. White cells in Q

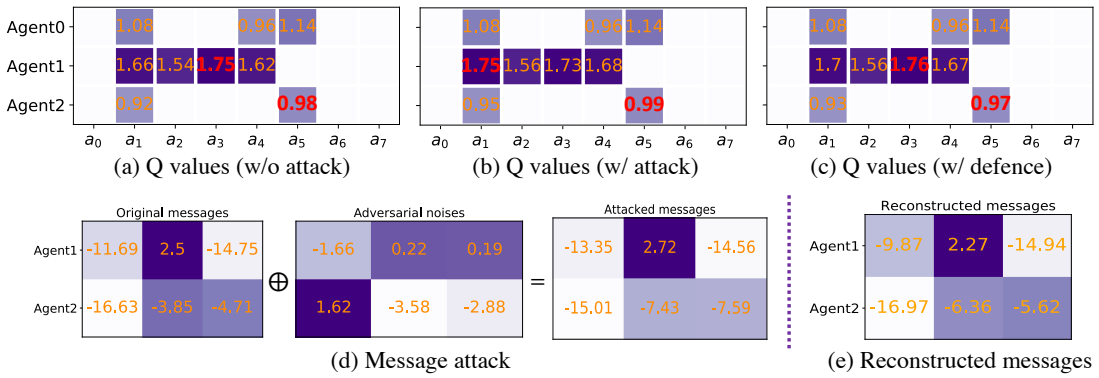


FIGURE 5.11: Q values and received messages under attack and defence. (a)-(c): Under attack, the optimal action of the benign agent 1 shifts from a_3 to a_1. After applying the message filter, its optimal action is recovered. (d)-(e): The attacked messages become quite different from the original messages, while the reconstructed messages are to similar the original.

values correspond to illegal actions at the current state, and the action with red Q value is optimal, e.g., a_5 of agent 2. When attacked by the agent 0, the optimal action of agent 1 shifts from a_3 to a_1, leading to sub-optimality. After applying the message filter, the decision of agent 1 is corrected to a_3. We further examine the messages received by agent 1 and agent 2. As in Fig. 5.11 (d)-(e), the attacked messages are quite different from the original messages, while after applying the message filter, the distance between the original messages and the reconstructed messages becomes closer.

5.3 Chapter Summary

In this chatper, we systematically examine the problem of adversarial communication in MACRL. The problem is of importance but has been largely neglected before. We first provide the formulation of adversarial communication. Then we propose an effective method to model message attacks in MACRL. Following that, we design a two-stage message filter to defend against message attacks. Finally, to improve robustness, we formulate the adversarial communication problem as a two-player zero-sum game and propose $\mathfrak{R}$-MACRL to improve the robustness. Experiments on a wide range of algorithms and tasks show that many state-of-the-art

MACRL methods are vulnerable to message attacks, while our algorithm can consistently recover multi-agent cooperation and improve the robustness of MACRL algorithms under message attacks.

Chapter 6

Effective Local Policy Improvement with Residual Actor

Optimizing long-term objective is a tough task in decision making. To ease the learning, in this chapter[1], we propose a novel optimizing strategy which first reconstructs the behavior policy and then improve the reconstructed policy in its nearby region. We study a specific problem, i.e., improving long-term user engagement in sequential recommendation, and evaluate our algorithm, ResAct, in RL-based recommender systems.

Sequential recommendation has achieved remarkable success in various fields such as news recommendation [29, 44, 100], digital entertainment [101–103], E-commerce [104, 105] and social media [106, 107]. Real-life products, such as Tiktok, have influenced the daily lives of billions of people with the support of sequential recommender systems. Different from traditional recommender systems which assume that the number of recommended items is fixed, a sequential recommender system keeps recommending items to a user until the user quits the current service/session [26, 108]. In sequential recommendation, as depicted in Figure 6.1, users have the option to

[1]The work in this chapter has been published as **Wanqi Xue**, Qingpeng Cai, Ruohan Zhan, Dong Zheng, Peng Jiang, Kun Gai, Bo An. ResAct: Reinforcing long-term engagement in sequential recommendation with residual actor [99]. *Proceedings of the 11th International Conference on Learning Representations* (**ICLR**), 2023.

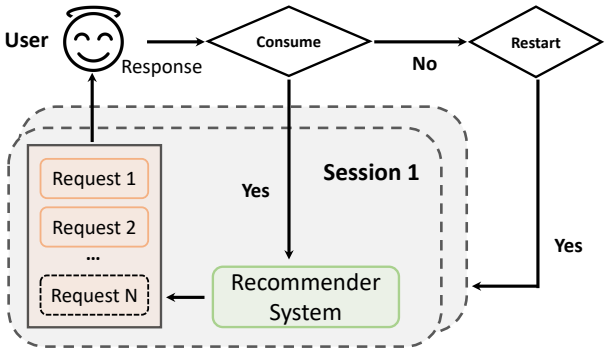


FIGURE 6.1: Sequential recommendation.

browse endless items in one session and can restart a new session after they quit the old one [25]. To this end, an ideal sequential recommender system would be expected to achieve i) low return time between sessions, i.e., high frequency of user visits; and ii) large session length so that more items can be browsed in each session. We denote these two characteristics, i.e., return time and session length, as long-term engagement, in contrast to immediate engagement which is conventionally measured by click-through rates [26]. Long-term engagement is preferred over immediate engagement in sequential recommendation as it directly affects product operational metrics such as daily active users (DAUs) and dwell time.

Despite its great importance, unfortunately, how to effectively improve long-term engagement in sequential recommendation remains largely uninvestigated. Relating the changes in long-term user engagement to a single recommendation is a tough problem [46]. Existing works on sequential recommendation have typically focused on estimating the probability of immediate engagement with various neural network architectures [26, 105]. However, they neglect to explicitly improve user stickiness such as increasing the frequency of visits or extending the average session length. There have been some recent efforts to optimize long-term engagement in sequential recommendation. However, they are usually based on strong assumptions such as recommendation diversity will increase long-term engagement [34, 109]. In fact, the relationship between recommendation diversity and long-term engagement is largely empirical, and how to measure diversity properly is also unclear [25].

Recently, reinforcement learning has achieved impressive advances in various sequential decision-making tasks, such as games [5, 7], autonomous driving [110] and robotics [111]. Reinforcement learning in general focuses on learning policies which maximize cumulative reward from a long-term perspective [112]. To this end, it offers us a promising framework to optimize long-term engagement in sequential recommendation [113]. We can formulate the recommender system as an agent, with users as the environment, and assign rewards to the recommender system based on users' response, for example, the return time between two sessions. However, back to reality, there are significant challenges. First, the evolvement of user stickiness lasts for a long period, usually days or months, which makes the evaluation of state-action value difficult. Second, probing for rewards in previously unexplored areas, i.e., exploration, requires live experiments and may hurt user experience. Third, rewards of long-term engagement only occur at the beginning or end of a session and are therefore sparse compared to immediate user responses. As a result, representations of states may not contain sufficient information about long-term engagement.

To mitigate the aforementioned challenges, we propose to learn a recommendation policy that is close to, but better than, the online-serving policy. In this way, i) we can collect sufficient data near the learned policy so that state-action values can be properly estimated; and ii) there is no need to perform online interaction. However, directly learning such a policy is quite difficult since we need to perform optimization in the entire policy space. Instead, our method, ResAct, achieves it by first reconstructing the online behaviors of previous recommendation models, and then improving upon the predictions via a **Res**idual **Act**or. The original optimization problem is decomposed into two sub-tasks which are easier to solve. Furthermore, to learn better representations, two information-theoretical regularizers are designed to confirm the expressiveness and conciseness of the features. We conduct experiments on a benchmark dataset and a real-world dataset consisting of tens of millions of recommendation requests. The results show that ResAct significantly outperforms previous state-of-the-art methods in various long-term

engagement optimization tasks.

6.1 Problem Statement: Sequential Recommendation

In sequential recommendation, users interact with the recommender system on a session basis. A session starts when a user opens the App and ends when he/she leaves. As in Figure 6.1, when a user starts a session, the recommendation agent begins to feed items to the user, one for each recommendation request, until the session ends. For each request, the user can choose to consume the recommended item or quit the current session. A user may start a new session after he/she exits the old one, and can consume an arbitrary number of items within a session. An ideal recommender system with a goal for long-term engagement would be expected to minimize the average return time between sessions while maximizing the average number of items consumed in a session. Formally, we describe the sequential recommendation problem as a Markov Decision Process (MDP) which is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$. $\mathcal{S} = \mathcal{S}_h \times \mathcal{S}_l$ is the continuous state space. $s \in \mathcal{S}$ indicates the state of a user. Considering the session-request structure in sequential recommendation, we decompose $\mathcal{S}$ into two disjoint sub-spaces, i.e., $\mathcal{S}_h$ and $\mathcal{S}_l$, which is used to represent session-level (high-level) features and request-level (low-level) features, respectively. $\mathcal{A}$ is the continuous action space [114, 115], where $a \in \mathcal{A}$ is a vector representing a recommended item. $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition function, where $p(s_{t+1}|s_t, a_t)$ defines the state transition probability from the current state s_t to the next state s_{t+1} after recommending an item a_t. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r(s_t, a_t)$ is the immediate reward by recommending a_t at state s_t. The reward function should be related to return time and/or session length; γ is the discount factor for future rewards. Given a policy $\pi(a|s) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, we define a state-action value function $Q^\pi(s, a)$ which outputs the expected cumulative reward (return) of taking an action a at state s

and thereafter following π:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{(s_{t'}, a_{t'}) \sim \pi} \left[r(s_t, a_t) + \sum_{t'=t+1}^{\infty} \gamma^{(t'-t)} \cdot r(s_{t'}, a_{t'}) \right]. \tag{6.1}$$

The optimization objective is to seek a policy $\pi(a|s)$ such that the return obtained by the recommendation agents is maximized, i.e., $\max_{\pi} \mathcal{J}(\pi) = \mathbb{E}_{s_t \sim d_t^{\pi}(\,\cdot\,), a_t \sim \pi(\,\cdot\,|s_t)} [Q^{\pi}(s_t, a_t)]$. Here $d_t^{\pi}(\,\cdot\,)$ denotes the state visitation frequency at step t under the policy π.

6.2 Reinforcing Long-term Engagement with Residual Actor

To improve long-term engagement, we propose to learn a recommendation policy which is broadly consistent with, but better than, the policies used to serve users online (online-serving policy). The online-serving policy is a historical policy or a mixture of policies which generate logged data to approximate the MDP in sequential recommendation. In this way, i) we have access to sufficient data near the learned policy so that state-action values can be properly estimated because the notorious extrapolation error is minimized [116]; and ii) the potential of harming the user experience is reduced as we can easily control the divergence between the learned new policy and the deployed policy (the online-serving policy) and there is no need to perform online interaction. Despite the advantages, directly learning such a policy is rather difficult because we need to perform optimization throughout the entire huge policy space. Instead, we propose to achieve it by first reconstructing the online-serving policy and then improving it. By doing so, the original optimization problem is decomposed into two sub-tasks which are more manageable.

Specifically, let $\hat{\pi}(a|s)$ denotes the policy we want to learn; we decompose it into $\hat{a} = a_{on} + \Delta(s, a_{on})$ where a_{on} is sampled from the online-serving policy π_{on}, i.e., $a_{on} \sim \pi_{on}(a|s)$, and $\Delta(s, a_{on})$ is the residual which is determined by a deterministic

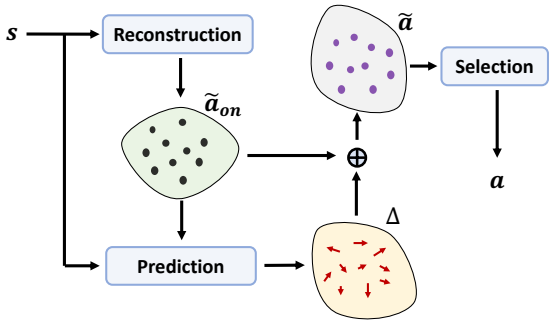


FIGURE 6.2: Workflow of ResAct.

actor. We expect that adding the residual will lead to higher expected return, i.e., $\mathcal{J}(\hat{\pi}) \geq \mathcal{J}(\pi_{on})$. As in Figure 6.2, our algorithm, ResAct, works in three phases:

i) **Reconstruction:** ResAct first reconstructs the online-serving policy, i.e., $\tilde{\pi}_{on}(a|s) \approx \pi_{on}(a|s)$, by supervised learning. Then ResAct samples n actions from the reconstructed policy, i.e., $\{\tilde{a}^i_{on} \sim \tilde{\pi}_{on}(a|s)\}^n_{i=1}$ as estimators of a_{on};

ii) **Prediction:** For each estimator $\tilde{a}^i_{on}$, ResAct predicts the residual and applies it to $\tilde{a}^i_{on}$, i.e., $\tilde{a}^i = \tilde{a}^i_{on} + \Delta(s, \tilde{a}^i_{on})$. We need to learn the residual actor to predict $\Delta(s, \tilde{a}_{on})$ such that $\tilde{a}$ is better than $\tilde{a}_{on}$ in general;

iii) **Selection:** ResAct selects the best action from the $\{\tilde{a}^i\}^n_{i=0}$ as the final output, i.e., $\arg\max_{\tilde{a}} Q^{\hat{\pi}}(s, \tilde{a})$ for $\tilde{a} \in \{\tilde{a}^i\}^n_{i=0}$.

In sequential recommendation, state representations may not contain sufficient information about long-term engagement. To address this, we design two information-theoretical regularizers to improve the expressiveness and conciseness of the extracted state features. The regularizers are maximizing mutual information between state features and long-term engagement while minimizing the entropy of the state features in order to filter out redundant information. The overview of ResAct is depicted in Figure 6.3 and we elaborate the details in the subsequent subsections.

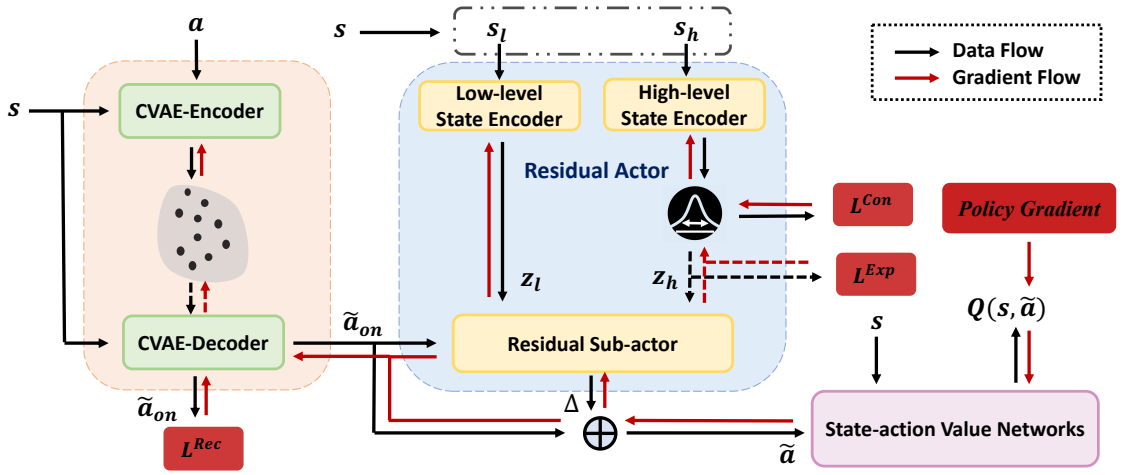


FIGURE 6.3: Schematics of our approach. The CVAE-Encoder generates an action embedding distribution, from which a latent vector is sampled for the CVAE-Decoder to reconstruct the action. The reconstructed action $\tilde{a}_{on}$, together with state features extracted by the high-level and low-level state encoders, are fed to the residual actor to predict the residual Δ. After adding the residual, the action and the state are sent to the state-action value networks, from which policy gradient can be generated. The framework can be trained in an end-to-end manner.

6.2.1 Reconstructing Online Behaviors

To reconstruct behaviors of the online-serving policy, we should learn a mapping $\tilde{\pi}_{on}(a|s)$ from states to action distributions such that $\tilde{\pi}_{on}(a|s) \approx \pi_{on}(a|s)$ where $\pi_{on}(a|s)$ is the online-serving policy. A naive approach is to use a model $D(a|s;\theta_d)$ with parameters θ_d to approximate $\pi_{on}(a|s)$ and optimize θ_d by minimizing

$$\mathbb{E}_{s,a_{on}\sim\pi_{on}(a|s)}\left[(D(a|s;\theta_d) - a_{on})^2\right]. \tag{6.2}$$

However, such deterministic action generation only allows for an instance of action and will cause huge deviation if the only estimator is not precise. To mitigate this, we propose to encode a_{on} into a latent distribution conditioned on s, and decode samples from the latent space to get estimators of a_{on}. By doing so, we can generate multiple action estimators by sampling from the latent distribution. The key idea is inspired by conditional variational auto-encoder (CVAE) [117]. We define the latent distribution $\mathcal{C}(s, a_{on})$ as a multivariate Gaussian whose parameters, i.e., mean and variance, are determined by an encoder $E(\cdot\,|s, a_{on};\theta_e)$ with parameters θ_e. Then for

each latent vector $c \sim \mathcal{C}(s, a_{on})$, we can use a decoder $D(a|s, c; \theta_d)$ with parameters θ_d to map it back to an action. To improve generalization ability, we apply a KL regularizer which controls the deviation between $\mathcal{C}(s, a_{on})$ and its prior which is chosen as the multivariate normal distribution $\mathcal{N}(0, 1)$. Formally, we can optimize θ_e and θ_d by minimizing the following loss:

$$L^{Rec}_{\theta_e,\theta_d} = \mathbb{E}_{s,a_{on},c}\left[(D(a|s, c; \theta_d) - a_{on})^2 + KL(\mathcal{C}(s, a_{on}; \theta_e)||\mathcal{N}(0, 1))\right]. \tag{6.3}$$

where $a_{on} \sim \pi_{on}(a|s)$ and $c \sim \mathcal{C}(s, a_{on}; \theta_e)$[2]. When performing behavior reconstruction for an unknown state s, we do not know its a_{on} and therefore cannot build $\mathcal{C}(s, a_{on}; \theta_e)$. As a mitigation, we sample n latent vectors from the prior of $\mathcal{C}(s, a_{on})$, i.e., $\{c^i \sim \mathcal{N}(0, 1)\}^n_{i=0}$. Then for each c^i, we can generate an estimator of a_{on} by using the decoder $\tilde{a}^i_{on} = D(a|s, c^i; \theta_d)$.

6.2.2 Local Policy Improvement: Learning to Predict the Optimal Residual

By learning the CVAE which consists of $E(\cdot\,|s, a_{on}; \theta_e)$ and $D(a|s, c; \theta_d)$, we can easily reconstruct the online-serving policy and sample multiple estimators of a_{on} by $\{\tilde{a}^i_{on} = D(a|s, c^i; \theta_d), c^i \sim \mathcal{N}(0, 1)\}^n_{i=0}$. For each $\tilde{a}^i_{on}$, we should predict the residual $\Delta(s, \tilde{a}^i_{on})$ such that $\tilde{a}^i = \tilde{a}^i_{on} + \Delta(s, \tilde{a}^i_{on})$ is better than $\tilde{a}^i_{on}$. We use a model $f(\Delta|s, a; \theta_f)$ with parameters θ_f to approximate the residual function $\Delta(s, a)$. Particularly, the residual actor $f(\Delta|s, a; \theta_f)$ consists of a state encoder and a sub-actor, which are for extracting features from a user state and predicting the residual based on the extracted features, respectively. Considering the bi-level session-request structure in sequential recommendation, we design a hierarchical state encoder consisting of a high-level encoder $f_h(s_h; \theta_h)$ and a low-level encoder $f_l(s_l; \theta_l)$ for extracting features from session-level (high-level) state s_h and request-level (low-level) state s_l, respectively. To conclude, the residual actor $f(\Delta|s, a; \theta_f) = \{f_h, f_l, f_a\}$

[2] $\mathcal{C}(s, a_{on}; \theta_e)$ is parameterized by θ_e because it is a multivariate Gaussian whose mean and variance are the output of the encoder $E(\cdot\,|s, a_{on}; \theta_e)$.

works as follows:

$$z_h = f_h(s_h;\theta_h), z_l = f_l(s_l;\theta_l); z = Concat(z_h, z_l); \Delta = f_a(z, a;\theta_a). \tag{6.4}$$

Where z_h and z_l are the extracted high-level and low-level features, respectively; z is the concatenation of z_h and z_l, and $f_a(z, a;\theta_a)$ parameterized by θ_a is the sub-actor. Here, $\theta_f = \{\theta_h, \theta_l, \theta_a\}$.

Given a state s and a sampled latent vector $c \sim \mathcal{N}(0, 1)$, ResAct generates an action with a deterministic policy $\hat{\pi}(a|s, c) = D(\tilde{a}_{on}|s, c;\theta_d) + f(\Delta|s, \tilde{a}_{on};\theta_f)$. We want to optimize the parameters $\{\theta_d, \theta_f\}$ of $\hat{\pi}(a|s, c)$ so that the expected cumulative reward $\mathcal{J}(\hat{\pi})$ is maximized. Based on the Deterministic Policy Gradient (DPG) theorem [118, 119], we derive the following performance gradients. We begin by deriving the gradients of $\mathcal{J}(\hat{\pi})$ with respect to the parameters of the residual actor.

$$\begin{aligned}
\nabla_{\theta_f}\mathcal{J}(\hat{\pi}) &= \iint p(c)p^{\hat{\pi}}(s)\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_f}\hat{\pi}(a|s, c)\mathrm{d}c\mathrm{d}s \\
&= \iint p(c)p^{\hat{\pi}}(s)\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_f} f(\Delta|s, a;\theta_f)|_{a=D(a|s,c;\theta_d)}\mathrm{d}c\mathrm{d}s \\
&= \mathbb{E}_{s,c}\left[\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_f} f(\Delta|s, a;\theta_f)|_{a=D(a|s,c;\theta_d)}\right]
\end{aligned} \tag{6.5}$$

The decoder $D(a|s, c;\theta_d)$ also affects the policy. The gradients of $\mathcal{J}(\hat{\pi})$ with respect to θ_d) is derived similarly:

$$\begin{aligned}
\nabla_{\theta_d}\mathcal{J}(\hat{\pi}) &= \iint p(c)p^{\hat{\pi}}(s)\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_d}\hat{\pi}(a|s, c)\mathrm{d}c\mathrm{d}s \\
&= \iint p(c)p^{\hat{\pi}}(s)\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_d} D(a|s, c;\theta_d)\mathrm{d}c\mathrm{d}s \\
&= \mathbb{E}_{s,c}\left[\nabla_a Q^{\hat{\pi}}(s, a)|_{a=\hat{\pi}(a|s,c)}\nabla_{\theta_d} D(a|s, c;\theta_d)\right]
\end{aligned} \tag{6.6}$$

Here $\hat{\pi}(a|s, c) = D(\tilde{a}_{on}|s, c;\theta_d) + f(\Delta|s, \tilde{a}_{on};\theta_f)$, $p(\cdot)$ is the probability function of a random variable, $Q^{\hat{\pi}}(s, a)$ is the state-action value function for $\hat{\pi}$.

To learn the state-action value function, referred to as *critic*, $Q^{\hat{\pi}}(s, a)$ in Eq. (6.5) and Eq. (6.6), we adopt Clipped Double Q-learning [120] with two models $Q_1(s, a;\theta_{q_1})$ and $Q_2(s, a;\theta_{q_2})$ to approximate it. For transitions (s_t, a_t, r_t, s_{t+1}) from logged data,

we optimize θ_{q_1} and θ_{q_2} to minimize the following Temporal Difference (TD) loss:

$$\begin{aligned} L_{\theta_{q_j}}^{TD} &= \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})}\left[(Q_j(s_t,a_t;\theta_{q_j})-y)^2\right], j=\{1,2\}; \\ y &= r_t+\gamma \min\left[Q_1^{'}(s_{t+1},\hat{\pi}^{'}(a_{t+1}|s_{t+1});\theta_{q_1}^{'}), Q_2^{'}(s_{t+1},\hat{\pi}^{'}(a_{t+1}|s_{t+1});\theta_{q_2}^{'})\right]. \end{aligned} \tag{6.7}$$

Where $Q_1^{'}$, $Q_2^{'}$, and $\hat{\pi}^{'}$ are target models whose parameters are soft-updated to match the corresponding models [120].

According to the DPG theorem, we can update the parameters θ_f in the direction of $\nabla_{\theta_f}\mathcal{J}(\hat{\pi})$ to gain a value improvement in $\mathcal{J}(\hat{\pi})$:

$$\theta_f \leftarrow \theta_f + \nabla_{\theta_f}\mathcal{J}(\hat{\pi}), \theta_f = \{\theta_h, \theta_l, \theta_a\}. \tag{6.8}$$

For θ_d, since it also needs to minimize $L_{\theta_e,\theta_d}^{Rec}$, thus the updating direction is

$$\theta_d \leftarrow \theta_d + \nabla_{\theta_d}\mathcal{J}(\hat{\pi}) - \nabla_{\theta_d}L_{\theta_e,\theta_d}^{Rec}. \tag{6.9}$$

Based on $\hat{\pi}(a|s,c)$, theoretically, we can obtain the policy $\hat{\pi}(a|s)$ by marginalizing out the latent vector c: $\hat{\pi}(a|s) = \int p(c)\hat{\pi}(a|s,c)\mathrm{d}c$. This integral can be approximated as $\hat{\pi}(a|s) \approx \frac{1}{n}\sum_{i=0}^{n}\hat{\pi}(a|s,c^i)$ where $\{c^i \sim \mathcal{N}(0,1)\}_{i=0}^{n}$. However, given that we already have a critic $Q_1(s,a;\theta_{q_1})$, we can alternatively use the critic to select the final output:

$$\begin{aligned} \hat{\pi}(a|s) &= \hat{\pi}(a|s,c^*); \\ c^* &= \arg\max_c Q_1(s,\hat{\pi}(a|s,c);\theta_{q_1}), c \in \{c^i \sim \mathcal{N}(0,1)\}_{i=0}^{n} \end{aligned} \tag{6.10}$$

6.2.3 Facilitating Feature Extraction with Information-theoretical Regularizers

Good state representations always ease the learning of models [121]. Considering that session-level states $s_h \in \mathcal{S}_h$ contain rich information about long-term engagement, we design two information-theoretical regularizers to facilitate the feature

extraction. Generally, we expect the learned features to have **Expressiveness** and **Conciseness**. To learn features with the desired properties, we propose to encode session-level state s_h into a stochastic embedding space instead of a deterministic vector. Specifically, s_h is encoded into a multivariate Gaussian distribution $\mathcal{N}(\mu_h, \sigma_h)$ whose parameters μ_h and σ_h are predicted by the high-level encoder $f_h(s_h; \theta_h)$. Formally, $(\mu_h, \sigma_h) = f_h(s_h; \theta_h)$ and $z_h \sim \mathcal{N}(\mu_h, \sigma_h)$ z_h is the representation for session-level state s_t. Next, we introduce how to achieve expressiveness and conciseness in z_h.

Expressiveness.

We expect the extracted features to contain as much information as possible about long-term engagement rewards, suggesting an intuitive approach to maximize the mutual information between z_h and $r(s, a)$. The mutual information $I_{\theta_h}(z_h; r)$ is defined according to

$$\begin{aligned} I_{\theta_h}(z_h; r) &= \iint p(z_h, r) \log \frac{p(z_h, r)}{p\left(z_h\right) p\left(r\right)} \mathrm{d}z_h \mathrm{d}r \\ &= \iint p_{\theta_h}(z_h) p(r|z_h) \log \frac{p(r|z_h)}{p\left(r\right)} \mathrm{d}z_h \mathrm{d}r \end{aligned} \tag{6.11}$$

However, estimating and maximizing mutual information is practically intractable. Inspired by variational inference [122], we derive a tractable lower bound for the mutual information objective. Considering that $KL(p(r|z_h)||q(r|z_h)) \geq 0$, by the definition of KL-divergence, we have $\int p(r|z_h) \log p(r|z_h) \mathrm{d}r \geq \int p(r|z_h) \log q(r|z_h) \mathrm{d}r$ where $q(r|z_h)$ is an arbitrary distribution. Here, we introduce $o(r|z_h; \theta_o)$ as a variational neural estimator with parameters θ_o of $p(r|z_h)$. Then,

$$\begin{aligned} I_{\theta_h}(z_h; r) &\geq \iint p_{\theta_h}(z_h) p(r|z_h) \log \frac{o(r|z_h; \theta_o)}{p\left(r\right)} \mathrm{d}z_h \mathrm{d}r \\ &= \iint p_{\theta_h}(z_h) p(r|z_h) \log o(r|z_h; \theta_o) \mathrm{d}z_h \mathrm{d}r + H(r) \end{aligned} \tag{6.12}$$

where $o(r|z_h; \theta_o)$ is a variational neural estimator of $p(r|z_h)$ with parameters θ_o, $H(r) = -\int p(r) \log p(r) \mathrm{d}r$ is the entropy of reward distribution. Since $H(r)$ only depends on user responses and stays fixed for the given environment, we can turn

to maximize a lower bound of $I_{\theta_h}(z_h; r)$ which leads to the following expressiveness loss:

$$\begin{aligned} L^{Exp}_{\theta_h,\theta_o} &= -\iint p_{\theta_h}(z_h)p(r|z_h)\log o(r|z_h;\theta_o)\mathrm{d}z_h\mathrm{d}r \\ &= -\iiint p(s)p_{\theta_h}(z_h|s)p(r|s,z_h)\log o(r|z_h;\theta_o)\mathrm{d}s\mathrm{d}z_h\mathrm{d}r \\ &= -\iiint p(s)p_{\theta_h}(z_h|s_h)p(r|s)\log o(r|z_h;\theta_o)\mathrm{d}s\mathrm{d}z_h\mathrm{d}r \\ &= \mathbb{E}_{s,z_h\sim p_{\theta_h}(z_h|s_h)}\left[-\int p(r|s)\log o(r|z_h;\theta_o)\mathrm{d}r\right] \\ &= \mathbb{E}_{s,z_h\sim p_{\theta_h}(z_h|s_h)}\left[\mathcal{H}(p(r|s)||o(r|z_h;\theta_o))\right] \end{aligned} \tag{6.13}$$

where s is state, s_h is session-level state, $p_{\theta_h}(z_h|s_h) = \mathcal{N}(\mu_h, \sigma_h)$, and $\mathcal{H}(\cdot\,||\,\cdot)$ denotes the cross entropy between two distributions. By minimizing $L^{Exp}_{\theta_h,\theta_o}$, we confirm the expressiveness of z_h.

Conciseness. If maximizing $I_\theta(z_h; r)$ is the only objective, we could always ensure a maximally informative representation by taking the identity encoding of session-level state ($z_h = s_h$) [122]; however, such an encoding is not useful. Thus, apart from expressiveness, we want z_h to be concise enough to filter out redundant information from s_h. To achieve this goal, we also want to minimize $I_{\theta_h}(z_h; s_h) = \iint p(s_h)p_{\theta_h}(z_h|s_h)\log\frac{p_{\theta_h}(z_h|s_h)}{p_{\theta_h}(z_h)}\mathrm{d}s_h\mathrm{d}z_h$ such that z_h is the minimal sufficient statistic of s_h for inferring r. Computing the marginal distribution of z_h, $p_{\theta_h}(z_h)$, is usually intractable. So we introduce $m(z_h)$ as a variational approximation to $p_{\theta_h}(z_h)$, which is conventionally chosen as the multivariate normal distribution $\mathcal{N}(0, 1)$. Since $KL(p_{\theta_h}(z_h)||m(z_h)) \geq 0$, we can easily have the following upper bound:

$$I_{\theta_h}(z_h; s_h) \leq \iint p(s_h)p_{\theta_h}(z_h|s_h)\log\frac{p_{\theta_h}(z_h|s_h)}{m(z_h)}\mathrm{d}s_h\mathrm{d}z_h. \tag{6.14}$$

Minimizing this upper bound leads to the following conciseness loss:

$$\begin{aligned} L^{Con}_{\theta_h} &= \int p(s_h)\left[\int p_{\theta_h}(z_h|s_h)\log\frac{p_{\theta_h}(z_h|s_h)}{m(z_h)}\mathrm{d}z_h\right]\mathrm{d}s_h; \\ &= \mathbb{E}_s\left[KL(p_{\theta_h}(z_h|s_h)||m(z_h))\right]. \end{aligned} \tag{6.15}$$

By minimizing $L^{Con}_{\theta_h}$, we achieve conciseness in z_h.

6.2.4 Overall Algorithm

We provide the learning process of ResAct in Algorithm 5. Particularly, the CVAE is trained to reconstruct the online-serving policy, the residual actor is trained for predicting the optimal residual for each reconstructed action, and the critic networks is trained to guide the optimization of the decoder in CVAE and the residual actor. For the target networks (line 2), $\{\theta_d, \theta_h, \theta_l, \theta_a, \theta_{q_1}\}$ is for the target policy, and $\{\theta_{q_1}, \theta_{q_2}\}$ is for the target critics. The execution process of ResAct is summarized in Algorithm 6. Only the decoder of the CVAE, the residual actor and one of the critics are used during execution.

Algorithm 5: ResAct-LEARNING

Input: Logged data collected by the online-serving policy
$\mathcal{D} = \{(s_t, a_t, r_t, s_{t+1})\}$

1 Initialize the CVAE: $\{E(c|s,a;\theta_e), D(a|s,c;\theta_d)\}$, the residual actor: $\{f_h(z_h|s_h;\theta_h), f_l(z_l|s_l;\theta_l), f_a(\Delta|z,a;\theta_a)\}$, the critic networks $\{Q_1(s,a;\theta_{q_1}), Q_2(s,a;\theta_{q_2})\}$, and the variational estimator $o(r|z_h;\theta_o)$
2 Set soft-update rate τ and initialize the target networks $\theta' \leftarrow \theta$ for $\theta \in \{\theta_d, \theta_h, \theta_l, \theta_a, \theta_{q_1}, \theta_{q_2}\}$
3 **for** $k = 1$ *to* K **do**
4 Sample a batch of transitions (s_t, a_t, r_t, s_{t+1}) from $\mathcal{D}$
5 $\theta_e \leftarrow \theta_e - \nabla_{\theta_e} L^{Rec}_{\theta_e,\theta_d}$ ($L^{Rec}_{\theta_e,\theta_d}$ is in Eq. 6.3)
6 Update θ_d according to Eq. 6.9
7 Update $\{\theta_h, \theta_l, \theta_a\}$ according to Eq. 6.8
8 $\theta_{q_j} \leftarrow \theta_{q_j} - \nabla_{\theta_{q_j}} L^{TD}_{\theta_{q_j}}$, $j = \{1, 2\}$ ($L^{TD}_{\theta_{q_j}}$ is in Eq. 6.7)
9 $\theta_h \leftarrow \theta_h - \nabla_{\theta_h} L^{Exp}_{\theta_h,\theta_o} - \nabla_{\theta_h} L^{Con}_{\theta_h}$
10 $\theta_o \leftarrow \theta_o - \nabla_{\theta_o} L^{Exp}_{\theta_h,\theta_o}$ ($L^{Exp}_{\theta_h,\theta_o}$ is in Eq. 6.13, and $L^{Con}_{\theta_h}$ is in Eq. 6.15)
11 Update the target networks: $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$ for $\theta \in \{\theta_d, \theta_h, \theta_l, \theta_a, \theta_{q_1}, \theta_{q_2}\}$
12 **end**

6.3 Experimental Evaluation

Algorithm 6: ResAct-EXECUTION

Input: State s, number of estimators n

Generate n estimators of a_{on}: $\{\tilde{a}^i_{on} = D(a|s, c^i; \theta_d), c^i \sim \mathcal{N}(0,1)\}^n_{i=0}$

for $\tilde{a}_{on} \in \{\tilde{a}^i_{on}\}^n_{i=0}$ **do**

  Predict the residual $\Delta = f(\Delta|s, \tilde{a}_{on}; \theta_f)$ as in Eq. 6.4

  Apply the residual: $\tilde{a} = \tilde{a}_{on} + \Delta$

end

$a^* = \arg\max_a Q_1(s, a; \theta_{q_1})$, $a \in \{\tilde{a}^i\}^n_{i=0}$

Output: Action a^*

We conduct experiments on a synthetic dataset *MovieLensL-1m* and a real-world dataset *RecL-25m* to demonstrate the effectiveness of ResAct. We are particularly interested in : *Whether ResAct is able to achieve consistent improvements over previous state-of-the-art methods? If yes, why?*

6.3.1 Experimental Settings

Datasets. As there is no public dataset explicitly containing signals about long-term engagement, we synthesize a dataset named *MovieLensL-1m* based on *MovieLens-1m* (a popular benchmark for evaluating recommendation algorithms) and collected a large-scale industrial dataset *RecL-25m* from a real-life streaming platform of short-form videos. *MovieLensL-1m* is constructed by assuming that long-term engagement is proportional to the movie ratings (5-star scale) in *MovieLens-1m*. *RecL-25m* is collected by tracking the behaviors of 99,899 users (randomly selected from the platform) for months and recording their long-term engagement indicators, i.e., return time and session length. The statistics of *RecL-25m* are provided in Table 6.1, where 25% and 75% denote the corresponding percentile. We did not count the average return time because there are users appearing only once whose return time may go to infinity. The state of a user contains information about gender, age, and historical interactions such as like rate and forward rate. The item to recommend is determined by comparing the inner product of an action and the embedding of videos [115].

TABLE 6.1: Statistics of *RecL-25m*.

	Users	**Sessions**	**Requests**
	99,899	6,126,583	25,921,753
	Avg return time (h)	**Avg session length**	**Avg # of sessions**
Mean	-	4.0449	61.3277
75%	11.2794	4.8792	85
25%	4.3264	2.1358	30

Designing rewards. The rewards of long-term engagement in *RecL-25m* are designed based on the statistics of the dataset. As a general guideline, we expect rewards to reflect the influence of recommending an item on a user. However, behaviors of users have large variance which makes the influence difficult to measure. For example, if we simply make rewards proportional to session length, or inversely proportional to return time, the recommender system would focus on improving the experience of high activity users, because by doing so it can obtain larger rewards. However, in reality, it is equally if not more important to facilitate the conversion of low activity user to high activity user, which requires us to improve the experience of low activity users. To address this issue, we turn to measuring the **relative** influence of an item. Concretely, we calculate the average return time δ^u_{avg} and the average session length η^u_{avg} for a user u, and use these two statistics to quantify rewards. For user u, given a time duration δ^u between two sessions, the corresponding reward is calculated by

$$r(\delta^u) = \left(\lfloor \frac{\min(\delta^u_{avg}, \delta_{75\%})}{\delta^u} \rfloor \right) .clip(0, 5) \tag{6.16}$$

where $\delta_{75\%}$ is the 75th percentile of the average return time for all users, which is designed to differentiate active users and inactive users. Rewards for the session length is calculated similarly as

$$r(\eta^u) = \left(\lfloor \frac{\eta^u}{\eta^u_{avg} \times 0.8} \rfloor \right) .clip(0, 5) \tag{6.17}$$

where η^u is the length of a session in the logged data of user u. Since δ^u and δ can only be calculated at session-level, without loss of generality, we provide rewards at the end of each session, where rewards for return time is assigned to the previous

session.

For *MovieLens-1m*, as the dataset does not contain any information about long-term user engagement, to generalize it to long-term engagement problem, we make an assumption that a user's long-term engagement is proportional to the movie ratings. Specifically, we assume that recommending a movie for which a user rates 2-stars will not affect engagement, a movie with 3-stars, 4-stars and 5-stars will benefit the long-term engagement by 1, 2, and 3, respectively. Recommending a movie with 1-star is harmful to engagement and will be given a negative reward, -1. The task in *MovieLensL-1m* is to maximize cumulative benefits on long-term engagement.

Evaluation Metric and Baselines. We adopt Normalised Capped Importance Sampling (NCIS) [123], a standard offline evaluation method [124, 125], to assess the performance of different policies. Given that π_β is the behavior policy, π is the policy to assess, we evaluate the value by

$$\tilde{J}^{NCIS}(\pi) = \frac{1}{|\mathcal{T}|} \sum_{\xi \in \mathcal{T}} \left[\frac{\sum_{(s,a,r)\in\xi} \tilde{\rho}_{\pi,\pi_\beta}(s,a) r}{\sum_{(s,a,r)\in\xi} \tilde{\rho}_{\pi,\pi_\beta}(s,a)} \right], \quad \tilde{\rho}_{\pi,\pi_\beta}(s,a) = \min\left(c, \frac{\phi_{\pi(s)}(a)}{\phi_{\pi_\beta(s)}(a)}\right). \tag{6.18}$$

Here $\mathcal{T}$ is the testing set with usage trajectories, $\phi_{\pi(s)}$ denotes a multivariate Gaussian distribution of which mean is given by $\pi(s)$, c is a clipping constant to stabilize the evaluation. We compare our method with various baselines, including classic reinforcement learning methods (DDPG, TD3), reinforcement learning with offline training (TD3_BC, BCQ, IQL), and imitation learning methods (IL, IL_CVAE).

Baselines. Our method is compared with various baselines, including classic reinforcement learning methods (DDPG, TD3), offline reinforcement learning algorithms (TD3_BC, BCQ, IQL), and imitation learning methods (IL, IL_CVAE):

- **DDPG** [119]: A reinforcement learning algorithm which concurrently learns a Q-function and a policy. It uses the Q-function to guide the optimization of the policy.

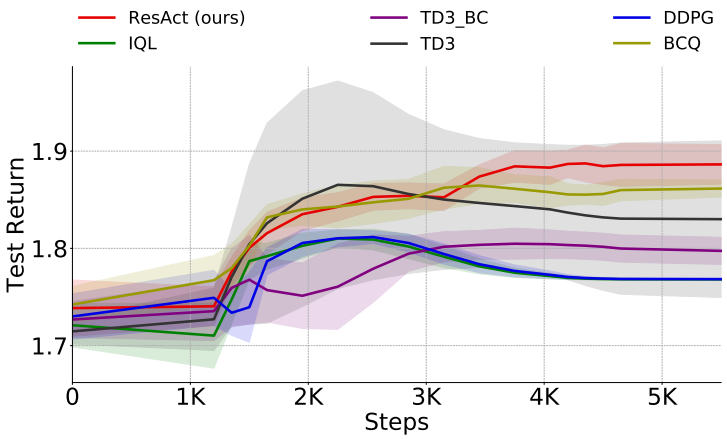


FIGURE 6.4: Learning curves of RL-based methods on *MovieLensL-1m*.

- **TD3** [120]: An off-policy reinforcement learning algorithm which applies clipped double-Q learning, delayed policy updates, and target policy smoothing.
- **TD3_BC** [126]: A reinforcement learning designed for offline training. It adds a behavior cloning (BC) term to the policy update of TD3.
- **BCQ** [116]: An off-policy algorithm which restricts the action space in order to force the agent towards behaving similarly to on-policy.
- **IQL** [127]: An offline reinforcement learning method which takes a state conditional upper expectile to estimate the value of the best actions in a state.
- **IL**: Imitation learning treats the training set as expert knowledge and learns a mapping between observations and actions under demonstrations of the expert.
- **IL_CVAE** [117]: Imitation learning method with the policy controlled by a conditional variational auto-encoder.

Our method emphasises on the learning and execution paradigm, and is therefore orthogonal to those approaches which focus on designing neural network architectures, e.g., GRU4Rec [26].

6.3.2 Overall Performance

MovieLensL-1m. We first evaluate our method on a benchmark dataset *MovieLensL-1m* which contains 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users. We sample the data of 5000 users as the training set, and use the data of the remaining users as the test set (with 50 users as the validation set). As in Table 6.2, our method, ResAct, outperforms all the baselines, indicating its effectiveness. We also provide the learning curve in Figure 6.4. It can be found that ResAct learns faster and is more stable than the baselines on *MovieLensL-1m*.

RecL-25m. We test the performance of ResAct on *RecL-25m* in three modes: i) Return Time mode, where the reward signal $r(\delta)$ is calculated by Eq. 6.16; ii) Session Length mode, where the reward signal $r(\eta)$ is calculated by Eq. 6.17; and iii) Both, where reward signal is generated by a convex combination of $r(\delta)$ and $r(\eta)$ with weights of 0.7 and 0.3 respectively. The weights are determined by real-world demands on the operational metrics. Among the 99,899 users, we randomly selected 80% of the users as the training set, of which 500 users were reserved for validation. The remaining 20% users constitute the test set. As shown in Table 6.3, our method significantly outperforms the baselines in all the settings. The classic reinforcement learning algorithms, e.g., DDPG and TD3, perform poorly in the tasks, which indicates that directly predicting an action is difficult. The decomposition of actions effectively facilitates the learning process. Another finding is that the offline reinforcement learning algorithms, e.g., IQL, also perform poorly, even though they are specifically designed to learn from logged data. By comparing with imitation learning, we find that the residual actor has successfully found a policy to improve an action, because behavior reconstruction cannot achieve good performance alone. To compare the learning process, we provide learning curves for those RL-based algorithms. Returns are calculated on the validation set with approximately 30,000 sessions. As in Figure 6.5, the performance of ResAct increases faster and is more stable than the other methods, suggesting that it is easier and more efficient to predict the residual than to predict an action directly.

TABLE 6.2: Performance comparison on MovieLensL-1m. The "±" indicates 95% confidence intervals.

	Return
DDPG	1.7429 ±0.0545
TD3	1.7363 ±0.0546
TD3_BC	1.7135 ±0.0541
BCQ	1.7898 ±0.0320
IQL	1.7360 ±0.0546
IL	1.7485 ±0.0310
IL_CVAE	1.7344 ±0.0316
ResAct (Ours)	**1.8123 ±0.0319**

TABLE 6.3: Performance comparison on RecL-25m in various tasks. The "±" indicates 95% confidence intervals.

	Return Time	Session Length	Both
DDPG	0.6375 ±0.0059	0.3290 ±0.0056	0.5908 ±0.0092
TD3	0.6756 ±0.0133	0.4015 ±0.0073	0.5498 ±0.0103
TD3_BC	0.6436 ±0.0059	0.3671 ±0.0037	0.5563 ±0.0050
BCQ	0.6837 ±0.0061	0.3836 ±0.0033	0.5915 ±0.0049
IQL	0.6296 ±0.0094	0.3430 ±0.0057	0.5579 ±0.0067
IL	0.6404 ±0.0058	0.3186 ±0.0032	0.5345 ±0.0048
IL_CVAE	0.6410 ±0.0058	0.3178 ±0.0031	0.5346 ±0.0047
ResAct (Ours)	**0.7980 ±0.0067**	**0.5433 ±0.0045**	**0.6675 ±0.0053**

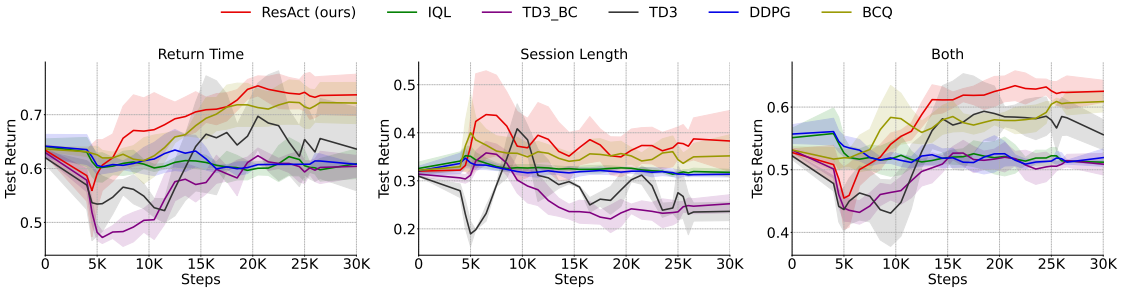


FIGURE 6.5: Learning curves of RL-based methods on *RecL-25m*, averaged over 5 runs.

6.3.3 Analyses and Ablations

How does ResAct work? To understand the working process of ResAct, we plot t-SNE [128] embedding of actions generated in the execution phase of ResAct. As in Figure 6.6, the reconstructed actions, denoted by the red dots, are located

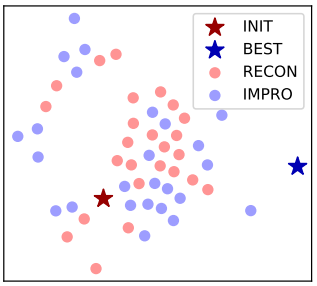


FIGURE 6.6: The t-SNE visualization of actions.

around the initial action (the red star), suggesting that ResAct successfully samples several online-behavior estimators. The blue dots are the t-SNE embedding of the improved actions, which are generated by imposing residuals on the reconstructed actions. The blue star denotes the executed action of ResAct. We can find that the blue dots are near the initial actions but cover a wider area then the red dots.

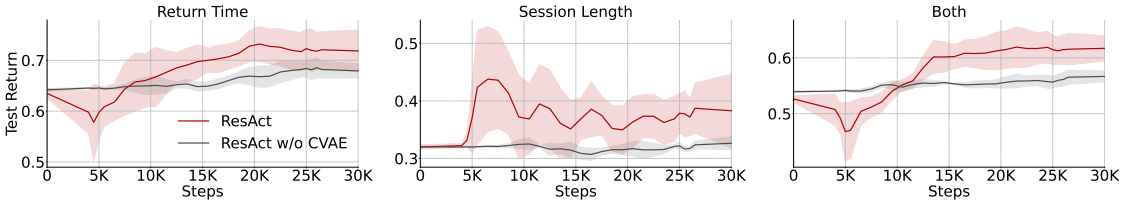


FIGURE 6.7: Learning curves for ResAct with CVAE and a deterministic reconstructor (w/o CVAE).

Effect of the CVAE. We design the CVAE for online-behavior reconstruction because of its ability to generate multiple estimators. To explore the effect of the CVAE and whether a deterministic action reconstructor can achieve similar performance, we disable the CVAE in ResAct and replace it with a feed-forward neural network. The feed-forward neural network is trained by using the loss in Equation 6.2. Since the feed-forward neural network is deterministic, ResAct does not need to perform the selection phase as there is only one candidate action. We provide the learning curves of ResAct with and without the CVAE in Figure 6.7. As we can find, there is a significant drop in improvement if we disable the CVAE. We deduce that this is because a deterministic behavior reconstructor can only generate one estimator, and if the prediction is inaccurate, performance will be severely harmed.

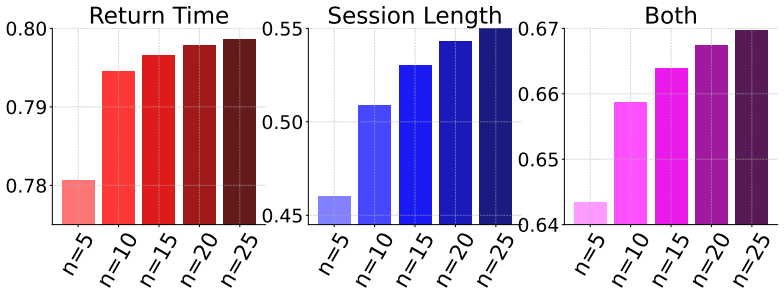


FIGURE 6.8: Ablations for the number of online-behavior estimators.

TABLE 6.4: Ablations for the information-theoretical regularizers. The "±" indicates 95% confidence intervals.

	Return Time	Session Length	Both
ResAct	**0.7980 ±0.0067**	**0.5433 ±0.0045**	**0.6675 ±0.0053**
w/o $L^{Exp}_{\theta_h,\theta_o}$	0.6610 ±0.0060	0.3895 ±0.0034	0.6074 ±0.0052
w/o $L^{Con}_{\theta_h}$	0.6944 ±0.0061	0.4542 ±0.0038	0.6041 ±0.0051
w/o $L^{Exp}_{\theta_h,\theta_o}$, $L^{Con}_{\theta_h}$	0.7368 ±0.0064	0.3854 ±0.0033	0.6348 ±0.0049

Number of Online-behavior Estimators. Knowing that generating only one action estimator might hurt performance, we want to further investigate how the number of estimators will affect the performance of ResAct. We first train a ResAct and then change the number of online-behavior estimators to 5, 10, 15, 20 and 25. As in Figure 6.8, consistent improvement in performance can be observed across all the three tasks as we increase the number of estimators. The fact suggests that generating more action candidates will benefit the performance, in line with our intuition. Because the sampling of action estimators is independent, the parallelization of ResAct is not difficult to implement and we can easily speed up the inference.

Information-theoretical Regularizers. To explore the effect of the designed regularizers, we disable $L^{Exp}_{\theta_h,\theta_o}$, $L^{Con}_{\theta_h}$ and both of them in ResAct, respectively. As shown in Table 6.4, the removal of any of the regularizers result in a significant drop in performance, suggesting that the regularizers facilitate the extraction of features and thus ease the learning process. An interesting finding is that removing both of the regularizers does not necessarily results in worse performance than removing only one. This suggests that we cannot simply expect either expressiveness or

conciseness of features, but rather the combination of both.

Sensitivity Analysis for the Reward Weights. When setting the reward weights in the Both mode, we use some usual empirical values by following the real-world requirements for operational metrics. The reward occurs only at the end of each session, which makes it representative for sequential recommendations. If we try other designs, only the value of the reward will change, not the frequency of learning signal. To justify that our algorithm is robust to different reward weights, we perform sensitivity analysis for the weights (return time: session length) of rewards in the Both mode. As shown in Table 6.5, our algorithm consistently outperforms the baselines under different reward weights.

TABLE 6.5: Sensitivity Analysis for the Reward Weights in the Both Mode.

	(0.7: 0.3)	(0.5: 0.5)	(0.3: 0.7)
DDPG	0.5908 ±0.0092	0.5040 ±0.0073	0.4172 ±0.0059
TD3	0.5498 ±0.0133	0.4941 ±0.0086	0.4385 ±0.0076
TD3_BC	0.5563 ±0.0050	0.4978 ±0.0043	0.4393 ±0.0038
BCQ	0.5915 ±0.0049	0.5261 ±0.0042	0.4605 ±0.0037
IQL	0.5579 ±0.0067	0.4812 ±0.0054	0.4046 ±0.0044
IL	0.5345 ±0.0048	0.4727 ±0.0041	0.4111 ±0.0036
IL_CVAE	0.5346 ±0.0047	0.4726 ±0.0041	0.4107 ±0.0036
ResAct (Ours)	**0.6675 ±0.0053**	**0.5948 ±0.0045**	**0.5220 ±0.0039**

TABLE 6.6: Performance comparison between ResAct and ResAct with uniformly augmented action candidates. The "±" indicates 95% confidence intervals.

	Return Time	Session length	Both
ResAct	0.7980 ±0.0067	0.5433 ±0.0045	0.6675 ±0.0053
ResAct + 20 candidates (uniform)	0.5501 ±0.0068	0.3489 ±0.0041	0.4839 ±0.0054

Quality of Online-behavior Estimators. Despite the selection phase, the quality of the online-behavior estimators, or the action candidates, still significantly affects the performance. On the one hand, the action candidate directly constitutes the final action. On the other hand, the sampled action candidates serve as inputs of the residual module and the selection module. It is certain that sampling an infinite number of action candidates will cover the best action which is sampled by

the CVAE. However, action candidates which are far from online-behavior policy may be incorrectly selected. The reason is that the residual module and the selection module are unlikely to encounter such out-of-distribution (OOD) actions and therefore cannot make accurate predictions. The distribution of action candidates should be as close as possible to the distribution of online services to ensure that the output of the residual and selection modules is reliable. We conduct experiments by uniformly sampling 20 action candidates and adding them to the action candidates reconstructed by the CVAE. As in Table 6.6, there is a significant decrease in performance even though we increase the number of action candidates.

6.4 Chapter Summary

In this chapter, we propose ResAct to reinforce long-term engagement in sequential recommendation. ResAct achieves effective local policy improvement with a residual actor. It works by first reconstructing behaviors of the online-serving policy, and then improving the reconstructed policy by imposing an action residual. By doing so, ResAct learns a policy which is close to, but better than, the deployed recommendation model. To facilitate the feature extraction, two information-theoretical regularizers are designed to make state representations both expressive and concise. We conduct extensive experiments on a benchmark dataset *MovieLensL-1m* and a real-world dataset *RecL-25m*. Experimental results demonstrate the superiority of ResAct over previous state-of-the-art algorithms in all the tasks. It is worthwhile to note that ResAct has potential limitations in aspect of training and inference speed, because it involves three phases which has to be processed sequentially. Techniques such as distributed machine learning might address this issue to some extend.

Chapter 7

Adaptive Policy Learning under Guidances of Human Preferences

In Chapter 6, we formulate sequential recommendation as Markov Decision Process and design a reward function to incentivize recommender systems behave properly. However, designing an appropriate reward function is very difficult especially in large-scale complex tasks like recommendation [46, 129, 130]. On one hand, the reward function should be aligned with our ultimate goal as much as possible. On the other hand, rewards should be sufficiently dense and instructive to provide step-by-step guidance to the agent. For immediate engagement, we can simply use metrics such as click-through rates to generate rewards [44, 115]. Whereas for long-term engagement, the problem becomes rather difficult because attributing contributions to long-term engagement to each step is really tough. If we only assign rewards when there is a significant change in long-term engagement, learning signals could be too sparse for the agent to learn a policy. Existing RL recommender systems typically define the reward function empirically [34, 99, 129] or use short-term signals as surrogates [46], which will severely violate the aforementioned requirements of consistency and instructivity.

To mitigate the problem, in this chapter[1], we propose a new training paradigm, recommender systems with human preferences (or **Pre**ference-based **Rec**ommender systems), which allows RL recommender systems to learn from human feedback/ preferences on users' historical behaviors rather than explicitly defined rewards. We demonstrate that RL from human preferences (or preference-based RL), a framework that has led to successful applications such as ChatGPT [132], is also applicable to recommender systems. Specifically, in PrefRec, there is a (virtual) teacher giving feedback about his/her preferences on pairs of users' behaviors. We use the feedback (stored in a preference buffer) to automatically train a reward model which generates learning signals for recommender systems. Such preferences are easy to obtain because no expert knowledge is required, and we can use technologies such as crowdsourcing to easily gather a large number of labeled data. Furthermore, to overcome the problem that the reward model may not work well for some unseen actions, we introduce a separate value function, trained by expectile regression, to assist the training of the critic in PrefRec[2]. Our main contributions are threefold:

- We propose a new framework, recommender systems with human preferences, to optimize long-term engagement. Our method can fully exploit the advantages of reinforcement learning in optimizing long-term goals, while avoiding the complex reward engineering in reinforcement learning.
- We design an effective optimization method for the proposed framwork, which uses an additional value function, expectile regression and reward model pretraining to improve the performance.
- We collect the first reinforcement learning from human feedback (RLHF) dataset for long-term engagement optimization problem in recommendation and propose three new tasks to evaluate the performance of recommender

[1]The work in this chapter has been published as **Wanqi Xue**, Qingpeng Cai, Zhenghai Xue, Shuo Sun, Shuchang Liu, Dong Zheng, Peng Jiang, Kun Gai, Bo An. PrefRec: Recommender systems with human preferences for reinforcing long-term user engagement [131]. *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (**KDD**), 2023.

[2]PrefRec adopts the framework of actor-critic in reinforcement learning.

systems. Experimental results demonstrate that PrefRec significantly outperforms previous state-of-the-art approaches on all the tasks.

7.1 Background

Long-term User Engagement in Recommendation. Long-term user engagement is an important metric in recommendation, and is typically being reflected as the stickiness of users to a product. In general, given a product, we expect users to spend more time on it and/or use it as frequently as possible. In this work, we assume that users interact with the recommender systems on a session basis: when a user accesses a product, such as an App, a session begins, and it ends when the user leaves. During each session, users can launch an arbitrary number of recommendation requests as they want. Such session-based recommender systems have been widely deployed in real-life applications such as short-form videos recommendation and news recommendation [25, 44, 108, 133]. We are particularly interested in increasing

i the number of recommended items that users consume during each visit;

ii the frequency that users visit the product.

Optimizing these two indicators is nontrivial because it is difficult to relate them to a single recommendation. For example, if a user increases its visiting frequency, we are not able to know exactly which recommendation leads to the increase. To this end, we propose to use reinforcement learning to take into account the potential impact on the future when making decisions.

Recommendation as a Markov Decision Process (MDP). Applying RL to recommender systems requires defining recommendation as a Markov Decision Process (MDP). Recommender systems can be described as an agent to interact with users, who act as the environment. Formally, we formulate recommendation as a Markov Decision Process (MDP) $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$:

- $\mathcal{S}$ is the continuous state space. $s \in \mathcal{S}$ indicates the state of a user which contains static information such as gender and age; and dynamic information, such as the rate of likes and retweets. A state is what the recommender system relies on to make decisions.

- $\mathcal{A}$ is the continuous action space, where $a \in \mathcal{A}$ is an action which has the same dimension as the representation of recommendation items. We determine the item to recommend by comparing the similarity of an action and item representations [99, 115].

- $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition function, where $p(s_{t+1}|s_t, a_t)$ defines the probability that the next state is s_{t+1} after recommending an item a_t at the current state s_t.

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. $r(s_t, a_t)$ determines how much the agent will be rewarded after recommending a_t at state s_t.

- γ is the discount factor. γ determines how much the agent cares about rewards in the distant future relative to those in the immediate future.

The recommendation policy $\pi(a|s) : \mathcal{S} \to \mathcal{A}$ is defined as a mapping from state to action. Given a policy $\pi(a|s)$, we define a state-action value function (Q-function) $Q^\pi(s, a)$ which generates the expected cumulative reward (return) of taking an action a at state s and thereafter following π:

$$Q^\pi(s_t, a_t) = \mathbb{E}_{(s_{t'}, a_{t'}) \sim \pi} \left[r(s_t, a_t) + \sum_{t'=t+1}^{\infty} \gamma^{(t'-t)} \cdot r(s_{t'}, a_{t'}) \right]. \tag{7.1}$$

We seek to optimize the policy $\pi(a|s)$ so that the return obtained by the recommendation agents is maximized:

$$\max_\pi \mathcal{J}(\pi) = \mathbb{E}_{s_t \sim d_t^\pi(\,\cdot\,), a_t \sim \pi(\,\cdot\,|s_t)} \left[Q^\pi(s_t, a_t) \right], \tag{7.2}$$

where $d_t^{\pi}(\cdot)$ denotes the state visitation frequency at step t under the policy π. By optimizing the above objective, the agent can achieve the largest cumulative return in the defined MDP.

Challenges of Designing the Reward Function. Despite that RL is a promising approach to optimize long-term user engagement, the application of RL heavily relies on a well-designed reward function. The reward function must be able to reflect the changes in long-term engagement at each time step. In the meantime, it should be able to provide instructive guidance to the agent for optimizing the policy. Practically, quantifying rewards properly is very challenging because it is really difficult to relate changes in long-term engagement to a single recommendation [46]. For example, when recommending a video to a user, we have no way of knowing how many videos the user will continue to watch on the platform before exiting the current session, and obviously, how this amount will be affected by the recommendation is even harder to know. For this reason, it is challenging to give a reward regarding the impact of recommended videos on the average video consumption of users. Similarly, recommending a video cannot be used to predict when the user will revisit the platform after exiting the current session. Designing rewards related to the visiting frequency of users is also difficult. As a compromise solution, one could only assign rewards at the beginning or the end of a session. However, this kind of reinforcement signals will be too sparse for the recommender systems to learn a reasonable policy, especially when the session length is large [99]. Existing methods either design the reward function empirically [34, 99, 129] or use immediate engagement signals as surrogates [46], which will cause deviation between the optimization objective and the real long-term engagement. For this reason, it is urgent to propose a framework to address the difficulties in reward designing when using RL to optimize long-term user engagement.

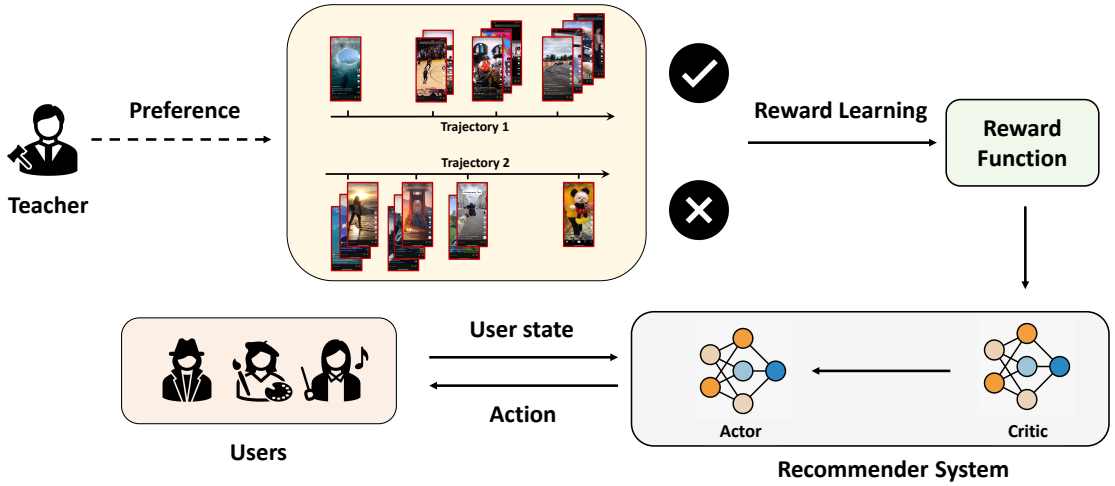


FIGURE 7.1: The framework of recommender systems with human preferences. A teacher provides feedback about his/her preferences between users' behavioral trajectories. Trajectory 1 demonstrates a trend from low-active to high-active and trajectory 2 shows an opposite tendency. Therefore, the teacher will prefer trajectory 1 to trajectory 2. With feedback of preferences, we can automatically train a reward function in an end-to-end manner. The preference-based recommender systems is then optimized by using rewards predicted by the learned reward function.

7.2 Adaptive Recommendation Policy with Human Preferences

To resolve the difficulties in designing the reward function, we propose a novel paradigm, **Pef**erence-based **Rec**ommender systems (PrefRec), which allows an RL recommender system to learn from preferences on users' historical behaviors rather than explicitly defined rewards. In this way, we can overcome the problems in designing the reward function when optimizing long-term user engagement. In this section, we first introduce how to utilize preferences to generate reinforcement signals for learning a recommender system. Then, we discuss how to optimize the performance of PrefRec by using expectile regression to better estimate the value function. Next, we propose to pre-train the reward function to stabilize the learning process. Last, we summarize the algorithm.

7.2.1 Reinforcing from Preferences

While the reward for a recommendation request is hard to obtain, preferences between two trajectories of users' behaviors are easy to determine. For example, if one trajectory shows a transition from low-active to high-active and the other shows an opposite trend or an insignificant change, we can easily indicate the preference between them. Labeling preferences does not require any expert knowledge, so we can easily use techniques such as crowdsourcing to obtain a large amount of feedback on preferences. In PrefRec, we assume that there is a human teacher providing preferences between user's behaviors and the recommender system uses this feedback to perform the task. There are mainly two advantages in using preferences: i) labeling the preference between a pair of trajectories is quite simple compared to designing rewards for every step; and ii) the recommender system is incentivized to learn the preferred behavior directly because reinforcement signals come from preferences.

We provide the framework of recommender systems with human preferences in Figure 7.1. As shown, there is a teacher providing preferences between a pair of users' behavioral trajectories[3]. The teacher could be humans or or even a program with labeling criteria. For trajectory 1, the user increases its visiting frequency gradually and consumes more and more items in each session, which indicates that it is satisfied with the current recommendation policy. On the contrary, trajectory 2 shows the user is becoming less and less active, suggesting that the current recommender system should be improved to better serve the user. The teacher will obviously prefer trajectory 1 to trajectory 2. After generating such preference data, we use them to automatically train a reward function $\hat{r}(s, a; \psi)$, parameterized by ψ, in an end-to-end manner. The preference-based recommender systems uses the predicted reward $\hat{r}(s, a; \psi)$ rather than hand-crafted rewards to update its policy.

Formally, a trajectory σ is a sequence of observations and actions $\{(s_1, a_1), \ldots, (s_T, a_T)\}$. Given a pair of trajectories (σ^0, σ^1), a teacher provides a feedback indicating which

[3]For a pair of trajectories, they may come from different users or from different periods of the same user.

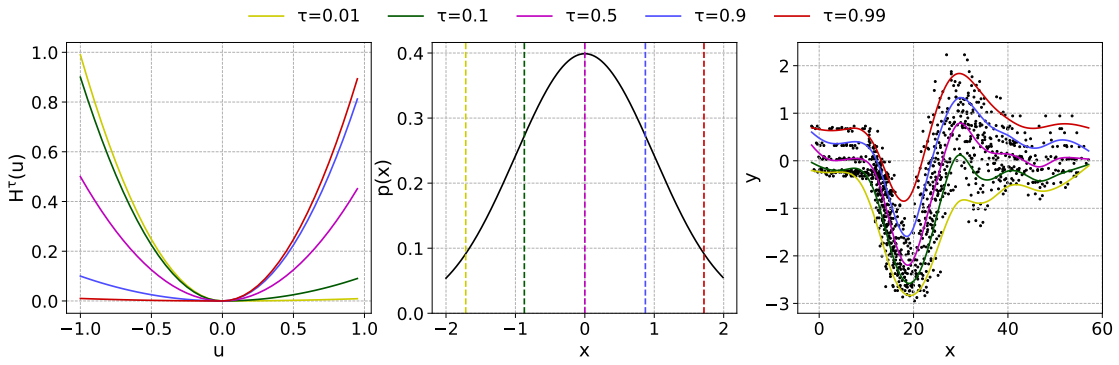


FIGURE 7.2: Left: Illustration of the expectile loss function $H^\tau(u)$. $\tau = 0.5$ corresponds to the MSE loss, and $\tau > 0.5$ upweights positives differences u. Center: Expectiles of a Gaussian distribution. Right: An example of expectile regression on a two-dimensional distribution.

trajectory is preferred, i.e., $y = (0, 1)$, $(1, 0)$ or $(0.5, 0.5)$, where $(0, 1)$ indicates trajectory σ^1 is preferred to trajectory σ^0, i.e., $\sigma^1 \succ \sigma^0$; $(1, 0)$ indicates $\sigma^0 \succ \sigma^1$; and $(0.5, 0.5)$ implies an equally preferable case. Each feedback is triple (σ^0, σ^1, y) which is stored in a preference buffer $\mathcal{D}_p = \{((\sigma^0, \sigma^1, y))_i\}_{i=1}^N$. We use a deep neural network with parameters ψ to predict the reward $\hat{r}(s, a; \psi)$ at a specific step (s, a). By following the Bradley-Terry model [134, 135], we assume that the teacher's probability of preferring a trajectory depends exponentially on the accumulated sum of the reward over the trajectory. Then the probability that trajectory σ^1 is preferred to trajectory σ^0 can be written as a function of $\hat{r}(s, a; \psi)$:

$$P_\psi \left[\sigma^1 \succ \sigma^0\right] = \frac{\exp\left(\sum_t \hat{r}\left(\mathbf{s}_t^1, \mathbf{a}_t^1; \psi\right)\right)}{\sum_{i\in\{0,1\}} \exp\left(\sum_t \hat{r}\left(\mathbf{s}_t^i, \mathbf{a}_t^i; \psi\right)\right)}. \tag{7.3}$$

Since the preference buffer $\mathcal{D}_p = \{((\sigma^0, \sigma^1, y))_i\}_{i=1}^N$ contains true labels of preferences, we can train the reward model $\hat{r}(s, a; \psi)$ through supervised learning, updating it by minimizing the cross-entropy loss:

$$\mathcal{L}_\psi = -\mathop{\mathbb{E}}_{(\sigma^0,\sigma^1,y)\sim\mathcal{D}_p} \left[y(0) \log P_\psi \left[\sigma^0 \succ \sigma^1\right] + y(1) \log P_\psi \left[\sigma^1 \succ \sigma^0\right]\right], \tag{7.4}$$

where $y(0)$ and $y(1)$ are the first and second elements of y, respectively. After learning the reward model $\hat{r}(s, a; \psi)$, we can use it to generate learning signals for training the recommendation policy.

7.2.2 Optimizing the Recommendation Policy

When learning the recommendation policy, there is a replay buffer $\mathcal{D}_r$ storing training data. Unlike conventional RL recommender systems, we store (s, a, s') rather than (s, a, r, s') in the buffer $\mathcal{D}_r$ because we cannot obtain explicit rewards from the environment[4]. We utilize the learned reward model $\hat{r}(s, a; \psi)$ to label the reward for a tuple (s, a). After labelling, an intuitive approach is to learn the Q-function by minimizing the following temporal difference (TD) error:

$$\mathcal{L}_\theta^Q = \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{D}_r} \left[(\hat{r}(s, a; \psi) + \gamma Q(s', \pi(\cdot\,|s'); \hat{\theta}) - Q(s, a; \theta))^2 \right], \tag{7.5}$$

where $\mathcal{D}_r$ is the replay buffer, $Q(s, a; \theta)$ is a parameterized Q-function, $Q(s, a; \hat{\theta})$ is a target network (e.g., with soft updating of parameters defined via Polyak averaging), and $\pi(\cdot\,|s)$ is the recommendation policy. However, in practice, we find that this method does not perform well. It may be because the recommendation policy $\pi(\cdot\,|s)$ will choose significantly different actions from the stored data, making the reward function and the Q-function unable to predict the corresponding values. To resolve this, we introduce a separate value function (V-function) that predicts how good or bad a state is. By doing so, we can eliminate the uncertainty that comes with the recommended policy. Instead of minimizing the TD error in Eq. 7.5, we turn to minimizing the following loss to learn the Q-function:

$$\mathcal{L}_\theta^Q = \mathop{\mathbb{E}}_{(s,a,s')\sim\mathcal{D}_r} \left[(\hat{r}(s, a; \psi) + \gamma V(s'; \eta) - Q(s, a; \theta))^2 \right], \tag{7.6}$$

where $V(s'; \eta)$ is the V-function with parameters η. Given the replay buffer $\mathcal{D}_r$, the relationship between Q-function and V-function is

$$V(s) = \mathbb{E}_a[Q(s, a)], (s, a) \in \mathcal{D}_r. \tag{7.7}$$

[4]With a slight abuse of notation, we use s' to denote the next state.

Conventionally, we can optimize the parameters of the V-function by minimizing the following Mean Squared Error (MSE) loss:

$$\mathcal{L}_{\eta}^{V} = \mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}_r} \left[(Q(s,a;\hat{\theta}) - V(s;\eta))^2 \right]. \tag{7.8}$$

However, such V-function corresponds to the behavior policy which collects the replay buffer $\mathcal{D}_r$. We want to achieve improvement upon the behavior policy. Inspired by expectile regression [127, 136], we let the V-function to regress the τ expectile ($\tau \geq 0.5$) of $Q(s,a)$ rather than the mean statistics as in Eq. 7.7. Then the loss for V-function becomes:

$$\mathcal{L}_{\eta}^{V} = \mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}_r} \left[H^{\tau}(Q(s,a;\hat{\theta}) - V(s;\eta)) \right], \tag{7.9}$$

where $H^{\tau}(u) = |\tau - \mathbb{I}(u<0)|u^2$, $\mathbb{I}(\cdot)$ is the indicator function. Specially, if $\tau = 0.5$, Eq. 7.8 and Eq. 7.9 are identical. For $\tau > 0.5$, this asymmetric loss (Eq. 7.9) down-weights the contributions of $Q(s,a;\hat{\theta})$ less than $V(s;\eta)$ while giving more weights to larger values (as in Figure 7.2, left). Figure 7.2 (right) illustrates the expectile regression on a two-dimensional distribution: increasing τ leads to more data points falling below the regression curve. Back to the learning of the V-function, the purpose is to let $V(s;\eta)$ to regress an above-average value. The Q-function is jointly trained with the V-function, while it also serves as the critic to guide the update of the recommendation policy, i.e., the actor. The recommendation policy is optimized by minimizing the loss:

$$\mathcal{L}_{\mu}^{P} = -\mathop{\mathbb{E}}_{(s,a)\sim\mathcal{D}_r} \left[Q(s,\pi(\cdot\,|s;\mu);\theta) \right], \tag{7.10}$$

where $\pi(\cdot, s;\mu)$ is the recommendation policy with parameters μ. We follow the standard paradigm of actor-critic in reinforcement learning here to optimize the recommendation policy.

Algorithm 7: Recommender Systems with Human Preferences

Input: Preference buffer $\mathcal{D}_p = \{(\sigma^0, \sigma^1, y)_i\}_{i=1}^N$, replay buffer $\mathcal{D}_r = \{(s, a, s')_i\}_{i=1}^M$, pre-train episodes K

Initialize the reward model $r(s, a; \psi)$, the Q-function $Q(s, a; \theta)$, the V-function $V(s; \eta)$, the recommendation policy $\pi(\cdot\,|s; \mu)$
Set soft-update rate λ and initialize the target Q-function $Q(s, a; \hat{\theta})$
for *pre-train episodes* $k = 1$ *to* K **do**
    Sample a mini-batch of preferences (σ^0, σ^1, y) from $\mathcal{D}_p$
    Update the reward model $r(s, a; \psi)$: $\psi \leftarrow \psi - \nabla_\psi \mathcal{L}_\psi$ (Eq. 7.4)
end
while *not end training* **do**
    Sample a mini-batch of transitions (s, a, s') for $\mathcal{D}_r$
    Label the sampled transitions by the reward model to obtain $(s, a, \hat{r}, s')$
    Update the V-function $V(s; \eta)$: $\eta \leftarrow \eta - \nabla_\eta \mathcal{L}_\eta^V$ (Eq. 7.9)
    Update the Q-function $Q(s, a; \theta)$: $\theta \leftarrow \theta - \nabla_\theta \mathcal{L}_\theta^Q$ (Eq. 7.6)
    Update the recommendation policy $\pi(\cdot\,|s; \mu)$: $\mu \leftarrow \mu - \nabla_\mu \mathcal{L}_\mu^P$ (Eq. 7.10)
    Soft-update the target Q-function $Q(s, a; \hat{\theta})$: $\hat{\theta} \leftarrow \lambda\theta + (1 - \lambda)\hat{\theta}$
    if *fine-tune reward model* **then**
        Sample a mini-batch of preferences (σ^0, σ^1, y) from $\mathcal{D}_p$
        Update the reward model $r(s, a; \psi)$: $\psi \leftarrow \psi - \nabla_\psi \mathcal{L}_\psi$ (Eq. 7.4)
    end
end

7.2.3 Pre-training the Reward Function

The reward function is used to automatically generate reinforcement signals to train the recommendation policy. However, a reward model that is not well-trained will cause the collapse of the recommendation policy. Thus, to stabilize the training, we propose to pre-train the reward function before starting the updating of the recommendation policy. Specifically, we first prepare a preference buffer that stores a pool of preference feedback between interaction histories. Then we initialize a deep neural network with the structure of multi-layer perceptron to learn from the preference buffer. The neural network is updated by minimizing the loss in Eq. 7.4. After training for several episodes, we start the training of the recommendation policy. In this phase, the reward model is updated simultaneously with the recommender systems. By doing so, the reward model can handle potential shift in human preferences and can therefore generate more accurate learning signals.

7.2.4 Overall Algorithm

We provide the overall algorithm of PrefRec in Algorithm 7. A preference buffer $\mathcal{D}_p$ and a replay buffer $\mathcal{D}_r$ are required for training PrefRec. From lines 1 to 2, we do the initialization for the deep neural networks and set hyper-parameters such as soft-update rate. From lines 3 to 5, we pre-train the reward model to confirm that it is able to provide reasonable learning signals when updating the recommendation policy. Lines 7 to 18 describe the training process of the recommendation policy. We first sample a mini-batch of transitions from the replay buffer $\mathcal{D}_r$. Then we label the samples data with the reward model. Following that, we update the parameters of the V-function, the Q-function and the recommendation policy accordingly. Finally, if fine-tuning the reward model is enabled, we will train the reward model to keep it up-to-date.

7.2.5 Discussions

PrefRec differs from both inverse reinforcement learning (IRL) [137] and model-based RL [138]. The key difference between PrefRec and IRL is that IRL requires costly expert demonstrations while PrefRec only requires simple label-like feedback. The reward function in PrefRec is learned by aligning with human preferences, whereas in IRL it is inferred from expert demonstrations. On the other hand, model-based RL relies on environmental rewards and aims to simplify learning by approximating the reward and transition functions. In contrast, PrefRec does not require environmental rewards and is designed to be learned without them.

7.3 Experimental Evaluation

We conduct extensive experiments on various of long-term user engagement optimization tasks. In particular, we want to answer the following four research questions (RQs):

- **RQ1**: Can the framework of PrefRec lead to improvement in long-term user engagement?
- **RQ2**: Whether PrefRec is able to outperform existing state-of-the-art methods?
- **RQ3**: Whether the learned reward signals are able to reflect the true underlying rewards?
- **RQ4**: How do the components in PrefRec contribute to the performance?

7.3.1 Preparing the dataset

Since PrefRec is a new framework in recommendation, there is no available dataset for evaluation. To prepare the dataset, we track complete interaction histories of around $100,000$ users from a leading short-form videos platform for months, during which over 25 million recommendation services are provided. For each user, we record its interaction history in a session-request form: the interaction history consists of several sessions with each session containing a number of recommendation requests (see Sec. 7.1). The recommender system provides service at each recommendation request where we record the state of a user and the action of the recommender system. Each state is a 245-dimensional vector containing the user's state information, such as gender, age and historical like rate. We applied scaling, normalization and clipping to each dimension of states in order to stabilize the input of models. An action is a 8-dimensional vector which is the representation of the recommended item at a request. We record the timestamp each time a user starts and exits a session. With the timestamp, we can calculate the duration that a user revisits the platform and thus can infer the visiting frequency. We measure changes in long-term user engagement at the end of each session for each user. Specifically, we first calculate the average session depth δ^u_{avg} and the average revisiting time ϵ^u_{avg} for each user u during the time span. For each session, δ^u_i denotes the number of requests in the i-th session of the user u. If δ^u_i is larger than the

average session depth δ^u_{avg}, we consider that there is an improvement in session depth. Similarly, we can calculate the revisiting time ϵ^u_i for session i and if it is less than the average revisiting time ϵ^u_{avg}, we consider that there is an improvement in visiting frequency. We quantify the changes in session depth and visiting frequency into six levels (level 0 to level 5; the higher the level, the more positive is the change) where the levels are determined by the following equations, respectively:

$$\begin{aligned} \mathbf{L}^u_i(depth) &= \left(\lfloor \frac{\delta^u_i}{\delta^u_{avg}} \rfloor \right) .clip(0,5) \\ \mathbf{L}^u_i(frequency) &= \left(\lfloor \frac{\epsilon^u_{avg}}{\epsilon^u_i} \rfloor \right) .clip(0,5) \end{aligned} \tag{7.11}$$

We provide the proportion of the calculated levels of all sessions in Figure 7.3. For session depth, most of sessions stay in level 0 and only very few of them are located in level 4 and 5. The visiting frequency demonstrates a more even distribution.

After processing the data, we can prepare the two buffers, i.e., the preference buffer $\mathcal{D}_p$ and the replay buffer $\mathcal{D}_r$, which are used in PrefRec. For the preference buffer $\mathcal{D}_p$, we uniformly sample $20,000$ pairs of users who have launched for more than 200 recommendation requests in the platform. We set the length of trajectory σ as 100 and randomly sample a segment of trajectories with this length from the interaction histories of the selected users. To generate the preferences, we write a scripted teacher who provides its feedback by comparing cumulative levels of changes on the trajectory segments. A trajectory with higher cumulative levels of changes (as defined in Eq. 7.11) is preferred by the scripted teacher. For the replay buffer $\mathcal{D}_r$, we randomly sample 80% of users as the training set and split their interaction histories into transitions (s, a, s') to fill up the replay buffer $\mathcal{D}_r$.

7.3.2 Baselines and Evaluation Metric

We compare PrefRec with a variety of baselines, including reinforcement learning methods for off-policy continuous control (DDPG, TD3, SAC), offline reinforcement learning algorithms (TD3_BC, BCQ, IQL), and imitation learning:

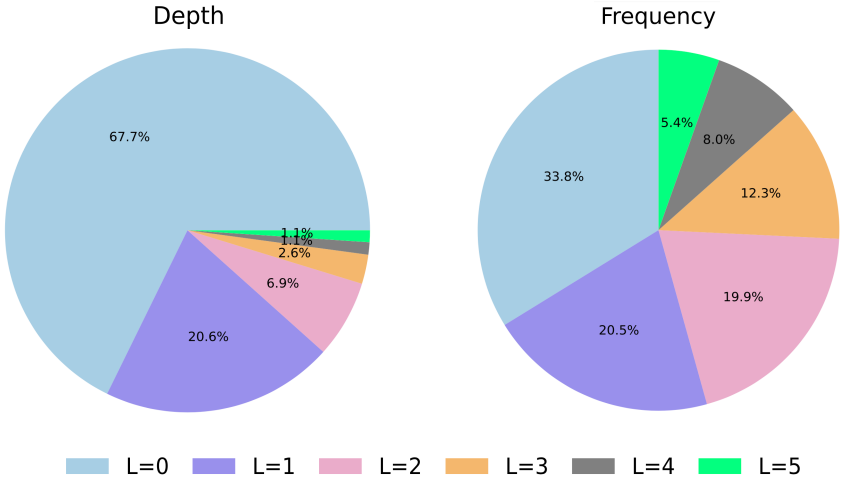


FIGURE 7.3: The proportion of the levels in session depth and visiting frequency.

- **DDPG** [119]: An off-policy reinforcement learning algorithm which concurrently learns a Q-function and a deterministic policy. The update of the policy is guided by the Q-function.

- **TD3** [120]: A reinforcement learning algorithm which is designed upon DDPG. It applies techniques such as clipped double-Q learning, delayed policy updates, and target policy smoothing.

- **SAC** [139]: An off-policy reinforcement learning algorithm trained to maximize a trade-off between expected return and entropy, a measure of randomness in the policy. It also incorporates tricks such as the clipped double-Q learning.

- **TD3_BC** [126]: An offline reinforcement learning algorithm designed based on TD3. It applies a behavior cloning (BC) term to regularize the updating of the policy.

- **BCQ** [116]: An offline algorithm which restricts the action space in order to force the agent towards behaving similarly to the behavior policy.

- **IQL** [127]: An offline reinforcement learning method which uses expectile regression to estimate the value of the best action in a state.

- **IL**: Imitation learning treats the behaviors in the replay buffer as expert knowledge and learns a mapping from observations to actions by using expert knowledge as supervisory signals.

Since our work focuses on addressing complex reward engineering when reinforcing long-term engagement and how to convey human intentions to RL-based recommender systems, we mainly make comparisons with classical RL algorithms. Works like FeedRec [34] emphasizes on designing DNN architecture and is orthogonal to PrefRec which focuses on the policy optimization process.

TABLE 7.1: Hyper-parameters of PrefRec.

Hyper-parameter	**Value**
Optimizer	Adam [97]
Actor Learning Rate	5×10^{-6}
Critic Learning Rate	5×10^{-5}
State Dimensions	245
Action Dimensions	8
Transitions Batch Size	4096
Preferences Batch Size	256
Normalized Observations	Ture
Gradient Clipping	False
Fine-Tuning	True
Discount Factor	0.9
Expectile Rate	0.7
Soft-update Rate	0.999
Segment Length	100
Preference Buffer Size	2×10^4
Replay Buffer Size	3×10^6
Number of Pre-train Epoch	3
Number of Train Epoch	5

Among the $10,0000$ users, we sample 80% of them as the training set and the remaining 20% users constitute the test set. For the test users, we store their complete interaction histories separately, in the session-request format. We adopt Normalised Capped Importance Sampling (NCIS) [123], a widely used standard offline evaluation method [124, 125], to evaluate the performance. Formally, the

score of a policy π is calculated by

$$\tilde{J}^{NCIS}(\pi) = \frac{1}{|\mathcal{U}|} \sum_{u \in \mathcal{U}} \left[\frac{\sum_{i=0}^{|\mathcal{T}_u|} \tilde{\rho}_i(\pi, \mathcal{T}_u) \mathbf{L}_i^u}{\sum_{i=0}^{|\mathcal{T}_u|} \tilde{\rho}_i(\pi, \mathcal{T}_u)} \right], \tag{7.12}$$

where $\mathcal{U}$ is the set of test users, $\mathcal{T}_u$ is the set of sessions of the user u, $\tilde{\rho}_i(\pi, \mathcal{T}_u)$ is the probability that the policy π follows the request trajectory of the i-th session in $\mathcal{T}_u$, and $\mathbf{L}_i^u$ is the level of change for the i-the session (as defined in Eq. 7.11). Intuitively, $\tilde{J}^{NCIS}(\pi)$ awards a policy with a high score if the policy has large probability to follow a good trajectory.

7.3.3 Implementation Details

To ensure fairness in comparison across all methods and experiments, a consistent network architecture is utilized. This architecture consists of a 3-layer Multi-Layer Perceptron (MLP) with 256 neurons in each hidden layer. The hyper-parameters for the PrefRec method are listed in Table 7.1. All methods were implemented using the PyTorch framework.

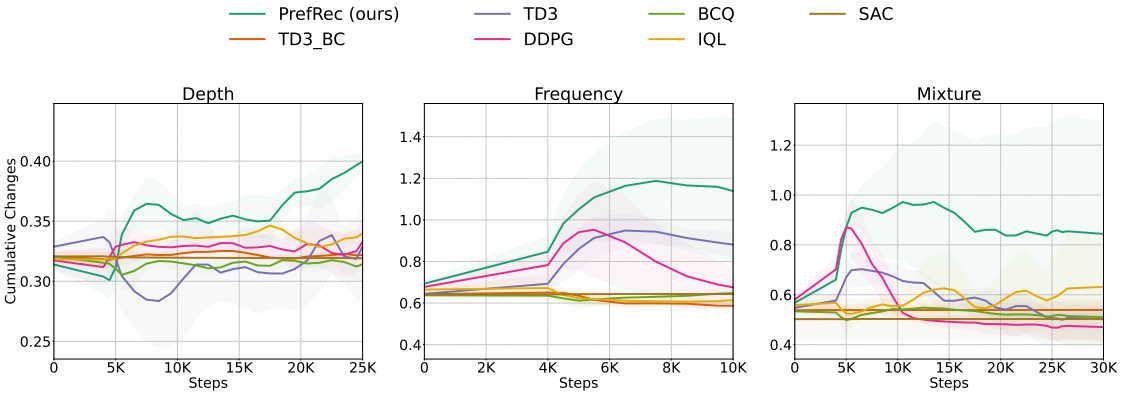


FIGURE 7.4: Learning curves of RL recommender systems under the framework of PrefRec, averaged over 5 runs.

TABLE 7.2: Overall performance comparisons on various long-term user engagement optimization tasks. The "±" indicates 95% confidence intervals.

	Depth	Frequency	Mixture
DDPG [119]	0.3211 ±0.0060	1.1408 ±0.0163	1.0642 ±0.0119
TD3 [120]	0.3495 ±0.0038	0.9714 ±0.0141	0.6423 ±0.0097
SAC [139]	0.3190 ±0.0031	0.6416 ±0.0058	0.5348 ±0.0047
TD3_BC [126]	0.3246 ±0.0032	0.6781 ±0.0060	0.5326 ±0.0047
BCQ [116]	0.3142 ±0.0033	0.6580 ±0.0060	0.5528 ±0.0052
IQL [127]	0.3311 ±0.0054	1.0354 ±0.0141	0.8230 ±0.0099
IL	0.3202 ±0.0031	0.6406 ±0.0058	0.5348 ±0.0047
PrefRec (ours)	**0.4229 ±0.0077**	**1.7706 ±0.0206**	**1.3788 ±0.0133**
% Improv.	21.00%	55.20%	29.56%

7.3.4 Overall Results

We conduct experiments to verify if the framework of PrefRec can improve long-term user engagement in terms of i) session depth; ii) visiting frequency; and iii) a mixture of the both. We randomly sample 500 users from the test set and use them to plot the learning curves of our method and the baselines. As in Fig. 7.4, PrefRec achieves a significant and consistent increase in cumulative long-term engagement changes in all the tasks, although it does not receive any explicit reinforcement signal. Similar phenomenon can also be observed in some of those generic reinforcement learning algorithms, such as DDPG. They demonstrate growth in specific tasks, though not that stable. The learning curves indicate that the learned reward function is able to provide reasonable reinforcement signals. and the framework of PrefRec provides an effective training paradigm to achieve reward-free recommendation policy learning. Next, we save the models with the best performance in training and test their performance on the whole test set. As in Table 7.2, those generic reinforcement learning algorithms perform poorly without explicit rewards. PrefRec is able to outperform all the baselines by a wide margin in all the three tasks, showing the effectiveness of the proposed optimization methods.

Despite utilizing only a single dataset, optimizing session depth and visiting frequency are distinct challenges. The results of the experiments demonstrate the

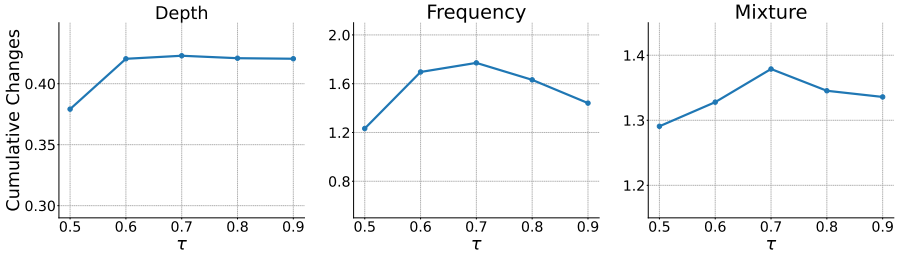


FIGURE 7.5: Ablations on the expectile regression factor τ.

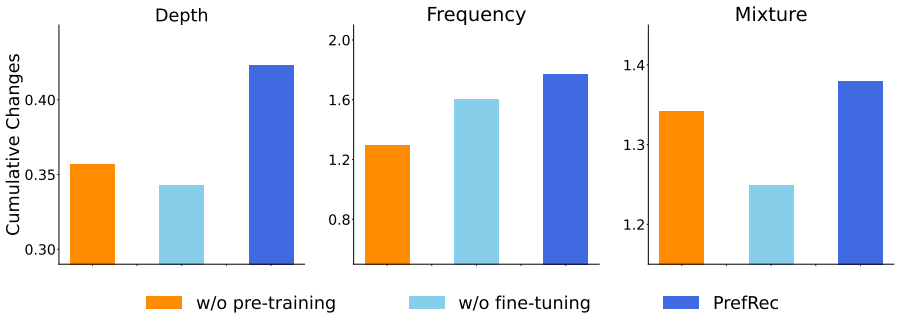


FIGURE 7.6: Ablations on reward function pre-training and reward function fine-tuning.

generalization ability of PrefRec as it consistently delivers improvements across both tasks. The optimization of session depth and visiting frequency are not necessarily interdependent; the algorithm that produces a deep session may not result in high visiting frequency, and similarly, high visiting frequency does not guarantee a deep session (as seen in Fig. 7.4).

7.3.5 Training Process of the Reward Function

To ensure the validity of the learning signals generated by the reward function, it must accurately predict the preferences that are utilized in the training phase. To evaluate the performance of the reward function, we have plotted its prediction accuracy in Fig. 7.7. The results show a noticeable improvement in accuracy as the training process advances. Additionally, we also plot the learning loss of the reward function in Fig. 7.8. The results indicate a substantial decrease in the loss, reducing it to a much lower level compared to its initial value. The improvement in prediction accuracy and the reduction in loss demonstrate that the reward function

is becoming increasingly effective at accurately predicting the preferences used in the training phase. This is crucial for ensuring that the learning signals generated by the reward function are reliable and accurate, enabling the model to learn from the preferences in an effective manner.

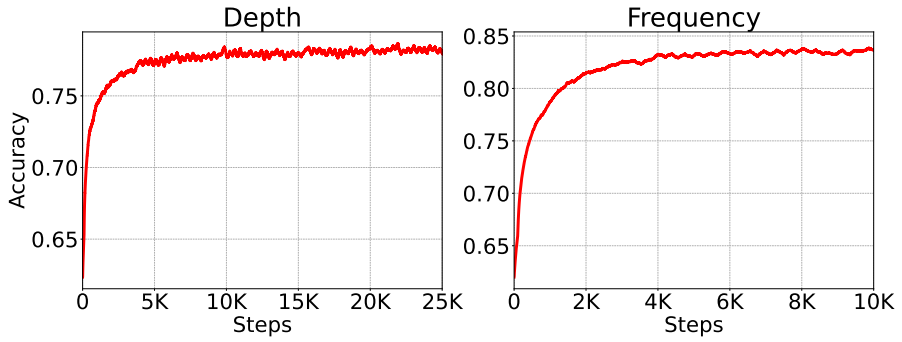


FIGURE 7.7: Prediction accuracy of the reward function.

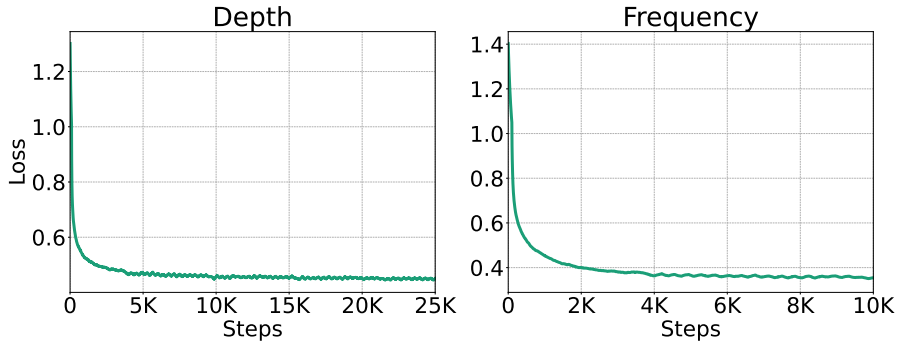


FIGURE 7.8: Learning loss of the reward function.

7.3.6 Ablations

To understand how the components in PrefRec affect the performance, we perform ablation studies on the expectile regression factor and the pre-training/fine-tuning of the reward function. As can be found in Table 7.5, when $\tau = 0.5$ where the expectile regression becomes identical to the regression to mean, there is an obvious drop in performance. The phenomenon indicates that the expectile regression with a proper τ contributes to the improvement in performance. Moreover, when comparing with the results of DDPG in Table 7.2, we can find that introducing a

separate V-function also benefits the performance since DDPG can be considered an algorithm whose $\tau = 0.5$ and without a V-function. Next, we study whether the pre-training and fine-tuning of the reward function is useful. As in Table 7.6, if we directly update the recommendation policy without pre-training the reward function, the performance decreases. Similarly, training the recommendation policy with a frozen reward function also degrades the performance. The results suggest that the pre-training and fine-tuning of the reward function are important to the performance.

7.4 Chapter Summary

In this chapter, we propose PrefRec, a novel paradigm of recommender systems, to improve long-term user engagement. PrefRec allows RL recommender systems to learn from preferences between users' historical behaviors rather than explicitly defined rewards. In this way, we can fully exploit the advantages of RL in optimizing long-term goals, while avoiding complex reward engineering. PrefRec uses the preferences to automatically learn a reward function. Then the reward function is applied to generate reinforcement signals for training the recommendation policy. We design an effective optimization method for PrefRec, which utilizes an additional value function, expectile regress and reward function pre-training to enhance the performance. Experiments demonstrate that PrefRec significantly and consistently outperforms the current state-of-the-art on various long-term user engagement optimization tasks.

Chapter 8

Conclusion and Future Directions

8.1 Conclusion

In this thesis, we systematically study the problem of decision-making in adversarial and uncertain environments. We propose several reinforcement learning algorithms to learn policies which are robust to adversaries and can quickly adapt to newly collected data.

We begin by studying how to learn robust policies in large-scale NSGs. In Chapter 3, we propose a new approach called NSG-NFSP, which aims to find Nash equilibria in extensive-form NSGs on a large scale. This algorithm utilizes deep neural networks to learn mappings from state-action pairs to values, which can represent either Q-values or probabilities. NSG-NFSP improves performance by equipping the NFSP attacker with high-level actions and effectively learning graph node embeddings. It surpasses state-of-the-art algorithms in terms of both scalability and solution quality. Moving on to Chapter 4, we introduce NSGZero as a data-efficient learning method for learning a non-exploitable policy in NSGs. We design three DNNs: the dynamics network, the value network, and the prior network, to facilitate efficient Monte Carlo tree search in NSGs. To enhance scalability,

we incorporate decentralized control into neural MCTS, enabling NSGZero to handle NSGs with a large number of resources. We also propose a learning paradigm for effective joint training in NSGZero to optimize the DNN parameters. Extensive experiments are conducted on various NSGs with diverse graph structures and scales. Empirical results demonstrate that NSGZero outperforms state-of-the-art algorithms in terms of data efficiency, achieving superior performance even with limited training experiences. The introduced two algorithms are potential to lead real-world application. For example, they can be used to guide the police department to interdict escaping criminals in urban city. And also, they might help the government to adjust the deployment of security resources in an area.

Second, in Chapter 5, we investigate the problem of robust communication in MACRL. This problem is important, but it has been largely neglected in the past. We first provide a formal definition of adversarial communication. Then, we propose an effective method for modeling message attacks in MACRL. Next, we design a two-stage message filter to defend against message attacks. Finally, we formulate the adversarial communication problem as a two-player zero-sum game and propose $\mathfrak{R}$-MACRL to improve robustness. Experiments on a wide range of algorithms and tasks show that many state-of-the-art MACRL methods are vulnerable to message attacks, while our algorithm can consistently recover multi-agent cooperation and improve the robustness of MACRL algorithms under message attacks.

Finally, we examine the problem of how to adapt recommendation policy to newly collected data for optimizing long-term user engagement in sequential recommendation. In Chapter 6, we introduce ResAct, a new algorithm for improving the performance of recommender systems. ResAct works by first reconstructing the behavior of the online-serving policy, and then improving the reconstructed policy by adding a residual to the actions. This results in a policy that is similar to the original policy, but which performs better. To make the state representations more expressive and concise, we design two information-theoretical regularizers. We evaluate ResAct on two datasets, *MovieLensL-1m* and *RecL-25m*, and show that it outperforms previous state-of-the-art algorithms on all tasks. In Chapter

7, we propose PrefRec which allows reinforcement learning recommender systems to learn from preferences between users' historical behaviors, rather than explicitly defined rewards. This allows us to take advantage of the strengths of RL, such as its ability to optimize long-term goals, while avoiding the complexity of reward engineering. PrefRec automatically learns a reward function from preferences, and then uses that reward function to generate reinforcement signals for training the recommendation policy. We design an effective optimization method for PrefRec that uses an additional value function, expectile regression, and reward function pre-training to improve performance. Experiments show that PrefRec significantly outperforms the current state-of-the-art on a variety of long-term user engagement optimization tasks.

8.2 Future Directions

In the final section of the thesis, we discuss some potential future directions that can further enhance the research in robust and adaptive policy learning.

One promising direction is to explore the integration of advanced large language models, such as GPT-4 or its successors, to enhance decision-making capabilities in complex and uncertain environments. By incorporating large language models, we can harness their ability to capture intricate relationships, patterns, and dependencies within environments. This integration can significantly contribute to more accurate and informed decision-making processes. It is worthwhile to investigate how these models can effectively understand and process textual information, leading to improved policy learning and decision-making outcomes. Additionally, transfer learning techniques can be explored using large language models, enabling the utilization of knowledge and experience gained in one problem domain to be applied to related domains. Fine-tuning and adapting pre-trained language models to new environments can facilitate faster and more efficient policy learning in diverse problem domains.

Another important direction is to explore the application of human/AI-in-the-loop RL techniques to leverage human expertise in the learning process. This involves designing interactive systems that allow human trainers, domain experts, or even language models to provide valuable feedback and guidance to the RL agent. By incorporating the iterative feedback loop between humans and AI agents, the learning process can be accelerated, policy performance can be improved, and the adaptability of learned policies can be enhanced in challenging environments. Investigating different approaches for effective collaboration between humans and AI agents within the RL framework will be crucial for achieving these benefits.

Bibliography

[1] Nitin Kamra, Umang Gupta, Fei Fang, Yan Liu, and Milind Tambe. Policy learning for continuous space security games using neural networks. In *AAAI*, pages 1103–1112, 2018. 8

[2] Nitin Kamra, Umang Gupta, Kai Wang, Fei Fang, Yan Liu, and Milind Tambe. Deep fictitious play for games with continuous action spaces. In *AAMAS*, pages 2042–2044, 2019. 8

[3] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *International Conference on Computers and Games*, pages 72–83. Springer, 2007. 8, 36

[4] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. 9, 44

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017. 9, 77

[6] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018. 9, 40

[7] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020. 9, 77

[8] Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Mohammadamin Barekatain, Simon Schmitt, and David Silver. Learning and planning in complex action spaces. In *ICML*, pages 4476–4486, 2021. 9

[9] Daewoo Kim, Sangwoo Moon, David Hostallero, Wan Ju Kang, Taeyoung Lee, Kyunghwan Son, and Yung Yi. Learning to schedule communication in multi-agent reinforcement learning. *ICLR*, 2019. 9

[10] Amanpreet Singh, Tushar Jain, and Sainbayar Sukhbaatar. Learning when to communicate at scale in multiagent cooperative and competitive tasks. *ICLR*, 2018.

[11] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *NeurIPS*, pages 7254–7264, 2018. 9, 53

[12] Abhishek Das, Théophile Gervet, Joshua Romoff, Dhruv Batra, Devi Parikh, Mike Rabbat, and Joelle Pineau. Tarmac: Targeted multi-agent communication. In *ICML*, pages 1538–1546, 2019. 9, 53, 63, 64

[13] Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *ICML*, pages 3040–3049, 2019. 9

[14] Woojun Kim, Jongeui Park, and Youngchul Sung. Communication in multi-agent reinforcement learning: Intention sharing. *ICLR*, 2021. 9, 53

[15] Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning nearly decomposable value functions via communication minimization. In *ICLR*, 2019. 10, 53, 63, 64, 68

[16] Sai Qian Zhang, Jieyu Lin, and Qi Zhang. Succinct and robust multi-agent communication with temporal message control. *arXiv preprint arXiv:2010.14391*, pages 17271–17282, 2020. 63

[17] Rundong Wang, Xu He, Runsheng Yu, Wei Qiu, Bo An, and Zinovi Rabinovich. Learning efficient multi-agent communication: An information bottleneck approach. In *ICML*, pages 9908–9918, 2020. 10, 53

[18] Huan Zhang, Hongge Chen, Chaowei Xiao, Bo Li, Mingyan Liu, Duane Boning, and Cho-Jui Hsieh. Robust deep reinforcement learning against adversarial perturbations on state observations. In *NeurIPS*, pages 21024–21037, 2020. 10

[19] Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *IJCAI*, pages 3756–3762, 2017. 10

[20] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *ICLR*, 2020. 10, 54

[21] Huan Zhang, Hongge Chen, Duane S Boning, and Cho-Jui Hsieh. Robust reinforcement learning on state observations with learned optimal adversary. In *ICLR*, 2021. 10

[22] James Tu, Tsunhsuan Wang, Jingkang Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Adversarial attacks on multi-agent communication. *arXiv preprint arXiv:2101.06560*, 2021. 10

[23] Jan Blumenkamp and Amanda Prorok. The emergence of adversarial communication in multi-agent reinforcement learning. *CoRL*, 2020. 10, 54

[24] Rupert Mitchell, Jan Blumenkamp, and Amanda Prorok. Gaussian process based message filtering for robust multi-agent cooperation in the presence of adversarial communication. *arXiv preprint arXiv:2012.00508*, 2020. 10, 54

[25] Yifei Zhao, Yu-Hang Zhou, Mingdong Ou, Huan Xu, and Nan Li. Maximizing cumulative user engagement in sequential recommendation: An online optimization perspective. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2784–2792, 2020. 10, 76, 101

[26] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. Session-based recommendations with recurrent neural networks. In *ICLR*, 2016. 10, 75, 76, 91

[27] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. BERT4Rec: Sequential recommendation with bidirectional encoder representations from transformer. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 1441–1450, 2019. 10

[28] Xiao Zhang, Haonan Jia, Hanjing Su, Wenhan Wang, Jun Xu, and Ji-Rong Wen. Counterfactual reward modification for streaming recommendation with delayed feedback. In *SIGIR '21: The 44th International ACM SIGIR*

Conference on Research and Development in Information Retrieval, pages 41–50. ACM, 2021. 11

[29] Qingyun Wu, Hongning Wang, Liangjie Hong, and Yue Shi. Returning is believing: Optimizing long-term user engagement in recommender systems. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 1927–1936, 2017. 11, 75

[30] Xing Zhao, Ziwei Zhu, and James Caverlee. Rabbit holes and taste distortion: Distribution-aware recommendation with evolving interests. In *WWW '21: The Web Conference 2021*, pages 888–899, 2021. 11

[31] Gediminas Adomavicius and YoungOk Kwon. Maximizing aggregate recommendation diversity: A graph-theoretic approach. In *Proc. of the 1st International Workshop on Novelty and Diversity in Recommender Systems*, pages 3–10, 2011. 11

[32] Ashton Anderson, Lucas Maystre, Ian Anderson, Rishabh Mehrotra, and Mounia Lalmas. Algorithmic effects on the diversity of consumption on spotify. In *Proceedings of The Web Conference 2020*, WWW '20, page 2155–2165. Association for Computing Machinery, 2020. 11

[33] Qingpeng Cai, Shuchang Liu, Xueliang Wang, Tianyou Zuo, Wentao Xie, Bin Yang, Dong Zheng, Peng Jiang, and Kun Gai. Reinforcing user retention in a billion scale short video recommender system. In *Companion Proceedings of the ACM Web Conference 2023*, page 421–426, 2023. 11

[34] Lixin Zou, Long Xia, Zhuoye Ding, Jiaxing Song, Weidong Liu, and Dawei Yin. Reinforcement learning to optimize long-term user engagement in recommender systems. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2810–2818, 2019. 11, 76, 99, 103, 114

[35] Bogdan Mazoure, Paul Mineiro, Pavithra Srinath, Reza Sharifi Sedeh, Doina Precup, and Adith Swaminathan. Improving long-term metrics in recommendation systems using short-horizon offline RL. *arXiv preprint arXiv:2106.00589*, 2021.

[36] Guy Shani, David Heckerman, and Ronen I. Brafman. An mdp-based recommender system. *J. Mach. Learn. Res.*, 6:1265–1295, 2005. 11

[37] Xueying Bai, Jian Guan, and Hongning Wang. A model-based reinforcement learning with adversarial training for online recommendation. In *NeurIPS*, pages 10734–10745, 2019. 11

[38] Shi-Yong Chen, Yang Yu, Qing Da, Jun Tan, Hai-Kuan Huang, and Hai-Hong Tang. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1187–1196, 2018.

[39] Qihua Zhang, Junning Liu, Yuzhuo Dai, Yiyan Qi, Yifan Yuan, Kunlun Zheng, Fan Huang, and Xianfeng Tan. Multi-task fusion via reinforcement learning for long-term user satisfaction in recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4510–4520, 2022. 11

[40] Minmin Chen, Yuyan Wang, Can Xu, Ya Le, Mohit Sharma, Lee Richardson, Su-Lin Wu, and Ed Chi. Values of user exploration in recommender systems. In *Proceedings of the 15th ACM Conference on Recommender Systems*, pages 85–95, 2021. 11

[41] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. Exploration: A study of count-based exploration for deep reinforcement learning. In *NeurIPS*, pages 2753–2762, 2017. 11

[42] Jing-Cheng Shi, Yang Yu, Qing Da, Shi-Yong Chen, and Anxiang Zeng. Virtual-taobao: Virtualizing real-world online retail environment for reinforcement learning. In *AAAI*, pages 4902–4909, 2019. 11

[43] Luo Ji, Qi Qin, Bingqing Han, and Hongxia Yang. Reinforcement learning to optimize lifetime value in cold-start recommendation. In *CIKM*, pages 782–791. ACM, 2021. 11

[44] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. DRN: A deep reinforcement learning framework for news recommendation. In *WWW*, pages 167–176, 2018. 11, 75, 99, 101

[45] Eugene Ie, Vihan Jain, Jing Wang, Sanmit Narvekar, Ritesh Agarwal, Rui Wu, Heng-Tze Cheng, Tushar Chandra, and Craig Boutilier. Slateq: A tractable decomposition for reinforcement learning with recommendation sets. In Sarit Kraus, editor, *IJCAI*, pages 2592–2599, 2019. 12

[46] Yuyan Wang, Mohit Sharma, Can Xu, Sriraj Badam, Qian Sun, Lee Richardson, Lisa Chung, Ed H Chi, and Minmin Chen. Surrogate for long-term user experience in recommender systems. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4100–4109, 2022. 12, 76, 99, 103

[47] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010. 12

[48] Wanqi Xue, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, and Chai Kiat Yeo. Solving large-scale extensive-form network security games via neural fictitious self-play. In *IJCAI*, pages 3713–3720, 2021. 13, 36, 44, 48

[49] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*, pages 327–334, 2011. 13, 14, 15

[50] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R Jennings. Optimal escape interdiction on transportation networks. In *IJCAI*, 2017. 15

[51] Youzhi Zhang, Qingyu Guo, Bo An, Long Tran-Thanh, and Nicholas R Jennings. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*, pages 1262–1269, 2019. 13, 16, 28, 36, 48

[52] Steven Okamoto, Noam Hazon, and Katia P Sycara. Solving non-zero sum multiagent network flow security games with attack costs. In *AAMAS*, pages 879–888, 2012. 13

[53] Branislav Bosansky, Christopher Kiekintveld, Viliam Lisy, and Michal Pechoucek. An exact double-oracle algorithm for zero-sum extensive-form games with imperfect information. *Journal of Artificial Intelligence Research*, 51:829–866, 2014. 13

[54] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016. 14, 17, 21

[55] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pages 4190–4203, 2017.

[56] Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *ICML*, pages 793–802, 2019. 14

[57] Gregory Farquhar, Laura Gustafson, Zeming Lin, Shimon Whiteson, Nicolas Usunier, and Gabriel Synnaeve. Growing action spaces. In *ICML*, pages 4335–4346, 2020. 14

[58] George W Brown. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1):374–376, 1951. 17

[59] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *ICML*, pages 805–813, 2015. 17

[60] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015. 17, 43

[61] Ji He, Jianshu Chen, Xiaodong He, Jianfeng Gao, Lihong Li, Li Deng, and Mari Ostendorf. Deep reinforcement learning with a natural language action space. In *ACL*, pages 1621–1630, 2016. 18, 38

[62] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip S Thomas. Learning action representations for reinforcement learning. In *ICML*, pages 941–950, 2019. 18

[63] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017. 18

[64] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *ICML*, pages 933–941, 2017. 20

[65] Julian RH Marino, Rubens O Moraes, Claudio Toledo, and Levi HS Lelis. Evolving action abstractions for real-time planning in extensive-form games. In *AAAI*, pages 2330–2337, 2019. 22

[66] Levi H. S. Lelis. Planning algorithms for zero-sum games with exponential action spaces: A unifying perspective. In *IJCAI*, pages 4892–4898, 2020. 22

[67] Tejas D Kulkarni, Karthik R Narasimhan, Ardavan Saeedi, and Joshua B Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *NeurIPS*, pages 3675–3683, 2016. 22

[68] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *SIGKDD*, pages 855–864, 2016. 23

[69] Wei Peng, Guohua Dong, Kun Yang, and Jinshu Su. A random road network model and its effects on topological characteristics of mobile delay-tolerant networks. *IEEE Transactions on Mobile Computing*, 13(12):2706–2718, 2013. 28

[70] Geoff Boeing. Osmnx: New methods for acquiring, constructing, analyzing, and visualizing complex street networks. *Computers, Environment and Urban Systems*, 65:126–139, 2017. 29, 46

[71] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *NeurIPS*, 2014. 31

[72] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *ICLR*, 2017. 31

[73] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *ICLR*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`. 31

[74] Sheldon M Ross. Goofspiel—the game of pure strategy. *Journal of Applied Probability*, pages 621–625, 1971. 33

[75] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *JAIR*, 32:289–353, 2008. 35, 53

[76] Wanqi Xue, Bo An, and Chai Kiat Yeo. Nsgzero: Efficiently learning non-exploitable policy in large-scale network security games with neural monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 4646–4653, 2022. 36

[77] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *AAMAS*, pages 66–83. Springer, 2017. 39

[78] Christopher D Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011. 40

[79] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016. 53

[80] Jakob N Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *NeurIPS*, pages 2145–2153, 2016. 53

[81] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *NeurIPS*, pages 2252–2260, 2016. 53, 63, 64, 67

[82] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and J Doug Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, pages 43–58, 2011. 54

[83] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010. 54

[84] Ian Goodfellow, Nicolas Papernot, Sandy Huang, Yan Duan, Pieter Abbeel, and Jack Clark. Attacking machine learning with adversarial examples. *OpenAI. https://blog. openai. com/adversarial-example-research*, 2017. 54

[85] Keen Security Lab Tencent. Experimental security research of tesla autopilot. *Tencent Security Keen Lab Blog*, 2019. URL `https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf`. 54

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, pages 5998–6008, 2017. 54, 64

[87] Wanqi Xue, Wei Qiu, Bo An, Zinovi Rabinovich, Svetlana Obraztsova, and Chai Kiat Yeo. Mis-spoke or mis-lead: Achieving robustness in multi-agent communicative reinforcement learning. In *AAMAS*, pages 1418–1426, 2022. 54

[88] Marc Lanctot, Vinicius Zambaldi, Audrūnas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, page 4193–4206, 2017. 55, 61

[89] Paul Muller, Shayegan Omidshafiei, Mark Rowland, Karl Tuyls, Julien Perolat, Siqi Liu, Daniel Hennes, Luke Marris, Marc Lanctot, Edward Hughes, et al. A generalized training approach for multiagent learning. In *ICLR*, 2019. 55

[90] Frans A Oliehoek, Christopher Amato, et al. *A Concise Introduction to Decentralized POMDPs*, volume 1. Springer, 2016. 55, 63

[91] Hubert Kirrmann. *Fault Tolerant Computing in Industrial Automation*. Switzerland: ABB Research Center, 2015. 57

[92] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 58

[93] H Brendan McMahan, Geoffrey J Gordon, and Avrim Blum. Planning in the presence of cost functions controlled by an adversary. In *ICML*, pages 536–543, 2003. 61

[94] Sanjeevan Ahilan and Peter Dayan. Correcting experience replay for multi-agent communication. *ICLR*, 2021. 63

[95] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *ICML*, pages 980–991, 2020. 63, 64, 67

[96] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft multi-agent challenge. *CoRR*, abs/1902.04043, 2019. 64, 65, 68

[97] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 66, 114

[98] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style,

high-performance deep learning library. In *NeurIPS*, pages 8026–8037, 2019. 66

[99] Wanqi Xue, Qingpeng Cai, Ruohan Zhan, Dong Zheng, Peng Jiang, Kun Gai, and Bo An. Resact: Reinforcing long-term engagement in sequential recommendation with residual actor. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=HmPOzJQhbwg`. 75, 99, 102, 103

[100] Gabriel de Souza Pereira Moreira, Sara Rabhi, Jeong Min Lee, Ronay Ak, and Even Oldridge. Transformers4Rec: Bridging the gap between NLP and sequential/session-based recommendation. In *Proceedings of the 15th ACM Conference on Recommender Systems*, pages 143–153, 2021. 75

[101] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. Sequential user-based recurrent neural network recommendations. In *Proceedings of the 11th ACM Conference on Recommender Systems*, pages 152–160, 2017. 75

[102] Jin Huang, Wayne Xin Zhao, Hongjian Dou, Ji-Rong Wen, and Edward Y Chang. Improving sequential recommendation with knowledge-enhanced memory networks. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 505–514, 2018.

[103] Bruno L Pereira, Alberto Ueda, Gustavo Penha, Rodrygo LT Santos, and Nivio Ziviani. Online learning to rank for sequential music recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 237–245, 2019. 75

[104] Xu Chen, Hongteng Xu, Yongfeng Zhang, Jiaxi Tang, Yixin Cao, Zheng Qin, and Hongyuan Zha. Sequential recommendation with user memory networks. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 108–116, 2018. 75

[105] Jiaxi Tang and Ke Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 565–573, 2018. 75, 76

[106] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3319–3327, 2020. 75

[107] Jérémie Rappaz, Julian McAuley, and Karl Aberer. Recommendation on live-streaming platforms: dynamic availability and repeat consumption. In *Proceedings of the 15th ACM Conference on Recommender Systems*, page 390–399, 2021. 75

[108] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z. Sheng, and Mehmet Orgun. Sequential recommender systems: Challenges, progress and prospects. In *IJCAI*, pages 6332–6338, 2019. 75, 101

[109] Choon Hui Teo, Houssam Nassif, Daniel Hill, Sriram Srinivasan, Mitchell Goodman, Vijai Mohan, and SVN Vishwanathan. Adaptive, personalized diversity for visual discovery. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 35–38, 2016. 76

[110] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2021. 77

[111] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 77

[112] Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An Introduction*. MIT press, 2018. 77

[113] Minmin Chen, Alex Beutel, Paul Covington, Sagar Jain, Francois Belletti, and Ed H Chi. Top-k off-policy correction for a reinforce recommender system. In *Proceedings of the 12th ACM International Conference on Web Search and Data Mining*, pages 456–464, 2019. 77

[114] Yash Chandak, Georgios Theocharous, James Kostas, Scott Jordan, and Philip Thomas. Learning action representations for reinforcement learning. In *International Conference on Machine Learning*, pages 941–950. PMLR, 2019. 78

[115] Dongyang Zhao, Liang Zhang, Bo Zhang, Lizhou Zheng, Yongjun Bao, and Weipeng Yan. Mahrl: Multi-goals abstraction based deep hierarchical reinforcement learning for recommendations. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 871–880, 2020. 78, 88, 99, 102

[116] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, pages 2052–2062, 2019. 79, 91, 113, 116

[117] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICML*, 2014. 81, 91

[118] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, pages 387–395, 2014. 83

[119] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016. 83, 90, 113, 116

[120] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, pages 1587–1596, 2018. 83, 84, 91, 113, 116

[121] Michael A Nielsen. *Neural Networks and Deep Learning*, volume 25. Determination Press San Francisco, CA, USA, 2015. 84

[122] Alexander A Alemi, Ian Fischer, Joshua V Dillon, and Kevin Murphy. Deep variational information bottleneck. *ICLR*, 2017. 85, 86

[123] Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. *NeurIPS*, 28, 2015. 90, 114

[124] Alexandre Gilotte, Clément Calauzènes, Thomas Nedelec, Alexandre Abraham, and Simon Dollé. Offline a/b testing for recommender systems. In *Proceedings of the 11th ACM International Conference on Web Search and Data Mining*, pages 198–206, 2018. 90, 114

[125] Mehrdad Farajtabar, Yinlam Chow, and Mohammad Ghavamzadeh. More robust doubly robust off-policy evaluation. In *ICML*, pages 1447–1456, 2018. 90, 114

[126] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *NeurIPS*, 34, 2021. 91, 113, 116

[127] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with in-sample Q-Learning. In *ICLR*, 2022. 91, 108, 113, 116

[128] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(11), 2008. 93

[129] Konstantina Christakopoulou, Can Xu, Sai Zhang, Sriraj Badam, Trevor Potter, Daniel Li, Hao Wan, Xinyang Yi, Ya Le, Chris Berg, et al. Reward shaping for user satisfaction in a REINFORCE recommender. *arXiv preprint arXiv:2209.15166*, 2022. 99, 103

[130] Qingpeng Cai, Ruohan Zhan, Chi Zhang, Jie Zheng, Guangwei Ding, Pinghua Gong, Dong Zheng, and Peng Jiang. Constrained reinforcement learning for short video recommendation, 2022. 99

[131] Wanqi Xue, Qingpeng Cai, Zhenghai Xue, Shuo Sun, Shuchang Liu, Dong Zheng, Peng Jiang, Gai Kun, and Bo An. Prefrec: Recommender systems with human preferences for reinforcing long-term user engagement. In *Proceedings of the 29th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2023. 100

[132] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. 100

[133] Ruohan Zhan, Changhua Pei, Qiang Su, Jianfeng Wen, Xueliang Wang, Guanyu Mu, Dong Zheng, Peng Jiang, and Kun Gai. Deconfounding duration bias in watch-time prediction for video recommendation. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 4472–4481, 2022. 101

[134] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324–345, 1952. 106

[135] Jongjin Park, Younggyo Seo, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Surf: Semi-supervised reward learning with data augmentation for feedback-efficient preference-based reinforcement learning. *ICLR*, 2022. 106

[136] Roger Koenker and Kevin F Hallock. Quantile regression. *Journal of economic perspectives*, 15(4):143–156, 2001. 108

[137] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, page 663–670, 2000. 110

[138] Łukasz Kaiser, Mohammad Babaeizadeh, Piotr Miłos, Błażej Osiński, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, Afroz Mohiuddin, Ryan Sepassi, George Tucker, and Henryk Michalewski. Model based reinforcement learning for atari. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=S1xCPJHtDB`. 110

[139] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018. 113, 116