Learning Strategic Group Formation for Coordinated Behavior with Multi-Agent Deep Reinforcement Learning

マルチエージェント深層強化学習による協調行動のための戦略的集団形成の学習

2021/6

Elhadji Amadou Oury DIALLO
ジャロ　エルハジ　アマドゥ　ウリィ

Learning Strategic Group Formation for Coordinated Behavior with Multi-Agent Deep Reinforcement Learning

マルチエージェント深層強化学習による協調行動のための戦略的集団形成の学習

2021/6

Waseda University
Graduate School of Fundamental Science and Engineering
Department of Computer Science and Communications Engineering, Research on Intelligent Software

Elhadji Amadou Oury DIALLO
ジャロ　エルハジ　アマドゥ　ウリィ

ABSTRACT

First, we examine if a team of robots or agents can develop geometric and tactical group formations by using deep reinforcement learning in adversarial multi-agent systems. This is a significant point underlying the control and coordination of multiple autonomous and intelligent agents. Although there are several possible techniques to solving this challenge, we are particularly interested in a fully end-to-end learning method where agents do not have any prior knowledge of the environment and its dynamics. We propose a scalable and distributed double DQN framework to train adversarial multi-agent systems. We show that a large number of agents can learn to maneuver, attack, and protect themselves cooperatively in diverse geometric shapes and battle tactics like encirclement, guerrilla warfare, frontal attack, flanking maneuver, and so on. We finally show that agents create emergent and collective flocking behaviors by using local views from the environment only.

Then, we propose a method using several variants of deep Q-network for learning strategic formations in large-scale adversarial multi-agent systems. The goal is to learn how to maximize the probability of acting jointly as coordinated as possible. This method is called the centralized training and decentralized testing (CTDT) framework that is based on the POMDP during training and dec-POMDP during testing. During the training phase, the centralized neural network's inputs are the collections of local observations of agents of the same team. Although agents only know their actions, the centralized network decides the joint action and subsequently distributes these actions to the individual agents. During the test, however, each agent uses a copy of the centralized network and independently decides its action based on its policy and local view. We show that deep reinforcement learning techniques using the CTDT framework can converge and generate several strategic group formations in large-scale multi-agent systems. We also compare the results using the CTDT with those derived from a centralized shared DQN and then we investigate the characteristics of the learned behaviors.

Finally, we investigate how a large-scale system of independently learning agents can collectively form acceptable two-dimensional patterns (*pattern formation*) from any initial configuration. We propose a decentralized multi-agent deep reinforcement learning architecture MAPF-DQN (Multi-Agent Pattern Formation DQN) in which a set of independent and distributed agents capture their local visual field and learn how to act to collectively form target shapes. Agents exploit their networks with a central replay memory and target networks that are used to store and update the representation of the environment as well as learning the dynamics of the other agents. We then show that agents trained on random patterns using MAPF-DQN can organize themselves into very complex shapes in large-scale environments. Our results suggest that the proposed framework achieves zero-shot generalization on most of the environments independently of the depth of view of agents.

ACKNOWLEDGMENTS

First and foremost, I would like to thank my supervisor Prof. Toshiharu Sugawara who provided me with the perfect guidance and freedom. I am also thankful to my sub-advisor Prof. Hiroshi Watanabe for his important feedback during our meetings. I am also thankful to Prof. Tetsuji Ogawa and Prof. Yuichi Sei for being on my Ph.D. committee and for their helpful feedback and comments.

I would not be where I am today without the support of my parents, siblings, uncles, aunts, grandparents, cousins, and friends. I am deeply grateful for their encouragement and financial support. They have always been here for me.

I am thankful to my co-authors, lab-mates, and friends from the Intelligent Software Lab. (ISL) at Waseda University. I am deeply grateful to Ayumi Sugiyama for helping me get settled in Japan and for his support throughout our cooperation. I have enjoyed my time at ISL.

I am also grateful to the Japanese Government for their generous financial support (MEXT scholarship) during my graduate studies.

I want to sincerely thank Lana Sinapayen for their support during my time at Sony Computer Science Laboratories. They have been one of the most incredible managers I have ever worked with. I have enjoyed every bit of communications and collaborations with Lana. I also want to thank all the scientists, engineers, interns, and HR managers I worked with at Sony CSL.

I am also thankful to all of my classmates and friends from Supdeco. Finally, I would like to express my utmost gratitude to my 5th and 6th-grade teacher, Monsieur Bandiougou Keita. He has been an inspiration to me. He taught me how to be independent and how to think critically. He went well beyond his pay grade to teach me important skills and gave me all the tools I needed to face my future. Sir, I will always be grateful to you for shaping me into the person I am today!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 MULTI-AGENT SYSTEMS

Multi-agent systems (Wooldridge 2009; Weiss 1999; Ferber 1999; Shoham and Leyton-Brown 2008) are systems that contain many autonomous agents in the same environment. They can be used to collectively solve a problem that is very difficult and complex or even impossible to solve by a single agent (Buşoniu, Babuška, and De Schutter 2010; Claus and Boutilier 1998; Panait and Luke 2005). Multi-agent systems are very useful in the sense that many systems can be modeled to cope with the limitations of the processing power of a single agent and to profit from many advantages of distributed systems such as robustness, parallelism, and scalability (Weiss 1993). For instance, many real-world systems are achieved by collective and cooperative effort. The need for collaboration between agents becomes even more evident when looking at examples such as traffic control (Dresner and Stone 2005; Wiering 2000; Tomlin, Pappas, and Sastry 1998), task allocation (Gerkey and Matarić 2004; Shapley 1953; Iijima et al. 2016; Miyashita, Hayano, and Toshiharu Sugawara 2015), ant colonies (Yeh and Toshiharu Sugawara 2016; Xiang and Lee 2008), and biological cells (Khan et al. 2003).

Definition 1.1. An agent is a software or robot system that can execute actions autonomously to reach a predefined goal. Intelligent agents act without the intervention of humans and have the ability to adapt their behavior to the changes of the environment dynamics. In a general MAS setting, agents are social in the sense that they can interact and communicate with each other to reach global or local goals.

Multi-agent systems (MASs) arise in a variety of domains including robotics (Dudek et al. 1996), distributed control (Aguilar et al. 2005), telecommunications (Hayzelden and Bigham 1998; Velthuijsen 1993; Toshiharu Sugawara 1990), and economics (Kaihara 2003; Lux and Marchesi 1999; Boutilier, Shoham, and Wellman 1997). The common pattern among all of the aforementioned examples is that the system consists of many agents that wish to reach certain global or individual

goals. While these agents can often communicate with each other by various means, such as observing each other and exchanging messages, decision-making in an intelligent MAS is challenging because the appropriate behavior of one agent is inevitably influenced by the behaviors of others, which are often uncertain and not observable.

In general, the preferences and (probabilistic) beliefs drive agents' behaviors. The state belief of an agent is important in the sense that each agent has a different attitude vis-à-vis teammates or opponents. An agent may generally have a very positive cooperative strategy, but if the agent believes that other agents will not cooperate or will get out the already established cooperation, she may probably slide out of the group formation. In short, the goal can only be reached if most of the agents work together, while the self-interested agents should be prevented from ruining the global task for the rest.

1.2 CHALLENGES OF MULTI-AGENT LEARNING

Intelligent agents can learn some interesting behaviors from nature and society, such as flocking (Reynolds 1987; Olfati-Saber 2006; Choi, Oh, and Horowitz 2009), group formation (J. P. Desai, J. Ostrowski, and Kumar 1998). For instance, in an adversarial environment, agents can learn how to limit interactions with adversaries by deliberately forming groups and integrating their know-hows of the world to optimize the likelihood of escaping adversaries and barriers. This behavior is very important in complex environments populated by a large number of agents that can sense only a very limited view (partial observability) of their environment. This collective behavior can be seen as the same strategy used by a group of humans or animals in real life to avoid starving and predators. The individual members of the team coordinate between them and generate collective behaviors that can be seen as collective intelligence because the knowledge or behavior of an isolated individual agent is not sufficient enough to learn anything useful from the environment. Hence, collective intelligence in MAS is preferred to the summation of the abilities of all agents when they work individually(Mataric 1993).

Although many studies addressed the issue of strategic group formation for coordination in the agent-based context. They usually implement predefined strategies to mimic animal's behavior. Thus, one main question is "How can agents learn to survive, defeat their

opponents, form a group by themselves without hand-designed strategies?". Unfortunately, it is not an easy task because it's hard to design a good reward scheme that makes a group of agents learn how to form a specific strategy rather than a different one. It is even harder because if agents decide to change their current strategy, they are taking the risk of annealing the already learned behaviors. In general, policies are goal-directed and can last for a relatively long time. These policies might just cease to exist when the goal no longer exists or agents achieved their predefined goal. Even if the agents do not generally have a global view of their environment, the local behavior of an individual agent can result in very complex global behaviors of the team.

In addition, scalability is very important in multi-agent systems (Rana and Stout 2000), particularly in learning systems where the environment is non-stationary because the dynamics change frequently and agents have to adapt their behaviors accordingly. Because agents are learning at the same time, any learned behavior could lead to a dynamical system, and occasionally even very basic configurations of the agents do lead to sophisticated behaviors (Shoham and Leyton-Brown 2008). Furthermore, if we have a lead agent – decision-maker – that decides which action should be taken by agents, the performance of the system might slow down.

Multi-agent learning (Stone and Veloso 2000; Buşoniu, Babuška, and De Schutter 2010; Tuyls and Weiss 2012; Panait and Luke 2005; Tan 1993) is a key technique in distributed artificial intelligence, and thus, computer scientists have been working on extending reinforcement learning (RL) (Sutton and Barto 1998) to multi-agent systems to identify appropriate behavior in complex systems, where Markov games (Littman 1994) have been recognized as the prevalent model of multi-agent reinforcement learning (MARL). We have two types of learning in multi-agent systems: centralized (the learning is done by an independent agent on its own) and distributed collective learning (learning is done by the agents as a group). Modeling multi-agent systems is a complex task due to the environmental dynamics, the action, and state spaces, as well as the type of agents because many real-world domains have very large state spaces and complex dynamics, requiring agents to reason over extremely high-dimensional observations, and thus, making optimal decisions in MAS intractable.

Recently, the so-called *deep-Q-network* achieved unexpected results by playing a set of Atari games, receiving only visual states (Mnih,

Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015). The same model architecture was used to learn different Atari games, and in some of them, the algorithm performed even better than a human player.

Definition 1.2. Multi-agent deep reinforcement learning (MADRL) is the learning technique of multiple agents trying to maximize their expected total discounted reward while coexisting within a Markov game environment whose underlying transition and reward models are usually unknown or noisy. Formally, agents use neural networks with a large number of layers as a function approximator to estimate their action values.

In MADRL, an agent's optimal behavior will be determined not just by the environment, but also by the behaviors of all other agents as well. Consequently, cooperativeness in multi-agent learning is a problem of great interest within game theory and AI. Despite the success of deep RL, research has not sufficiently been done to extend these techniques in a multi-agent context except in some specific studies (Foerster et al. 2017; Peng et al. 2017; Omidshafiei et al. 2017).

1.3 STATE-OF-THE-ART

The number of publications related to deep reinforcement learning and multi-agent systems continues to increasingly grow since deep reinforcement learning was proven to work in complex single-agent cases. These studies often focused on analyzing multi-agent systems and their challenges, exploring cooperative behaviors. Recent surveys are related to non-stationary environments, agent modeling, transfer learning in multi-agent settings, and critique of multi-agent learning.

Recent multi-agent studies can be classified in four different categories (Hernandez-Leal, Kartal, and Matthew E Taylor 2019):

- **Category 1: analysis of emergent behaviors**. In practice, papers in this category, typically do not present novel algorithms; instead, they study and assess DRL techniques in a multi-agent setting, such as DQN, PPO, and others. We can discover publications in this category that looked at emergent behaviors in the three different multi-agent scenarios: cooperative, competitive, and mixed competition-cooperation strategy.

- **Category 2: learning to communicate**. These studies look at a sub-field wherein agents could exchange messages via ordinary communication methods or through a global experience replay memory. This topic is slowly gaining some traction, yet it hasn't gotten much of the deserved exposure in the multi-agent learning literature.

- **Category 3: Learning to cooperate**.

  While learning to communicate is a nascent area of research, enabling collaboration in learning agents has a long history in multi-agent systems. Most of the research in this area is studied in either cooperative or mixed competition-cooperation settings. Furthermore, the literature on this topic often uses ideas from classical multi-agent learning and adapt them to multi-agent deep reinforcement learning techniques.

- **Category 4: Agents modeling**.

  Understanding other agents is beneficial not just for cooperating, but also for modeling opponents, predicting their strategies and objectives, and accounting for other agents' learning behavior.

  For a thorough review of the different categories, see (Hernandez-Leal, Kartal, and Matthew E Taylor 2019). The contributions of this thesis belong mainly to categories 1 and 3.

1.4 CONTRIBUTIONS AND OUTLINE OF THIS THESIS

Throughout this thesis, we will address the fundamental question: can deep reinforcement learning agents find the strategic group formation by combining the local views of individual agents in a multi-agent context and adapt to the change of opponent strategies in an adversarial environment. When we apply deep reinforcement learning techniques in MAS, we should always ask the following questions:

- What will happen when the number of agents drastically increases?

- Will the network converge?

- Will the agents be able to learn interesting strategies or policies that help them to achieve their goal or sub-goal?

- How much cooperation is required and can be achieved by the agents?

- how can agents learn which action they shall perform under given circumstances?

On one end, we have independent learners trying to optimize their own behavior without any form of communication with the other agents, and they only use the feedback received from the environment. Whereas on the other end, we have joint learners where every agent reports every step they take to every other agent before proceeding to the next step.

These are very important questions since the complexity and dimensionality of the environment grow rapidly as the population of agents increases. This thesis is an attempt to explore how agents can dynamically learn cooperative and coordinated behavior using deep RL in adversarial and cooperative MAS (Panait and Luke 2005) with a large number of agents and huge state space.

Chapter 2 is an introductory chapter that introduces general deep reinforcement learning techniques. It loosely provides the mathematical background of recent deep reinforcement learning algorithms and the importance of exploration and exploitation in RL. The next chapters outline the main contributions of this thesis.

In chapter 3, we examine whether a team of agents can learn geometric and strategic group formation by using deep reinforcement learning in a large-scale mixed cooperation-competition strategy multi-agent environment. We proposed a fully end-to-end scalable and distributed DRL framework to train and test agents without any prior knowledge of the environment and its dynamics. The results show that agents using our fully end-to-end framework create emergent and collective flocking behaviors by using local views only. This chapter is based on the following papers: (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018a) and (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b).

In chapter 4, we propose a plug-and-play framework to improve the previous framework. This framework is called centralized training and decentralized testing (CTDT). The environment is modeled as a POMDP during training and dec-POMDP during testing. The results proved that CTDT agents could optimally generate better strategies in

a shorter training time. This chapter is based on the following papers: (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2019)

In chapter 5, we propose MAPF-DQN (multi-agent pattern formation DQN) to investigate how a large-scale system of independently learning agents can collectively form 2D patterns (pattern formation) from any initial configuration. The results suggest that the MAPF-DQN framework achieves zero-shot generalization on most of the tested environments independently of agents' configurations. This chapter is based on the following papers: (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2020b) and (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2020a)

In all proposed frameworks, agents have individual goals that they should achieve by themselves without any supervision. We also investigate how the learned behaviors change according to the environment dynamics including reward schemes and learning techniques. Moreover, we extended different deep reinforcement learning techniques such as DQN (Mnih, Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015), double DQN (Hasselt 2010; Van Hasselt, Guez, and Silver 2016), dueling network architectures (Wang et al. 2015) and dueling architectures combined with double DQN by proposing a distributed and scalable MAS framework that can handle a very large number of agents.

The results will help us better understand how agents behave and interact with each other in complex environments and how agents coherently choose their actions such that the resulting joint actions are optimal. Moreover, the findings of this research can probably be applied to solve a wide range of domain-specific problems with little effort (Matthew E. Taylor and Stone 2009; Boutsioukis, Partalas, and Vlahavas 2011). The next chapter will give the necessary mathematical background of deep reinforcement learning.

2

BACKGROUND

This chapter contains material from (Elhadji Amadou Oury Diallo, Ayumi Sugiyama, and Toshiharu Sugawara 2020). The function of RL is to learn what to do next, and how to map conditions to suitable actions in order to boost a cumulative numerical feedback through a continuous interaction between agents and the environment. Essentially, The main goal of RL is to find the optimal action-selection policy.

2.1 MODEL FREE DEEP REINFORCEMENT LEARNING

In RL, agent i learns an optimal control policy π. At each step, i observes the current state s, chooses an action a using π, and receives a reward r. Then, the environment transits to the next state s_{t+1}. The cumulative discounted future reward to be maximized at time t is:

$$R_t = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t, \tag{2.1}$$

where T is the terminal time step and $\gamma \in [0, 1]$. The Q-value of a given policy π represents the utility of action a at state s, where the utility is:

$$Q_\pi(s, a) = \mathbb{E}\left[R_{t+1} \mid s_t = s, a_t = a\right] \tag{2.2}$$

The optimal Q-value Q_* is traditionally expressed as

$$Q_*(s, a) = \max_\pi \mathbb{E}\left[R_{t+1} \mid s_t = s, a_t = a\right] \tag{2.3}$$

Q-learning (Watkins 1989) is an approach that iteratively estimates the Q-function using the Bellman equation:

$$Q_*(s, a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a_{t+1}} Q_*(s_{t+1}, a_{t+1}) \mid s, a\right] \tag{2.4}$$

2.2 DEEP Q NETWORK (DQN)

Originally, Q-learning uses a table containing the Q-values of state-action pairs where we assume that a state is the screen image of a

pong game. If we apply the same preprocessing as in (Silver et al. 2016), which takes four last screen images, resizes them to 84×84, and converts them to grayscale with 256 gray levels, we would have $256^{84\times84\times4} \approx 10^{67970}$ possible game states. In the problem we try to solve, the state also usually consists of several numbers (position, velocity, RGB values) and so our state space would be almost infinite.

Because we cannot use any table to store such a huge numbers of values, we can directly extend tabular Q-learning (Watkins 1989) to deep reinforcement learning framework by using a neural network function approximator with the collection of weights $\boldsymbol{\theta}$. The weights $\boldsymbol{\theta}$ can be trained by minimizing a sequence of loss functions $\mathcal{L}_t(\boldsymbol{\theta}_t)$ that changes at each time step t:

$$\mathcal{L}\left(\theta_t\right) = \mathbb{E}_{s,a,r,s_{t+1}}\left[\left(y_t - Q\left(s,a;\boldsymbol{\theta}_t\right)\right)^2\right], \tag{2.5}$$

where y_t is the target Q-value and is defined as

$$\begin{aligned} y_t &= \mathbb{E}\left[R_{t+1} + \gamma \max_a Q_*\left(s_{t+1},a;\boldsymbol{\theta}_t\right) \mid s,a\right] && (2.6)\\ &= R_{t+1} + \gamma \max_a Q\left(s_{t+1},a;\boldsymbol{\theta}_t\right) && (2.7)\\ &= R_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_a Q\left(s_{t+1},a;\boldsymbol{\theta}_t\right);\boldsymbol{\theta}_t\right) && (2.8) \end{aligned}$$

A target network (Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015) with parameters $\boldsymbol{\theta}^-$, updated every K steps from the online network, improves the stability of DQN. Hence, y_t becomes

$$y_t = R_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_a Q\left(s_{t+1},a;\boldsymbol{\theta}_t^-\right);\boldsymbol{\theta}_t^-\right) \tag{2.9}$$

The derivative of the loss function with respect to the weights is given by $\nabla_{\boldsymbol{\theta}_t}\mathcal{L}(\boldsymbol{\theta}_t)$:

$$\nabla_{\boldsymbol{\theta}_t}\mathcal{L}(\boldsymbol{\theta}_t) = \mathbb{E}_{s,a,r,s_{t+1}}\left[\left(y_t - Q\left(s,a;\boldsymbol{\theta}_t\right)\right)\nabla_{\boldsymbol{\theta}_t}Q\left(s,a;\boldsymbol{\theta}_t\right)\right] \tag{2.10}$$

Instead of using an accurate estimate of the above gradient, it is often convenient and practical to use Stochastic Gradient Descent to optimize the weights of the loss function. Therefore, we can update the weights of the online network after every timestep. We also replace the expectations by a tuple of experiences $\langle s,a,r,s_{t+1}\rangle$ sampled from a replay memory such that the current parameter $\boldsymbol{\theta}$ is adjusted in the direction of the loss with respect to $\boldsymbol{\theta}$ as follows

$$\nabla_{\boldsymbol{\theta}_t}\mathcal{L}(\boldsymbol{\theta}_t) = \left(y_t - Q\left(s,a;\boldsymbol{\theta}_t\right)\right)\nabla_{\boldsymbol{\theta}_t}Q\left(s,a;\boldsymbol{\theta}_t\right). \tag{2.11}$$

2.3 DQN WITH DOUBLE Q-LEARNING (DDQN)

The conventional Q-learning (Watkins 1989) algorithm overestimates the action-state values (Hasselt 2010). This is due to the use of the max function when updating the Q-values. One solution is Double DQN (Van Hasselt, Guez, and Silver 2016), which consists of using two independent value functions that are randomly updated, resulting in two sets of weights, $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^{'}$. Then we use one set of weights to determine the greedy policy and the other to determine its value. Given the fact that we already have two different value functions in DQN, y_t^{DDQN} becomes

$$y_t^{(DDQN)} = R_{t+1} + \gamma Q\left(s_{t+1}, \arg\max_a Q\left(s_{t+1}, a; \boldsymbol{\theta}_t\right); \boldsymbol{\theta}_t^{-}\right) \tag{2.12}$$

Double DQN sometimes underestimates rather than overestimates the expected Q-values. However, the chance of both estimators underestimating or overestimating at the same action is low.

2.4 DQN DUELING NETWORK ARCHITECTURES (DUELDQN)

In many reinforcement learning problems, it is unnecessary to estimate the action value for each action, as some actions have little to no value for a given state. Dueling networks (Wang et al. 2015) decompose the Q-Network into a value stream $V(s; \boldsymbol{\theta}, \beta)$ and an advantage stream $A(s, a; \boldsymbol{\theta}, \alpha)$. The value stream expresses how desirable a state would be for an agent while the advantage stream compares how much better taking a certain action would be compared to the others. The relationship between value and advantage streams is expressed in the following formula:

$$A_\pi(s, a) = Q_\pi\left(s, a\right) - V_\pi\left(s\right). \tag{2.13}$$

This decomposition helps to generalize the learning for the state values. The output layer of this architecture is a combination of the value stream output and the advantage streams output. The aggregator is given by

$$Q\left(s, a; \boldsymbol{\theta}, \zeta, \beta\right) = V\left(s; \boldsymbol{\theta}, \beta\right) + \left(A\left(s, a; \boldsymbol{\theta}, \zeta\right) - \frac{1}{|A|}\sum_a A\left(s, a; \boldsymbol{\theta}, \zeta\right)\right), \tag{2.14}$$

where β is a parameter of V, ζ is a parameter of A, and $|A|$ represents the length of the advantage stream.

2.5 EXPLORATION AND EXPLOITATION

Exploration vs. exploitation is one of the main dilemmas in RL. The optimal policy for an agent is to always select the action found to be most effective based on history (exploitation). On the other hand, to improve or learn a policy, the agent must explore a new state it has never observed before (exploration) by taking non-optimal actions. We use ε-greedy to balance between exploration and exploitation with $0 \leq \varepsilon \leq 1$. At every timestep, the agent acts randomly with probability ε and acts according to the current policy with probability $1 - \varepsilon$. In practice, it is common to start with $\varepsilon = 1$ and to progressively decay ε as the training progresses until we reach the desired ε.

3

DECENTRALIZED TRAINING AND DECENTRALIZED TESTING FRAMEWORK

This chapter contains material from (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018a) and (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b).

3.1 INTRODUCTION

We address the fundamental question: can deep reinforcement learning agents find the strategic group formation by combining the local views of individual agents in a multi-agent context and adapt to the change of opponent strategies in an adversarial environment. When we apply deep reinforcement learning techniques in MAS, we always ask what will happen when the number of agents drastically increases. Will the network converge? Will the agents be able to learn interesting strategies or policies that help them to achieve their goal or sub-goal? These are very important questions since the complexity rises as the population of agents increases.

Although the main goal of this paper is to investigate the aforementioned questions in adversarial multi-agent systems with a large number of agents and state space. We assume that a single network is shared with all team members in our implementation. This means that the existence of a centralized manager like a general, but we believe that there are still a number of challenges to control the coordination of so many agents (100 to 400 agents in our experiments). It is not obvious the coordinated behavior can really emerge using deep reinforcement learning techniques. It also seems that having a network for each agent is not optimal in a large scale environment in which we have up to 400 agents per team. We would need a larger cluster of GPUs to train the system. So having teammates share the same network is adapted to quite complicated situations in which agents have to learn complicated and difficult group strategies by themselves.

3.2 RELATED WORK

In previous research, some parts of the process were hand-designed. They generally calculate the expected outcome that could be derived if a group or flock was formed. Then, they select the optimal groups to form after estimating all possible outcomes. And finally, they divide the set of agents into exhaustive and disjoint groups. One drawback of this method is that the system becomes very complex as the number of agents increases. In addition, we cannot always design and predict all possible and useful formations in a complex environment.

Balch and Arkin (1998) presented a behavior-based multi-robot team formation and formation-keeping. They demonstrated that robots could form teams to reach navigational goals, avoid hazards and simultaneously remain in formation. J. P. Desai, J. Ostrowski, and Kumar (1998), J. P. Desai, J. P. Ostrowski, and Kumar (2001), and Jaydev P Desai (1998) proposed a graph theoretical method of modeling a multi-pattern formation of robotic agents. They dealt with situations where agents have to briefly cease their formation to avoid obstacles. Barfoot and Clark (2004) studied the motion planning for formations of mobile robots, where the robots used predetermined geometrical constraints throughout their travel.

In general, flocking is studied so many in physics (Toner and Tu 1998; Levine, Rappel, and Cohen 2000) (or biophysics) using multi-agent simulations. For instance, Nathan and Barbosa (2006) studied the emergence of V-like formations during bird flight by introducing a distributes positioning rules to guide agents' movements. They also try to find the basic behavior that generates a flock and collective behavior strategies using mathematical framework. One drawback of these methods is that the strategies are always given. In other words, the agents cannot generally discover a set of new strategies. Unfortunately, defining such behaviors manually is tedious and complex.

In the aforementioned papers, the shape of formation is important and agents do not learn how to behave by themselves. In our case, how to behave is much more important and our number of agents is significantly larger. In that sense, we propose a scalable and distributed end-to-end double DQN (Van Hasselt, Guez, and Silver 2016) framework that can handle a very large number of agents. We also propose a method to constitute the input of the neural network function approximator by stacking the local view of each agent. We then explore how

agents can dynamically learn cooperative and coordinated behavior which motivates strategic team formations in an adversarial MAS. We finally address the question of whether agents can build formation, alliances or groups by using their local views only. We think that applying deep reinforcement learning techniques to multi-agent systems (Elhadji A. O. Diallo, A. Sugiyama, and T. Sugawara 2017; Leibo et al. 2017; Tampuu et al. 2017; Foerster et al. 2017; Omidshafiei et al. 2017) can have various applications. However, these techniques are very hard to train and their convergence is not always guaranteed.

3.3 PROBLEM

3.3.1 *Multi-agent reinforcement learning*

A Multi-agent Markov game is defined as $\langle N, S, A, R, T \rangle$, where $N = \{1, \ldots, n\}$ contains n agents, S is the state space, $A = A^1 \times \cdots \times A^n$ represents the joint action space of agents (where A_i is the action space of agent i), $R : S \times A \to \mathbb{R}$ is the common expected reward function; and $T : S \times A \times S \to [0, 1]$ is a probabilistic transition function. At each timestep t, agent i takes action a_t^i sampled from the policy π_t^i in state $s_t \in S$, where $\pi^i : S \times A^i$ and $s_t = \left(s_t^1, \ldots, s_t^n\right)$. When a joint action $a_t = \left(a_t^1, \ldots a_t^n\right)$ is executed, the environment transits from s_t to s_{t+1} with a probability $p(s_{t+1}|s_t, a_t) \in T$ and agent i receives a reward $r_t^i = R(s_{t+1}|s_t^i, a_t^i)$. The goal of the agents is to find a deterministic joint policy $\pi = \left(\pi^1, \ldots \pi^n\right)$ so as to maximize the sum of their individual reward $r_t = \sum_{i=1}^{n} r_t^i$.

3.3.2 *Adversarial MAS environment*

In this paper, we used a well-known adversarial multi-agent domain, the so-called "Battle game" (Sukhbaatar, Szlam, and Fergus 2016), in which two teams of many agents are fighting against each other. Agents of the same team cooperate with their teammates to find some strategies in order to defeat the opponent team. The main goal is to kill as much opponents as possible by invading the neighbor territories and by using some warfare strategies.

Our environment operates on two (2) teams of n agents and runs from some initial positioning of the agents inside a square in two-

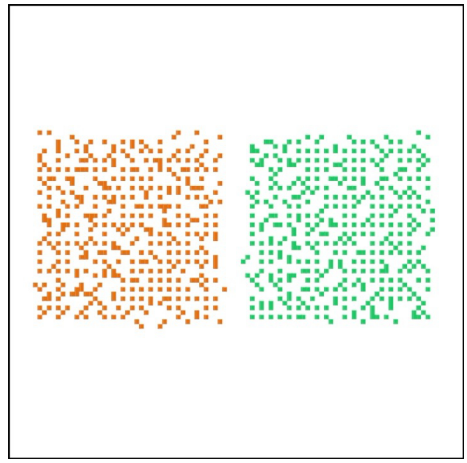


Figure 3.1: Initial state

dimensional space for each team. Each agent is randomly assigned a unique ID initially and keeps the same ID during the whole training. During each episode, the agents start at a random position inside the initially allocated area (see Fig. 3.1. The IDs of alive and dead agents are recorded in order to constitute the input of each team's network at any given timestep.

An agent can take up to 7 actions: going up, going down, going left, going right, turning left, turning right and shooting. An agent is dead after being attacked three (3) times. Agents can only attack the nearby opponents located one cell ahead its current position. To simplify the environment dynamics, an agent is not able to attack an opponent located outside of its own field of view.

In this environment, we consider homogeneous agents with same speed and action space. They also have the same capabilities to sense the environment by direct perception. Each network has a Double DQN to give each agent the instruction for strategic attack or defense. The game ends when all agents on one team have been killed or after $1,000$ timesteps. The winning team is the one who has the highest number of alive agents.

3.3.3 *Reward scheme*

Each agent receives a reward of -0.01 after each step. An agent gets $+5$ when it kills an opponent, and receives -5 when the agent is killed

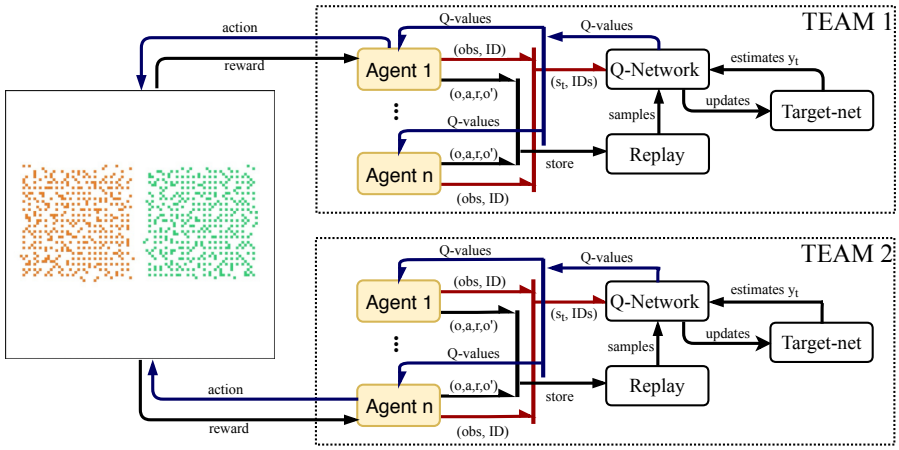


Figure 3.2: Scalable adversarial MAS architecture. The environment constitutes of two teams of n agents ($n = 100, 200, 300, 400$) each at the beginning of each episode.

by an opponent. An agent receives a small negative reward of -0.01 whenever it attacks an empty position. This reduces the number of attempts an agent can attack empty positions, and thus making the training slightly faster. By reward shaping (Ng 2003; Skinner 1990), an agent receives $+1$ for attacking an opponent, and -1 if it is being attacked. The global reward of each team is the sum of all the rewards received by agents of the same time at any timestep t, $r_t = \sum_{i=1}^{n} r_t^i$.

3.4 METHOD

3.4.1 *Proposed learning framework*

We proposed a distributed learning framework for training adversarial multi-agent systems with a large number of agents (Fig. 3.2). In our framework, each agent has its own local view s_t^i, its own action space a_t^i and its own reward r_t^i. At every timestep t, an individual agent observes its local environment, takes an action, and receives a local scalar reward. Agents exclusively observe their unique actions and therefore do not watch each other's actions. At every timestep, one joint action $a_t = \left(a^1, \ldots, a^n\right)$ is taken and the environment generates a global state $s_t = \left(s^1, \ldots, s^n\right)$ from wherein every individual agent i receives its local view s^i. We assume that an agent acts on the basis of its

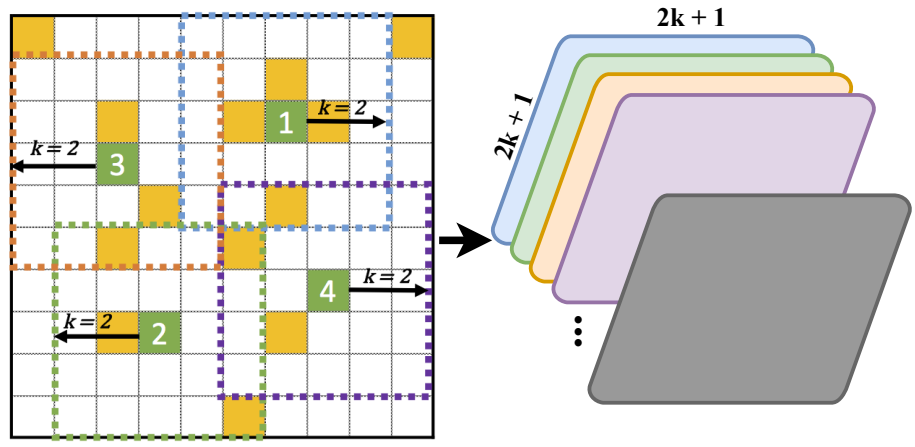


Figure 3.3: 3D Tensor observation.

individual policy and local view, and we do not assume any additional communication between agents. This seems to embed communication implicitly through the actions, local observations and states. Recall that agents have the same speed and equal processing capabilities.

In this model, all members of the same team use the same neural network. The deep Q network of each team receives a "tensorized" observation at time t (see section 3.4.2). Each agent asynchronously infers its action from the output of the network which contains the Q-values of every possible action for agents. Agents have the capacity to rapidly reach proficiency by using our proposed framework. Because the network outputs all possible q-values for each agent, we can apply different exploration strategies or learning techniques. For example, we can use a greedy exploitation strategy where the agents take the maximum q-values, or by using ε-greedy. We can also use different neural network structures for each team such as deep recurrent Q-network (Hausknecht and Stone 2015), dueling networks (Wang et al. 2015), or deep deterministic policy gradient (Lillicrap et al. 2015).

3.4.2 *Observation*

The main environment is represented as a square grid. The image is converted into grayscale. And then every agent uses a filtering method to delimit its local view controlled by the parameter k from Fig. 3.3. The input of each network is a tensorized stack of all local views of agents at

any given timestep. Each agent senses distances to neighboring agents within a radius k from its current position. Each channel represents the local view s_t^i of each individual agent of the same team. Using only this information, a network can learn a policy that is able to establish and maintain a certain group formation and to cooperatively locate and eliminate the opponents.

This representation can be thought of as an aggregation of all inputs from different sensors of many robots. For instance, if we have many cleaning or patrolling robots, we could use this method to represent the input of their learning techniques. One advantage of this representation is that we can simulate, for example, the state availability of any robot, by including a probabilistic function that selects the availability of a robot state. In our implementation, the local view s_t^i of a dead agent is $s_t^i = \mathbf{1}_{(2k+1,2k+1)}$, an identity matrix of size $(2k+1, 2k+1)$ whose entries are 1. This practically means that the local views of a dead agent is white as is the background.

3.4.3 *Exploration vs exploitation*

Exploration vs. exploitation is one of the main dilemmas in RL. The optimal policy for an agent is to always select the action found to be most effective based on history (exploitation). On the other hand, to improve or learn a new policy, the agent must explore new state it has never observed before (exploration) by taking non-optimal actions. We use ε-greedy (Sutton and Barto 1998) to balance between exploration and exploitation with $0 \leq \varepsilon(t) \leq 1$. At every timestep, the agent acts randomly with probability ε and acts according to current policy with probability $1 - \varepsilon(t)$. In practice, it is common to start with $\varepsilon = 1$ and to progressively decay ε as the training is going until we reach the desired ε. In general, we could use a fixed ε value. However, this is not always an optimal choice, because after many episodes of training, we would have a more accurate estimation of the policies and we could reduce the amount of exploration.

We decayed the values of ε based on the *piecewise linear decay* function by dividing the training episodes into many subsets and then linearly decreasing the value of ε from the maximum to the minimum predefined ε value for each interval. However, this is rarely done in reality, as determining a decay function requires a tedious parameter search, and is extremely domain dependent. The resulting values are shown

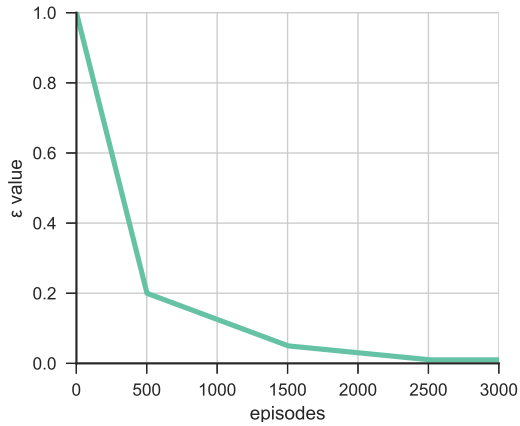


Figure 3.4: ε-greedy piecewise linear decay.

in Fig. 3.4. The initial and final value of ε are, respectively, 1 and 0.01. In this case, our agents will be able to extensively explore during the first episodes and use the optimal policy without much exploration in the end of the training as agents get more knowledge about the environment.

3.4.4 *Double DQN*

Recall that, in RL, agent i learns an optimal control policy π^i with a goal of maximizing cumulative discounted future reward at time t_0 as $R_t = \sum_{t=t_0}^{T} \gamma^{t-t_0} r_t$, where T is the terminal timestep and $\gamma \in [0, 1]$.

Q-learning (Watkins 1989) is an approach that iteratively estimate the Q-function using the Bellman optimality $Q^*(s, a) = \mathbb{E}\left[r + \gamma \max_a Q^*(s_{t+1}, a) | s, a\right]$. We can use a parameterized value function $Q(s, a; \boldsymbol{\theta}_t)$ to estimate the Q-values when the task is complex. Then, we update the parameters by using the following formula: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta(y_t - Q(s_t, a_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t; \boldsymbol{\theta}_t)$, where α is a scalar step size and y_t^Q is the target value function.

$$y_t^Q = R_{t+1} + \gamma \max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t) \tag{3.1a}$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t) \tag{3.1b}$$

DQN Mnih, Kavukcuoglu, Silver, Graves, et al. 2013; Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015 is an extension of Q-learning that uses a stack of convolutional neural network and fully connected layers to estimate

$Q(s, a; \boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are the weights of the network network function approximator. We use a separate network to compute y_t^{DQN}. The second or target network would have the exact same design as that of the DQN network, but with fixed properties $\boldsymbol{\theta}^-$ that are updated after every τ from the online network. This leads to more stable training because it keeps the target $\boldsymbol{\theta}^-$ fixed for a while. Now, we can can rewrite Eq. 3.1 as:

$$y_t^{DQN} = R_{t+1} + \gamma \max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t) \tag{3.2a}$$

$$= R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \boldsymbol{\theta}^-); \boldsymbol{\theta}^-) \tag{3.2b}$$

And we finally update the neural network weights by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta(y_t^{DQN}(s_t, a_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(s_t, a_t; \boldsymbol{\theta}_t) \tag{3.3}$$

The traditional Q-learning (Watkins 1989) algorithm overestimates the action-state values (Hasselt 2010). This is due to the use of the max function when updating the Q-values. One solution is Double DQN (Van Hasselt, Guez, and Silver 2016) that consists of using two independent value functions that are randomly update, resulting in two sets of weights, $\boldsymbol{\theta}^1$ and $\boldsymbol{\theta}^2$. Then we use one set of weights to determine the greedy policy and the other to determine its value. Given the fact that we already have two different value functions in DQN, y_t^{DDQN} becomes:

$$y_t^{DDQN} = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}^-) \tag{3.4}$$

We stored the experience (s, a, r, s_{t+1}) in a replay buffer $\mathcal{D}$ and sampled mini-batches (L.-J. Lin 1992) from it to train the network by using the standard mean squared error (MSE) loss. In the beginning, the buffer is filled with experiences played by random agents.

$$\mathcal{L}(\boldsymbol{\theta}_t) = \mathbb{E}_{s,a,r,s_{t+1}\sim\mathcal{D}}\left[\left(y_t^{DDQN} - Q(s, a, \boldsymbol{\theta}_t)\right)^2\right] \tag{3.5}$$

Double DQN is shown to sometimes underestimate rather than overestimate the expected Q-values. However, the chance of both estimators underestimating or overestimating at the same action is low. See Algorithm 1 for more details.

Algorithm 1: Adversarial multi-agent concurrent DQN with Double Q-learning

for *team* $k = 1$ **to** 2 **do**
  Initialize online network and target network with random weights $\boldsymbol{\theta}^k = \boldsymbol{\theta}^{(k,-)}$
  Initialize experience replay memory $\mathcal{D}^k$
  for *episode* $= 1$ **to** *number_episodes* **do**
    Populate team k with n agents
    for *timestep* $t = 1$ **to** *max_timesteps* **do**
      for *agent* $i = 1$ **to** n **do**
        compute the current state of agent s_t^i
        $a_t^i \leftarrow \begin{cases} \text{Select random action } a_t^i & \text{with probability } \varepsilon \\ \arg\max_{a_t^i} Q(s_t^i, a_t^i; \boldsymbol{\theta}^k) & \text{with probability } 1-\varepsilon \end{cases}$
        Apply action a_t^i and observe reward r_t^i and next state s_{t+1}^i
        Store transition $\langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle$ in the experience buffer $\mathcal{D}^k$
        Uniformly sample batch of transitions $\langle s_j^i, a_j^i, r_j^i, s_{j+1}^i \rangle$ from $\mathcal{D}^k$
        if s_{j+1}^i *is terminal* **then**
          $y_j^i \leftarrow r_j^i$
        else
          $y_j^i \leftarrow r_j^i + \gamma Q\left(s_{t+1}^i, \arg\max_{a^i} Q\left(s_{t+1}^i, a^i; \boldsymbol{\theta}_t^k\right); \boldsymbol{\theta}_t^{k,-}\right)$
        end
        if $\left| y_j - Q\left(s, a; \boldsymbol{\theta}_t^k\right) \right| < \delta$ **then**
          $\mathcal{L}(\boldsymbol{\theta}_t^k) \leftarrow \frac{1}{2}\left(y_t - Q\left(s, a; \boldsymbol{\theta}_t^k\right)\right)^2$
        else
          $\mathcal{L}(\boldsymbol{\theta}_t^k) \leftarrow \delta\left| y_j - Q\left(s, a; \boldsymbol{\theta}_t^k\right) \right| - \frac{1}{2}\delta^2$
        end
        $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t^k) = \left(y_j - Q\left(s, a; \boldsymbol{\theta}_t^k\right)\right) \nabla_{\boldsymbol{\theta}} Q\left(s, a; \boldsymbol{\theta}_t^k\right)$
        $\zeta_t^k = (1-\gamma_1)\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t^k) + \gamma_1 \zeta_{t-1}^k$
        $\varkappa_t^k = (1-\gamma_2)\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t^k)\right)^2 + \gamma_2 \varkappa_{t-1}^k$
        $\hat{\zeta}_t^k = \frac{\zeta_t^k}{(1-(1-\gamma_1)^t)}$
        $\hat{\varkappa}_t^k = \frac{\varkappa_t^k}{(1-(1-\gamma_2)^t)}$
        $v_t^k = \eta \frac{\hat{\zeta}^k}{\sqrt{\hat{\varkappa}_t^k + \epsilon}}$
        $\boldsymbol{\theta}_{t+1}^k = \boldsymbol{\theta}_t^k - v_t^k$
      end
      Clear dead agents
      Decay the health of agents who have been shot
      Every τ step set $\boldsymbol{\theta}^{(k,-)} \leftarrow \boldsymbol{\theta}^k$
    end
    Clear the replay memory $\mathcal{D}^k$
  end
end

3.4.5 *Adam Optimizer*

One approach to adaptively calculate the learning rates is Adam or Adaptive Moment Estimation (Kingma and Ba 2014). Adam (Kingma and Ba 2014), like Adadelta (Zeiler 2012), Adagrad (Duchi, Hazan, and Singer 2011), and RMSprop (Tieleman and Hinton 2012) not only preserves an exponentially decreasing mean of previous gradients ξ_t similar to a momentum, but also the previous residual gradients $\varkappa$. These gradients are calculated as follows:

$$\xi_t = (1 - \gamma_1) g_t + \gamma_1 \xi_{t-1} \tag{3.6}$$

$$\varkappa_t = (1 - \gamma_2) g_t^2 + \gamma_2 \varkappa_{t-1} \tag{3.7}$$

As ξ_t and $\varkappa_t$ are initialized as vectors of 0's, Kingma and Ba (2014) discovered that the gradients are particularly skewed towards zero during the beginning, more so when both γ_1 and γ_2 are nearly equal to 1. They corrected the bias of the first and second derivatives, to compensate for the aforementioned distortions, as

$$\hat{\xi}_t = \frac{\xi_t}{(1 - (1 - \gamma_1)^t)} \tag{3.8}$$

$$\hat{\varkappa}_t = \frac{\varkappa_t}{(1 - (1 - \gamma_2)^t)} \tag{3.9}$$

Finally, the weights are updated as

$$v_t = \eta \frac{\hat{\xi}}{\sqrt{\hat{\varkappa}_t} + \epsilon} \tag{3.10}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - v_t \tag{3.11}$$

3.5 EXPERIMENTS AND DISCUSSION

3.5.1 *Experimental settings*

In this experiment, we train all networks from scratch to ensure a smooth convergence in the learning process by using *stochastic gradient descent* and ADAM optimizer (Kingma and Ba 2014). We use the following parameters for all our experiments: learning rate $\alpha = 0.00025$, discount factor $\gamma = 0.99$, an input of shape $(n, 2k + 1, 2k + 1)$. We

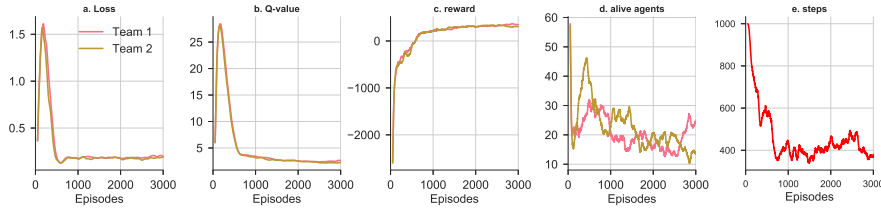


Figure 3.5: Training result of our adversarial multi-agent with $n = 100$ and $k = 3$.

update the target network every $10,000$ timesteps. To stabilize learning, we feed the network with medium size mini-batches of 250 samples. An episode is terminated after $1,000$ even if we still have alive agents on both teams. The double DQN (Van Hasselt, Guez, and Silver 2016) network structure is similar to the one from (Mnih, Kavukcuoglu, Silver, Rusu, et al. 2015) with slightly different parameters. The experimental results in the following sections describe the average values of ten (10) experimental runs with different random seeds.

3.5.2 *Convergence and improvement of group behaviors*

Fig. 3.5 represents the training performance of our baseline implementation where we have 100 agents per team with a narrow local field of view ($k = 3$). The average squared error of each team after each individual episode is shown in Fig. 3.5a. In the beginning, the losses were high because of the high values of $\varepsilon(t)$. After more or less than $1,000$ episodes and smaller values of $\varepsilon(t)$, each network's training loss converges to a very low value close to null. The environment is non-stationary because both teams are simultaneously learning. The best way to know if teams are learning strategic formations is to check if both networks converge to approximately the same values. In this case, the difference between the two errors is negligible.

By using double DQN (Van Hasselt, Guez, and Silver 2016), we make sure that the networks do not overestimate the Q-values, and consequently, the action-state values will converge to the actual Q-values. The average Q-values of each team is shown in Fig. 3.5b. Agents were taking some random actions by sampling from high values of $\varepsilon(t)$ in the beginning. After training sufficiently enough, agents started to take good actions based on their near-optimal policies.

Fig. 3.5c shows the average reward per episode for each team during training time. We see that both networks tend to have approximately the same reward. Our reward scheme could make a group of agents learn how to efficiently form various strategic group formations based on their opponents' strategies.

Fig. 3.5d shows the number of remaining (alive) agents at the end of each episode for each team. This helps us evaluate how often each team wins and to know exactly how many agents are still alive at the end of an episode. First, the team with the largest number of agents at time t uses some tactics to eliminate as many opponents as possible in a very short time before the opponents' counter-attack. Then, both teams reach a kind of equilibrium in which they use similar group formation strategies (mainly frontal attack). And finally, agents periodically update their strategies to effectively defend themselves against the opponents. It is worth to mention that the training is a bit slow when $k = 3$(see Fig. 3.5e)

3.5.3 *Scalability*

The input of each network is smaller when k is smaller because the input shape is $(n, (2k+1), (2k+1))$. That mainly justified why we observed low loss values for $k = 3, 4$ (Fig. 3.5a, Fig. 3.6a, Fig. 3.6f and Fig. 3.6k). Moreover, all networks started to use their optimal policies after $1,000$ episodes. From a practical perspective, this is a good sign because the performance of MAS often depends on how quickly the agents reach a global convergence. No matter what the values of k are, all networks converge to the same Q-values soon or late (Fig. 3.6b, Fig. 3.6g, and Fig. 3.6l). As the number of agents increases, the networks converge to the actual Q-values independently to their local field of views.

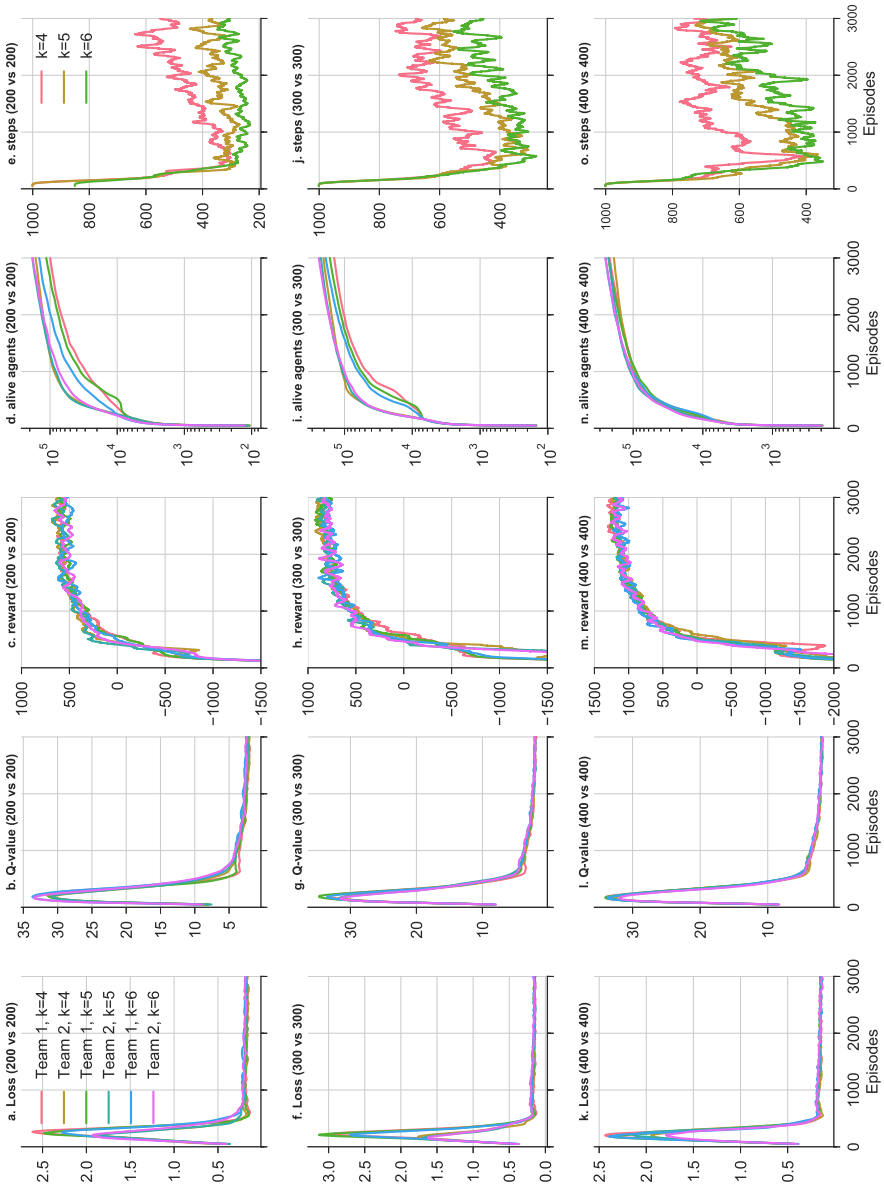


Figure 3.6: Training performance of our adversarial multi-agent with different number of agents ($n = 200, 300, 400$) and field of view ranges ($k = 4, 5, 6$). (d), (i), and (n) show the cumulative number of alive agents.

Despite the fact that all settings did converge, we can see that the agents with $k = 3$ take much longer to learn a new strategy or to adapt their behaviors. This can be explained by the fact that with a small field of views, agents cannot see the surrounding opponents, hence their lack of good defense strategies. As k increases, the average number of steps is much lower for $k = 5, 6$. After $1,000$ episodes, both teams started to learn not only how to attack but how to defend themselves for a relatively longer period. That's why the number of steps increased for $k = 4, 5, 6$ and not for $k = 3$ (Fig. 3.6e, Fig. 3.6j, and Fig. 3.6o). A wider range of view allows agents to acquire excellent assault and defensive methods at the same time.

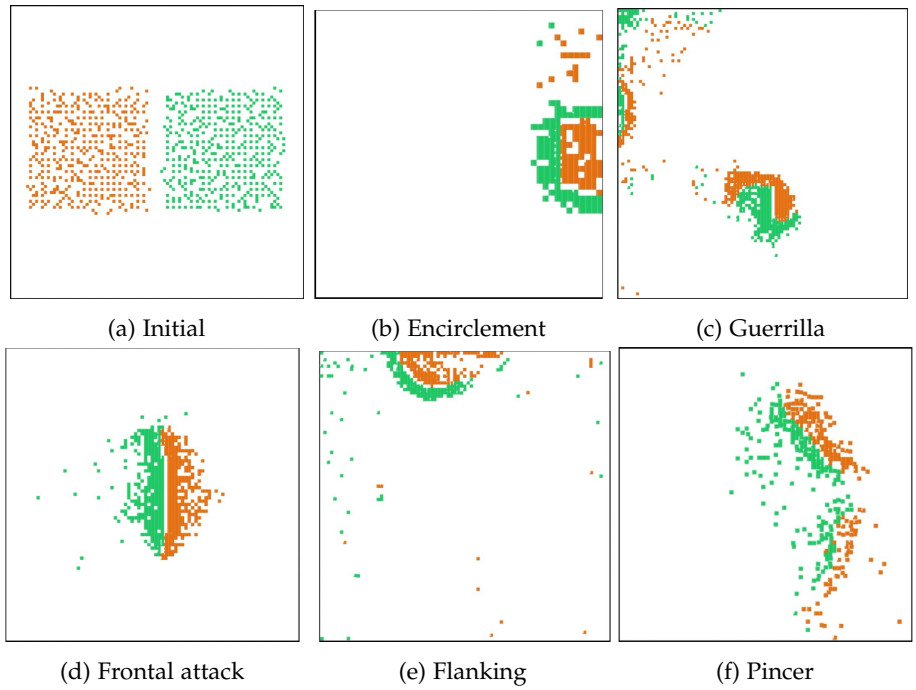
Even though both sides acquired similar behaviors and can kill about the same amount of enemies with a a wider range of view k, they likewise swiftly co-adapt themselves based on the enemies' actions by developing some strong defense techniques. We should also note all networks have approximately the same values as the number of agents increases ($n = 100, 200, 300, 400$). From Fig. 3.5, Fig. 3.6, and Table 3.1, we can conclude that our proposed learning framework is scalable in non-stationary and adversarial learning environments with a large number of agents.

Table 3.1: Training performance: Average values over 10 experiments with different number of agents and field of view sizes.

#agents	k	Loss		#alive agents		Reward		Q-value		steps
		Team 1	Team 2	Team 1	Team 2	Team 1	Team 2	Team 1	Team 2	
100	3	**0.303**	**0.281**	21.109	23.051	**128.474**	118.974	5.348	5.118	460.576
	4	0.353	0.345	25.284	30.736	117.003	**140.332**	6.243	6.255	301.951
	5	0.355	0.358	**32.486**	28.959	118.917	108.344	6.816	6.556	256.376
	6	0.383	0.379	31.509	**31.998**	102.145	104.566	**7.133**	**7.035**	**244.876**
200	3	**0.295**	**0.261**	33.761	39.060	215.191	168.515	5.128	4.890	684.943
	4	0.352	0.305	34.373	56.187	**215.993**	255.494	5.627	5.701	473.456
	5	0.354	0.316	39.574	**62.378**	203.631	**267.478**	5.705	5.936	377.649
	6	0.383	0.349	**49.901**	62.092	202.672	217.087	**6.560**	**6.393**	**320.083**
300	3	**0.299**	**0.242**	43.794	54.378	286.032	159.127	5.180	4.852	794.569
	4	0.336	0.269	49.365	70.015	288.406	277.681	5.458	5.386	612.715
	5	0.355	0.277	56.808	76.424	**316.786**	**296.996**	**5.733**	**5.566**	505.819
	6	0.338	0.281	**65.234**	**81.860**	298.364	292.835	5.675	5.529	**441.929**
400	3	**0.310**	**0.262**	51.825	64.898	393.773	242.853	5.358	5.160	878.280
	4	0.313	0.303	75.286	65.314	404.345	378.065	5.645	5.519	692.123
	5	0.324	0.315	78.226	77.399	**451.360**	**425.521**	**5.768**	**5.733**	583.225
	6	0.318	0.295	**85.507**	**86.543**	434.689	406.367	5.716	5.509	**529.913**

3.5.4 *Learned team strategies*

In the following, we describe the most common observed strategic group formations learned by agents. It seems like most of the following strategies always appeared in all experiments. As we train agents, we might observe slightly variant tactics which do have the same roots. For instance, it is not unusual to observe an improved attack and/or defense tactics appear again. Agents prefer short-term and aggressive strategies to long-term and safe ones sometimes.



(a) Initial (b) Encirclement (c) Guerrilla

(d) Frontal attack (e) Flanking (f) Pincer

Figure 3.7: Some learned team strategies.
Videos available at https://drive.google.com/drive/folders/1QLdiLvO_nSO2VLoYsJHngNwZwKvdqMgG?usp=sharing

Encirclement. This situation is highly dangerous for the encircled team. The agents of the encircled team can be subject to an attack from several sides. However, if there are some obstacles inside the environment or on one side of it, it would be much harder to achieve a full encirclement attack. (Fig. 3.7b)

Guerrilla warfare. It is a strategy in which a small number of agent use mobility tactics to fight a larger and less-mobile opponent team. Agents tend to avoid confrontation with a larger number of agents but seek and attack small groups while minimizing their losses. The main goal is to inflict damage on a target and immediately move away from the location from where the attack did happened to avoid the opponents' defense or counter-attack (hit-and-run tactics, fire-and-move, shoot-and-scoot) (Fig. 3.7c).

Frontal attack. This is a straightforward and aggressive advance of agents toward the enemy's frontline. By attacking the enemy's front, the attacking team expose themselves to the enemy's full defensive capabilities. Unfortunately, this strategy is sometimes proved to be increasingly suicidal when the team is outnumbered (Fig. 3.7d).

Flanking maneuver This consists of a movement of agents of the same team around a flank to achieve an advantageous position over enemies. This is a safe strategy in which agents are not risking themselves, while at the same time they gradually weaken the opponents. Agents easily form echelons in which they are diagonally aligned. Each agent is behind and to the right (left) of an ahead agent – right (left) echelon. However, this strategy is not always safe when the team is outnumbered. (Fig. 3.7e)

Pincer movement. Similarly to the flanking maneuver, this is a tactic in which agents of the same team simultaneously attack both flanks of the opponent team. This generally leads to a frontal attack on each flank. By also attacking the opponent team from the rear, the opposing team might be easily encircled. (Fig. 3.7f)

3.5.5 *Discussion*

Our reward structure appears to make this domain a negative-sum game as points are lost for time and sores are balanced for attacks and kills only during the beginning of the training. In other words, the positive and negative rewards of all will add up to less than zero only during the first episodes when the value of ε are high. Then, the environment becomes a zero-sum game before generating positive-sum outcomes in which the sum of positive and negative rewards is greater than zero. This becomes possible when the policies of each network are at least semi-optimal. In addition, it is interesting to observe that random agents are not able to generate coordinated behaviors.

In our framework, each agent has its own Q-values and they act jointly in a decentralized manner. Therefore, the problem is much more complex than having centralized learning, where two agents compete with each other in a grid world. One of the main advantages of our method is that we do not necessarily need to retrain the network from scratch for a different number of agents. Even though our agents are homogeneous, it is easy to see that our agents can adapt their strategies against the change of opponents' strategies. This is due to the fact that agents are trained on a very large number of gameplay such that they have seen nearly every scenario combination and have taken many possible actions and know which ones have the most resultant value.

4

CENTRALIZED TRAINING AND DECENTRALIZED TESTING FRAMEWORK

This chapter contains material from (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2019).

4.1 INTRODUCTION

Many studies have addressed the issue of strategic group formation for coordination in the agent-based context (J. P. Desai, J. P. Ostrowski, and Kumar 2001; Barfoot and Clark 2004; Reynolds 1987), they usually implement some predefined strategies by mimicking animal or human behavior. One key question is how agents can learn to survive, defeat their opponents, and form a group by themselves without any hand-designed strategy. Unfortunately, it is not easy to design a good reward scheme that makes a group of agents learn how to form a specific strategy rather than individual ones (Elhadji Amadou Oury Diallo, Ayumi Sugiyama, and Toshiharu Sugawara 2020). It is even harder if agents decide to change their current strategy because they are taking the risk of annealing the already learned behaviors. Besides, most of the previous studies focused on an environment with a small number of agents for the sake of simplicity, and they either assumed that the environment is fully observable (Sukhbaatar, Szlam, and Fergus 2016; He et al. 2016) or the training and testing phases are somehow centralized (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b).

This paper aims to propose an end-to-end learning framework that can generate coordinated collective behaviors to control a very large number of agents in an adversarial multi-agent system. We first propose a framework for training variants multi-agent deep reinforcement learning techniques such as deep Q-network (DQN), double DQN (DDQN), dueling DQN (duelDQN), and dueling double DQN (DuelDDQN) – all of which are based on the dec-POMDP framework (Bernstein et al. 2002; Pynadath and Tambe 2002). In our framework, agents of the same team use a centralized network during the training phase. The network

receives a joint observation and outputs a joint action. However, during the test phase, each agent uses its network and infers its action by using its local observation only. This is called a *centralized training and decentralized framework* (CTDT). We demonstrate that the CTDT method can converge and generate strategic group formations for a large-scale multi-agent system in a non-stationary and adversarial environment. We then compare these experimental results with those derived from the centralized DQN framework (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b). Finally, we analyze the characteristics of the learned strategies.

4.2 RELATED WORK

Leibo et al. (2017) analyzed the dynamics of policies learned by multiple independent learning agents, each of which used an independent DQN. They introduced the concept of sequential social dilemmas in fruit gathering and wolfpack hunting games and showed how different behaviors can emerge from competition over shared resources in conflicting situations. He et al. He et al. 2016 presented a neural-based model that jointly learns a policy and the behavior of its opponent and found that opponent modeling is often necessary for competitive multi-agent systems. Then, they proposed a method that automatically discovers different strategy patterns of the opponents without any extra supervision from the system.

Lowe et al. (2017a) demonstrated the complexity of using deep reinforcement learning in non-stationary environments since each agent's policy is changing. They also proposed a framework utilizing an ensemble of policies resulting in a more resilient overall policy that only uses local information at execution time. Sukhbaatar, Szlam, and Fergus (2016) proposed a framework in which agents need to communicate by using specific communication protocols. In their framework, cooperating agents learn to communicate amongst themselves before taking any action. Because the protocol is learned, it is very hard to interpret the language used by agents. It is also not easy to increase the number of agents due to the communication cost between agents.

Nathan and Barbosa (2006) studied the emergence of V-like formations during bird flight by introducing distributed positioning rules to guide agent movements. They also tried to identify the basic behavior that characterizes a flock and collective behavior strategies using a

mathematical framework. One drawback of this method is that the strategies are always given. Unfortunately, the system becomes very complex as the number of agents increases. Also, we cannot always design and predict all possible and useful formations in a complex environment.

Scalability is another important element in multi-agent systems (Rana and Stout 2000; Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b), particularly in learning systems where the environment is non-stationary, as the dynamics change frequently and agents have to adapt their behaviors accordingly. The simultaneous learning of the agents means that every learning rule leads to a dynamic and complex system, and sometimes even very simple learning rules can lead to very complex global behaviors (Shoham and Leyton-Brown 2008). In contrast, in our method, a large number of agents learn how to cooperate without any explicit communication and opponent modelling strategies. This leads to improved performance and scalability over communicative agents.

4.3 BACKGROUND

4.3.1 *Dec-POMDP*

An optimal approach to model this problem could be the dec-POMDP (Bernstein et al. 2002; Pynadath and Tambe 2002) framework. A dec-POMDP or decentralized partially observable Markov decision process is expressed as $\langle \mathcal{D}, \mathcal{S}, \boldsymbol{\mathcal{A}}, \mathcal{T}, \mathcal{R}, \boldsymbol{\Omega}, \mathcal{O}, h, \mathcal{I} \rangle$, in which

- $\mathcal{D}$ is a set of n homogeneous and/or heterogeneous agents;
- $\mathcal{S}$ is defined as a finite set of all possible states s;
- $\boldsymbol{\mathcal{A}}$ represents the joint actions of all agents, where $\boldsymbol{\mathcal{A}} = \mathcal{A}^1 \times ... \times \mathcal{A}^n$ and A^i is the finite action of agent $i \in \mathcal{D}$;
- $\mathcal{T}$ is a probabilistic transition;
- $\mathcal{R}$ is the reward agents get immediately;
- $\boldsymbol{\Omega}$ contains the combined joint observations of agents, where $\boldsymbol{\Omega} = \mathcal{O}^1 \times ... \times \mathcal{O}^n$ and $\mathcal{O}^i$ is the set of observations available to any given agent;
- $\mathcal{O}$ is expressed as the observational probabilities function;

- h is a positive integer specifying the given environment's horizon; and
- $\mathcal{I}$ initially contains the first states during the beginning.

When a joint action $\mathbf{a}_t = \langle a_t^1, \dots a_t^n \rangle$ is executed, the state of the environment transits from s_t to s_{t+1} (where $s_t, s_{t+1} \in S$) with a probability $p(s_{t+1}|s_t, \mathbf{a}_t) \in \mathcal{T}$ and a reward $r_t^i = \mathcal{R}(s_{t+1}|s_t^i, a_t^i)$. A state s can be approximated by a *kth order history* approach which uses the last k observations and actions. In our case, $s_t = \langle \mathbf{o}_t, \mathbf{o}_{t-1}, \mathbf{a_{t-1}} \rangle$. This approach can manage any latent state information compared to using directly the latest observation as the state.

The goal of the agents is to find a deterministic joint policy $\mathfrak{ß} = \langle \pi^1, \dots \pi^n \rangle$ so as to maximize the sum of their individual reward $r_t = \sum_{i=1}^{n} r_t^i$. Agents act following their observation and no direct communication is needed. In this framework, agents only have access to their actions and do not observe each other's actions or observations. Hence, throughout execution, agents do not really have recourse to a "Markovian" feedback (Bernstein et al. 2002). This means that agents need to find an optimal way of summarizing their individual history. In other words, they need to store and reuse the sequence of their previous actions and observations.

4.4 METHODS

4.4.1 *Environment and Reward Scheme*

We used a well-known adversarial multi-agent domain, the *battle* or *combat game* described in (Sukhbaatar, Szlam, and Fergus 2016), in which two teams of agents are fighting against each other (Fig. 3.2). Agents of the same team cooperate with their teammates to find strategies to defeat the opponent team. The main goal is to kill as many opponents as possible by invading the neighboring territories and by using various warfare strategies. Our environment features two teams of n agents and runs from the initial positioning of the agents inside a square in two-dimensional space for each team. We should also mention that agents since they also have the same capabilities to sense the environment by direct perception, and have also the same speed and action space.

An agent can take up to seven actions: going up, going down, going left, going right, turning left, turning right, and shooting. An agent is

Table 4.1: Reward scheme of the adversarial environment.

Event	Reward
Step	-0.01
Death	-5
Kill opponent	$+5$
Attack empty position	-0.01
Attack opponent	$+1$
Attacked by opponent	-1

dead after being attacked (shot) three times. An agent can only attack the nearby opponents located one cell ahead of its current position. To simplify the environment dynamics, an agent is not able to attack an opponent located outside of its field of view. We also assumed that an agent cannot shoot any of its teammates. An *episode* ends when all agents on one team have been killed or after $1,000$ timesteps.

At the start of the first episode, each agent is randomly assigned a unique ID and keeps the same ID during the whole training. During each episode, the agents start at a random position inside the initially allocated area. The IDs of alive and dead agents are recorded to constitute the input of each team's network at any given timestep. The winning team is the one who has the highest number of alive agents.

The reward scheme is presented in detail in Table 4.1. Besides, an agent receives $+1$ for attacking an opponent successfully and -1 if it is being attacked. The global reward of each team is the sum of all the rewards received by agents of the same team at any timestep t, $r_t = \sum_{i=1}^{n} r_t^i$.

4.4.2 *Proposed Learning Framework*

The main environment is represented as a gridworld in which every agent uses a filtering method to delimit its local view (Fig. 4.1). The input of each network is a "tensorized" stack of all local views of agents at any given timestep. Each agent senses distances to neighboring agents within a radius of k from its current position. Each channel represents the local view o_t^i of each agent of the same team. Using only this information, a team can learn a policy that is able to establish and

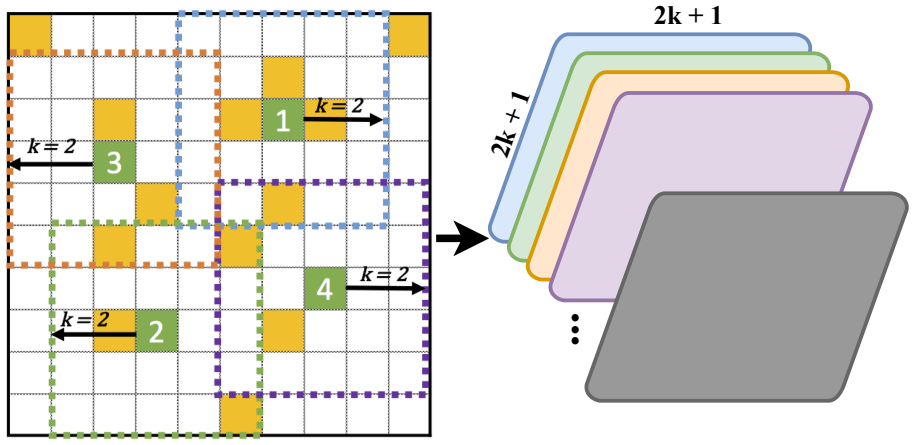


Figure 4.1: 3D tensor observation. Each channel represents the local view o_t^i of an agent, which is an image-like screen extraction in grayscale.

maintain a certain group formation and to cooperatively locate and kill most of its opponents.

This representation can be thought of as an aggregation of inputs from the different sensors of many robots. For instance, if we have many cleaning or patrolling robots, we could use this method to represent the input of their learning techniques. One advantage of this representation is that we can simulate, for example, the state of any robot, by including a probabilistic function that selects the availability of a robot state.

We propose a centralized training (Fig. 4.2) and decentralized testing (CTDT) framework for training adversarial multi-agent systems with a large number of agents. This framework can be described as a POMDP during the training phase and a dec-POMDP during the testing phase. In our framework, each agent has its own local view o_t^i, its own action space a_t^i, and its own reward r_t^i. Agents only know their actions and do not observe others' actions. At every timestep, one joint action $\mathbf{a}_t = \langle a^1, \ldots, a^n \rangle$ is taken and the environment moves to a new state (see Algorithm 2).

During the *training phase* in the CTDT, all members of the same team use a centralized neural network. The deep Q-Network of each team receives a "tensorized" observation at time t (see Figs. 4.3 and 4.4) with the IDs as shown in Fig. 4.2. Each agent of the same team jointly infers its action from the same network that outputs the Q-values of every possible action for all agents. After all agents have received

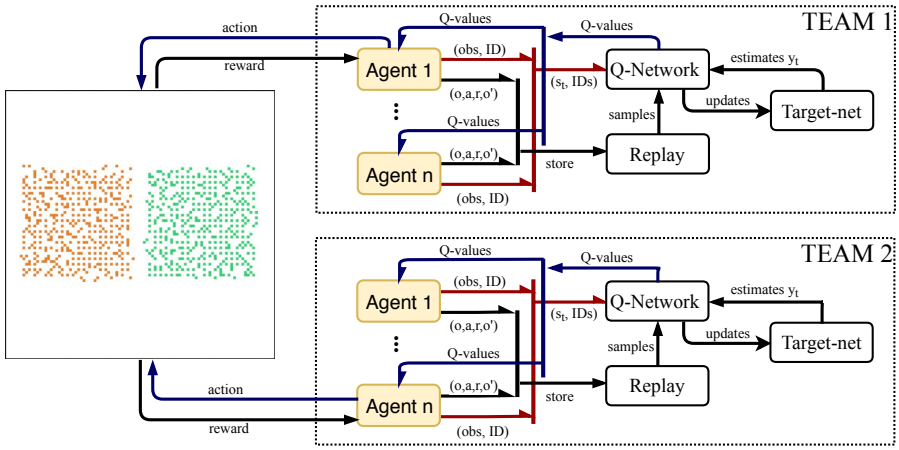


Figure 4.2: Scalable adversarial MAS architecture. The environment constitutes two teams of n agents ($n = 100, 200, 300, 400$) each at the beginning of each episode. During the training phase, agents use a centralized network

their actions, we combine them to constitute the synchronous joint action $\mathbf{a}_t = \langle a^i, \ldots, a^n \rangle$ and apply it to the environment. This implicitly embeds communication through the actions, local observations, and states. Because the network outputs all possible Q-values for each agent, we can apply different exploration strategies or learning techniques. For example, we can use a greedy exploitation strategy where the agents take the maximum Q-values, or by using ε-greedy (see section 4.4.5).

During the *test phase* of the CTDT, each agent acts based on its policy and local view and no additional communication is required. For this purpose, the agent copies the trained network into its local memory. Then, the last fully-connected layer of the network is removed and replaced by a layer with seven outputs (the number of actions). Meanwhile, the remaining parts of the network are considered as a fixed feature extractor. In other words, the weights of the centralized networks are "frozen" and only the weights of the output layer get updated. The network independently receives the view of the local agent with its ID and then it outputs an action for the agent. With this way of fine-tuning, we could train the network with n agents, and test it with a different number of agents per team, which greatly increases the ability of our policy to generalize and avoid over-fitting.

4.4.3 *Prioritized Experience Replay*

The learning process becomes smooth by storing an agent's experiences and uniformly sampling batches of them to train the network. This helps the network to learn about immediate actions while considering past experiences. We store the experiences as a tuple of $\langle s, \mathbf{a}, r, s_{t+1} \rangle$ (see Fig. 4.2). Instead of randomly sampling experiences, prioritized experience replay (Schaul et al. 2015) takes transitions that are not in alignment with our current network estimate. Furthermore, we store the residual error between the actual Q-value and the target Q-value of a transition. In fact, the TD error δ,

$$\delta = \left| y_t - Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right) \right|, \tag{4.1}$$

will be converted to a priority of the j-th experience τ_j using

$$\tau_j = \left(\delta + \mu\right)^{\lambda}, \tag{4.2}$$

where $0 \leq \lambda \leq 1$ determines how much prioritization is used and μ is a very small positive value. We then use the priority as a probability distribution for sampling an experience j (Schaul et al. 2015), in which j is selected by

$$\phi(j) = \frac{\tau_j}{\sum_k^K \tau_k}, \tag{4.3}$$

where $\sum_k^K \tau_k$ is the total of all priorities and K is the size of the mini-batch. This is called *proportional prioritization*. Each team stores the most recent experiences of all teammates in its own replay memory.

4.4.4 *Network Architectures*

Because we have two different teams fighting against each other, we cannot use one neural network to control both teams. In a small-scale problem, it would be easier to use a neural network for each agent. Unfortunately, having a network for each agent is not optimal in a large-scale environment in which we have up to 400 agents per team. We would also have to find an optimal and distributed way of reducing the communication delays between the environment and the agents. Therefore, we believe that having a centralized network per team is still a very complicated and challenging problem.

Algorithm 2: Multi-agent DQN / DDQN.

Initialize online network and target network with random weights $\boldsymbol{\theta} = \boldsymbol{\theta}^{-}$
Initialize experience replay memory $\mathcal{D}$
for *episode* = 1 **to** *num_episodes* **do**
  Populate the team with n agents
  for *timestep* t = 1 **to** *max_timesteps* **do**
    Extract the joint observation $\mathbf{o}_t = \langle o^1, \ldots, o^n \rangle$
    $\mathbf{a}_t \leftarrow \begin{cases} \text{Select random action } \mathbf{a}_t & \text{with prob } \varepsilon \\ \arg\max_{\mathbf{a}_t} Q(s_t, \mathbf{a}_t; \boldsymbol{\theta}) & \text{with prob } 1-\varepsilon \end{cases}$ Apply action $\mathbf{a}_t$ and observe reward r_t and next state s_{t+1}
    Store transition $\langle s_t, \mathbf{a}_t, r_t, s_{t+1} \rangle$ in $\mathcal{D}$
    Sample a mini-batch of m transitions $\langle s_j, \mathbf{a}_j, r_j, s_{j+1} \rangle$ from $\mathcal{D}$ by using the prioritized experience replay mechanism.
    if s_{j+1} *is terminal* **then**
      $y_j \leftarrow r_j$
    else
      $y_j \leftarrow r_j + \gamma Q\left(s_{t+1}, \arg\max_{\mathbf{a}} Q\left(s_{t+1}, \mathbf{a}; \boldsymbol{\theta}_t\right); \boldsymbol{\theta}_t^{-}\right)$
    end
    if $\left| y_j - Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right) \right| < \delta$ **then**
      $\mathcal{L}(\boldsymbol{\theta}_t) \leftarrow \frac{1}{m}\left(y_t - Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right)\right)^2$
    else
      $\mathcal{L}(\boldsymbol{\theta}_t) \leftarrow \delta\left| y_j - Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right) \right| - \frac{1}{m}\delta^2$
    end
    $\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) = \left(y_j - Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right)\right) \nabla_{\boldsymbol{\theta}} Q\left(s, \mathbf{a}; \boldsymbol{\theta}_t\right)$
    $\xi_t = (1-\gamma_1)\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t) + \gamma_1 \xi_{t-1}$
    $\varkappa_t = (1-\gamma_2)\left(\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t)\right)^2 + \gamma_2 \varkappa_{t-1}$
    $\hat{\xi}_t = \frac{\xi_t}{(1-(1-\gamma_1)^t)}$
    $\hat{\varkappa}_t = \frac{\varkappa_t}{(1-(1-\gamma_2)^t)}$
    $v_t = \eta \frac{\hat{\xi}}{\sqrt{\hat{\varkappa}_t + \epsilon}}$
    $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - v_t$
    Clear dead agents
    Decay the lives of agents who have been shot
  end
  Every τ step set $\boldsymbol{\theta}^{-} \leftarrow \boldsymbol{\theta}$
end

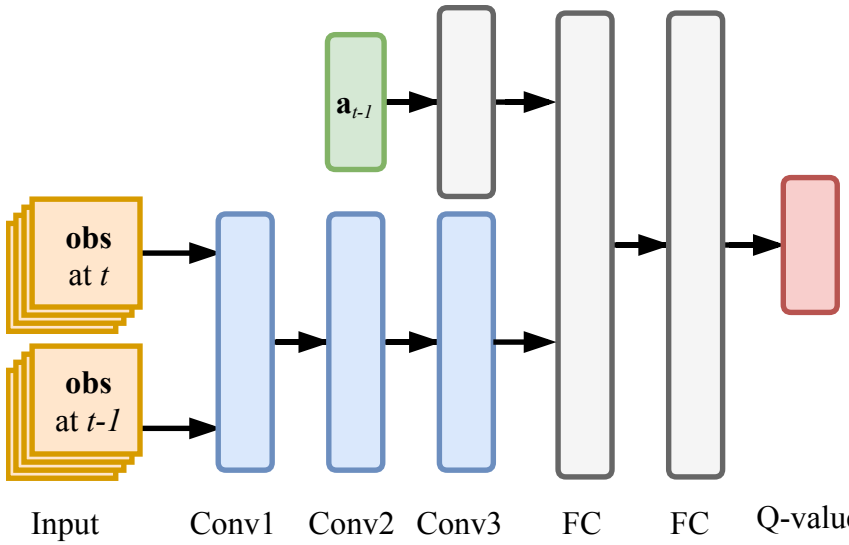


Figure 4.3: DQN/DDQN network architecture.

The network architectures used in our experiment are shown in Figs. 4.3 and 4.4. We used a stack of convolutional layers to consider regions of an image and to maintain spatial relationships between the pixels. The inputs are the observations at time t, $t-1$ and the actions at $t-1$. The input implicitly contains all of the relevant information about the controlled agents' situations, along with their speeds and directions. The output of our network is a joint action consisting of elements that are one of the seven actions specified in 4.4.1.

4.4.5 *Exploration*

Exploration vs. exploitation is one of the main dilemmas in RL. The optimal policy for an agent is to always select the action found to be most effective based on history (exploitation). On the other hand, to improve or learn a new policy, the agent must explore a new state it has never observed before (exploration) by taking non-optimal actions. We use ε-greedy (Sutton and Barto 1998) with decay to create a balance between exploration and exploitation with $0 \leq \varepsilon(t) \leq 1$. At every timestep, the agent acts randomly with probability ε and acts according to the current policy with probability $1 - \varepsilon(t)$. In practice, it is common to start with $\varepsilon = 1$ and to progressively decay ε as the training moves

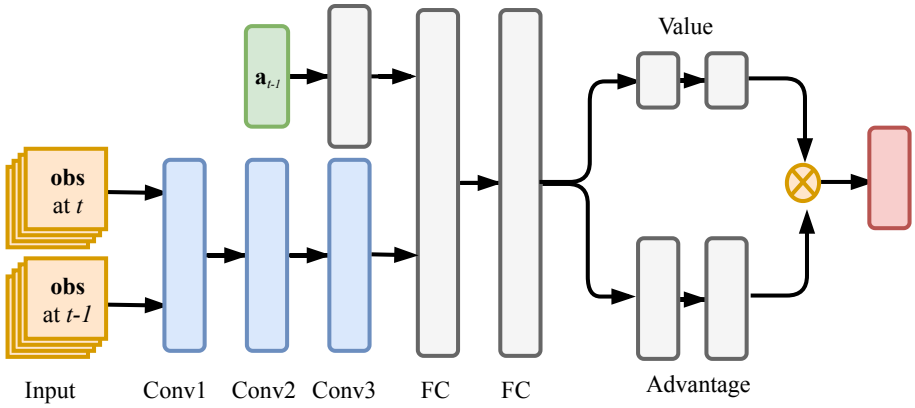


Figure 4.4: Dueling DQN/DDQN network architecture.

forward until we reach the desired ε. We could use a fixed ε value, but this is not always an optimal choice because, after many episodes of training, we would have a more accurate estimation of the policies and could reduce the amount of exploration.

We decay the values of ε based on the *piece-wise linear decay* function by dividing the training episodes into several subsets and then linearly decreasing the value of ε from the maximum to the minimum predefined ε value for each interval. However, this is rarely done in reality, as determining a decay function requires a tedious parameter search, and is extremely domain-dependent. The initial and final values of ε are, respectively, 1 and 0.01. In this case, our agents will be able to extensively explore during the first episodes and use the optimal policy without much exploration at the end of the training.

4.5 EXPERIMENTS AND DISCUSSION

We compare the results of the learned behaviors by the CTDT framework with those trained by using the centralized network reported in (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b). We also investigate the performance of strategies learned by different deep reinforcement learning techniques such as DDQN, DuelDQN, and DuelDDQN.

4.5.1 *Parameter Settings*

For all the experiments, we train the networks from scratch using the following parameters: learning rate $\alpha = 0.00025$, discount factor $\gamma = 0.99$, and an input of shape $(n, 2k+1, 2k+1)$. The loss function is optimized by using the Adam optimizer (Kingma and Ba 2014). We update the target network every $10,000$ timesteps. To stabilize learning, we feed the network with medium size mini-batches of 256 samples. First, we train each network against a copy of itself (self-play) during $5,000$ episodes by using the ε-greedy strategy. We then use the learned weights to test the performance of each network against DQN agents without ε-greedy exploration, which is the baseline of our evaluation. The experiments were run on a single machine with 4 Nvidia GTX 1080 Ti (12 GB per GPU) and a Xeon processor of 48 cores with 128 GB of RAM. The experimental results in the following sections describe the average values of fifty experimental runs with different random seeds during the test phase of the CTDT.

4.5.2 *Performance evaluation*

The environment is non-stationary because all agents are simultaneously learning. One way of evaluating the performance of each team is to check the convergence rate. In addition, it is always useful to compare other metrics such as the average reward per episode, the average Q-values, and the number of steps to finish an episode. Figure 4.5 describes the average reward per episode during the test phase for teams of n agents ($n = 100, \ldots, 400$). All techniques perform similarly for a smaller environment with 100 agents. We start noting discrepancies for $n > 100$. As we can see, DuelDDQN agents always achieve the highest reward during training and testing phases. Furthermore, if we omit the small negative reward for each step, the network will converge. This is simply a best practice for reducing the number of steps taken by agents to solve the tasks, and it often accelerates the convergence without any theoretical guarantee. It is highly likely that the learned behaviors would be the same.

Figure 4.6 shows the *ratio of alive agents* at the end of each episode. This ratio is calculated as $\varphi = v/\omega$, where v is the number of alive agents of the learning team at the end of each episode and ω is the number of alive DQN agents at the end of each episode. The value

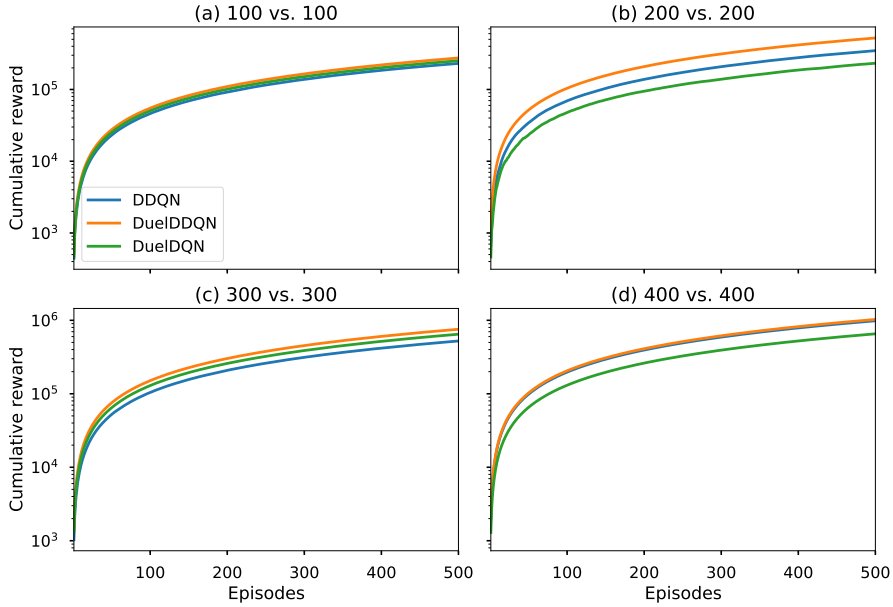


Figure 4.5: Cumulative reward against DQN after 5,000 training episodes.

of φ embeds how many opponents are killed and how much a team is protecting itself against the opponents. A lower ratio means that the team is taking too much risk and is only maximizing the reward without minimizing the number of steps taken. Notwithstanding the fact that all techniques have various ratios independent of the number of agents, all teams have learned good strategies for helping them win against DQN agents.

Figure 4.7 shows how many steps a team takes to finish each episode. Recall that an episode ends after 1,000 steps. As shown in the figure, DuelDDQN networks take fewer steps to finish episodes and always win against the DQN agents. From a practical perspective, this is a good sign because it shows that the DuelDDQN network convergence is faster and better compared to the results by using a centralized network (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b).

From Figs. 4.8, we can see that the best technique in this specific environment is DuelDDQN because it always achieves the highest rewards in a very short time compared to the other techniques. DDQN is the second-best method. This can be explained by the fact that they both implement a double Q-learning update, which tends to converge

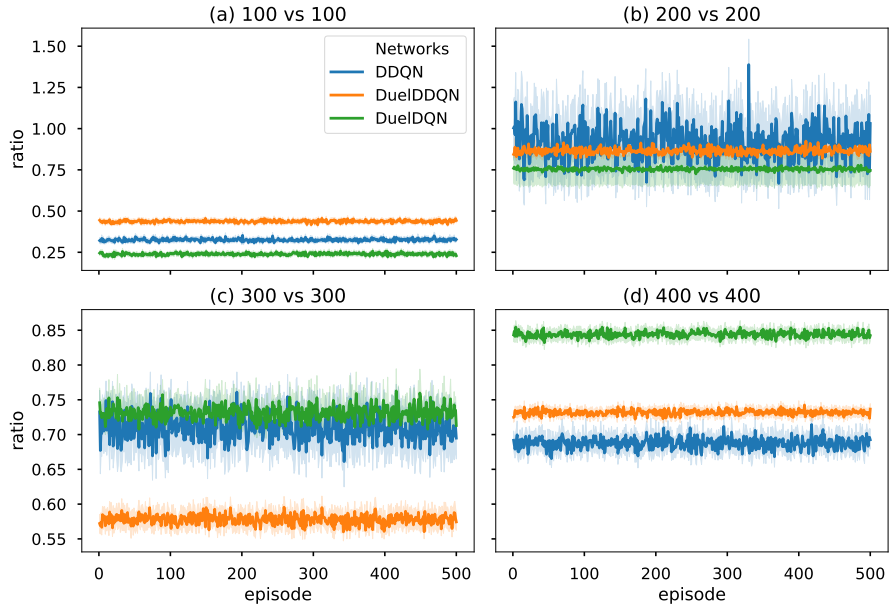


Figure 4.6: Ratio of alive agents (φ) after $5,000$ training episodes.

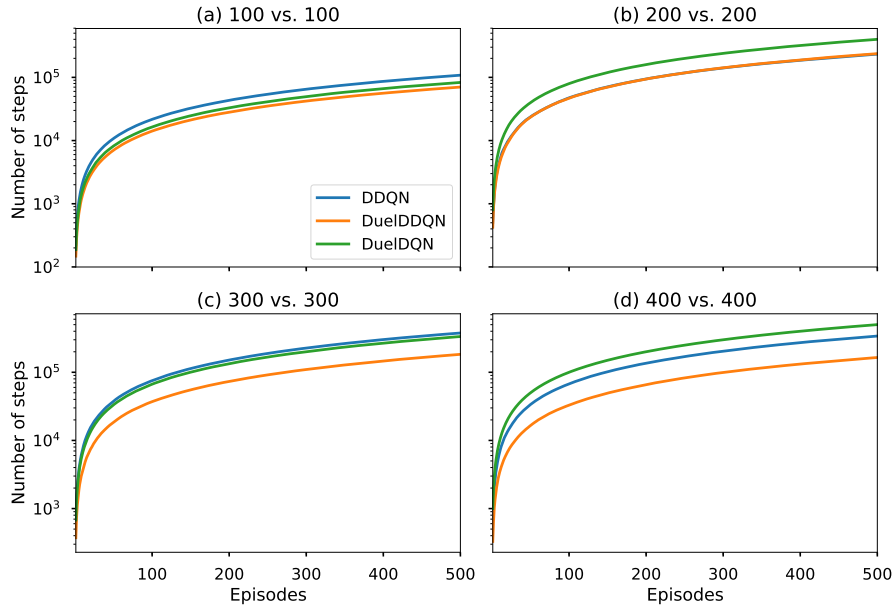


Figure 4.7: Cumulative number of steps against DQN after $5,000$ training episodes.

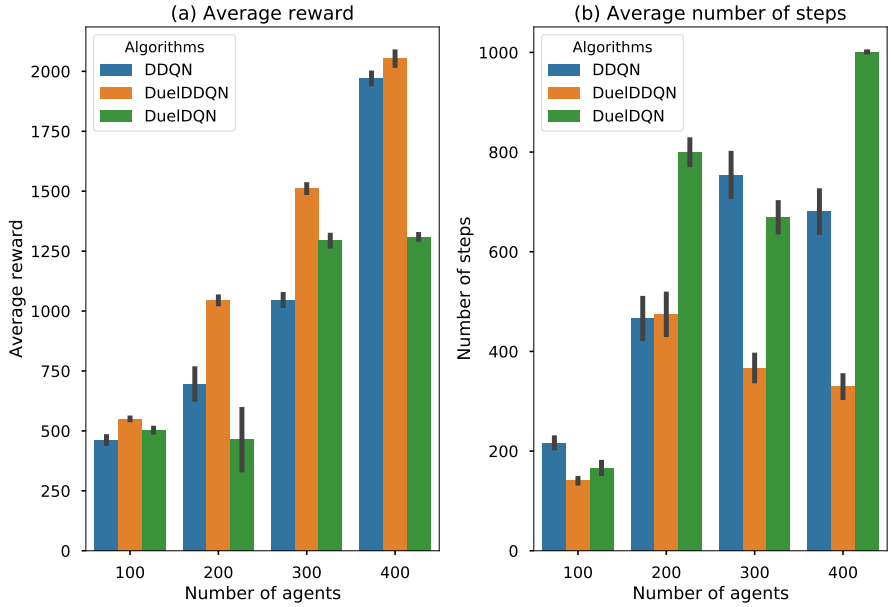


Figure 4.8: Average reward and number of steps after 5,000 training episodes.

faster to the right Q-values. Splitting the Q-values into advantage and value streams, in addition to the double Q-learning update, make the network robust and accelerate its convergence. Furthermore, as we can see from Tables 4.2 and 4.3, our proposed model outperforms the POMDP model proposed in which agents always use a centralized DQN (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b). This improvement stems from the fact that our agents use their history (s_{t-1} and $\mathbf{a}_{t-1}$) whenever they learn their actions. This helps them to better optimize their strategies by selecting the best actions and movements according to their prior history.

4.5.3 *Learned Strategies*

As we train agents, we might observe slightly variant tactics that have the same roots. For instance, it is not unusual to observe an improved attack and/or defense tactics appear again. First, the team with the largest number of agents at time t uses some tactics to eliminate as many opponents as possible in a very short time before the opponents' counter-attack. Then, both teams reach a kind of equilibrium in which

Table 4.2: Average rewards and standard deviations during the test phase, i.e. after 5,000 training episodes.

n agents	POMDP	DuelDQN	DDQN	DuelDDQN
100	332 ± 37	461 ± 14	503 ± 7	$\mathbf{550 \pm 4}$
200	548 ± 59	463 ± 25	695 ± 63	$\mathbf{1044 \pm 14}$
300	850 ± 97	1046 ± 23	1293 ± 22	$\mathbf{1510 \pm 16}$
400	1228 ± 129	1310 ± 10	1970 ± 23	$\mathbf{2052 \pm 28}$

Table 4.3: Average steps per episode and standard deviations during the test phase, i.e. after 5,000 training episodes.

n agents	POMDP	DuelDQN	DDQN	DuelDDQN
100	405 ± 89	166 ± 11	216 ± 10	$\mathbf{140 \pm 5}$
200	512 ± 57	800 ± 25	$\mathbf{466 \pm 40}$	474 ± 40
300	737 ± 94	670 ± 30	754 ± 43	$\mathbf{366 \pm 25}$
400	850 ± 83	1000 ± 0	680 ± 41	$\mathbf{329 \pm 22}$

they use similar group formation strategies. Finally, agents periodically update their strategies to defend themselves against their opponents. Agents often prefer short-term and aggressive strategies to long-term and safe ones. It seems that most of the strategies shown in Fig. 4.9 always appeared in all experiments.

We also note that all networks have approximately the same strategies as the number of agents increases. This is because agents are trained on such a large number of game-plays that they have seen nearly every scenario combination and have taken many possible actions, so they know which ones have the most value. Agents tend to avoid confrontation with a larger number of agents but seek and attack small groups while minimizing their losses (encirclement). This is a safe strategy in which agents are not risking themselves, while at the same time they gradually weaken the opponents. Unfortunately, this strategy sometimes proves to be suicidal when the team is outnumbered. In addition, it is interesting to observe that random agents are not always able to generate coordinated behaviors.

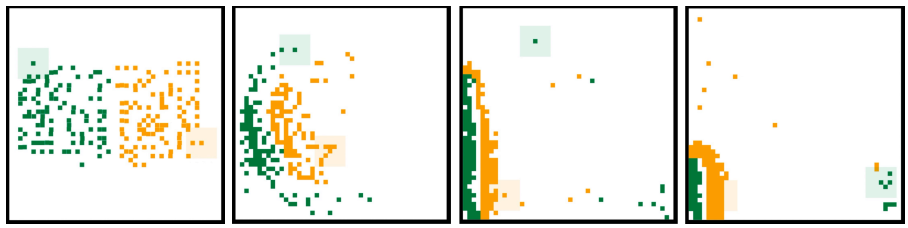


Figure 4.9: Example of an encirclement strategy.
See Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018a and Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b for a description of some strategies. A video is available at https://drive.google.com/file/d/19CAAh8tj9LKnFRq55C4QtLGxp0D6qJVS/view?usp=sharing.

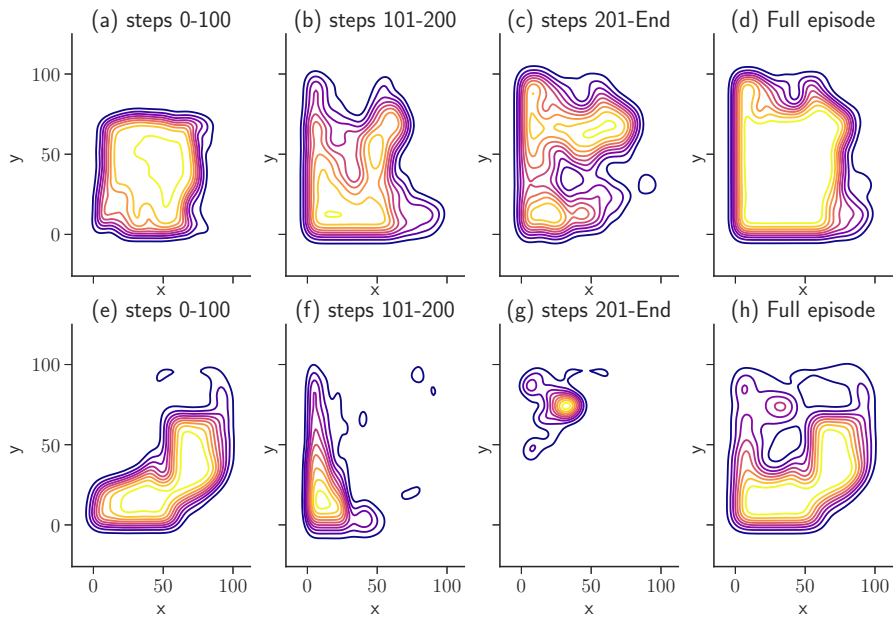


Figure 4.10: Kernel density estimation of the number of visits by team. DDQN, after 2,000 episodes, $n = 400$. Team 1: top row, team 2: bottom row.

5

MULTI-AGENT PATTERN FORMATION

This chapter contains material from (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2020b) and (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2020a).

5.1 INTRODUCTION

Suppose, for an instance, that an instructor needs her n students in the play area to form a 2D shape such as a circle so that, for example, they can play a game. The instructor may draw a guideline on the ground as a rule or even give every student a particular position to move to. Now, imagine a scenario where the instructor does not give such help. Indeed, even without such help, the kids may, in any case, have the option to form an adequately decent estimation of the circle if every one of them moves depending on the movement of others by directly observing their neighborhood region. If successful, this method can be called a distributed solution to the circle formation problem for children (Suzuki and Yamashita 1999).

By analogy, we utilized a methodology based on the previous example (Suzuki and Yamashita 1999) to control a large-scale multi-agent systems of homogeneous teams. The principal idea is to give every agent the opportunity to execute a straightforward estimation of its states and accordingly plan its actions depending on the actions and states of the remaining agents so that the agents as a team will cooperatively and collectively form the target pattern. This type of distributed control of cooperative systems, in which mobile agents work together to perform cooperative tasks, is challenging. The challenge is that such a system is expected to have the ability to self-organize itself by learning cooperative behaviours without any human intervention. It must be emphasized that, self-organization of a real-world (drones, cars, ...) multi-agent system is much harder to achieve as agents are required to learn more complex tasks.

Our framework could be applied to many realistic problems such as coordinated multi-robot exploration (Burgard et al. 2005), multi-robot

navigation (Balch and Hybinette 2000), shape constraints in crowd simulation (Gu and Deng 2011), and multi-robot animation for entertainment (Alonso-Mora et al. 2012). In general, the ability for the multi-agent team to cooperate appears to rely enormously upon (1) the characteristics of the problem to solve, (2) the global properties assigned to the team, and (3) the distinctive capabilities of each agent. Instances of the global properties are the capacity to recognize at any rate their team members, to concur on a common global direction (sense of direction), or to concede to a typical handedness (Dieudonné, Petit, and Villain 2010).

In this paper, we investigate how a large-scale system of independently learning agents can collectively form acceptable two-dimensional patterns (*pattern formation*) from any initial pattern and configuration. On our assumptions, an agent is to be regarded as a point in the plane that autonomously moves according to a given rule. In general, an agent observes the environment, computes its next position with a given algorithm, and moves to its next position until it finds its goal position. We assume that the agents are homogeneous and anonymous in the sense that they have no identifiers. We also assume that the agents are uniform in the sense that all agents synchronously execute a common algorithm. Each agent has no access to the global state and its actions are solely done in terms of its local observation (Yamauchi, Uehara, and Yamashita 2015).

This research proposes an end-to-end decentralized learning architecture in which agents (1) do not explicitly communicate; (2) use a centralized replay memory to share knowledge among agents who share the same global goal; and (3) use a centralized target network to take into account the dynamics of others and provide a quantitative estimate of how each agent perceives and is perceived by its team members. We call this method *Multi-Agent Pattern Formation with Deep Q-Network* (MAPF-DQN). The goal is to control the overall shape of a robot team by using only the local information provided by agents' sensors. Interestingly, the positions or goals of the individual agents in the group are not explicitly controlled. An agent should concurrently and independently learn to locate its goal position and consequently plan a smooth trajectory towards it. Our model is invariant to the number of agents per team. This makes it easier to transfer the learned behaviours of one team to another one with different a number of agents. We fur-

ther show that agents using MAPF-DQN can learn complex cooperative strategies in environments with progressively increasing complexities.

5.2 RELATED WORK

To date, the vast majority of current cooperative multi-agent approaches have been based either on centralized methods in which a lead agent predicts the behaviour of all agents or on a distributed approach in which the agents have a full view and understanding of their environment and dynamics or sometimes on completely independent learning systems that do not have anything in common (Gupta, Egorov, and Kochenderfer 2017). The easiest way to implement the decentralized method is to use Q-learning to estimate independent Q-value functions of each agent by considering them to be part of the integral environment. As others start to learn as well, the environment becomes non-stationary. So, this uncertainty limits such approaches from expanding to more than a few agents.

A handful of the latest works uses the centralized learning and decentralized execution paradigm (Lowe et al. 2017b; Hüttenrauch, Adrian, Neumann, et al. 2019). This consists of actor-critics methods in which the critic is centralized and uses all the available information from the environment during training. However, throughout execution, agents use their independent network to compute their actions in a completely distributed and independent fashion. For eample, MADDPG (Lowe et al. 2017b) uses the joint actions and states with the critic and establishes strategies for each actor by using DDPG (Lillicrap et al. 2015) and concatenating information from other agents.

Hüttenrauch, Adrian, Neumann, et al. (2019) argued that most of the recent multi-agent deep reinforcement learning algorithms are limited when they are applied to swarm systems without special care. One of their argument is that concatenating the information received from different agents is not optimal when you have an environment with a dynamic number of agents. They also argued that this could disregard the inherent permutation invariance of identical agents sine qua non to swarm systems and that doing so would hardly scale to large-scale systems. To remediate these limitations, Hüttenrauch, Adrian, Neumann, et al. (2019) proposed a method in which they treat the observation perceived from nearby agents as samples of a random

variable and then encode the current distribution of the agents by using mean feature embedding (Smola et al. 2007).

An alternative to learning systems would be optimization-based algorithms. However, to control the strategies of agents, optimization-based methods (J. Lin, Morse, and Anderson 2007; Jadbabaie, J. Lin, and Morse 2003) often oversimplify the model of agents by assuming unrealistic assumptions in well defined tasks or environments (J. Lin, Morse, and Anderson 2007; Ranjbar-Sahraei et al. 2012; Zhou et al. 2016) or having a full view of the environment (Zhou et al. 2016).

On the swarm optimization side, Xu et al. (2010) tackles the problem of homogeneous multi-agent pattern formation by using a natural swarm algorithm inspired by the particle swarm optimization and by using a virtual pheromone as the messaging protocols. This was proposed to limit the complexity of communication channels as the number of agents increases. In their method, the agents leave virtual pheromone in the environment. The pheromone is updated based on their local observations. However, agents still have to explicitly communicate by broadcasting the pheromone density to other agents located in their communication range.

Regardless, in many real-world applications, most of these hypotheses are infeasible or superfluous. As an example, when the number of agents is large, it then becomes apparent that most of the centralized methods will not scale because agents would have to share their data to the lead agent. Therefore, in case of agents with communication abilities, we cannot continually make sure that the communication channels are maintained up-to-date and not congested.

5.3 PRELIMINARIES

5.3.1 *Pattern formation problem*

Given a pattern $P = \{p_1, ..., p_n\}$, or a set of landmarks on an environment that is a two-dimensional grid, and $R = \{1, ..., n\}$ a set of n anonymous agents. Let $F(t) = \{f_1(t), ..., f_n(t)\}$ be the formed pattern by agents where $f_i(t) \in \mathbb{R}^2$ are the coordinates of the agent i at time t in the environment and $\mathbb{R}$ is the set of real numbers. The goal of the pattern formation problem is to find a near-optimal and decentralized algorithm to such an extent that from any initial distributions of the

agents positions, they will, in the long run, organize themselves to form the target shape.

In general, the initial configuration of the pattern in the environment is unknown to the agents. Instead, they have to explore the environment in its entirety in order to find the set of coordinates of the goal positions before moving towards them. However, they have no access to a global view of the environment. A consequence of the approach just outlined is that agents will eventually maximize their rate of success. We say that the agents successfully form a target pattern P from any initial position of agents $F(0)$ when their final positions $F(T)$ is 85% similar to the target pattern P, i.e. $(F(T) \approx P)$ where T is the terminal timestep of an episode. Note that each agent can only observe a subset of P located in its local view only.

5.3.2 *Model*

Most real-world problems can hardly be modeled as an (PO)MDP because the agents have limited communication and partial or noisy observations provided by their sensors. Clearly, they would have to learn cooperative and coordinated behaviours by using only their local information. For this reason, we use the dec-POMDP framework (Bernstein et al. 2002).

At every step, the environment transits from s_t to s_{t+1} with a probability $p(s_{t+1}|s_t, \mathbf{a}_t) \in \mathcal{T}$ when all agents execute a joint action $\mathbf{a}_t = \langle a_t^1, \dots a_t^n \rangle$. Then agent i receives a reward $r_t^i = \mathcal{R}(s_{t+1}|s_t, a_t^i)$. The observation o can be approximated by a *zth order history* approach which uses the last z observations and actions (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2018b). This is very important because the agents no longer have a Markovian signal; they can't neither observe the state nor estimate the belief b as in POMDPs. Therefore, this approach can manage any latent state information compared to using directly the current observation as the input of an agent. Note that doing so makes our agents *non-oblivious* because each agent r_i uses the same algorithm ψ and the past observations and actions of ψ. This gives the agents the ability to remember important information such as the positions of teammates and/or the target pattern.

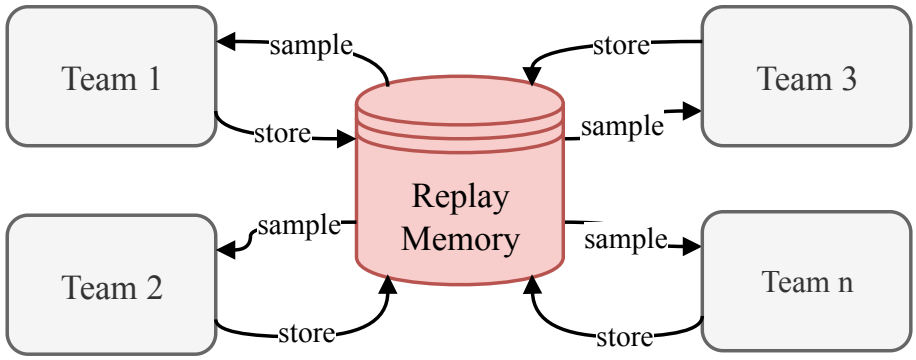


Figure 5.1: General architecture with one or more homogeneous teams.

5.4 METHODS

5.4.1 *General multi-team architecture*

We propose a decentralized system (Figs. 5.1 and 5.2) that can handle a dynamic number of homogeneous teams for the problem of multi-agent pattern formation. This is a concurrent team learning in which agents are divided into teams in the two dimensional space. It is well-suited to cooperative multi-agent systems in which a team needs to have information about others in order to make their decisions, i.e., every team learns to improve parts of the global team by sharing its experiences with others and reusing others' experiences at the same time (Fig. 5.1). Each agent has a limited visible visual field of depth k (shape $= [2k+1, 2k+1]$) in the two dimensional space and an agent can observe its teammates, the obstacles and the walls within its neighborhood area. However, the interactions between the agents are not explicitly modelled; they instead have to learn them by observing other agents' movements and reusing others' experiences which are randomly sampled from the centralized experience replay memory.

5.4.2 *Distributed architecture of a team*

The architecture of a team is shown in Fig. 5.2. Each agent concurrently and independently learns its behaviour based only on its local observations (Fig. 5.3). In this framework, every agent has its own neural network which is shown in Fig. 5.3. By using a centralized target

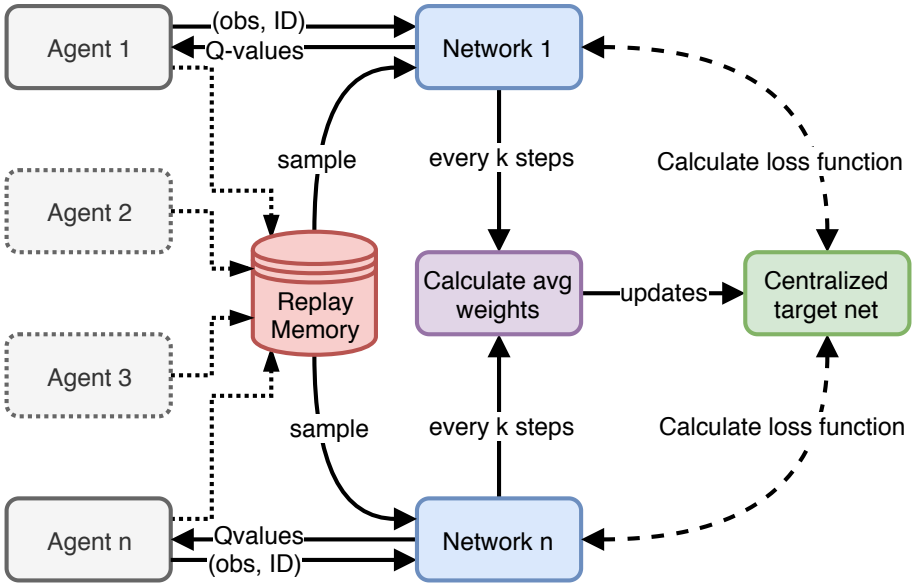


Figure 5.2: Distributed model with centralized replay memory and averaged target network.

network and replay memory, we naively mimic the team modeling framework by learning about other agents in the environment so as to make good estimates of their actions. By doing so, we alter the dynamics such that every agent can perceive and co-adapt to the dynamics of the environment. This also helps them estimate the current policies of others and thus, achieve a better cooperation (Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2020a). This is equivalent to providing to an agent the internal belief representation required to cooperate with its teammate and potentially avoid conflicts.

The goal positions of individual agents are not defined and must be instead collaboratively learned by agents. By doing that, agents learn to agree on their tasks or assignments in a coordinated manner. We have already seen that the only way they learn to coordinate is restricted to using only their local observation which might contains the positions of other agents and goal positions.

With our method the network continuously adapts and updates the goals depending on the environment dynamics until all agents reach their goals. An agent receives a reward of $+10$ if it finds a landmark and decide to commit to it and receives a punishment of -0.05 at every time step. The last part should potentially demotivate them to spend

long time without finding goals. The key components of this framework are: *domain randomization*, *centralized experience replay memory*, and a *common target network* calculated by the average of each agent's weights.

5.4.3 *Domain randomization*

It is well known that transfer learning is hard with reinforcement learning policies. To avoid this trap, we use some ideas from robotics domain randomization. The goal of domain randomization is to train the agents on different environments with random properties and dynamics. In our case, the agents are trained to form themselves into randomly generated shapes. In other words, the positions of the target shapes are randomly generated and scattered all over the environment at the beginning of every episode with a variant number of targets up to 600 simulated agents during training. This will help the system to adapt to different shape during the test phase as they were trained on so many different and variable shapess. Hence, the agents will learn to generalize well to unseen pattern shapes.

5.4.4 *Centralized experience replay memory and common target network*

Each agent of the team has its own main network to behave autonomously. It stores and updates its representation of the environment by randomly sampling from the replay memory. As a consequence, each agent independently computes its patrolling plan by taking advantage of other agents knowledge without exchanging coordination messages. It is important to combine this approach with a good sampling strategy because the existence of multiple agents accessing and updating a centralized memory may result in earlier memories of some agents to be overwritten.

The centralized target network is updated by the average weights of individual agents' networks. In other words, the target network $\boldsymbol{\theta}^{-}$ is a single network that combines the individual agents' networks (θ) of the same team by taking the average of their weights.

$$\boldsymbol{\theta}^{-} = \frac{1}{n} \sum_{i=1}^{n} \theta^{i}. \tag{5.1}$$

The new target value becomes

$$y_t = R_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta); \boldsymbol{\theta}^-). \tag{5.2}$$

By doing so, we also ensure that an agent can react to the previous actions and rewards of others by taking into account their dynamics. As a result, this somehow provides some sort of communication to cope with the local view. Moreover, this framework seems to provide the same characteristics of a team with implicit communication and global view. This is still better than most actor-critic methods such as MADDPG (Lowe et al. 2017b) in which the central critic limits the ability of agents to generalize their learned behaviours to different environments or even the same environment with different number of agents or action space. The agents are homogeneous and anonymous, so they cannot be distinguished by their appearance. Besides, our framework works in a completely distributed mode and agents have no preference for their goal destinations.

5.4.5 *Observation and network architecture of an agent*

With this approach, agents rely on themselves to make decisions instead of a centralized leader. However, without centralized coordination, our framework works in a complete distributed mode, using only the local environmental information provided by the agents' sensors. Agents do not have a global knowledge of their environment and should achieve good results with smooth motions in a relatively short time. The observation of an agent (Fig. 5.3) is encoded as a multi-channel image in which every channel represents a different feature from its view as in (Zheng et al. 2018; Sunehag et al. 2018; Elhadji Amadou Oury Diallo and Toshiharu Sugawara 2019).

Every channel is a binary matrix with 1 indicating the presence of an object and 0 otherwise. The first channel represents positions of the wall and obstacle. The second one represent the position of the agent itself, and the third channel is the position of the goals in that local view. If we have more than one team, we add a new channel for each different team. Finally, the last channel contains all other information such as the goal positions of all other teams. This representation could be pushed further by using ideas from neural-symbolic computing (Garcez et al. 2019) that provides the ability to learn from experiences and at the same time the ability to reason from what has been learned.

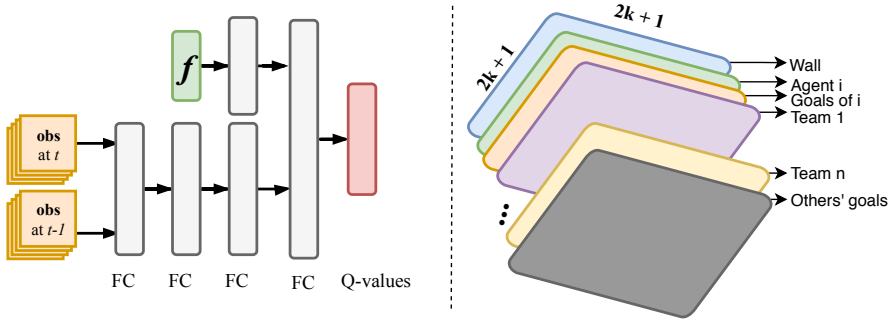


Figure 5.3: Left: Network architecture. Right: spatial observation of an agent.

The neural network architecture of each agent is shown in Fig. 5.3. It consists of fully connected layers only. The inputs are the spatial observations of agents (Fig. 5.3) at t and $t-1$ and a feature vector $\boldsymbol{f}$ from the environment containing the last action, last reward, relative position, number of goal points in their local views. This is very similar to the state representation in (Zheng et al. 2018) with the only difference being that we reused the history to make it conform to our dec-POMDP model.

5.5 EXPERIMENTS AND DISCUSSION

5.5.1 *Experimental settings*

The following results are average of 10 experimental runs with different random seeds by using our proposed framework. We assume that there is no noise in agents' observations and that the target shape is static during every episode. In all experiments, the target network is updated after every $10,000$ steps which also means that the average target network is calculated at the same time. We set the learning rate to 0.0001, $\gamma = 0.9$, batch size of 128 samples, and used ε-greedy as the exploration strategies. The values of ε are linearly decayed from 0.5 to 0.01 during the first 5000 of the 10000 training episodes. The maximum samples that the shared experience replay memory can have at a time is $1,000,000$.

We use a centralized system as a baseline, in which a team's strategy is computed by a central agent and subsequently communicated to all teammates. We also compare our method against a discrete action space

version of MADDPG (Lowe et al. 2017b). To improve the effectiveness of our method and its ability to generalize on completely random and unseen environment, the agents were trained on a large 2-dimensional grid graph (150×150) with utmost 700 randomly generated landmarks at the beginning of each episode.

5.5.2 *Results with one team*

First, we trained our model with only one team of utmost 700 agents during training and a varying number of agents during testing. Figure 5.4 shows the average reward and completion rate of agents with different visual field depths. We do not provide the result of the centralized method when $k = 1$ as that would generate an observation range of 3×3 which would be too small for a convolutional layer with a 3×3 kernel. With a smaller view range ($k = 1$), MADDPG achieved the highest reward, followed by the centralized network while agents using our method struggle to learn how to organize themselves into the target shapes. As the view range increases ($k \geq 2$), the proposed method steadily improves while the performance of MADDPG becomes less stable. The instability of MADDPG is due to the exchange of observation during training. Moreover, the centralized framework is not stable either for a smaller view ranges. This is probably caused by the limited scope of the information from their local observations. In summary, with a large enough view range, agents can solve their tasks by using either our proposed method or a centralized system with a lead agent.

While our method has a similar average success rate as the centralized method for $k \geq 4$, it also achieved the highest rewards among all of the tested methods. This means that agents using MAPF-DQN take less time to learn acceptable strategies. Surprisingly, we observe in all methods, agents do prefer to finish most of their tasks during the first hundred steps before slowly trying to complete and improve the global shape (Fig. 5.9). In addition, MADDPG agents trained on randomly generated patterns cannot generalize well on unseen and structured patterns when the number of agents is large as in Fig 5.9.

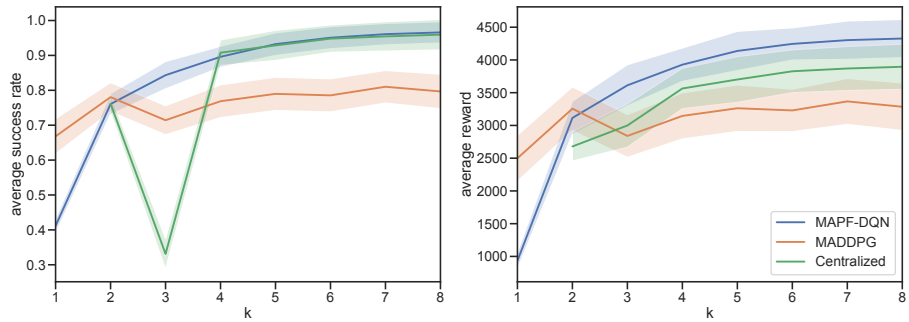


Figure 5.4: Left: average success rate. Right: average reward.

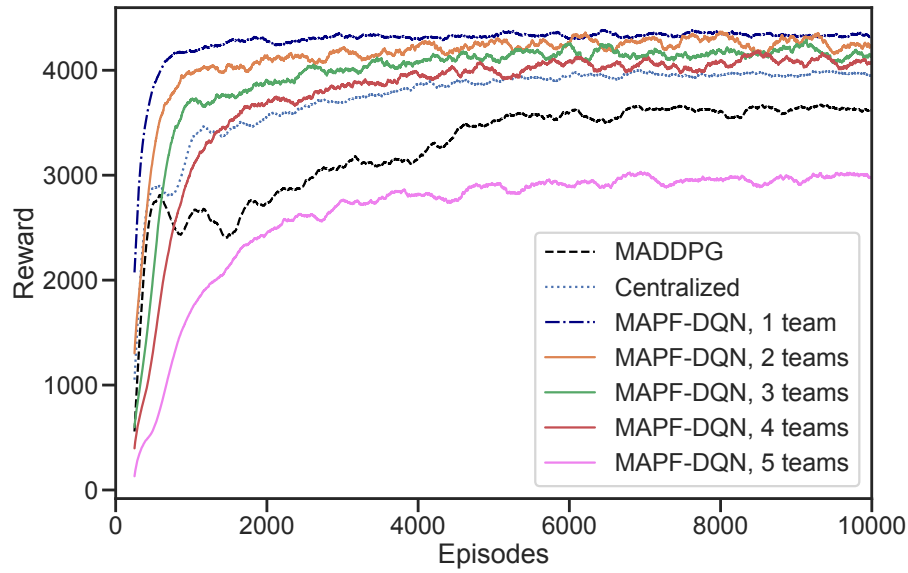


Figure 5.5: Multi-team average reward.

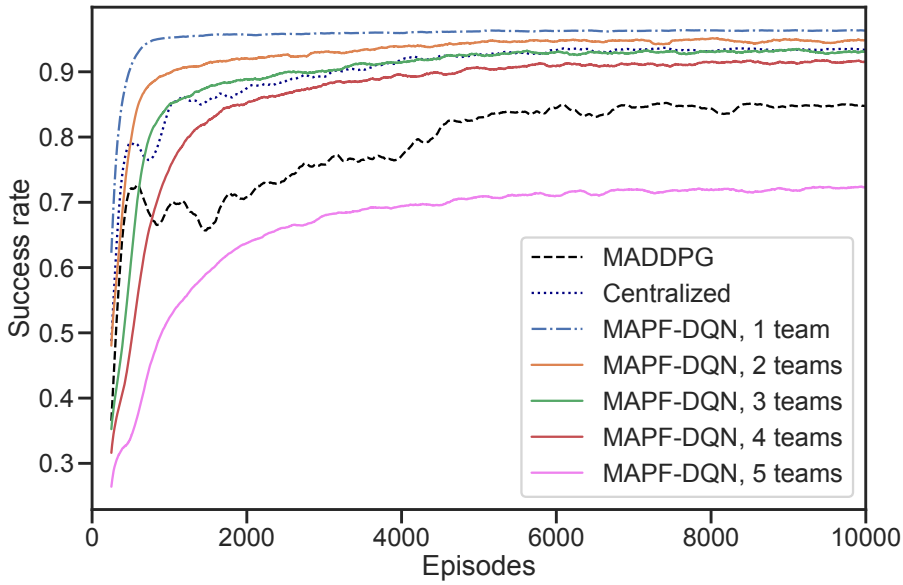


Figure 5.6: Multi-team average success rate.

5.5.3 *Results with many teams*

To evaluate the robustness of our method, we train it with a different number of homogeneous teams. The advantage of a multi-team system is that we could eventually use MAPF-DQN with heterogeneous teams with different action spaces, velocities, and learning techniques. The number of agents per team is equal to the number of landmarks divided by the number of teams. Figures 5.6 and 5.5 show that our method with 1 to 4 teams do also outperform all other methods during training even though its performance keeps decreasing as the number of teams increases. Surprisingly, we observe a sudden performance drop when the number of teams is 5, which becomes even worse than MADDPG(Lowe et al. 2017b).

Figure 5.7 shows the distribution of rewards for our framework in environment with different number of teams. Our method is more confident when we just have one group as the dimension of the observation is smaller and contains less noise as can be seen in the reward density estimation. The confidence is slightly shrinking as the number of teams increases and the distribution of the rewards starts varying in a larger range. Finally, our method becomes less confident about

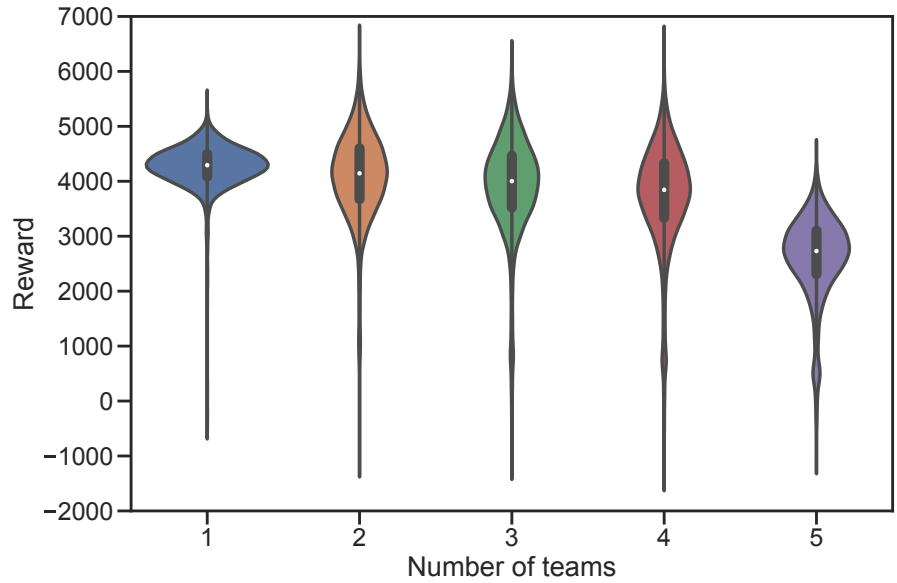


Figure 5.7: Distribution of rewards.

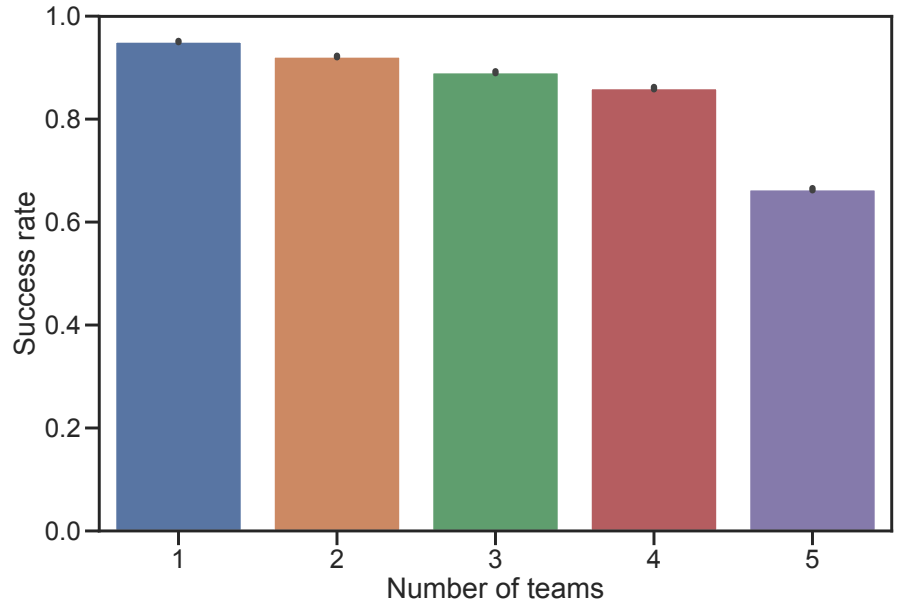


Figure 5.8: Multi-team average success rate for $k = 6$.

its strategies as the number of teams becomes larger. It also shows that our method works better with smaller number of teams (up to 4) and does not scale for teams of more than four teams. Therefore, the behaviours become less predictable for a team of five teams. Though, this is not surprising because we know the dimension of the observation of an agent increases as the number of teams increases. This make the dynamics of the environment noisy and thus, very hard to predict. In addition, Figure 5.8 shows that our method is robust up to 4 teams, from which it becomes less stable and confident about its predictions.

5.5.4 *Generalization*

After evaluating the behaviours of our proposed method with different team sizes with different number of agents, and different observation range without any fine-tuning, our results suggest that the proposed framework achieves zero-shot generalization on all environments independently of the depth of view of agents. Figure 5.9 shows 550 agents trying to organize themselves into an X shape in a 150×150 grid-graph. At the end of this episode, our proposed method achieves the highest completion rate of 94%, followed by the centralized architecture with 87% and finally MADDPG (Lowe et al. 2017b) with 73%. Agents tend to finish the large chunk of their tasks during the first few steps. However, agents with different techniques have different behaviors of representing the shape. For example, with our method, agents tend to start from the center of the shape and progressively explore the rest of the environment (Fig. 5.9), while with MADDPG (Lowe et al. 2017b), the agents, do not really have any preferences, they start exploring the environment in its entirety.

This difference can justify why MADDPG (Lowe et al. 2017b) agents do not achieve a good result at the end in large-scale environment. The other reason is that MADDPG (Lowe et al. 2017b) hardly scale in large environment because agents have to share their information, i.e., the more agents you have, the higher the complexity of the observation of an agent is. In contrast, our agents hardly explore the whole environment by starting from the center, which often a good behaviour, but you could easily see the limitations when you have a disconnected or discontinue shape. However, we still think that the behaviour of MADDPG (Lowe et al. 2017b) could be well-suited for small-scale environment with less agents and simple target shapes.

Figure 5.10 shows n agents trying to organize themselves into different shapes in a 150×150 grid-graph by using MAPF-DQN. These shapes consist of an "O" shape, a dolphin, a house, and a mandala. As in Fig 5.9, the agents finish the bulk of the tasks during the very first steps of an episode and then subsequently try to improve their coverage. Even though all teams have utmost 700 agents during training, we can see that they can more or less generalize well in environment with much larger number of agents during the test phase. We can also see that the generalization is somewhat independent to the number of agents but highly tied to the pattern themselves. For example, the dolphin requires only 864 agents and mandala 1669, but our method has a better result with the later.

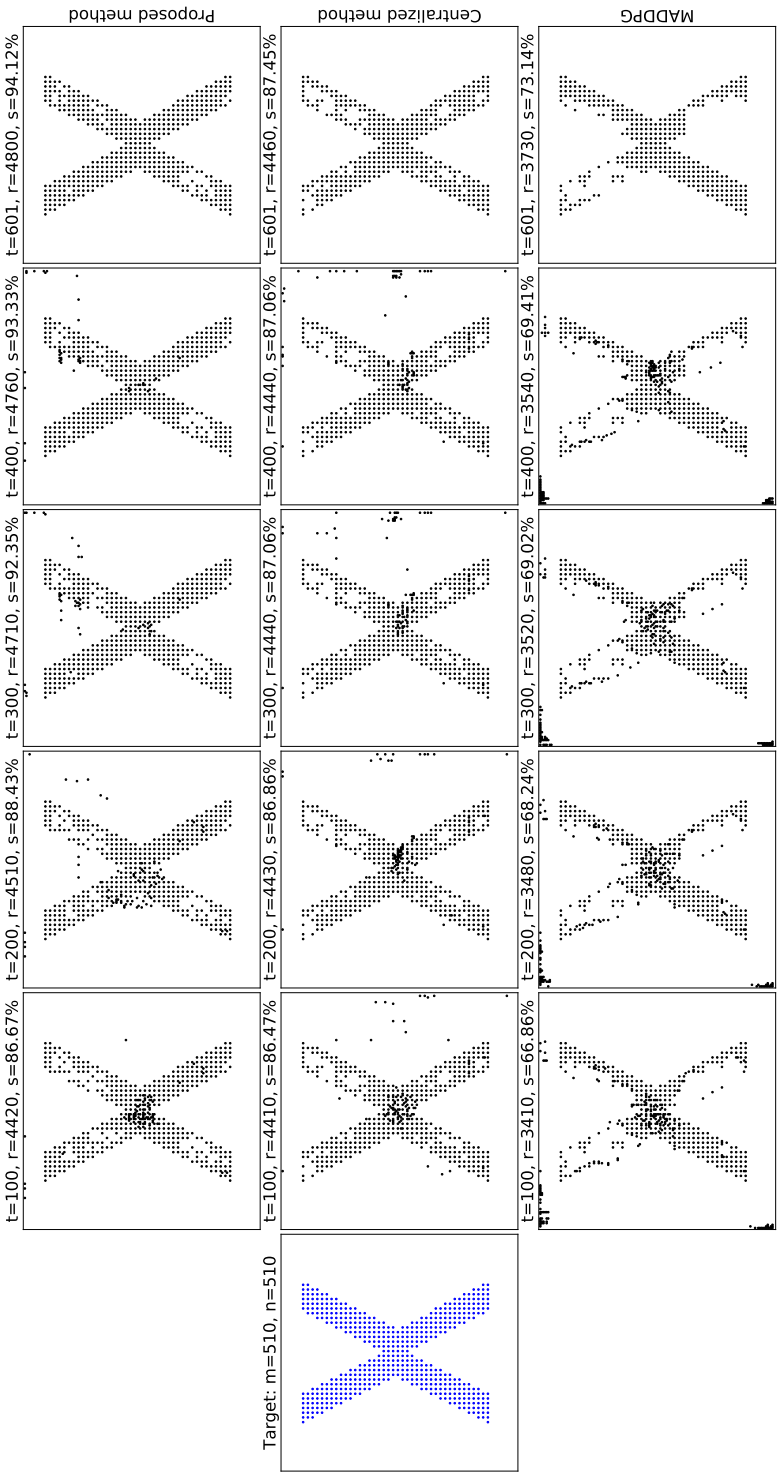


Figure 5.9: 510 agents try to organize themselves into an X shape in a 150×150 grid-graph.

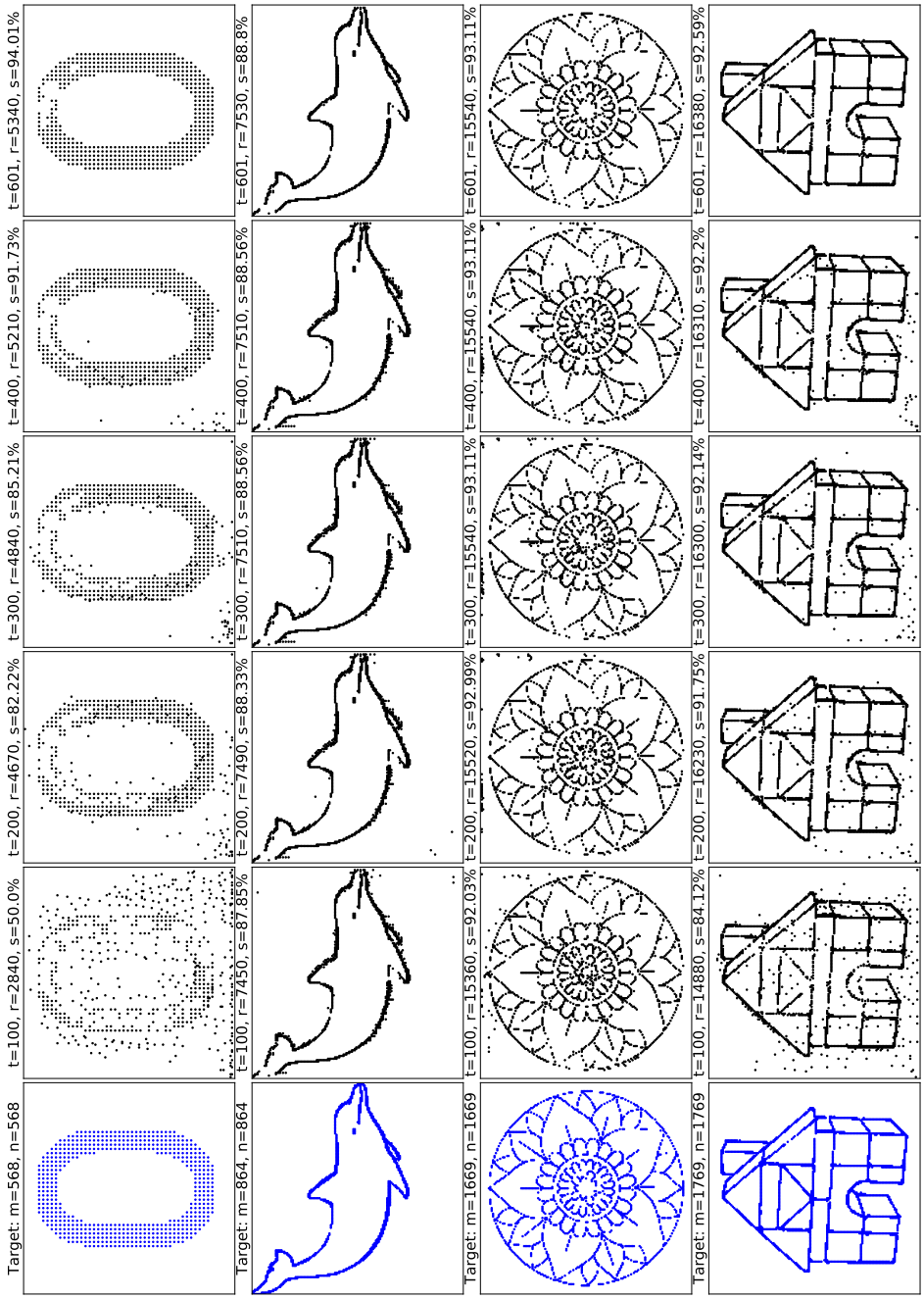


Figure 5.10: n agents trying to organize themselves into different shapes in a 150×150 grid-graph.

6

SOME LESSONS LEARNED

In this chapter, I will try to put in writing some of the most important lessons learned throughout some of the experiments during the execution of this thesis. It is an attempt to explain how the findings of this thesis could be used in real-world applications and what could potentially limit a wide adoption of multi-agent deep reinforcement techniques.

6.1 IMPORTANCE OF EXPERIENCE REPLAY

It is well known that one could research deep reinforcement learning without the need to use an experience replay buffer. However, it has been shown that using an experience replay memory is generally important. This is the place where you put additional information that will help you rank the importance of an agent's experiences. It is worth mentioning that any probabilistic sampling methods but random sampling appears to be better. This could be calculating the drift between the predicted q-values and the actual ones and consequently use that information to prioritize how the examples from the replay memory are sampled.

6.2 ROBUSTNESS OF CENTRALIZED LEARNING WITH DECENTRALIZED EXECUTION

Almost any real-world environment could potentially match this framework. It could be argued that applying multi-agent actor-critic naively performs poorly even in simplistic conditions. One advantage of this paradigm is the ability to use any further information needed during the learning phase and subsequently removed the extra information during the execution phase.

6.3 ADVANTAGES OF PARAMETER SHARING

The basic concept consists of training a single network wherein the network's weights are shared by agents. Indeed, this could easily lead to all agents have nearly identical policies. Even though the policies are similar, they are different since the observations of agents are different. However, this could easily lead to overfitting whenever agents are near each other, they might have the same observation, thus the same policy, and consequently, bog down in local optima.

6.4 LIMITATIONS OF MEMORY CAPABILITY

Traditional reinforcement learning algorithms cannot memorize useful information from the environment. This could be problematic and the system cannot improve the long-term credit assignment issue. On one hand, modeling the environment as a Dec-POMDP model could help reduce the problem because dec-POMDP reuses part of an agent's history when they compute their new actions. A sweet spot would be using Dec-POMDP in combination with LSTM as network structures for sequential decision making.

6.5 DANGER OF OVERFITTING

Whenever agents are trained against rule-based agents in a competition scenario, they have been shown to overfit by learning policing countering the rules of their opponents. However, when those rules are slightly changed during execution, the performance of agents drops drastically. There are many ways to counter this problem. You could implement domain randomization by training agents on different environments with random properties and dynamics. Or, the agents could be trained to compete against simultaneously learning agents through *self-play* or by implementing different learning algorithms.

6.6 UNPREDICTABILITY AND SAFETY ISSUES

Among all of the problems of multi-agent deep reinforcement, the unpredictability and safety issues are the main limit to the wide use of these techniques in real-world applications. You would always

expect the agents to select the most optimal policies after training, however, there is neither a theoretical guarantee nor a practical one that ascertains that agents would behave as predicted. From time to time, you can observe agents forget everything they have learned from the environment dynamics by just over-training them (training them over a longer time horizon) or by introducing a simple and negligible random event. One solution could be to model the environment with constraints by incorporating some safeguards. An example of such a constrained model would be constraint-MDP.

Also, it could be interesting to theoretically study, or at least investigate a theory of equilibrium in large-scale MADRL. The results would show us when it is opportune to stop the training phase and if there is an equilibrium from which there are theoretical guarantees that the system will act safely according to the principles of the designer.

There are many other lessons. However, it is worth mentioning that some or most of these shortcomings are being mitigated right now.

7 CONCLUSION

In chapter 3, we demonstrated that agents keep forming groups whenever new ones are necessary. Also, the formations are dissolved whenever it is beneficial to do so. The networks have learned how to effectively position and move agents for the emergence of group formation during and after training. This proves that grouping provides greater protection against opponents. We confirmed that the local behavior of an individual could conspire to determine very complex global behaviors of multi-agent systems. While we believe that our results show how agents optimally choose their strategies to form alliances, we also recognize that further investigations are needed. These include investigating what will happen if we have obstacles inside the environment and if agents don't have the same speed or field of view.

In chapter 4, we demonstrated that the networks could generate strategic group formations in a non-stationary and adversarial environment. We also showed that our models outperformed the POMDP model in all our experimental settings. This demonstrates the importance of reusing previous actions and states to infer new actions. Moreover, combining DDQN and DuelDQN generated better strategies. We conclude that our results show how agents optimally choose their strategies to form alliances, but we also recognize that further investigation is needed. Such investigations would include examining what would happen if we have obstacles inside the environment and if agents do not have the same speed or fields of view. Future research could further study the learned strategies by describing how they are modeled, or how they can be represented as output from the system. It would be helpful to do this, rather than simply implying that some strategies must exist since we see them via direct observation.

In chapter 5, we showed that agents using our method can organize themselves into a complex 2-dimensional pattern even though they were trained on random patterns. Our results showed that the proposed framework achieved zero-shot generalization on unseen environments without retraining the agents independently of the depth of their views. We finally showed that our framework could generate complex strate-

gies when the team is divided into independent homogeneous groups. However, our method does not scale when we have a large number of groups or teams. Finally, it would be interesting to investigate our method for multi-pattern formations in which agents are expected to achieve smooth transitions between given patterns. Also, it would be interesting to explore our framework with heterogeneous instead of homogeneous groups and further investigate the robustness of this framework.

BIBLIOGRAPHY

Aguilar, José, Mariela Cerrada, Gloria Mousalli, Franklin Rivas, and Francisco Hidrobo (2005). „A multiagent model for intelligent distributed control systems." In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer, pp. 191–197 (cit. on p. 1).

Alonso-Mora, Javier, Andreas Breitenmoser, Martin Rufli, Roland Siegwart, and Paul Beardsley (2012). „Image and animation display with multiple mobile robots." In: *The International Journal of Robotics Research* 31.6, pp. 753–773 (cit. on p. 52).

Balch, Tucker and Ronald C Arkin (1998). „Behavior-based formation control for multirobot teams." In: *IEEE transactions on robotics and automation* 14.6, pp. 926–939 (cit. on p. 14).

Balch, Tucker and Maria Hybinette (2000). „Social potentials for scalable multi-robot formations." In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 1. IEEE, pp. 73–80 (cit. on p. 52).

Barfoot, Tim D and Christopher M Clark (2004). „Motion planning for formations of mobile robots." In: *Robotics and Autonomous Systems* 46.2, pp. 65–78 (cit. on pp. 14, 33).

Bernstein, Daniel S, Robert Givan, Neil Immerman, and Shlomo Zilberstein (2002). „The complexity of decentralized control of Markov decision processes." In: *Mathematics of operations research* 27.4, pp. 819–840 (cit. on pp. 33, 35, 36, 55).

Boutilier, Craig, Yoav Shoham, and Michael P Wellman (1997). „Economic principles of multi-agent systems." In: *Artificial Intelligence* 94.1-2, pp. 1–6 (cit. on p. 1).

Boutsioukis, Georgios, Ioannis Partalas, and Ioannis P. Vlahavas (2011). „Transfer Learning in Multi-Agent Reinforcement Learning Domains." In: *EWRL* (cit. on p. 7).

Burgard, Wolfram, Mark Moors, Cyrill Stachniss, and Frank E Schneider (2005). „Coordinated multi-robot exploration." In: *IEEE Transactions on robotics* 21.3, pp. 376–386 (cit. on p. 51).

Buşoniu, Lucian, Robert Babuška, and Bart De Schutter (2010). „Multi-agent reinforcement learning: An overview." In: *Innovations in multi-agent systems and applications-1*. Springer, pp. 183–221 (cit. on pp. 1, 3).

Choi, Jongeun, Songhwai Oh, and Roberto Horowitz (2009). „Distributed learning and cooperative control for multi-agent systems." In: *Automatica* 45.12, pp. 2802–2814 (cit. on p. 2).

Claus, Caroline and Craig Boutilier (1998). „The dynamics of reinforcement learning in cooperative multiagent systems." In: *AAAI/IAAI* 1998, pp. 746–752 (cit. on p. 1).

Desai, J. P., J. Ostrowski, and V. Kumar (May 1998). „Controlling formations of multiple mobile robots." In: *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*. Vol. 4, 2864–2869 vol.4 (cit. on pp. 2, 14).

Desai, J. P., J. P. Ostrowski, and V. Kumar (Dec. 2001). „Modeling and control of formations of nonholonomic mobile robots." In: *IEEE Transactions on Robotics and Automation* 17.6, pp. 905–908 (cit. on pp. 14, 33).

Desai, Jaydev P (1998). „Motion planning and control of cooperative robotic systems." PhD thesis (cit. on p. 14).

Diallo, Elhadji A. O., A. Sugiyama, and T. Sugawara (Dec. 2017). „Learning to Coordinate with Deep Reinforcement Learning in Doubles Pong Game." In: *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 14–19 (cit. on pp. 15, 85).

Diallo, Elhadji Amadou Oury and Toshiharu Sugawara (2018a). „Learning coordination in adversarial multi-agent dqn with dec-pomdps." In: *workshop on Reinforcement Learning under Partial Observability, 32nd Neurips* (cit. on pp. 6, 13, 49, 86).

Diallo, Elhadji Amadou Oury and Toshiharu Sugawara (2018b). „Learning strategic group formation for coordinated behavior in adversarial multi-agent with double DQN." In: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, pp. 458–466 (cit. on pp. 6, 13, 33–35, 43, 45, 47, 49, 55, 85).

Diallo, Elhadji Amadou Oury and Toshiharu Sugawara (2019). „Coordination in Adversarial Multi-Agent with Deep Reinforcement Learning Under Partial Observability." In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, pp. 198–205 (cit. on pp. 7, 33, 59, 85).

Diallo, Elhadji Amadou Oury and Toshiharu Sugawara (2020a). „Multi-Agent Pattern Formation with Deep Reinforcement Learning.“ In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 10, pp. 13779–13780 (cit. on pp. 7, 51, 57, 85).

Diallo, Elhadji Amadou Oury and Toshiharu Sugawara (2020b). „Multi-Agent Pattern Formation: a Distributed Model-Free Deep Reinforcement Learning Approach.“ In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8 (cit. on pp. 7, 51, 85).

Diallo, Elhadji Amadou Oury, Ayumi Sugiyama, and Toshiharu Sugawara (2017). „Coordinated behavior by deep reinforcement learning in doubles pong game.“ In: *The Japanese Joint Agent Workshops and Symposium (JAWS)* (cit. on p. 86).

Diallo, Elhadji Amadou Oury, Ayumi Sugiyama, and Toshiharu Sugawara (2020). „Coordinated behavior of cooperative agents using deep reinforcement learning.“ In: *Neurocomputing* 396, pp. 230–240 (cit. on pp. 9, 33, 85).

Dieudonné, Yoann, Franck Petit, and Vincent Villain (2010). „Leader election problem versus pattern formation problem.“ In: *International Symposium on Distributed Computing*. Springer, pp. 267–281 (cit. on p. 52).

Dresner, Kurt and Peter Stone (2005). „Multiagent traffic management: An improved intersection control mechanism.“ In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. ACM, pp. 471–477 (cit. on p. 1).

Duchi, John, Elad Hazan, and Yoram Singer (2011). „Adaptive subgradient methods for online learning and stochastic optimization.“ In: *Journal of Machine Learning Research* 12.Jul, pp. 2121–2159 (cit. on p. 23).

Dudek, Gregory, Michael RM Jenkin, Evangelos Milios, and David Wilkes (1996). „A taxonomy for multi-agent robotics.“ In: *Autonomous Robots* 3.4, pp. 375–397 (cit. on p. 1).

Ferber, Jacques (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*. Vol. 1. Addison-Wesley Reading (cit. on p. 1).

Foerster, Jakob, Nantas Nardelli, Gregory Farquhar, Philip Torr, Pushmeet Kohli, Shimon Whiteson, et al. (2017). „Stabilising experience replay for deep multi-agent reinforcement learning.“ In: *arXiv preprint arXiv:1702.08887* (cit. on pp. 4, 15).

Garcez, Artur d'Avila, Marco Gori, Luis C Lamb, Luciano Serafini, Michael Spranger, and Son N Tran (2019). „Neural-Symbolic Comput-

ing: An Effective Methodology for Principled Integration of Machine Learning and Reasoning." In: *arXiv preprint arXiv:1905.06088* (cit. on p. 59).

Gerkey, Brian P and Maja J Matarić (2004). „A formal analysis and taxonomy of task allocation in multi-robot systems." In: *The International Journal of Robotics Research* 23.9, pp. 939–954 (cit. on p. 1).

Gu, Qin and Zhigang Deng (2011). „Formation sketching: an approach to stylize groups in crowd simulation." In: *Proceedings of Graphics Interface 2011*. Canadian Human-Computer Communications Society, pp. 1–8 (cit. on p. 52).

Gupta, Jayesh K, Maxim Egorov, and Mykel Kochenderfer (2017). „Cooperative multi-agent control using deep reinforcement learning." In: *International Conference on Autonomous Agents and Multiagent Systems*. Springer, pp. 66–83 (cit. on p. 53).

Hasselt, Hado V (2010). „Double Q-learning." In: *Advances in Neural Information Processing Systems*, pp. 2613–2621 (cit. on pp. 7, 11, 21).

Hausknecht, Matthew and Peter Stone (2015). „Deep recurrent q-learning for partially observable mdps." In: *CoRR, abs/1507.06527* (cit. on p. 18).

Hayzelden, Alex LG and John Bigham (1998). „Heterogeneous multi-agent architecture for ATM virtual path network resource configuration." In: *International Workshop on Intelligent Agents for Telecommunication Applications*. Springer, pp. 45–59 (cit. on p. 1).

He, He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III (2016). „Opponent modeling in deep reinforcement learning." In: *International Conference on Machine Learning*, pp. 1804–1813 (cit. on pp. 33, 34).

Hernandez-Leal, Pablo, Bilal Kartal, and Matthew E Taylor (2019). „A survey and critique of multiagent deep reinforcement learning." In: *Autonomous Agents and Multi-Agent Systems* 33.6, pp. 750–797 (cit. on pp. 4, 5).

Hüttenrauch, Maximilian, Sosic Adrian, Gerhard Neumann, et al. (2019). „Deep reinforcement learning for swarm systems." In: *Journal of Machine Learning Research* 20.54, pp. 1–31 (cit. on p. 53).

Iijima, Naoki, Masashi Hayano, Ayumi Sugiyama, and Toshiharu Sugawara (2016). „Analysis of task allocation based on social utility and incompatible individual preference." In: *Technologies and Applications of Artificial Intelligence (TAAI), 2016 Conference on*. IEEE, pp. 24–31 (cit. on p. 1).

Jadbabaie, Ali, Jie Lin, and A Stephen Morse (2003). „Coordination of groups of mobile autonomous agents using nearest neighbor rules." In: *Departmental Papers (ESE)*, p. 29 (cit. on p. 54).

Kaihara, Toshiya (2003). „Multi-agent based supply chain modelling with dynamic environment." In: *International Journal of Production Economics* 85.2, pp. 263–269 (cit. on p. 1).

Khan, Salim, Ravi Makkena, Foster McGeary, Keith Decker, William Gillis, and Carl Schmidt (2003). „A multi-agent system for the quantitative simulation of biological networks." In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*. ACM, pp. 385–392 (cit. on p. 1).

Kingma, Diederik P. and Jimmy Ba (2014). „Adam: A Method for Stochastic Optimization." In: *CoRR* abs/1412.6980. arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (cit. on pp. 23, 44).

Leibo, Joel Z, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel (2017). „Multi-agent reinforcement learning in sequential social dilemmas." In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 464–473 (cit. on pp. 15, 34).

Levine, Herbert, Wouter-Jan Rappel, and Inon Cohen (Dec. 2000). „Self-organization in systems of self-propelled particles." In: *Phys. Rev. E* 63 (1), p. 017101 (cit. on p. 14).

Lillicrap, Timothy P, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra (2015). „Continuous control with deep reinforcement learning." In: *arXiv preprint arXiv:1509.02971* (cit. on pp. 18, 53).

Lin, Jie, A Stephen Morse, and Brian DO Anderson (2007). „The multi-agent rendezvous problem. Part 2: The asynchronous case." In: *SIAM Journal on Control and Optimization* 46.6, pp. 2120–2147 (cit. on p. 54).

Lin, Long-Ji (1992). „Self-improving reactive agents based on reinforcement learning, planning and teaching." In: *Machine learning* 8.3-4, pp. 293–321 (cit. on p. 21).

Littman, Michael L (1994). „Markov games as a framework for multi-agent reinforcement learning." In: *Machine Learning Proceedings 1994*. Elsevier, pp. 157–163 (cit. on p. 3).

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch (2017a). „Multi-Agent Actor-Critic for Mixed Cooperative-

Competitive Environments.“ In: *CoRR* abs/1706.02275. arXiv: 1706.02275. URL: http://arxiv.org/abs/1706.02275 (cit. on p. 34).

Lowe, Ryan, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch (2017b). „Multi-agent actor-critic for mixed cooperative-competitive environments.“ In: *Advances in Neural Information Processing Systems*, pp. 6379–6390 (cit. on pp. 53, 59, 61, 63, 65).

Lux, Thomas and Michele Marchesi (1999). „Scaling and criticality in a stochastic multi-agent model of a financial market.“ In: *Nature* 397.6719, p. 498 (cit. on p. 1).

Mataric, Maja J (1993). „Designing emergent behaviors: From local interactions to collective intelligence.“ In: *Proceedings of the Second International Conference on Simulation of Adaptive Behavior*. Cambridge, MA, USA: MIT Press, pp. 432–441. URL: http://dl.acm.org/citation.cfm?id=171174.171225 (cit. on p. 2).

Miyashita, Yuki, Masashi Hayano, and Toshiharu Sugawara (2015). „Self-organizational reciprocal agents for conflict avoidance in allocation problems.“ In: *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*. IEEE, pp. 150–155 (cit. on p. 1).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller (2013). „Playing atari with deep reinforcement learning.“ In: *arXiv preprint arXiv:1312.5602* (cit. on pp. 3, 7, 20).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. (2015). „Human-level control through deep reinforcement learning.“ In: *Nature* 518.7540, p. 529 (cit. on pp. 4, 7, 10, 20, 24).

Nathan, Andre and Valmir Carneiro Barbosa (2006). „V-like formations in flocks of artificial birds.“ In: *CoRR* abs/cs/0611032. arXiv: cs/0611032. URL: http://arxiv.org/abs/cs/0611032 (cit. on pp. 14, 34).

Ng, Andrew Y (2003). „Shaping and policy search in reinforcement learning.“ PhD thesis. University of California, Berkeley (cit. on p. 17).

Olfati-Saber, R. (Mar. 2006). „Flocking for multi-agent dynamic systems: algorithms and theory.“ In: *IEEE Transactions on Automatic Control* 51.3, pp. 401–420 (cit. on p. 2).

Omidshafiei, Shayegan, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian (2017). „Deep Decentralized Multi-task Multi-Agent RL under Partial Observability.“ In: *arXiv preprint arXiv:1703.06182* (cit. on pp. 4, 15).

Panait, Liviu and Sean Luke (2005). „Cooperative multi-agent learning: The state of the art." In: *Autonomous agents and multi-agent systems* 11.3, pp. 387–434 (cit. on pp. 1, 3, 6).

Peng, Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang (2017). „Multiagent Bidirectionally-Coordinated nets for learning to play StarCraft combat games." In: *arXiv preprint arXiv:1703.10069* (cit. on p. 4).

Pynadath, David V and Milind Tambe (2002). „The communicative multiagent team decision problem: Analyzing teamwork theories and models." In: *Journal of artificial intelligence research* 16, pp. 389–423 (cit. on pp. 33, 35).

Rana, Omer F. and Kate Stout (2000). „What is Scalability in Multi-agent Systems?" In: *Proceedings of the Fourth International Conference on Autonomous Agents*. AGENTS '00. Barcelona, Spain: ACM, pp. 56–63 (cit. on pp. 3, 35).

Ranjbar-Sahraei, Bijan, Faridoon Shabaninia, Alireza Nemati, and Sergiu-Dan Stan (2012). „A novel robust decentralized adaptive fuzzy control for swarm formation of multiagent systems." In: *IEEE Transactions on Industrial Electronics* 59.8, pp. 3124–3134 (cit. on p. 54).

Reynolds, Craig W (1987). „Flocks, herds and schools: A distributed behavioral model." In: *ACM SIGGRAPH computer graphics*. Vol. 21. 4. ACM, pp. 25–34 (cit. on pp. 2, 33).

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2015). „Prioritized experience replay." In: *arXiv preprint arXiv:1511.05952* (cit. on p. 40).

Shapley, Lloyd S (1953). „Stochastic games." In: *Proceedings of the national academy of sciences* 39.10, pp. 1095–1100 (cit. on p. 1).

Shoham, Yoav and Kevin Leyton-Brown (2008). *Multiagent systems: Algorithmic, game-theoretic, and logical foundations*. Cambridge University Press (cit. on pp. 1, 3, 35).

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). „Mastering the game of Go with deep neural networks and tree search." In: *nature* 529.7587, pp. 484–489 (cit. on p. 10).

Skinner, Burrhus Frederic (1990). *The behavior of organisms: An experimental analysis*. BF Skinner Foundation (cit. on p. 17).

Smola, Alex, Arthur Gretton, Le Song, and Bernhard Schölkopf (2007). „A Hilbert space embedding for distributions." In: *International Con-*

ference on Algorithmic Learning Theory. Springer, pp. 13–31 (cit. on p. 54).

Stone, Peter and Manuela Veloso (2000). „Multiagent systems: A survey from a machine learning perspective." In: *Autonomous Robots* 8.3, pp. 345–383 (cit. on p. 3).

Sugawara, Toshiharu (Mar. 1990). „A cooperative LAN diagnostic and observation expert system." In: *Ninth Annual International Phoenix Conference on Computers and Communications. 1990 Conference Proceedings*, pp. 667–674 (cit. on p. 1).

Sukhbaatar, Sainbayar, Arthur Szlam, and Rob Fergus (2016). „Learning Multiagent Communication with Backpropagation." In: *NIPS* (cit. on pp. 15, 33, 34, 36).

Sunehag, Peter, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. (2018). „Value-decomposition networks for cooperative multi-agent learning based on team reward." In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 2085–2087 (cit. on p. 59).

Sutton, Richard S and Andrew G Barto (1998). *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge (cit. on pp. 3, 19, 42).

Suzuki, Ichiro and Masafumi Yamashita (1999). „Distributed anonymous mobile robots: Formation of geometric patterns." In: *SIAM Journal on Computing* 28.4, pp. 1347–1363 (cit. on p. 51).

Tampuu, Ardi, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente (2017). „Multiagent cooperation and competition with deep reinforcement learning." In: *PloS one* 12.4, e0172395 (cit. on p. 15).

Tan, Ming (1993). „Multi-agent reinforcement learning: Independent vs. cooperative agents." In: *Proceedings of the tenth international conference on machine learning*, pp. 330–337 (cit. on p. 3).

Taylor, Matthew E. and Peter Stone (2009). „Transfer Learning for Reinforcement Learning Domains: A Survey." In: *Journal of Machine Learning Research* 10, pp. 1633–1685 (cit. on p. 7).

Tieleman, Tijmen and Geoffrey Hinton (2012). „Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude." In: *COURSERA: Neural networks for machine learning*, pp. 26–31 (cit. on p. 23).

Tomlin, Claire, George J Pappas, and Shankar Sastry (1998). „Conflict resolution for air traffic management: A study in multiagent hybrid systems." In: *IEEE Transactions on automatic control* 43.4, pp. 509–521 (cit. on p. 1).

Toner, John and Yuhai Tu (Oct. 1998). „Flocks, herds, and schools: A quantitative theory of flocking." In: *Phys. Rev. E* 58 (4), pp. 4828–4858 (cit. on p. 14).

Tuyls, Karl and Gerhard Weiss (2012). „Multiagent learning: Basics, challenges, and prospects." In: *AI Magazine* 33.3, p. 41 (cit. on p. 3).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). „Deep Reinforcement Learning with Double Q-Learning." In: *AAAI*. Vol. 16, pp. 2094–2100 (cit. on pp. 7, 11, 14, 21, 24).

Velthuijsen, Hugo (Aug. 1993). „Distributed artificial intelligence for runtime feature-interaction resolution." In: *Computer* 26.8, pp. 48–55 (cit. on p. 1).

Wang, Ziyu, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas (2015). „Dueling network architectures for deep reinforcement learning." In: *arXiv preprint arXiv:1511.06581* (cit. on pp. 7, 11, 18).

Watkins, Christopher John Cornish Hellaby (1989). „Learning from delayed rewards." PhD thesis. King's College, Cambridge (cit. on pp. 9–11, 20, 21).

Weiss, Gerhard (1993). „Learning to coordinate actions in multi-agent systems." In: *Proceedings of the international joint conference on artificial intelligence*, pp. 311–316 (cit. on p. 1).

Weiss, Gerhard (1999). *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT press (cit. on p. 1).

Wiering, MA (2000). „Multi-agent reinforcement learning for traffic light control." In: *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*, pp. 1151–1158 (cit. on p. 1).

Wooldridge, Michael (2009). *An introduction to multiagent systems*. John Wiley & Sons (cit. on p. 1).

Xiang, Wei and Heow Pueh Lee (2008). „Ant colony intelligence in multi-agent dynamic manufacturing scheduling." In: *Engineering Applications of Artificial Intelligence* 21.1, pp. 73–85 (cit. on p. 1).

Xu, Huaxing, Haibing Guan, Alei Liang, and Xinan Yan (2010). „A multi-robot pattern formation algorithm based on distributed swarm intelligence." In: *2010 Second International Conference on Computer Engineering and Applications*. Vol. 1. IEEE, pp. 71–75 (cit. on p. 54).

Yamauchi, Yukiko, Taichi Uehara, and Masafumi Yamashita (2015). „Pattern formation problem for synchronous mobile robots in the three dimensional euclidean space." In: *arXiv preprint arXiv:1509.09207* (cit. on p. 52).

Yeh, ChiaWei and Toshiharu Sugawara (2016). „Solving Coalition Structure Generation Problem with Double-Layered Ant Colony Optimization." In: *Advanced Applied Informatics (IIAI-AAI), 2016 5th IIAI International Congress on*. IEEE, pp. 65–70 (cit. on p. 1).

Zeiler, Matthew D (2012). „ADADELTA: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701* (cit. on p. 23).

Zheng, Lianmin, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu (2018). „MAgent: A many-agent reinforcement learning platform for artificial collective intelligence." In: *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on pp. 59, 60).

Zhou, Zhengyuan, Wei Zhang, Jerry Ding, Haomiao Huang, Dušan M Stipanović, and Claire J Tomlin (2016). „Cooperative pursuit with Voronoi partitions." In: *Automatica* 72, pp. 64–72 (cit. on p. 54).

Zhu, Zean, Elhadji Amadou Oury Diallo, and Toshiharu Sugawara (2020). „Learning Efficient Coordination Strategy for Multi-step Tasks in Multi-agent Systems using Deep Reinforcement Learning." In: *ICAART (1)*, pp. 287–294 (cit. on p. 85).

PUBLICATIONS

Journals

- Elhadji Amadou Oury Diallo, Ayumi Sugiyama, and Toshiharu Sugawara (2020). „Coordinated behavior of cooperative agents using deep reinforcement learning." In: *Neurocomputing* 396, pp. 230–240

International Conferences

- Elhadji Amadou Oury Diallo and Toshiharu Sugawara (2020b). „Multi-Agent Pattern Formation: a Distributed Model-Free Deep Reinforcement Learning Approach." In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8

- Elhadji Amadou Oury Diallo and Toshiharu Sugawara (2020a). „Multi-Agent Pattern Formation with Deep Reinforcement Learning." In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 10, pp. 13779–13780

- Zean Zhu, Elhadji Amadou Oury Diallo, and Toshiharu Sugawara (2020). „Learning Efficient Coordination Strategy for Multi-step Tasks in Multi-agent Systems using Deep Reinforcement Learning." In: *ICAART (1)*, pp. 287–294

- Elhadji Amadou Oury Diallo and Toshiharu Sugawara (2019). „Coordination in Adversarial Multi-Agent with Deep Reinforcement Learning Under Partial Observability." In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, pp. 198–205

- Elhadji Amadou Oury Diallo and Toshiharu Sugawara (2018b). „Learning strategic group formation for coordinated behavior in adversarial multi-agent with double DQN." in: *International Conference on Principles and Practice of Multi-Agent Systems*. Springer, pp. 458–466

- Elhadji A. O. Diallo, A. Sugiyama, and T. Sugawara (Dec. 2017). „Learning to Coordinate with Deep Reinforcement Learning in

Doubles Pong Game.“ In: *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 14–19

Workshops

- Elhadji Amadou Oury Diallo and Toshiharu Sugawara (2018a). „Learning coordination in adversarial multi-agent dqn with dec-pomdps.“ In: *workshop on Reinforcement Learning under Partial Observability, 32nd Neurips*

Domestic Conferences

- Elhadji Amadou Oury Diallo, Ayumi Sugiyama, and Toshiharu Sugawara (2017). „Coordinated behavior by deep reinforcement learning in doubles pong game.“ In: *The Japanese Joint Agent Workshops and Symposium (JAWS)*

COLOPHON

This thesis was typeset in LaTeX using classicthesis style and biblatex for the bibliography. Most of the figures were generated using Matplotlib.