Multi-Agent Reinforcement Learning in Large Complex Environments

by

Sriram Ganapathi Subramanian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Scott Sanner Associate Professor, Dept. of Industrial Engineering & Dept. of Computer Science, University of Toronto
Supervisors:	Mark Crowley Associate Professor Dept. of Electrical and Computer Engineering University of Waterloo
	Kate Larson Professor, Cheriton School of Computer Science University of Waterloo
Internal-External Member:	Pascal Poupart Professor, Cheriton School of Computer Science University of Waterloo
Internal Member:	Zhou Wang Professor, Dept. of Electrical and Computer Engineering University of Waterloo
Internal Member:	Seyed Majid Zahedi Assistant Professor Dept. of Electrical and Computer Engineering University of Waterloo

Author's Declaration

Statement of Contributions

The contents of this thesis includes material present as part of the following published papers in journal and conferences. In all of these papers, I was the first author responsible for conceptualizing the research directions, laying out the research questions and hypotheses, designing and performing the experimental and theoretical analyses, and writing the draft manuscripts. Other co-authors involved professors who provided feedback, edited the manuscript drafts, contributed intellectual inputs, and provided general supervision of the research.

1. Chapter 3 contains work published as part of [255]. Sriram Ganapathi Subramanian, Matthew E. Taylor, Kate Larson, & Mark Crowley (2022). Multi-Agent Advisor Q-Learning. Journal of Artificial Intelligence Research, 74, 1-74.

2. Chapter 5 contains work published as part of [248]. Sriram Ganapathi Subramanian, Pascal Poupart, Matthew E. Taylor, & Nidhi Hegde (2020). Multi Type Mean Field Reinforcement Learning. International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Aukland, New Zealand.

3. Chapter 6 contains work published as part of [252]. Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, & Pascal Poupart (2021). Partially Observable Mean Field Reinforcement Learning. International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), London, United Kingdom.

4. Chapter 7 contains work published as part of [253]. Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, & Pascal Poupart (2022). Decentralized Mean Field Games. Proceedings of the AAAI conference on Artificial Intelligence (AAAI), Vancouver, Canada.

Abstract

Multi-agent reinforcement learning (MARL) has seen much success in the past decade. However, these methods are yet to find wide application in large-scale real world problems due to two important reasons. First, MARL algorithms have poor sample efficiency, where many data samples need to be obtained through interactions with the environment to learn meaningful policies, even in small environments. Second, MARL algorithms are not scalable to environments with many agents since, typically, these algorithms are exponential in the number of agents in the environment. This dissertation aims to address both of these challenges with the goal of making MARL applicable to a variety of real world environments.

Towards improving sample efficiency, an important observation is that many real world environments already, in practice, deploy sub-optimal or heuristic approaches for generating policies. A useful possibility that arises is how to best use such approaches as *advisors* to help improve reinforcement learning in multi-agent domains. In this dissertation, we provide a principled framework for incorporating action recommendations from online sub-optimal advisors in multi-agent settings. To this end, we propose a general model for learning from external advisors in MARL and show that desirable theoretical properties such as convergence to a unique solution concept, and reasonable finite sample complexity bounds exist, under a set of common assumptions. Furthermore, extensive experiments illustrate that these algorithms: can be used in a variety of environments, have performances that compare favourably to other related baselines, can scale to large state-action spaces, and are robust to poor advice from advisors.

Towards scaling MARL, we explore the use of mean field theory. Mean field theory provides an effective way of scaling multi-agent reinforcement learning algorithms to environments with many agents, where other agents can be abstracted by a virtual mean agent. Prior work has used mean field theory in MARL, however, they suffer from several stringent assumptions such as requiring fully homogeneous agents, full observability of the environment, and centralized learning settings, that prevent their wide application in practical environments. In this dissertation, we extend mean field methods to environments having heterogeneous agents, and partially observable settings. Further, we extend mean field methods to include decentralized approaches. We provide novel mean field based MARL algorithms that outperform previous methods on a set of large games with many agents. Theoretically, we provide bounds on the information loss experienced as a result of using the mean field and further provide fixed point guarantees for Q-learning-based algorithms in each of these environments.

Subsequently, we combine our work in mean field learning and learning from advisors to show that we can achieve powerful MARL algorithms that are more suitable for real world environments as compared to prior approaches. This method uses the recently introduced attention mechanism to perform per-agent modelling of others in the locality, in addition to using the mean field for global responses. Notably, in this dissertation, we show applications in several real world multi-agent environments such as the Ising model, the ride-pool matching problem, and the massively multi-player online (MMO) game setting (which is currently a multi-billion dollar market).

Acknowledgements

I would like to sincerely thank my supervisors, Mark Crowley and Kate Larson, for their unwavering support throughout my PhD journey. They always made a sincere effort to provide me the best PhD experience possible by minimizing disruptions, and providing all the resources needed to complete my research objectives. They were ever ready to contribute their expertise towards coming up with concrete research questions, prototyping solutions, writing publications, and responding to reviewers. We had numerous research discussions, where they refined my ideas, critiqued (positively) my thoughts, and provided a structure to my research. Their timely help and motivation enabled me to overcome several challenges, that includes navigating a global pandemic, and dealing with multiple paper rejections. It is fair to say that they took a personal interest in my success for which I am highly thankful.

My utmost thanks to Pascal Poupart and Matthew E. Taylor (University of Alberta), who were incredible collaborators that provided much of their potentially limited time in engaging with my research. They provided timely and high quality feedback at several stages of my research journey. They greatly enhanced the quality of this dissertation, and I am very grateful for their support.

I would like to extend my thanks to Seyed Majid Zahedi and Zhou Wang, who were part of my doctoral committee. They provided critical feedback and useful ideas for my research. This helped me view my work with different perspectives.

My research required heavy computation that was provided by Compute Canada, Microsoft Azure, and Vector Institute. My sincere thanks to each of these institutions and their compute support teams.

Above all I would like to express my gratitude to my parents. They provided a highly education focussed environment at home throughout my childhood and teenage years. Education was given the utmost importance and this is the single most critical contributor towards my decision of pursuing higher learning. They have given me utmost freedom and much unconditional love throughout the many years required to complete a doctoral dissertation, for which I am highly grateful.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Reinforcement learning (RL) and deep learning have been combined effectively in the past decade to give super human performances in a large suite of single-agent games [167]. RL can be regarded as a sub-field of machine learning, where an agent learns to act optimally in a dynamic environment using weak return signals (called rewards). Deep reinforcement learning (DRL) algorithms combine RL and deep learning, where deep neural networks are trained to approximate either the policy or the value function, effectively learning good mappings from states to actions. DRL has been used in some single-agent real world environments for resource management and allocation [157], robot management [129], and personalized recommendation systems [315]. Still, DRL has not been generally applicable in many real world problems, due to problems of sample efficiency [311] and the nature of multiple agent interaction in these real world domains [314].

RL is also being used in the field of multi-agent systems, where more than one agent is learning simultaneously in the same environment [178]. Single-agent RL algorithms are not directly applicable in multi-agent environments due to problems like non-stationarity [97], credit-assignment [36], and the curse of dimensionality [303]. An excellent survey of multi-agent RL (MARL) from Hernandez et al. [98] describes these problems in detail. Regarding non-stationarity, in MARL, the transition function and the reward function for each agent depends on the joint action of all agents. Hence, from a single-agent's perspective, these functions can change over time based on dynamically changing strategies of other agents in the environment. However, single-agent RL approaches typically assume stationary reward and transition functions for learning [256] and are not directly suited for multi-agent environments. The multi-agent credit assignment problem [277] deals with the issue of each agent requiring to infer its contributions to the obtained reward. For example, in cooperative environments, all coordinating agents may only obtain global rewards which depends on policies of all agents together and not each individual agent. The curse of dimensionality in MARL arises due to the nature of the multi-agent state-action spaces growing exponentially with the number of agents in the environment [39]. Further, problems from single-agent RL such as sample complexity [33] are potentially magnified in the multi-agent case. Many real world environments such as combating wildfires, fighting pandemics, resource allocation operations, and disaster responses are inherently large multi-agent systems, where each agent interacts simultaneously with many other agents. These domains can be classified as cooperative (fighting wildfires), competitive (demand and supply logistics), or mixed (multi party elections with coalitions). Thus, applying RL algorithms in real world domains will need to explicitly handle the multi-agent nature of these domains in addition to addressing the other challenges in multi-agent systems mentioned above.

MARL has been traditionally focused on simple toy problems involving two (or few) player games with a very restrictive set of applications [278]. However, there is a large set of real world domains, that span different areas such as wildfire fighting [115], financial applications like stock trading and portfolio optimization [114], and industrial applications like the ride-pool matching problem (RMP) [6], where multi-agent learning algorithms would be highly beneficial. An effort towards enabling such applications will greatly benefit the dual fields of RL and multi-agent systems.

In this dissertation, we will theoretically and empirically address some of the most compelling problems in the field of MARL that prevents its wide application to large and complex problems. This dissertation aims to be a significant step towards making MARL algorithms applicable to many real world domains.

1.1 Research Problem And Contributions

We focus our research on two fundamental problems in MARL.

The first problem is the issue of sample efficiency. Sample efficiency is the problem of learning effectively from every data sample. Each data sample in RL constitutes an "experience" for the agent interacting with the environment. To put this problem in context, the best RL algorithms need millions of data samples to learn at the level of human performance, while humans can learn such policies using only a few samples (a good example is performances in Atari games [167]). This problem extends to the field of MARL as well. To provide an example, the AlphaStar algorithm that is used to solve the multi-agent StarCraft challenge [206], requires 200 years of game play data for training to learn good policies. Another example is the OpenAI Five algorithm that provides state-of-the-art performances in the multi-agent game of Dota [20], which requires to be trained on 180 years of game play data involving computational infrastructure consisting of 256 GPUs and 128,000 CPU cores [184]. In many real world domains, this is a critical issue as there is a relative paucity of data with which learning agents can learn effectively. However, many of these real-world domains have existing scientific models and other heuristics that while potentially sub-optimal, are still used in practice to generate policies. Our proposed approach integrates this existing information (which we will encapsulate as *advisors*) to bootstrap MARL agents in many real world problems, since learning from scratch is highly challenging due to its poor sample efficiency. This approach falls within the paradigm of *learning from demonstrations* (LfD) [213]. As the agents begin learning, the classic RL/MARL algorithms train their policies purely by random exploration [291] in the environment. This involves trying out different actions in every situation (state) to figure out the best possible action. In a large environment, this is not optimal as the number of different possibilities are exponentially large. In this context, an advisor can help in targeted exploration by suggesting an action for each situation, which the agents can then try out and evaluate. The learning agents must be able to rely on advisor feedback and use that to inform their policies, especially when the agents are uncertain about their own policies (typically at the beginning of learning). Eventually, learning agents must be able to surpass the sub-optimal advisors in performance given sufficient training. Though prior works have considered LfD in MARL [225], they contain several limiting assumptions such as using restrictive multi-agent environments (for example two-agent zero-sum games), requiring access to near-optimal experts, presence of offline demonstrations, and requiring all demonstrations to come from a single demonstrator. In our approach to solve this problem, we will first provide a principled framework, called *ADvising Multiple Intelligent Reinforcement Agents* (ADMIRAL) that integrates action recommendations from an online sub-optimal advisor in non-restrictive *general-sum stochastic game* settings. In this approach, we restrict each agent to have access to at-most one advisor. Subsequently, we will relax this condition and propose a new framework that allows each agent to have access to multiple advisors. Here, the agents will learn to pick the best advisor at a given state with the objective of learning good MARL policies faster. In this way, this dissertation contributes principled MARL solutions for learning from advisor(s) that do not require limiting assumptions seen in prior works.

The second challenge is the issue of scaling MARL to environments with large numbers of agents (many agents). While many MARL approaches in recent years have addressed problems with large state and action spaces, the number of agents has still been restricted to a few (tens or less) agents [70, 95]. Since, traditionally, MARL algorithms have been exponential in the number of agents [104], these algorithms are intractable in environments with many agents. In the last decade, mean field theory [236] has been effectively combined with MARL to give tractable solutions [88, 136, 303]. Mean field theory replaces all agent interactions with an effective interaction on a single "virtual" agent called the mean field. Hence, the many agent problem has been reduced to a two agent problem, where the second agent is a mean field aggregate of all other agents in the environment. With this model, all current MARL algorithms become tractable as each agent needs to formulate best responses only to one other agent (i.e., the mean field). In spite of having this scalability advantage, mean field approaches do not come for free. Certain conditions must be met which limit its use in practice. The conditions imposed by prior work and suitable relaxations to these conditions provided by this dissertations are given as follows.

- The first condition involves an assumption that all agents in the environment are fully homogeneous (i.e., have the same state space, action space, and reward functions). However, in large real world systems, many agents will have different abilities, objectives and interests. We provide a new approach that groups different agents with such differences into different sets (called *types*), such that, within a type the mean field homogeneity assumption still holds, but the assumption is not required to hold across types. This analysis will extend previous research [303] on combining mean field theory and RL to environments with the presence of heterogeneous agents.

- The second limiting condition is a requirement that all agents can access global information (that includes information about each of the other agents) in mean field environments. While this assumption is strong in general, especially in large scale systems with many agents, it is impractical to assume that every agent will have access to global information of the other agents in the environment. Such information can only be obtained if the strategies of all other agents are known in advance. We relax this assumption and maintain a distribution over strategies that are updated at each time step based on the varying number of agents that each agent observes in an environment.

- The third limiting requirement is that of using centralized learning approaches. Building centralized infrastructure over environments with many agents is prohibitively hard, especially in terms of computation. In addition, such a system is susceptible to several failure modes (such as network failures). We provide a new mean field setting that will relax this requirement and extend mean field methods to include possibly *decentralized* solution methods.

- Finally, mean field approaches assume that no per-agent modelling of others is required and that every agent has the same impact on the learning of other agents. Under the assumption of a very large (possibly infinite in the limit) number of agents in the environment, each individual agents' impact on the environment is specified to be infinitesimal [136], which leads to all agents only calculating best responses to the mean field (no per-agent modelling of other agents). However, in practice, real-world environments have only finite agents and each agent is impacted more by some agents than others. In this dissertation, we provide a mean field algorithm that uses the recently introduced *attention mechanism* [282] to learn the different levels of importance needing to be attributed to a finite set of nearby agents that can have a high impact on the performance of the central agent (a representative learning agent). Here the attention mechanism is used to model other agents nearby (local impact) and the mean field is used to model agents further away (global impact). This leads to a scalable algorithm that is still capable of providing best responses to the behaviour of other (potentially impactful) agents nearby.

In general, mean field methods in RL are also sample inefficient since they learn from scratch (same as in any RL/MARL algorithm). We will extend our work for learning from advisors in MARL to the mean field setting to address this challenge.

Along with empirical research advances that are needed to use MARL algorithms in large scale environments, theoretical underpinnings of these algorithms must also be investigated to get a clear understanding of their performance. Hence, a theoretical analysis of proposed MARL algorithms is an important component of our research. RL algorithms are in their essence, fixed point iteration problems, where the algorithms iterate until more learning is not desirable or possible [262]. Previous research has given fixed point guarantees for many well known RL algorithms [259, 291]. In the multi-agent case the theoretical guarantees involve a form of convergence to an equilibrium, such as the Nash equilibrium [122, 230]. In this equilibrium, no agent has an incentive to unilaterally deviate from its policy. In this sense, an understanding of the field of game-theory and the theory of learning in games [75] contribute to research advancements in MARL [36, 180]. Particularly, this dissertation will focus on providing a convergence guarantee in self-play and proving that each agent provides best responses to other agent strategies, to show that the different contributed algorithms are principled (or that the agents playing these algorithms are rational). This is in line with the conditions provided by Bowling and Veloso [30, 31].

1.2 Research Area

The area of focus for this research is concisely captured by the Venn diagram in Figure 1.1. Our area of research lies in the intersection of four areas. Here we will provide a short summary of each of these areas focussing on their relation to each other and our research focus. More details are provided in the upcoming chapters.

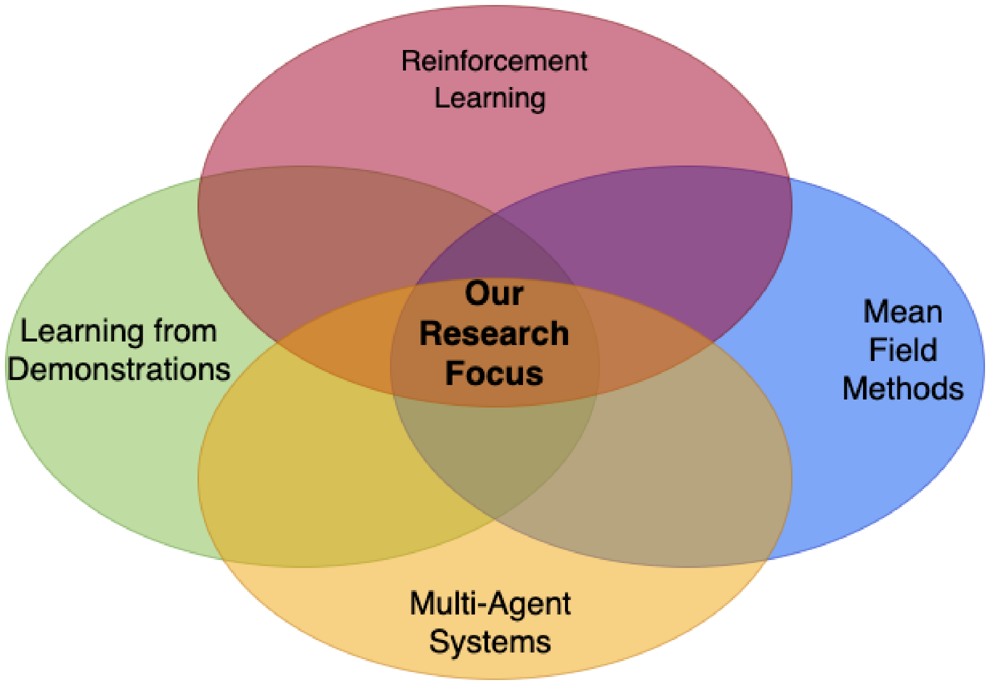


Figure 1.1: The research focus of this dissertation in relation to other fields.

The first field of interest is that of RL [256], which we have introduced previously. RL has been traditionally focussed on single-agent learning environments [186], while it can also extend to multi-agent environments [97]. Since RL suffers from the problem of sample efficiency, learning from demonstrations (LfD) [214] techniques are used to make RL training faster [191]. LfD is an independent field of research that has been investigated for over 40 years [57]. LfD approaches try to replace traditional learning that learns from scratch and other time consuming hand-crafted programming processes for agents/robots with demonstrations of an expert's approach to the task [214]. Multi-agent systems (MAS) [240] is another area of research that is relevant to our work. This field investigates the behaviour of systems containing more than one (intelligent) agent learning to take decisions simultaneously. Learning in MAS generally involves the use of RL. Consequently, RL approaches have been extensively studied in multi-agent systems forming its own sub-field i.e., MARL [97]. Finally, mean field methods is an area of research under game theory [221], that studies strategic decision-making of independent agents which are part of large populations [136]. Our work aims to improve MARL training using LfD, and scale MARL algorithms using mean field approaches. Hence, it lies at the intersection of all of these four fields of research.

1.3 Thesis Overview

Thesis Statement: The following thesis statement captures the core contributions of this dissertation:

MARL is not currently applicable to a vast majority of large-scale real-world environments due to two fundamental limitations:

1. *Poor sample efficiency in MARL. This dissertation considers learning from demonstration based solutions and provides principled approaches that relax limiting assumptions in prior works.*

2. *Inability to scale to environments with many agents in them. This dissertation considers solutions based on mean field learning and provides principled approaches that relax limiting assumptions in prior works.*

Subsequently, combining these two solution approaches (that relaxes the fundamental limitations) can lead to powerful MARL solutions that are potentially suitable for a large class of real-world problems, and have clear advantages over state-of-the-art algorithms (baselines) from the MARL literature.

Now that we have motivated our research problem and outlined our solution approach, we will go into more details in the next chapters. The organization of the rest of this dissertation is described here.

In Chapter 2, we will provide some background information and summarize prior research work in the field of RL, MARL, and game theory. To be concise, we will focus on prior works and concepts that are most directly related to our research. We will begin by providing some fundamentals of RL and then move on to extending this to the multi-agent setting and finally to the mean field setting.

In Chapter 3, we consider the problem of using advisors in MARL (specifically the non-restrictive *general-sum stochastic game* [219] setting). We provide a principled framework for incorporating action recommendations from online sub-optimal advisors in multi-agent settings. In this chapter we restrict each agent to have access to at-most one advisor during training. We provide practical algorithms and analyze them theoretically and empirically.

The core contents of this chapter are published in the Journal of Artificial Intelligence Research (JAIR) [255] and is available on arXiv [254].

In Chapter 4, we will relax the restriction of one advisor from the previous chapter. This chapter considers the problem of simultaneously learning from multiple independent advisors in MARL (specifically the general-sum stochastic games). We will provide a principled framework for multiple agent learning in the presence of multiple advisors. Using this framework, we will propose suitable MARL algorithms and analyze them empirically as well as theoretically.

In Chapter 5, we will switch focus to scaling MARL algorithms to many agent environments using mean field theory. We give the theoretical foundations and experimental results of our approach on extending mean field theory and MARL to multiple types in the environment. The core contents of this chapter have been published as a full paper in AAMAS-2020 [248] and is available on arXiv [249].

In Chapter 6, we study the issue of partial observation in many agent RL, and analyze two different classes of problems in this scenario. We provide two practical algorithms that work on each of these scenarios and establish convergence properties theoretically. The core contents of this chapter have been published as a full paper in AAMAS-2021 [252] and is available on arXiv [250].

In Chapter 7 we extend mean field learning methods to approaches involving decentralized training protocols. We will propose a new mean field system, called *decentralized mean field games* (DMFGs) and provide a suitable decentralized solution concept for this system. We provide practical algorithms for learning in DMFGs and analyze their performances in large scale games. The core contents of this chapter have been published as a full paper in AAAI-2022 [253] and is available on arXiv [251].

In Chapter 8 we provide an algorithm that improves the training of MARL algorithms in many agent environments using action advising and mean field learning. This algorithm will combine the advantages of the algorithms provided in previous chapters. Further, this chapter will explore the use of a transformer architecture [282] in mean field algorithms and demonstrate the advantages of this method.

In Chapter 9, we conclude the dissertation by providing a summary of its major contributions and a discussion of important avenues for future work.

Chapter 2

Background And Prior Work

In this chapter, we will go through the important background material for this dissertation. We start with the introduction of single-agent reinforcement learning (RL) methods including Deep RL (DRL). Subsequently, we will expand the discussions to include stochastic games and MARL. The focus will be on providing detailed discussions about common topics that will appear in all the remaining chapters. Other topics that are specific to each of the upcoming chapters are briefly mentioned here, while the details are deferred until the respective chapters.

2.1 Reinforcement Learning

We will first introduce the single-agent RL framework and then also describe some research advances using a combination of deep learning and reinforcement learning. This field is highly dynamic, with many new publications being released every year. We will try to keep things as brief as possible by highlighting the seminal papers in this field along with the papers that are most relevant to our research.

Single agent reinforcement learning [256] is the most common form of reinforcement learning in the literature [9]. In RL, decision problems are modelled as Markov Decision Processes (MDPs) [256]. We formally define an MDP as follows.

Definition 1. *A Markov decision process (MDP) is defined as* $\langle \mathcal{S}, A, R, T, \gamma \rangle$ *where* $\mathcal{S}$ *is the set of states,* A *is a set of actions,* $R : S \times A \mapsto \mathbb{R}$ *is the reward function,* $T : S \times A \times S \mapsto [0, 1]$ *is the transition function and* $0 \leq \gamma < 1$ *is the discount factor.*

Given an MDP, an agent starts in some state s, takes some action $a \in A$, and then transitions to a state s' with probability $T(s, a, s')$ where it collects some reward $R(s, a)$. A stochastic stationary policy can be represented as, $\pi : \mathcal{S} \rightarrow \Delta(A)$, where $\Delta(A)$ denotes the space of probabilistic distributions over the agent's action space. The stationary policy can also be deterministic (represented as $\pi : \mathcal{S} \rightarrow A$), in which case, it specifies a deterministic action for each state s. The policy is a behaviour policy if it depends on the entire history of game play. This is expressed as $\pi_t = H_t \rightarrow \Delta(A)$, where $h_t \in H_t, h_t = (s_0, a_0, \ldots, s_{t_1}, a_{t-1}, s_t)$. Alternatively, the policy is a stationary policy if the policy is independent of time, expressed as $\pi_t = \overline{\pi}$ for all t.

The objective of an agent in the MDP is to determine an optimal policy π^*, that maximizes the sum of expected discounted future rewards. In an MDP, it can be proved that restricting policies to only stationary policies does not result in any loss of optimality [196], and hence, it is sufficient to only consider stationary policies for determining the optimal policy [256]. The value, or expected discounted sum of future rewards, of following some policy π when starting in state s is defined as,

$$v(s, \pi) = \sum_{t=0}^{\infty} \gamma^t E[r_t | s_0 = s, \pi] \tag{2.1}$$

where r_t is the reward collected at time t.

To determine the optimal policy, it is typical to consider an iterative search technique [196] that tries to obtain the fixed point of the *Bellman* equation expressed as,

$$v(s, \pi^*) = \max_a \{R(s, a) + \gamma \sum_{s'} T(s, a, s') v(s', \pi^*)\}. \tag{2.2}$$

It is guaranteed that a solution (fixed point) to the iterative update provided in Eq. 2.2 exists [256]. The policy π^* that satisfies Eq. 2.2 is guaranteed to be the optimal policy [256, 291]. The value of the optimal policy is defined as the optimal value (i.e., the optimal value provides the maximum expected discounted sum of future rewards obtainable starting from each possible state in the MDP).

The algorithms that learn and maintain explicit estimates of the dynamics T are called model based. In contrast, algorithms that use interactions with the environment to directly learn a value function or policy are model free. In this dissertation, we will only focus on model free approaches. The current state-of-the-art in RL for learning in general environments without an explicitly known (or easily obtainable) model is to use model free approaches [10]. The model based approaches tend to be computationally more demanding and even comparatively small inaccuracies in these models lead to instability in learning [169]. Model free methods can be mostly divided into value-based and direct policy search approaches, as described next.

2.1.1 Q-learning

Q-learning [291] is a widely used value-based RL algorithm that learns an optimal Q-value, based on which the optimal policy is derived. Q-learning works naturally in single-agent settings with a discrete action space. Many variants of this algorithm have also been proposed [87, 309]. The classic Q-learning algorithm, does a tabular update for the Q-value of each state, action pair in the environment. This algorithm is model-free, with the tabular values being updated each time an action is taken using the Bellman update equation,

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha[(r_t + \gamma \max_a Q_t(s',a)) - Q_t(s,a)] \tag{2.3}$$

where $\alpha \in [0,1]$ is the learning rate.

The Eq. 2.3 is guaranteed to have a fixed point, denoted as $Q^*(s,a)$, which provides the maximum possible expected discounted sum of rewards obtainable after taking action a in state s [291]. The optimal policy is then, to return the action that maximizes the optimal Q-value at each state. The relation between the value function and the Q-function can be expressed as $v(s,\pi^*) = \max_a Q^*(s,a)$.

In Q-learning it is common to use a different policy for deciding actions as compared to the policy being updated (also known as the *target* policy). This makes Q-learning an *off-policy* algorithm. Alternatively, algorithms that use the target policy for deciding actions are known as *on-policy* algorithms.

The on-policy variant of Q-learning is called Sarsa [256]. The Sarsa update replaces the max operator in Equation 2.3 with the next action a' at state s' from the current policy. This can be expressed as (where the action a' at the next state s' is determined before the performing the Q-update),

$$Q_{t+1}(s,a) = Q_t(s,a) + \alpha[(r_t + \gamma Q_t(s',a')) - Q_t(s,a)]. \tag{2.4}$$

2.1.2 Policy Gradients And Actor Critics

Value based methods like Q-learning optimize a value function from which an optimal policy is obtained. Policy gradient methods, on the other hand, optimize the policy directly and can work with high dimensional state and action spaces as they do not need a look-up tabular representation like the value based methods [131]. Some measure of performance is used to update the parameters of the policy using stochastic gradient descent. The policy gradient methods are derived from the classic *stochastic policy gradient theorem* [257]. This can be expressed as[1],

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{s\sim\rho^\pi, a\sim\pi_\theta}\Big[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a)\Big] \tag{2.5}$$

where ρ^π is the (discounted) state distribution under the stochastic policy π_θ (where θ represents a vector of parameters), $J(\pi_\theta)$ represents the expected discounted sum of rewards or the performance objective under the policy π_θ, and Q^π represents the action-value function of the policy π_θ.

An important practical value of the stochastic policy gradient theorem is the reduction of computation of the gradient of the performance objective to an expectation. Given this theorem, a wide variety of practical algorithms have been derived by estimating the expectation using one or more samples. The key difference between these algorithms is the particular method used to estimate the action-value function Q^π.

The well known REINFORCE [296] algorithm uses the returns of full episode trajectories (i.e., the *monte-carlo* returns) to update the parameters of the policy. The gradient update for REINFORCE uses the following modified form of Eq. 2.5 [257, 256],

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T-1}\Big[\nabla_\theta \log \pi_\theta(a_t|s_t)\gamma^t G_t\Big] \tag{2.6}$$

where $G_t = \sum_{t'=t+1}^{T} \gamma^{t'-t-1} r_{t'}$ and γ is the discount factor[2].

The REINFORCE algorithm is known to have a high variance in practice [131]. To mitigate this problem, an equivalent expression to Eq. 2.5 is used, that includes a baseline function denoted as $b(s)$ (function that only depends on the state and independent of the action). Generally the state-value function is used as the baseline. Let the state-value function be $V_w(s)$, where w represents a vector of parameters. This baseline can be used with REINFORCE to provide the REINFORCE with baseline update, expressed as,

[1]We superscript functions by π instead of π_θ to simplify notation.

[2]Strictly speaking, Eq. 2.6 should use an approximation instead of an exact equality since an expectation is approximated using samples. However, we still use the equality (assuming the availability of infinite samples in the limit) as done by previous works [228]. We will continue to use the equality for Eq. 2.7 – Eq. 2.9 as well.

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T-1} \Big[\nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t [G_t - V_w(s_t)] \Big]. \tag{2.7}$$

Actor-critic algorithms are a generalized version of policy gradients which also use the state-value function as the baseline, where this function acts as a "critic" to the actions of the policy denoted as the "actor". Actor-critic algorithms consider the temporal difference update to bootstrap learning instead of the monte-carlo update used by the REINFORCE methods. This can be seen as a single sample estimate (as opposed to a single trajectory estimate in REINFORCE methods). The actor-critic update leads to algorithms that have reduced variance [256]. The modified stochastic policy gradient update is provided by the expression [3],

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T-1} \Big[\nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t (r_t + \gamma V_w(s_{t+1}) - V_w(s_t)) \Big]. \tag{2.8}$$

An alternate means is to use the advantage function which represents the difference between the action value function and the state value function [131]. The advantage function only focuses on which actions are better than the others, rather than obtaining an exact estimate of each action for comparison. This function is then used to critique the actions of the policy (actor). The updated form of Eq. 2.5 using the advantage function is provided by,

$$\nabla_\theta J(\pi_\theta) = \sum_{t=0}^{T-1} \Big[\nabla_\theta \log \pi_\theta(a_t|s_t) \gamma^t \Big(r_t + \gamma \max_a Q_w(s_{t+1}, a) - \sum_a \pi_\theta(a|s_t) Q_w(s_t, a) \Big) \Big]. \tag{2.9}$$

Further, under continuous action spaces, Silver et al. [228] provides a deterministic policy version of Eq. 2.5 which is known as the *deterministic policy gradient theorem*. This is expressed as,

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_{s \sim \rho^\mu} \Big[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s, a)|_{a=\mu_\theta(s)} \Big], \tag{2.10}$$

where $\mu_\theta : \mathcal{S} \rightarrow A$ is a deterministic policy and ρ^μ represents the (discounted) state distribution under the deterministic policy μ_θ (where θ represents a vector of parameters).

[3]As shown by Sutton et al. [257] a compatible function approximator with parameter w can be used to approximate the true action value function Q^π without any bias.

Also, Q^{μ} denotes the action-value function under the policy μ_θ, and $J(\mu_\theta)$ represents the expected discounted sum of rewards or the performance objective under the policy μ_θ. Note that in Eq. 2.10, functions use μ as superscript instead of μ_θ to simplify notation. The deterministic policy gradient theorem has been extended to large state-action space environments by using neural networks as the function approximator [147]. This gives rise to the *deep deterministic policy gradient* (DDPG) algorithm [147], which is a strong baseline in environments having continuous action spaces (more details are in Section 2.2).

2.2 Deep Reinforcement Learning

The major limitations of tabular based RL methods are that they are generally not applicable to high dimensional state and action space environments. The tabular methods have typically, linear space requirements in the number of states and polynomial space requirements in the number of actions. In recent years, deep neural networks have proved to be good function approximators [32], that can readily approximate the Q-values or the policies using networks weights. In deep reinforcement learning (DRL), the functions approximated using the neural network are typically parameterized by weights. For example, the standard Q-function is parameterized as $Q(s, a; \theta)$, where θ is the weights of the neural network. Since the space complexity of a neural network is constant in the number of states and actions, this function approximation makes RL applicable to large environments with high dimensional state and action spaces. Additionally, other functions approximations proposed for RL, suffer from the need for hand crafted features to represent the state [141]. Deep neural networks tend to avoid such problems.

A good example of a value-based method using deep neural networks is the deep Q-learning (DQN) [167]. The DQN algorithm extends the vanilla Q-learning algorithm discussed in Section 2.1.1 to large state-action spaces, by using neural networks to approximate the Q-function. The DQN algorithm introduced three major techniques for obtaining good results while using neural networks along with value based methods.

- In the first technique, DQN uses an *experience replay buffer* that stores experience tuples of the form $< s, a, r, s' >$ and uses these experience tuples for training the network weights.
- The second technique, involves the use of a *secondary target network* in addition to the primary evaluation (eval) network, that stabilizes the training. Each of these networks take in the state as the input and provides the Q-values of all the actions as

the output. The weights of the target network are replaced using the weights of the eval network once every fixed number of learning iterations.

- The third technique is the use of *convolutional layers* that can process a game's input screen and provide an encoding to serve as the state value that is provided as an input to the eval network.

DQN minimizes the loss (mean square error) between the target Q estimate and the eval Q estimate to train the primary eval network. Let the eval network parameters be represented by π and the target network parameters be represented by ϕ. The loss uses the temporal difference Bellman error: $L(\phi) = \frac{1}{K}\sum(r + \gamma \max_a Q_\pi(s', a) - Q_\phi(s, a))^2$ from Eq. 2.3, to update the parameters of the eval network. Here K represents the number of sample experiences (from the replay buffer) considered for the update.

The deep deterministic policy gradient [147] (DDPG) method extends the deterministic policy gradient algorithm by using deep neural networks. This algorithm is off-policy and model free much like DQN. However, since the algorithm optimizes the policy space directly, this algorithm applies to continuous state and action spaces, in which the DQN algorithm is not applicable. The other techniques introduced in DQN, like using experience replay and target networks, are retained in the DDPG algorithm too. DDPG uses the polyvak averaging technique, to slowly update the target network instead of just copying over the parameters from the eval net, every fixed number of steps, as done in DQN. The target network parameters updated under polyvak averaging is provided by $\pi = \tau\phi + (1-\tau)\pi$, where $\tau \in (0, 1]$. This kind of update can also be integrated into DQN [123].

The actor critic method has a popular synchronous algorithm called Advantage Actor-Critic (A2C) method (refer to Eq. 2.9) as well as an asynchronous version called as the Asynchronous Advantage Actor-Critic (A3C) [166]. The A3C method parallelizes training using multiple threads of a CPU, where each thread is a "worker" that can interact with the environment, collect experience tuples, and locally compute gradients for the policy. All these local gradients are finally passed to the global network which is updated. Since this uses an on-policy update, this algorithm does not maintain an experience replay buffer. A more recent method known as Actor Critic with Experience Replay (ACER) uses an off-policy learning scheme to make use of the experience replay technique [290]. Another algorithm known as Importance Weighted Actor-Learner Architecture (IMPALA) [63], uses entire trajectories of experiences to synchronize with a centralized learner.

A completely different class of methods choose a *trust region* or a policy search space (by restricting the length of the step-size) and then determine a point of improvement within this trusted region. There are two well-known algorithms in this line of work. The first is trust region policy optimization (TRPO) algorithm that considers a constrained optimization objective, where the trust region constraint is effectively a constraint on the KL divergence distance between the old policy and the new policy at each time step [215]. The second is the proximal policy optimization (PPO) algorithm that uses a simpler unconstrained optimization objective which directly constraints the ratio between the old and the new policies without imposing any hard constraints or using the KL divergence distance [216]. PPO is a first-order method that uses a clipped surrogate objective instead of the hard KL divergence constraint in TRPO (second-order method). Empirically, PPO has been found to be easier to implement, faster to train, and having a performance that is at least as good as TRPO in most environments [216]. Subsequently, PPO has emerged as a strong baseline for continuous control environments.

2.3 Stochastic Games And Multi-agent Reinforcement Learning

Our research will focus on general sum stochastic games. We will present the foundational concepts of these games along with a brief review of papers that introduced these ideas. We will also present some definitions and notations from these works that we use in our research.

Multi-agent Reinforcement Learning (MARL) involves environments with more than one agent, all learning simultaneously. This adds additional layers of complication since the optimal policy of an agent depends on the policies followed by the other agent(s) as well. A standard framework for MARL is *stochastic games*, a generalization of both MDPs from single-agent learning and repeated games from game theory which model agents' interactions. We formally define a stochastic game as follows,

Definition 2. *A stochastic game is defined as* $\langle \mathcal{S}, N, \mathbf{A}, P, \mathbf{R}, \beta \rangle$ *where* $\mathcal{S}$ *is a finite set of states,* N *is the finite set of agents,* $|N| = n$*, and* $\mathbf{A} = A^1 \times \ldots \times A^n$ *is the set of joint actions, where* A^i *is the finite action set of an agent* i*, and* $\boldsymbol{a} = (a^1, \ldots, a^n) \in \mathbf{A}$ *is the joint action where an agent* i *takes action* $a^i \in A^i$*. Furthermore,* $P : S \times \mathbf{A} \times S \mapsto [0, 1]$ *is the transition function,* $\boldsymbol{R} = \{R^1, \ldots, R^n\}$ *is the set of reward functions, where* $R^i : S \times \mathbf{A} \mapsto \mathbb{R}^n$ *is the reward function of the agent* i*, and* β *is the discount factor* $(0 \leq \beta < 1)$ [4].

In a stochastic game, the common assumption is that all agents share the same set of states $\mathcal{S}$ (where $\mathcal{S}$ is the state space), which contains information about all agents

[4]In some chapters we will use γ to denote the discount factor. We will clarify the notations, as appropriate, everywhere.

Game	Left	Right
Up	10,8	0,4
Down	2,0	-1,3

Table 2.1: Example of a stage game having two players. Each player can choose to perform one of the two available actions.

participating in the stochastic game [219]. The environment provides the (global) state to each agent participating in the stochastic game. At each time step t, each agent i observes the current state $s \in \mathcal{S}$ and takes a local action $a^i \in A^i$ (where A^i is called as the action space of the agent i). Subsequently, the agent obtains a reward r^i according to its reward function R^i, and the joint action $\boldsymbol{a}$ of all agents in the environment at state s. The transition function, P, determines the transition of the environment to the next state, s'. This transition depends on the current state (s) and the joint action ($\boldsymbol{a}$) of all agents in the environment. Further, the transition function is fixed and satisfies the constraint, $\sum_{s' \in \mathcal{S}} P(s'|s, \boldsymbol{a}) = 1$ for all $s \in \mathcal{S}$ and $\boldsymbol{a} \in \boldsymbol{A}$. Given a stochastic game, a *joint policy* is represented by $\boldsymbol{\pi} = (\pi^1, \dots, \pi^n)$, where π^i is the stochastic policy followed by an agent i. As in the single-agent setting, each individual agent tries to maximize their value function. However, this optimization depends on the policies of others: $v^i(s, \pi^1, \dots, \pi^n) = \sum_{t=0}^{\infty} \beta^t \, \mathbb{E}(r^i_t | \pi^1, \dots, \pi^n, s_0 = s)$.

There are several formulations of the stochastic game model. The *general-sum* model is the most general formulation, where the rewards that an agent receives at any time step can be related to the rewards obtained by other agents in an arbitrary fashion. Special cases of general-sum games are *zero-sum* games that restrict the sum of rewards obtained by all the agents at any time step to be zero, and identical interest *coordination* games that require all agents in the environment to obtain the same numerical reward at every time step. We will use the general-sum formulation in this dissertation. Additionally, as introduced in Littman [150], we consider stochastic games that are incomplete games with perfect information. Here, every agent can observe the previous action and resulting rewards obtained by all agents and respond to it, however, they do not have access to the transition dynamics of the environment or the reward functions of other agents.

It is useful to think of a stochastic game as a multi-period stage game. In a stage game, agents select an individual action and then receive some (possibly different) payoff, which depends on the joint action taken. An example of a two-player stage game is provided in Table 2.1. Here there are two agents (row-player and column-player), and each of these players can choose one of the two available actions. The row player can choose to move 'Up' or 'Down', and the column player can choose to move 'Left' or 'Right'. Given the actions of the two agents, the Table 2.1 provides the payoff for each agent. The row player's payoff are given in the first numbers of each cell in Table 2.1, and the column player's payoff are given in the second numbers of each cell in Table 2.1. For example, if the two agents choose to perform the actions 'Up' and 'Left', then the row player will receive a payoff of 10 and the column player will receive a payoff of 8. The stage game can be formally defined as follows.

Definition 3. *An n-player stage game is defined as $(\mathbf{A}, M^1, \ldots, M^n)$, where $M^k : \mathbf{A} \mapsto \mathbb{R}$ is agent k's payoff function, specifying a payoff for agent k for each possible joint action $(a_1, \ldots, a_n) \in \mathbf{A}$.*

The main solution concept we are interested in is the *Nash equilibrium* [172], namely a stable point in the joint policy-space. We first formally define a Nash equilibrium in a stage game, before moving on to the generalization of this concept in stochastic games. We switch terminology slightly and will refer to agents' strategies to be consistent with the game-theoretic literature, although we can use strategy and policy interchangeably. In particular, an agent's strategy in a stage game is simply a probability distribution over actions, given the underlying state s. Let ϕ^k be the strategy of agent k in the stage game and ϕ^{-k} be the product of strategies of all agents other than k, $\phi^{-k} \triangleq \phi^1 \cdots \phi^{k-1} \cdot \phi^{k+1} \cdots \phi^n$.

Definition 4. *A joint strategy $(\phi^1, \ldots, \phi^n)$ can be considered as a Nash equilibrium for the stage game $(\mathbf{A}, M^1, \ldots, M^n)$, for $k = 1, \ldots, n$, if*

$$\phi^k \phi^{-k} M^k \geq \hat{\phi}^k \phi^{-k} M^k, \quad \textit{for all } \hat{\phi}^k \in \phi(A^k) \tag{2.11}$$

where $\phi(A^k)$ is the set of all probability distributions over A^k.

The term $\phi^k \phi^{-k} M^k$ is a scalar value. The product of strategies denotes the product of probabilities of taking specific actions by an agent. This is multiplied with the value of that action as denoted by M^k. The dimensionality of M^k is equal to the action space of the agent k (i.e., $|A^k|$). Going back to the example in Table 2.1, $(10, 8)$ represents a Nash equilibrium point at this stage game. This is obtained when the row player is playing strategy $[1, 0]$ (i.e., action 'Up' with probability 1 and action 'Down' with probability 0), and the column player is playing strategy $[1, 0]$ (i.e., action 'Left' with probability 1 and action 'Right' with probability 0).

For a stochastic game, the strategies for an agent apply to the entire time horizon of the game.

Definition 5. *In a stochastic game Γ, a Nash equilibrium is a tuple of n strategies $(\pi^1_*, \ldots, \pi^n_*)$, such that for all states $s \in \mathcal{S}$ and agents $i = 1, \ldots, n$,*

$$v^i(s, \pi^1_*, \ldots, \pi^{i-1}_*, \pi^i, \pi^{i+1}_*, \ldots, \pi^n_*) \leq v^i(s, \pi^1_*, \ldots, \pi^n_*) \tag{2.12}$$

for all $\pi^i \in \Pi^i$ where Π^i is the set of strategies available to the agent i.

That is, no agent has incentive to unilaterally change their strategy in a Nash equilibrium. Now, we define the Nash Q-function [104] as follows,

Definition 6. *Agent i's Nash Q-function is defined as the sum of the agent i's immediate reward and its discounted future rewards when all agents follow a joint Nash equilibrium strategy $(\pi_*^1, \dots, \pi_*^n)$*

$$Q_*^i(s, \boldsymbol{a}) = r^i(s, \boldsymbol{a}) + \beta \textstyle\sum_{s' \in \mathcal{S}} P(s'|s, \boldsymbol{a}) v^i(s', \pi_*^1, \dots, \pi_*^n) \tag{2.13}$$

where $r^i(s, \boldsymbol{a})$ is the immediate one-stage reward of the agent i at state s and the corresponding joint action $\boldsymbol{a}$, and $v^i(s', \pi_*^1, \dots, \pi_*^n)$ is the agent i's total discounted reward over infinite periods starting from s', given that all agents follow the joint equilibrium strategy.

The Q-values of the Nash Q-function are denoted as the *Nash Q-value* in Hu and Wellman [104]. Finally, we define an approximate Nash equilibrium concept, an ϵ-equilibrium, which bounds the benefits of agents' deviations from the joint Nash equilibrium strategy.

Definition 7. *In a stochastic game Γ, a joint strategy $(\pi_{*'}^1, \dots, \pi_{*'}^n)$ is an (ϵ)-equilibrium if it satisfies (for all $\pi^{i'} \in \Pi^i$ and $\forall s$)*

$$v^i(s, \pi_{*'}^1, \dots, \pi_{*'}^{i-1}, \pi^{i'}, \pi_{*'}^{i+1}, \dots, \pi_{*'}^n) - v^i(s, \pi_{*'}^1, \dots, \pi_{*'}^n) \leq \epsilon \tag{2.14}$$

As seen by the definition of the Nash equilibrium (Definition 5), given the strategies of the other agents, the Nash Equilibrium guarantees that any given agent is obtaining the best possible payoff. This is the best guarantee we can provide in a general-sum stochastic game setting with fully independent agents [104], without any restrictions on the nature of the environment or the agent. Hence, we choose to use the Nash equilibrium as our solution concept.

2.3.1 Theoretical MARL Literature

There are several fundamental challenges in theoretical MARL beyond those of the single-agent RL setting. The most important is the problem of having non-unique learning goals in MARL. Though the most common learning goal has been convergence to a Nash equilibrium as in Definition 5 [222], other learning goals such as the convergence to a cyclic equilibrium (agents can cycle through stationary policies) [324], and no-regret based learning (as compared to the best possible stationary policy) have also been investigated [29, 30]. Under general-sum settings with two or more fully independent agents, the Nash equilibrium remains to be the best possible global guarantee and hence, the most useful solution concept [104]. However, it is common to have multi-agent environments that have more than one Nash equilibrium point. In this case, restrictive assumptions on the nature of the stochastic game [104] and/or nature of the reward function (such as linear-quadratic games) [5] are required to ensure the existence of a unique solution concept, which provides a suitable point for the independent agents to converge.

In cooperative settings that have global rewards (all agents obtain the same reward), convergence to a global optimum for cooperation has been established for Q-learning based methods [262]. This global optimum constitutes a Nash equilibrium of the cooperative game [313]. In zero-sum competitive settings, minimax updates (each agent chooses an action that maximizes the smallest value it can receive) [284] can be used to provide Bellman equations with unique solutions that constitute the Nash equilibrium of the game [219]. This equilibrium point can be found using linear programming techniques [281]. In general-sum settings, Hu and Wellman [104] provide a convergence guarantee for Q-learning methods (with Nash equilibrium as the solution concept), under a set of restrictive assumptions. Specifically, Hu and Wellman [104] proved that the Q-updates of an agent j, using the Nash payoff at each stage eventually converges to its Nash Q value (Q^j_*), which is the action-value obtained by the agent j when all agents follow the joint Nash equilibrium policy for infinite periods (refer Definition 6). Utility maximizers of rational agents were shown to converge to a Nash equilibrium in infinitely repeated games by Kalai et al. [122]. Singh et al. [230] showed that gradient ascent on the expected payoff in a simple two player, two action, general sum repeated game, converges to the Nash payoff. Bowling [29] introduced a method that combines the principles of convergence and no regret learning in multi-agent systems. In general, proving theoretical guarantees in MARL requires information structures that capture global information about the policies/strategies of all other agents [313]. Hence, almost all works mentioned in this sub-section use centralized training. Some recent works have also provided finite-time analyses, determining lower/upper bounds on the sample efficiency of MARL algorithms in stochastic games [154, 234].

2.3.2 Empirical MARL Literature

In Table 2.2 we summarize some important empirical MARL algorithms and provide the salient features of each algorithm. All the algorithms can be classified to be either value based or policy gradient based methods, and they work in either fully cooperative, fully competitive or cooperative-competitive settings. The structure of this table is based on the work by Hernandez et al. [98]. Though there have been many recent developments in the field of empirical MARL (especially deep MARL), we will restrict our discussion to works that are (broadly) relevant to this dissertation. We refer the reader to Hernandez et al. [98] for a more comprehensive survey of the field.

The most common approach in empirical MARL is to directly reuse strong algorithms from the single-agent RL literature. This approach has been explored extensively in multi-agent settings, though this violates the Markovian stationarity assumptions in the RL algorithms [98] (as discussed in Chapter 1). Particularly, the algorithms use decentralized learning and are called independent learners. Independent Q-learning (IL) [264] is a popular algorithm in this space. Here each agent learns its own policy and assumes that other agents are simply a part of the environment. No explicit modelling of opponents is done in any way. These algorithms are easily scalable to large environments with many agents and have other performance advantages in simple environments [160]. One caveat is that these algorithms can completely fail when the opponents are learning adaptive strategies [223], since this breaks the assumptions of stationarity.

Deep learning has also been effectively combined with MARL to give good performances. In the cooperative settings, communication has been an important area of research. Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-agent Learning (DIAL) are two strong methods that use deep neural networks to communicate messages in a MARL setting [71]. Memory derived communication has also been tried in conjunction with policy gradient methods [156]. Partially observable cooperative domains were considered by Gupta et al. [89], who used parameter sharing between agents to enhance learning. Only one global network is trained between all the cooperating agents. Lowe et al. [156] extended the well-known RL algorithm, Deep Deterministic Policy Gradients (DDPG) to the space of MARL by introducing the Multi-Agent Deep Deterministic Policy Gradients (MADDPG), which trains a centralized critic with local actors. MADDPG was shown to work better than many baseline RL algorithms in both cooperative and competitive settings [156]. The Deep Reinforcement Opponent Network (DRON) was introduced by He et al. [95]. This network is very similar to the DQN, but has an extra state variable to denote the approximate policy of the opponent. This approximate policy of the opponent is the policy that the agent "thinks" the opponent is playing. The Q values of the actions of an agent are jointly modelled along with the opponent policy. The DRON contains a Q-network that learns actions for a given state and a opponent network that learns a representation of the opponent policy from observations. He et al. [95] also introduced the DRON-concat model. In this algorithm the extracted features from the Q network are embedded in separate hidden spaces using convolutional networks. The concatenation of a Q-network and the opponent policy is jointly used to predict Q-values. Foerester et al. [72] introduced the

Algorithm	Summary	Setting	Training Protocol	Method
RIAL [71]	Message passing using recurrent networks and parameter sharing	Cooperative	Centralized	Value based
DIAL [71]	Message passing by sharing gradients	Cooperative	Decentralized	Value based
MADDPG [156]	Centralized critic uses information about others, decentralized actors use local information	Mixed	Centralized	Policy gradient based
DRON [95]	Train separate network for modelling opponent(s)	Mixed	Decentralized	Value based
LOLA [72]	Incorporate a one step look-ahead of policy of other agent(s)	Mixed	Decentralized	Policy gradient based
PSRO [134]	Approximate best responses to mixtures of opponent strategies	Cooperative and Competitive	Decentralized	Policy gradient based
PS-DQN / PS-TRPO [89]	Parameter sharing for emergent cooperative behaviour	Cooperative	Centralized	Value/Policy gradient based
MFQ [303]	Best responses only towards the mean field action using Q-networks	Mixed	Centralized	Value based
MFAC [303]	Best responses only towards the mean field action using actor-critics	Mixed	Centralized	Policy gradient based
Independent Q-learning [264]	All other agent(s) assumed to be part of the environment	Mixed	Decentralized	Value based

Table 2.2: Table to capture key features of some important deep MARL algorithms in the literature.

Learning with Opponent Learning Awareness (LOLA) algorithm. LOLA is a method in which each agent shapes the anticipated learning of the other agents in the environment. The learning rule includes an additional term that accounts for the impact of one agents' policy on the anticipated parameter update of the other agents. LOLA with policy gradient and opponent modelling is used in this study. LOLA uses one-step look ahead of opponent learning. This is done by using the expected reward after the opponent updates its policy with one naive learning step and incorporating this update into the expectation. The learning rule updating is done using approximation of derivatives which uses policy gradients. The algorithm assumes self-interested agents and the agents are allowed to learn cooperation or competition strategies with opponents based on the game situation. This algorithm does not assume the nature of opponents (competitive or cooperative) at compile time, but learns emergent cooperation or competitive strategies during game play. Another strong algorithm in MARL literature is the Policy Space Response Oracle (PSRO) introduced by Lanctot et al. [134]. This algorithm computes meta strategies for policy selection based on approximate best responses to mixtures of policies of opponents. A new meta strategy is calculated at intermittent times during training. These are developed as tit-for-tat responses to the strategies the opponent chooses to play at different times during game play. Thus, this algorithm develops best response policies for every policy the opponent chooses to play. The game starts with a meta agent with a single random policy. The policy is refined further and more policies are added during the game play.

There are also many other recent publications in the field of DRL and multi-agent systems [14, 142, 263] that have applied the empirical advances from RL to the multi-agent space. Unfortunately, almost all of them, do not provide any convergence guarantees or performance guarantees beyond a small class of problems or environments. Since such "black-box" solutions are generally unacceptable for many safety critical application domains like fighting wildfires, we will focus on giving theoretical bounds and guarantees while integrating the empirical advances in our approaches.

As a general rule, almost all the MARL algorithms discussed so far have an exponential dependency in the number of agents in both space and time [98, 104]. This makes these algorithms intractable in environments with many agents as is typically seen in large real world settings like wild fire fighting [115] and smart grid utility management [231]. One exception is a class of methods that use the mean field theory [15] for scaling algorithms to large environments. Under the assumptions of homogeneity and presence of a very large number of agents, mean field methods aggregate all the other agents in the environment into a single virtual agent [136]. Every agent calculates best-responses only to the mean field and this makes MARL algorithms tractable, since effectively the environment is reduced into a two-agent environment which makes these algorithms have a constant dependence on the number of agents (in both time and space) [303]. Two strong algorithms in this space are the mean field Q-learning (MFQ) and the mean field actor-critic (MFAC) algorithms introduced by Yang et al. [303]. In spite of having this scalability advantage, these mean field methods contain several restrictive assumptions such as requiring: full homogeneity, full observability, and centralized training protocols [304], that severely restrict their ability to be applied in practical environments.

Additionally, MARL algorithms, in general, suffer from the issue of sample efficiency [98], with very few papers in the past decade addressing this problem directly [132]. However, some notable research has been done in the space of single-agent RL to address the issue of sample efficiency. One popular approach has been to use human expert demonstrations to speed up the learning process [268, 289]. Another approach that was suggested is the use of hierarchical reinforcement learning. This method uses action abstractions and divides the tasks into sub-tasks for effective learning [56]. Action reuse from sub-optimal policies proved to be a useful solution [65]. More recently, Hester et al. [100] have released a DQN variant known as deep Q learning from demonstrations (DQfD) that has shown good performances in many standard benchmarks. This algorithm pre-trains the Q-network using demonstration data before actual interaction with the environment. Li et al. [145] introduce an idea of using two levels of Q learning updates to learn effectively from multiple conflicting experts with varying expertise [145]. Despite this progress, lots of work need to be done to make RL truly sample efficient, as even after using these methods, DRL is reported to need large samples for learning effectively in large environments [311]. For example, around 200 millions frames are needed by A3C and DQN algorithms to provide human level performances in Atari games [166, 167]. Now, multi-agent deep RL (MDRL) requires even more samples than DRL to learn effectively since the non-stationary opponent/opponents have to be encoded in the state space. Despite this situation, Hernandez et al. [98] report that most MDRL papers do not report computational resources like CPU/GPU usage and wall-clock computation time, making it difficult to predict the potential applicability of all of these algorithms in real world scenarios.

2.4 Conclusion

In this chapter, we have given some important background and prior work that are most related to our work. The challenges in MARL discussed in this chapter ties well with our fundamental research aims discussed in Chapter 1.

Chapter 3

Multi-Agent Advisor Q-Learning

In the last decade, there have been significant advances in multi-agent reinforcement learning (MARL) but there are still numerous challenges, such as high sample complexity and slow convergence to stable policies, that need to be overcome before wide-spread deployment is possible (as discussed in Chapter 1). However, many real-world environments already, in practice, deploy sub-optimal or heuristic approaches for generating policies. An interesting question that arises is how to best use such approaches as *advisors* to help improve reinforcement learning in multi-agent domains. In this chapter, we provide a principled framework for incorporating action recommendations from online sub-optimal advisors in multi-agent settings. We describe the problem of *ADvising Multiple Intelligent Reinforcement Agents* (ADMIRAL) in nonrestrictive *general-sum stochastic game* environments and present two novel Q-learning based algorithms: **ADMIRAL - Decision Making (ADMIRAL-DM)** and **ADMIRAL - Advisor Evaluation (ADMIRAL-AE)**, which allow us to improve learning by appropriately incorporating advice from an advisor (ADMIRAL-DM), and evaluate the effectiveness of an advisor (ADMIRAL-AE). These two algorithms are extensively analyzed both theoretically and empirically.

The contributions of this chapter are summarized as follows:

- A principled framework for incorporating action recommendations from online sub-optimal advisors in multi-agent settings (specifically the non-restrictive general-sum stochastic games).
- Two practical Q-learning algorithms:
    - ADMIRAL-DM: Learns a best response policy in the multi-agent environment, with the aid of action recommendations from an external advisor.

  - ADMIRAL-AE: Evaluates an advisor in the given multi-agent setting.

- Theoretical fixed point guarantees, for both ADMIRAL-DM and ADMIRAL-AE, while learning in general-sum stochastic game environments with an arbitrary number of agents. Notably, these guarantees are provided under less restrictive assumptions as compared to prior works.

- Extensive experiments, illustrating that these algorithms:
  - Can be used in a variety of multi-agent environments.
  - Have performances that compare favourably to other related baselines.
  - Can scale to environments with large state-action spaces.
  - Robust to poor advice from advisors.

In this chapter we restrict each agent to have access to at-most one advisor during training (we will relax this in the next chapter). The core contents of this chapter are published in the Journal of Artificial Intelligence Research (JAIR) [255] and is available on arXiv [254].

3.1 Introduction

As discussed previously, RL research is growing and expanding rapidly, however, this method still finds only limited applications in practical real-world settings [58]. One major reason for this is that RL algorithms typically have high sample complexity and can learn effective policies only after experiencing millions of data samples in simulation [120]. Multi-agent reinforcement learning (MARL) extends RL to domains where more than one agent learn simultaneously in the environment [220]. Moving from single-agent to multi-agent settings introduces new challenges including non-stationary environments and the curse-of-dimensionality [98], while concerns from single-agent RL such as exploration-exploitation trade-offs and sample efficiency remain [307]. In MARL environments, it has been reported that learning complex tasks from scratch is even impractical due to its poor sample complexity [225]. In this regard, it becomes necessary for agents to obtain guidance from an external source to have any possibility of scaling up to real-world domains. Furthermore, during the early stages of learning, agents' policies may be quite random and dangerous, which makes it almost impossible to use them in real-world environments. Thus, it is hard to improve upon these policies by only using direct interactions with the environment. In this chapter, we aim to tackle the problem of improving sample efficiency in MARL through the use of other available sources of knowledge, particularly during the early stages of training.

In single-agent RL use of external knowledge sources such as *advisors* to drive exploration has been successful in a variety of domains. The advisors provide actions to the agent at different states to bootstrap learning by targeted exploration [170]. However, the biases of sub-optimal advisors pose a challenge to successful learning [79]. Further, many of these approaches do not directly extend to multi-agent environments due to the additional complications present in the multi-agent environments. Though learning from external sources of knowledge has been explored in multi-agent settings, many previous works assume the presence of fully optimal experts [90, 173, 310]. Generally, they entail additional assumptions such as having simplified environments with only two agents [148] and consider restrictive environments such as competitive zero-sum settings [286] or fully cooperative settings where all agents share a common goal [139, 173, 187]. Additionally, some approaches such as [148] restrict themselves to small multi-agent environments with discrete state and action spaces. The use of sub-optimal advisors in multi-agent general-sum settings with an arbitrary number of agents has been less explored, and to the best of our knowledge, there has been no comprehensive analysis of this approach, especially from a theoretical perspective.

3.1.1 Motivational Examples

We describe two motivational examples relevant to the goals of our chapter. These examples provide an intuition about the kind of practical problems where our approach can be used and clarify the potential impact of this line of research. The first example uses a cooperative wildfire response setting and the second example uses the competitive product marketing domain.

Motivational Example 1: Wildfire response is a complicated process that requires systematic planning of important resources and a good understanding of wildfire behaviour for proper estimation and combat [271]. The firefighters and fire managers need to make many critical decisions related to wildfire control, and on many occasions, these decisions could be the difference between life and death [272]. Additionally, these decisions have a high ecological impact. This is a multi-agent problem (multiple fire-fighters aim to fight fire) where artificial learning agents can learn suitable policies to aid in fire-fighting efforts [179]. Machine learning, particularly reinforcement learning, has a huge potential in this area but has been underutilized so far [116]. Notably, in these sustainability-based domains, there is a general paucity of data [279], since obtaining good quality high-resolution sensor data is expensive and hard. Hence, current state-of-the-art MARL algorithms are incapable of being used in such problems due to poor sample efficiency. Notably, current practical fire-fighting efforts use physics-based models [210] that help in predicting the spread of fires given current location and intensity. Despite being the current state-of-the-art, these models are sub-optimal, have low accuracies [113], and possess other problems like under-prediction bias and lack of generalizability to regions outside North America [51]. Thus, we have particular knowledge sources that are not optimal but are still used in practice (particularly due to lack of alternatives). Our work in this chapter will enable MARL training to use these physics-based models to speed up learning. The policies that the MARL algorithms will finally arrive at will have the potential to be better than these physics-based models since the MARL algorithms will also simultaneously learn from data.

Motivational Example 2: Multi-agent algorithms have the potential to learn from available data and formulate effective marketing and price management strategies to improve financial profit for companies [76]. However, the problem of poor sample efficiency prevents the usage of MARL for this problem, since many companies would find it difficult to procure sufficient data for MARL training. There are many mathematical marketing models in the literature that typically help companies formulate marketing strategies [62], however, these models are sub-optimal, with scope for improvement, especially in adapting to changing trends [241]. These models can serve as external knowledge sources that MARL training can leverage to learn good policies.

Hence, learning from possibly sub-optimal external content sources, which we broadly refer to as "advisors", is useful for MARL training. We formally introduce this problem and study it further in this chapter.

3.1.2 Related Work

Imitation learning includes various methods for learning the behaviour of advisors, the simplest being *behaviour cloning*, where supervised learning is used to mimic the advisor policy. This method dates back to the early 90s, where agents were shown to be successful in copying the behaviour of the demonstrator in autonomous driving tasks such as road following and perception [192, 193]. This method comes with certain theoretical guarantees, where prior works have conducted formal analysis and established that near-optimal advisors are the easiest to imitate, requiring far fewer demonstrations than sub-optimal advisors to achieve the same performance as the advisors being imitated [261]. Behaviour cloning is hard to generalize to different unseen environments since the agent is learning in a supervised fashion [126]. Prior works in the area of behaviour cloning have also introduced methods to detect and safeguard against a few noisy/bad demonstrations [109, 316], however, in general, the demonstrations are assumed to be near-optimal to enable learning reasonable policies. Further, it has been noticed that behaviour cloning methods are prone to a problem of *distribution drift*, where the trajectory distribution at the test time drifts away from the distribution learned from the advisor during training [198, 209]. In autonomous driving environments, on-policy data collection has been shown to mitigate this problem to some extent [212]. Some recent approaches propose off-policy solutions, along with techniques of expanding the input (image)-action space using data-augmentation methods [48, 135]. However, several other limitations like dataset bias and high variance in neural network-based solutions have been reported to limit the application of behaviour cloning to real-world environments [49].

Another popular imitation learning framework is *inverse reinforcement learning (IRL)* [176], where the objective is to learn the reward function from demonstrations. This framework typically assumes that the environment does not have a reward function and/or it is difficult to formulate good reward functions, but expert demonstrations of good behaviour are easier to obtain. A good example is autonomous driving, where formulating a complete reward function that covers all scenarios is hard while obtaining demonstrations of good driver behaviour is much simpler. Initial approaches to IRL used maximum margin methods where an initial estimate of the reward function for the demonstrator keeps being iteratively improved, such that the performance of the demonstrator is at least more than a "margin" of the previous reward estimate for the demonstrator [1, 200]. This iteration is repeated until no such improvement is possible. The problem with the maximum margin methods are their sensitivity to noise and imperfection in the demonstrator behaviour. To alleviate this problem, probabilistic approaches using principles of maximum entropy have been proposed for the IRL framework [27, 322]. These approaches reason over a set of possible behaviours, rather than monotonically improving upon estimates of reward or policies. Neural networks have also been considered to learn a suitable reward function, where convolutional networks aim to map the relationship between input images to final rewards [297]. Several techniques from supervised learning such as Gaussian processes [199] and ensemble methods [55] have also been used in the IRL paradigm [144, 201]. A recent approach, Generative Adversarial Imitation Learning (GAIL), aims to recover the policy of the expert directly instead of extracting an explicit reward function using Generative Adversarial Networks (GANs) [102]. Since GAIL is not learning a reward function, it may not be considered an IRL technique and since it is not learning in a supervised fashion it may not be considered a behaviour cloning technique as well. This approach opened up a new class of methods in the intersection of imitation learning and generative adversarial networks [133, 165]. Some of these approaches aim to extract an explicit reward function from the demonstrations using GANs [68, 73] and these can be considered to fall within the IRL framework.

The DAGGER algorithm introduced by [209] is yet another imitation learning method. This algorithm has strong guarantees of performance while learning stationary deterministic policies in environments with an online advisor that can be queried interactively for additional feedback. However, the major aim of this algorithm is to obtain a policy that guarantees "no-regret" under its induced distribution of states and does not aim to improve upon the provided demonstrator. This method belongs to a wider set of online algorithms that aim to provide the no-regret guarantee while learning based on demonstrations from a perfect advisor [44, 94, 218].

In general, imitation learning methods assume fully optimal or near-optimal advisors (or *experts*), with the major goal being to copy the policy or behaviour of the experts. Since MARL problems are non-stationary, there is little expectation of obtaining perfect experts. Rather, we expect guidance on action choices that aid agents in learning and improving over time. Further, several multi-agent imitation-based methods in the literature are restricted to cooperative games [17, 26] or games with strict restrictions on the nature of the reward function (such as having linear relations to some underlying feature) [202, 292], in addition to assuming the availability of (near) optimal experts. On the other hand, other approaches based on IRL in multi-agent settings are restricted to zero-sum games [149, 286]. Some recent approaches aim to apply IRL in general-sum games [233, 310], however the assumption of availability of perfect experts are present in these works as well. A different approach from [194] studies imitating more experienced peers in a multi-agent setting. However, this work considers a very restrictive environment, where each agent's dynamics is independent of other agents. Further, strict assumptions on the reward function exist, such as obtaining the same numerical reward over a part of the state space and having an independent reward function that does not depend on other agents' actions. Such assumptions are hard to verify in real-world multi-agent environments. Pure imitation methods generally lack the ability to exceed the performance of the available expert/advisor.

Another approach, *Learning from Demonstrations (LfD)*, combines the imitation-based behaviour cloning approach of learning from expert demonstrations and the reinforcement learning-based approach of directly learning from the environment to reach a suitable goal. Here, the objective is not to simply perform imitation learning, but to use imitation learning as a bootstrap mechanism that can speed up the training of RL agents. The RL algorithm enables further fine-tuning of the policy learned from imitation, which provides an opportunity for improving upon the performance of the advisor and learning goal-oriented policies. The algorithms in this approach use the environmental rewards along with expert/advisor demonstrations collected offline. Unlike the IRL paradigm, the environmental rewards are assumed to be available, and the rewards need not be extracted from suitable expert demonstrations. Our work in this chapter is most related to the LfD framework. Early works in this area studied the LfD approach using model-based reinforcement learning, which found applications in classical RL environments like cart-pole [213] and robot arm learning to balance a pendulum [11]. More recently, the model-free RL approaches gained prominence, especially after the exceptional performance shown by Deep Q-learning (DQN) on Atari games [167]. Model-free RL using replay buffers for training have been successful in the LfD framework as well [46, 191]. One state-of-the-art algorithm, *Deep Q-learning from Demonstrations (DQfD)* [100] pre-trains the agent using demonstration data, keeps this data permanently for training, and combines an imitation-based hinge loss with a temporal difference (TD) error. This additional loss helps DQfD learn faster from demonstrations, but also makes DQfD prone to the problem of overfitting to the demonstration data. *Normalized Actor-Critic (NAC)* [119] drops the imitation loss and hence is more robust to imperfect demonstrations (from bad, almost adversarial advisors) than DQfD. However, we find that the performance of DQfD is at least as good as NAC for reasonable advisors (due to the imitation loss). Goecks et al. [83] introduce the Cycle-of-Learning (CoL) algorithm that provides a novel LfD mechanism in which additional human inputs can be obtained during training in environments where humans are present in the loop to help agents train faster. The agents can make use of the human feedback in addition to the demonstration from other sources for training. Notably, LfD algorithms have found numerous applications in robotics. Some early applications have focussed on teaching primitive movements of motors to policy gradient based RL algorithms that can learn to perform a suite of relatively simple robotic tasks, such as manoeuvring gaps [189, 270]. Recently, [198] introduce an effective algorithm that can learn highly complex dexterous manipulation such as object relocation and in-hand manipulation in response to sensor inputs. They introduce an algorithm, Demonstration Augmented Policy Gradient (DAPG) that uses an on-policy policy gradient [257] update as opposed to the off-policy nature of prior approaches such as Hester et al. [100]. Zhu et al. [321] provides yet another approach that uses the LfD technique for robot manipulation tasks such as block lifting, block stacking and pouring liquids, where the agents need to learn effective visuomotor policies that can take actions in response to image inputs from a camera or sensor. The well-known RL algorithm, Deep Deterministic Policy Gradients (DDPG) [147] has been adapted to the LfD framework to incorporate human demonstrations and learn continuous control object manipulation robotic tasks such as peg-insertion and clip-insertion in both real and simulated environments [283].

While powerful, the requirement for offline demonstrations commonly seen in prior works on LfD is not a good fit for MARL. In MARL, since the environments are non-stationary with dynamic opponents, real-time action advising would be more useful as the advisors can teach agents to adapt to changing opponents. Recently, some multi-agent approaches have used the LfD method, where the algorithms can, in-theory, do without fully optimal advisors [105, 227, 286]. These works, however, are applicable only to a restrictive set of MARL environments. The Alpha-Go approach from [227] and the approach from [286] are restricted to zero-sum competitive games and cannot naturally extend to general-sum games. The work by [105] is designed for a very particular application (StarCraft Micromanagement), where the authors require the availability of specialized human-made opponents that contain specific pre-defined tactics about game-playing. In MARL, prior works have also studied peer-to-peer teaching, where each agent can learn when and what advice needs to be extended to peers in addition to learning how to use the available advice and improve its own learning [183, 226, 305]. The agents can switch between the roles of student or teacher at different points of time based on the situation. However, as evident from the setting, this method is only applicable to fully cooperative environments.

Expert demonstrations have also been used for *reward shaping* in single-agent RL [138], but this undermines the convergence guarantees of Q-learning based algorithms. Using prior knowledge to define a potential function over the state space provides an approach known as potential-based reward shaping, which preserves the total order over policies and does not affect the convergence guarantees [175, 295]. The approach of reward shaping has been popular in single-agent RL [158], and very recently adopted to the MARL setting as well [54, 77, 298]. The work by [77] uniformly redistributes the rewards accumulated at the end of a trajectory, to each state-action pair along the length of the trajectory. This approach is based on independent learning, which hurts convergence guarantees in the MARL setting [265]. Although it shows good empirical performance in single-agent tasks, this approach performs poorly in many MARL tasks, since the credit-assignment is not always accurate [298]. On the other hand, the approach by [298] adapts potential-based reward shaping to the MARL setting. There are two important limitations of this potential-based reward shaping approach, formulated by [298] in MARL. The first is that the shaping advice is a heuristic that needs to come from an expert who has complete prior knowledge about the entire problem domain and is capable of designing these heuristics. Obtaining such experts for complex MARL tasks is not always possible. Second, the shaping advice is provided at the beginning and then fixed for the duration of training. However, MARL requires adaptive advising at different parts of the state based on opponent behaviour.

Another group of methods such as *human agent transfer* (HAT) [268] aim to summarize limited offline demonstrations (from sources like humans) into decision tree-based expert rules that boost learning online. *Confidence-based human-agent transfer* (CHAT) [289] improves HAT by adding confidence measurements to safeguard against bad demonstrations. Both these methods demonstrate good performance in a multi-agent "Keep Away" game, although the algorithms themselves are single-agent only. These algorithms are independent methods that consider the other agent(s) as part of the state and do not track opponent actions or perform any kind of opponent modelling. In MARL environments, changes in opponent behaviour play a crucial role in determining the agent rewards [220]. Particularly in competitive environments, these agents are expected to adapt between risk-seeking and risk-averse strategies based on the nature of their opponents [50]. Without tracking opponent behaviour, this adaptability is not possible, since the differences in opponent behaviour in the same state would not stimulate different responses by the learning agent. A related approach known as the *Teacher-Student Framework* [274] aims at accelerating the learning process under limited communication with an advisor. Almost all works in this framework assume either fully optimal or a moderate level of expertise for the advisors [7, 312].

At their core, RL (and MARL) algorithms are fixed point iterative methods that iterate recursively until no further iterations are desirable or required [153]. An RL algorithm's ability to converge to a fixed point provides a clear picture of the goal towards which the algorithm is progressing. The fixed point defines the completion of the task of an RL agent in the given environment and is like a terminal point in the sequence of RL iterations. For example, single-agent RL methods use the optimal Q-value that provides the maximum expected discounted sum of rewards in the given environment as the fixed point [256]. In MARL the fixed point is defined by the solution concept of the game. We note that many of the prior works referenced here do not contain a theoretical analysis of the learning algorithms that provide conditions for fixed point guarantees regarding learning in MARL environments. Since all these methods involve the presence of external sources in the learning process, it is unknown if previous guarantees in RL convergence extend to these approaches. Without such guarantees, it is unclear whether the algorithms will learn reasonable policies in any generic environment (beyond those considered in the chapter) and whether the algorithms will progress towards obtaining a suitable solution for the current problem. Since there are many solutions concepts in multi-agent environments [220], the kind of solutions that are likely to be obtained by these algorithms are unclear. Some approaches establish the existence of unique solution concepts in the particular model of multi-agent environment considered, yet still lack fixed point guarantees for any RL method and theoretical guarantees of arriving at the established solution concept by any learning algorithm [233, 310].

From the above discussion, we see that there are five fundamental limitations of existing algorithms that learn from external sources of knowledge, which hinders their applicability to real-world multi-agent environments. All prior works can be seen to contain one or more of these limitations. 1) Strict assumptions on the quality of advisors, 2) algorithms designed as single-agent based independent methods that consider other agents as simply part of the state in the environment, though the actions of these agents strongly influence the rewards for the learning agent, 3) offline advising, where demonstrations are collected and used for training agents offline, which is not well-suited for MARL due to the adaptive nature of opponents, where real-time feedback is critical, 4) algorithms designed towards a restricted class of MARL environments that are not generally applicable to many other environments, and 5) lack of thorough analysis for the conditions under which theoretical fixed point guarantees can be provided.

3.1.3 Our Approach

In this chapter, we study advising in MARL under the stochastic game model [219] and aim to resolve the five major limitations of prior methods discussed in Section 3.1.2. We will explore the use of advisors in multi-agent reinforcement learning (MARL) under general-sum settings, where advisors suggest (possibly sub-optimal) actions to different agents in the environment. The advisors can belong to a broad class of categories, such as pre-trained policies, rule-based systems or other systems that continue to learn and/or adapt during gameplay. We do not make any assumptions or place constraints on the quality or type of the advisors themselves. The advisors are assumed to be available online so that agents can get real-time feedback while training in dynamic MARL environments. We will also assume that each agent has access to at most one advisor. Communication between the agent and the advisor is assumed to be free. The advisor receives the state of an agent and provides an action recommendation for the current state. These action recommendations can be deterministic or stochastic.

We introduce a principled framework for studying the problem of **ADvising Multiple Intelligent Reinforcement Agents** (ADMIRAL). We propose two Q-learning based algorithms [291]. The first algorithm, **ADvising Multiple Intelligent Reinforcement Agents - Decision Making** (ADMIRAL-DM), learns to act in the environment using advisor-guidance, while the second, **ADvising Multiple Intelligent Reinforcement Agents - Advisor Evaluation** (ADMIRAL-AE), provides a principled method to evaluate the usefulness of the advisor in the current MARL context. To the best of our knowledge, we are the first to propose a method to evaluate a knowledge source before using it for learning in MARL[1]. We empirically study the performance of our algorithms in suitable test-beds,

[1]In their comprehensive survey of literature that aims at reusing knowledge to accelerate MARL, Silva

along with a comparison to related baselines. Theoretically, we establish conditions under which we can provide fixed point guarantees regarding the learning of our ADMIRAL algorithms in general-sum stochastic games.

Specifically, our contributions in this chapter are:

- Introducing a general paradigm for learning from external advisors in MARL.
- Analyzing two important challenges in learning from advisors in MARL.
- Presenting a suitable algorithm for each of these challenges.
- Establishing conditions for appropriate fixed point guarantees in these algorithms.
- Proving that it is possible to provide convergence results under less restrictive assumptions compared to prior work.
- Empirically showing that our algorithm can adapt and perform well in many challenges in MARL.

3.2 Advisor Q-Learning

In this section, we introduce the problem of ADvising Multiple Intelligent Reinforcement Agents (ADMIRAL). We have a set of agents that can either take an action using their own policy or consult an advisor that provides action recommendations (deterministic or stochastic), given the current state, at each time step. Each agent has access to at most one advisor. An advisor can be any external source of knowledge, such as a rule-based agent, a pre-trained policy, or any other system that continues to learn during gameplay. The advisor is assumed to be available online with the possibility of providing instantaneous action recommendations to an agent. Further, we consider a centralized training setting, where agents can observe the state, the local actions, and rewards of all other agents. Another specification is that in our setting, the advisor and agent communication is free, while the agents cannot communicate amongst themselves. There is no communication amongst the agents themselves since establishing reliable communication protocols amongst every

and Costa [225] state that several prior works [7, 312] assume (at least) a moderate level of expertise for the advisors for action advising and are only applicable to single-agent environments, in line with our discussions in Section 3.1.2. While [226] relax the assumption of optimal advisors by allowing agents to advise each other (in cooperative games), they do not provide a method to evaluate the available agents/advisors before using them.

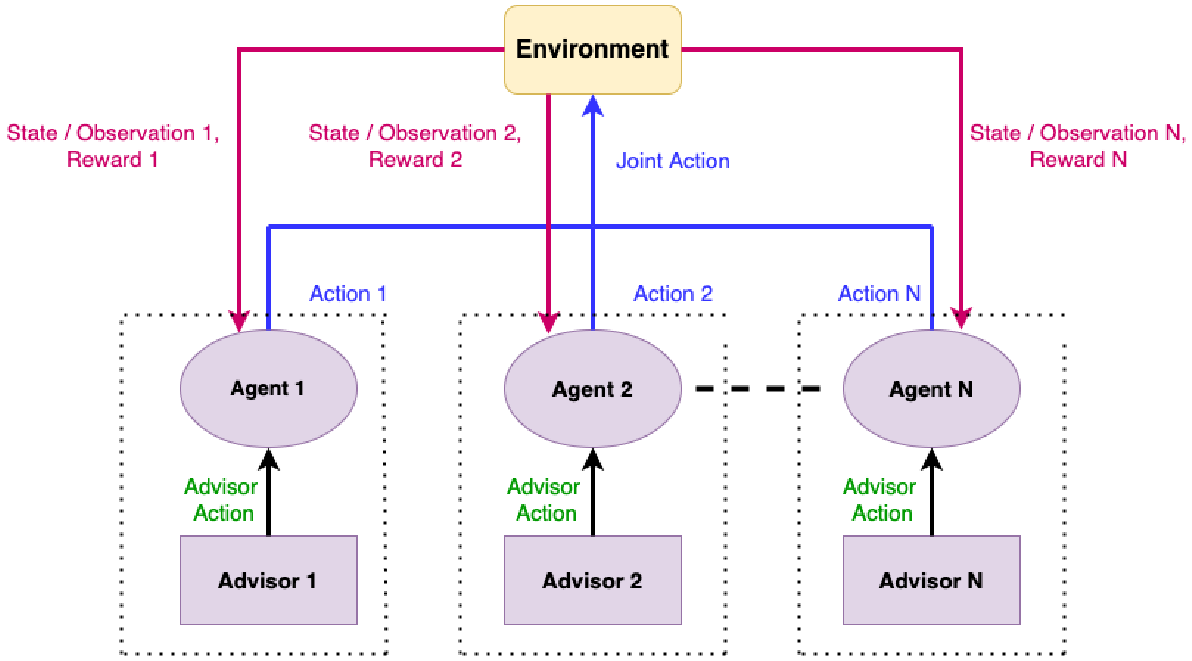


Figure 3.1: Architecture of the ADMIRAL-DM algorithm. Each agent has access to at most one advisor. The advisor provides action recommendations to the agent, and the agent can decide how much to rely on the advisor.

single agent may be prohibitively expensive in large multi-agent environments. For example, in the case of wildfire fighting, it has been noted that communication, even if available, could be very limited since the individual agents (fire-fighters) may be very far apart [190]. However, all agents can receive global inputs from satellite/airborne sensors [140]. The centralized setting we consider is similar to that in prior works [104]. Subsequently, in Section 3.2.4 we will show that our method can be adapted to the popular centralized training and decentralized execution [156] paradigm, which provides a suitable relaxation of the centralization assumption.

We study two challenges that arise when learning from advisors in MARL and provide algorithms for each problem. The first challenge is learning a policy with the help of an advisor. We introduce an algorithm for this challenge, which we call ADvising Multiple Intelligent Reinforcement Agents - Decision Making (ADMIRAL-DM). In this setting, each agent aims to learn a suitable policy that provides the best responses to the opponent(s) and performs effectively in the given multi-agent environment. An agent has access to a (possibly sub-optimal) advisor that could be leveraged to improve the speed of learning.

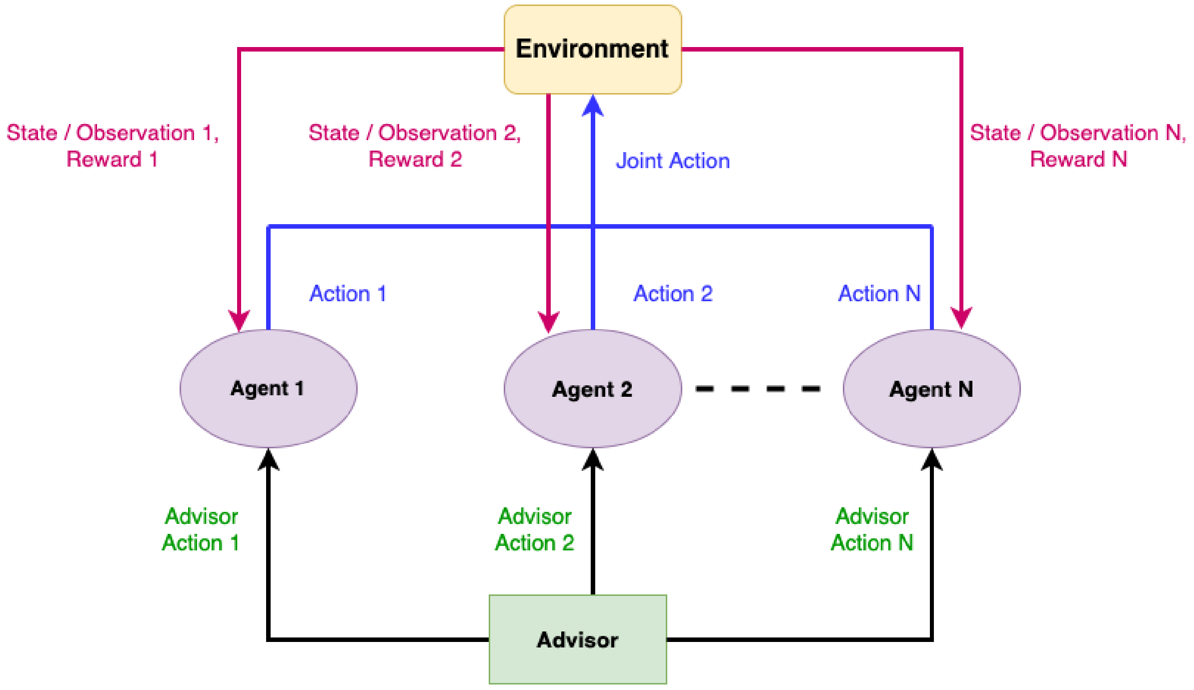


Figure 3.2: Architecture of the ADMIRAL-AE algorithm. All agents aim to evaluate a single advisor. The advisor can be queried by each agent to obtain action recommendations, which the agent can choose to execute.

Hence, at each time step, the agent could choose to follow its own policy or that of the advisor. In early stages of learning, the dependence on the advisor is greater, and this dependence gradually declines as the agent's policy improves. If an agent does not receive an action recommendation at some time step, they can simply use their own current policy. Hence, we do not require the advisor to be capable of providing action advice at every state in the state space. A schematic of this setting is provided in Figure 3.1.

The second challenge is the evaluation of the advisor itself. We provide an algorithm called ADvising Multiple Intelligent Reinforcement Agents - Advisor Evaluation (ADMIRAL-AE) that tackles this challenge. Before using an advisor, it is beneficial to evaluate it to determine whether the advisor will provide effective advice. Hence, we propose a 'pre-learning' phase (i.e., a distinct phase before the beginning of training of ADMIRAL-DM) where the ADMIRAL-AE is used with the goal of getting a good understanding of the capabilities of the advisor in the current environment. We assume that a single advisor exists in the system and this advisor could be evaluated by one or more agents. A schematic of this setting is provided in Figure 3.2.

Many real-world multi-agent application domains may have a state-of-the-art solution method being currently used in practice. This method could be normally useful, but may not be suitable for every context and against every possible opponent in MARL environments. Hence, it is possible to face situations in which the available advisor provides advice that is optimal or close to optimal for some states, but whose advice is poor in others. Intuitively, to learn well, agents must listen more to the advisor when the available advisor is good and well-suited to the current context and listen less (or not at all) to the available advisor when it is bad. To make this issue more concrete, recall the wildfire example discussed in Section 3.1.1. A well-known fire simulator model is the Farsite [69] simulator that is actively used in practice to model the spread of fire. This fire simulator model predicts the spread of fire in the future and forms the basis of fire-control strategies of firefighters. Notably, through extensive experimentation, it has been reported that this model does not give satisfactory performances under certain environmental conditions such as extreme downslope winds [323]. However, since real fires are affected by many dynamic environmental factors at the same time, coming up with suitable heuristics for deciding the fit of the given model is almost impossible. Thus, it is important to understand when an advisor is providing useful advice and when its advice has limitations. Always relying on a poor advisor may lead to poor policies being learned or hurt the overall sample complexity of the algorithms. Thus, we recommend evaluating the advisor before using it for learning, if possible. We argue that it is important to decouple the problem of "advisor-evaluation" where the objective is to study the suitability of the advisor in the given MARL environment from the problem of "decision-making" which aims to improve the training of MARL algorithms by making use of advisor knowledge.

We propose to conduct a 'pre-learning' phase before training the decision-making algorithm. In this phase we perform an "advisor-evaluation" study, either in simulation or in real-world environments (particularly in environments that are not safety-critical like recommendation systems and board games such as chess) that helps to answer two important questions before beginning to learn from the advisor. 1) Does the advisor have some good knowledge of the domain that could be helpful for the MARL algorithms during training? and 2) How much should one listen to the given advisor? Performing advisor evaluation before beginning the process of learning helps the agent gain a good understanding of the advisor before learning from it, which would help in leveraging it effectively in learning a suitable decision-making policy. Most importantly, in MARL, advisors (especially good ones) could continue to learn and/or adapt online, during gameplay, based on the nature of opponents. Such advisors cannot be evaluated effectively unless their learning and evolution is captured using a principled method designed to evaluate them. If advisors are being evaluated by agents along with agents simultaneously using them for learning decision-making policies, the evaluation becomes limited and advisors are prone to be discarded quickly based on the metrics of performance and consistency at the early stages of training. During this time, the advisor could be still evolving its strategies based on the nature of the opponent(s), and hence, the evaluation is not accurate. For example, in the algorithm CHAT from [289], the confidence of an agent on a demonstrator is determined based on the demonstrator's consistency in action recommendations for the same state. This approach works well in the single-agent context. However, in MARL, due to the adaptive nature of opponents, good advisors could adapt based on the opponent and possibly evolve mixed (stochastic) strategies that will provide different actions at the same state. An approach such as CHAT would reduce the confidence of such an advisor, but this is not accurate since the advisor is good and should be leveraged more for better performance.

3.2.1 Decision-Making Using Advisor

Our first algorithm learns to act in the environment by leveraging the available advisor.

ADMIRAL-DM is described in Algorithm 1. First, for simplicity, we assume that all the agents in the environment use the same algorithmic steps for learning, as done in prior work [104]. Subsequently, the same algorithm can be used in other scenarios where different agents use different algorithms for learning as well. Further, as in [104], we assume that all agents maintain a copy of the Q-updates of the other agents. This is possible since, during training, agents are in a centralized setting and can observe the local actions and rewards of all other agents at each time step. This helps in predicting the actions of opponents needed for providing the best responses.

A learning agent (represented by j) starts with an arbitrary initialization of its Q-value $Q_0^j(s, \boldsymbol{a})$. One such assignment could be to set $Q_0^j(s, \boldsymbol{a}) = 0$, for all agents j, all states $s \in \mathcal{S}$ and actions $a^1 \in A^1, \ldots, a^n \in A^n$. Recall, in this setting, each agent has access to an online advisor that it could query during learning. Whenever the agent needs to choose an action, it does so based on its current Q-value, the advisor's recommendation, or simply a random action, as the case may be (lines 8–15). The dependence on the advisor's recommendation and the random exploration is captured by two hyperparameters, ϵ'_t and ϵ_t, respectively. This action is subsequently executed and the actions and rewards of the other agents are observed, including the next state s' (line 6). During training, at each time step, the agent picks the possible next actions of other agents in line 7 using its copy of other agents Q-values. Then, in lines 8–15, the agent j picks its next action based on ADMIRAL-DM algorithm's policy which chooses a random action and an advisor action with diminishing probabilities, and a greedy action with increasing probabilities, such that it becomes greedy

Algorithm 1 ADvising Multiple Intelligent Reinforcement Agents - Decision Making (ADMIRAL-DM)

```
1: Set $t = 0$, get the initial state $s_0$. Let the learning agent be indexed by $j$
2: For all $j$, $s \in \mathcal{S}$, and $a^j \in A^j$, let $Q^j_0(s, \boldsymbol{a}) = 0$, where $\boldsymbol{a} = (a^1, \dots, a^n)$. Initialize a value for hyperparameters $\epsilon$ and $\epsilon'$ (i.e. value for $\epsilon_0$ and $\epsilon'_0$)
3: Define policy derived from $Q$ to return a random action with probability $\epsilon_t$, advisor suggested action with probability $\epsilon'_t$ and greedy action with probability $1 - \epsilon_t - \epsilon'_t$
4: Choose $a^j_0$ at state $s_0$ for each $j$ using policy derived from $Q$
5: while $Q^j$ is not constant for each $j \in \{1, \dots, n\}$ do
6:     For each agent $j$, execute the action $a^j_t$ and observe $r^1_t, \dots, r^n_t$; $a^1_t, \dots, a^n_t$; and $s_{t+1} = s'$
7:     For each $j$, choose the next greedy action for all other agents from the copy of their respective policies using $s'$. The next greedy actions are chosen using the current observed actions of other agents
8:     For each $j$, let $u$ be a uniform random number between 0 and 1
9:     if $u < \epsilon'_t$ then
10:        Obtain next action $a^j_{t+1} = a^{j'}$ from the advisor (using state $s'$)
11:    else if $u > \epsilon'_t$ and $u < \epsilon_t$ then
12:        Set the next action $a^j_{t+1} = a^{j'}$ as a random action from the action space $A^j$
13:    else
14:        Choose a greedy action $a^j_{t+1} = a^{j'}$ from the $Q$-function using $s'$ and the next greedy actions of other agents
15:    end if
16:    Update $Q^j_t$ for each $j \in \{1, \dots, n\}$ using Eq. 3.1
17:    Let $t := t + 1$
18:    For each agent $j$, set the current action $a^j = a^{j'}$ and current state $s = s'$
19:    At the end of each episode, linearly decay $\epsilon_t$ and $\epsilon'_t$
20: end while
```

in the limit with infinite exploration (GLIE). Thus, the agent is guaranteed to train without any further advisor influence after some finite time t in the training process. Accordingly, the dependence on the advisor's recommendation is decayed linearly (line 19). In this process, the dependence of an agent is more on the advisor during the earlier stages of learning, when its own policy is quite bad. This dependence gradually reduces as its own policy improves. The Q-values are updated (line 16) following an update scheme given by,

$$Q^j_{t+1}(s, \boldsymbol{a}) = (1 - \alpha_t)Q^j_t(s, \boldsymbol{a}) + \alpha_t[r^j_t + \beta Q^j_t(s', \boldsymbol{a'})] \tag{3.1}$$

where $\boldsymbol{a} = (a^1, \ldots, a^n)$ denotes the actions for all agents at state s and $\boldsymbol{a'} = (a^{1'}, \ldots, a^{n'})$ denotes the actions for all the agents at state s'. β denotes the discount factor, and $\alpha_t \in (0, 1)$ is the learning rate. The other variables have the usual meaning described in Chapter 2. The algorithm's steps are repeated continuously until either the Q-values fully converge or come within a small threshold of convergence, as is commonly done in practice [256].

The ADMIRAL-DM algorithm's time and space complexity can be compared to the NashQ algorithm from Hu and Wellman [104]. At each time step, a learning agent j needs to update $(Q^1, \ldots, Q^n)$, for all states $s \in \mathcal{S}$, and all actions $a^1 \in A^1, \ldots, a^n \in A^n$. Let the total number of states in the environment be represented by $|\mathcal{S}|$, and $|A^j|$ be the total number of actions in the action space of the agent j. Further, assuming that $|A^1| = \cdots = |A^n| = |A|$, we get the total number of entries in Q^j to be $|\mathcal{S}||A|^n$. If the learning agent needs to update a total of n Q-tables, then the space complexity can be given by $n|\mathcal{S}||A|^n$. Thus, regarding the space complexity, a tabular version of ADMIRAL-DM is linear in the number of states, polynomial in the number of actions, and exponential in the number of agents, which is the same as the guarantees for the NashQ algorithm in Hu and Wellman [104]. However, in the case of time complexity, ADMIRAL-DM has the same guarantees as given for the space complexity, since in the worst case, each entry in the table needs to be queried before updating a Q-value. Note that this is better than the algorithm by Hu and Wellman [104], which had exponential time complexity in the states and actions, even in the case of a two-player game. This is because the NashQ algorithm has a requirement of determining the Nash equilibrium at each stage game, which has exponential time complexity even for two-player games [174]. We do not have this requirement.

3.2.2 Evaluation Of Advisors

The second challenge is that of evaluating the advisor to determine its nature and its suitability for the given context. As described in the previous sub-section, the ADMIRAL-DM uses an advisor if one exists. In this sub-section, we provide an algorithm (ADMIRAL-AE) that evaluates a potential advisor and helps in guiding the configuration of Algorithm 1 by setting the initial value of ϵ'. The objective is to make an agent following ADMIRAL-DM listen more to good advisors and listen less (or not at all) to bad advisors. The ADMIRAL-AE is used in the 'pre-learning' phase discussed earlier, where agent(s) are evaluating the advisor in the context of the given environment and opponents.

We start with a definition of an advisor strategy.

Definition 8. *In a stage game,* $(\boldsymbol{A}, M^1, \dots, M^n)$ *an advisor strategy (or advisor solution) is a tuple of* n *strategies* $(\sigma^1, \dots, \sigma^n)$*, an advisor specifies for all* n *agents.*

Since the advisor is defined to be a general model that receives a state and provides action recommendations to an agent in the multi-agent environment, the advisor, in general, is capable of predicting the actions of other agents as well. All the advisor's predictions and/or recommendations towards every agent in the environment constitute the advisor solution as in Definition 8. Here, we do not restrict our setting to environments where the advisor can predict the actions of all agents. In practice, it is possible to encounter situations where the advisor cannot predict or recommend actions to some agents. In this case, these agents can get random or placeholder strategies in the advisor solution formulated in Definition 8.

Similar to our decision-making setting, we first provide an algorithm that will have all the agents in the environment using the same algorithmic steps. In each state s, and time t during training, we form a stage game $(Q^1_t(s), \dots, Q^n_t(s))$ using the Q-values of all agents. Here, the notation $Q^j_t(s) = (Q^j_t(s, a^1), \dots, Q^j_t(s, a^n))$. The advisor receives the state and provides its predictions/recommendation for each agent, which will form the advisor solution $(\sigma^1_t(s), \dots, \sigma^n_t(s))$ for the stage game $(Q^1_t(s), \dots, Q^n_t(s))$. In the stochastic game, having access to the state and the advisor allows an agent to have access to the full advisor solution at every state $s \in \mathcal{S}$ for all time t.

Algorithm 2 describes our ADMIRAL-AE algorithm. A learning agent j starts with an arbitrary value of $Q^j_0(s, \boldsymbol{a})$, which represents the value at the initial time step $t = 0$. We define an action selection scheme (lines 5–12) that chooses to directly use the advisor's recommendation with probability η', a random action with probability η and an action that maximizes the Q-values at the current state with probability $1 - \eta - \eta'$. The idea is to mix between directly following the advisor at the current time step and choosing an action that maximizes the value of following the advisor at later stages for the action selection. We also perform a small percentage of random actions to ensure sufficient exploration of the environment. At each time t, the agent j observes the current state s, and takes a local action a^j (line 13) and observes the action of all agents (including itself), the reward it obtains and the new state s'. Note that, unlike the decision-making setting, here we do not require all agents to maintain copies of the updates of other agents and hence the rewards of other agents are not required to be observed by the agent j. The agent then obtains the advisor solution (Definition 8) from the advisor for the next state s' (line 14). Subsequently, each agent j updates its Q-value as follows (using $\beta \in [0, 1)$, as the discount factor) :

$$Q^j_{t+1}(s, \boldsymbol{a}) = (1 - \alpha_t)Q^j_t(s, \boldsymbol{a}) + \alpha_t[r^j_t + \beta Advisor Q^j_t(s')] \tag{3.2}$$

Algorithm 2 ADvising Multiple Intelligent Reinforcement Agents - Advisor Evaluation (ADMIRAL-AE)

```
1: Set $t = 0$ and get the initial state $s_0$. Let the learning agent be indexed by $j$
2: For all $j$, $s \in \mathcal{S}$, and $a^j \in A^j$, let $Q^j_0(s, \boldsymbol{a}) = 0$ where $\boldsymbol{a} = (a^1, \ldots, a^n)$
3: Set the value of hyperparameters $\eta$ and $\eta'$
4: while $Q^j$ is not constant for each $j \in \{1, \ldots, n\}$ do
5:     For each $j$, let $u$ be a uniform random number between 0 and 1
6:     if $u < \eta'$ then
7:         Obtain action $a^j_t$ for the current state $s_t$ from the advisor
8:     else if $u > \eta'$ and $u < \eta$ then
9:         Set the action $a^j_t$ as a random action from the action space $A^j$
10:    else
11:        Choose a greedy action $a^j_t$ from the $Q$-function using $s_t$ and the observed previous actions of other agents
12:    end if
13:    Execute $a^j_t$ and then observe $a^1_t, \ldots, a^n_t$; $r^j_t$; and $s_{t+1} = s'$ for each $j \in \{1, \ldots, n\}$
14:    Obtain the advisor solution from the advisor for state $s'$
15:    Update $Q^j_t$ for each $j \in \{1, \ldots, n\}$ using Eq. 3.2
16:    Let $t := t + 1$ and current state $s_t = s_{t+1}$
17: end while
```

where $\alpha_t \in (0, 1)$ is the learning rate. The term $AdvisorQ^j_t(s')$, is the total value (payoff) that the agent j will obtain at the state s' when all agents (including itself) play the advisor solution. This is calculated as $AdvisorQ^j_t(s') = \sigma^1_t(s') \cdots \sigma^n_t(s') \cdot Q^j_t(s')$, where $(\sigma^1_t(s'), \ldots, \sigma^n_t(s'))$ denotes the advisor solution at state s' and time t. This can be seen as a solution to the stage game $(Q^1_t(s'), \ldots, Q^n_t(s'))$, since the value of each agent's payoff at state s' is reflected in their corresponding Q-values at state s'. Since the advisor recommendation to each agent can be a stochastic sample, the $\sigma^j_t(s')$ is interpreted as a vector that contains the probability of taking each action in the action space of j. Similarly, from the Q-function of the agent j, we can obtain $Q^j_t(s')$, which consists of the Q-value of taking each action in the action space of j. Hence, $AdvisorQ^j_t(s')$ is a scalar value obtained using a component-wise multiplication of the advisor solution and the Q-values.

The Q-values of all agents are updated using the advisor strategies at each iteration, as given in line 15 of Algorithm 2. The above-described steps continue until convergence, or until the Q-values come within a small threshold of convergence, as in ADMIRAL-DM.

Recall that the primary purpose of this algorithm is advisor evaluation. After implementing ADMIRAL-AE using the given advisor in the 'pre-learning' phase, the performance of the algorithm can be used to determine ϵ_0' (hyperparameter of ADMIRAL-DM) in different ways. We provide one heuristic in this dissertation. We propose that the performance of ADMIRAL-AE (in terms of cumulative rewards) be compared against the maximum possible performance of any algorithm (maximum cumulative rewards). The ADMIRAL-AE's performance using the given advisor should lie between the maximum possible performance in that environment and the performance of ADMIRAL-AE using a random advisor. This can then be normalized in the range of $[0, 1]$ to determine a value for ϵ_0'. This normalization is given in Eq. 3.3.

$$\epsilon_0' = \frac{CR - RCR}{MCR - RCR} \tag{3.3}$$

where CR denotes the cumulative reward obtained by ADMIRAL-AE using the advisor (averaged across multiple trials), RCR denotes the cumulative reward obtained by ADMIRAL-AE using a random advisor (averaged across multiple trials), MCR denotes the maximum possible cumulative reward in the given environment [2]. To be more accurate, a correction can be applied to MCR to compensate for the loss in performance from random exploration in ADMIRAL-AE (hyperparameter η).

After obtaining the value of ϵ_0' from ADMIRAL-AE, this hyperparameter is used in the training of ADMIRAL-DM where its value is linearly decayed in line with the steps in Algorithm 1. We experimentally illustrate these steps later in Section 3.4.1. A more elaborate study demonstrating the effectiveness of this technique is given in Section 3.4.4.

Another way of using ADMIRAL-AE is to study the effectiveness of the available advisor against simulated or baseline opponents (Section 3.4.2). An important advantage of ADMIRAL-AE is in situations of adapting advisors, as discussed earlier. An experimental illustration of this advantage is given in Appendix A.3. The ADMIRAL-AE algorithm is off-policy, as the update policy (line 15) and policy followed (lines 5–12) are different. Due to this off-policy nature of ADMIRAL-AE, we do not require an agent to follow the advisor at every state in this setting. The evaluation is happening through the Q-values, while the action selection policy can be independent of the policy being updated as in any off-policy algorithm. The convergence guarantees in off-policy methods do not require using specific action selection policies as long as sufficient exploration is guaranteed [111, 256].

[2] We clarify that in this method if the RCR and MCR are not very "tight," the difference between almost similar performing advisors could become small. However, this will not be a problem for our method, since the ADMIRAL-DM can also learn directly through environmental interactions.

We would like to clarify that our main algorithm, ADMIRAL-DM, uses the advisor in a fully-online fashion and does not require any 'pre-learning' for implementation. ADMIRAL-AE is a principled method that helps in determining how much to listen to an advisor (through the hyperparameter ϵ'_0) in an optional 'pre-learning' phase. If the 'pre-learning' phase is not conducted, an approximate value for ϵ'_0 can still be obtained using experimental heuristics. Using such an approximate value is not a problem since ADMIRAL-DM is also capable of learning from environmental rewards (through direct environmental interactions), in addition to the advisor. Here the advisor only aims to accelerate the process of training. Hence, the use of ADMIRAL-AE does not violate our contribution of relaxing the offline limitation in prior methods.

In a tabular implementation of the ADMIRAL-AE algorithm, both space and time complexities will be linear in the number of states, polynomial in the number of actions, and exponential in the number of agents, same as that described for ADMIRAL-DM. However, the space and time complexity of ADMIRAL-AE can be represented by $|\mathcal{S}||A|^n$. Here, notice that the complexity does not have the product term of the number of agents n, unlike the requirement for ADMIRAL-DM. This is due to the fact that ADMIRAL-AE does not have the requirement of each agent maintaining copies of the Q-values of other agents as in ADMIRAL-DM. As described in the previous sub-section, this time complexity of ADMIRAL-AE is much better than that of NashQ [104].

3.2.3 Illustrative Example For Algorithm 2

In this sub-section, we show the calculations of some steps in the Q-updates of Algorithm 2 for a single state system, to serve as a demonstration of this algorithm. Our objective is to provide a practical illustration of the various steps involved since, to the best of our knowledge, pre-evaluation of advisors have not been considered before. Since Algorithm 1 has similarities to the well-known Q-learning algorithm [291], we omit a demonstrative example for Algorithm 1.

Let us consider a two agent game with all the initial Q-values set to 0. The first agent (column agent) can perform two actions "Up" and "Down" and the second agent (row agent) can also perform two actions "Left" and "Right". The system has only one state, s_1. Let the learning rate α be 0.9 and discount factor β be 0.9. Let us assign the hyper-parameters $\eta = 0.05$ and $\eta' = 0.45$. At the initial state, at time $t = 0$, the stage game constructed from the Q-values of both the agents is given in Table 3.1a.

At state s_1, in time $t = 0$, let us assume that both agents decide to use the advisor recommended actions "Up" and "Left" respectively. They execute these actions and obtain

(Q^1_0, Q^2_0)	Left	Right
Up	(0,0)	(0,0)
Down	(0,0)	(0,0)

(a) Initial stage game at time $t = 0$

(Q^1_1, Q^2_1)	Left	Right
Up	(1.8, 1.8)	(0,0)
Down	(0,0)	(0,0)

(b) Stage game at time $t = 1$

(Q^1_2, Q^2_2)	Left	Right
Up	(2.34, 2.34)	(0,0)
Down	(0,0)	(0,0)

(c) Stage game at time $t = 2$

Table 3.1: Stage games constructed in the example.

a reward of 2 each. Now the agents receive the next state, which is s_1 (single state system). Let the advisor solution at this state for the column agent be $\sigma^1_0(s_1) = (1, 0)$ and that for the row agent be $\sigma^2_0(s_1) = (1, 0)$. The first value of the tuple in this notation is the probability of taking the first action, and the second value of the tuple is the probability of the second action for the respective agents. This means that the advisor recommends both the agents to perform the "Up" and "Left" actions respectively, with probability 1, and the other action with probability 0. The Q update will be as follows:

$$\begin{aligned}
&Q^1_1(s_1, \text{Up}, \text{Left}) = Q^1_0(s_1, \text{Up}, \text{Left}) + \alpha\Big(r^1_0 + \beta Advisor Q^1_0(s_1) - Q^1_0(s_1, \text{Up}, \text{Left})\Big)\\
&Q^1_1(s_1, \text{Up}, \text{Left}) = Q^1_0(s_1, \text{Up}, \text{Left}) + \alpha\Big(r^1_0 + \beta\sigma^1_0(s_1) \cdot \sigma^2_0(s_1) \cdot Q^1_0(s_1) - Q^1_0(s_1, \text{Up}, \text{Left})\Big)\\
&Q^1_1(s_1, \text{Up}, \text{Left}) = 0 + 0.9\Big(2 + 0.9 \times 0 - 0\Big)\\
&Q^1_1(s_1, \text{Up}, \text{Left}) = 1.8
\end{aligned} \tag{3.4}$$

The superscript for Q represents the agent index (1 represents the column player). The above equation also holds for the row agent and the new stage game with Q-values at state $s1$, in time $t = 1$ is given in Table 3.1b.

Now, at state s_1 and time $t = 1$, let us assume that the agents decide to perform the best actions from their respective Q-values, i.e., "Up" and "Left" again. The actions are executed, and the agents obtain a reward of 2 each. The next state is observed, and let the advisor solution here be $\sigma_1^1(s_1) = (0.5, 0.5)$ and $\sigma_1^2(s_1) = (0.5, 0.5)$. This means that the advisor assigns a probability of 0.5 for each of the actions for both the agents. Again, we calculate the Q-update for the column agent,

$$\begin{aligned}
&Q_2^1(s_1, \text{Up}, \text{Left}) = Q_1^1(s_1, \text{Up}, \text{Left}) + \alpha\Big(r_1^1 + \beta Advisor Q_1^1(s_1) - Q_1^1(s_1, \text{Up}, \text{Left})\Big)\\
&Q_2^1(s_1, \text{Up}, \text{Left}) = Q_1^1(s_1, \text{Up}, Left) + \alpha\Big(r_1^1 + \beta\sigma_1^1(s_1) \cdot \sigma_1^2(s_1) \cdot Q_1^1(s_1) - Q_1^1(s_1, \text{Up}, \text{Left})\Big)\\
&Q_2^1(s_1, \text{Up}, \text{Left})\\
&= 1.8 + 0.9\Big(2 + 0.9\big((0.5)^2 \times 1.8 + (0.5)^2 \times 0 + (0.5)^2 \times 0 + (0.5)^2 \times 0\big) - 1.8\Big)\\
&Q_2^1(s_1, \text{Up}, \text{Left}) = 2.34
\end{aligned} \tag{3.5}$$

The above equation also holds for the row agent and the new stage game at $t = 2$ is given in Table 3.1c. Similarly, the Q-values continue to be updated until convergence or until the values come to a small threshold of convergence.

In the above steps, we have demonstrated the Q-values of different actions at the given state based on the advisor solutions. Such a process, at convergence, will lead to the agent(s) evaluating the advisor obtain a value for the advisor at each state in the state space of the environment.

3.2.4 Neural Network And Actor-Critic Implementations

It is well known that tabular algorithms are not applicable for environments with large state and action spaces in RL [166]. All our algorithms can be extended to large state-action environments using function approximations as is common in RL [166, 167], where neural networks serve as the function approximators. In this section, we give a neural network-based implementation of ADMIRAL-DM and ADMIRAL-AE.

We incorporate techniques introduced in the well-known Deep Q-learning (DQN) algorithm [167] with the update rule in ADMIRAL-DM to obtain its neural network implementation in Algorithm 3. To highlight differences from the tabular implementation, we make note of some parts of Algorithm 3. All agents maintain two networks (evaluation and target) throughout the training process. Both the evaluation and target networks start with the same configuration. The evaluation network is updated periodically at every training step and used for action selection at each step. The target network provides the target value for calculating the Bellman errors during training and is updated less frequently compared to the evaluation network. In line 18 of Algorithm 3, all agents store the experience tuples in their respective replay buffers. After every episode in lines 21 – 25, the evaluation networks are trained using the temporal difference (TD) errors as the loss function. The TD target is obtained from the Bellman equation given in Eq. 3.1. After every finite number of training steps, the target network parameters are updated by copying over values from the evaluation network in line 26 as previously performed in [167].

Similar to our approach with ADMIRAL-DM, we incorporate the techniques of DQN [167] with the update rule in ADMIRAL-AE to obtain Algorithm 4. The important changes are similar to our discussion with the ADMIRAL-DM case, where the algorithm uses two networks and a replay buffer for training. The target for the loss function is obtained from Eq. 3.2 (line 21 in Algorithm 4). If more exploration is desired, ϵ need not be decayed in these implementations.

We also extend Algorithm 3 to an actor-critic method — **ADvising Multiple Intelligent Reinforcement Agents - Decision Making (Actor-Critic)** abbreviated as ADMIRAL-DM(AC). This algorithm uses the Q-function as the critic and the policy derived from Q as the actor. The algorithm follows a *Centralized Training and Decentralized Execution* (CTDE) scheme [156], where the critic uses the information associated with other agents during the training time and the actors can act independently without access to other agent information during execution. This allows our methods to be applicable in environments where global information (i.e., information associated with other agents) is available during training but not available during execution such as, for example, autonomous driving [319]. The CTDE scheme extends our algorithm to partially observable environments, where the actor can just use the local observations of the agent for action selection (during both training and execution), while the critic can use the joint observation of all agents during training. As discussed in [156] in the simplest case, the Q-function (used as the critic) would consist of the observations of all agents, but it could also include additional state information when available. The critic is not required during execution in this setting. Furthermore, ADMIRAL-DM(AC) makes our method applicable to continuous action spaces as well.

We provide the complete pseudocode for the actor-critic implementation of ADMIRAL-DM in Algorithm 5. All agents maintain two networks during training. The first network is the value network that serves as a critic, and the second network is a policy network that serves as the actor (line 1). After each experience, the value (critic) network is updated in line 16 using the TD errors (from Eq. 3.1) as the loss function. The actor is updated in line 17 using policy gradients. Since the algorithm is maintaining a stochastic policy which explores naturally, we do not need to perform a random action selection for exploration

Algorithm 3 ADMIRAL-DM Neural Network Implementation

```
1: Initialize $Q_{\phi^j}, Q_{\pi^j}$, where $\phi$ represent the evaluation (eval) net and $\pi$ represents the target net, for all $j \in \{1, \ldots, n\}$. Initialize a value for hyperparameters $\epsilon$ and $\epsilon'$ (i.e. value for $\epsilon_0$ and $\epsilon'_0$)
2: At $t = 0$, get the initial state $s_0$ from the environment
3: Let the learning agent be indexed by $j$
4: For all $s \in \mathcal{S}$, $a^j \in A^j$, and $j \in \{1, \ldots, n\}$, let $Q^j_t(s, a^1, \ldots, a^n) = 0$
5: Define policy derived from $Q$ to return the random action $a$ with probability $\epsilon_t$, advisor suggested action with probability $\epsilon'_t$ and greedy action with probability $1 - \epsilon_t - \epsilon'_t$
6: Choose action $a^j_0$ for $s_0$ using policy derived from $Q$ for each $j \in \{1, \ldots, n\}$
7: **while** training is not finished **do**
8:     For each agent $j$, execute the action $a^j_t$ and observe $r^1_t, \ldots, r^n_t$; $a^1_t, \ldots, a^n_t$; and $s_{t+1} = s'$
9:     For each $j$, choose the next greedy action for all other agents from the copy of their respective policies using $s'$. The next greedy actions are chosen using the current observed actions of other agents
10:    For each $j$, let $u$ be a uniform random number between 0 and 1
11:    **if** $u < \epsilon'_t$ **then**
12:        Obtain next action $a^j_{t+1} = a^{j'}$ from the advisor (using state $s'$)
13:    **else if** $u > \epsilon'_t$ and $u < \epsilon_t$ **then**
14:        Set the next action $a^j_{t+1} = a^{j'}$ as a random action from the action space $A^j$
15:    **else**
16:        Choose a greedy action $a^j_{t+1} = a^{j'}$ from $Q_{\phi^j}$ using $s'$ and the next greedy actions of other agents
17:    **end if**
18:    Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'} \rangle$ in replay buffer $\mathscr{D}$, where $s = s_t$, $\boldsymbol{a} = a^1_t, \ldots, a^n_t$; $\boldsymbol{r} = r^1_t, \ldots, r^n_t$ and $\boldsymbol{a'} = a^1_{t+1}, \ldots, a^n_{t+1}$; for each agent $j$
19:    Set the current action $a^j_t = a^j_{t+1}$ and the current state $s_t = s_{t+1}$; for each agent $j$
20:    At the end of each episode linearly decay $\epsilon'_t$ and $\epsilon_t$
21:    **while** $j = 1$ to $n$ **do**
22:        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'} \rangle$ from $\mathscr{D}$
23:        Set $y^j = r^j + \beta Q_{\pi^j}(s', \boldsymbol{a'})$ according to Eq. 3.1
24:        Update the $Q$ eval network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s, \boldsymbol{a}))^2$
25:    **end while**
26:    Update the parameters of the target network for each agent by copying over the eval network every $\mathcal{T}$ steps: $\pi^j \leftarrow \phi^j$
27: **end while**
```

Algorithm 4 ADMIRAL-AE Neural Network Implementation

```
1: Initialize $Q_{\phi^j}, Q_{\pi^j}$, where $\phi$ represent the evaluation (eval) net and $\pi$ represents the target net, for all $j \in \{1, \ldots, n\}$. Initialize the hyperparameters $\eta$ and $\eta'$
2: At $t = 0$, get the initial state $s_0$ from the environment
3: Let the learning agent be indexed by $j$
4: For all $s \in \mathcal{S}$, $a^j \in A^j$, and $j \in \{1, \ldots, n\}$, let $Q^j_t(s, a^1, \ldots, a^n) = 0$
5: Set the value of hyperparameters $\eta$ and $\eta'$
6: while training is not finished do
7:     For each $j$, let $u$ be a uniform random number between 0 and 1
8:     if $u < \eta'$ then
9:         Obtain action $a^j_t$ for the current state $s_t$ from the advisor
10:    else if $u > \eta'$ and $u < \eta$ then
11:        Set the action $a^j_t$ as a random action from the action space $A^j$
12:    else
13:        Choose a greedy action $a^j_t$ from $Q_{\phi^j}$ using $s_t$ and the observed previous actions of other agents
14:    end if
15:    For each agent $j$, execute the action $a^j_t$ and observe $r^j_t; a^1_t, \ldots, a^n_t$, and $s_{t+1} = s'$
16:    For each agent $j$, obtain the advisor action $a^1_{e,t+1}, \ldots, a^n_{e,t+1}$ for all agents at state $s'$
17:    For each agent $j$, store $\langle s, \boldsymbol{a}, r^j, s', \boldsymbol{a'} \rangle$ in replay buffer $\mathscr{D}$, where $s = s_t$, $\boldsymbol{a} = a^1_t, \ldots, a^n_t$ and $\boldsymbol{a'} = a^1_{e,t+1}, \ldots, a^n_{e,t+1}$
18:    Set the current state $s_t = s_{t+1}$
19:    while $j = 1$ to $n$ do
20:        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, r^j, s', \boldsymbol{a'} \rangle$ from $\mathscr{D}$
21:        Set $y^j = r^j + \beta AdvisorQ_{\pi^j}(s')$ according to Eq. 3.2
22:        Update the $Q$ eval network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s, \boldsymbol{a}))^2$
23:    end while
24:    Update the parameters of the target network for each agent by copying over the eval network every $\mathcal{T}$ steps: $\pi^j \leftarrow \phi^j$
25: end while
```

(unlike in the other algorithms).

3.3 Theoretical Results

In this section, we first show that Q-updates following Algorithm 2 will converge to an ϵ-equilibrium in the stochastic game. From the update rule provided in Eq. 3.2, we note that this equilibrium corresponds to the value of the advisor, which is the action-value function that provides the expectation of immediate reward and future discounted rewards when all agents follow the advisor solutions for infinite periods starting from the current

Algorithm 5 ADMIRAL-DM(AC) Neural Network Implementation

```
1: Initialize $V_{\phi^j}, \pi_{\theta^j}$, the critic and actor networks for all $j \in \{1, \ldots, n\}$. Initialize a value for
   hyperparameters $\epsilon$ and $\epsilon'$ (i.e. value for $\epsilon_0$ and $\epsilon'_0$)
2: At $t = 0$, get the initial state $s_0$
3: Let the learning agent be indexed by $j$
4: For all $s \in \mathcal{S}$ and $a^j \in A^j$, $j \in \{1, \ldots, n\}$, let $Q^j_t(s, a^1, \ldots, a^n) = 0$
5: Define policy derived from $Q$ to return the random action $a$ with probability $\epsilon_t$, advisor
   suggested action with probability $\epsilon'_t$ and greedy action with probability $1 - \epsilon_t - \epsilon'_t$
6: For each $j$ sample action $a^j_0$ from the actor $\pi_{\theta^j}$ at state $s_0$
7: while training is not finished do
8:     Execute the action $a^j_t$ and observe $r^1_t, \ldots, r^n_t; a^1_t, \ldots, a^n_t$; and $s_{t+1} = s'$, for all agents $j$
9:     For each $j$, let $u$ be a uniform random number between 0 and 1
10:    if $u < \epsilon'_t$ then
11:        Obtain next action $a^j_{t+1} = a^{j'}$ from the advisor (using state $s'$)
12:    else
13:        Choose an action $a^j_{t+1} = a^{j'}$ using the respective actor $\pi_{\theta^j}(s')$
14:    end if
15:    Set $y^j = r^j + \beta V^j_{\phi^j}(s', \boldsymbol{a'}^{-j})$, where $\boldsymbol{a'}^{-j} = (a'^1, \ldots, a'^{j-1}, a'^{j+1}, \ldots, a'^N)$, for all $j$
16:    For each $j$, update the critic by minimizing the loss $\mathcal{L}(\phi^j) = (y^j - V^j_{\phi^j}(s, \boldsymbol{a}^{-j}))^2$, where
   $\boldsymbol{a}^{-j} = (a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^N)$ and $s = s_t$
17:    For each $j$, update the actor using the log loss $\mathcal{J}(\theta^j) = \log \pi_{\theta^j}(a^j|s)\mathcal{L}(\phi^j)$
18:    Set the current action $a^j = a^{j'}$ and the current state $s = s'$ for each agent $j$
19:    At the end of each episode linearly decay $\epsilon'_t$ and $\epsilon_t$
20: end while
```

state and joint action. This ϵ of the ϵ-equilibrium will depend on the nature of the advisor used. Further, we prove that the Q-updates following Algorithm 1 converges to the Nash Q-value, thus finding the Nash equilibrium of the stochastic game.

The primary convergence result for Q-learning based algorithms in a general-sum stochastic game was provided by Hu and Wellman [104]. However, this result relies on a very restrictive assumption that states that every stage game of the stochastic game contains a Nash equilibrium that is either a global optimum or a saddle point. Additionally, an agent must use the payoff at this equilibrium to update its Q-value in every stage game of the stochastic game. As shown by Bowling [28], this assumption implies that every stage game should use the same kind of equilibrium, it cannot oscillate between being a global optimum or saddle point between stage games. There is almost no game that satisfies this condition in practice [104]. We will show in this section that the convergence results in our setting can be provided under a set of assumptions weaker than that used by Hu and Wellman[104].

In this section, we provide three important theorems with their detailed proofs. The proofs of each theorem depend on a set of lemmas that we provide. We have included the statement of these lemmas in this section, while the complete proofs of the lemmas can be found in Appendix A.1.

We start by providing a general result for stochastic processes. Theorem 1 is a technical result, extending a result of Szepesvari and Littman [262], which will form the foundation of our convergence result in Theorem 2. Theorem 1 aims to relax a requirement of contraction conditions in the result from Szepesvari and Littman [262]. The presence of these conditions would necessitate strong assumptions in a MARL setting as shown in Hu and Wellman [104], which we aim to avoid. Towards our Theorem 1, we restate some formal definitions relating to translation and invariance of operators from Szepesvari and Littman [262] in Appendix A.2 to stay self-contained.

Theorem 1. *Let $\mathscr{X}$ be an arbitrary set and assume that $\mathcal{B}$ is the space of bounded functions over $\mathscr{X}$. Let $T : \mathcal{B} \to \mathcal{B}$, be an arbitrary operator. Let $F \subseteq \mathcal{B}$, be a subset of $\mathcal{B}$ and let $\mathcal{F}_0 : F \to 2^{\mathcal{B}}$ be a mapping that associates subsets of $\mathcal{B}$ with the elements of F. Let v^* be a fixed point of T and let $\mathcal{T} = (T_0, T_1, \dots)$ be a sequence of random operators, T_t mapping $\mathcal{B} \times \mathcal{B}$ to $\mathcal{B}$, that approximate T at v^* and for initial values from $\mathcal{F}_0(v^*)$. Further, assume that $\mathcal{F}_0$ is invariant under $\mathcal{T}$. Let $V_0 \in \mathcal{F}_0(v^*)$, and define $V_{t+1} = T_t(V_t, V_t)$. If there exist random functions $0 \leq F_t(x) \leq 1$ and $0 \leq G_t(x) \leq 1$ satisfying the conditions below with probability 1 (w. p. 1), then V_t converges to a point $(v^* - S)$[3] w. p. 1 in the norm of $\mathcal{B}(\mathscr{X})$:*

[3]Note that the variable S here does not denote the state space but a deviation from the fixed point v^*. We use the (calligraphic) $\mathcal{S}$ to denote the state space.

1. For all U_1 and $U_2 \in \mathcal{F}_0$ and all $x \in \mathscr{X}$,

$$T_t(U_1, v^*)(x) - T_t(U_2, v^*)(x) = G_t(x)(U_1(x) - U_2(x)).$$

2. For all U and $V \in \mathcal{F}_0$, and all $x \in \mathscr{X}$, we can find a finite sequence $k_t(x)$ such that,

$$T_t(U, v^*)(x) - T_t(U, V)(x) = F_t(x)(||v^* - V|| + \lambda_t + k_t(x)||v^* - V||)$$

where $\lambda_t \to 0$ w. p. 1 as $t \to \infty$ and $k_t(x)$ is finite for all values of x and t.

3. $k_t(x)$ converges to a finite point (independent of time) $K(x)$, as $t \to \infty$.

4. For all $l > 0$, $\Pi_{t=l}^{n} G_t(x)$ converges to 0 uniformly in x as $n \to \infty$.

5. There exists $0 \leq \gamma < 1$ such that for all $x \in \mathscr{X}$ and large enough t, $F_t(x) = \gamma(1 - G_t(x))$

The point S can be represented by the equation $S(x) = \frac{1}{\hat{\beta}}(\gamma C_1 + K(x)C_1)$, if $K(x) \neq 0$, $\forall x \in \mathscr{X}$, where $0 < \hat{\beta} \leq 1$ and C_1 is a small positive constant. If $K(x) = 0$ $\forall x \in \mathscr{X}$, then $S(x) = 0$ $\forall x$.

Before providing the proof of Theorem 1, we provide an intuitive grasp of the result by relating the different variables in Theorem 1 to RL. The operators T and $\mathcal{T}$ are similar to Bellman operators commonly seen in Q-learning, where T_t is the component of $\mathcal{T}$ at time t. The $\mathcal{B}$ is the space of all Q-functions and $\mathcal{F}_0$ provides a mapping on the subsets of $\mathcal{B}$ (since a set of n elements has 2^n subsets, the $\mathcal{F}_0$ maps to $2^{\mathcal{B}}$). Specific instances (U, V) of $\mathcal{F}_0$ are considered. These can be seen as particular Q-functions. The variable x denotes the parameters of the Q-function (state, action pair). Here v^* is the fixed point of the Q-function (Q^*). The $G_t(x)$ and $F_t(x)$ are functions of the learning rate (α_t). These relations will become more explicit in our upcoming results.

Although our proof for Theorem 1 is structurally similar to that from Szepesvari and Littman [262], there are significant differences in our detailed proof arguments which stems from the differences in the nature of our results. First, [262] required an inequality condition in condition 2 to hold for all x and all t. We do not have this requirement. Our condition 2 uses an exact equivalence, includes an additional term to capture the difference in the other terms and the constraint on this additional term needs to hold only as $t \to \infty$ (condition 3). As a consequence, we are restricted to showing convergence to a point close to the fixed point v^*, instead of exactly to v^*. Second, as discussed in [262], condition 2 in their theorem combined with the other conditions turns T_t operator into a contraction condition for all t which is hard to satisfy or ensure in multi-agent environments. While [104] use a very restrictive assumption to overcome this problem, we show that this problem can be altogether avoided using our Theorem 1 (which is the core motivation for this theorem). We also use an exact equivalence in condition 1 and condition 5 to avoid the contraction condition. The complete proof of Theorem 1 is given next.

Proof. Let U_0 be a value function in $\mathcal{F}_0(v^*)$ and let $U_{t+1} = T_t(U_t, v^*)$. Since T_t approximates T at v^*, U_t converges to $Tv^* = v^*$ w. p. 1 uniformly over $\mathscr{X}$. Let

$$\delta_t(x) = U_t(x) - V_t(x) \tag{3.6}$$

and let,

$$\Delta_t(x) = v^*(x) - U_t(x). \tag{3.7}$$

We know that $\Delta_t(x)$ converges to 0 because U_t converges to v^*. We will show that δ_t converges to a point (independent of t) S, w. p. 1, which implies that V_t converges to the point $(v^* - S)$.

Now from the conditions of Theorem 1 we have,

$$\begin{aligned}
&\delta_{t+1}(x) = U_{t+1}(x) - V_{t+1}(x)\\
&= T_t(U_t, v^*)(x) - T_t(V_t, V_t)(x)\\
&= T_t(U_t, v^*)(x) - T_t(V_t, v^*)(x) + T_t(V_t, v^*)(x) - T_t(V_t, V_t)(x)\\
&= G_t(x)(U_t(x) - V_t(x)) + F_t(x)(||v^* - V_t|| + \lambda_t + k_t(x)||v^* - V||)\\
&= G_t(x)\delta_t(x) + F_t(x)(||v^* - V_t|| + \lambda_t + k_t(x)||v^* - V||) \qquad (3.8)\\
&= G_t(x)\delta_t(x) + F_t(x)(||v^* - U_t + U_t - V_t|| + \lambda_t + k_t(x)||v^* - U_t + U_t - V_t||)\\
&= G_t(x)\delta_t(x) + F_t(x)(||\delta_t + \Delta_t|| + \lambda_t + k_t(x)(||\delta_t + \Delta_t||))\\
&\overset{1}{\approx} G_t(x)\delta_t(x) + F_t(x)(||\delta_t|| + \lambda_t + k_t(x)||\delta_t|| + ||\Delta_t|| + k_t(x)||\Delta_t||)\\
&= G_t(x)\delta_t(x) + F_t(x)(||\delta_t|| + k_t(x)||\delta_t|| + \epsilon_t).
\end{aligned}$$

The (1) comes from the fact that Δ_t is guaranteed to converge to 0 in the limit, so the effect of splitting the sum under the norm is negligible. Regarding the last step, let us denote the term $(\lambda_t + ||\Delta_t|| + k_t(x)||\Delta_t||)$ by ϵ_t as all these terms converge to 0 in the limit. Another assumption in Theorem 1 is that $k_t(x)$ converges to $K(x)$ in the limit. We consider two cases, in which the first case is $K(x) = 0 \ \forall x \in \mathcal{X}$ and the second case is when $K(x) \neq 0$. In the first case, notice that, the theorem will then effectively change to Theorem 1 in [262], where the authors prove that $\delta_t(x)$ converges to 0 and hence $V_t(x)$ converges to $v^*(x)$.

For the second case, we provide the proof for the process δ_t to converge to a point (independent of time) by keeping a modified process $\hat{\delta}_t$ bounded by rescaling δ_t. Since δ_t is a homogeneous process, it can be written in the form $\delta_{t+1} = G_t(\delta_t, ||\Delta_t|| + \lambda_t)$, such that $\hat{\beta} G_t(x, y) = G_t(\hat{\beta} x, \hat{\beta} y)$ holds for all $\hat{\beta} > 0$. Now we will prove that, if $\hat{\delta}_t$ converges to a point S, then δ_t will also converge to another point, that is $\frac{1}{\hat{\beta}} S$, where $\hat{\beta}$ is the scale factor applied to δ_t to get the modified process $\hat{\delta}_t$.

Similar to Szepesvari and Littman [262], we will begin by considering a homogenous process and another equivalent formulation of this process that can be obtained by keeping it bounded by scaling.

Let us consider a process of the form,

$$x_{t+1} = G_t(x_t, \epsilon_t) \tag{3.9}$$

where $G_t : \mathcal{B} \times \mathcal{B} \rightarrow \mathcal{B}$ is a homogeneous random function, i.e.,

$$G_t(\hat{\beta} x, \hat{\beta} \epsilon) = \hat{\beta} G_t(x, \epsilon) \tag{3.10}$$

holds for all $\hat{\beta} > 0$, x and ϵ. We want to prove that x_t converges to some point independent of t.

Now, consider another process, that is obtained from modifying process in Eq. 3.9 by keeping it bounded by re-scaling, namely the process

$$y_{t+1} = \begin{cases} G_t(y_t, \epsilon_t) & \text{if:} ||G_t(y_t, \epsilon_t)|| \leq C \\ C G_t(y_t, \epsilon_t) / ||G_t(y_t, \epsilon_t)|| & \text{otherwise} \end{cases} \tag{3.11}$$

We denote the solution of Eq. 3.9 corresponding to an initial condition of $x_0 = \omega$ and a sequence $\epsilon = \{\epsilon_k\}$ by $x_t(\omega, \epsilon)$. Similarly, we denote the solution of Eq. 3.11 corresponding to the initial condition of $y_0 = \omega$ and the sequence ϵ by $y_t(\omega, \epsilon)$.

Next, we state a lemma that provides a relationship between convergence of the sequence represented by x_t and the sequence represented by y_t. The result is used later in Lemma 4.

Lemma 1. *Let us fix an arbitrary positive constant C, an arbitrary w_0, and a sequence ϵ. Then provided that*

(i) $y_t(w_0, \epsilon)$ converges to a point (independent of t) $\mathcal{D}$.

(ii) The sequence ϵ converges to 0 in the limit ($t \to \infty$).

The homogeneous process $x_t(w_0, \epsilon)$ converges to a point $\frac{1}{\hat{\beta}}\mathcal{D}$ w. p. 1, where $\hat{\beta}$, satisfying $0 < \hat{\beta} \leq 1$, is the scaling factor applied.

Next, we state another lemma that provides conditions for the convergence of a cascade of two converging processes. Again, this result will be used later in Lemma 4.

Lemma 2. *Let X and Y be normed vector spaces, $U_t : X \times Y \to X (t = 0, 1, 2, \dots)$ be a sequence of mappings, and $\theta_t \in Y$ be an arbitrary sequence. Let $\theta_\infty \in Y$ and $x_\infty \in X$. Consider the sequences $x_{t+1} = U_t(x_t, \theta_\infty)$, and $y_{t+1} = U_t(y - t, \theta_t)$ and suppose that x_t and θ_t converge to x_∞ and θ_∞ respectively, in the norm of the appropriate spaces.*

Let L_k^θ be the uniform Lipschitz index of $U_k(x, \theta)$ with respect to θ at θ_∞ and, similarly, let $L_t^{\mathscr{X}}$ and L_t^θ satisfy the relations $L_t^\theta \leq C(1 - L_t^{\mathscr{X}})$, and $\Pi_{m=t}^{\infty} L_m^{\mathscr{X}} = 0$ where $C > 0$ is some constant and $t = 0, 1, 2, \dots$, then $\lim_{t \to \infty} ||y_t - x_\infty|| = 0$.

Now, we show that stochastic processes having certain special structure will converge to some point independent of t, under a set of conditions. This result will also be used later in the proof of Lemma 4.

Lemma 3. *Let $\mathcal{Z}$ be an arbitrary set and consider the process*

$$x_{t+1}(z) = G_t(z)x_t(z) + F_t(z)(C + k_t(z)C) \tag{3.12}$$

where $x_1, F_t, G_t \geq 0$ are random processes, $||x_1|| < C < \infty$ w. p. 1 for some $C > 0$, and z is an element in $\mathcal{Z}$. Assume that for all k, $\lim_{n \to \infty} \Pi_{t=k}^{n} G_t(z) = 0$ uniformly in z w. p. 1 and $F_t(z) = \gamma(1 - G_t(z))$, for some $0 \leq \gamma < 1$, and $\forall z \in \mathcal{Z}$, w. p. 1. Also, $k_t(z)$ converges to $K(z)$ in the limit. Then, $x_t(z)$ converges to a point $D(z) = \gamma(C + K(z)C)$ w. p. 1.

Given the previous results, we are now in a position to conclude the proof by showing that a stochastic process that can be represented by Eq. 3.8 will converge to a point independent of t, which is our required result.

Lemma 4. *Consider an equation of the form*

$$x_{t+1}(z) = G_t(z)x_t(z) + F_t(z)(||x_t|| + \epsilon_t + k_t(z)||x_t||) \tag{3.13}$$

where the sequence ϵ_t converges to zero w. p. 1. Assume that for all k, $\lim_{n\to\infty}\Pi_{t=k}^{n}G_t(z) = 0$ uniformly in z w. p. 1 and $F_t(z) = \gamma(1 - G_t(z))$, for some $0 \le \gamma < 1$, and $\forall z \in \mathcal{Z}$, w. p. 1. Assume further that $k_t(z)$ is finite, and it converges to $K(z)$ in the limit ($t \to \infty$). Then $x_t(z)$ converges to a point represented by $S'(z) = \frac{1}{\hat{\beta}}(\gamma C_1 + K(z)C_1)$, where C_1 is a small positive constant, w. p. 1 uniformly over $\mathcal{Z}$. Here $\hat{\beta}$ is a scaling factor satisfying $0 < \hat{\beta} \le 1$.

Hence, we have proved that Eq. 3.6 has converged to a point independent of t, and thus Theorem 1 follows. The expression for point S is derived in Lemma 4.

□

Let us define a relaxation process of the form

$$V_{t+1}(x) = (1 - f_t(x))V_t(x) + f_t(x)[P_tV_t](x) \tag{3.14}$$

where $0 \le f_t(x) \le 1$ is a relaxation parameter and the sequence $P_t : \mathcal{B}(\mathscr{X}) \to \mathcal{B}(\mathscr{X})$ can be considered a randomized version of the operator T. Let us consider a process U_t such that $\mathbb{E}[V_t(x)] = U_t(x)$. Also let, $\mathbb{E}[P_tV_t](x) = TV(x)$. Now we can state the following corollary to Theorem 1.

Corollary 1. *Assume that the process defined by*

$$U_{t+1}(x) = (1 - f_t(x))U_t(x) + f_t(x)[P_tv^*](x) \tag{3.15}$$

converges to v^* w. p. 1. Assume further that the following conditions hold:

1. *There exist a scalar γ satisfying $0 \le \gamma < 1$ and a sequence $\lambda_t \ge 0$ converging to 0 w. p. 1 such that $||P_tv^* - P_tV|| = \gamma||v^* - V|| + \lambda_t + \gamma k_t(x)||v^* - V||$ holds for all $V \in \mathcal{B}(\mathscr{X})$ and for finite $k_t(x)$.*
2. *$k_t(x)$ converges to finite point $K(x)$ as t goes to ∞.*
3. *$0 \le f_t(x) \le 1$, $t \ge 0$, and $\sum_{t=1}^{n} f_t(x)$ goes to infinity uniformly in x as $n \to \infty$.*

Then, the iteration defined by Eq. 3.14 converges to a point $(v^* - S)$, where S is defined as in Theorem 1.

Proof. Here, P_t is a randomized version of an operator T. It can be proved that a process of the form,

$$U_{t+1}(x) = (1 - f_t(x))U_t(x) + f_t(x)[P_tV](x) \tag{3.16}$$

will converge to TV w. p. 1 where $V \in \mathcal{B}(\mathscr{X})$. The convergence is due to Lemma 5 below.

Lemma 5. *Let $\mathcal{F}_t$ be an increasing sequence of σ-fields, let $0 \leq \alpha_t$ and w_t be random variables such that α_t and w_{t-1} are $\mathcal{F}_t$ measurable. Assume that the following hold w. p. 1: $E[w_t|\mathcal{F}_t, \alpha_t \neq 0] = A$, $E[w_t^2|\mathcal{F}_t] < B < \infty$, $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < C < \infty$ for some $B, C > 0$. Then the process*

$$Q_{t+1} = (1 - \alpha_t)Q_t + \alpha_t w_t \tag{3.17}$$

converges to A w. p. 1.

Let the random operator sequence $T_t : \mathcal{B}(\mathscr{X}) \times \mathcal{B}(\mathscr{X}) \to \mathcal{B}(\mathscr{X})$ be defined by

$$T_t(U, V)(x) = (1 - f_t(x))U(x) + f_t(x)[P_t V](x). \tag{3.18}$$

We know that the operator T_t approximates T at v^*, since the process defined by Eq. 3.16 converges to TV for all $V \in \mathcal{B}(\mathscr{X})$ by Lemma 5 and the process defined by Eq. 3.15 converges to v^* by definition. Moreover, observe that V_t as defined by Eq. 3.14 satisfies $V_{t+1} = T_t(V_t, V_t)$. Due to conditions 1,2, and 3, it can be readily verified that coefficients $G_t(x) = 1 - f_t(x)$, and $F_t(x) = \gamma f_t(x)$ satisfy the rest of the conditions of Theorem 1, and this yields that the process V_t converges to $(v^* - S)$ w. p. 1.

□

Now, we define the P_t operator in the context of Algorithm 2.

Definition 9. *Let $Q = (Q^1, \dots, Q^n)$. We define an operator $P_t : \mathcal{Q} \to \mathcal{Q}$ such that $P_t Q = (P_t Q^1, \dots, P_t Q^n)$, where*

$$P_t Q^k(s, \boldsymbol{a}) = r_t^k(s, \boldsymbol{a}) + \beta \sigma_t^1(s') \cdots \sigma_t^n(s') Q^k(s') \tag{3.19}$$

and $k = (1, \dots, n)$, s' is the state at time $t+1$, and $(\sigma_t^1(s'), \dots, \sigma_t^n(s'))$ is an advisor solution for the stage game $(Q^1(s'), \dots, Q^n(s'))$ at time t.

The P_t operator's value depends on the advisor solution. Next, we will state some assumptions. The first two are commonly used in RL [111, 229].

Assumption 1. *Every state $s \in \mathcal{S}$ and action $a^j \in A^j$, for each agent $j \in \{1, \dots, n\}$, are visited infinitely often, and the reward function for all agents stay bounded.*

Assumption 2. *For all s, t, and $\boldsymbol{a}$, $0 \leq \alpha_t(s, \boldsymbol{a}) < 1$, $\sum_{t=0}^{\infty} \alpha_t(s, \boldsymbol{a}) = \infty$, $\sum_{t=0}^{\infty} [\alpha_t(s, \boldsymbol{a})]^2 < \infty$.*

Assumption 3. *The advisor solution to any stage game $(Q^1(s), \dots, Q^n(s))$ pertaining to state $s \in \mathcal{S}$ will become stationary in the limit ($t \to \infty$). In other words, the advisor can adapt its solutions, but in the limit it is guaranteed to settle down and provide the same advisor strategy for a particular state $s \in \mathcal{S}$.*

Assumption 3 allows the advisor to learn and adapt during gameplay. However, in the limit, the advisor is required to converge and provide the same strategy. Now, directly, it can be seen that this assumption is much weaker than the restrictive assumption in Hu and Wellman [104]. Firstly, Assumption 3 does not involve the computation of Nash equilibrium of every stage game in the stochastic game. Secondly, agents need not use the Nash equilibrium of every stage game to update their Q-values unlike the assumption in Hu and Wellman [104]. If this condition were needed, then the advisor will need to only suggest the Nash equilibrium at every stage game, greatly reducing the scope of the advisor. Thirdly, the Nash equilibrium at every stage game does not need to be a global optimum or saddle point. This condition in [104] is not satisfied in any practical game, which we have relaxed.

Next, we state a lemma needed to prove convergence.

Lemma 6. *Assume that α_t satisfies Assumption 2 and the mapping $P_t : \mathcal{Q} \to \mathcal{Q}$ satisfies the condition that, there exists a scalar γ satisfying $0 \le \gamma < 1$, a sequence $\lambda_t \ge 0$ converging to zero w. p. 1, and a finite sequence $k_t(s)$ such that $||P_t Q - P_t Q_*|| = \beta ||Q - Q_*|| + \lambda_t + \beta k_t(s) ||Q - Q_*||$ for all Q, and all $s \in \mathcal{S}$. Assume further that, $k_t(s)$ converges to a finite point $K(s)$ in the limit. Additionally, $Q_*(s, \boldsymbol{a}) = E[P_t Q_*(s, \boldsymbol{a})]$, then the iteration defined by*

$$Q_{t+1}(s, \boldsymbol{a}) = (1 - \alpha_t) Q_t(s, \boldsymbol{a}) + \alpha_t [P_t Q_t(s, \boldsymbol{a})] \tag{3.20}$$

converges to $(Q_* - S)$ w. p. 1, where S is as given in Theorem 1.

Theorem 2 proves that the Q-updates in Algorithm 2, converges to an ϵ-equilibrium consistent with Definition 7. The Q-values at convergence denotes the value of the advisor. The proof of this theorem will be an application of Lemma 6. The bound ϵ of the ϵ-equilibrium will depend on the advisor through its advisor solutions.

As proved by [67], every stochastic game is guaranteed to have at least one Nash equilibrium point. However, there can be more than one Nash equilibrium point, in which case, the Q_* in Theorem 2, could be the Nash Q-function of any Nash equilibrium strategy. We do not require the Nash equilibrium point of the stochastic game to be unique in Theorem 2 as assumed in prior works [104].

Theorem 2. *Under the Assumptions 1, 2, and 3, the Q-functions updated by Eq. 3.2 converges to a bounded distance from the Nash Q-function $Q_* = (Q_*^1, \ldots, Q_*^n)$, represented as $(Q_* - S)$, in the limit $(t \to \infty)$. The point S is as given in Theorem 1.*

Proof. We will state a lemma before we begin the proof of the theorem.

Lemma 7. *For a n-player stochastic game, $E[P_t Q_*] = Q_*$ where $Q_* = (Q_*^1, \ldots, Q_*^n)$.*

The proof of the Theorem 2 is a direct application of Lemma 6, which establishes convergence given three conditions. The condition pertaining to the expectation relationship is satisfied by Lemma 7.

To satisfy the first condition (distances between Q-functions), we will begin by providing two expressions pertaining to distances between Q-functions seen in Lemma 6. The first expression gives a formal way of determining the distance between any Q-function and the Nash Q-function.

Consider two Q functions, $Q, Q_* \in \mathcal{Q}$. Then we can get,

$$\begin{aligned} &||Q - Q_*|| \triangleq max_j max_s ||Q^j(s) - Q_*^j(s)|| \\ &= max_j max_s max_{a^1,\ldots,a^n} ||Q^j(s, a^1, \ldots, a^n) - Q_*^j(s, a^1, \ldots, a^n)|| \end{aligned} \tag{3.21}$$

where $Q_* = (Q_*^1, \ldots, Q_*^n)$, Q_*^j is the Nash Q-function of the agent j.

The second expression below pertains to distances under corresponding P_t operators. Consider two Q functions $Q, Q_* \in \mathcal{Q}$. Then we can get,

$$\begin{aligned} &||P_t Q - P_t Q_*|| \triangleq max_j ||P_t Q^j - P_t Q_*^j|| \\ &= max_j max_s ||\beta \sigma_t^1(s) \cdots \sigma_t^n(s) Q^j(s) - \beta \sigma_{*,t}^1(s) \cdots \sigma_{*,t}^n(s) Q_*^j(s)|| \\ &= max_j \beta ||\sigma_t^1(s) \cdots \sigma_t^n(s) Q^j(s) - \sigma_{*,t}^1(s) \cdots \sigma_{*,t}^n(s) Q_*^j(s)||. \end{aligned} \tag{3.22}$$

Here, $(\sigma_t^1(s), \ldots, \sigma_t^n(s))$ is an advisor solution for the stage game $(Q^1(s), \ldots, Q^n(s))$ at time t and $(\sigma_{*,t}^1(s'), \ldots, \sigma_{*,t}^n(s'))$ is an advisor solution for the stage game $(Q_*^1(s'), \ldots, Q_*^n(s'))$ at time t. Also Q_*^j is the Nash Q-function of agent j.

In the second step, the corresponding reward terms from the P_t operators get cancelled from Definition 9. Since the state s changes for every t, the dependence on max_s in Eq. 3.22 is removed in the last step.

$$||P_tQ - P_tQ_*|| = \beta||Q - Q_*|| + \lambda_t + \beta k_t(s)||Q - Q_*|| \tag{3.23}$$

We state the equation in Lemma 6 again in Eq. 3.23. Now, we can find a finite $k_t(s)$ such that the Eq. 3.23 is satisfied for all $Q \in \mathcal{Q}$ and all $s \in \mathcal{S}$. This is due to the fact that all the other terms in that expression are guaranteed to be finite due to Assumption 1. This satisfies another condition of Lemma 6.

To satisfy the last condition of Lemma 6, we rewrite the Eq. 3.23, to get an expression for $k_t(s)$ in the limit $(t \to \infty)$,

$$\begin{aligned}
k_t(s) &= \tfrac{||P_tQ - P_tQ_*||}{\beta||Q - Q_*||} - 1 \\
k_t(s) &= \tfrac{\max_j \beta|\sigma_t^1(s)\cdots\sigma_t^n(s)Q^j(s) - \sigma_{*,t}^1(s)\cdots\sigma_{*,t}^n(s)Q_*^j(s)|}{\beta||Q - Q_*||} - 1 \\
k_t(s) &= \tfrac{\max_j |\sigma^1(s)\cdots\sigma^n(s)Q^j(s) - \sigma_*^1(s)\cdots\sigma_*^n(s)Q_*^j(s)|}{||Q - Q_*||} - 1 \\
k_t(s) &= K(s).
\end{aligned} \tag{3.24}$$

The first expression in Eq. 3.24 is obtained as λ_t is guaranteed to go to 0 in the limit. In the second expression, Eq. 3.22 is being used. The third expression is from Assumption 3. Now, from Eq. 3.24, we can see that $k_t(s)$ is a constant for a given state that has no dependence on time t. This satisfies the last condition of Lemma 6.

Hence, all the conditions of Lemma 6 are satisfied and, therefore, the process $Q_{t+1} = (1 - \alpha_t)Q_t + \alpha_t[P_tQ_t]$ converges to a bounded distance from the Nash equilibrium $(Q_* - S)$. Thus, the equilibrium point reached is an epsilon equilibrium, where the epsilon value can be given by the point S. The expression for this point is as given in Theorem 1 with the value of K being given in Eq. 3.24. Here the agents can still unilaterally deviate and possibly obtain an additional payoff consistent with Definition 7. □

Now, we move on to providing theoretical guarantees for Algorithm 1. We will show in Theorem 3 that this algorithm will converge to a Nash equilibrium under a set of assumptions. Again, these assumptions are weaker than others previously considered in literature. To prove this convergence result for Algorithm 1 we will retain the first two assumptions but replace Assumption 3 with two other assumptions.

Assumption 4. *The algorithm is Greedy in the limit with Infinite Exploration (GLIE).*

Assumption 5. *The Nash equilibrium is a global optimum or saddle point in every stage game of the stochastic game.*

Assumption 4 allows the policy to explore, but in the limit ($t \to \infty$) this assumption requires the policy to choose greedy actions. Assumption 5 is strong, however, [104] show that similar assumptions are required to prove convergence in theory but not necessary to observe convergence in practice, which is consistent with our observations as well. Thus, even if such assumptions are violated in practice, convergence is still observed. Further, it is to be noted that Assumption 5 is a weaker condition than the assumption in Hu and Wellman [104], since it does not require calculating the Nash equilibrium at each stage game and using the same to update the Q-values.

Theorem 3. *When we update the Q-functions according to Eq. 3.1, they converge to the Nash Q-function under Assumptions 1, 2, 4, and 5, in the limit ($t \to \infty$).*

Theorem 3 proves that the Q-updates in Algorithm 1 converges to the Nash equilibrium strategies. The proof is along the lines of Theorem 1 in [229], but involves significant modifications to cater to the multi-agent scenario.

Proof. The proof will involve using two lemmas from previous work, one from Jaakkola et al. [111] and the other from Hu and Wellman [104]. We will start the proof of this theorem by stating the first lemma.

Lemma 8. *A random iterative process*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x) \tag{3.25}$$

where $x \in X$, $t = 0, 1, \ldots, \infty$, converges to zero with probability one (w. p. 1) if the following properties hold:

1. The set of possible states X is finite.

2. $0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t^2(x) < \infty$ w. p. 1, where the probability is over the learning rates α_t.

3. $||\mathbb{E}\{F_t(x)|\mathscr{P}_t\}||_W \leq \mathscr{K}||\Delta_t||_W + c_t$, where $\mathscr{K} \in [0, 1)$ and c_t converges to zero w. p. 1.

4. $\boldsymbol{var}\{F_t(x)|\mathscr{P}_t\} \leq K(1 + ||\Delta_t||_W)^2$, where K is some constant.

*Here $\mathscr{P}_t$ is an increasing sequence of σ-fields that includes the past of the process. In particular, we assume that $\alpha_t, \Delta_t, F_{t-1} \in \mathscr{P}_t$. The notation $||\cdot||_W$ refers to some (fixed) weighted maximum norm and the notation **var** refers to the variance.*

Let us define a Nash operator P_t, consistent with the definition in [104]. The Nash operator is defined using the following equation,

$$P_t Q^k(s, a^1, \ldots, a^n) = \mathbb{E}_{s' \sim p}[r_t^k(s, a^1, \ldots, a^n) + \gamma \pi_*^1(s') \cdots \pi_*^n(s') Q^k(s')] \tag{3.26}$$

where s' is the state at time $t+1$, $(\pi_*^1(s'), \ldots, \pi_*^n(s'))$ is the Nash equilibrium solution for the stage game $(Q^1(s'), \ldots, Q^n(s'))$, and p is the transition function. Q^k denotes the Q-value of a representative agent k.

Lemma 9. *Under Assumption 5, the Nash operator as defined in Eq. 3.26 forms a contraction mapping with the fixed point being the Nash Q-value of the game.*

Now, since the P_t operator forms a contraction mapping, $||P_t Q - P_t Q_*|| \leq \beta ||Q - Q_*||$, is satisfied for some $\beta \in [0, 1)$ and all Q. Here Q_* is the Nash Q-value.

We will apply Lemma 8 to show that the Q-values converge to the Nash Q-value. We will drop the agent index in all the expressions for simplicity. The rest of the proof is conducted for the Q-values of a representative agent.

The first two conditions of Lemma 8 are satisfied from the assumptions. Comparing Eq. 3.25 and Eq. 3.1 we get that x can be associated with the state action pairs $(s, a^1, \ldots, a^n)$ and $\Delta_t(s_t, a_t)$ can be associated with $Q_t(s, a^1, \ldots, a^n) - Q_*(s, a^1, \ldots, a^n)$. Here, $Q_*(s, a^1, \ldots, a^n)$ can be considered as the Nash Q-value (Q-values under the Nash Q-function).

Now we get

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x) F_t(x), \tag{3.27}$$

where

$$\begin{aligned} F_t(x) &= r_t + \gamma v^{Nash}(s_{t+1}) - Q_*(s_t, a_t^1, \ldots, a_t^n) + \gamma[Q_t(s_{t+1}, a_{t+1}^1, \ldots, a_{t+1}^n) - v^{Nash}(s_{t+1})] \\ &\triangleq r_t + \gamma v^{Nash}(s_{t+1}) - Q_*(s_t, a_t^1, \ldots, a_t^n) + C_t(s_t, a_t^1, \ldots, a_t^n) \\ &\triangleq F_t^Q(s_t, a_t^1, \ldots, a_t^n) + C_t(s_t, a_t^1, \ldots, a_t^n). \end{aligned} \tag{3.28}$$

We define $F_t(s_t, a_t^1, \ldots, a_t^n) = F_t^Q(s_t, a_t^1, \ldots, a_t^n) = C_t(s_t, a_t^1, \ldots, a_t^n) = 0$ if $(s, \boldsymbol{a}) \neq (s_t, \boldsymbol{a}_t)$. We also define $\boldsymbol{a} = (a_1, \ldots, a_n)$ and $\boldsymbol{a}_t = (a_{1,t}, \ldots, a_{n,t})$. Let the σ-field generated by all the random variables $(s_t, \alpha_t, a_t^1, \ldots, a_t^n, r_{t-1}, \ldots, s_1, \alpha_1, a_1, Q_0)$ be represented by $\mathscr{P}_t$. Now, all the Q-values are $\mathscr{P}_t$ measurable which makes Δ_t and F_t, $\mathscr{P}_t$ measurable and this satisfies the measurability condition of Lemma 8.

Hu and Wellman [104] showed that $v^{Nash}(s_{t+1}) \triangleq v^k(s', \pi_*^1, \ldots, \pi_*^n) = \pi_*^1(s') \cdots \pi_*^n(s') Q_*^k(s')$ (see the proof in Lemma 10 of [104]). Hence, from Lemma 9, we can show that the $\mathbb{E}[F_t^Q]$ forms a contraction mapping. This can be done using the fact that $\mathbb{E}(P_t Q_*) = Q_*$ (Lemma 7). Here, the norm is the maximum norm on the joint action.

Now, we have the following for all t,

$$||\,\mathbb{E}[F_t^Q(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]|| \leq \gamma ||Q_t(s_t, a_t^1, \ldots, a_t^n) - Q_*(s_t, a_t^1, \ldots, a_t^n)|| = \gamma ||\Delta_t||. \tag{3.29}$$

Now from Eq. 3.28,

$$\begin{aligned} &||\,\mathbb{E}[F_t(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]|| \leq ||\,\mathbb{E}[F_t^Q(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]|| + ||\,\mathbb{E}[C_t(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]|| \\ &\leq \gamma ||\Delta_t|| + ||\,\mathbb{E}[C_t(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]||. \end{aligned} \tag{3.30}$$

This satisfies the third condition of Lemma 8 if $c_t = ||\,\mathbb{E}[C_t(s_t, a_t^1, \ldots, a_t^n)|\mathscr{P}_t]||$ converges to 0 w. p. 1.

Let us rewrite the definition of the C_t,

$$\begin{aligned} C_t(s_t, a_t^1, \ldots, a_t^n) &= \gamma[Q_t(s_{t+1}, a_{t+1}^{1'}, \ldots, a_{t+1}^{n'}) - v^{Nash}(s_{t+1})] \\ C_t(s_t, a_t^1, \ldots, a_t^n) &= \gamma[\max Q_t(s_{t+1}, a_{t+1}^{1'} \ldots, a_{t+1}^{n'}) - v^{Nash}(s_{t+1})] \\ C_t(s_t, a_t^1, \ldots, a_t^n) &= \gamma[v(s_{t+1}) - v^{Nash}(s_{t+1})]. \end{aligned} \tag{3.31}$$

In the second step, we are using the assumption that the Q-value is GLIE. The max operator operates over the action space of the representative agent.

According to Assumption 5, the Nash equilibrium could only be a global optimum or a saddle point. Now, if it is a global optimum, the value of maximizing all the actions in Eq. 3.31 will lead to the global optimum for all the agents and this will be the Nash payoff, thus leading to C_t evaluating to 0 in the limit. Furthermore, [104] show that a global optimum is always a Nash equilibrium and all global optima are guaranteed to have equal values. Alternatively, if the Nash equilibrium is a saddle point, consider a stage game, with saddle point equilibrium payoff, σ and π. Then, $\sigma^k\sigma^{-k}Q^k(s) \geq \pi^k\sigma^{-k}Q^k(s)$, as deviating from the equilibrium when the others are playing the equilibrium strategy will leave an agent worse off by definition of a Nash equilibrium. Also, $\pi^k\pi^{-k}Q^k(s) \leq \pi^k\sigma^{-k}Q^k(s)$, as in a saddle point, if others deviate the agent should be better off (see Definition 13 in [104]). Thus, we will get the relation, $\pi^k\pi^{-k}Q^k(s) \leq \sigma^k\sigma^{-k}Q^k(s)$. Since σ and π are saddle points, the previous argument holds without the loss of generality. Hence, the following is also true, $\sigma^k\sigma^{-k}Q^k(s) \leq \pi^k\pi^{-k}Q^k(s)$. Thus, the value obtained is the same in the saddle points and the value would be Nash value if all the agents are being greedy given the strategies of all other agents. Thus, we have proved that for all cases C_t converges to 0 in the limit.

The fourth condition of the Lemma 8 is satisfied since we have the reward to be bounded (Assumption 1) and we know the variance of F_t^Q is bounded [111].

Thus, it follows from Lemma 8 that the process Δ_t converges to 0 and hence, Q_t converges to the Nash Q-function Q_*. $\square$

Theorem 3 shows that complicated steps in algorithms such as Nash-Q [104] to predict the actions of other agents using an equilibrium calculation are unnecessary. The other agents' current policy can be used directly instead of the equilibrium calculations. Assumption 5 is strong enough to ensure that such processes converge. Also, Theorem 3 proves convergence without any restrictions on the nature of advisors. They could be sub-optimal or adversarial. Thus, in both the Theorem 2 and Theorem 3, we have proved fixed point guarantees in general-sum stochastic games with weaker assumptions than those used by earlier work [104]. We also note that Theorem 3 just assumes that the advisor influence decays to 0 in the limit, and so is also applicable to learning algorithms without advisors.

To conclude, from the Theorem 2 and Theorem 3 we see that both our algorithms (i.e., ADMIRAL-AE and ADMIRAL-DM) have a suitable fixed point and a guarantee of converging to that fixed point in the limit. This shows that our algorithms are theoretically grounded.

3.4 Experiments

We experimentally validate our algorithms, showing their effectiveness in a variety of situations using different testbeds. We also demonstrate superior performance to common baselines previously used in literature. The source code for the experiments has been open sourced [247].

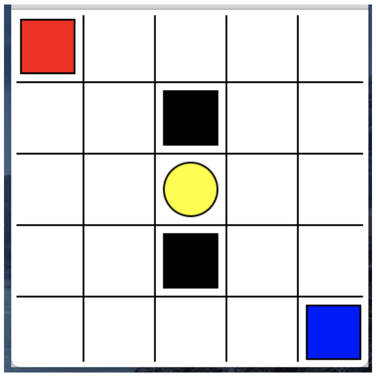


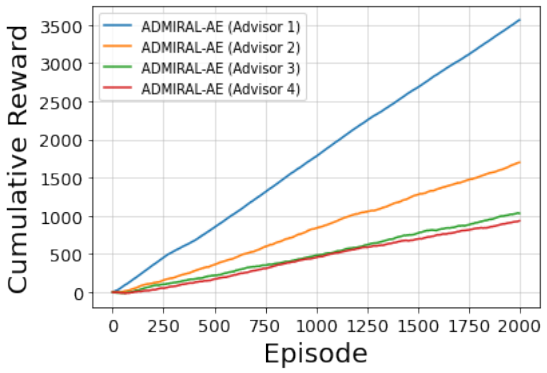
Figure 3.3: Grid Maze environment.

3.4.1 Experimental Results - Tabular Version

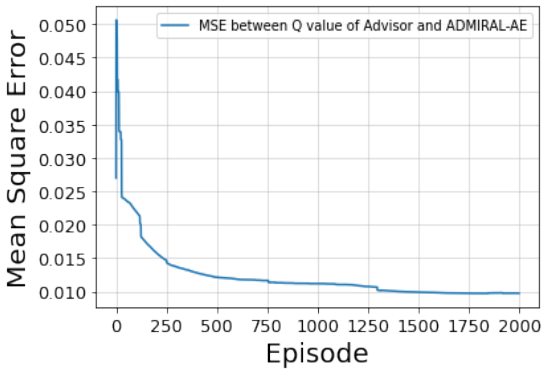
The objective of this section is to provide a simple illustration of the tabular version of our algorithms using different kinds of advisors.

For our first experiment, we investigate the performance of ADMIRAL-AE, where we control the quality of the advisors. To this end, we use a 5×5 Grid Maze environment for the empirical evaluation. The schematic representation for this environment is available in Figure 3.3. Here the red and blue agents are trying to reach the yellow goal. The agents must learn to avoid the black pitfalls while reaching the goal. This is a cooperative game where the two agents must learn to perfectly coordinate the task of reaching the goal to achieve maximum reward. However, the agents are unable to communicate directly. An agent has to learn to take the correct actions to reach the goal in relation to the observed behaviour of the other agent. The game terminates if at least one of the agents reaches the goal state or hits a pitfall. We implemented four rule-based advisors of decreasing quality, from Advisor 1, which provides the best action for each state (relative to the other advisors), to Advisor 4, which makes random suggestions. The state in this game corresponds to the positions (grid coordinates) of both the agents and the action involves moving in one of the four cardinal directions. A detailed description of the environment, reward function, action selection, and more details on the advisors are in Appendix A.4.

First we conduct the 'pre-learning' phase where we study the performance of ADMIRAL-AE using the different advisors in the Grid Maze domain. In our implementations, both agents play a separate instance of the same algorithm (ADMIRAL-AE with a particular advisor). We consider the cumulative rewards obtained by ADMIRAL-AE with each of the advisors over a period of 2000 episodes of training. Figure 3.4(a) shows the average result over five runs. Since perfect coordination gives the large positive rewards, we can see from



(a) Performance of ADMIRAL-AE using 4 different advisors.



(b) Mean Square Error (MSE), between current Q-value of ADMIRAL-AE using Advisor 1 and the Q_σ, which is the true value of the advisor.

Figure 3.4: Experimental findings using the tabular version of ADMIRAL-AE with different advisors. (a) shows that ADMIRAL-AE with the best advisor (Advisor 1) gives the best overall performance, and ADMIRAL-AE with the worst advisor (Advisor 4) gives the worst overall performance. (b) shows that the MSE between the Q-value from ADMIRAL-AE and the value of the advisor Q_σ progressively reduces, and hence ADMIRAL-AE evaluates correctly. All results show an average of five runs, and they have negligible standard deviation.

Figure 3.4(a) that the ADMIRAL-AE implementation with the best advisor (Advisor 1) results in the highest performance for the task. The performance progressively degrades from Advisor 1 to Advisor 4. We highlight that the performance of ADMIRAL-AE depends on the quality of the advisor, with the best advisor (Advisor 1) leading to the best overall performance. This also shows that we would find most value in learning from Advisor 1, as compared to the other advisors. Thus, from Figure 3.4(a) we conclude that there is great value in using ADMIRAL-AE when the quality and suitability of advisors are the objective of study.

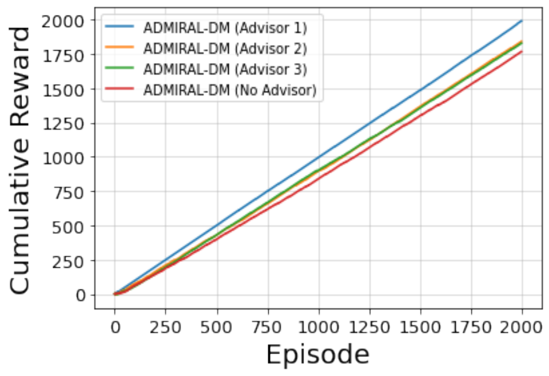
Further, we plot the mean square error (MSE) between the Q-values (of every state and joint action) obtained from playing ADMIRAL-AE (using the best advisor, Advisor 1) at the end of each episode and the true Q-value of the advisor (denoted as Q_σ). As mentioned before, the true Q-value of the advisor captures the expectation of the sum of the immediate reward and the discounted sum of future rewards obtained, when all agents follow the advisor's strategy for infinite periods starting from the current state and joint action. We obtain this value by running trajectories from each state and joint action pair until the end of the episode and calculating the expected discounted sum of rewards. The plot of the MSE is given in Figure 3.4(b). From this figure, we see that the MSE approaches close to zero after about 2000 episodes of training, which shows that the ADMIRAL-AE algorithm correctly evaluates the advisor. This experiment provides another illustration for the effectiveness of ADMIRAL-AE in evaluating the given advisor.

In Table 3.2 we use the average cumulative performances (5 runs) of ADMIRAL-AE along with all the advisors (from Figure 3.4(a)) to find a value for ϵ'_0, the hyperparameter that controls the advisor influence in ADMIRAL-DM. Recall that this is a major objective of the 'pre-learning' phase. We use the Eq. 3.3 given in Section 3.2.2. The performance of ADMIRAL-AE using each of the advisors should lie between the maximum possible performance and the performance of ADMIRAL-AE using a random advisor. We normalize the average performances between the range of $[0, 1]$, which will be used as the initial value of ϵ' (or ϵ'_0) in ADMIRAL-DM. The maximum possible performance in Table 3.2 is adjusted for random exploration, which is approximately 5% of all actions. Hence, we subtract this portion from the theoretical possible maximum performance of 4000 for this domain. The initial values of ϵ' that would pertain to the advisors, are given in the last column of Table 3.2. From this table, it can be seen that the ϵ'_0 value is high for Advisor 1, since there is good value to be gained in listening to this advisor. On the other hand, ϵ'_0 is lower for other advisors and for Advisor 4 this value is 0, which means there will be no advisor influence. Since Advisor 4 only provides random advice, the agent is better off listening to its own policy rather than following actions suggested by Advisor 4. Thus, using the values given in Table 3.2, ADMIRAL-DM would listen more to the good advisor and listen less (or not at all) to the bad advisors, as was our goal.

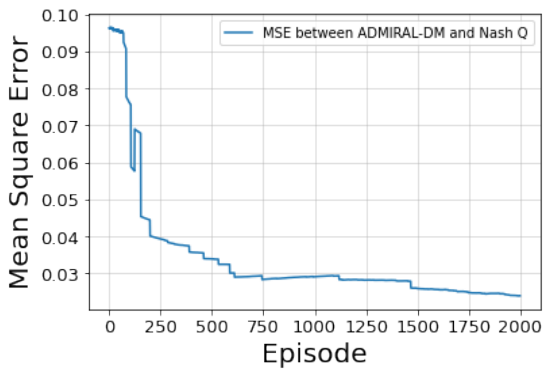
Next, we show an illustration of the tabular implementation of our ADMIRAL-DM algorithm (Algorithm 1). We use the same Grid Maze domain along with the same four advisors for this study. In this setting, we have the ADMIRAL-DM algorithm training along with each of the advisors for 2000 episodes. In each implementation, both agents use the same algorithm for training (ADMIRAL-DM with an advisor) similar to the previous experiments. We set the initial value of ϵ' (or ϵ'_0) to the values obtained from ADMIRAL-AE (given in Table 3.2). As described, since the Advisor 4 is bad, the value of ϵ'_0 is set to 0 and there is no influence from this advisor during learning. Further, for all the implementations, the advisor influence through ϵ' is linearly decayed during training, as described in the algorithmic steps for ADMIRAL-DM. More details regarding the game conditions and reward functions are given in Appendix A.4. The results (cumulative rewards) are in Figure 3.5(a), where we plot the averages of five experimental runs. We note that learning from the best advisor (Advisor 1) using ADMIRAL-DM obtains the best overall performance, while ADMIRAL-DM with the other advisors requires more episodes

Advisor	Average cumulative reward (rounded to nearest 10)	Maximum possible cumulative reward (adjusted for random exploration)	Average performance of ADMIRAL-AE using a random advisor (rounded to nearest 10)	Normalized value (rounded up to nearest first decimal)
Advisor 1	3560	3800	930	1
Advisor 2	1700	3800	930	0.3
Advisor 3	1030	3800	930	0.1
Advisor 4	930	3800	930	0

Table 3.2: Finding ϵ_0' using ADMIRAL-AE for the Grid Maze environment.



(a) Performance of ADMIRAL-DM in the Grid Maze domain.



(b) Mean Square Error (MSE), between current Q-value of ADMIRAL-DM (using Advisor 1) and Q_*, which is the Nash Q-value.

Figure 3.5: Experimental findings on the Grid Maze domain with both agents playing a tabular implementation of ADMIRAL-DM with different advisors with hyperparameter ϵ_0' obtained from Table 3.2. (a) shows that using a tuned value for ϵ_0', gives a good performance for all the implementations of ADMIRAL-DM with the different advisors. However, better advisors help in getting a relatively better performance. (b) shows that the MSE between the current Q-estimate of ADMIRAL-DM using Advisor 1, and the Nash Q-value progressively reduces. All results show an average of five runs, and they have negligible standard deviation.

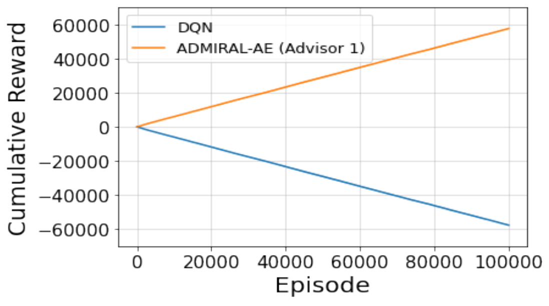
to obtain similar performances to that of the Advisor 1. Since Advisor 1 teaches useful strategies, ADMIRAL-DM using Advisor 1 sees a good performance early on in training, even when there have been only limited interactions with the environment. This shows the value of positive influences from good advisors for improving the sample efficiency of MARL algorithms.

Now, we show that the Q-values of an agent following the ADMIRAL-DM algorithm converges to the Nash Q-value of the stochastic game. In Figure 3.5(b) we plot the MSE between the Q-values (of every state and joint action) of ADMIRAL-DM using the Advisor 1, and the Nash Q-value. To obtain the Nash Q-value we construct the Nash policy and obtain the value of this policy by running trajectories from each state and joint action pair till the end of the episode and calculating the expected discounted sum of rewards (similar to obtaining the value of the advisor in the previous experiment). In this environment, the Nash equilibrium strategies will provide the actions of perfect coordination that obtains large positive rewards. The MSE in Figure 3.5(b) approaches very close to zero after 2000 episodes of training, showing that the Q-values following ADMIRAL-DM finds the Nash Q-value in the limit.

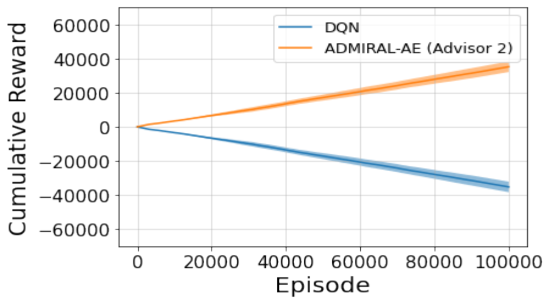
3.4.2 Experimental Results - Function Approximation - ADMIRAL-AE

We present results for our advisor evaluation algorithm (Algorithm 2) on the large state-action Pommerman environment [203]. The objective is to conduct the 'pre-learning' phase to evaluate a set of advisors and pick a suitable ϵ'_0 for learning using ADMIRAL-DM, which we study in the upcoming sub-sections. We use a two-agent version of Pommerman, which we denote as Domain OneVsOne of Pommerman (we will consider another domain of Pommerman shortly). Pommerman is a complex multi-agent domains, with each state containing more than 200 elements describing the position of the board, special features like bombs, and the position of other agents. Each agent can perform 6 actions, which include moving in the grid and laying bombs to kill the opponent. The reward function in Pommerman is quite sparse with the agents getting a $+1$ for winning the game, -1 for losing or a draw, with nothing in between. This game is general-sum since both agents get -1 for a draw. There is a maximum of 800 steps and the games where there are no winners after 800 steps are declared to be a draw. It is very hard for RL agents to learn good performance in Pommerman due to difficulties in balancing the twin goals of the killing of opponent and protecting themselves [78].

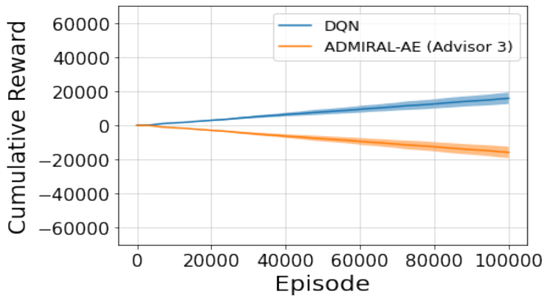
For these experiments, we use the neural network implementation of the ADMIRAL-AE



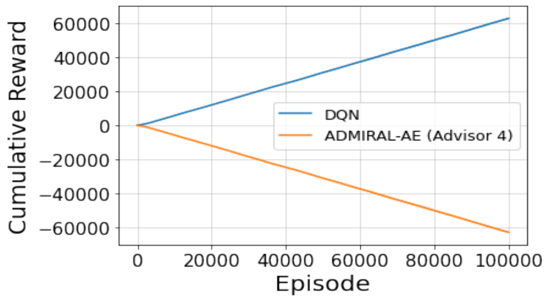
(a) Advisor 1



(b) Advisor 2



(c) Advisor 3



(d) Advisor 4

Figure 3.6: Analysis of ADMIRAL-AE algorithm on Pommerman (Domain OneVsOne) against (single-agent) DQN. The standard deviation in (a) and (d) are very small (negligible). The best advisor (Advisor 1) makes the agent reach the best overall performance. The performance steadily decreases from Advisor 1 to Advisor 4. All results are averages of 10 experimental runs ((a) and (d) have negligible standard deviations).

algorithm (Algorithm 4). We consider four different advisors. Advisors are arranged in decreasing quality of advice, with Advisor 1 providing the best advice and Advisor 4 proving the worst (e.g. random advice). The Advisors 1 and 2 have a positive influence on learning, as they can teach many useful techniques to win the game, while Advisor 4 has a negative influence on learning. The Advisor 3 is also capable of teaching some useful strategies and in general, is better than a random advisor (Advisor 4). However, it is much worse compared to Advisor 1 or Advisor 2. Appendix A.4 contains complete descriptions of the advisors and the implementation details of the algorithms used.

We conduct all experimental runs on 100,000 Pommerman games (episodes). Each episode is a full Pommerman game containing a maximum of 800 steps. Each experiment has a DQN (single-agent version as introduced in Mnih et al. [167]) agent and an ADMIRAL-AE agent training and competing against each other. The experiments analyze the performance of ADMIRAL-AE with each of the four advisors against the common opponent (DQN). The performance is plotted in Figure 3.6. We repeat the experiments 10 times and plot the averages and standard deviations. We observe that using the best advisor (Advisor 1) clearly results in the best performance of the ADMIRAL-AE algorithm, with an overall cumulative reward reaching around 60,000 (Figure 3.6(a)). The second-best advisor (Advisor 2) results in cumulative reward around 35,000 (Figure 3.6(b)). When the ADMIRAL-AE uses Advisor 3 and Advisor 4, DQN results in better performance cumulatively than the ADMIRAL-AE algorithm (Figures 3.6(c) and (d)). The results (Figure 3.6) show that the ADMIRAL-AE algorithm can distinguish between different quality advisors. From Figure 3.6, it is clear that the advisor of choice for learning in this domain is Advisor 1. This result is obtained by running a separate instance of the ADMIRAL-AE algorithm with each of the advisors. This is consistent with our description of possible ways of evaluating the advisors using the ADMIRAL-AE algorithm in Section 3.2.2.

In Table 3.3 we tabulate the results and normalize the average performances to obtain a suitable value for ϵ'_0 (using Eq. 3.3). The procedure is the same as that adopted in Section 3.4.1. We adjust the column for maximum possible performance value for 10% random exploration as done in Table 3.2. The Advisor 1 along with its initial value of ϵ' is used in the next sub-section for learning using the ADMIRAL-DM method.

Advisor	Average cumulative reward (rounded to nearest 1000)	Maximum possible cumulative reward (adjusted for random exploration)	Average performance of the ADMIRAL-AE using a random advisor (rounded to nearest 1000)	Normalized value (rounded up to nearest first decimal)
Advisor 1	58000	90000	-63000	0.8
Advisor 2	35000	90000	-63000	0.7
Advisor 3	-16000	90000	-63000	0.4
Advisor 4	-63000	90000	-63000	0

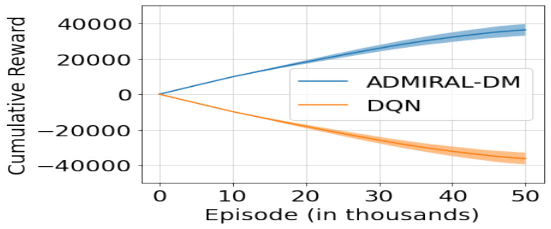
Table 3.3: Finding ϵ'_0 using ADMIRAL-AE for the Pommeran Domain OneVsOne against DQN.

3.4.3 Experimental Results - Function Approximation - ADMIRAL-DM

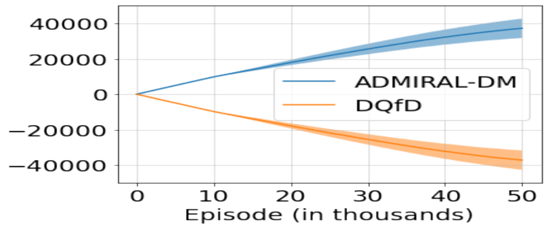
We now show that it is possible to extend our tabular ADMIRAL-DM method to function approximation based implementations that make our algorithms more generally applicable to environments with large state-action spaces. We will use the neural network-based version of ADMIRAL-DM as discussed in Section 3.2.4. Additionally, we show that ADMIRAL-DM is capable of outperforming several strong baselines from literature.

We perform comparative experiments in three domains. All our experiments in this section are repeated 30 times, and we plot the mean and standard deviation. The important elements of our experimental domains are mentioned here, while the complete details of the domains and implementation details of all algorithms are in Appendix A.4. Neural network implementations of decision-making algorithms (ADMIRAL-DM, ADMIRAL-DM(AC)) are used in this sub-section. The first domain we consider is Domain OneVsOne of Pommerman introduced in Section 3.4.2. Our baselines are DQfD, CHAT, and DQN. We perform 50,000 episodes of training, where the algorithms train against specific opponents. Each episode is a full Pommerman game (lasting a maximum of 800 steps). All the algorithms relying on demonstrations (DQfD, CHAT, ADMIRAL-DM, and ADMIRAL-DM(AC)) use the Advisor 1 considered in Section 3.4.2. The probability of using the advisor action (ϵ'_t in Algorithm 1) starts from 0.8 (obtained from Table 3.3) and linearly decays to be close to zero at the end of training for both ADMIRAL-DM and ADMIRAL-DM(AC). To provide data for offline pretraining in the case of DQfD, two instances of Advisor 1 is used to play many Pommerman games that generate the required data. The DQfD is pretrained with all of this data, before entering the training phase of our experiments.

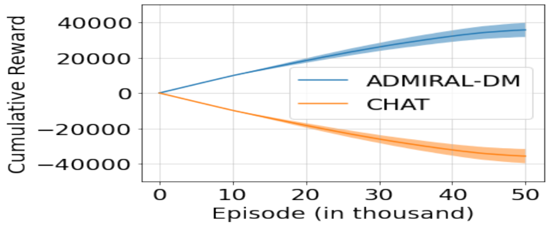
After the training phase, the trained algorithms enter a face-off competition of 10,000 games where there is no more training, no further exploration and additionally ADMIRAL-DM and ADMIRAL-DM(AC) play without any advisor influence. ADMIRAL-DM(AC) is a CTDE technique, which only performs decentralized execution in face-off using the trained actor-network. We plot the cumulative rewards in the training phase (Figure 3.7 (a), (b), (c), (d)), from which it can be seen that ADMIRAL-DM's performance is better than the baselines (DQN, DQfD, and CHAT). The face-off plots in Figure 3.7(e) show that ADMIRAL-DM wins more games on average against all the other baselines, showing its dominance. DQfD relies on pretraining, which is harder in MARL, as the nature of opponents that an agent will face during competition is impossible to determine upfront. The algorithms that use online advisors to give real-time feedback (that captures the changing nature of the opponent) tend to do better. DQfD has also been previously reported to have over-fitting issues [79], which is likely to hurt its performance more in multi-agent



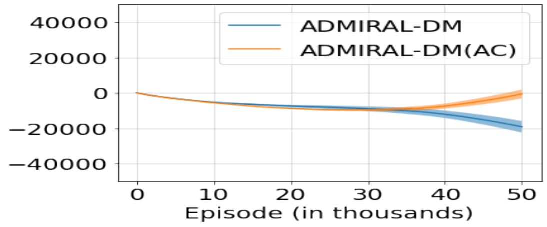
(a) ADMIRAL-DM vs DQN



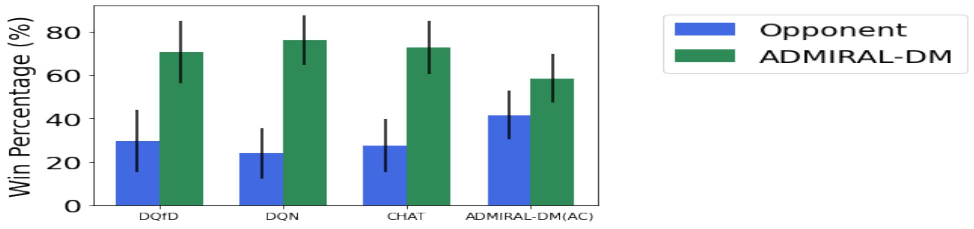
(b) ADMIRAL-DM vs DQfD



(c) ADMIRAL-DM vs CHAT



(d) ADMIRAL-DM vs ADMIRAL-DM(AC)



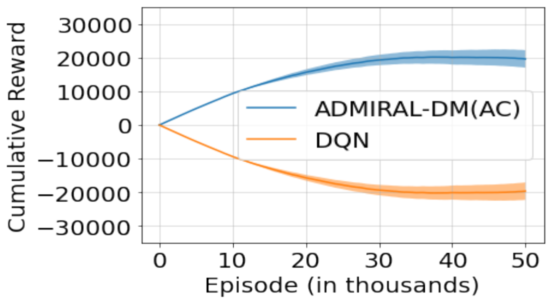
(e) Faceoff against ADMIRAL-DM

Figure 3.7: Pommerman competition against ADMIRAL-DM. The ADMIRAL-DM beats all baselines in the execution phase. In the training phase ADMIRAL-DM(AC) performs better than ADMIRAL-DM in a head-to-head challenge.

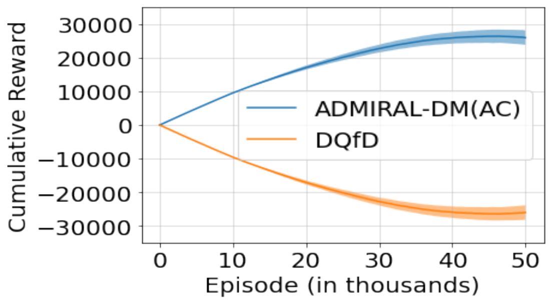
environments as compared to single-agent environments. In multi-agent environments, it is more important to be able to generalize to unseen dynamic opponent behaviour, which is different from that seen in pre-collected demonstration data. As discussed previously, CHAT maintains a confidence measure on the advisor, which depends on the advisor's consistency in action recommendations at different states. In MARL, this measure is not completely reliable, since even good advisors may need to formulate stochastic action recommendations as responses to the opponent. DQN, on the other hand, learns directly from interaction experiences and cannot learn from advisor inputs. This is a disadvantage in environments where external sources of knowledge, such as advisors, are available to be leveraged. Furthermore, since our baselines are independent algorithms (that consider opponents to be part of the state), they lose out to ADMIRAL-DM, which explicitly tracks opponent action. ADMIRAL-DM loses to ADMIRAL-DM(AC) during training (Figure 3.7 (d)). Though ADMIRAL-DM(AC) shows slower learning overall (as it is training both actor and critic), it ultimately learns a higher performing policy. One important reason is that the actor-critic method trains a stochastic policy that can explore naturally, whereas the Q-learning method needs a hyperparameter to conduct a forced exploration (ϵ-greedy). Another reason could be that ADMIRAL-DM(AC) learns from each recent experience, while the ADMIRAL-DM has delayed learning using the replay buffer. However, in the face-off, ADMIRAL-DM has an edge over ADMIRAL-DM(AC) (Figure 3.7(e)), probably due to being centralized. Since the performance of ADMIRAL-DM and ADMIRAL-DM (AC) in the face-off results given in Figure 3.7(e) are close, we perform a Fischer's exact test for the average performances to check statistical significance. We get $\mathrm{p} < 0.03$ which shows that this result is statistically significant (we treat $\mathrm{p} < 0.05$ as statistically significant as in common practice).

Next, we conduct similar experiments with ADMIRAL-DM(AC) that show it outperforms the baselines. The ADMIRAL-DM(AC) is explicitly compared to all the baselines in a training and face-off scheme similar to that done with ADMIRAL-DM. To recall, we perform training experiments of 50,000 full Pommerman games and face-off contests of 10,000 games, where the trained agents compete against each other without any further training or advisor influence. The results are plotted in Figure 3.8, where the ADMIRAL-DM(AC) shows better performance than the baselines (in both training and face-off phases). In training, ADMIRAL-DM(AC) dominates all the other baselines by winning around 20,000 – 30,000 games out of the 50,000 games conducted. As observed in the previous experiments, the ADMIRAL-DM(AC) algorithm's learning is slower than that of the Q-learning variant, making the overall number of games won (captured by cumulative rewards), against the baselines to be lower than that of the corresponding training of ADMIRAL-DM against the baselines in Figure 3.7. In the face-off contests, the ADMIRAL-DM(AC) algorithm wins more than 50% of the games against all baselines except ADMIRAL-DM. As noted previously, the ADMIRAL-DM algorithm has a slight edge in performance over that of ADMIRAL-DM(AC) in the face-off stage.

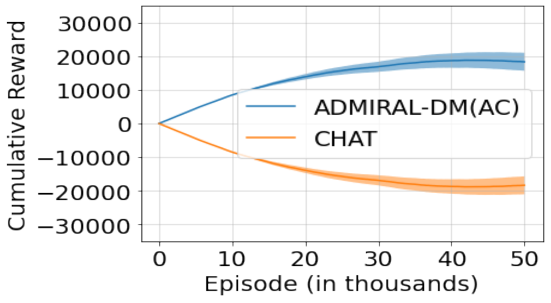
Next, we use two cooperative domains from the Stanford Intelligent Systems Laboratory (SISL) [89]. These experiments have two phases — training and execution. The algorithms train for 1000 games in the training phase and then enter an execution phase, where they execute the trained policy for 100 games. We choose to set the value of advisor influence ϵ'_t to 0.8 at the start of training and linearly decay it the same way as in the above experiments with Pommerman (since we are using good advisors). In the execution phase, there is no



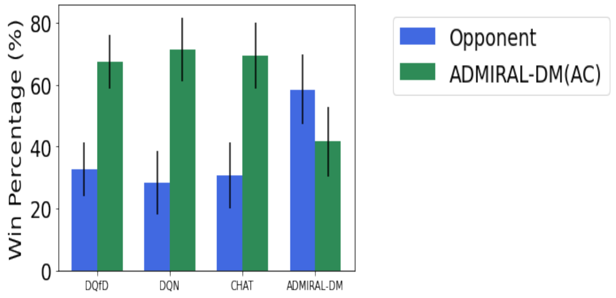
(a) ADMIRAL-DM(AC) vs DQN



(b) ADMIRAL-DM(AC) vs DQfD



(c) ADMIRAL-DM(AC) vs CHAT

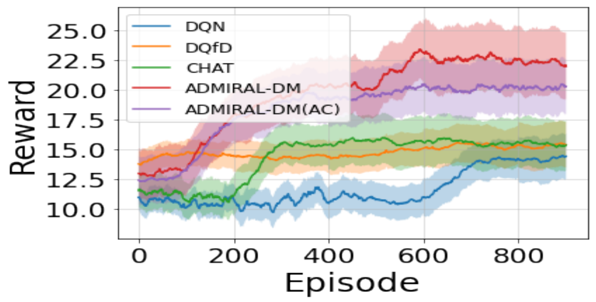


(d) Faceoff against ADMIRAL-DM(AC)

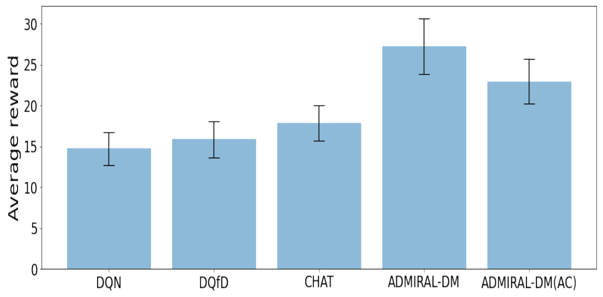
Figure 3.8: Pommerman competition against ADMIRAL-DM(AC). ADMIRAL-DM(AC) defeats all other baselines in both the training and execution phases.

further exploratory actions for all algorithms, and no more influence of the advisor for ADMIRAL-DM and ADMIRAL-DM(AC).

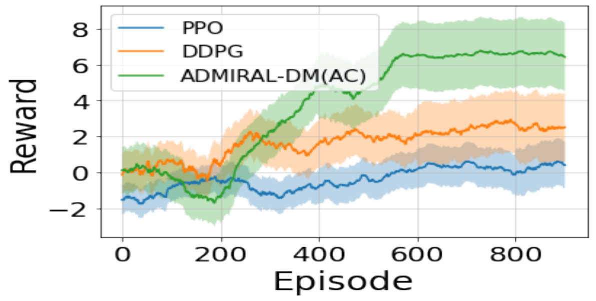
The first SISL environment is a Pursuit environment, which contains 8 pursuers controlled by learning algorithms, trying to capture 30 evaders moving randomly in the grid-based environment. Rewards have a local structure, where the pursuers participating in the capture of an evader or the pursuers encountering evaders are rewarded individually. The game is general-sum and does not have a global reward structure. The local reward structure helps to tackle the issue of credit assignment. We use a pre-trained policy of DQN (trained for 1000 episodes) as the advisor. We plot the reward obtained (averaged per agent) for the training and execution phases in Figures 3.9(a) and (b). The execution performance bars in Figure 3.9(b) is the average performance across the 100 execution games. The results show that ADMIRAL-DM has better performance than all other baselines, including DQN used as the advisor, in both phases. Thus, our algorithm can ultimately outperform the advisor. This environment is highly non-stationary (due to having more number of learning



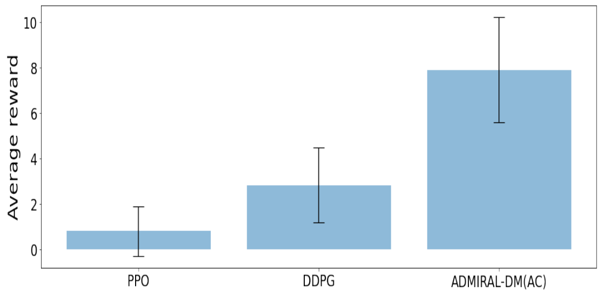
(a) Pursuit-Training



(b) Pursuit-Execution



(c) Waterworld - Training



(d) Waterworld - Execution

Figure 3.9: SISL Environments - Training and Execution. The ADMIRAL algorithms give a better performance than all the baselines in both the phases. Training graphs have been smoothed with a running average of 100.

agents), so completely centralized ADMIRAL-DM has an edge over ADMIRAL-DM(AC) which uses decentralized actors.

Since some performances in Figure 3.9(b) are close, we perform an unpaired 2-sided t test for statistical significance. Regarding the performances of CHAT and ADMIRAL-DM(AC) we get a value of $p < 0.02$ and similarly for the performances of ADMIRAL-DM and ADMIRAL-DM(AC) we get a $p < 0.02$, which shows that both these comparisons are statistically significant.[4]

Our second SISL environment is the continuous action space Waterworld environment, which has 5 pursuer agents trying to consume food and avoid poison. The actions are continuous-valued thrust, that the agents can apply to move in a particular direction, and with a desired speed. Here, multiple pursuer agents need to work together to consume food.

[4]We consider $p < 0.05$ as statistically significant.

Agents get rewards based on foods captured and punishments based on poison consumed. Rewards have a local structure similar to the Pursuit environment. The advisor here is a pre-trained proximal policy optimization (PPO) [216] agent (trained for 1000 episodes). Similar to the Pursuit environment, we plot the performances for both training and execution phases (averaged per agent) in the Waterworld environment (see Figures 3.9(c) and (d)). The ADMIRAL-DM(AC) alone is used for these experiments, as the Q-learning variant is not applicable for continuous action spaces. We use two popular RL algorithms for continuous control, PPO and deep deterministic policy gradients (DDPG) [147] as baselines. The results show that ADMIRAL-DM(AC) has better performance than others in both phases (Figure 3.9(c) and (d)). The ADMIRAL-DM(AC) algorithm has two important advantages over the other baselines here. The first advantage is that it is capable of leveraging an advisor. The second advantage is that ADMIRAL-DM(AC) is trained in a centralized fashion by tracking the opponent behaviour while the other algorithms are independent methods. Still, ADMIRAL-DM(AC) is decentralized in execution. Notably, the ADMIRAL-DM(AC) algorithm also improves upon PPO, used as the advisor, similar to our observation in the Pursuit environment.

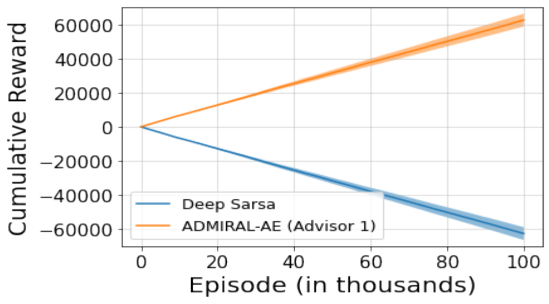
In both the above SISL experiments, we see a small improvement in performance in the execution phases for both algorithms, ADMIRAL-DM and ADMIRAL-DM(AC), as compared to the final training performances. This is due to the fact that, at the end of the training, there is still a small amount of exploration and advisor influence (1 % of actions) involved, whereas during execution both these influences are completely removed, which contributes to a net improvement in performance.

To summarize, our experimental results show that the ADMIRAL-DM and ADMIRAL-DM(AC) algorithms make the best use of advisors in multi-agent settings compared to the other state-of-the-art algorithms. After the advisor influence completely stops, the performance of ADMIRAL-DM and ADMIRAL-DM(AC) is better than the others. We have also demonstrated that our methods can be extended to continuous action spaces and work in decentralized environments using the popular CTDE technique.

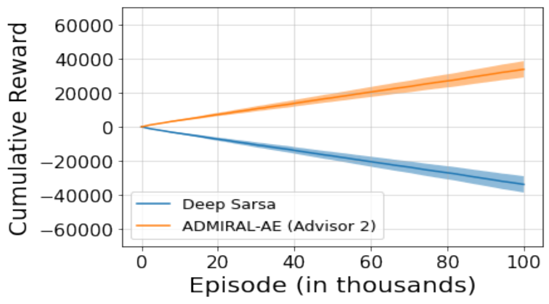
3.4.4 Performance Of ADMIRAL-DM Under The Influence Of Different Advisors

Next we study the impact of using the ADMIRAL-AE in a 'pre-learning' phase to determine the value of ϵ'_0. Towards the same, we would like to use an algorithm to serve as a common opponent. We choose to use a different algorithm compared to the baselines considered in the previous sub-section (where the objective was to show better performances of ADMIRAL-DM as compared to these baselines). The algorithm we choose to use as the opponent is Deep Sarsa, which is similar to DQN but uses a “Sarsa-like” [256] Bellman update for the Q-values. We clarify that our objective in this section is not to show better performances against any baseline (which we have already done in Section 3.4.3).

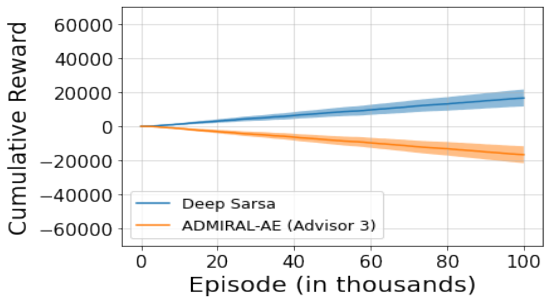
Further, in this section, we provide additional experiments that evaluate ADMIRAL-DM on different advisors with the common opponent (Deep Sarsa) and show that ADMIRAL-DM is capable of recovering from bad action-advice. All results reported in this section use averages and standard deviations of 30 runs.



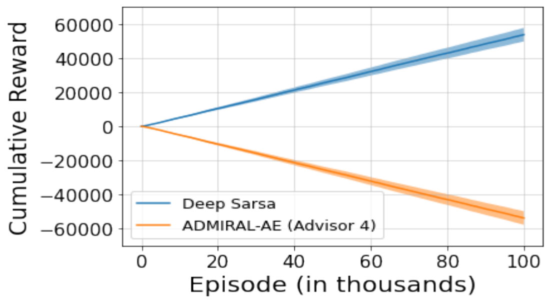
(a) Advisor 1



(b) Advisor 2



(c) Advisor 3



(d) Advisor 4

Figure 3.10: Results in Domain OneVsOne of Pommerman using different advisors with ADMIRAL-AE and Deep Sarsa. The best advisor (Advisor 1) gives the highest overall performance while the worst advisor (Advisor 4) gives the lowest performance.

We describe two sets of experiments with two different domains of Pommerman. In the first set of experiments, we use the neural network implementation of ADMIRAL-DM and ADMIRAL-AE on the Domain OneVsOne of Pommerman using the four different advisors introduced in Section 3.4.2. To recall, Advisor 1 is the best advisor who can give the best action (relative to other advisors) at all states and Advisor 4 is the worst. The quality of advisors reduces from Advisor 1 to Advisor 4. As mentioned, we use a common opponent as an agent playing Deep Sarsa.

First, we wish to evaluate the given advisors against the performance of Deep Sarsa in the 'pre-learning' phase. To do this, we run a series of training experiments, where we implement ADMIRAL-AE using each of the four advisors against Deep Sarsa. The results are plotted in Figure 3.10. As observed earlier, the best advisor leads to the best overall performance and the worst advisor leads to the worst performance. Using these performances the ϵ'_0 values are determined in Table 3.4 (using Eq. 3.3). It can be seen that the suggested value of ϵ'_0 is highest (0.9) for the best advisor and the smallest (0) for the worst advisor. These values of ϵ'_0 show that the agent will listen more to the good advisors and listen less (or not at all) to the bad ones.

Advisor	Average cumulative reward (rounded to nearest 1000)	Maximum possible cumulative reward (adjusted for random exploration)	Average performance of agent using a random advisor (rounded to nearest 1000)	Normalized value (rounded up to nearest first decimal)
Advisor 1	63000	90,000	-54000	0.9
Advisor 2	34000	90,000	-54000	0.7
Advisor 3	-16400	90,000	-54000	0.3
Advisor 4	-54000	90,000	-54000	0

Table 3.4: Finding an initial value for ϵ'_0 using ADMIRAL-AE for the Pommeran Domain OneVsOne against Deep Sarsa.

Next, we run the ADMIRAL-DM algorithm against Deep Sarsa, using each of the four advisors. We use four initial values of ϵ'_0 for each of the advisors, where one of these values corresponds to the choice of ϵ'_0 as obtained from our previous experiment with ADMIRAL-AE reflected in Table 3.4. In addition to these four values, we also consider a value of 0 for ϵ'_0, which considers the performance of ADMIRAL-DM with no advisor inputs to serve as a baseline. As done previously, the value of ϵ' is decayed linearly, during training, for ADMIRAL-DM in all the experiments. All training is conducted for 100,000 episodes with the advisor influence (ϵ') being linearly decayed to 0 at 50,000 episodes, *i.e.* there is no advisor influence after 50,000 episodes. Each episode is a complete Pommerman game involving a maximum of 800 steps as in the previous experiments. More details of the game conditions and the advisors can be found in Appendix A.4. The results showing the performance of ADMIRAL-DM in each of these experiments are presented in Figure 3.11.

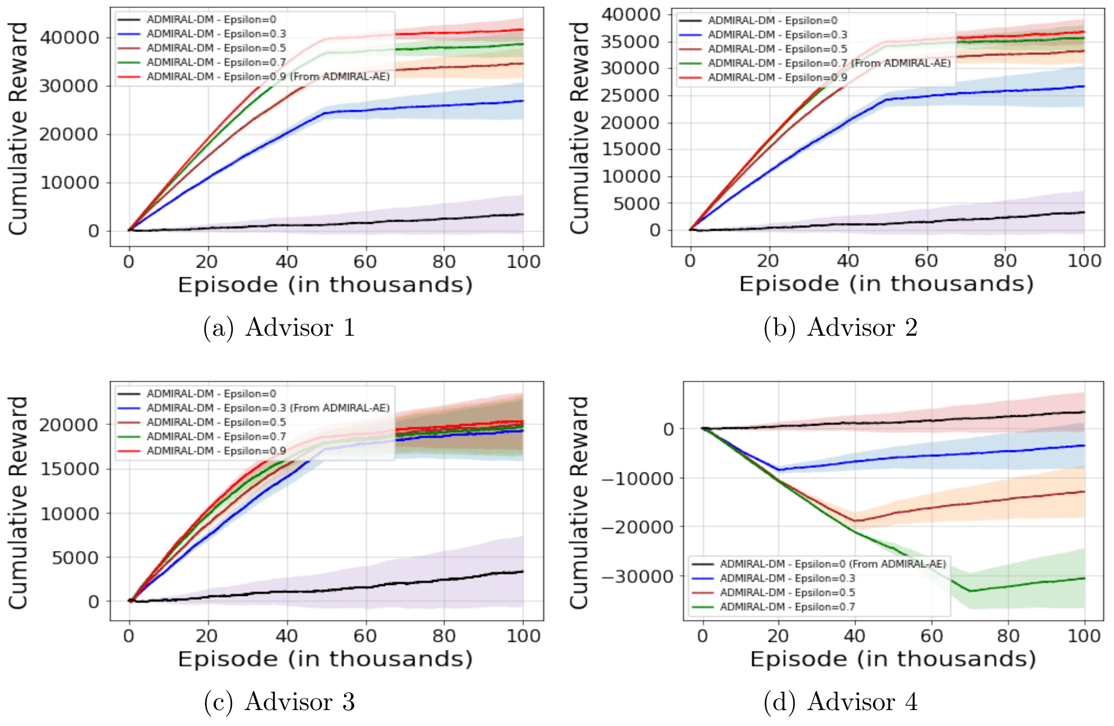
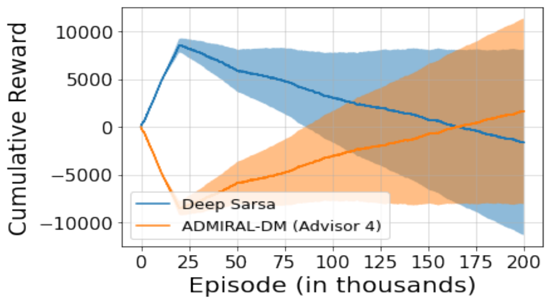
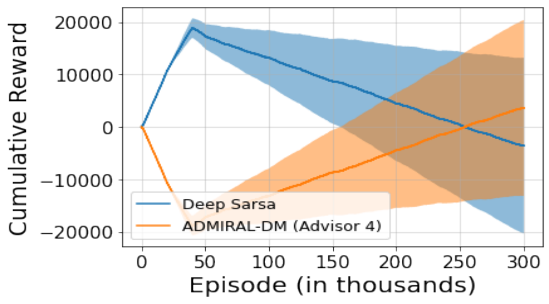


Figure 3.11: Results in Domain OneVsOne of Pommerman using different advisors with ADMIRAL-DM and Deep Sarsa. The result plots show the performance of ADMIRAL-DM in the training against Deep Sarsa. The results show that ϵ'_0 value obtained from the performance of ADMIRAL-AE in Table 3.4 show either the best performance or is very close to the best possible performances amongst all the ϵ'_0 values.

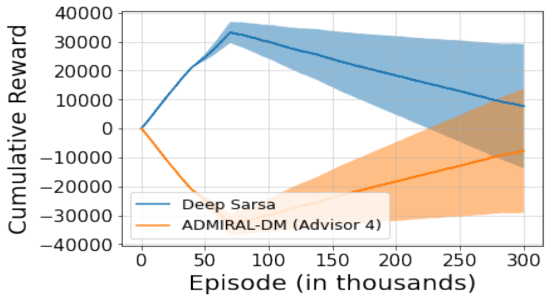
We highlight several observations. Figure 3.11(a) and (b) show that ADMIRAL-DM, using the good advisors (Advisor 1 and Advisor 2), achieves the maximum overall performance since the positive influence from the good advisors helps. However, if the advisor influence is limited ($\epsilon' = 0.3$), the good advisors only have a limited impact. As the value of ϵ'_0 increases, we see that the performance of ADMIRAL-DM using the first two advisors improves (Figures 3.11(a) and (b)), as expected. Since Advisor 1 is even better than Advisor 2, we see from Figures 3.11(a) and (b) that ADMIRAL-DM using Advisor 1 clearly shows superior performance to that of Advisor 2 for the highest value of ϵ'_0, 0.9.



(a) OneVsOne - $\epsilon' = 0.3$



(b) OneVsOne - $\epsilon' = 0.5$



(c) OneVsOne - $\epsilon' = 0.7$

Figure 3.12: Results of ADMIRAL-DM vs Deep Sarsa using the Advisor 4, which is the worst advisor among the ones considered. The plots correspond to the OneVsOne domain. All the figures show that, ADMIRAL-DM is capable of recovering from bad action advice. Greater influence of the bad advisor (larger ϵ'_0) leads to a larger time needed for recovering from the bad action influence.

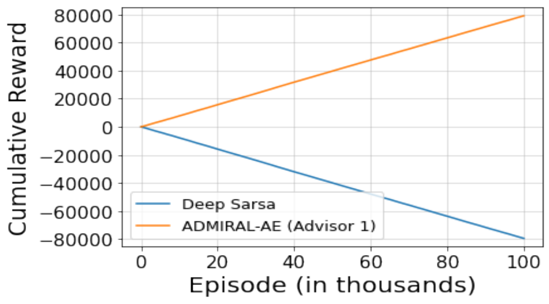
When ϵ'_0 values were lower (such as 0.5), performance using both Advisor 1 and Advisor 2 are comparable since these advisors did not have many opportunities to make an impact. Notably, ADMIRAL-DM using Advisor 1 shows the best performance for $\epsilon' = 0.9$, the value obtained from ADMIRAL-AE as seen in Table 3.4. ADMIRAL-DM using Advisor 2 almost provides the same performances for the highest values of ϵ'_0, 0.7 and 0.9. Additional inputs from this advisor are not as useful (compared to Advisor 1), since it is weaker. Hence, a value of $\epsilon' = 0.7$ as obtained from ADMIRAL-AE is sufficient for this advisor.

Turning our attention to the performance of ADMIRAL-DM with the third advisor, we find that it is significantly inferior compared to the other two advisors, yet still has a limited positive influence (Figure 3.11(c)). Furthermore, the performance using this advisor is considerably better than using no advisor at all ($\epsilon' = 0$). However, while using Advisor 3, we notice that, as the values of ϵ_0' increases, there is no appreciable improvement in performance. This shows that more influence of a comparatively less effective advisor does not lead to much improvement in performance. Again, the value suggested by ADMIRAL-AE (0.3), comes very close to the best possible performance with other values of ϵ_0'.

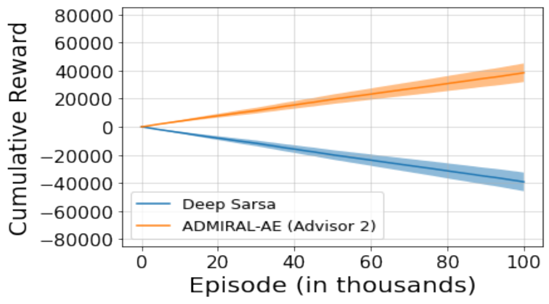
The performance of ADMIRAL-DM using the last advisor (Advisor 4) is interesting. This advisor has a negative influence on learning and makes ADMIRAL-DM lose for the first few episodes (Figure 3.11(d)). However, ADMIRAL-DM recovers after the advisor influence wanes in all cases. As the value of ϵ_0' increases, we see that further influence from the bad advisor is detrimental and a larger number of episodes is required before ADMIRAL-DM shows signs of recovery from the poor advice. Hence, the best value of ϵ_0', in this case, is 0, since listening to this advisor only harms learning. Again, this was the value obtained for Advisor 4, in Table 3.4. We present a more elaborate set of results on the experiments with Advisor 4 in Figure 3.12. Here we show that for all cases of ϵ_0', ADMIRAL-DM is capable of recovering from bad action-advising and after a suitable number of episodes, can overtake the performance of Deep Sarsa. However, higher values of ϵ_0' makes the learning from the bad advisor problematic, since while ADMIRAL-DM shows signs of recovery it still cannot overtake the cumulative performance of Deep Sarsa even after 300,000 episodes (Fig: 3.12(c)).

We now provide a brief description of another domain of Pommerman. Domain TwoVsTwo is a larger version of Pommerman, where there are a total of four agents, with two of the four belonging to the same team. The state space is much larger than Domain OneVsOne, with each state containing 372 elements. The reward function remains sparse, with the two agents belonging to the winning team getting +1 and the two agents belonging to the losing team getting -1 at the end of the game. In case of a draw, all the agents get -1. In Domain TwoVsTwo, we consider one team of Deep Sarsa and one team of the ADMIRAL-DM. This makes this domain harder than the Domain OneVsOne, as the agents must learn to cooperate amongst the members of the same team and compete against the members of the opponent team to win the game. The Domain TwoVsTwo is a mixed competitive-cooperative domain, which is different from Domain OneVsOne that had only pure competition. We use the same four advisors as considered before for the Domain TwoVsTwo.

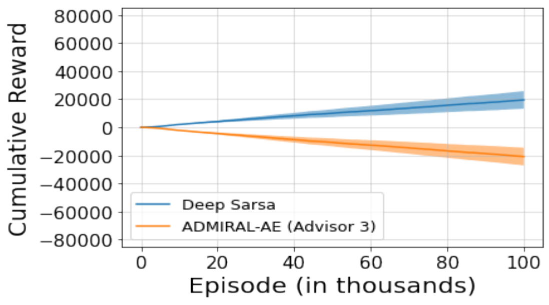
Similar to the experiments with Domain OneVsOne, we first evaluate the advisors against Deep Sarsa using the ADMIRAL-AE algorithm. The results are presented in Figure 3.13. Again, the best advisor gives the maximum overall performance and the worst advisor results in the minimum performance. The ϵ_0' values are determined based on these results in Table 3.5 (using Eq. 3.3). These values are used in further experiments using ADMIRAL-DM and Deep Sarsa in the Domain TwoVsTwo of Pommerman.



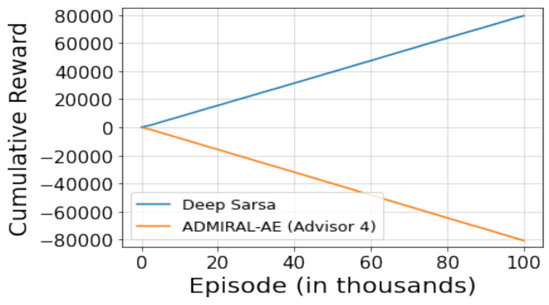
(a) Advisor 1



(b) Advisor 2



(c) Advisor 3



(d) Advisor 4

Figure 3.13: Results in Domain TwoVsTwo of Pommerman using different advisors with ADMIRAL-AE and Deep Sarsa. The best advisor (Advisor 1) gives the highest overall performance while the worst advisor (Advisor 4) gives the lowest performance. The standard deviation of (a) and (d) are negligible.

Our next set of experiments train in the Domain TwoVsTwo with each agent in one team of Pommerman agents playing ADMIRAL-DM and each agent in the other team playing Deep Sarsa. We consider four different initial values of ϵ' (where one of these values will correspond to the value obtained in Table 3.5), as we did in the OneVsOne domain. In addition to these ϵ'_0 values, we continue to use the value of $\epsilon'_0 = 0$ as the baseline. This corresponds to the situation of ADMIRAL-DM learning with no advisor influence. All training is run for 100,000 episodes, with the value of ϵ' decaying linearly to 0 at 50,000 episodes. Thus, there is no more influence from advisors for the last 50,000 episodes of training. The results showing the performance of the team playing ADMIRAL-DM (in the competition against a team playing Deep Sarsa) are in Figure 3.14. The ADMIRAL-DM performances show similar characteristics to that seen in Domain OneVsOne. ADMIRAL-DM, using the best advisor (Advisor 1), shows the best performance for all values of ϵ'_0

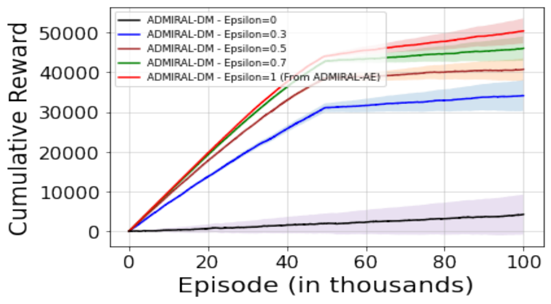
Advisor	Average cumulative reward (rounded to nearest 1000)	Maximum possible cumulative reward (adjusted for random exploration)	Average performance of agent using a random advisor (rounded to nearest 1000)	Normalized value (rounded up to nearest first decimal)
Advisor 1	79000	90000	-81000	1
Advisor 2	39000	90000	-81000	0.7
Advisor 3	-21000	90000	-81000	0.4
Advisor 4	-81000	90000	-81000	0

Table 3.5: Finding ϵ_0' using ADMIRAL-AE for the Pommeran Domain TwoVsTwo against Deep Sarsa.

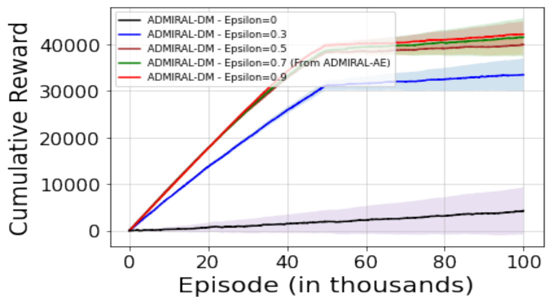
and its performance keeps improving with the increase in value of ϵ_0' (refer Figure 3.14(a)). Notably, comparing Figure 3.14(a) and Figure 3.11(a), the performance of ADMIRAL-DM using the best advisor is better in the case of Domain TwoVsTwo as compared to the Domain OneVsOne. This suggests that a good advisor has comparatively higher impact when tasks get harder. This is because there are far more strategies that need to be learned to do well in this domain and the opponent learning from scratch needs more time for learning the hard task. A good advisor, on the other hand, can teach the different strategies needed much faster and provide an early lead for ADMIRAL-DM. Similar observations apply to the performances of ADMIRAL-DM using Advisor 2 as well (refer Figure 3.14(b) and Figure 3.11(b)).

For Advisor 3, the performance does not change much with varying values of ϵ_0' (refer Figure 3.14(c)) as observed in the case of Domain OneVsOne. Comparing Figure 3.14(c) and Figure 3.11(c), we note that the best performance of ADMIRAL-DM using Advisor 3 is better for the Domain TwoVsTwo as compared to Domain OneVsOne, reinforcing our inferences earlier about a higher potential for impact in useful advisors when the tasks get harder. Regarding Advisor 4, the greater negative influence from this advisor necessitates a longer time for recovery (refer Figure 3.14(d)).

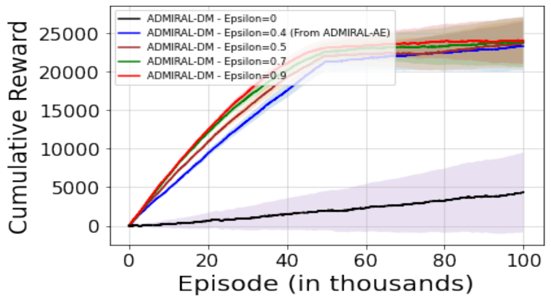
Again, from Figure 3.14, for all the four advisors, we observe that the value for ϵ_0' as obtained from Table 3.5 gives the best possible performance or comes quite close to the best possible performance, compared to other possible values of ϵ_0'. This shows the advantage of



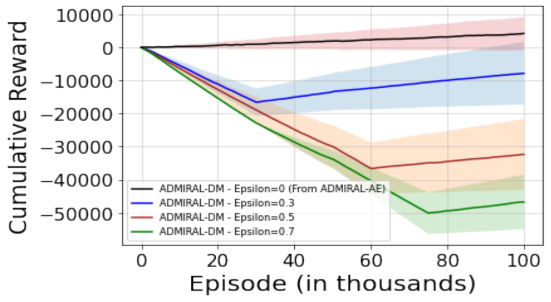
(a) Advisor 1



(b) Advisor 2



(c) Advisor 3

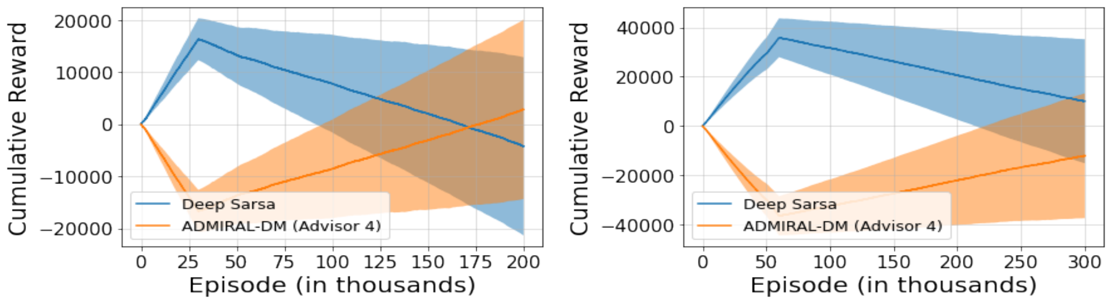


(d) Advisor 4

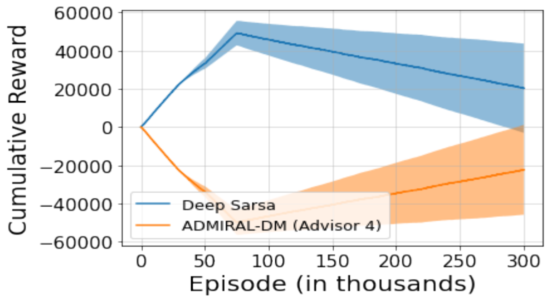
Figure 3.14: Results in Domain TwoVsTwo of Pommerman using different advisors with ADMIRAL-DM and Deep Sarsa. The result plots show the performance of team playing ADMIRAL-DM in training competitions against Deep Sarsa. For this domain too, ϵ'_0 value obtained from the performance of ADMIRAL-AE in Table 3.5 show either the best performance or is very close to the best possible performances amongst all the ϵ'_0 values.

evaluation using ADMIRAL-AE. In Figures 3.15(a), (b) and (c), ADMIRAL-DM shows signs of recovery for all values of ϵ'_0 while using Advisor 4 for learning. However, for Advisor 4, the smaller the value of ϵ'_0 the better. When making a comparison between Figure 3.12 and Figure3.15, we note that, as the negative influence from the advisor increases (through a larger ϵ'_0) the time needed for recovery also rises in the case of Domain TwoVsTwo and is greater than that needed in the Domain OneVsOne. As the complexity of the tasks increase, the agents need a lot more time to learn good policies that recovers the loss of performance from bad action-advice. This shows that negative influence from an advisor is more costly in the case of harder MARL tasks/environments as against comparatively simpler environments.

From our experiments, we conclude that ADMIRAL-DM is capable of recovering from



(a) TwoVsTwo - $\epsilon' = 0.3$ (b) TwoVsTwo - $\epsilon' = 0.5$



(c) TwoVsTwo - $\epsilon' = 0.7$

Figure 3.15: Results of ADMIRAL-DM vs. Deep Sarsa using the Advisor 4 in the TwoVsTwo domain. Similar to the first domain, all the figures show that ADMIRAL-DM is capable of eventually recovering from bad action advice.

bad advisor recommendations and is able to suitably leverage advisors who have some positive influence on learning. However, if possible, it is best to evaluate an advisor using the ADMIRAL-AE method and obtain a suitable initial value for the hyperparameter that determines the advisor influence (ϵ'_0). This helps in learning good policies faster.

3.4.5 Summary

To summarize, in Section 3.4.1 we showed an experimental illustration of our algorithms in a tabular domain. We showed that, while using ADMIRAL-AE, the best advisor gives the best overall performance. Further, ADMIRAL-AE provides a suitable value for the hyperparameter ϵ'_0, which when used by ADMIRAL-DM subsequently, provides good performances with different types of advisors. We also provided an experimental illustration of our theoretical convergence results in the case of both ADMIRAL-DM and

ADMIRAL-AE. In Section 3.4.2 we provided an illustration of ADMIRAL-AE in a large environment with neural networks as function approximators. Again, we illustrated that using ADMIRAL-AE with the best advisor provides the best performance amongst other (comparatively worse) advisors. Obtaining an appropriate value for ϵ'_0 from Section 3.4.2 we showed that ADMIRAL-DM and ADMIRAL-DM(AC) provide better performances than a set of baselines in Section 3.4.3. We tested our algorithms in both competitive and cooperative domains, as well as settings with discrete and continuous action spaces.

In Section 3.4.4, we used two Pommerman domains to illustrate that using ADMIRAL-AE to obtain a suitable value for ϵ'_0, provides the best performance for ADMIRAL-DM. Hence, when possible, it would be best to use ADMIRAL-AE for determining ϵ'_0 using a pre-learning phase. Also, we illustrated that ADMIRAL-DM is capable of recovering from bad action advice from advisors if appropriate values for ϵ'_0 cannot be determined before training ADMIRAL-DM.

Additionally, in Appendix A.3 we show another important advantage of using the principled method of ADMIRAL-AE for advisor evaluation in environments having dynamically learning and adapting advisors. We show that a principled method like ADMIRAL-AE would find a suitable value for ϵ'_0 when it is presented with a learning advisor, where other methods based on simple heuristics may have a high chance of failure.

3.5 Conclusion

In this chapter, we introduced the problem of learning under the influence of external advisors in MARL. We provided a principled framework for MARL algorithms learning to use advisors. Using Q-learning based methods, we proposed two MARL algorithms for this problem. We conducted theoretical analyses of these algorithms, establishing conditions under which fixed point guarantees can be provided regarding their learning in general-sum stochastic games. We proved that previous theoretical results can extend to this setting under a comparatively weaker set of assumptions than previously considered. Empirically, we showed that our algorithms can be scaled to domains with large state-action spaces using traditional function approximators like neural networks. We also introduced an additional actor-critic variant of our ADMIRAL-DM algorithm that can operate under the CTDE paradigm and can learn in environments with continuous action spaces. Our empirical results further established the superiority of our algorithms compared to standard baselines. Furthermore, we have shown that our methods would be useful in a wide variety of problems and that the algorithms can recover from the influence of weak/bad advisors during learning.

While there is a rich body of literature on the use of external knowledge sources in single-agent RL [24], MARL provides additional challenges which mean that not all the results and approaches can transfer over directly. We discussed the important problems of directly using the single-agent based methods that learn from external sources in MARL. Additionally, we performed direct comparison experiments to elucidate a few of these problems. Our approach to learning from advisors in MARL may look more complex compared to other single-agent approaches, however, the non-stationarity of the environment makes learning under the influence of advisors in MARL considerably more challenging. In MARL, quick adaptation to the changing environment is the key to better performance [151]. Our approach of using an online advisor is a more appropriate formulation of advisors in MARL, as real-time feedback against non-stationary opponents are critical for learning effective multi-agent policies, as demonstrated in our experiments.

Importantly, we consider a general setting, where we had no restrictions on the type or quality of the advisor, and no restrictions on the relation between the reward functions of different agents (general-sum). Particularly in MARL, the assumption of optimal advisors could be overly strong, since performance depends on the nature of opponents. The advisor could be capable of providing good feedback in strategizing against a particular class of opponents yet be useless against another class of opponents. Furthermore, a sub-optimal advisor could be good only in a very narrow portion of the state space, which is still useful for an agent learning from scratch in this environment. By explicitly allowing nonrestrictive **sub-optimal** advisors, our work is more widely applicable than previous methods that make an assumption of optimal (or near-optimal) experts to help RL training [82, 209, 232].

Chapter 4

Learning from Multiple Independent Advisors in Multi-agent Reinforcement Learning

As discussed in the previous chapters, MARL typically suffers from the problem of sample inefficiency, where learning suitable policies involves the use of many data samples. Learning from external demonstrators is a possible solution that mitigates this problem. However, most prior approaches (including our previous chapter) in this area assume the presence of a single demonstrator. Leveraging multiple knowledge sources (i.e., *advisors*) with expertise in distinct aspects of the environment could substantially speed up learning in complex environments. This chapter considers the problem of simultaneously learning from multiple independent advisors in multi-agent reinforcement learning. The approach leverages a two-level Q-learning architecture, and extends this framework from single-agent to multi-agent settings. We provide principled algorithms that incorporate a set of advisors by both evaluating the advisors at each state and subsequently using the advisors to guide action selection. We provide theoretical convergence and sample complexity guarantees. Experimentally, we validate our approach in three different test-beds and show that our algorithms give better performances than baselines, can effectively integrate the combined expertise of different advisors, and learn to ignore bad advice.

Our contributions in this chapter are summarized as follows:

- A principled framework for learning from multiple independent sub-optimal advisors in MARL. Two practical algorithms (one using Q-learning and the other using actor-critic) using this framework.

- Theoretical fixed point guarantee for the tabular Q-learning method in general-sum stochastic game environments.
- Extensive theoretical finite-time analysis of the tabular Q-learning method under polynomial learning rates and linear learning rates. Results show at-most linear dependence on the number of agents, which is superior to prior works that provide exponential dependence on the number of agents [154, 234].
- Extensive experimental results in three different multi-agent testbeds (competitive, cooperative, and mixed settings), that show superior performances for the provided algorithms as compared to a suite of related baselines.

4.1 Introduction

As discussed in the previous chapter, using external sources of knowledge that help in accelerating MARL training is one solution to improving MARL sample efficiency [17], which has extensive support in literature [225]. However, most prior work include two limiting assumptions. First, all demonstrations need to come from a single demonstrator [34]. However, in complex MARL environments, since agents learn policies that meet the twin goals of responding to changing opponent(s) and environments [150], a learner can likely benefit from multiple knowledge sources that have expertise in different parts of the environment or different aspects of the task. Second, all demonstrations are near-optimal (i.e., from an “expert”) [191]. In practice, these knowledge sources are typically sub-optimal, and we broadly refer to them as *advisors* (to differentiate them from experts).

In this chapter, we provide an approach that simultaneously leverages multiple different (sub-optimal) advisors for MARL training. Since the advisors may provide conflicting advice in different states, an algorithm needs to resolve such conflicts to take advantage of all the advisors effectively. We propose a two-level learning architecture and formulate a Q-learning algorithm for simultaneously incorporating multiple advisors in MARL, improving upon the work of Li et al. [145] which introduces the idea of two-level learning in single-agent RL. This architecture uses one level to evaluate advisors and the other learns values for actions. Further, we extend our approach to an actor-critic variant that applies to the centralized training and decentralized execution (CTDE) setting [156]. Since RL is a fixed point iterative method [262], we provide convergence results, proving that our Q-learning algorithm converges to a Nash equilibrium [171]. Additionally, we provide a finite-time analysis of our algorithm, and prove that it has (at most) a linear dependence on the number of agents in the worst case when using a linear or polynomial learning rate. This result is superior to prior approaches that show an exponential dependence [154, 234]. Finally, we run an experimental study within three different multi-agent environments, showing improved performance relative to standard baselines.

Since we relax the two limiting assumptions regarding learning from demonstrators in MARL, our hope is that this approach will spur successes in real-world applications, such as autonomous driving [96] and fighting wildfires [115], where MARL could use existing (sub-optimal) solutions as advisors. Particularly, the motivational examples provided in Chapter 3 (Section 3.1.1) are also relevant here, with the additional possibility of each agent having access to multiple sub-optimal advisors in these domains (as opposed to a single advisor considered in Chapter 3).

4.2 Related Work

In this section, we provide a reference to relevant literature for learning from external knowledge sources in RL/MARL. Since we already included an elaborate discussion in Section 3.1.2 of Chapter 3, we will keep this section brief just highlighting some important details relevant to this chapter.

Imitation learning (IL) is one approach to learning from external knowledge sources in RL. Within IL, the literature is divided into two parts, behaviour cloning [192] which mimics the expert policy, and inverse reinforcement learning which learns a reward function from the expert demonstrations [176]. More details on IL is already covered in Section 3.1.2.

As in Chapter 3, our work in this chapter is most related to the approach of *reinforcement learning from expert demonstrations* RLED [191]. The most popular RLED technique is *deep Q-learning from demonstrations* (DQfD) [100], which combines a temporal difference (TD) loss, an L2 regularization loss, and a classification loss that encourages actions to be close to that of the demonstrator. Another method, *normalized actor-critic* (NAC) [119], drops the classification loss and is more robust under imperfect demonstrations. However, NAC is prone to weaker performances than DQfD under good demonstrations due to the absence of classification loss. A different approach, *human agent transfer* (HAT) from [268], extracts information from limited demonstrations using a classifier, while *confidence-based human-agent transfer* (CHAT) [289] improves HAT by using a confidence measurement to safeguard against sub-optimal demonstrations. A related approach is the teacher-student framework [274], where a pretrained policy (teacher) can be used to provide limited advice to a learning agent (student). Subsequent works expand this framework towards interactive learning [7], however, almost all works in this area assume (at-least) a moderate level expertise for the teacher. Moreover, these are all independent methods primarily suited for single-agent environments.

Furthermore, external knowledge sources have also been used in MARL [225], where prior works often assume fully optimal experts [202, 292] or are only applicable to restrictive settings, such as fully cooperative or zero-sum competitive games [183, 226, 227, 287, 305]. The work in [226] introduced a framework where an agent can learn from its peers in a shared learning environment, in addition to learning from the environmental rewards. Here the advisor agent(s) can be sub-optimal, however this work only applies to cooperative environments. The work in [125] and [302] provide a cooperative teaching framework for hierarchical learning. For multi-agent general-sum environments, *advising multiple intelligent reinforcement agents - decision making* (ADMIRAL-DM) algorithm that we introduced in Chapter 3 is a suitable Q-learning approach that incorporates real-time information from a single online advisor.

One limitation of many prior works is the assumption of a single source of demonstration. In MARL, it may be possible to obtain advisors from different sources of knowledge that provide conflicting advice. In the single agent setting, Gimelfarb et al. [80] present a Bayesian model combination approach that uses demonstrations from multiple independent experts for reward shaping. This work requires the availability of near-optimal experts. Additionally, reward shaping is typically hard to get working in practice, where the agents have the possibility of ending up with a completely different behaviour than originally intended [35]. Also, convergence guarantees in RL may not necessarily be preserved under reward shaping [175]. Directly reshaping the policy using policy shaping [86] is an alternative to reward shaping. This has also been explored in the context of multiple expert learning using Bayesian approaches by Gimelfarb et al. [81]. However, the policy shaping approaches contain several limitations, such as difficulty in annealing the policy under non-stationarity and difficulty in credit assignment [81], which are exacerbated in multi-agent settings. Regarding action advising from multiple (possibly sub-optimal) demonstrators for the single-agent setting, Li et al. [145] provide the two-level Q-learning (TLQL) algorithm that learns simultaneously from multiple advisors. The TLQL maintains two Q-networks, where the first Q-network (high-level) keeps track of each advisor's performance and the second Q-network (low-level) learns the quality of each action. We improve upon TLQL and make it applicable to MARL.

4.3 Background

We consider the general-sum stochastic game setting, where the reward functions of the different agents can be related in any arbitrary fashion as introduced in Chapter 2. In this section, we introduce the TLQL algorithm from Li et al. [145].

Two-level Q-learning: The TLQL algorithm [145] enables single-agent learning under the simultaneous presence of multiple advisors providing conflicting demonstrations. Here, the challenge is to determine which advisor to trust in a given state. In this regard, the TLQL contains two Q-tables, a high-level Q-table (abbreviated as high-Q) and a low-level Q-table (abbreviated as low-Q). The high-Q stores the value of the $\langle s, ad \rangle$ pair, where $s \in S$ represents the state and $ad \in Ad$ represents an advisor (where Ad is the set of all advisors). The high-Q also stores the value of following the RL policy in addition to each advisor. The low-Q maintains the value of each state-action pair.

At each time step, the agent observes the state and selects an advisor (or the RL policy) from the high-Q using the ϵ-greedy strategy. If the high-Q returns an advisor, then the advisor's recommended action is performed. If the RL policy is returned, then an action is executed from the low-Q based on the ϵ-greedy strategy. The low-level table is updated using the vanilla Q-learning Bellman update [291]. Subsequently, the high-Q values are updated using a **synchronization** step. In this step, when an advisor's action is performed, the value of the advisor in the high-Q is simply assigned the value of that action from the low-Q. Finally, the high-Q value of the RL policy is updated using the relation $highQ(s, RL) = \max_a lowQ(s, a)$. This synchronization update of high-Q preserves the convergence guarantees, due to the policy improvement guarantee in single-agent Q-learning [256].

There are two significant limitations of TLQL. First, the high-Q that represents the value of the advisors also depends on the RL policy through the synchronization step. This Q value represents the value of taking the action suggested by the advisor at the current state and then following the RL policy from the next state onward. This definition is problematic since at the beginning of training, the RL policy is sub-optimal, and the objective is to accelerate learning by relying on external advisors and avoid using the RL policy at all. As advisors are evaluated at each state using the RL policy, it is likely that the most effective advisor among the set of advisors is not being followed until the RL policy improves. At this stage, it might be possible to simply follow the RL policy itself, defeating the purpose of learning from advisors. Second, the advisors have not been evaluated at the beginning of learning. Hence, it is impossible to find the most suitable advisor to follow, from the available advisors. While TLQL simply follows a ϵ-greedy exploration strategy,

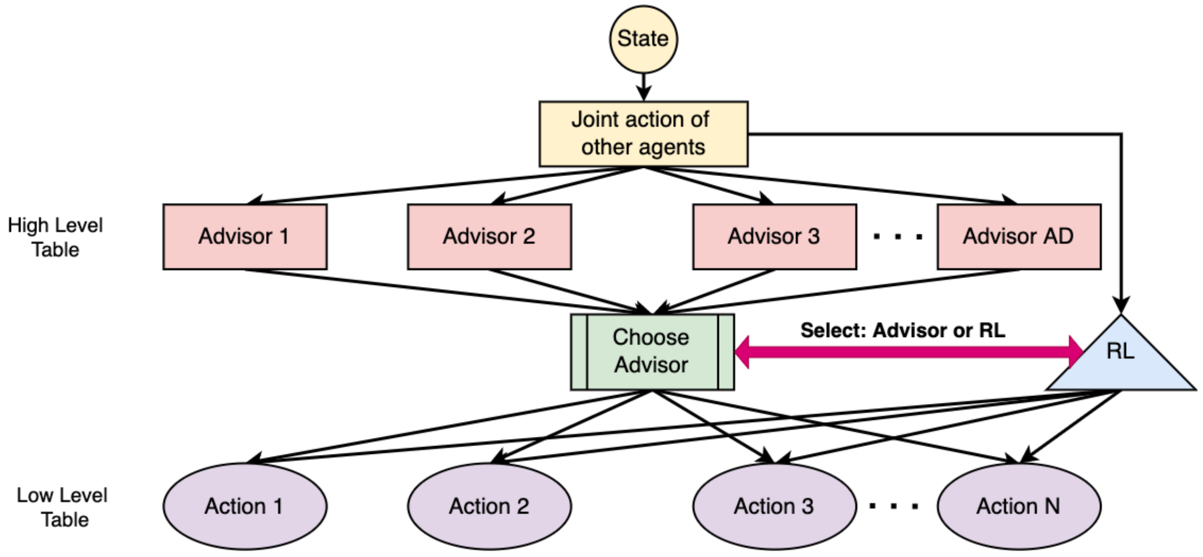


Figure 4.1: Structure of MA-TLQL, for a representative agent having access to AD advisors.

this approach could take many data samples to figure out the right advisor. We address both these limitations in our approach.

4.4 Two-level Architecture in MARL

We consider a general-sum stochastic game, where there are a set of agents $j \in \{1, \ldots, N\}$ that are learning a policy with an objective of providing a best response to the other agents as well as the environment. Each agent can access a set (same or different) of (possibly sub-optimal) advisors $ad^j \in \{ad^j_1, \ldots, ad^j_{AD}\}$, where ad^j represents an advisor of the agent j. Each advisor ad^j can be queried by the agent j to obtain an action recommendation at each state of the stochastic game. These online advisors provide real-time action-advice to the agent, which helps in learning to dynamically adapt to opponents. We consider a centralized training setting and assume 1) the advisors are fixed and do not learn, 2) the communication between agents and their advisors is free, 3) there is no communication directly between learning agents, 4) the environment is fully observable (i.e., an agent can observe the global state, all actions, and all rewards), and 5) the state and action spaces are discrete.

To make TLQL applicable to multi-agent settings, we parameterize both the Q-functions with the joint actions, as is common in practice [150]. Also, we do not maintain the RL policy in the high-Q table and do not perform a synchronization step. These steps are no longer needed to preserve the convergence results in multi-agent settings, since we do not have a policy improvement guarantee (unlike in single-agent settings) [264]. Instead, we choose to use the probabilistic policy reuse (PPR) technique [66], where a hyperparameter ($\epsilon' \in [0, 1]$) decides the probability of following any advisor(s) (i.e., using the high-Q) or the agents' own policy (i.e., using the low-Q) for action selection, at each time step during training. This hyperparameter starts with a high value (maximum dependence on the available advisor(s)) at the beginning of training and is decayed (linearly) over time. After some finite time step during training, the value of this hyperparameter goes to 0 (no further dependence on any advisor(s)) and the agent only uses its low-Q (own policy) for action selection. This helps in two ways: 1) in the limit ($t \to \infty$), a learning agent has the possibility of recovering from poor advising, and 2) eventually the trained agent can be deployed without the need for any advisor(s).

The general structure of our proposed *Multi-Agent Two-Level Q-Learning* (MA-TLQL) algorithm is given in Figure 4.1. Since we are in a fully observable setting, like [104], we specify that each agent maintains copies of the Q-tables of other agents from which it can obtain the joint actions of other agents for the current state. If such copies cannot be maintained, agents could use the previously observed actions of other agents for the joint action as done in prior works [254, 303]. As introduced in [145], we use the two-level architecture, where each agent will maintain a high-Q as well as a low-Q. The high-Q provides a value for the $\langle s, \boldsymbol{a}^{-j}, ad^j \rangle$ tuples, where $\boldsymbol{a}^{-j} = \{a^1, \dots, a^{j-1}, a^{j+1}, \dots, a^N\}$ is the joint action of all agents except agent j. As stated earlier, the objective of the high-Q is to determine the value of an advisor. The high-Q is updated using an evaluation update given by (with α as the learning rate and γ as the discount factor),

$$\begin{aligned} &highQ^j_{t+1}(s, \boldsymbol{a}^{-j}, ad^j) \\ &= highQ^j_t(s, \boldsymbol{a}^{-j}, ad^j) + \alpha\Big((r^j_t + \gamma highQ^j_t(s', \boldsymbol{a'}^{-j}, ad^j) - highQ^j_t(s, \boldsymbol{a}^{-j}, ad^j)\Big) \end{aligned} \tag{4.1}$$

where s and s' are the states at t and $t+1$, $\boldsymbol{a}^{-j}$ and $\boldsymbol{a'}^{-j}$ are joint actions at s. and s', respectively.

As described previously, a hyperparameter is used to decide between choosing to follow an advisor or the RL policy. If the agent follows an advisor, the high-Q values are used to select an advisor using an ensemble selection technique. Let, $\mathcal{Q} = \{highQ^j_1(s, \boldsymbol{a}^{-j}, ad^j_1), \dots, highQ^j_N(s, \boldsymbol{a}^{-j}, ad^j_M)\}$, be the set of all high-Q values of M advisors advising the same action a^j. Then the value of vote for action a^j, at state s and joint action $\boldsymbol{a}^{-j}$, denoted by $\mathcal{V}^j(s, \boldsymbol{a}^{-j}, a^j)$, is given by,

$$\mathcal{V}^j(s, \boldsymbol{a}^{-j}, a^j) = \max \mathcal{Q} + \textstyle\sum_{i \neq \arg\max \mathcal{Q}, highQ^j \in \mathcal{Q}} \frac{1}{\mu(s)} highQ^j_i. \tag{4.2}$$

Here, $\mu(s)$ is the number of times the agent has visited the state s. From Eq. 4.2, if an action a^j is recommended only by a single advisor, the value of the vote for that action will be equal to the advisor's high-Q value. After the value of votes for all actions is calculated, the selected action is the action with the maximum value of vote. In this way, when a state is visited many times, the advisor with the best high-Q estimate is likely to be followed (wisdom of individual). When a state is visited only a few times, then the action suggested by a majority of advisors is selected (wisdom of crowd). Subsequently, the advisor's action is executed and the high-Q of the advisor is updated using Eq. 4.1.

If the agent decides to use its RL policy, it uses its low-Q, which contains a value for the $\langle s, \boldsymbol{a}^{-j}, a^j \rangle$ tuples. At each time step, the low-Q is updated using a control update as given by (with α as the learning rate and γ as the discount factor),

$$\begin{aligned} &lowQ^j_{t+1}(s, \boldsymbol{a}^{-j}, a^j) \\ &= lowQ^j_t(s, \boldsymbol{a}^{-j}, a^j) + \alpha\Big(r^j_t + \gamma \max_{a'^j} lowQ^j_t(s', \boldsymbol{a}'^{-j}, a'^j) - lowQ^j_t(s, \boldsymbol{a}^{-j}, a^j)\Big). \end{aligned} \tag{4.3}$$

Now we describe how MA-TLQL addresses the limitations in TLQL described in Section 4.3. Note that the high-Q in MA-TLQL maintains the Q values of the advisor themselves, i.e., the value of following the advisor's policy from the current state onward (see Eq. 4.1). In this way, the coupling between the advisor values and the RL policy is removed (no synchronization). This addresses the first limitation of TLQL. Further, note that MA-TLQL uses an ensemble technique to leverage advisor knowledge during the early stages of learning. In later stages of learning, it switches to following the best advisor according to the high-Q estimates. This approach addresses the second limitation of TLQL. In Appendix B.5, we present a toy example that illustrates the limitations of TLQL, and the subsequent improvement induced by our update methods. While these improvements also apply to single-agent settings, we will keep our attention to multi-agent settings as is the focus of this chapter.

We provide the complete pseudocode for a tabular implementation of the MA-TLQL algorithm in Section 4.5 (Algorithm 6). Further, we extend this approach to large state-action environments using a neural network based implementation (Algorithm 7), which uses a target network and a replay buffer, as in the Deep Q-learning (DQN) algorithm [167]. We also provide an actor-critic implementation (Algorithm 8) which is suitable for CTDE [156]. We will refer to this algorithm as *multi-agent two-level actor-critic* (MA-TLAC). In MA-TLAC we have two actors and two critics (high-level and low-level), where the respective

Q-functions serve as the critic and the corresponding policies serve as the actors. In this CTDE method, agents can obtain global information (including actions and rewards of other agents) during training, while the agents only require access to its local observation (and no further information) during execution. This makes our method applicable to partially observable environments as in [156]. MA-TLAC applies to continuous state spaces as well.

4.5 Algorithm Pseudocode

A complete pseudocode of a tabular implementation of our Q-learning based algorithm (MA-TLQL) is given in Algorithm 6. All agents initialize a low-Q table and a high-Q table in line 2. Then at each state, all agents choose to perform an action in lines 8–20. This action can come from the advisor or the RL policy as described in Section 4.4. Then the action is executed, and the next state and reward are observed in line 21. Finally, the Q values for the low-Q as well as the high-Q are updated (line 22 and line 23) according to equations presented in Section 4.4. The value of ϵ' is linearly decayed from a high-value to a value close to zero during training (line 24).

To make Algorithm 6 applicable to high dimensional state and action spaces, we provide a function approximation-based implementation of MA-TLQL in Algorithm 7. Here neural networks are used as the function approximator, and the algorithm uses a separate target network and a replay buffer for training, as introduced in the well-known DQN algorithm [167]. The agent maintains a high-Q network and two low-Q networks (evaluation and target networks) and updates these networks using the temporal difference (T.D.) errors with the update equations presented in Section 4.4. If the full state of the stochastic game is not available, the agent can simply use its observation instead of the state, as applicable in most function approximation-based RL methods.

We also extend Algorithm 7 to an actor-critic implementation described in Algorithm 8. This algorithm is called multi-agent two-level actor-critic (MA-TLAC). This algorithm uses the policy as the actor and the Q-values as the critic, consistent with prior work [130]. We maintain two actors, and two critics to reflect the two-level (high and low) nature of our algorithm. The high-level actor determines an advisor and the high-level critic helps train the high-level actor, using the T.D. errors. Similarly, the low-level actor determines the appropriate action, with the low-level critic providing the T.D. errors for training. In MA-TLAC, since we use a separate actor network for advisor selection, we do not use the ensemble technique from Eq. 4.2. Instead, the high-level actor directly chooses one amongst the given advisors for the current state. The advantage of this algorithm is that it can be implemented using the popular CTDE paradigm [156], since the actors do not require the actions of other agents for action/advisor selection. In CTDE, global information (i.e., information from other agents) is available during training time but not available during execution. This CTDE based implementation also allows our method to be used in partially observable domains, since the actors can use the local observations for action/advisor selection while the critic can use the joint actions and states during training, as described in [156]. Also, since the ensemble technique (Eq. 4.2) is not used in MA-TLAC, it is also applicable to continuous state space environments as well (unlike MA-TLQL which is only applicable to environments with discrete state spaces).

All the algorithm pseudocodes provided in this section assume that all agents are using the same algorithmic steps for learning where it can maintain copies of updates of other agents, as done in prior work [104]. If this is not possible, the agents would directly use the observed previous actions of other agents for its updates.

Algorithm 6 MA-TLQL Tabular Method

```
1: Let $Ad^j$ denote a set of advisors available to the agent $j$.
2: For all $j \in 1, \ldots, N$, $s \in S$, and $a^j \in A^j$: $lowQ^j(s, a^j, \boldsymbol{a}^{-j}) \leftarrow 0$ where $\boldsymbol{a}^{-j} = [a^1, \ldots, a^{j-1}, a^{j+1}, \ldots, a^N]$
3: For all $s \in S$, $ad^j \in Ad^j$, and for all $j \in 1, \ldots, N$: $highQ^j(s, \boldsymbol{a}^{-j}, ad^j) \leftarrow 0$
4: Initialize a value for hyperparameters $\epsilon$ and $\epsilon'$ and $\eta$
5: while training is not finished do
6:     For each agent $j$, get the current state $s$
7:     For each agent $j$, get the joint actions of other agents $\boldsymbol{a}^{-j}$ at state $s$ using the respective copies and previous actions of all agents
8:     For each agent $j$, let $u$ be a uniform random number between 0 and 1
9:     if $u < \epsilon'$ then
10:        Let $u'$ be a uniform random number between 0 and 1
11:        if $u' < \eta$ then
12:            Choose an advisor using the high-$Q$ values of agent $j$ for the current state and joint action of other agents from Eq. 4.2 and use its action as the current action $a_t^j$
13:        else
14:            Set the advisor $ad^j$ as a random advisor from $Ad^j$ and use its action as the current action $a_t^j$.
15:        end if
16:    else if $u > \epsilon'$ and $u < \epsilon$ then
17:        Set the action $a_t^j$ as a random action from the action space $A^j$
18:    else
19:        Choose a greedy action $a_t^j$ from the low-$Q$ value using $s$ and the joint action of other agents
20:    end if
21:    Execute the joint action $\boldsymbol{a}$, observe joint reward $\boldsymbol{r}$ and the next state $s'$, where $\boldsymbol{a} = [a^1, \ldots, a^N]$ and $\boldsymbol{r} = [r^1, \ldots, r^N]$
22:    Update value of low-$Q$ for the agent $j$ using Eq. 4.3. Obtain the next actions for other agents $\boldsymbol{a'}^{-j}$ from the respective copies and previous actions of other agents
23:    If an advisor was chosen, update value of high-$Q$ of the advisor for the agent $j$ using Eq. 4.1
24:    At the end of each episode, linearly decay $\epsilon'$
25: end while
```

Algorithm 7 MA-TLQL Neural Network Method

```
1: Let $Ad^j$ denote a set of advisors available to the agent $j$
2: Initialize $Q_{\phi^j}, Q_{\phi^j_-}$ for all $j \in 1, \dots, N$ (to denote low-$Q$). Initialize $Q_{\theta^j}, Q_{\theta^j_-}$ for all $j \in 1, \dots, N$ (to denote high-$Q$)
3: Initialize a value for hyperparameters $\epsilon$ and $\epsilon'$ and $\eta$
4: while training is not finished do
5:     For each agent $j$, get the current state $s$
6:     For each agent $j$, get the joint actions of other agents $\boldsymbol{a}^{-j}$ at state $s$ using the respective copies and previous actions of all agents
7:     For each agent $j$, let $u$ be a uniform random number between 0 and 1
8:     if $u < \epsilon'$ then
9:         Let $u'$ be a uniform random number between 0 and 1
10:        if $u' < \eta$ then
11:            Choose an advisor using the high-$Q$ values of agent $j$ from Eq. 4.2 for the current state and joint action of other agents using the high-$Q$, $Q_{\theta^j}$, and use its action as the current action $a^j_t$
12:        else
13:            Set the advisor $ad^j$ as a random advisor from $Ad^j$ and use its action as the current action $a^j_t$.
14:        end if
15:    else if $u > \epsilon'$ and $u < \epsilon$ then
16:        Set the action $a^j_t$ as a random action from the action space $A^j$
17:    else
18:        Choose a greedy action $a^j_t$ from the low-$Q$, $Q_{\phi^j}$, using $s$ and the joint action of other agents
19:    end if
20:    Execute the joint action $\boldsymbol{a}$, observe joint reward $\boldsymbol{r}$ and the next state $s'$, where $\boldsymbol{a} = [a^1, \dots, a^N]$ and $\boldsymbol{r} = [r^1, \dots, r^N]$
21:    For each agent $j$, store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'} \rangle$ in replay buffer $\mathcal{D}^j$, where $\boldsymbol{a} = [a^1, \dots, a^N]$, $\boldsymbol{a'} = [a'^1, \dots, a'^N]$. Obtain the next actions for other agents $\boldsymbol{a'}^{-j}$ from the respective copies and previous actions of other agents
22:    If an advisor was used, for each agent $j$, store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'}, ad^j \rangle$ in replay buffer $\mathcal{D}'^j$, where $ad^j$ is the advisor
23:    Set the next state $s'$ as the current state $s$
24:    At the end of each episode, linearly decay $\epsilon'$
25:    while j = 1 to N do
26:        Sample a minibatch of K experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'} \rangle$ from $\mathcal{D}^j$
27:        Set $y^j = r^j + \gamma \max_{a'^j} Q_{\phi^j_-}(s', \boldsymbol{a'}^{-j}, a'^j)$ according to Eq. 4.3
28:        Update the $Q$-network $\phi^j$ by minimizing the loss $\mathcal{L}(\phi^j) = \frac{1}{K}\sum(y^j - Q_{\phi^j}(s, \boldsymbol{a}^{-j}, a^j))^2$
29:        Sample a minibatch of K experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \boldsymbol{a'}, ad^j \rangle$ from $\mathcal{D}'^j$
30:        Set $y^j = r^j + \gamma Q_{\theta^j_-}(s', \boldsymbol{a'}^{-j}, ad^j)$ according to Eq. 4.1
31:        Update the $Q$-network $\theta^j$ by minimizing the loss $\mathcal{L}(\theta^j) = \frac{1}{K}\sum(y^j - Q_{\theta^j}(s, \boldsymbol{a}^{-j}, ad^j))^2$
32:    end while
33:    Update the parameters of the target network for each agent by copying over the evaluation network every $\mathcal{T}$ steps: $\phi^j_- \leftarrow \phi^j$ and $\theta^j_- \leftarrow \theta^j$
34: end while
```

Algorithm 8 MA-TLAC

```
1: Let $Ad^j$ denote a set of advisors available to the agent $j$
2: Initialize $Q_{\phi^j}, \pi_{\phi^j_-}$, the low-level critic and actor networks for all $j \in \{1, \dots, n\}$
3: Initialize $Q_{\theta^j}, \pi_{\theta^j_-}$, the high-level critic and actor networks for all $j \in \{1, \dots, n\}$
4: Initialize a value for hyperparameters $\epsilon$ and $\epsilon'$ and $\eta$
5: while training is not finished do
6:     For each agent $j$, get the current state $s$
7:     For each agent $j$, let $u$ be a uniform random number between 0 and 1
8:     if $u < \epsilon'$ then
9:         Let $u'$ be a uniform random number between 0 and 1
10:        if $u' < \eta$ then
11:            Choose an advisor $ad^j$ using the high-level actor $\pi_{\theta^j}$, for the agent $j$, for the current state $s$, and use its action as the current action $a^j$
12:        else
13:            Set the advisor $ad^j$ as a random advisor from $Ad^j$ and use its action as the current action $a^j$
14:        end if
15:    else if $u > \epsilon'$ and $u < \epsilon$ then
16:        Set the action $a^j$ as a random action from the action space $A^j$
17:    else
18:        Choose a greedy action $a^j$ from the low-level actor, $\pi_{\phi^j}$, using $s$
19:    end if
20:    Execute the joint action $\boldsymbol{a}$, observe joint reward $\boldsymbol{r}$ and the next state $s'$, where $\boldsymbol{a} = [a^1, \dots, a^N]$ and $\boldsymbol{r} = [r^1, \dots, r^N]$
21:    For each agent $j$, obtain the joint actions of other agents $\boldsymbol{a}^{-j}$ (current observed actions of other agents) at state $s$
22:    Set $y^j = r^j + \gamma \max_{a'^j} Q_{\phi^j}(s', \boldsymbol{a'}^{-j}, a'^j)$ according to Eq. 4.3
23:    For each $j$, update the low-level critic by minimizing the loss $\mathcal{L}(\phi^j) = (y^j - Q_{\phi^j}(s, \boldsymbol{a}^{-j}, a^j))^2$
24:    For each $j$, calculate the advantage estimate using the relation $A(s, \boldsymbol{a}^{-j}, a^j) = y^j - \sum_{a^j} \pi_{\phi^j_-}(a^j|s) Q_{\phi^j}(s, \boldsymbol{a}^{-j}, a^j)$
25:    For each $j$, update the low-level actor using the log loss $\mathcal{J}(\phi^j_-) = \log \pi_{\phi^j_-}(a^j|s) A(s, \boldsymbol{a}^{-j}, a^j)$
26:    If an agent $j$ used an advisor $ad^j$, then update the advisor's $Q$-estimate.
27:    For each $j$, set $y^j = r^j + \gamma Q_{\theta^j}(s', \boldsymbol{a'}^{-j}, ad^j)$ according to Eq. 4.1
28:    Obtain the next actions for other agents $\boldsymbol{a'}^{-j}$ from the respective copies
29:    For each $j$, update the high-level critic by minimizing the loss $\mathcal{L}(\theta^j) = (y^j - Q_{\theta^j}(s, \boldsymbol{a}^{-j}, ad^j))^2$ where $ad^j$ is the advisor chosen by the agent $j$
30:    For each $j$, calculate the advantage estimate using the relation $A(s, \boldsymbol{a}^{-j}, ad^j) = y^j - \sum_{ad^j} \pi_{\theta^j_-}(ad^j|s) Q_{\theta^j}(s, \boldsymbol{a}^{-j}, ad^j)$
31:    For each $j$, update the high-level actor using the log loss $\mathcal{J}(\theta^j_-) = \log \pi_{\theta^j_-}(a^j|s) A(s, \boldsymbol{a}^{-j}, ad^j)$
32:    Set the next state as the current state $s = s'$
33:    At the end of each episode, linearly decay $\epsilon'$
34: end while
```

4.6 Theoretical Results

We present a convergence guarantee for the tabular MA-TLQL and characterize convergence rate. For the theoretical guarantees, we rely on the works of [22] and [64], which provide several fundamental results on the nature of stochastic iterative functions. We apply these to MA-TLQL in general-sum stochastic games using the three standard assumptions that we introduced in Chapter 3. These are Assumptions 1, 2, and 5.

Now we prove our theoretical results. First, we provide the convergence guarantee for the low-Q. Recall, the PPR technique guarantees that the MA-TLQL dependence on high-Q is only until a finite time step during training. After this step, the agent only uses its low-Q for action selection. As the convergence result is given in the time limit ($t \to \infty$), the influence of high-Q can be neglected for this result.

Theorem 4. *Given Assumptions 1, 2, 5, the low-Q values of an agent j converges to its Nash Q value in the limit ($t \to \infty$).*

Proof. Our proof will be along the lines of Theorem 3 in Chapter 3. We will provide the complete proof here to stay self contained.

Let us start with a lemma from prior work.

Lemma 10. *A random iterative process*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x) \tag{4.4}$$

where $x \in X$, $t = 0, 1, \ldots, \infty$, converges to zero with probability one (w. p. 1) if the following properties hold:

1. The set of possible states X is finite.

2. $0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t^2(x) < \infty$ w. p. 1, where the probability is over the learning rates α_t.

3. $||\mathbb{E}\{F_t(x)|\mathscr{P}_t\}||_W \leq \mathscr{K}||\Delta_t||_W + c_t$, where $\mathscr{K} \in [0, 1)$ and c_t converges to zero w. p. 1.

4. $\boldsymbol{var}\{F_t(x)|\mathscr{P}_t\} \leq K(1 + ||\Delta_t||_W)^2$, where K is some constant.

*Here $\mathscr{P}_t$ is an increasing sequence of σ-fields that includes the past of the process. In particular, we assume that $\alpha_t, \Delta_t, F_{t-1} \in \mathscr{P}_t$. The notation $||\cdot||_W$ refers to some (fixed) weighted maximum norm and the notation **var** refers to the variance.*

Across this proof, since we are only focusing on the low-Q values, with a small abuse of notation, we will use Q to denote the low-Q values. Now, we define a Nash operator P_t, using the following equation,

$$P_t Q^k(s, \boldsymbol{a}) = \mathbb{E}_{s' \sim p}[r_t^k(s, \boldsymbol{a}) + \gamma \pi_*^1(s') \cdots \pi_*^n(s') Q^k(s')] \tag{4.5}$$

where s' is the state at time $t+1$, $(\pi_*^1(s'), \ldots, \pi_*^n(s'))$ is the Nash equilibrium solution for the stage game $(Q^1(s'), \ldots, Q^n(s'))$, and p is the transition function. Q^k denotes the Q-value of a representative agent k.

Lemma 11. *Under Assumption 5, the Nash operator as defined in Eq. 4.5 forms a contraction mapping with the fixed point being the Nash Q-value of the game.*

Now, since the P_t operator forms a contraction mapping, $||P_t Q^j - P_t Q_*^j|| \leq \gamma ||Q^j - Q_*^j||$, is satisfied for some $\gamma \in [0, 1)$ and all Q^j. Here Q_*^j is the Nash Q-value of the agent j.

The objective is to apply Lemma 10 to show that the low-Q in MATLQL converge to the Nash Q values.

The first two conditions of Lemma 10 are satisfied from the Assumption 1 and Assumption 2. Now, comparing Eq. 4.4 and Eq. 4.3 we get that x can be associated with the state joint action pairs $(s, \boldsymbol{a})$ and $\Delta_t(s_t, a_t)$ can be associated with $Q_t^j(s, \boldsymbol{a}) - Q_*^j(s, \boldsymbol{a})$. Here, $Q_*^j(s, \boldsymbol{a})$ is the Nash Q value of the agent j.

Now we get

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x) F_t(x), \tag{4.6}$$

where

$$\begin{aligned} F_t(x) &= r_t^j + \gamma v^{Nash,j}(s_{t+1}) - Q_*^j(s_t, \boldsymbol{a_t}) + \gamma[\max_{a^j} Q_t^j(s_{t+1}, \boldsymbol{a}_{t+1}) - v^{Nash,j}(s_{t+1})] \\ &\triangleq r_t^j + \gamma v^{Nash,j}(s_{t+1}) - Q_*^j(s_t, \boldsymbol{a}_t) + C_t^j(s_t, \boldsymbol{a}_t) \qquad (4.7) \\ &\triangleq F_t^{Q,j}(s_t, \boldsymbol{a}_t) + C_t^j(s_t, \boldsymbol{a}_t) \end{aligned}$$

The Nash value function $v^{Nash,j}(s)$ of an agent j is defined as the expected cumulative discounted future rewards obtained by the agent j, given that all agents follow the Nash policy $\boldsymbol{\pi}_*$.

Here, we set, $F_t(s_t, \boldsymbol{a}_t) = F_t^{Q,j}(s_t, \boldsymbol{a}_t) = C_t^j(s_t, \boldsymbol{a}_t) = 0$ if $(s, \boldsymbol{a}) \neq (s_t, \boldsymbol{a}_t)$. Let the σ-field generated by all the random variables $(s_t, \alpha_t, a_t^1, \ldots, a_t^n, r_{t-1}, \ldots, s_1, \alpha_1, a_1, Q_0)$ be represented by $\mathscr{P}_t$. Now, all the Q-values are $\mathscr{P}_t$ measurable which makes Δ_t and F_t, $\mathscr{P}_t$ measurable and this satisfies the measurability condition of Lemma 10.

Hu and Wellman [104] showed that $v^{Nash,j}(s_{t+1}) \triangleq v^j(s', \pi_*^1, \ldots, \pi_*^n) = \pi_*^1(s') \cdots \pi_*^n(s') Q_*^j(s')$ (see the proof in Lemma 10 of [104]). Hence, from Lemma 11, we can show that the $\mathbb{E}[F_t^Q]$ forms a contraction mapping. This can be done using the fact that $\mathbb{E}(P_t Q_*) = Q_*$ (refer to Lemma 11 in [104]). Here, the norm is the maximum norm on the joint action.

Now, we have the following for all t,

$$||\,\mathbb{E}[F_t^{Q,j}(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]|| \leq \gamma ||Q_t^j(s_t, \boldsymbol{a}_t) - Q_*^j(s_t, \boldsymbol{a_t})|| = \gamma ||\Delta_t|| \tag{4.8}$$

Now from Eq. 4.7,

$$\begin{aligned} &||\,\mathbb{E}[F_t(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]|| \\ &\leq ||\,\mathbb{E}[F_t^{Q,j}(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]|| + ||\,\mathbb{E}[C_t^j(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]|| \\ &\leq \gamma ||\Delta_t|| + ||\,\mathbb{E}[C_t^j(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]|| \end{aligned} \tag{4.9}$$

This satisfies the third condition of Lemma 10 if $c_t = ||\,\mathbb{E}[C_t^j(s_t, \boldsymbol{a}_t)|\mathscr{P}_t]||$ converges to 0 with probability 1 (w. p. 1.). From the definition of C_t^j and the Assumption 3, it can be shown that the value of C_t^j converges to 0 in the limit of time (see Theorem 3 in Subramanian et al. [254]).

Thus, it follows from Lemma 10 that the process Δ_t converges to 0 and hence, low-Q value for an agent j, converges to Nash Q value Q_*^j.

□

Next, we provide sample complexity bounds for MA-TLQL. Instead of explicitly considering the high-Q values, we specify that the underlying joint policy has a covering time of L. The covering time specifies an upper bound on the number of time steps needed for all state-joint action pairs to be visited at least once starting from any state-joint action pair. Further, since the action selection is based on the low-Q values in the limit, we are most interested in the sample complexity of low-Q, where the dependence on the high-Q is effectively represented by L.

Regarding sample complexity, as is done in [64], we distinguish between two kinds of learning rates. Consider the following equation for the low-Q (rewriting Eq. 4.3 and dropping *low* for simplicity),

$$Q^j_{t+1}(s_t, \boldsymbol{a}_t) = \big(1 - \alpha^\omega_t(s_t, \boldsymbol{a}_t))(Q^j_t(s_t, \boldsymbol{a}_t)\big) + \alpha^\omega_t(s_t, \boldsymbol{a}_t)\big(r^j_t + \gamma \max_{a^j} Q^j_t(s_{t+1}, \boldsymbol{a}_{t+1})\big). \tag{4.10}$$

The value of $\alpha^\omega_t(s, \boldsymbol{a}) = \frac{1}{[\#(s,\boldsymbol{a},t)^\omega]}$, where $\#(s, \boldsymbol{a}, t)$ is the number of times until t that the joint action $\boldsymbol{a}$ is performed at s. Here, we consider $\omega \in (1/2, 1]$. The learning rate is linear if $\omega = 1$, and the learning rate is polynomial if $\omega \in (1/2, 1)$.

The next theorem provides a lower bound on the number of time steps needed for convergence in the case of a polynomial learning rate. From Assumption 1, let us specify that all rewards for the agent j are bounded by $R^j_{\max}$. We will consider a variable $Q^j_{\max}$, which denotes the maximum possible low-Q value for the agent j, which is bounded by $Q^j_{\max} = R^j_{\max}/(1 - \gamma)$. We will also use another variable $\beta = (1 - \gamma)/2$ to state our results.

Theorem 5. *Let us specify that with probability at least $1-\delta$, for an agent j, $||Q^j_T - Q^j_*||_\infty \leq \epsilon$. The bound on the rate of convergence of low-Q, Q^j_T, with a polynomial learning rate of factor ω is given by (with Q^j_* as the Nash Q-value of the agent j)*

$$T = \Omega\Big(\Big(\frac{L^{1+3\omega} Q^{2,j}_{\max} \ln(\frac{|S|\Pi_i|A_i|Q^j_{\max}}{\delta\beta\epsilon})}{\beta^2\epsilon^2}\Big)^{1-\omega} / L + \Big((\frac{L}{\beta} \ln \frac{Q^j_{\max}}{\epsilon} + 1)/2\Big)^{\frac{1}{1-\omega}}\Big). \tag{4.11}$$

Proof. Before beginning to proof, similar to [64], we clarify that we do not need to assume that the state-joint action pairs are being generated by any particular strategy. Same as in Theorem 4, across this proof, we will use Q to denote the low-Q values. For a representative agent j, we will focus on the value of $r^j_t = ||Q^j_t - Q^j_*||$ and the aim is to bound the time until $r^j_t \leq \epsilon$. Here the norm denotes the maximum difference of the Q values across all states and joint actions. The proof of this theorem follows the Theorem 4 in Dar and Mansour [64]. While the work of Dar and Mansour was restricted to single-agent MDPs, our result extends the analysis of Dar and Mansour to the general-sum stochastic game setting.

In line with the Eq. 4.10, let us consider a stochastic iterative process of the form,

$$X^j_{t+1}(i) = (1 - \alpha_t(i))X^j_t(i) + \alpha_t(i)((H_t X^j_t)(i) + w^j_t(i)). \tag{4.12}$$

As mentioned in Section 4.6, let us specify $\beta = \frac{1-\gamma}{2}$. Also, consider a constant $Q^j_{\max}$ where the $Q^j_{\max}$ denotes the maximum low-Q value possible to be obtained in the stochastic game by the agent j. Hence, the relation $||X_0|| \leq Q^j_{\max}$ holds. Further, let us consider a sequence D^j_k, with $D^j_1 = Q^j_{\max}$ and $D^j_{k+1} = (1-\beta)D^j_k$ for all $k \geq 1$. By the nature of this construction, the sequence D^j_k is guaranteed to converge to 0, since at each step the value of D^j_k is being continuously multiplied by a fractional value. Now we can prove the following result.

Lemma 12. *For every k, there exists a time τ_k such that, for any $t \geq \tau_k$ we have $||X^j_t|| \leq D^j_k$.*

The Lemma 12 guarantees that at time $t \geq \tau_k$, for any i the value of $||X^j_t(i)||$ is in the interval $[-D^j_k, D^j_k]$.

We can state the following lemma, where we bound the number of iterations until $D^j_i \leq \epsilon$.

Lemma 13. *For $m \geq \frac{1}{\beta}\ln(Q^j_{\max}/\epsilon)$ we have $D^j_m \leq \epsilon$.*

Proof. Since we have that $D^j_1 = Q^j_{\max}$ and $D^j_i = (1-\beta)D^j_{i-1}$, we will need a m that satisfies $D^j_m = Q^j_{\max}(1-\beta)^m \leq \epsilon$. By taking a logarithm on both sides, we get

$$\begin{aligned}
&\ln\Big((Q^j_{\max})(1-\beta)^m\Big) \leq \ln(\epsilon) \\
&\ln(Q^j_{\max}) + m\ln(1-\beta) \leq \ln(\epsilon) \\
&\implies \ln(Q^j_{\max}) - \ln(\epsilon) \leq -m\ln(1-\beta) \\
&\implies \ln(Q^j_{\max}/\epsilon) \leq m\textstyle\sum_{k=0}^{\infty}\beta^k/k \\
&\implies 1/\beta\ln(Q^j_{\max}/\epsilon) \leq m.
\end{aligned} \tag{4.13}$$

In the last step we omit the higher powers since β is a small fraction. This proves the result. ☐

Now, we define a sequence of times τ_k, with reference to the MA-TLQL low-level Q-updates with a polynomial learning rate. Let us define $\tau_{k+1} = \tau_k + L\tau^\omega_k$. Here the term ω denotes the decay factor of the learning rate with $\alpha^\omega_t = \frac{1}{(t+1)^\omega}$ for $\omega \in (1/2, 1)$. The term $L\tau^\omega_k$ specifies the number of steps needed to update each state joint action pair at least τ^ω_k times. The time between the τ_k and τ_{k+1} is denoted as the k^{th} iteration. Now we provide a definition for the number of times a state joint action pair is visited.

Definition 10. *Let $n(s, \boldsymbol{a}, t_1, t_2)$ be the number of times that the state joint action pair $(s, \boldsymbol{a})$ was performed in the time step interval $[t_1, t_2]$.*

Before providing the suitable bounds, we would like to provide some equations that relate the stochastic iterative technique given in Eq. 4.12 with the Q-update in Eq 4.10.

First we provide a slightly modified definition for a stochastic game, that will be useful for our further analysis (convey the meaning of our variables). Consider a stochastic game that can be defined as follows:

Definition 11. *A stochastic game is defined as $\langle \mathcal{S}, N, \mathbf{A}, P, \mathbf{R}, \gamma \rangle$ where $\mathcal{S}$ is a finite set of states, N is the finite set of agents, $|N| = n$, and $\mathbf{A} = A^1 \times \ldots \times A^n$ is the set of joint actions, where A^j is the finite action set of an agent j, and $\boldsymbol{a} = (a^1, \ldots, a^n) \in \mathbf{A}$ is the joint action where an agent j takes action $a^j \in A^j$. Furthermore, $P_{i,k}(\boldsymbol{a}) : S \times \mathbf{A} \times S \mapsto [0, 1]$ is the transition function that provides the probability of reaching state k from state i when all agents are performing the joint action $\boldsymbol{a} \in \mathbf{A}$ in state i, $\boldsymbol{R}(s, \boldsymbol{a}) = \{R^1(s, \boldsymbol{a}), \ldots, R^n(s, \boldsymbol{a})\}$ is the set of reward functions, where $R^j(s, \boldsymbol{a}) : S \times \mathbf{A} \mapsto \mathbb{R}^n$ is the reward function of the agent j, and γ is the discount factor satisfying $0 \leq \gamma < 1$.*

Towards the same, we define an operator H that can be represented as

$$(HQ^j)(i, \boldsymbol{a}) = \textstyle\sum_{k=0}^{|S|} P_{ik}(\boldsymbol{a})(R^j(i, \boldsymbol{a}) + \gamma \max_{b^j \in A^j} Q^j(k, \boldsymbol{b})). \tag{4.14}$$

Here the $\boldsymbol{b} = \{b^j, b^{-j}\}$, where b^{-j} denotes the joint action of all agents except the agent j.

Rewriting the Q-function with H we get,

$$Q^j_{t+1}(i, \boldsymbol{a}) = (1 - \alpha_t(i, \boldsymbol{a}))Q^j_t(i, \boldsymbol{a}) + \alpha_t(i, \boldsymbol{a})((HQ^j_t)(i, \boldsymbol{a}) + w^j_t(i, \boldsymbol{a})). \tag{4.15}$$

Let $\bar{i}$ be the state that is reached by performing joint action $\boldsymbol{a}$ at time t in state i and $r^j(i, \boldsymbol{a})$ be the reward observed by the agent j at state i; then

$$w^j_t(i, \boldsymbol{a}) = r^j(i, \boldsymbol{a}) + \gamma \max_{b^j \in A^j} Q^j_t(\bar{i}, \boldsymbol{b}) - \textstyle\sum_{k=0}^{|S|} P^j_{ik}(\boldsymbol{a})\Big(R^j(i, \boldsymbol{a}) + \gamma \max_{b^j \in A^j} Q^j_t(k, \boldsymbol{b})\Big). \tag{4.16}$$

From this construction w^j_t is bounded by $Q^j_{\max}$ for all t and has zero expectation. Further we will define two other sequences $W^j_{t;\tau}$ and $Y^j_{t;\tau}$ where τ represents some initial time. These are given by the following equations.

$$W^j_{t+1;\tau_k}(s,\boldsymbol{a}) = (1-\alpha^\omega_t(i))W^j_{t;\tau_k}(s,\boldsymbol{a}) + \alpha^\omega_t(i)w^j_t(s,\boldsymbol{a}) \tag{4.17}$$

where $W^j_{\tau_k;\tau_k}(s,\boldsymbol{a}) = 0$. The value of $W^j_{t;\tau_k}$ bounds the contributions of $w^j_t(s,\boldsymbol{a})$, to the value of Q^j_t, starting from an arbitrary τ_k. Now we have

$$Y^j_{t+1;\tau_k}(s,\boldsymbol{a}) = (1-\alpha^\omega_t(i))Y^j_{t;\tau_k}(s,\boldsymbol{a}) + \alpha^\omega_t(s,\boldsymbol{a})\gamma D^j_k \tag{4.18}$$

where $Y^j_{\tau_k;\tau_k} = D^j_k$.

Next, we can state a lemma that will bound the Q-functions w.r.t the sequences $W^j_{t;\tau_k}$ and $Y^j_{t;\tau_k}$.

Lemma 14. *For every state s and joint action $\boldsymbol{a}$ and time τ_k, we have*

$$\begin{aligned} &-Y^j_{t;\tau_k}(s,\boldsymbol{a}) + W^j_{t;\tau_k}(s,\boldsymbol{a}) \\ &\leq Q^j_*(s,\boldsymbol{a}) - Q^j_t(s,\boldsymbol{a}) \\ &\leq Y^j_{t;\tau_k}(s,\boldsymbol{a}) + W^j_{t;\tau_k}(s,\boldsymbol{a}) \end{aligned} \tag{4.19}$$

From the Lemma 14 we see that a bound on the difference r^j_t depends on the bound for $Y^j_{t;\tau_k}$ and $W^j_{t;\tau_k}$. So we can bound $Y^j_{t;\tau_k}$ and $W^j_{t;\tau_k}$ separately and two bounds together will provide a bound for r^j_t.

We first provide a result on the nature of the sequence $Y^j_{\tau_k;\tau_k}$.

Lemma 15. *The sequence $Y^j_{t;\tau_k}$ is a monotonically decreasing sequence.*

Next, we provide a bound on the value of $Y^j_{t;\tau_k}$.

Lemma 16. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate and assume that for any $t \geq \tau_k$ we have $Y^j_{t;\tau_k}(s,\boldsymbol{a}) \leq D_k$. Then for any $t \geq \tau_k + L\tau^\omega_k = \tau_{k+1}$ we have $Y^j_{t;\tau_k}(s,\boldsymbol{a}) \leq D^j(\gamma + \frac{2}{e}\beta)$.*

From Lemma 16 we have provided a bound for the term $Y^j_{t;\tau_k}$ for $t = \tau_{k+1}$ which will automatically hold for all $t \geq \tau_{k+1}$, since the term $Y^j_{t;\tau_k}$ is monotonically decreasing (from Lemma 15) and deterministic (from Eq. 4.18 and Lemma 14).

Next we bound the term $W^j_{t;\tau_k}$ by $(1-\frac{2}{e})\beta D^j_k$. The sum of the bounds for $W^j_{t;\tau_k}(s,\boldsymbol{a})$ and $Y^j_{t;\tau_k}(s,\boldsymbol{a})$, would be $(\gamma+\beta)D^j_k = (1-\beta)D^j_k = D^j_{k+1}$, as desired.

Now we state a definition for a sequence $\eta^{k,t,j}_i(s,\boldsymbol{a})$ and $W^{l,j}_{t;\tau_k}$.

Definition 12. *Let*

$$\begin{aligned}
&W^j_{t;\tau_k}(s,\boldsymbol{a})\\
&= (1-\alpha^\omega_t(s,\boldsymbol{a}))W^j_{t-1;\tau_k}(s,\boldsymbol{a}) + \alpha^\omega_t(s,\boldsymbol{a})w^j_t(s,\boldsymbol{a})\\
&= \textstyle\sum_{i=1}^{t-\tau_k} \eta^{k,t,j}_{i+\tau_k}(s,\boldsymbol{a})w^j_{i+\tau_k}(s,\boldsymbol{a}), \qquad (4.20)
\end{aligned}$$

where

$$\eta^{k,t,j}_{i+\tau_k}(s,\boldsymbol{a}) = \alpha^\omega_{i+\tau_k}(s,\boldsymbol{a})\Pi^t_{l=\tau_k+i+1}(1-\alpha^\omega_l(s,\boldsymbol{a})).$$

For bounding the sequence $W^j_{t;\tau_k}$ we consider the interval $t \in [\tau_{k+1}, \tau_{k+2}]$. First we provide a lemma that bounds the coefficients in this interval and bounds the influence of $w^j_t(s,\boldsymbol{a})$ in this interval.

Lemma 17. *Let $\tilde{w}^{j,t}_{i+\tau_k}(s,\boldsymbol{a}) = \eta^{k,t,j}_{i+\tau_k}(s,\boldsymbol{a})w^j_{i+\tau_k}(s,\boldsymbol{a})$, then for any $t \in [\tau_{k+1}, \tau_{k+2}]$ the random variable $\tilde{w}^j_{i+\tau_k}(s,\boldsymbol{a})$ has zero mean and bounded by $(L/\tau_k)^\omega Q^j_{\max}$*

Now, let us define $W^{l,j}_{t;\tau_k}(s,\boldsymbol{a}) = \sum_{i=1}^l \tilde{w}^t_{i+\tau_k}(s,\boldsymbol{a})$. The objective is to prove that this is a martingale difference sequence having bounded differences.

Lemma 18. *For any $t \in [\tau_{k+1}, \tau_{k+2}]$ and $1 \le l \le t$ we have that $W^{l,j}_{t;\tau_k}(s,\boldsymbol{a})$ is a martingale sequence that satisfies*

$$|W^{l,j}_{t;\tau_k}(s,\boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s,\boldsymbol{a})| \le (L/\tau_k)^\omega Q^j_{\max} \qquad (4.21)$$

The next lemma bounds the term $W^j_{t;\tau_k}$.

Lemma 19. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate. With probability at least $1-\frac{\delta}{m}$ we have that, for every state-joint action pair $|W^j_{t;\tau_k}(s,\boldsymbol{a})| \le (1-\frac{2}{e}\gamma D_k)$ for any $t \in [\tau_{k+1}, \tau_{k+2}]$, i.e.*

$$Pr\Big[\forall s, \boldsymbol{a} \forall t \in [\tau_{k+1}, \tau_{k+2}] : |W_{t;\tau_k}(s,\boldsymbol{a})| \le (1-\tfrac{2}{e})\beta D_k\Big] \ge 1-\tfrac{\delta}{m} \qquad (4.22)$$

given that

$$\tau_k = \Theta\Big(\big(\tfrac{L^{1+3\omega}Q^{2,j}_{\max}\ln(Q^j_{\max}|S|\Pi_i|A|_i m/(\delta\beta D_k))}{\beta^2 D_k^2}\big)^{1/\omega}\Big) \qquad (4.23)$$

Now that we have bounded for each iteration the time needed to achieve the desired probability $1 - \frac{\delta}{m}$. The next lemma provides a bound for error in all iterations.

Lemma 20. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate. With probability $1-\delta$, for every iteration $k \in [1, m]$ and time $t \in [\tau_{k+1}, \tau_{k+2}]$ we have $|W^j_{t;\tau_k}(s, \boldsymbol{a})| \leq (1 - \frac{2}{e})\beta D^j_k$, i.e.,*

$$Pr\Big[\forall k \in [1, m], \forall t \in [\tau_{k+1}, \tau_{k+2}], \forall s, \boldsymbol{a} : |W^j_{t;\tau_k}(s, \boldsymbol{a})| \leq (1 - \tfrac{2}{e})\beta D^j_k\Big] \geq 1 - \delta \tag{4.24}$$

given that

$$\tau_0 = \Theta\Big(\big(\frac{L^{1+3\omega} Q^{2,j}_{\max} \ln(Q^j_{\max}|S|\Pi_i|A|_i m/(\delta\beta\epsilon))}{\beta^2\epsilon^2}\big)^{1/\omega}\Big) \tag{4.25}$$

The next lemma solves the recurrence $\sum_{i=0}^{m-1} L\tau_i^\omega + \tau_0$ and derives the time complexity.

Lemma 21. *Let*

$$a_{k+1} = a_k + La_k^\omega = a_0 + \sum_{i=0}^{k} La_i^\omega \tag{4.26}$$

Then for any constant $\omega \in (0, 1)$, $a_k = \Omega(a_0^{1-\omega}/L + L^{\frac{1}{1-\omega}}((k+1)/2)^{\frac{1}{1-\omega}})$.

Finally, the proof of Theorem 5 follows from Lemmas 16, 20, 13, and 21.

Specifically, from the relation in Lemma 21 substitute the value of a_0 from the Lemma 20 (value of τ_0), and value of k from Lemma 13 (lower bound for m). From the Lemma 16 and Lemma 20, we see that the condition required in Lemma 14 is satisfied to provide a lower bound.

□

Assuming the same action spaces for all agents (i.e. $|A_1| = |A_2| = \cdots = |A_N| = |A|$), we note that the dependence on the number of agents is $\ln|A|^N = N\ln|A|$. Overall this results in a sub-linear dependence on the number of agents based on the value of ω. Under the stated assumptions, this result is far superior to prior works that report an exponential dependence on the number of agents when learning in general-sum stochastic game environments (with an arbitrary number of agents) for convergence to a Nash equilibrium [154, 234]. Further, the dependence on the state space and action space is sub-linear ($\ln|S|$), and the dependence on the covering time is $\Omega(L^{2\omega-3\omega^2} + L^{1/1-\omega})$, which is a polynomial dependence.

The next theorem considers the linear learning rate case.

Theorem 6. *Let us specify that with probability at least $1-\delta$, for an agent j, $||Q^j_T - Q^j_*||_\infty \leq \epsilon$. The bound on the rate of convergence of low-Q, Q^j_T, with a linear learning rate is given by*

$$T = \Omega\Big((L + \psi L + 1)^{\frac{1}{\beta} \ln \frac{Q^j_{\max}}{\epsilon}} \frac{Q^{2,j}_{\max} \ln(\frac{|S|\Pi_i|A_i|Q^j_{\max}}{\delta\beta\epsilon\psi})}{\beta^2\epsilon^2\psi^2}\Big), \tag{4.27}$$

where ψ is a small arbitrary positive constant satisfying $\psi \leq 0.712$.

Proof. In this proof, we first aim to show that the size of the kth iteration is $L(1+\psi)\tau_k$ for some positive constant $\psi \leq 0.712$. The covering time property guarantees that in $(1+\psi)L\tau_k$ steps, each pair of state-joint actions are performed at least $(1+\psi)\tau_k$ times. The sequence of times in this case is $\tau_{k+1} = \tau_k + (1+\psi)L\tau_k$. As in the proof of Theorem 5, we will first bound $Y^j_{t;\tau_k}$ and then bound $W^j_{t;\tau_k}$. As in the proof of Theorem 4, we will use Q to denote the low-Q values across this proof as well. The proof of this theorem follows the Theorem 5 in Dar and Mansour [64]. While the work of Dar and Mansour was restricted to single-agent MDPs, our result extends the analysis of Dar and Mansour to the general-sum stochastic game setting.

Similar to our approach in the proof of Theorem 5, first we provide a bound on the value of $Y^j_{t;\tau_k}$.

Lemma 22. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. Assume that for $t \geq \tau_k$ we have that $Y^j_{t;\tau_k}(s, \boldsymbol{a}) \leq D^j_k$. Then for any $t \geq \tau_k + (1+\psi)L\tau_k = \tau_{k+1}$ we have that $Y^j_{t;\tau_k}(s, \boldsymbol{a}) \leq (\gamma + \frac{2}{2+\psi}\beta)D^j_k$*

The following lemma enables the use of Azuma's inequality.

Lemma 23. *For any $t \geq \tau_k$ and $1 \leq l \leq t$ we have that $W^{l,j}_{t;\tau_k}(s, \boldsymbol{a})$ is a martingale sequence, which satisfies,*

$$|W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s, \boldsymbol{a})| \leq \frac{Q^j_{\max}}{n(s, \boldsymbol{a}, 0, t)} \tag{4.28}$$

The following lemma provides a bound for the stochastic term $W^j_{t;\tau_k}$.

Lemma 24. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. With probability at least $1-\frac{\delta}{m}$ we have that for every state-joint action pair $|W^j_{t;\tau_k}(s, \boldsymbol{a})| \leq \frac{\psi}{2+\psi}\beta D^j_k$, for any $t > \tau_{k+1}$ and any positive constant $\psi \leq 0.712$, i.e.,*

$$Pr\Big[\forall t \in [\tau_{k+1}, \tau_{k+2}] : W^j_{t;\tau_k}(s, \boldsymbol{a}) \leq \tfrac{\psi}{2+\psi}\beta D^j_k\Big] \geq 1 - \tfrac{\delta}{m} \tag{4.29}$$

given that $\tau_k \geq \Theta(\frac{Q^{2,j}_{\max} \ln(Q^j_{\max}|S||\Pi_i|A|_i m)/(\psi\delta\beta D_k)}{\psi^2\beta^2 D_k^2})$.

We have bounded for each iteration the time needed to achieve the desired probability of $1 - \frac{\delta}{m}$. The following lemma provides a bound for the error in all the iterations.

Lemma 25. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. With probability* $1-\delta$*, for every iteration* $k \in [1, m]$*, time* $t \in [\tau_{k+1}, \tau_{k+2}]$*, and any positive constant* $\psi \leq 0.712$*, we have* $|W^j_{t;\tau_k}| \leq \frac{\psi\beta D^j_k}{2+\psi}$*, i.e.,*

$$Pr\Big[\forall k \in [1, m], \forall t \in [\tau_{k+1}, \tau_{k+2}] : |W^j_{t;\tau_k}| \leq \frac{\psi\beta D^j_k}{2+\psi}\Big] \geq 1 - \delta \tag{4.30}$$

given that $\tau_0 = \Theta\Big(\frac{Q^{2,j}_{\max} \ln(Q^j_{\max}|S||A|m/(\delta\beta\epsilon\psi))}{\psi^2\beta^2\epsilon^2}\Big)$

Finally, the Theorem 6 follows from Lemmas 25, 22, 13, and the fact that $a_{k+1} = a_k + (1+\psi)La_k = a_0((1+\psi)L+1)^k$.

Specifically, substitute the value of k from Lemma 13 (lower bound for m), value of a_0 from Lemma 25 (value of τ_0), and see that Lemma 22 and Lemma 25 satisfy the condition for the lower bound in Lemma 14.

□

Theorem 6 shows that the bound is linear in the number of agents and sub-linear in the state and action spaces. Under the stated assumptions, this linear dependence on the number of agents is also superior to the current state-of-the-art results that report an exponential dependence on the number of agents when learning in general-sum stochastic game environments [154, 234]. Note, the dependence on the covering time in Theorem 6 could be much worse than that of Theorem 5, depending on the value of $Q^j_{\max}$ and ϵ. Since the value of ϵ is small, the dependence is certainly worse than that obtained for the polynomial learning rate case. Also, the dependence on $Q^j_{\max}$ is exponential as opposed to a polynomial dependence for Theorem 5.

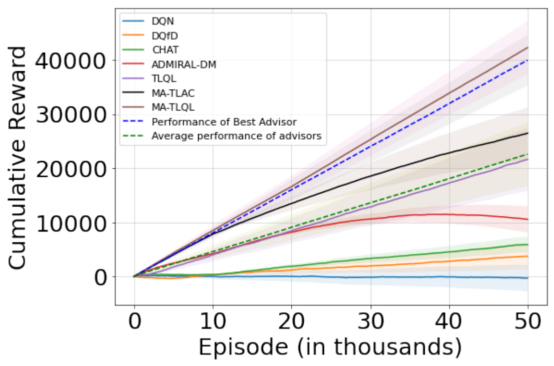
4.7 Experiments and Results

We consider three different experimental domains, one each for competitive, cooperative, and mixed settings, where each agent has access to a set of four advisors. We use neural network implementations of MA-TLQL and MA-TLAC, along with 5 other baselines:

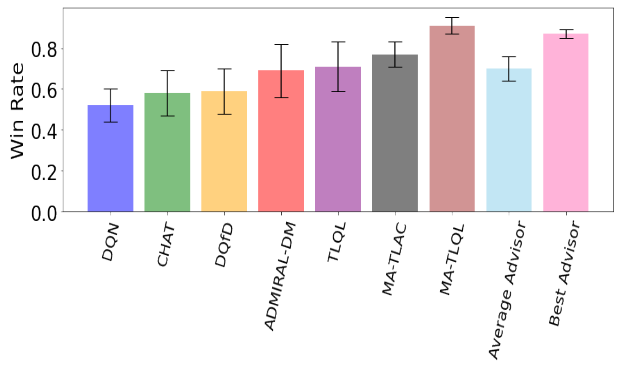
DQN [167], DQfD [100], CHAT [289], ADMIRAL-DM [254], and TLQL [145]. Since CHAT and ADMIRAL-DM assume the presence of a single advisor, we use a weighted random policy approach for implementing these two algorithms in the multiple-advisor setting, as done in [145]. If different advisors provide different actions at the same state, each action is weighted based on the number of advisors suggesting that action. For DQfD, during pre-training [100], we populate the replay buffer using advisor demonstrations from all the available advisors. For all our experiments, we will describe the critical details here, while the complete description is in Appendix B.2. All the experiments are repeated 30 times, with averages and standard deviations reported. For statistical significance we use the unpaired 2-sided t test and report p-values, where $p < 0.05$ is considered significant. The tests compare the highest performing algorithm (typically MA-TLQL) with the second-best baseline and best/average advisor performance. Appendix B.3 provides the hyperparameter details. and Appendix B.4 contains the wall clock times.

We conduct a total of seven experiments. Experiments 1–4 use the competitive, two-agent version of Pommerman [203] that we considered in Chapter 3. To recall, this is a complex environment, with each state containing roughly 200 elements related to agent position and special features (e.g., bombs). The reward function is sparse: agents only receive a terminal reward of $\{-1, 0, +1\}$. The experiments are conducted in two phases. In the first phase (training), our algorithms and the baseline algorithms train against a standard DQN opponent for 50,000 episodes, where we plot the cumulative rewards obtained. During this phase, algorithms can use advisors to accelerate training. In the second phase (execution), we test the performance of the trained policies against DQN for 1000 episodes, where we plot the win rate (fraction of games won) for each algorithm. During this phase, agents cannot access advisors, take no exploratory actions, and do not learn. All advisors pertaining to these experiments are rule-based agents.

Experiment 1: Our first experiment uses a set of four advisors ranked in terms of quality from Advisor 1 to Advisor 4. Here, Advisor 1 is the best advisor, capable of teaching the agent all skills needed to win the game of Pommerman, and Advisor 4 only suggests random actions. In Pommerman, there is a fixed set of six skills that an agent needs to master to be able to win [203]. Since this set of advisors can teach all these skills, we say the agent has access to a *sufficient set* of advisors. We plot the training and execution performances in Figure 4.2(a) and (b) respectively, including the performance of the best and average advisors (average of all Advisors 1–4) against DQN. MA-TLQL gives the best performance ($p < 0.01$) and is the only algorithm providing a better performance than the best advisor ($p < 0.11$) in both training and execution. MA-TLAC performs better than the average advisor ($p < 0.04$). None of the others show better performances



(a) Training

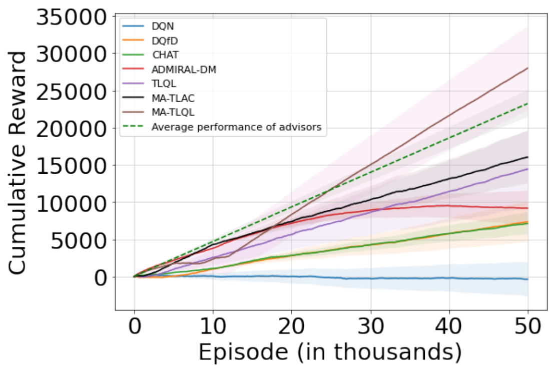


(b) Execution

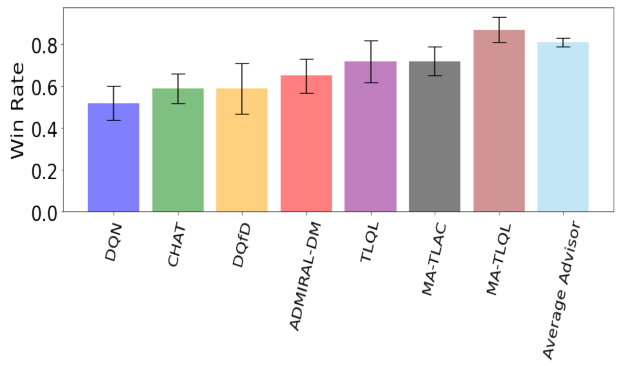
Figure 4.2: Results of two-agent version of Pommerman with four sufficient advisors of different quality.

than the average advisor. CHAT and ADMIRAL-DM are not capable of leveraging and distinguishing amongst a set of advisors. DQfD uses pre-training, which is not very effective in the non-stationary multi-agent context. Learning from online advising is preferable in MARL. Also, DQfD and CHAT are independent techniques that are not actively tracking the opponent's performance. While TLQL is capable of learning from multiple advisors, its independent nature in addition to coupling of advisor values with the RL policy reduces its effectiveness in multi-agent environments. MA-TLQL gives a better performance than MA-TLAC ($p < 0.01$). As noted previously, the Q-learning family of algorithms tends to induce a positive bias while using the maximum action value, which leads to providing the best possible response [280]. This explains the superior performance of MA-TLQL. We conclude that MA-TLQL is capable of leveraging a set of good and bad advisors. Further, the training results in Figure 4.2(a) show that MA-TLQL is able to learn a better policy faster than the baselines by using advisors ($p < 0.01$). The evaluation results in Figure 4.2(b) show that amongst all algorithms trained for the same number of episodes, MA-TLQL provides the best performance, when deployed without any advisors ($p < 0.01$). Both observations point to better sample efficiency in MA-TLQL. Additional experiments in Section 4.8 show that MA-TLQL comes to relying more on good advisors than poor advisors as compared to the other baselines.

Experiment 2: We use the same domain as in Experiment 1, but with a different set of advisors. Now, all four advisors can teach strictly different Pommerman skills. For example, Advisor 1 can teach how to escape the enemy (and nothing else), and Advisor 2 can teach



(a) Training

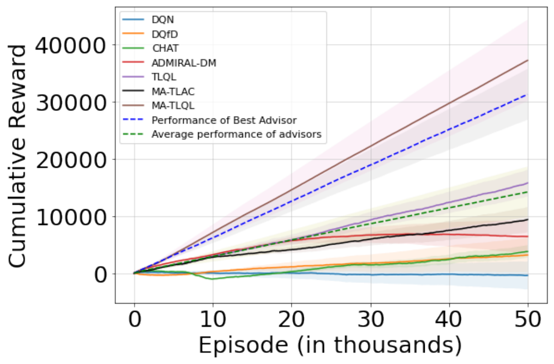


(b) Execution

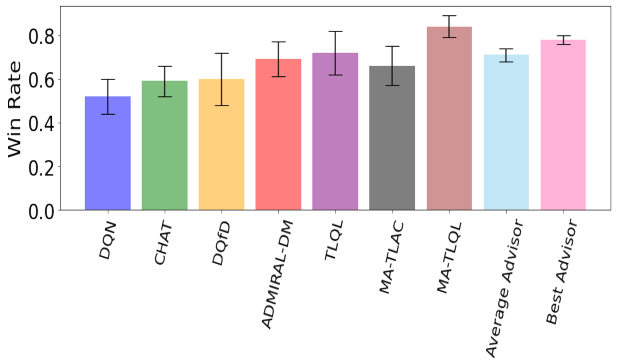
Figure 4.3: Results of two-agent version of Pommerman with four sufficient advisors of similar quality.

how to obtain necessary power-ups (and nothing else — full details are in Appendix B.2). These advisors provide psuedo-random action advice in states outside their expertise. This set of advisors is also a sufficient set. Now, learning agents must decide what advisor to listen to in the current state. From the training and execution results in Figure 4.3(a) and (b), we see that MA-TLQL gives the best overall performance ($p < 0.02$), exceeding the average performance of the four advisors ($p < 0.05$). Since all four advisors have similar quality, and none of the advisors dominates the others, we only choose to use the average performance of the four advisors in this experiment for comparison (the performance differences between advisors are small). We conclude that MA-TLQL is capable of leveraging the combined knowledge of a set of advisors with different individual expertise, at different time steps, during learning. MA-TLQL is also capable of learning faster than other baselines during the training phase, and providing a better performance as compared to the other baselines during the execution phase, consistent with our observations in Experiment 1.

Experiment 3: We use the same domain as in Experiment 1 but with a different set of four advisors. These advisors are similar to the set of advisors in our first experiment, where Advisor 1 gives the best advice throughout the domain, and Advisor 4 is random. However, this set of advisors is *not* capable of teaching all the strategies (i.e, Pommerman skills) needed to win in Pommerman, and compose an insufficient set (more details in Appendix B.2). It is critical for agents to learn from the environment in addition to the advisors. Training and execution results in Figure 4.4 shows the superior performance of MA-TLQL, the only algorithm that outperforms the best advisor ($p < 0.05$) and all



(a) Training



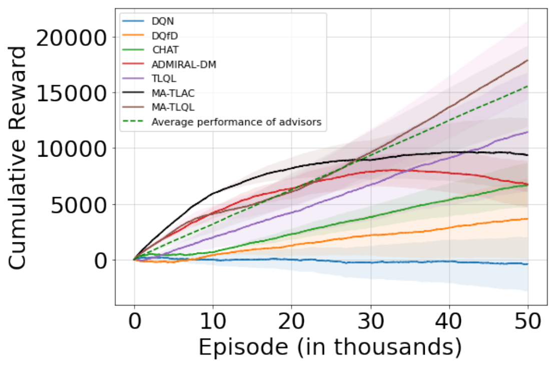
(b) Execution

Figure 4.4: Results of two-agent version of Pommerman with four insufficient advisors of different quality.

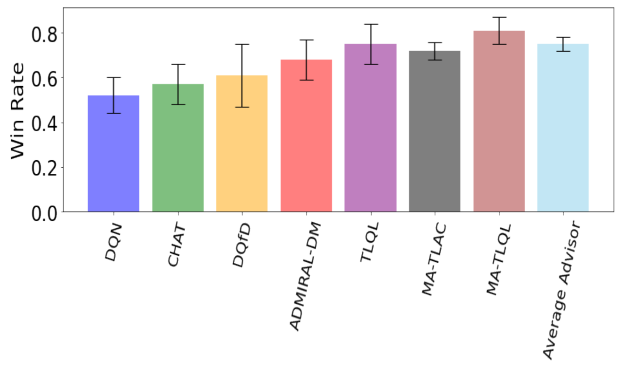
baselines ($p < 0.02$). Surprisingly, TLQL performs better than MA-TLAC ($p < 0.02$), likely due to the positive bias of Q-learning. This experiment reinforces the observation that MA-TLQL is capable of learning from good advisors and avoids bad advisors (also see Section 4.8). Since MA-TLQL clearly outperforms the best advisor, this experiment demonstrates that MA-TLQL can learn from both, advisors and through direct interactions with the environment, hence having a much improved sample efficiency as compared to other algorithms that learn only from the environment. This is observed during both training and execution.

Experiment 4: This experiment is similar to Experiment 2: four advisors have similar quality, but each understands a different Pommerman skill. However, our set of advisors in this experiment are insufficient to teach all the skills in Pommerman, and the agent must also learn from the environment. Figure 4.5 shows the training and execution results. We observe that MA-TLQL is capable of leveraging the advisor and learning from the environment to obtain the best performance, compared to the baselines ($p < 0.04$) and advisors ($p < 0.05$). Experiment 4 also shows that MA-TLQL learns from both, advisors and direct environmental interactions. It also leverages the combined expertise of advisors. This makes it more sample efficient than prior algorithms.

Experiment 5: We now switch to a four-agent version of Pommerman, which is two vs. two. To recall, this domain was already considered in Chapter 3. This domain is a



(a) Training

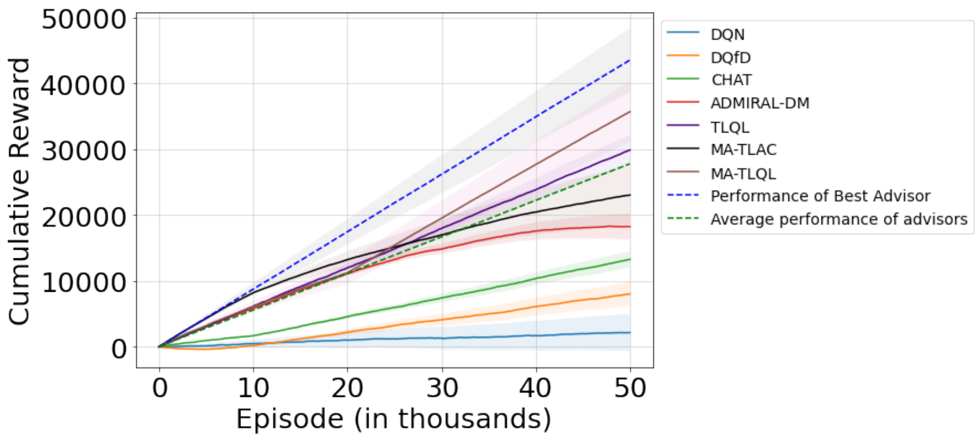


(b) Execution

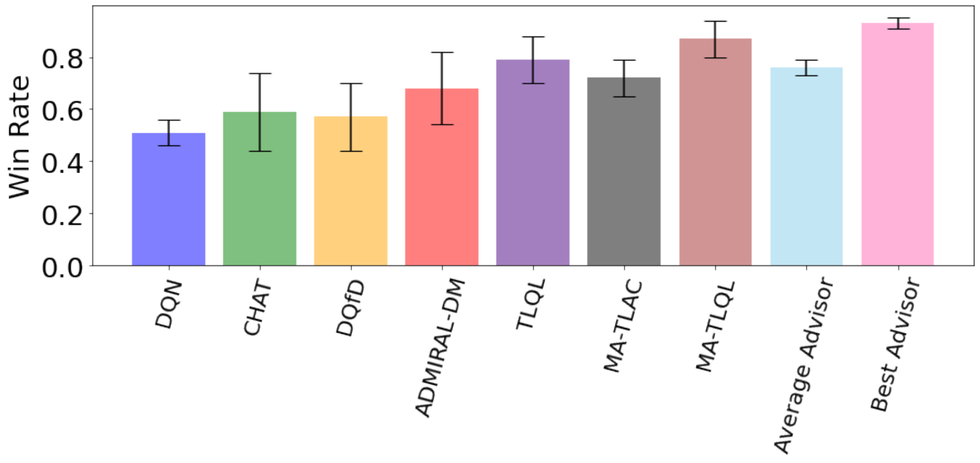
Figure 4.5: Results of two-agent version of Pommerman with four insufficient advisors of similar quality.

mixed setting as agents need to learn cooperative as well as competitive skills. Overall, this is a more complex domain with a larger state space. We consider four sufficient advisors of different quality, similar to Experiment 1. We conduct two phases — training (for 50,000 episodes) and execution (for 1000 episodes). The training and execution results in Figure 4.6 show that MA-TLQL provides the best performance compared to the baselines ($p < 0.04$) but does not perform better than the best available advisor. Since this is a more complex domain, MA-TLQL needs a larger training period for learning good policies. However, MA-TLQL still performs better than the average performance of the four advisors ($p < 0.03$). We conclude that although MA-TLQL's performance suffers in the more difficult mixed setting, it still outperforms all the other baselines and is still capable of distinguishing between good and bad advisors (see also Section 4.8). From both training and execution results in Figure 4.6, we note that MA-TLQL has a superior sample efficiency as compared to the other baselines.

Experiment 6: This experiment switches to the cooperative Pursuit domain [89], that we used in Chapter 3. To recall, in this environment, there are eight pursuer learning agents that learn to capture a set of 30 randomly moving targets (evaders). There are subtle differences in the current setting as compared to that considered in Chapter 3 (complete details about this setting are in Appendix B.2). We use four pretrained DQN networks as the advisors, learning for 500, 1000, 1500, and 2000 episodes, respectively. We again have two phases — training and execution. During the training phase, all algorithms are trained for 2000 episodes. The trained networks are then used in the execution phase for



(a) Training

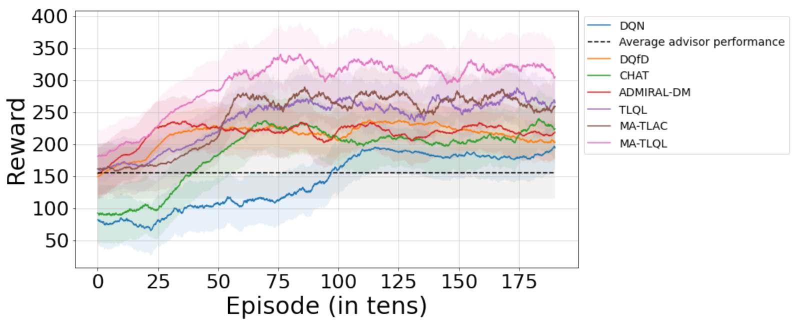


(b) Execution

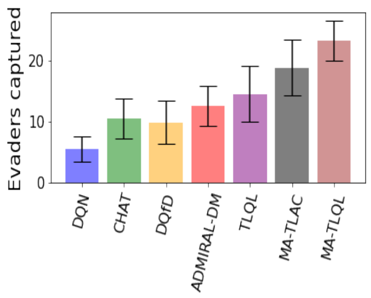
Figure 4.6: Results of the mixed setting (team version of Pommerman).

100 episodes with no further training or influence from advisors. Figure 4.7(a) plots the average rewards obtained during training and the Figure 4.7(b) plots the number of targets captured in the execution phase, where MA-TLQL shows the best performance ($p < 0.03$). Hence, MA-TLQL can outperform all baselines in a cooperative environment as well.

Experiment 7: This final experiment considers a mixed cooperative-competitive Predator-Prey environment which is a part of the Multi Particle Environment (MPE) suite [156]. Our implementation uses a discrete action space and a continuous state space (more details

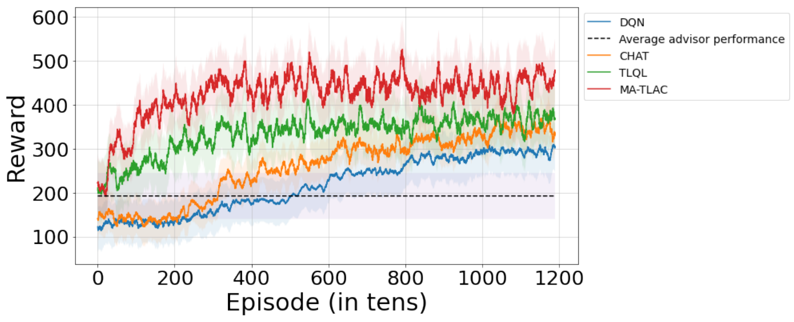


(a) Training

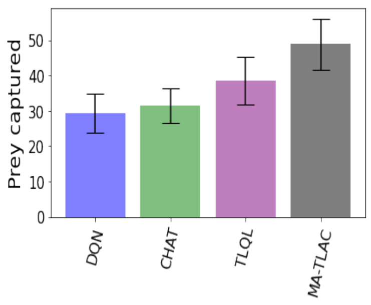


(b) Execution

Figure 4.7: Results in a fully cooperative Pursuit game.



(a) Training



(b) Execution

Figure 4.8: Results in a mixed CTDE based setting (Predator-Prey)

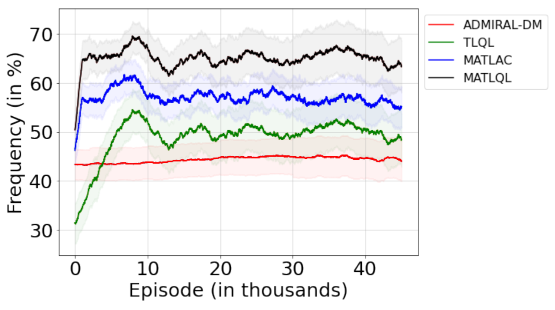
in Appendix B.2). There are a total of eight predators trying to capture eight prey (prey are not removed, but respawned upon capture). In our experiment, each algorithm trains the predators while the prey is trained using a standard DQN opponent. The experiments have two phases of training and execution, which is modelled as a CTDE setting. Here each agent obtains information about the actions and rewards of all other agents during training, but only has local observation during execution. Since this environment requires decentralization during execution, we omit the fully centralized MA-TLQL and ADMIRAL-DM. We also omit DQfD since it gave poor performances previously. As in Experiment 6, we use four pretrained DQN (predator) networks as advisors (trained for 1000, 2000, 7000, and 12000 episodes). The training phase is conducted for 12000 episodes and the execution is conducted for 100 episodes. The execution results in Figure 4.8(b) plots the average prey captured by each algorithm, which shows that MA-TLAC performs better than other algorithms ($p < 0.03$). Further, training results in Figure 4.8(a) show that MA-TLAC is the most sample efficient compared to other algorithms as it is able to leverage the available advisors better than the other algorithms. MA-TLAC quickly outperforms all other algorithms during training ($p < 0.03$).

From all the p-values across the seven experiments, we note that most of our observations are statistically significant. Despite observing MA-TLQL outperforming the best advisor in many of the experiments, some of these comparisons are not statistically significant (i.e., $p \geq 0.05$).

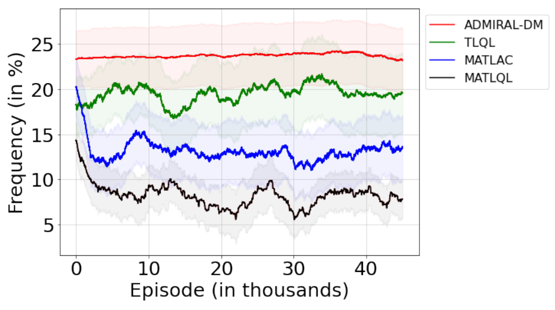
To summarize, Table 4.1 provides a brief description of all of our experimental settings along with the associated configuration of advisors.

4.8 Frequency Of Listening To Advisors

This section plots the frequency of listening to each advisor in some of our experiments. We would like to show that the MA-TLQL listens more to the good advisor and avoids the bad advisor more than other related baselines. For these experiments, we consider the TLQL [145] and ADMIRAL-DM [289] algorithms for comparison. Since the CHAT implementation uses the same method to choose advisors as ADMIRAL-DM (weighted random policy approach), we will omit CHAT for these results (performance is similar to ADMIRAL-DM). DQfD uses pretraining and does not choose advisors in an online fashion; hence we omit DQfD for these experiments as well.



(a) Good Advisor (Advisor 1)



(b) Bad Advisor (Advisor 4)

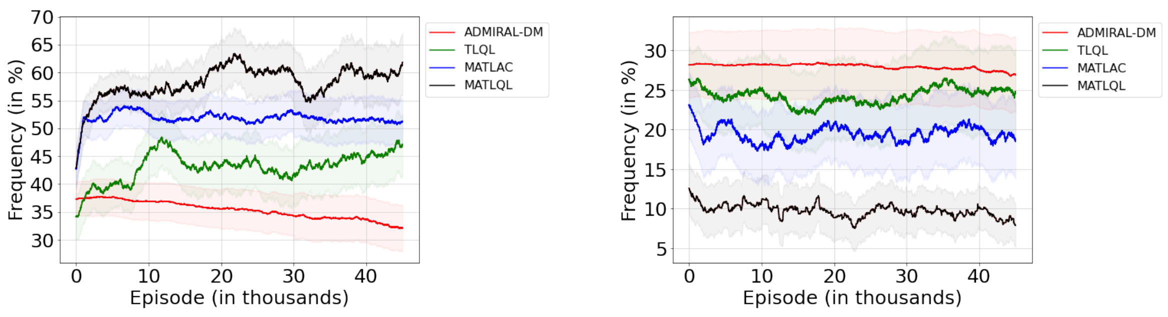
Figure 4.9: Frequency of listening to advisors in the two-agent Pommerman experiment with four sufficient advisors of different quality (Experiment 1).

We consider our first experiment in Section 4.7 (Experiment 1), where we had a set of four sufficient advisors of different quality. The first advisor (Advisor 1) had a better

Exp	Domain	Type	Advisors	# of training agents
1	Two-agent Pommerman	Competitive	4 sufficient advisors with different quality	2
2	Two-agent Pommerman	Competitive	4 sufficient advisors with similar quality	2
3	Two-agent Pommerman	Competitive	4 insufficient advisors with different quality	2
4	Two-agent Pommerman	Competitive	4 insufficient advisors with similar quality	2
5	Four-agent Pommerman	Mixed	4 sufficient advisors with different quality	4
6	Pursuit SISL	Cooperative	4 insufficient advisors with different quality	8
7	Predator-Prey MPE	Mixed (CTDE)	4 insufficient advisors with different quality	8

Table 4.1: Description of experimental settings.

quality than the others, and the agents must listen more to this advisor. On the other hand, Advisor 4 only suggested random actions and the agents are expected to avoid listening to this advisor. In Figure 4.9(a), we plot a curve that corresponds to the percentage of time steps an algorithm listened to Advisor 1 out of all the time steps the algorithm had an oppourtunity to listen to one of the available advisors. From the plots, we see that MA-TLQL listens more (compared to the other baselines) to this advisor (Advisor 1) from the beginning until the end of training. Since the MA-TLQL uses an ensemble technique to choose an advisor, this gives it a distinct advantage in the early stages of training. Further, since MA-TLQL performs an explicit evaluation of the advisors independent of the RL policy, it manages to listen more to the correct advisor as compared to other baselines. MA-TLAC also listens more to the good advisor as compared to the other baselines (TLQL, ADMIRAL-DM). Since TLQL couples the advisor evaluation with the RL policy, it listens a lot less to the good advisor as compared to MA-TLQL. Also, TLQL considers the RL policy as part of the high-level table, which makes it less reliant on advisors. This could be a problem when good advisors are available. In Figure 4.9(b), we plot the percentage of each algorithm listening to the bad advisor (Advisor 4). We see that MA-TLQL has the least dependence on this advisor as compared to all other algorithms. This reinforces our observation that MA-TLQL is most likely to choose to listen to the correct advisors in this multi-agent setting.

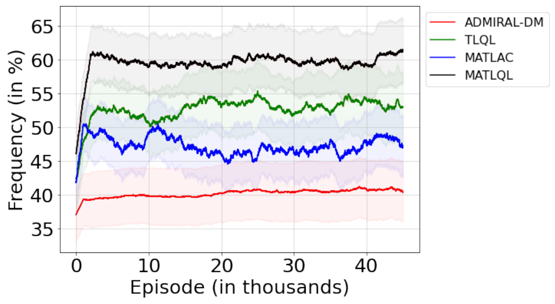


(a) Good Advisor (Advisor 1)   (b) Bad Advisor (Advisor 4)

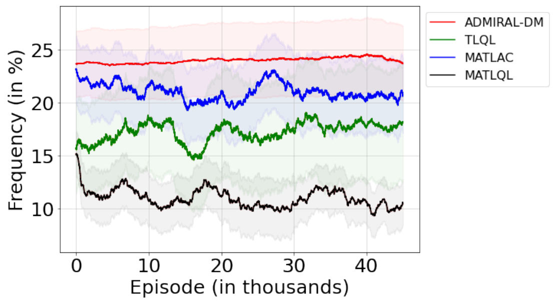
Figure 4.10: Frequency of listening to advisors in the two-agent Pommerman experiment with four insufficient advisors of different quality (Experiment 3).

We plot the percentage of listening to Advisor 1 and Advisor 4 in Experiment 3 from Section 4.7 that had an insufficient set of four advisors of decreasing quality. The results in Figure 4.10(a) and (b) show that MA-TLQL listens more to the good advisor and less to the bad advisor, same as our observations for Experiment 1.

Similarly, we plot the percentages of listening to the good and bad advisor in the



(a) Good Advisor (Advisor 1)



(b) Bad Advisor (Advisor 4)

Figure 4.11: Frequency of listening to advisors in the team Pommerman experiment with four sufficient advisors of different quality (Experiment 5).

Pommerman team environment (mixed setting) used in Experiment 5 in Figure 4.11(a) and (b). Here we plot the results for one of the two pommerman agents playing in the same team (the other agent's results are similar). Once again, we note that MA-TLQL listens more to the correct advisor than the other algorithms and better avoids the bad advisors compared to the other algorithms.

4.9 Conclusion

This chapter provided a principled approach for learning from multiple independent advisors in MARL. Inspired by Li et al. [145], we present a two-level architecture for multi-agent environments. We discuss two limitations in TLQL and address these limitations in our approach. Also, we provide a fixed point guarantee and sample complexity bounds regarding the learning of MA-TLQL. Additionally, we provided an actor-critic implementation that can work in the CTDE paradigm. Further, we performed an extensive experimental analysis of MA-TLQL and MA-TLAC in cooperative, competitive, and mixed settings, where we show that these algorithms are capable of suitably leveraging a set of advisors, and provide a better performance as compared to relevant baselines.

Chapter 5

Multi Type Mean Field Reinforcement Learning

Mean field theory provides an effective way of scaling multi-agent reinforcement learning algorithms to environments with many agents that can be abstracted by a virtual mean agent. In this chapter, we extend mean field multi-agent algorithms to multiple types. The types enable the relaxation of a core assumption in mean field reinforcement learning, which is that all agents in the environment are playing almost similar strategies and have the same goal. We conduct experiments on three different testbeds for the field of many agent reinforcement learning, based on the standard MAgents framework. We consider two different kinds of mean field environments: a) Games where agents belong to predefined types that are known a priori and b) Games where the type of each agent is unknown and therefore must be learned based on observations. We introduce new algorithms for each type of game and demonstrate their superior performance over state-of-the-art algorithms that assume that all agents belong to the same type and other baseline algorithms in the MAgent framework.

The contributions of this chapter are summarized as follows:

- Principled approach to learning in heterogeneous agent environments using mean-field multi-agent reinforcement learning algorithms. The approach groups agents into *types*, where agents are homogeneous within a type, but heterogeneous across types.
- Practical mean field reinforcement learning algorithms for learning in two type-based environments:

  - Known types: Agent types are known to all agents (common knowledge) upfront.
  - Unknown type: Types of others are unknown/unavailable and need to be determined during training.

- Theoretical performance bounds and convergence analysis for the provided algorithms.

- Experimental illustrations of performances of the provided mean field algorithms in heterogeneous many agent environments, along with demonstrated superior performances as compared to standard baselines.

The core contents of this chapter have been published as a full paper in AAMAS-2020 [248] and is available on arXiv [249].

5.1 Introduction

As discussed previously, the current MARL algorithms implemented for many agents involve the mean field theory and require some strong assumptions about the game environment to perform adequately. The important mean field approximation reduces a many agent problem into a simplified two agent problem where all the other participating agents are approximated as a single mean field. This mean field approximation, however, would be valid only for scenarios where all the agents in the environment can be considered similar to each other in objectives and abilities (i.e., fully homogeneous). Real world applications often have a set of agents that are diverse, and therefore it is virtually impossible to aggregate them into a single mean field.

In this chapter, we introduce the concept of multiple types to model agent diversity in mean field approximation for many agent reinforcement learning. The types are groupings applied to other agents in the environment in such a way that agents of the same "type" are "similar" in the sense that they play similar strategies and have similar goals. They are considered to be homogeneous, and the mean field approximation can be applied. Now, the modelling agent can consider each type to be a distinct agent, which has to be modelled separately. Within each type, the mean field approximation should still be reasonable as the agents within one particular type are more related to each other than other agents from different types. Thus, the many agent interaction is effectively reduced to M agent interactions where M is the number of types. Note that this is more complex than the simple two agent interaction considered by previous research and this can approximate a real world dynamic environment in a much better way.

Most real world applications for many agent reinforcement learning can be broadly classified into two categories. The first category involves applications where we have predefined types and the type of each agent is known a priori. Some common applications include games with multiple teams (like quiz competitions), multiple party elections with coalitions, airline price analysis with multiple airlines forming an alliance beforehand, etc. We call these applications predefined *known* type scenarios. The other category are applications that involve a large number of agents that may have different policies due to differences in their rewards, actions or observations. Common examples are demand and supply scenarios, stock trading scenarios, etc., where one type of agent may be risk averse while another type may be risk seeking. We call these applications pre-defined *unknown* type scenarios, since there are no true underlying types like the previous case and a suitable type therefore must be assigned through observations.

5.2 Background

Mean field theory approximates many agent interactions in a multi-agent environment into two agent interactions [136] where the second agent corresponds to the mean effect of all the other agents. This allows domains with many agents that were previously considered intractable to be revisited and scalable approximate solutions to be devised [162, 303]. Mean field reinforcement learning (MFRL) applies mean field approximation to stochastic games [303]. In the paper by Yang et al. [303], the multi-agent Q function for stochastic games is decomposed additively into local Q functions that capture pairwise interactions:

$$Q^j(s, \mathbf{a}) = \frac{1}{n^j} \sum_{k \in \eta(j)} Q^j(s, a^j, a^k). \tag{5.1}$$

Here n^j is the number of neighbours of the agent j and $\eta(j)$ is the index set of neighbouring agents. As specified in Yang et al. [303], in many agent environments each agent may be impacted only by a finite number of agents that are "closer" (according to some distance measure) to the state of an agent. For instance, in battle environments, nearby agents pose a greater threat than far away agents (which may not pose any threat). This zone of influence for an agent is defined as *neighbourhood* in this dissertation [1].

Notably, Yang et al. [303] showed that the pairwise decomposition can be well approximated by the mean field Q function $Q^j(s, \mathbf{a}) \approx Q^j_{MF}(s, a^j, \overline{a}^j)$ under the condition of fully

[1]Note that it may not be possible to specify concrete neighbourhoods in some environments, in which case the entire environment will constitute the neighbourhood (i.e., all agents in the environment may impact the learning of each individual agent).

homogeneous agents. The mean action $\overline{a}^j$ on the neighbourhood $\eta(j)$ of the agent j is expressed as $\overline{a}^j = \frac{1}{n^j}\sum_{k\in\eta(j)} a^k$ where a^k is the action of each neighbour k. In the case of discrete actions, a^k is a one-hot vector encoding and $\overline{a}^j$ is a vector of fractions corresponding to the probability that each action may be executed by an agent at random.

The mean field Q function can be updated in a recurrent manner as follows,

$$Q^j_{t+1}(s, a^j, \overline{a}^j) = (1-\alpha)Q^j_t(s, a^j, \overline{a}^j) + \alpha[r^j + \gamma v^j_t(s')] \tag{5.2}$$

where r^j is the reward obtained. The s, s' are the old and new states respectively. α_t is the learning rate and γ denotes the discount factor. The value function $v^j_t(s')$ for agent j at time t is given by,

$$v^j_t(s') = \sum_{a^j} \pi^j_t(a^j|s', \overline{a}^j)\, \mathbb{E}_{a_i^{-j}\sim\pi_{i,t}^{-j}}\, Q^j_t(s', a^j, \overline{a}^j). \tag{5.3}$$

Here, the term $\overline{a}^j$ denotes the mean action of all the other agents apart from j. The mean action for all the agents is first calculated using the relation, $\overline{a}^j = \frac{1}{N^j}\sum_k a^k, a^k \sim \pi^k_t(\cdot|s, \overline{a}^k_-)$, where π^k_t is the policy of the agent k at time t, and $\overline{a}^k_-$ represents the previous mean action for the neighbours of the agent k. N^j denotes the total number of agents in the neighbourhood of the agent j. Then, the Boltzmann policy for each agent j is calculated using β, which is the Boltzmann softmax parameter,

$$\pi^j_t(a^j|s, \overline{a}^j) = \frac{exp(-\beta Q^j_t(s, a^j, \overline{a}^j))}{\sum_{a^{j'}\in A^j} exp(-\beta Q^j_t(s, a^{j'}, \overline{a}^j))}. \tag{5.4}$$

5.3 Mean Field MARL and Types

We consider environments where there are M types that the neighbouring agents can be classified into. In this chapter, we assume that the Q function decomposes additively according to a partition of the agents into X^j subsets that each include one representative agent of each type. This decomposition can be viewed as a generalization of pairwise decompositions to multiple types since each term depends on a representative from each type.

Let the standard multi-agent Q function be $Q^j(s, \boldsymbol{a})$, then,

$$Q^j(s, \boldsymbol{a}) = \frac{1}{X^j} \sum_{i=1}^{X^j} [Q^j(s, a^j, a_1^{k_i}, a_2^{k_i}, \dots, a_M^{k_i})]. \quad (5.5)$$

Here we have a total of M types and a_m^k denotes the action of agent k belonging to type m in the neighbourhood of agent j. Notice that this representation of the Q function given in Eq. 5.5 includes the interaction with each one of the types and is not a simple pairwise interaction as done by Yang et al. [303]. Let us assume that we have a scheme in which we can classify each agent into one of these subsets. Note that we do not need each group to contain an equal number of agents as we can always make a new subset that contains one agent of a type and other agents to be place holder agents (dead agents) of other types. We will relax this requirement of decomposition shortly and in practice we do not need to make these subsets at all.

5.3.1 Mean Field Approximation

We assume discrete action spaces and use a one hot representation of the actions as in Yang et al. [303]. The one hot action of each agent k belonging to type m in the neighbourhood of agent j is represented as $a_m^{k_m} = \overline{a}_m^j + \hat{\delta}^{j,k_m}$ where $\overline{a}_m^j$ is the mean action of all agents in the neighbourhood of agent j belonging to type m and $\hat{\delta}^{j,k_m}$ is the deviation between the action of an agent and the mean action of its type.

Let $\delta^{j,k_i} = [\hat{\delta}^{j,k_1}; \hat{\delta}^{j,k_2}; \cdots; \hat{\delta}^{j,k_M}]$ be a vector obtained by the concatenation of all such deviations of the agents in the neighbourhood of the agent j belonging to each of the M types (all agents of a single subset). Similar to [303], we apply Taylor's theorem to expand the Q function in Equation 5.5 to get,

$Q^j(s, \mathbf{a}) = \frac{1}{X^j} \sum_{i=1}^{X^j} Q^j(s, a^j, a_1^{k_i}, a_2^{k_i}, \ldots, a_M^{k_i})$

$= \frac{1}{X^j} \sum_{i=1}^{X^j} [Q^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j)$

$+\nabla_{\overline{a}_1^j, \ldots, \overline{a}_M^j} Q^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j) \cdot \delta^{j,k_i}$

$+\frac{1}{2}\delta^{j,k_i} \cdot \nabla^2_{\tilde{a}_1^j, \ldots, \tilde{a}_M^j} Q^j(s, a^j, \tilde{a}_1^j, \ldots, \tilde{a}_M^j) \cdot \delta^{j,k_i}]$

$= Q^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j)$

$+\nabla_{\overline{a}_1^j, \ldots, \overline{a}_M^j} Q^j(s, a^j, \overline{a}_1^j, \overline{a}_2^j, \ldots, \overline{a}_M^j) \cdot [\frac{1}{X^j} \sum_{i=1}^{X^j} \delta^{j,k_i}]$

$+\frac{1}{2X^j} \sum_{i=1}^{X^j} [\delta^{j,k_i} \cdot \nabla^2_{\tilde{a}_1^j, \ldots, \tilde{a}_M^j} Q^j(s, a^j, \tilde{a}_1^j, \tilde{a}_2^j, \ldots, \tilde{a}_M^j) \cdot \delta^{j,k_i}]$

$= Q^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j) + \frac{1}{2X^j} \sum_{i=1}^{X^j} [R^j_{s,a^j}(a^{k_i})] \approx Q^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j)$

where $R^j_{s,a^j}(a^k) \triangleq \delta^{j,k} \cdot \nabla^2_{\tilde{a}_1^j, \ldots, \tilde{a}_M^j} Q^j(s, a^j, \tilde{a}_1^j, \tilde{a}_2^j, \ldots, \tilde{a}_M^j) \cdot \delta^{j,k}$

which is the Taylor polynomial remainder. The summation term $[\frac{1}{X^j} \sum_{i=1}^{X^j} \delta^{j,k_i}]$ sums out to 0. Finally, if we ignore the remainder terms R^j_{s,a^j}, we obtain the following approximation,

$$Q^j(s, \mathbf{a}) \approx Q^j_{MTMF}(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j). \tag{5.6}$$

The magnitude of this approximation depends on the deviation $\hat{\delta}^{j,k_m}$ between each action $a_m^{k_m}$ and its mean field approximation $\overline{a}_m^j$. More precisely, we can quantify the overall effect of the mean field approximation by the average deviation $\sum_k ||\hat{\delta}_k||_2/N$. Theorems 7 and 8 show that the average deviation is reduced as we increase the number of types and Theorem 9 provides an explicit bound on the approximation in Eq. 5.6 based on the average deviation.

Theorem 7. *When there are two types in the environment, but they have been considered to be the same type, the average deviation induced by the mean field approximation is bounded as follows,*

$$\frac{\sum_k ||\hat{\delta}_k||_2}{N} \leq \frac{K_1}{N}\epsilon_1 + \frac{K_2}{N}\epsilon_2 + \frac{K_1}{N}\alpha_1 + \frac{K_2}{N}\alpha_2 \tag{5.7}$$

where N denotes the total number of agents in the environment; K_1 and K_2 denote the total number of agents of types 1 and 2, respectively; ϵ_1 and ϵ_2 are bounds on the average deviation for agents of types 1 and 2 respectively.

$$\tfrac{1}{K_1}(\textstyle\sum_{k_1}||a_{k_1}-\overline{a}_1||_2) \leq \epsilon_1; \tfrac{1}{K_2}(\textstyle\sum_{k_2}||a_{k_2}-\overline{a}_2||_2) \leq \epsilon_2$$

Similarly, α_1 and α_2 denote the errors induced by using a single type (instead of two types) in the mean field approximation.

$$\alpha_1 = ||\overline{a}_1 - \overline{a}||_2; \alpha_2 = ||\overline{a}_2 - \overline{a}||_2 \tag{5.8}$$

Furthermore, a_{k_1} denotes an action of an agent belonging to type 1 and a_{k_2} denotes an action of an agent belonging to type 2. Here $\overline{a}$ denotes the mean field action of all the agents, $\overline{a}_1$ denotes the mean action of all agents of type 1 and $\overline{a}_2$ denotes the mean action of all agents of type 2.

Proof. Since we have considered every agent to belong to only one type, the deviation between the agent's action and the overall mean action is the error estimate. Hence, we will have the following,

$$\frac{\sum_k ||\hat{\delta}_k||_2}{N} = \frac{\sum_k ||a^j - \overline{a}^j||_2}{N}$$

$$\frac{\sum_k ||\hat{\delta}_k||_2}{N} = \tfrac{1}{N}(\textstyle\sum_{k_1}||a_{k_1}-\overline{a}|| + \sum_{k_2}||a_{k_2}-\overline{a}||).$$

The superscript (j) and subscript (2) has been dropped for simplicity.

$$\begin{aligned}
&= \tfrac{1}{N}(\textstyle\sum_{k_1}||a_{k_1}-\overline{a_1}+\overline{a}_1-\overline{a}|| + \sum_{k_2}||a_{k_2}-\overline{a_2}+\overline{a_2}-\overline{a}||) \\
&\leq \tfrac{1}{N}(\textstyle\sum_{k_1}||a_{k_1}-\overline{a_1}|| + \sum_{k_1}||\overline{a}_1-\overline{a}|| + \sum_{k_2}||a_{k_2}-\overline{a_2}|| + \sum_{k_2}||\overline{a_2}-\overline{a}||) \\
&= \tfrac{1}{N}(\textstyle\sum_{k_1}||a_{k_1}-\overline{a_1}|| + K_1||\overline{a}_1-\overline{a}|| + \sum_{k_2}||a_{k_2}-\overline{a_2}|| + K_2||\overline{a_2}-\overline{a}||) \\
&\leq \tfrac{K_1}{N}\epsilon_1 + \tfrac{K_2}{N}\epsilon_2 + \tfrac{K_1}{N}\alpha_1 + \tfrac{K_2}{N}\alpha_2.
\end{aligned} \tag{5.9}$$

□

Theorem 8. *When there are two types in the environment, and they have been considered to be different types, the average deviation induced by the mean field approximation is bounded as follows,*

$$\frac{\sum_k ||\hat{\delta}_k||_2}{N} \leq \frac{K_1}{N}\epsilon_1 + \frac{K_2}{N}\epsilon_2. \tag{5.10}$$

The variables have the same meaning as in Theorem 7.

Proof. In this scenario we will have,

$$\begin{aligned} &\tfrac{\sum_k ||\hat{\delta}_k||_2}{N} = \tfrac{1}{N}(\textstyle\sum_{k_1} ||a_{k_1} - \overline{a}_1||_2 + \sum_{k_2} ||a_{k_2} - \overline{a}_2||_2) \\ &= \tfrac{K_1}{N}\tfrac{\sum_{k_1} ||a_{k_1} - \overline{a}_1||_2}{K_1} + \tfrac{K_2}{N}\tfrac{\sum_{k_2} ||a_{k_2} - \overline{a}_2||_2}{K_2} \leq \tfrac{K_1}{N}\epsilon_1 + \tfrac{K_2}{N}\epsilon_2. \end{aligned} \tag{5.11}$$

□

We presented Theorems 7 and 8 to demonstrate the reduction in the bound on the average deviation as we increase the number of types from 1 to 2. Similar derivations can be performed to demonstrate that the bounds on the average deviation decrease as we increase the number of types (regardless of the true number of types).

Let ϵ be a bound on the average deviation achieved based on a certain number of types: $\frac{\sum_{k=1}^{X} ||\delta a||_2}{X} \leq \epsilon$. The following theorem bounds the error of the approximate mean field Q function as a function of ϵ and the smoothness L of the exact Q function.

Theorem 9. *When the Q function is additively decomposable according to Equation 5.5, and it is L-smooth, then the Multi Type Mean Field Q function provides a good approximation bounded by*

$$|Q^j(s, \mathbf{a}) - Q^j_{MTMF}(s, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M)| \leq \frac{1}{2}L\epsilon. \tag{5.12}$$

Proof. We rewrite the expression for the Q function as $Q(a) \triangleq Q^j(s, a^j, \overline{a}^j_1, \overline{a}^j_2, \cdots, \overline{a}^j_M)$. Suppose that Q is L-smooth, where its gradient ∇Q is Lipschitz-continous with constant L such that for all $a, \overline{a}$,

$$||\nabla Q(a) - \nabla Q(\overline{a})||_2 \leq L||a - \overline{a}||_2 \tag{5.13}$$

where $||.||_2$ indicates the l_2-norm. Note that all the eigenvalues of $\nabla^2 Q$ can be bounded in the symmetric interval $[-L, L]$. As the Hessian matrix $\nabla^2 Q$ is real symmetric and hence diagonalizable, there exists an orthogonal matrix U such that $U^T[\nabla^2 Q]U = \Lambda \triangleq diag[\lambda_1, \ldots, \lambda_D]$. It then follows that,

$$\delta a \cdot \nabla^2 Q \cdot \delta a = [U\delta a]^T \Lambda [U\delta a] = \textstyle\sum_{i=1}^{D} \lambda_i [U\delta a]_i^2 \tag{5.14}$$

with

$$-L||U\delta a||_2 \leq \textstyle\sum_{i=1}^{D} \lambda_i [U\delta a]_i^2 \leq L||U\delta a||_2. \tag{5.15}$$

Let, $\hat{\delta}_m a = a_m^j - \overline{a}_m^j$, consistent with the previous definition. Recall that the term δ is then $\delta a = [\hat{\delta}_1 a; \hat{\delta}_2 a; \cdots; \hat{\delta}_M a]$, where a is the one-hot encoding for D actions, and $\overline{a}$ is a D-dimensional categorical distribution. Then, it can be shown that,

$$||U(\delta a)||_2 = ||\delta a||_2. \tag{5.16}$$

Consider the term $\frac{1}{2X}\sum_{k=1}^{X} R_{s,a^j}^j(a^k)$. Since L is the maximum eigenvalue, from Equation 5.14 (with a slight abuse of notation),

$$R = \delta a \cdot \nabla^2 Q \cdot \delta a = [U\delta a]^T \Lambda [U\delta a] = \sum_i \lambda_i [U\delta a]_i^2 \leq L||U\delta a||_2. \tag{5.17}$$

Therefore, from Equation 5.16:

$$R \leq L||U\delta a||_2 = L||\delta a||_2. \tag{5.18}$$

Thus,

$$\frac{1}{2X}\sum_k R \leq \frac{L}{2}\sum_k \frac{||\delta a||_2}{X} \leq \frac{L\epsilon}{2}. \tag{5.19}$$

□

Thus, in this chapter, we modify the mean field Q function to include a finite number of types that each have a corresponding mean field. The Q function then considers a finite number of interactions across types.

5.3.2 Mean Field Update

The mean action $\overline{a}_i^j$ represents the mean action of the neighbours of agent j belonging to type i. As in the paper by Yang et al. [303], the mean field Q function can be updated in a recurrent manner,

$$Q_{t+1}^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j) = (1-\alpha)Q_t^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j) + \alpha[r^j + \gamma v_t^j(s')] \tag{5.20}$$

where r^j is the reward obtained. The s and s' are the old and new states respectively. α_t is the learning rate. The value function $v_t^j(s')$ for agent j at time t is given by,

$$v_t^j(s') = \textstyle\sum_{a^j} \pi_t^j(a^j|s', \overline{a}_1^j, \ldots, \overline{a}_M^j)\, \mathbb{E}_{a^{-j}\sim\pi_t^{-j}}[Q_t^j(s', a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j)] \tag{5.21}$$

Here, the term $\overline{a}_i^j$ denotes the mean action of all the other agents apart from j belonging to type i. In all of our implementations, the mean action for all the types is first calculated using the relation

$$\overline{a}_i^j = \frac{1}{N_i^j}\sum_k a_i^k, a_i^k \sim \pi_t^k(\cdot|s, \overline{a}_{1-}^k, \ldots, \overline{a}_{M-}^k) \tag{5.22}$$

where π_t^k is the policy of agent k (in j's neighbourhood) and $\overline{a}_{i-}^k$ represents the previous mean action for the neighbours of agent k belonging to type i. N_i^j is the total number of agents of type i in j's neighbourhood. Then, the Boltzmann policy for each agent j is

$$\pi_t^j(a^j|s, \overline{a}_1^j, \ldots, \overline{a}_M^j) = \frac{exp(\beta Q_t^j(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j))}{\sum_{a^{j\prime}\in A^j} exp(\beta Q_t^j(s, a^{j\prime}, \overline{a}_1^j, \ldots, \overline{a}_M^j))} \tag{5.23}$$

where β is the Boltzmann softmax parameter.

By iterating through Equations 5.21, 5.22 and 5.23, the mean actions and respective policies of all agents keep improving. We prove in Theorem 11 that this approach converges to a fixed point within a small bounded distance of the Nash equilibrium. In Appendix C.1, we give a specific example for a case in which the Multi Type Mean Field algorithm does better than the simple mean field method.

We first make three mild assumptions about the Multi Type Mean Field update and then state a lemma and prove a theorem for stochastic processes before giving Theorem 11. These assumptions are Assumption 1, 2, and 5 already specified in Chapter 3. For the rest of the section, we provide the proofs for all our theorems here, while the poofs of the lemmas used in these theorems can be found in Appendix C.3.

Lemma 26. *Under Assumption 5, the Nash operator $\mathscr{H}^{Nash}$ forms a contraction mapping under the complete metric space from $\mathcal{Q}$ to $\mathcal{Q}$ with the fixed point being the Nash Q value of the entire game ($\boldsymbol{Q}_*$), i.e., $\mathscr{H}^{Nash}\boldsymbol{Q}_* = \boldsymbol{Q}_*$.*

Proof. Refer to Theorem 17 in [104] for a detailed proof. □

We define a new operator $\mathscr{H}^{MTMF}$ which is the Multi Type Mean Field operator. We differentiate this from the Nash operator used above. This operator is defined as (where $\boldsymbol{Q} \triangleq [Q^1, \dots, Q^N]$),

$$\mathscr{H}^{MTMF}\boldsymbol{Q}(s, \mathbf{a}) \triangleq \mathbb{E}_{s' \sim p}[\mathbf{r}(s, \mathbf{a}) + \gamma \mathbf{v}^{MTMF}(s')]. \tag{5.24}$$

The Multi Type Mean Field value function can be defined as $\mathbf{v}^{MTMF}(s) \triangleq [v^1(s), \dots, v^N(s)]$. This is the value function obtained from Equation 5.21. Also, $\boldsymbol{r}(s, \boldsymbol{a}) = [r^1(s, \boldsymbol{a}), \dots, r^N(s, \boldsymbol{a})]$. Now using the same principle of Lemma 26 on the Multi Type Mean Field operator, we can show that the Multi Type Mean Field operator also forms a contraction mapping (additionally refer to Proposition 1 in Yang et al. [303]).

Theorem 10. *The random process Δ_t defined in $\mathcal{R}$ as*

$$\Delta_{t+1}(x) = (1 - \alpha)\Delta_t(x) + \alpha F_t(x) \tag{5.25}$$

converges to a constant S with probability 1 (w.p.t 1) when

$$1) \qquad 0 \le \alpha \le 1, \textstyle\sum_t \alpha = \infty, \sum_t \alpha^2 < \infty \tag{5.26}$$

$$2) \qquad x \in \mathscr{X}; |\mathscr{X}| < \infty \tag{5.27}$$

where $\mathscr{X}$ is the set of possible states,

$$3) \qquad ||\,\mathbb{E}[F_t(x)|\mathscr{F}_t]||_w \le \gamma||\Delta_t||_w + K \tag{5.28}$$

where $\gamma \in [0, 1)$ and K is finite.

$$4) \qquad \boldsymbol{var}[F_t(x)|\mathscr{F}_t] \le K_2(1 + ||\Delta_t||_w^2) \tag{5.29}$$

with constant $K_2 > 0$ and finite.

Here $\mathscr{F}_t$ denotes the filtration of an increasing sequence of σ-fields including the history of processes; α_t, Δ_t, $F_t \in \mathscr{F}_t$ and $||\cdot||_w$ is a weighted maximum norm. The value of this constant $S = \frac{\psi C_1 + \alpha|K|}{\alpha\beta_0}$ where $\psi \in (0, 1)$ and C_1 is the value with which the scale invariant iterative process is bounded. β_0 is the scale factor applied to the original process.

Proof. This lemma follows from Theorem 1 in [111].

First, we state a result pertaining to stochastic processes, and then we proceed to use this in the proof.

Lemma 27. *If we have a stochastic process of the form*

$$y_{n+1} - py_n = d \tag{5.30}$$

where n goes from 0 to ∞, then the general solution of this process can be given by

$$\begin{aligned} &\textit{if } p = 1: \\ &y_n = dn + a; \\ &\textit{if } p \neq 1: \\ &y_n = \tfrac{d}{1-p} + (a - \tfrac{d}{1-p})p^n \end{aligned} \tag{5.31}$$

where $y_0 = a$.

Now from Lemma 27, notice that if we want a general stochastic process of that form to converge we need the coefficient p to be a fraction (as we take a limit to ∞ in the solution only a p which is a fraction will give a converged result). Note that this result only needs d to be finite, and it can be any finite number. If we iterate to infinity then we can apply the limit to the solution which will converge to $y_n = \frac{d}{1-p}$.

Next, we introduce three lemmas that will build over successively and clarify ties to the literature in stochastic iterative techniques [111]. Once these three results are provided, the proof of the theorem will follow from them.

Lemma 28. *A random process*

$$w_{n+1}(x) = (1 - \alpha_n(x))w_n(x) + \beta_n(x)r_n(x) \tag{5.32}$$

converges to zero w.p.1, if the following conditions are satisfied:

1) $\sum_n \alpha_n(x) = \infty, \sum_n \alpha_n^2(x) < \infty,$

$\sum_n \beta_n(x) = \infty$ *and* $\sum_n \beta_n^2(x) < \infty$

uniformly over x w.p.1

2) $$\mathbb{E}\{r_n(x)|P_n, \beta_n\} = 0 \quad \text{and} \quad \mathbb{E}\{r_n^2(x)|P_n, \beta_n\} \leq C \tag{5.33}$$

w.p.1, where
$P_n = \{w_n, w_{n-1}, \ldots, r_{n-1}, r_{n-1}, \ldots,$
$\alpha_{n-1}, \alpha_{n-2}, \ldots, \beta_{n-1}, \beta_{n-2}, \ldots\}$

All the random variables are allowed to depend on the past P_n. $\alpha_n(x)$ and $\beta_n(x)$ are assumed to be non-negative and mutually independent given P_n.

Lemma 29. *Consider the stochastic iteration*

$$X_{n+1}(x) = G_n(X_n, Y_n, x) \tag{5.34}$$

where G_n is a sequence of functions and Y_n is a random process. Let $(\Omega, \mathscr{F}, \mathscr{P})$ be a probability space. If the following are satisfied:

1) The process is scale invariant. That is w.p.1 for all $\omega \in \Omega$

$$G(\beta X_n, Y_n(\omega), x) = \beta G(X_n, Y_n(\omega), x). \tag{5.35}$$

2) If we can keep $||X_n||$ bounded by scaling the process then X_n would converge to a constant D w.p.1 under condition 1.

The original process will converge to a constant $D_1 = \frac{D}{\beta_0}$ where β_0 is the scaling factor that was applied to $||X_n||$.

Lemma 30. *A stochastic process X_n which is bounded by the relation*

$$|X_{n+1}(x)| = (1-\alpha)|X_n(x)| + \gamma\beta C_1 + K \tag{5.36}$$

converges to a constant D w.p.1 provided

1) $x \in S$, where S is a finite set.

2) $\sum_n \alpha = \infty, \sum_n \alpha^2 < \infty$, $\sum_n \beta = \infty$, $\sum_n \beta^2 < \infty$, $\mathbb{E}\{\beta|P_n\} \leq E\{\alpha|P_n\}$, uniformly over x w.p.1,
where

$$P_n = \{w_n, w_{n-1}, \ldots, r_{n-1}, r_{n-1}, \ldots, \alpha, \beta\}$$

and α, β and γ are assumed to be non negative.

3) K is finite.

4) $\gamma \in (0, 1)$

5) The original process X_n is scale invariant.

The convergence point of the original process will then be $D = \frac{K+\gamma\beta C_1}{\alpha\beta_0}$ where β_0 is the scaling factor applied to the original process.

Now, we ready to state the final result.

Lemma 31. *A random iterative process*

$$\Delta_{n+1}(x) = (1 - \alpha)\Delta_n(x) + \beta F_n(x)$$

converges to a constant D w.p.1 under the following conditions:

1) $x \in S$, where S is a finite set.

2) $\sum_n \alpha = \infty, \sum_n \alpha^2 < \infty$, and $\sum_n \beta = \infty$, $\sum_n \beta^2 < \infty$, and $\mathbb{E}\{\beta|P_n\} \leq E\{\alpha|P_n\}$, uniformly over x w.p.1.

3) $|| \, \mathbb{E}\{F_n(x)|P_n, \beta\}||_w \leq \gamma||\Delta_n||_w + K$, where $\gamma \in (0, 1)$ and K is finite.

*4) **var**$\{F_n(x)|P_n, \beta\} \leq C(1 + ||\Delta_n||_W)^2$, where C is some constant. Here*

$$P_n = \{X_n, X_{n-1}, \ldots, F_{n-1}, \ldots, \alpha, \beta\}$$

stands for the past at step n. $F_n(x)$ is allowed to depend on the past. α and β are assumed to be non negative. The notation $||.||$ refers to some weighted maximum norm.

The value of this constant $D = \frac{\psi C_1+\beta|K|}{\alpha\beta_0}$ where $\psi \in (0, 1)$ and C_1 is the constant with which the iterative process is bounded. β_0 is the scaling factor that was applied to the original process.

The Lemma 31 directly proves Theorem 10.

□

Theorem 11. *When updating $Q^j(s, a^j, \bar{a}^j_1, \ldots, \bar{a}^j_M)$ according to Equations 5.20, 5.21, 5.22 and 5.23, for all agents j, the multi-agent Q function will converge to a bounded distance of the Nash Q function under the Assumptions 1, 2 and 5, expressed as,*

$$\boldsymbol{Q}_*(s, \boldsymbol{a}) - \boldsymbol{Q}_t(s, \boldsymbol{a}) \leq D - S.$$

where $S = \frac{\psi C_1 + \alpha\gamma|D|}{\alpha\beta_0}$. Here $D = \frac{1}{2}L\epsilon$, from Theorem 9. The joint Nash Q function is denoted as $\boldsymbol{Q}_* = [Q^1_*, \ldots, Q^N_*]$, where Q^j_* denotes the Nash Q-value of the agent j (value received by the agent j in a Nash equilibrium), and $\boldsymbol{Q}_t = [Q^1_t, \ldots, Q^N_t]$.

Proof. The proof of convergence of this theorem is structurally similar to that discussed by the authors in [104] and [303]. We provide the proof here, with changes necessitated by the multiple type case.

From [104], we formally define the Nash operator $\mathscr{H}^{Nash}$ as,

$$\mathscr{H}^{Nash}\boldsymbol{Q}(s, \boldsymbol{a}) \triangleq \mathbb{E}_{s' \sim p}[\boldsymbol{r}(s, \boldsymbol{a}) + \gamma \mathbf{v}^{Nash}(s')]. \tag{5.37}$$

where $\boldsymbol{Q} \triangleq [Q^1, \ldots, Q^N]$ and $\boldsymbol{r}(s, \boldsymbol{a}) \triangleq [r^1(s, \boldsymbol{a}), \ldots, r^N(s, \boldsymbol{a})]$.

The Nash value function is $\mathbf{v}^{Nash}(s) \triangleq [v^1_{\boldsymbol{\pi}_*}(s), \ldots, v^N_{\boldsymbol{\pi}_*}(s)]$. Here the joint Nash policy is represented as $\boldsymbol{\pi}_*$. The Nash value function is calculated with the assumption that all agents are following $\boldsymbol{\pi}_*$ from the initial state s.

We need to apply Theorem 10 to prove Theorem 11. By subtracting $Q_*(s, \boldsymbol{a})$ on both sides of Equation 5.20 and in relation to Equation 5.25,

$$\begin{aligned} \Delta_t(x) &= \mathbf{Q}_t(s, a^j, \bar{a}^j_1, \ldots, \bar{a}^j_M) - \mathbf{Q}_*(s, \boldsymbol{a}) \\ \mathbf{F}_t(x) &= \mathbf{r}_t + \gamma \mathbf{v}^{MTMF}_t(s_{t+1}) - \mathbf{Q}_*(s_t, \boldsymbol{a}_t) \end{aligned} \tag{5.38}$$

where $x \triangleq (s_t, \boldsymbol{a}_t)$ denotes the visited state-joint action pair at the time t.

In Theorem 9, we proved a bound for the actual Q function and the Multi Type Mean Field Q function. We apply that in Equation 5.38, to get the following equation for Δ,

$$
\begin{aligned}
&\Delta_t(x) = \mathbf{Q}_t(s, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M) - \mathbf{Q}_*(s, \boldsymbol{a}) \\
&\Delta_t(x) = \mathbf{Q}_t(s, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M) + \mathbf{Q}_t(s, \boldsymbol{a}) - \mathbf{Q}_t(s, \boldsymbol{a}) - \mathbf{Q}_*(s, \boldsymbol{a}) \\
&\Delta_t(x) \leq |\mathbf{Q}_t(s, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M) - \mathbf{Q}_t(s, \boldsymbol{a})| + \mathbf{Q}_t(s, \boldsymbol{a}) - \mathbf{Q}_*(s, \boldsymbol{a}) \\
&\Delta_t(x) \leq \mathbf{Q}_t(s, \boldsymbol{a}) - \mathbf{Q}_*(s, \boldsymbol{a}) + D
\end{aligned}
\tag{5.39}
$$

where $D = \frac{1}{2}L\epsilon$.

The aim is to prove that the four conditions of Theorem 10 hold and that Δ in Equation 5.39 converges to a constant S according to Theorem 10 and thus the MTMF Q function in Equation 5.39 converges to a point whose distance to the Nash Equilibrium is bounded. In Equation 5.25, $\alpha(t)$ refers to the learning rate and hence the first condition of Theorem 10 is automatically satisfied. The second condition is also true as we are dealing with finite state and action spaces.

Let $\mathscr{F}_t$ be the σ-field generated by all random variables in the history time t - $(s_t, \alpha_t, a_t, r_{t-1}, \ldots, s_1, \alpha_1, \mathbf{a}_1, \mathbf{Q}_0)$. Thus, $\mathbf{Q}_t$ is a random variable derived from the historical trajectory up to time t.

To prove the third condition of Theorem 10, from Equation 5.38,

$$
\begin{aligned}
&\boldsymbol{F}_t(s_t, \boldsymbol{a}_t) = \mathbf{r}_t + \gamma \mathbf{v}^{MTMF}_t - \mathbf{Q}_*(s_t, \boldsymbol{a}_t) \\
&= \mathbf{r}_t + \gamma \mathbf{v}^{Nash}_t(s_{t+1}) - \mathbf{Q}_*(s_t, \boldsymbol{a}_t) + \gamma[\mathbf{v}^{MTMF}_t(s_{t+1}) - \mathbf{v}^{Nash}_t(s_{t+1})] \\
&= (\mathbf{r}_t + \gamma \mathbf{v}^{Nash}_t(s_{t+1}) - Q_*(s_t, \boldsymbol{a}_t)) + C_t(s_t, \boldsymbol{a}_t) \triangleq \boldsymbol{F}^{Nash}_t(s_t, \boldsymbol{a}_t) + C_t(s_t, \boldsymbol{a}_t).
\end{aligned}
\tag{5.40}
$$

From Lemma 26, $\boldsymbol{F}^{Nash}_t$ forms a contraction mapping with the norm $||\cdot||_\infty$ being the maximum norm on $\boldsymbol{a}$. Thus, from Equation 5.39 we get,

$$
||\,\mathbb{E}[\boldsymbol{F}^{Nash}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t]||_\infty \leq \gamma ||\boldsymbol{Q}_* - \boldsymbol{Q}_t||_\infty \leq \gamma ||D - \Delta_t||_\infty. \tag{5.41}
$$

Now, applying Equation 5.41 in Equation 5.40,

$$
\begin{aligned}
&||\,\mathbb{E}[F_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t]||_\infty = ||F^{Nash}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t||_\infty + ||\boldsymbol{C}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t||_\infty \\
&\leq \gamma ||D - \Delta_t||_\infty + ||\boldsymbol{C}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t||_\infty \\
&\leq \gamma ||\Delta_t||_\infty + ||\boldsymbol{C}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t||_\infty + \gamma ||D||_\infty \leq \gamma ||\Delta_t||_\infty + \gamma |D|.
\end{aligned}
\tag{5.42}
$$

Since we are taking the max norm, the last two terms in the right-hand side of Equation 5.42 are both positive and finite. We can prove that the term $||C_t(s_t, \boldsymbol{a}_t)||$ converges to 0 w.p.1. The proof involves the use of Assumption 5 (refer to Theorem 1 in [303]). We use this fact in the last step of Equation 5.42. Hence, the third condition of Theorem 10 is satisfied. The value of constant $K = \gamma|D| = \gamma|\frac{1}{2}L\epsilon|$.

For the fourth condition we use the fact that the MTMF operator $\mathscr{H}^{MTMF}$ forms a contraction mapping. Hence, $\mathscr{H}^{MTMF}\mathbf{Q}_* = \mathbf{Q}_*$ and it follows that,

$$\begin{aligned}
&\mathbf{var}[\boldsymbol{F}_t(s_t, \boldsymbol{a}_t)|\mathscr{F}_t] = E[(r_t + \gamma \mathbf{v}_t^{MTMF}(s_{t+1}) - \boldsymbol{Q}_*(s_t, \boldsymbol{a}_t))^2] \\
&= E[(\boldsymbol{r}_t + \gamma \mathbf{v}_t^{MTMF}(s_{t+1}) - \mathscr{H}^{MTMF}(\boldsymbol{Q}_*))^2] \qquad (5.43) \\
&= \mathbf{var}[\boldsymbol{r}_t + \gamma \mathbf{v}_t^{MTMF}(s_{t+1})|\mathscr{F}_t] \leq K_2(1 + ||\Delta_t||_W^2).
\end{aligned}$$

In the last step, the left side of the equation contains the reward and the value function as the variables. The reward is bounded by Assumption 1 and the value function is also bounded by being updated recursively by Equation 5.21 (MTMF is a contraction operator). So we can choose a positive, finite K_2 such that the inequality holds.

Finally, with all conditions met, it follows from Theorem 10 that Δ_t converges to constant S w.p.1. The value of this constant is $S = \frac{\psi C_1 + \alpha\gamma|D|}{\alpha\beta_0}$ from Lemma 2 and using the value of K derived above. Therefore, from Equation 5.39 we get,

$$\mathbf{Q}_*(s, \boldsymbol{a}) - \mathbf{Q}_t(s, \boldsymbol{a}) \leq D - S \leq \tfrac{1}{2}L\epsilon - S. \qquad (5.44)$$

Hence, the multi-agent Q function converges to a point within a bounded distance from the real Nash equilibrium of the game. The distance is a function of the error in the type classification and the closeness of resemblance of each agent to its type. □

5.4 Implementation

We propose two algorithms based on Q-learning to estimate the Multi Type Mean Field Q function for known and unknown types. The algorithms, denoted as MTMFQ (Multi Type Mean Field Q-learning), train an agent j to minimize the loss function $\mathcal{L}(\phi^j) = (y^j - Q_{\phi^j}(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j))^2$. Here $y^j = r^j + \gamma v_{\phi_-^j}^{MTMF}(s')$ (from Eq. 5.20) is the target value used to calculate the temporal difference (T.D.) error using the weights ϕ_-^j. Now, the gradient is obtained as,

$$\nabla_{\phi^j}\mathcal{L}(\phi^j) = 2(Q_{\phi^j}(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j) - y^j) \times \nabla_{\phi^j} Q_{\phi^j}(s, a^j, \overline{a}_1^j, \ldots, \overline{a}_M^j). \tag{5.45}$$

Algorithm 9 describes the Multi Type Mean Field Q-learning (MTMFQ) algorithm when agent types are known. In this algorithm, different groups of agents in the environment are considered as types. An agent models its relation to each type separately and ultimately chooses the action that provides maximum benefit in the face of competition against the different types. This is referred to as version 1 of MTMFQ. This algorithm deals with multiple types in contrast to MFQ described in the paper by Yang et al. [303]. In Line 8, each agent is chosen, and its neighbours are considered. The neighbours are classified into different types and in each type a new mean action is calculated (Line 9). In Lines 14 – 19, the Q networks are updated as done in common practice [167] for all the agents in the environment. At Line 12, the current actions are added to a buffer containing previous mean actions.

Version 2 of MTMFQ (see Algorithm 10) deals with the second type of scenario, where the type of each agent is unknown. An additional step doing K-means clustering is introduced to determine the types. Once we recognize the type of each agent, the algorithm is very similar to Algorithm 9. The clustering does not necessarily recognize the types correctly and the overall process is not as efficient as the known type case. But we will show that this way of approximate type determination is still better than using only a single mean field for all the agents. For the implementation we use neural networks, but it can be done without neural networks too. We only need a way to recursively update Equations 5.21, 5.22 and 5.23.

Note that the total number of types in the environment is unknown, and the agent does not need to guess the correct number of types. Generally, when more types are used, the approximate multi-agent Q function will be closer to the exact Nash Q function as shown by the bounds in Theorems 7, 8, 9 and 11. There is no risk of overfitting when using more types. In the limit, when there is one type per agent, we recover the exact multi-agent Q function. The only drawback is an increase in computational complexity. The code for the experiments has been open sourced [244].

5.5 Experiments And Results

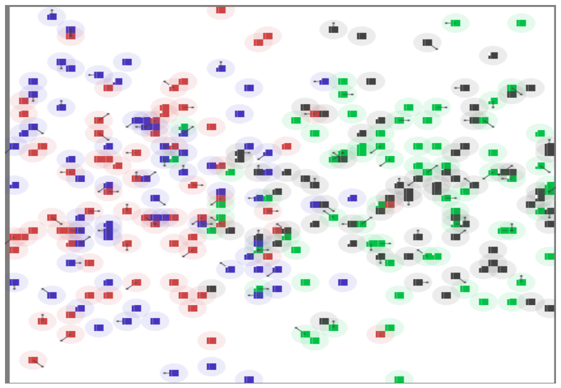
We report results with three games designed within the MAgents framework: the Multi-Team-Battle, Battle-Gathering and the Predator-Prey domains. In the first two games, Multi-Team-Battle and the Battle-Gathering game, the conditions are such that the different groups (or teams) are fully known upfront. In the third game, the conditions are such

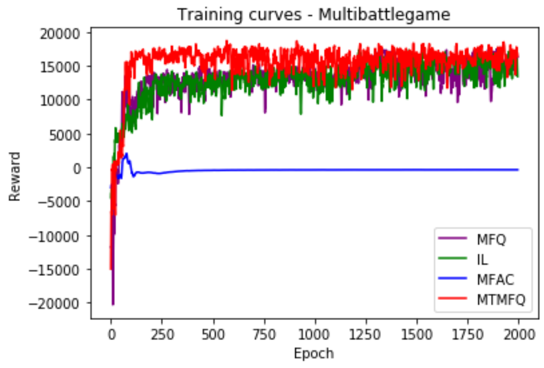
Algorithm 9 Multi Type Mean Field Q-learning for known types

```
1: Initialize the number of types $M$ and total number of agents $N$.
2: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, for all agents $j$.
3: Initialize the mean action for each type $\overline{a}^j_1$, $\overline{a}^j_2, \ldots, \overline{a}^j_M$, for each agent $j \in 1, \ldots, N$.
4: Initialize the total number of steps (T) and total number of episodes (E).
5: while Episode < E do
6:     while Step < T do
7:         For each agent $j$ choose action $a^j$ from $Q_{\phi^j}$ according to Eq. 5.23 with the current mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ and the exploration rate $\beta$.
8:         For each agent j, compute the new mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ according to Eq. 5.22.
9:         Execute the joint action $\mathbf{a} = [a^1, \ldots, a^N]$. Observe the rewards $\mathbf{r} = [r^1, \ldots, r^N]$ and the next state $s'$.
10:        Store $\langle s, \mathbf{a}, \mathbf{r}, s', \overline{\mathbf{a}_1}, \ldots, \overline{\mathbf{a}_M} \rangle$ in replay buffer $D$, where $\overline{\mathbf{a}_i}$ is the mean action for type $i$ in the neighbourhood. The $\mathbf{a}$ captures all the $N$ agents.
11:    end while
12:    while $j = 1$ to $N$ do
13:        Sample a minibatch of $K$ experiences $\langle s, \mathbf{a}, \mathbf{r}, s', \overline{\mathbf{a}_1}, \ldots, \overline{\mathbf{a}_M} \rangle$ from $D$.
14:        Sample action $a^j_-$ from $Q_{\phi^j_-}$ with $\overline{a}^j_{i\_} \leftarrow \overline{a}^j_i$ for each type $i$.
15:        Set $y^j = r^j + \gamma v^{MTMF}_{\phi^j_-}(s')$ according to Eq. 5.20.
16:        Update the Q network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M))^2$.
17:    end while
18:    Update the parameters of the target network for each agent $j$ with learning rate $\tau$; $\phi^j_- \leftarrow \tau \phi^j + (1 - \tau) \phi^j_-$.
19: end while
```

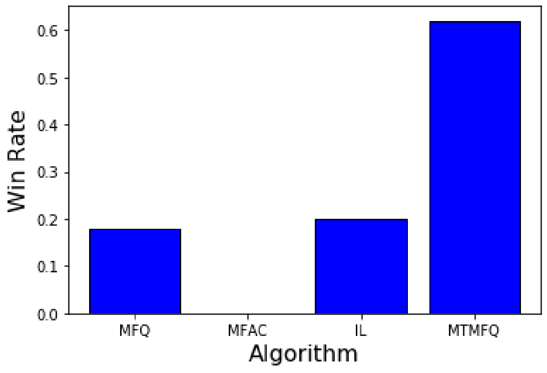
Algorithm 10 Multi Type Mean Field Q-learning for unknown types

```
1: Initialize the number of types $M$ and total number of agents $N$.
2: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, for all agents $j$.
3: Initialize the mean action for each type $\overline{a}^j_1, \overline{a}^j_2, \ldots, \overline{a}^j_M$, for each agent $j \in 1, \ldots, N$.
4: Initialize the total number of steps (T) and total number of episodes (E).
5: Initialize every agent to a type at random. Initialize an array $A$ containing the previous action of all agents.
6: Maintain a buffer $B$ for storing the last $C$ actions of all agents. $C$ is determined by the conditions of the environment. Initialize all values to 0.
7: while Episode < E do
8:     while steps < T do
9:         For each agent $j$ choose action $a^j$ from $Q_{\phi^j}$ according to Eq. 5.23 with the current mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ and the exploration rate $\beta$.
10:        For each agent j, compute the new mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ according to Eq. 5.22.
11:        Execute the joint action $\mathbf{a} = [a^1, \ldots, a^N]$. Observe the rewards $\mathbf{r} = [r^1, \ldots, r^N]$ and the next state $s'$.
12:        Store $\langle s, \mathbf{a}, \mathbf{r}, s', \overline{\mathbf{a}_1}, \ldots, \overline{\mathbf{a}_M} \rangle$ in replay buffer $D$, where $\overline{\mathbf{a}_i}$ is the mean action for type $i$ in the neighbourhood. The $\mathbf{a}$ captures all the $N$ agents.
13:        Store each action $[a^1, \ldots, a^N]$ in the array $A$. Update the Buffer $B$ with the last action taken by all agents.
14:        Perform a K-means clustering on $B$ with the number of clusters equal to the number of types $M$.
15:        Reassign the agents to different types based on the cluster in K-means.
16:    end while
17:    while $j = 1$ to $N$ do
18:        Sample a minibatch of $K$ experiences $\langle s, \mathbf{a}, \mathbf{r}, s', \overline{\mathbf{a}_1}, \ldots, \overline{\mathbf{a}_M} \rangle$ from $D$.
19:        Sample action $a^j_-$ from $Q_{\phi^j_-}$ with $\overline{a}^j_{i\_} \leftarrow \overline{a}^j_i$ for each type $i$.
20:        Set $y^j = r^j + \gamma v^{MTMF}_{\phi^j_-}(s')$ according to Eq. 5.20.
21:        Update the Q network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M))^2$.
22:    end while
23:    Update the parameters of the target network for each agent $j$ with learning rate $\tau$; $\phi^j_- \leftarrow \tau\phi^j + (1-\tau)\phi^j_-$.
24: end while
```

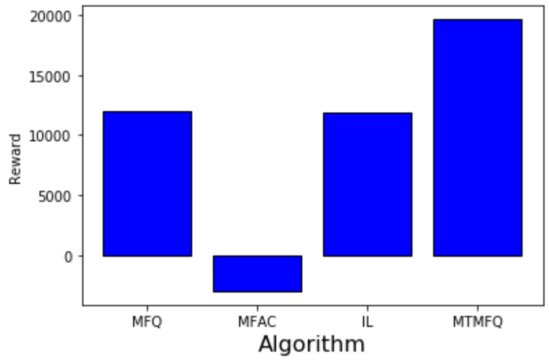

(a) Game domain: Multi-Team-Battle



(b) Multi-Team-Battle training



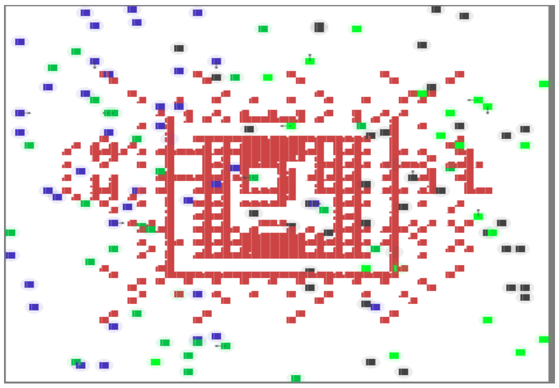
(c) Win rate for each algorithm

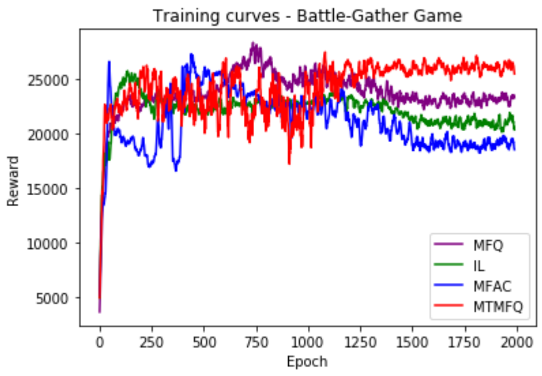


(d) Total rewards taken as an average per episode

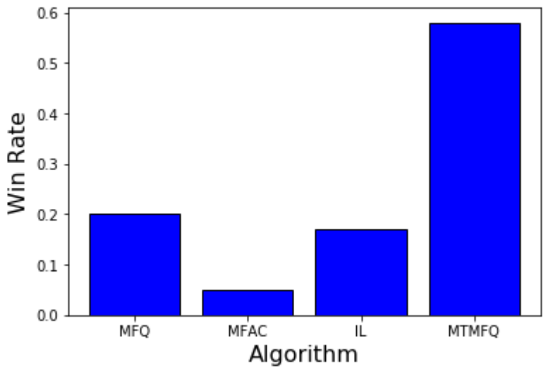
Figure 5.1: Multi-Team-Battle Game Results.

that the types of the agents are initially unknown. Hence, the agents must also learn the identity of the opponent agents during game play. Multi-Team-Battle and Gathering are two separately existing MAgent games, which have been combined to obtain the Battle-Gathering game used in this chapter. Predator-Prey domain is also obtained from combining Multi-Team-Battle with another existing MAgent game (Pursuit). In preparation for each game, agents train for 2000 episodes of game play against different groups training using the same algorithm, which is referred to as the first stage. Next, in the second stage, they enter into a faceoff against other agents trained by other algorithms where they fight each other for 1000 games. We report the results for both stages. We repeat all experiments 50 times and report the averages. As can be seen from the nature of the experiments, variances can be quite large across individual experimental runs.

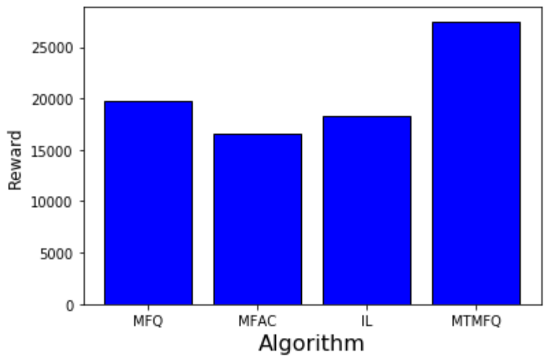

(a) Game domain: Battle-Gathering



(b) Battle-Gathering training



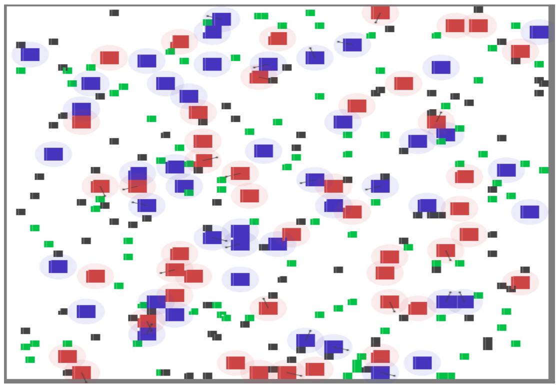
(c) Win rate for each algorithm

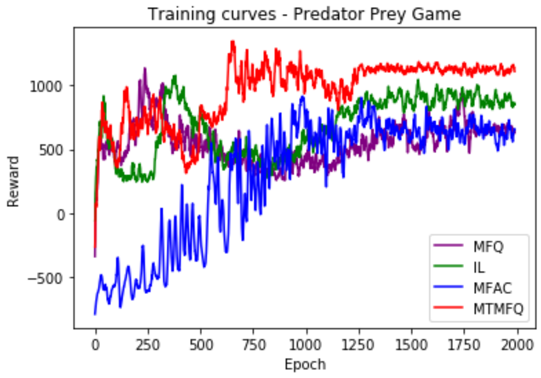


(d) Total Rewards taken as an average per episode.

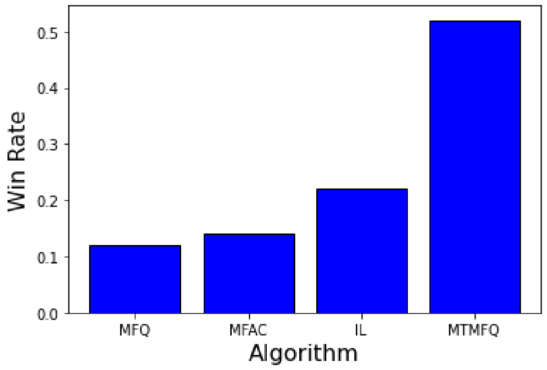
Figure 5.2: Battle-Gathering Game Results.

In the first game (Multi-Team-Battle, see Figure 5.1(a)), there are four teams (different colours in Figure 5.1(a)) fighting against each other to win a battle. Here, agents belonging to one team are competing against agents from other teams and cooperating with agents of the same team. During the first stage, a team does not know how the other teams will play during the faceoff — so it clones itself to three additional teams with slightly different rewards in order to induce different strategies. This is similar to self play. Each team receives a reward equal to the sum of the local rewards attributed to each agent in the team. The reward function is defined in such a way that it encourages local cooperation in killing opponents and discourages getting attacked and dying in the game. The reward functions for different agent groups are also subtly different (refer to Appendix C.2 for the details). Let us call each of these groups: Group A, Group B, Group C, and Group D. We maintain

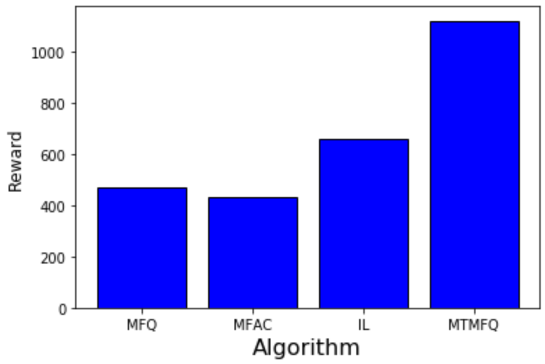

(a) Game domain: Predator-Prey



(b) Predator-Prey training



(c) Win rate for each algorithm



(d) Total rewards taken as an average per episode

Figure 5.3: Predator-Prey Game Results.

a notion of favourable opponents and each group gets slightly higher rewards for killing the favourable opponents than the others. Four algorithms, namely MFQ [303], MFAC [303], Independent Q Learning (IL) [264], and MTMFQ are trained separately where each of these groups trains its own separate network using the same algorithm (all agents within a group train the same network). We start all the battles with 72 agents of each group for training and testing. Group A from MTMFQ, Group B from IL, Group C from MFQ, and Group D from MFAC enter the faceoff stage where they fight each other for 1000 games. Each game (or episode) has a maximum of 5000 steps and a game is considered to be won by the group/groups that have the most number of agents alive at the end of the game.

The results of the first training stage are reported in Figure 5.1(a). We report the cumulative rewards from all agents in Group A for each algorithm. These rewards are for

Group A against the other 3 groups. Since the different groups are just clones of each other the reward curves for other groups are similar to that of Group A. In the training stage, the teams trained by different algorithms did not play against each other, but simply against the cloned teams trained by the same algorithm. At the beginning of training, for about 100 episodes the agents are still exploring, and their strategies are not well differentiated yet. As a result, MTMFQ's performance is still comparable to the performance of other algorithms. At this stage, the assumption of a single type is fine. As training progresses, each group begins to identify the favourable opponents and tries to make a targeted approach in the battle. When such differences exist across a wide range of agents, the MTMFQ algorithm shows a better performance than the other techniques as it explicitly considers the presence of different types in the game. Overall, we observe that MTMFQ has a faster convergence than all other algorithms, and it also produces higher rewards at the end of the complete training. This shows that MTMFQ identifies favourable opponents early, but the other algorithms struggle longer to learn this condition. The MFAC algorithm is the worst overall. This is consistent with the observation by Yang et al. [303], where the authors give particular reasons for this bad performance, including the greedy in the limit with infinite exploration (GLIE) assumption and a positive bias of Q-learning algorithms. This algorithm is not able to earn an overall positive reward upon the complete training. Figure 5.1(c) shows the win rate of the teams trained by each algorithm (MTMFQ, MFQ, MFAC, and IL) in a direct faceoff of 1000 episodes. In the face off, each group is trained by a different algorithm and with a different reward function that induces different strategies. The only algorithm that handles different strategies among the opponent teams is MTMFQ, and therefore it dominates the other algorithms with a win rate of 60%. Figure 5.1(d) reinforces this domination.

The second domain is the Battle-Gathering game (Figure 5.2(a)). In this game, all the agent groups compete for food resources that are limited (red denotes food particles and other colours are competing agents) in addition to killing its opponents as in the Multi-Team-Battle game. Hence, this game is harder than the first one. All the training and competition are similar to the first game.

Figure 5.2(b) reports the results of training in the Battle-Gathering game. Like the Multi-Team-Battle game, we plot the rewards obtained by Group A while fighting other groups for each algorithm. Again, MTMFQ shows the strongest performance in comparison to the other three algorithms. The MFQ technique performs better than both MFAC and IL. In this game, MTMFQ converges in around 1500 episodes, while the other algorithms take around 1800 episodes to converge. The win rates shown in Figure 5.2(c) and the total rewards reported in Figure 5.2(d) also show the dominance of MTMFQ.

The third domain is the multi-agent Predator-Prey (Figure 5.3(a)). Here we have two distinct types of agents — predator and prey, each having completely different characteristics. The prey are faster than the predators and their ultimate goal is to escape the predators. The predators are slower than the prey, but they have higher attacking abilities than the prey. So the predators try to attack more and kill more prey. We model this game as an unknown type scenario where the types of the other agents in the competition are not known before hand (refer to Appendix C.2 for more details). The MTMFQ algorithm plays the version with unknown types (Algorithm 10). Here we have four groups with the first two groups (Groups A and B) being predators and the other two groups (Groups C and D) being prey. Each algorithm will train two kinds of predator agents and two kinds of prey agents. All these agents are used in the playoff stage. In the playoff stage we have 800 games where we change the algorithm of predator and prey at every 200 games to maintain a fair competition. For the first 200 games, MTMFQ plays Group A, MFQ plays Group B, IL plays Group C, and MFAC plays Group D. In the next 200 games, MFAC plays Group A, MTMFQ plays Group B, MFQ plays Group C and IL plays Group D, and so on. We start all training and testing episodes with 90 prey and 45 predators for each group. Winning a game in the playoff stage is defined in the same way as the previous two games. Notice that this makes it more fair, as predators have to kill a lot more prey to win the game (as we start with more prey than predators) and prey have to survive longer. In this setup, the highly different types of agents make type identification easier for MTMFQ (as the types are initially unknown). The prey execute more move actions while the predators execute more attack actions. This can be well differentiated by clustering.

The results of the first training stage are reported in Figure 5.3(b). MTMFQ has comparable or even weaker performance than other algorithms in the first 600 episodes of the training and the reasoning is similar to the reasoning in the Multi-Team-Battle game (the agent strategies are not sufficiently differentiated for multiple types to be useful). Notice that for this game, MTMFQ takes many more episodes than the earlier two games to start dominating. This is because of the inherent hardness of this domain compared to the other domains (very different and unknown types). Similar to observations in the other domains, MTMFQ converges earlier (after around 1300 episodes as opposed to 1700 for the other algorithms). This shows its robustness to the different kinds of opponents in the environment. MTMFQ also gains higher cumulative rewards than the other algorithms. Win rates in Figure 5.3(c) show that MTMFQ still wins more games than the other algorithms, but the overall percentage of games won is less than the other domains. Thus, irrespective of the difficulty of the challenge we can see that MTMFQ has an upper hand. The lead of MTMFQ is also observed in Figure 5.3(d).

5.6 Conclusion

In this chapter, we extended the notion of mean field theory to multiple types in MARL. We demonstrate that reducing many agent interactions to simple two agent interactions does not give very accurate solutions in environments where there are clearly different teams/types playing different strategies. We perform suitable experiments using MAgents and demonstrate superior performances using a type based approach. We hope that this chapter will provide a different dimension to the mean field theory based MARL research.

One limitation of our approach is that it is computationally more expensive than the MFRL method without types. If we really have only one type in the environment, then our method would add more compute time and not necessarily produce a better result.

Chapter 6

Partially Observable Mean Field Reinforcement Learning

As discussed previously, traditional multi-agent reinforcement learning algorithms are not scalable to environments with more than a few agents, since these algorithms are exponential in the number of agents. Recent research has introduced successful methods to scale multi-agent reinforcement learning algorithms to many agent scenarios using mean field theory. However, almost all previous work in the field of mean field learning assume that an agent has access to exact cumulative metrics regarding the mean field behaviour of the system, which it can then use to take its actions. In this chapter, we relax this assumption and maintain a distribution to model the uncertainty regarding the mean field of the system. We consider two different settings for this problem. In the first setting, only agents in a fixed neighbourhood are visible, while in the second setting, the visibility of agents is determined at random based on distances. For each of these settings, we introduce a Q-learning based algorithm that can learn effectively. We prove that this Q-learning estimate stays very close to the Nash Q-value (under a common set of assumptions) for the first setting. We also empirically show our algorithms outperform multiple baselines in three different games in the MAgents framework, which supports large environments with many agents learning simultaneously to achieve possibly distinct goals.

The contributions of this chapter are summarized as follows:

- Principled approach to learning in mean field environments where global information about other agent policies are not available.
- Two different settings for partially observable mean field studies:

  - Fixed Observation Radius (FOR): This setting considers environments with agents having full observation within a fixed radius from itself (and no observation outside this fixed radius).
  - Probabilistic Distance-based Observability (PDO): This setting considers environments where each agent observes different other agents at random based on relative distances.

- Theoretical fixed point guarantees for a tabular Q-learning based mean field reinforcement learning algorithm in partially observable settings.

- Experimental illustrations of performances of the provided mean field algorithms in partially observable many agent environments, along with demonstrated superior performances as compared to standard baselines.

The core contents of this chapter have been published as a full paper in AAMAS-2021 [252] and is available on arXiv [250].

6.1 Introduction

Mean field theory has been used to scale MARL to many agent scenarios in previous research efforts [301, 303], most of which have assumed some notion of aggregation that is made available by an engine or is directly observable in the environment. For example, Guo et al. [88] assume that a population distribution parameter can be obtained from the game engine and Yang et al. [303] assume that the mean action of all agents in the environment can be observed directly by all agents.

Partial observability is an important research area in single agent reinforcement learning (RL) [93, 124, 320], but these advances are not applicable to the many agent RL paradigm, since the stationary environment assumption is broken. Also, partial observation in single agent RL corresponds only to partial observability of state features, but in many agent systems this could also correspond to partial observability of other agents.

This chapter relaxes the assumption that agents observe the aggregate state variable in a mean field update. Instead, we maintain a belief over the aggregate parameter that is used to help agent action selection. We focus on discrete action space Markov decision processes (MDPs) and modify the update rules from Yang et al. [303] to relax the assumptions of (1) global state availability and (2) exact mean action information for all agents. We consider two settings in this chapter. The *Fixed Observation Radius (FOR)* setting assumes that all agents in each agent's small field of view are always observed (and those outside the radius are not). The *Probabilistic Distance-based Observability (PDO)* setting relaxes FOR such that we model the probability of an agent seeing another agent as a function of the distance between them (and this distribution defines what agents are "viewable"). We introduce a new Q-learning algorithm for both settings, addressing the **Partially Observable Mean Field (POMF) Q-learning** problem, using Bayesian updates to maintain a distribution over the mean action parameter.

This chapter's contributions are to (1) introduce two novel POMF settings, (2) introduce two novel algorithms for these settings, (3) prove that the first algorithm ends up close to the Nash Q-value [104], and (4) empirically show that both algorithms outperform existing baselines in three complex, large-scale tasks. We will assume stationary strategies as do other related previous work [104, 303].

6.2 Background

This chapter will intersect the areas of RL, stochastic games and MFRL. We will provide a brief introduction to MFRL from Yang et al. [303] and highlight some limitations of this method in the context of the current chapter (as opposed to our discussions in Chapter 5).

Mean field reinforcement learning extends the stochastic game framework to environments where the number of agents are very large (possibly infinite in the limit) [136]. All agents are assumed to be homogeneous and independent. In this case, all the agents in the environment can be approximated as a single virtual agent to which the learning agent (called the *central agent*) formulates best response strategies. Yang et al. [303] approximates the multi-agent Q-function by the mean field Q-function (MFQ) using an additive decomposition and Taylor's expansion (Eq. 6.1).

$$Q^j(s_t, \mathbf{a}_t) \approx Q^j(s_t, a^j_t, \overline{a}^j_t) \tag{6.1}$$

The MFQ is recurrently updated using Eqs. 6.2 – 6.5:

$$Q^j(s_t, a^j_t, \overline{a}^j_t) = (1-\alpha)Q^j(s_t, a^j_t, \overline{a}^j_t) + \alpha[r^j_t + \gamma v^j(s_{t+1})] \tag{6.2}$$

$$\text{where } v^j(s_{t+1}) = \sum_{a^j_{t+1}} \pi^j(a^j_{t+1}|s_{t+1}, \overline{a}^j_t) Q^j(s_{t+1}, a^j_{t+1}, \overline{a}^j_t) \tag{6.3}$$

$$\overline{a}^j_t = \frac{1}{N^j} \sum_{k \neq j} a^k_t, a^k_t \sim \pi^k(\cdot|s_t, \overline{a}^k_{t-1}) \tag{6.4}$$

$$\text{and } \pi^j(a^j_t|s_t, \overline{a}^j_{t-1}) = \frac{\exp(-\beta Q^j(s_t, a^j_t, \overline{a}^j_{t-1}))}{\sum_{a^{j'}_t \in A^j} \exp(-\beta Q^j(s_t, a^{j'}_t, \overline{a}^j_{t-1}))} \tag{6.5}$$

r^j_t is the reward for agent j, s_t is the (global) old state, s_{t+1} is the (global) resulting state, α is the learning rate, v^j is the value function of j and β is the Boltzmann parameter. a^j_t denotes the (discrete) action of agent j represented as a one-hot encoding whose components are one of the actions in the action space. The $\overline{a}^j_t$ is the mean action of all other agents apart from j and π denotes the Boltzmann policy. Finally, $\gamma \in [0, 1)$ denotes the discount factor. In Eq. 6.3, there is no expectation over $\overline{a}^j$, because Yang et al. [303] guaranteed that the MFQ updates will be greedy in the limit ($t \to \infty$). Finally, N^j is the number of agents in the neighbourhood of j. Further, Yang et al. [303] use the Nash equilibrium (NE) as the solution concept and prove that the MFQ updates results in the Q-values converging to the Nash Q-value of the stochastic game in the time limit ($t \to \infty$). We highlight that, for the mean action calculation in Eq. 6.4, the policies of all other agents needs to be maintained by the central agent. Now, this policy can only be obtained by observing all other agents at every time step, which is a strong assumption in a large environment with many agents. Yang et al., overcome this problem by introducing *neighbourhoods*. However, typically, practical real-world many-agent settings are dynamic environments (i.e., environments in which agents dynamically move around in the domain). In such settings, the neighbourhood needs to be large enough to contain the whole environment for these methods to work, as agents can go in and out of the neighbourhoods and go out of vicinity otherwise, which will make the computation of mean action as in Eq. 6.4 inapplicable (particularly since NE is used as the solution concept which requires accurate global observation). In our work, we will relax this strong assumption in estimating the mean field action. We will not assume the observability of all other agents.

6.3 Partially Observable Mean Field Q-Learning: FOR

In this section, we study the Fixed Observation Radius (FOR) version of our problem, where all agents within a fixed neighbourhood from the central agent are visible to the central agent, and the others are not visible. Our setting is same as that in Yang et al. [303]

but we proceed to relax the assumption of global state observability. We modify the update in Eqs. 6.2 – 6.5 by maintaining a categorical distribution for the mean action parameter $(\overline{a})$. We will only use the local state s^j of agent j and not the global state. Eq. 6.6 gives our corresponding Q update equation. Since the conjugate prior of a categorical distribution is the Dirichlet distribution, we use a Dirichlet prior for this parameter. Let L be the size of the action space. Let η denote the parameters of the Dirichlet $(\eta_1, \ldots, \eta_L)$, θ denote a categorical distribution $(\theta_1, \ldots, \theta_L)$, and $\mathcal{X}$ denote an observed action sample $(x_1, \ldots, x_G)$ of G agents. Then the Dirichlet for agent j can be given by $\mathcal{D}^j(\theta|\eta) \propto \theta_1^{\eta_1 - 1} \cdots \theta_L^{\eta_L - 1}$ and the likelihood is given by $p(\mathcal{X}|\theta) \propto \theta_1^{[\mathcal{X}=1]} \cdots \theta_L^{[\mathcal{X}=L]} \propto \theta_1^{c_1} \cdots \theta_L^{c_L}$, where $[X = i]$ is the Iverson bracket, which evaluates to 1 if $X = i$ and 0 otherwise. This value corresponds to the number of occurrences of each category $(c_1, \ldots, c_L)$, denoted by c. Using a Bayesian update, the posterior is a Dirichlet distribution given by Eq. 6.8 where the parameters of this Dirichlet are given by $\mathcal{D}^j(\theta|\eta + c)$.

The modified Q updates are:

$$Q^j(s_t^j, a_t^j, \tilde{a}_t^j) = (1-\alpha)Q^j(s_t^j, a_t^j, \tilde{a}_t^j) + \alpha[r_t^j + \gamma v(s_{t+1}^j)] \tag{6.6}$$

$$\tag{6.7}$$

$$\mathcal{D}^j(\theta) \propto \theta_1^{\eta_1 - 1 + c_1} \cdots \theta_L^{\eta_L - 1 + c_L}; \quad \mathcal{D}^j(\theta|\eta + c) \tag{6.8}$$

$$\text{Where } v^j(s_{t+1}^j) = \sum_{a_{t+1}^j} \pi^j(a_{t+1}^j | s_{t+1}^j, \tilde{a}_t^j) Q^j(s_{t+1}^j, a_{t+1}^j, \tilde{a}_t^j) \tag{6.9}$$

$$\tilde{a}_{i,t}^j \sim \mathcal{D}^j(\theta; \eta + c); \quad \tilde{a}_t^j = \frac{1}{\mathcal{S}} \sum_{i=1}^{i=\mathcal{S}} \tilde{a}_{i,t}^j \tag{6.10}$$

$$\text{and } \pi^j(a_t^j | s_t^j, \tilde{a}_{t-1}^j) = \frac{\exp(-\beta Q^j(s_t^j, a_t^j, \tilde{a}_{t-1}^j))}{\sum_{a_t^{j'} \in A^j} \exp(-\beta Q^j(s_t^j, a_t^{j'}, \tilde{a}_{t-1}^j))} \tag{6.11}$$

We have replaced the mean field aggregation from Yang et al. (Eq. 6.4) with the Bayesian updates of the Dirichlet distribution from Eq. 6.8 and we take $\mathcal{S}$ samples from this distribution in Eq. 6.10 to estimate the partially observable mean action $(\tilde{a})$. This approach relaxes the assumption of complete observability of the global state. We use samples from the Dirichlet to introduce noise in the mean action parameter, enabling further exploration and helping agents to escape local optima. Being stuck in a local optimum is one of the major reasons for poor performance of learning algorithms in large scale systems. For example, Guo et al. [88] show that MFQ and Independent Q-learning (IL) [264] remain stuck at a local optimum and do not move towards a global optimum in many settings, even after many training episodes. Yang et al. also report that the Mean Field Actor Critic (MFAC) and MFQ algorithms may remain stuck at a local optimum for a long period of training episodes in a simple Gaussian squeeze environment as the number of agents becomes exponentially large. Intuitively, this problem is even worse in a partially observable setting as the agents get a smaller observation and their best response policy is directed towards this observation sample. Sampling methods as in Eq. 6.10 are also used in established algorithms like Thompson sampling [273, 185]. Finally, we update the Boltzmann policy like Yang et al. in Eq. 6.11. We provide more theoretical guarantees for our update equations in Section 6.7.

This version of our problem is generally applicable to many different environments. However, in some domains, agents may not be able to see all the other agents in the vicinity, but closer agents will have a high probability of being seen. The next section considers a new version of our problem where some special kinds of distributions are used to model the observed agents.

6.4 Partially Observable Mean Field Q-Learning: PDO

This section considers the Probabilistic Distance-based Observability (PDO) problem, assuming that each agent can observe other agents with some probability that decreases as distance increases.

We introduce a distance vector $\mathscr{D}$ that represents the distance of other agents in the environment to the central agent. Hence, $\mathscr{D} = (d_1, \ldots, d_N)$, where d_i denotes the distance of agent i from the central agent. We use the exponential distribution to model the probability of the distance of agent i from the central agent. Exponential distribution assigns a higher probability to smaller distances and this probability exponentially drops off as distance increases. Since, in a large environment the agents that matter are closer to the central agent, than far off, the exponential distribution is appropriate to model this variable. We drop the subscript of d for clarity. This distribution is parameterized by $\hat{\theta}$ so that $d|\hat{\theta} \propto \exp(\hat{\theta})$. Since the conjugate prior of the exponential distribution is the gamma distribution, we use a gamma prior, and the prior distribution is parameterised by α, β. Hence, we write $\hat{\theta} \propto Gamma(\hat{\alpha}, \hat{\beta})$.

We also maintain an additional parameter b_i that determines whether a given agent i is visible to the central agent j. The variable b_i takes two values: 1 if this agent is in the field of view and 0 if this agent is not in the field of view. Again, we will drop the subscript of b. We maintain a Bernoulli distribution conditioned on the distance d. The probability that an agent at a distance d is visible is given by $Pr(b = 1|d, \lambda) = \lambda e^{-d\lambda}$. Note that this is not an exponential distribution, but rather a Bernoulli distribution with a probability defined by the same algebraic formula as the exponential distribution. In this setting, we will assume that the central agent will see varying numbers of other agents based on this distribution. Since the parameter λ cannot be estimated by an agent from observation (the agent needs to know which other agents it is seeing and not seeing to infer λ), we will assume that the scalar value of λ is common knowledge for all the agents. Since $\lambda e^{-\lambda d}$ should be in $[0, 1]$ because it is a probability, only λ values in $[0, 1]$ satisfy this requirement. We will fix the value of λ to be 1, but it could be any other value in the given range. This gives a definite distribution that determines the conditional of b. We are particularly interested in the posterior term $Pr(\hat{\theta}|d, b = 1)$, which denotes the probability of $\hat{\theta}$, given the distance d of another agent i, and that i is in the field of view of the central agent j.

$$Pr(\hat{\theta}|d, b = 1) \propto Pr(d|\hat{\theta}, b = 1)Pr(\hat{\theta}|b = 1) \propto Pr(d|\hat{\theta}, b = 1)Pr(\hat{\theta}) \tag{6.12}$$

In the last term of Eq. 6.12, the variable θ does not depend on the variable b, so we remove the conditional. Now consider,

$$Pr(d|\hat{\theta}, b = 1) = Pr(d, b|\hat{\theta})/Pr(b|\hat{\theta}) \tag{6.13}$$

$$= Pr(b = 1|\hat{\theta}, d)Pr(d|\hat{\theta})/Pr(b = 1|\hat{\theta}) \tag{6.14}$$

$$= Pr(b = 1|d)Pr(d|\hat{\theta})/Pr(b = 1|\hat{\theta}) \tag{6.15}$$

$$= \lambda e^{-d}\lambda\hat{\theta}e^{-d\hat{\theta}}/\int_d Pr(b = 1|d)Pr(d|\hat{\theta}) \tag{6.16}$$

$$= e^{-d}\hat{\theta}e^{-d\hat{\theta}}/\int_{d=0}^{d=\infty}\hat{\theta}e^{-d\hat{\theta}}\lambda e^{-d\lambda} = e^{-d}\hat{\theta}e^{-d\hat{\theta}}/\int_{d=0}^{d=\infty}\hat{\theta}e^{-d(\hat{\theta}+1)} \tag{6.17}$$

$$= e^{-d}\hat{\theta}e^{-d\hat{\theta}}(\hat{\theta} + 1)/\hat{\theta} \tag{6.18}$$

Applying Eq. 6.18 in Eq. 6.12,

$$Pr(\hat{\theta}|d, b = 1) \propto Gamma(\hat{\alpha}, \hat{\beta}) \times e^{-d}\hat{\theta}e^{-d\hat{\theta}}(\hat{\theta} + 1)/\hat{\theta}$$

$$\propto \hat{\theta}^{\hat{\alpha}-1}e^{-\hat{\beta}\hat{\theta}} \times e^{-d}\hat{\theta}e^{-d\hat{\theta}}(\hat{\theta} + 1)/\hat{\theta}$$

$$\propto \hat{\theta}^{\hat{\alpha}}e^{-\hat{\theta}(\hat{\beta}+d-d/\hat{\theta})} + \hat{\theta}^{\hat{\alpha}-1}e^{-\hat{\theta}(\hat{\beta}+d-d/\hat{\theta})}$$

The posterior of $\hat{\theta}$ is therefore given by a mixture of Gamma distributions (i.e., $Gamma(\hat{\alpha} + 1, d + \hat{\beta} - d/\hat{\theta})$ and $Gamma(\hat{\alpha}, d + \hat{\beta} - d/\hat{\theta})$). We can obtain a single Gamma posterior, corresponding to a projection of this mixture of Gammas, by updating the new value of $\hat{\alpha}$ as $\hat{\alpha} + 0.5$. We sample from this single Gamma distribution to get a new parameter $\overline{\lambda}$ (Eq. 6.22). We denote the Gamma distribution for the agent j using superscript j (Eq. 6.21). Hence, we get:

$$Q^j(s^j_t, a^j_t, \tilde{a}^j_t, \overline{\lambda}^j_t) = (1 - \alpha)Q^j(s^j_t, a^j_t, \tilde{a}^j_t, \overline{\lambda}^j_t) + \alpha[r^j_t + \gamma v^j(s^j_{t+1})] \tag{6.19}$$

$$\tag{6.20}$$

$$\hat{\theta}_P \propto Gamma^j(\hat{\alpha} + 1, d + \hat{\beta} - d/\hat{\theta}) + Gamma^j(\hat{\alpha}, d + \hat{\beta} - d/\hat{\theta}) \tag{6.21}$$

Where:

$$\overline{\lambda}^j_{i,t} \sim [Gamma^j(\hat{\alpha} + 0.5, d + \hat{\beta} - d/\hat{\theta})]; \quad \overline{\lambda}^j_t = \frac{1}{\mathcal{G}}\sum_{i=1}^{i=\mathcal{G}} \overline{\lambda}^j_{i,t} \tag{6.22}$$

$$\tag{6.23}$$

$$v^j(s^j_{t+1}) = \sum_{a^j_{t+1}} \pi^j(a^j_{t+1}|s^j_{t+1}, \tilde{a}^j_t, \overline{\lambda}_t)Q^j(s^j_{t+1}, a^j_{t+1}, \tilde{a}^j_t, \overline{\lambda}^j_t) \tag{6.24}$$

$$\pi^j(a^j_t|s^j_t, \tilde{a}^j_{t-1}, \overline{\lambda}^j_{t-1}) = \frac{\exp(-\beta Q^j(s^j_t, a^j_t, \tilde{a}^j_{t-1}, \overline{\lambda}^j_{t-1}))}{\sum_{a^{j'}_t \in A^j} \exp(-\beta Q^j(s^j_t, a^{j'}_t, \tilde{a}^j_{t-1}, \overline{\lambda}^j_{t-1}))} \tag{6.25}$$

All the variables above have the same meaning as in Eqs. 6.6 – 6.11. The estimate of the $\tilde{a}$ is obtained as in Eq. 6.10. The $\hat{\theta}_P$ denotes the new (posterior) value of $\hat{\theta}$. The $\overline{\lambda}$ parameter is updated by sampling from the Gamma distribution, as in Eq. 6.22, by taking $\mathcal{G}$ samples.

6.5 Complexity Analysis

A tabular version of our algorithm is linear in the number of states, polynomial in the number of actions, and constant in the number of agents. The guarantees are similar to the paper from Hu and Wellman [104], except that their algorithm is exponential in the number of agents. The time complexity is also same as the space complexity as in the worst case, each entry in the table has to be accessed and updated.

Note that the approach by Yang et al. [303] has exponential space complexity in the number of agents, since each agent has to maintain the Q tables for every other agent to obtain the action of other agents in Eq. 6.3. This is much worse than our space complexity.

6.6 Algorithm Implementations

The implementation of POMFQ follows prior work [303] that uses neural networks — Q-functions are parameterized using weights ϕ, but tabular representations or other function approximators should also work. Our algorithms are an integration of the respective update equations with Deep Q-learning (DQN) [167]. Algorithm 11 gives pseudo code for the algorithm for the "FOR" case and Algorithm 12 for the "PDO" case. The lines in Algorithm 12 that have changed from Algorithm 11 are marked in blue.

6.7 Theoretical Results

The goal of this section is to show that our FOR Q-updates are guaranteed to converge to the Nash Q-value. We will begin by providing a technical result that is generally applicable for any stochastic processes of which the Q-function is a specific example. Then we have a sequence of theorems that lead us to bound the difference between the POMF Q-value and the Nash Q-value in the limit ($t \to \infty$). We outline a number of common assumptions that are needed to prove these theorems. For the purposes of a direct comparison of the POMF Q-function and the Nash Q-function, we assume that we have a system of N agents where agents have the full global state available and thus have the ability to perform a MFQ update (Eqs. 6.2 – 6.5) or a POMFQ update (Eqs. 6.6 – 6.11). By the definition of a Nash equilibrium, every agent should have the knowledge of every other agent's strategy. To recall, in a Nash equilibrium, no agent will have an incentive to unilaterally deviate, given the knowledge of other agent strategies. Our objective is also to make a direct comparison

Algorithm 11 Partially observable mean field Q Learning - FOR

```
1:  Initialize the weights of $Q$-functions $Q_{\phi^j}, Q_{\phi^j_-}$ for all agents $j \in 1, \ldots, N$.
2:  Initialize the Dirichlet parameter $\mathcal{D}^j(\theta)$ for all agents $j$.
3:  Initialize the mean action $\overline{a}^j$ for each agent $j \in 1, \ldots, N$.
4:  Initialize the total steps (T) and total episodes (E).
5:  while Episode < E do
6:      while Step < T do
7:          For each agent $j$, sample $a^j$ from the policy induced by $Q_{\phi^j}$ according to Eq. 6.11
            with the current mean action $\tilde{a}^j$ and the exploration rate $\beta$.
8:          For each agent $j$, update its Dirichlet distribution (Eq. 6.8).
9:          For each agent $j$, compute the new mean action $\tilde{a}^j$ (Eq. 6.10).
10:         Execute the joint action $\mathbf{a} = [a^1, \ldots, a^N]$. Observe the rewards $\mathbf{r} = [r^1, \ldots, r^N]$
            and the next state $\mathbf{s'} = [s'^1, \ldots, s'^N]$.
11:         Store $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s'}, \tilde{\mathbf{a}} \rangle$ in replay buffer $B$, where $\tilde{\mathbf{a}}$=$[\tilde{a}^1, \ldots, \tilde{a}^N]$ is the mean action.
12:     end while
13:     while $j = 1$ to $N$ do
14:         Sample a minibatch of K experiences $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s'}, \tilde{\mathbf{a}} \rangle$ from $B$.
15:         Set $y^j = r^j + \gamma v^{POMF}_{\phi^j_-}(s')$ according to Eq. 6.9.
16:         Update Q network by minimizing the loss $L(\phi^j)$=$\frac{1}{k}\sum(y^j - Q_{\phi^j}(s^j, a^j, \tilde{a}^j))^2$.
17:     end while
18:     Update params of target network for each agent $j$: $\phi^j_- \leftarrow \tau\phi^j + (1-\tau)\phi^j_-$.
19: end while
```

between the POMFQ update and the MFQ update and hence we will use the FOR setting algorithms of POMFQ update in the theoretical analysis as it is most directly related to MFQ. In this section, we will show that a representative agent j's Q-value will remain at least within a small distance of the Nash Q-value in the limit ($t \to \infty$) as it performs a POMFQ update, which tells us that, in the worst case, the agents stay very close to the Nash equilibrium. We provide the complete proofs for all our theorems in this section. The proofs of some theorems are broken into smaller lemmas (for simplicity), whose proofs can be found in Appendix D.1. In a mean field setting, the homogeneity of agents allows us to drop the agent index j [136] for the value and Q-function, which we adopt for clarity. Also, "w.p.1" represents "with probability one".

Consider an update equation of the following form (using the Tsitsiklis [276] formulation):

$$x_i(t+1) = x_i(t) + \alpha_i(t)(F_i(x^i(t)) - x_i(t) + w_i(t)) \tag{6.26}$$

Algorithm 12 Partially observable mean field Q Learning - PDO

```
1: Initialize the weights of $Q$ functions $Q_{\phi^j}, Q_{\phi^j_-}$ for all agents $j \in 1, \ldots, N$.
2: Initialize the Dirichlet parameter $\theta$ in $\mathcal{D}^j(\theta)$ for all agents $j$.
3: Initialize $\hat{\alpha}$ and $\hat{\beta}$ in $Gamma^j(\hat{\alpha}, \hat{\beta})$ for all agents $j \in 1, \ldots, N$.
4: Initialize the mean action $\overline{a}^j$ for each agent $j \in 1, \ldots, N$.
5: Initialize the total steps (T) and total episodes (E).
6: while Episode < E do
7:     while Step < T do
8:         For each agent $j$ sample $a^j$ from the policy induced by $Q_{\phi^j}$ according to Eq. 6.25
   with the current mean action $\tilde{a}^j$ and the exploration rate $\beta$.
9:         For each agent $j$ update its Dirichlet distribution (Eq. 6.8).
10:        For each agent $j$, update its Gamma distribution (Eq. 6.21).
11:        For each agent $j$, compute the new mean action $\tilde{a}^j$ (Eq. 6.10).
12:        For each agent j, update parameter $\overline{\lambda}$ (Eq. 6.22).
13:        Execute the joint action $\mathbf{a} = [a^1, \ldots, a^N]$. Observe the rewards $\mathbf{r} = [r^1, \ldots, r^N]$
   and the next state $\mathbf{s'} = [s'^1, \ldots, s'^N]$.
14:        Store $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s'}, \tilde{\mathbf{a}}, \overline{\boldsymbol{\lambda}} \rangle$ in replay buffer $B$, s.t. $\tilde{\mathbf{a}}$=$[\tilde{a}^1, ..., \tilde{a}^N]$, $\overline{\boldsymbol{\lambda}}$=$[\overline{\lambda}^1, ..., \overline{\lambda}^N]$
15:    end while
16:    while $j = 1$ to $N$ do
17:        Sample minibatch of K experiences $\langle \mathbf{s}, \mathbf{a}, \mathbf{r}, \mathbf{s'}, \tilde{\mathbf{a}}, \overline{\boldsymbol{\lambda}} \rangle$ from $B$.
18:        Set $y^j = r^j + \gamma v^{POMF}_{\phi^j_-}(s')$ according to Eq. 6.24.
19:        Update Q network by minimizing $L(\phi^j)$=$\frac{1}{k}\sum(y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j, \overline{\lambda}))^2$.
20:    end while
21:    Update params of target network for each agent $j$: $\phi^j_- \leftarrow \tau\phi^j + (1-\tau)\phi^j_-$.
22: end while
```

Here, $x(t)$ is the value of vector x at time t and $x_i(t)$ denotes its ith component. Let, F be a mapping from $\mathscr{R}^n$ into itself. Let $F_1, \ldots, F_n : \mathscr{R}^n \to \mathscr{R}$ be the component mappings of F, that is $F(x) = (F_1(x), \ldots, F_n(x))$ for all $x \in \mathscr{R}^n$. Also, $w_i(t)$ is a noise term, and $x^i(t)$ can be defined as $x^i(t) = (x_1(\tau^i_1(t)), \cdots, x_n(\tau^i_n(t)))$, where each $\tau^i_j(t)$ satisfies $0 \leq \tau^i_j(t) \leq t$.

Next, we state some assumptions. The first three are the same as those in Tsitsiklis [276], but we modify the fourth assumption. The first assumption guarantees that old information is eventually discarded with probability one. The second assumption is a measurability condition and the third assumption is the learning rate condition, both of which are common in RL [262, 276]. The Assumption 9, is a condition on the F mapping, which is a weaker version than the fourth assumption in Tsitsiklis [276].

Assumption 6. *For any i and j, $\lim_{t\to\infty} \tau^i_j(t) = \infty$ w.p.1.*

Assumption 7. *a) $x(0)$ is $\mathcal{F}(0)$-measurable*
b) For every i, j, and t, $w_i(t)$ is $\mathcal{F}(t+1)$-measurable
c) For every i, j, and t, $\alpha_i(t)$ and $\tau^i_j(t)$ are $\mathcal{F}(t)$-measurable
d) For every i and t, we have $\mathbb{E}[w_i(t)|\mathcal{F}(t)] = 0$
e) For deterministic constants A and B,

$$\mathbb{E}[w_i^2(t)|\mathcal{F}(t)] \leq A + Bmax_j max_{\tau\leq t}|x_j(\tau)|^2$$

Assumption 8. *The learning rates satisfy $0 \leq \alpha_i(t) < 1$.*

Assumption 9. *a) The mapping F is monotone; that is, if $x \leq y$, then $F(x) \leq F(y)$*
b) The mapping F is continuous
c) In the limit ($t \to \infty$), the mapping F is bounded in an interval $[x^* - D, x^* + D]$, where x^* is some arbitrary point
d) If $e \in \mathcal{R}^n$ is the vector with all components equal to 1, and p is a positive scalar then, $F(x) - pe \leq F(x - pe) \leq F(x + pe) \leq F(x) + pe$

Now, we will state our first theorem. Theorem 12 is a technical result that we obtain by extending Theorem 2 in Tsitsiklis [276]. We will use this result to derive the main result in Theorem 15.

Theorem 12. *A stochastic process of the form given in Eq. 6.26 remains bounded in the range $[x^* - 2D, x^* + 2D]$ in the limit, if Assumptions 6 – 9 hold, and if the process is guaranteed not to diverge to infinity. D is the bound on the F mapping in Assumption 9(c).*

Proof. First we provide a brief sketch of the proof. Since the stochastic process in Eq. 6.26 is guaranteed to stay bounded (Assumption 9(c)), one can find other processes that lower bounds and upper bounds this process. Let us assume that we can show that the process in Eq. 6.26 always stays bounded by these two processes after some finite time t (that is for all $t' \geq t$). Now, if we can prove that the process A is upper bounded by a finite value, this value will be the upper bound of the process in Eq. 6.26 after t as well. Similarly, the lower bound of L will be its lower bound (after time t).

Now, we provide the complete proof of the result.

Let p be an arbitrary scalar such that

$$x^* - pe \leq x(t) \leq x^* + pe; \quad \forall t > t'.$$

Such a p is possible because the theorem assumes that the process $x(t)$ will not diverge to infinity. Here t' is a point after which the mapping F is bounded in $[x^* - D, x^* + D]$. We can find such a t' with a high probability due to Assumption 9(c). Our proof lies in the space of t such that $t > t'$. Now, let us consider, $L^0 = (L_1^0, \ldots, L_n^0) = x^* - pe$ and $A^0 = (A_1^0, \ldots, A_n^0) = x^* + pe$. Recall that p and D are scalars and e is a vector with all components equal to 1. From now on we will use p and D to denote pe and De respectively. Let us define two sequences A^k and L^k such that

$$A^{k+1} = \frac{A^k + F(A^k) + D}{2}, \quad k \geq 0 \tag{6.27}$$

and

$$L^{k+1} = \frac{L^k + F(L^k) - D}{2}, \quad k \geq 0$$

Now we can provide the following results.

Lemma 32. *For every $k \geq 0$, we have*

$$F(A^k) \leq A^{k+1} \leq A^k + D, \tag{6.28}$$

and

$$F(L^k) \geq L^{k+1} \geq L^k - D \tag{6.29}$$

Lemma 33. *The sequence A^k will converge to a point upper bounded by $x^* + 2D$ and the sequence L^k will converge to a point lower bounded by $x^* - 2D$.*

We will now show that for every k, there exists some time t_k such that,

$$L^k \leq x(t) \leq A^k, \quad \forall t \geq t_k \tag{6.30}$$

Once this is proved, $x(t)$ can be shown to remain in the bound $x^* - 2D$ and $x^* + 2D$ from Lemma 33. We can see that Eq. 6.30 is true for $k = 0$, with $t_0 = 0$, by definition of A^0 and L^0. Proceeding with induction on k, we fix some k and assume that there exists some t_k so that the Eq. 6.30 holds. Let t'_k be such that for every $t \geq t'_k$ for every i, j, we have $\tau_j^i(t) \geq t_k$. Such a t'_k exists because of Assumption 6.

In particular, we have

$$L^k \leq x^i(t) \leq A^k, \quad \forall t \geq t'_k$$

Let $W_i(0) = 0$ and

$$W_i(t+1) = (1-\alpha_i(t))W_i(t) + \alpha_i(t)w_i(t), \quad t \geq t_0$$

Now, we can rearrange the terms to get

$$W_i(t+1) - \alpha_i(t)w_i(t) = (1-\alpha_i(t))W_i(t)$$

This sequence will converge to 0 in the limit ($t \to \infty$) by the condition on $\alpha_i(t)$ from Assumption 8. This assumption implies that

$$\Pi_{\tau=0}^{\infty}(1-\alpha_i(\tau)) = 0.$$

Thus the sequence W_i will converge to $\alpha_i(t)w_i(t)$. Now, by Assumption 7, we can redefine $w_i(t)$ such that it goes to 0 in the limit ($t \to \infty$). We then have $\lim_{t\to\infty} W_i(t) = 0$.

For any time t_0, we also define $W_i(t_0; t_0) = 0$ and

$$W_i(t+1; t_0) = (1-\alpha_i(t))W_i(t; t_0) + \alpha_i(t)w_i(t), \quad t \geq t_0 \tag{6.31}$$

For every t_0 then we can have $\lim_{t\to\infty} W_i(t; t_0) = 0$ using a similar argument as above (also see Lemma 2 in Tsitsiklis [276]).

We also define a sequence $X_i(t), t \geq t'_k$ by letting $X_i(t'_k) = A_i^k$ and

$$X_i(t+1) = (1-\alpha_i(t))X_i(t) + \alpha_i(t)F_i(A^k), \quad t \geq t'_k \tag{6.32}$$

Next, we present a lemma on this sequence.

Lemma 34.

$$x_i(t) \leq X_i(t) + W_i(t; t'_k), \forall t \geq t'_k$$

We define a δ_k be equal to minimum of $(A_i^k + 2D - F_i(A^k))/4$, where the minimum is taken over all i for which $(A_i^k + 2D - F_i(A^k))$ is positive. δ_k is well defined and positive due to Lemma 33 and the lower bound of A^k and upper bound of F.

Let us define a $t''_k \geq t'_k$,

$$\Pi_{\tau=t'_k}^{t''_k-1}(1-\alpha_i(\tau)) \leq \frac{1}{4} \tag{6.33}$$

and for all $t \geq t''_k$ and all i.

$$W_i(t;t'_k) \leq \delta_k \tag{6.34}$$

The condition in Eq. 6.33 is possible due to Assumption 8. The condition in Eq. 6.34 is possible because $W_i(t;t'_k)$ from Eq. 6.31 converges to 0 as discussed earlier. This leads to the following lemma.

Lemma 35. *We have $x_i(t) \leq A_i^{k+1}$, for all i and $t \geq t''_k$*

By an entirely symmetrical argument, we can also establish that $x_i(t) \geq L_i^{k+1}$ for all t greater than some t'''_k. This concludes the proof of the theorem.

□

For the further proofs, we will require three more assumptions same as specified in Chapter 3. These assumptions are Assumption 1, 2, and 5.

Let $\tilde{a}_i$ be a component of vector $\tilde{a}$ and $\overline{a}_i$ be a component of vector $\overline{a}$. Now, we make a comparison between mean actions of the MFQ update (Eq. 6.4) and the POMFQ update (Eq. 6.10).

Theorem 13. *The MFQ mean action and the POMFQ mean action both satisfy*

$$|\tilde{a}_{i,t} - \overline{a}_{i,t}| \leq \sqrt{\frac{1}{2n}\log\frac{2}{\delta}}$$

as time $t \rightarrow \infty$, with probability $>= \delta$, where n is the number of samples observed. $\tilde{a}$ is the mean action as obtained from the Dirichlet in Eq. 6.10 and $\overline{a}$ is the mean action in Eq. 6.4.

Proof. Using the Hoeffding's inequality, if a set of random variables $(X_1, \cdots, X_n)$ are bounded by the intervals $[a_i, b_i]$, then the following is true:

$$P(|\overline{X} - \mathbb{E}[\overline{X}]| \geq u) \leq 2\exp\left(\frac{-2n^2u^2}{\sum_{i=1}^{n}(b_i-a_i)^2}\right) \tag{6.35}$$

where n is the number of samples. P denotes the probability and u is an arbitrary bound.

Eq. 6.10 samples some $\tilde{a}$ to estimate the partially observable Q function Q_{POMF}, as seen in update Eq. 6.9. This is set to be $\overline{X}$ in Eq. 6.35 and its expected value then will be true mean field action $\overline{a}$. In this setting, we have assumed that all agents have global state availability. Therefore all the agents in the environment are visible and all actions of the agents can be used to update the Dirichlet distribution, which holds the estimate of POMFQ mean action. Note that in each step, a very large number of agent actions are visible (we assume there are N agents in the environment where N is very large). Thus, the Dirichlet mean will become close to the true underlying $\overline{a}$. Since the agent is taking only a finite sample from the Dirichlet to update the POMF Q-function, the empirical mean will be $\tilde{a}$.

Therefore, from Eq. 6.35 we have,

$$P(|\tilde{a}_{i,t} - \overline{a}_{i,t}| \geq u) \leq 2\exp\left(\frac{-2n^2u^2}{\sum_{j=1}^{n}(b_j - a_j)^2}\right)$$

The samples are in the range $[0, 1]$ and therefore we set $b_j = 1$ and $a_j = 0$ We set the right hand side of Eq. 6.35 to δ to get the relation

$$u = \sqrt{\tfrac{1}{2n}\log\tfrac{2}{\delta}}$$

which proves the theorem.

□

Theorem 14. *When the Q-function is Lipschitz continuous (with constant M) with respect to mean actions, then the POMF Q-function will satisfy the following relationship:*

$$|Q^{POMF}(s_t, a_t, \tilde{a}_{t-1}) - Q^{MF}(s_t, a_t, \overline{a}_{t-1})| \leq M \times L \times \log\frac{2}{\delta} \times \frac{1}{2n} \tag{6.36}$$

as $t \to \infty$ with probability $\geq (\delta)^{L-1}$, where $L = |A|$ and n is the number of samples.

Proof. Consider a Q-function that is Lipschitz continuous for all $\overline{a}$ and $\tilde{a}$. Then we get,

$$|Q(s_t, a_t, \tilde{a}_{t-1}) - Q(s_t, a_t, \overline{a}_{t-1})| \leq M|\tilde{a}_{t-1} - \overline{a}_{t-1}|$$

Now, from Theorem 13, we get,

$$|Q(s_t, a_t, \tilde{a}_{t-1}) - Q(s_t, a_t, \overline{a}_{t-1})| \leq M \times L \times (\sqrt{\tfrac{1}{2n} \log \tfrac{2}{\delta}})^2$$

In the first step, we are taking the magnitude of the difference between the mean action vectors, hence we multiply the bound in Theorem 13 with all the components of the vectors ($\overline{a}$ and $\tilde{a}$). The total number of components are equal to the action space L. Since the bound for Theorem 13 is with probability $\geq \delta$ the probability of this theorem would be at least δ^{L-1}, since we have L random variables, and when we fix the first $L-1$ random variables, the last one is deterministic as all the components of $\overline{a}$ satisfy the relation that sum of the individual components will be 1 (one hot encoding). All the components of $\tilde{a}$ also satisfy this relation as $\tilde{a}$ is a normalized sample obtained from the Dirichlet.

Then, from the definition of POMF Q-function and MFQ Q-function, the theorem follows.

$\square$

From Theorem 14, we can see that in a similar setting, the POMFQ updates will not see a significant degradation in performance as compared to the MFQ updates. The probability of this holding is inversely proportional to the size of the action space available to each agent. In Theorem 14, the bound is between two Q-functions with the same state and action, but with different mean actions. Let $Z = M \times L \times \log \frac{2}{\delta} \times \frac{1}{2n}$ and from Theorem 14, $|Q^{POMF}(s_t, a_t, \tilde{a}_{t-1}) - Q^{MF}(s_t, a_t, \overline{a}_{t-1})| \leq Z$. Now, we would like to directly compare the value estimates of POMFQ and MFQ updates. Consider two different actions a^j and b^j for agent j. Under the assumption that the mean field Q-function is K-Lipschitz continuous with respect to actions,

$$|Q^{MF}(s^j_t, a^j_t, \overline{a}^j_{t-1}) - Q^{MF}(s^j_t, b^j_t, \overline{a}^j_{t-1})| \leq K|a^j_t - b^j_t| \leq K\sqrt{2} \qquad (6.37)$$

In the last step, we applied the fact that all components of a^j and b^j are less than or equal to 1 (a one hot encoding). Assume that the optimal action for Q^{POMF} is a^* and for Q^{MF} is b^*. Now consider,

$$|v^{POMF}(s_{t+1}) - v^{MF}(s_{t+1})| \tag{6.38}$$

$$= |\max_{a_{t+1}} Q^{POMF}(s_{t+1}, a_{t+1}, \tilde{a}_t) - \max_{b_{t+1}} Q^{MF}(s_{t+1}, b_{t+1}, \overline{a}_t)| \tag{6.39}$$

$$= |Q^{POMF}(s_{t+1}, a^*_{t+1}, \tilde{a}_t) - Q^{MF}(s_{t+1}, a^*_{t+1}, \overline{a}_t) \tag{6.40}$$

$$+ Q^{MF}(s_{t+1}, a^*_{t+1}, \overline{a}_t) - Q^{MF}(s_{t+1}, b^*_{t+1}, \overline{a}_t)| \leq Z + K\sqrt{2} \triangleq D \tag{6.41}$$

In the first step we apply the fact that the Boltzmann policy will become greedy in the limit ($t \to \infty$). The last step is coming from Eqs. 6.36 and 6.37. We also reiterate that the Lipschitz continuity assumptions on the Q-function are consistent with prior work [303].

Theorem 15. *When we update the Q functions using the partially observable update rule in Eq. 6.6, the process satisfies the condition in the limit ($t \to \infty$):*

$$|Q^*(s_t, \boldsymbol{a}_t) - Q^{POMF}(s_t, a_t, \tilde{a}_t)| \leq 2D$$

when Assumptions 1, 2, and 5 hold. Here Q^* is the Nash Q-value and D is the bound for value functions in Eq. 6.41. This holds with probability at least δ^{L-1}, where $L = |A|$.

Proof. We start by proving all the assumptions needed for Theorem 12 and then apply Theorem 12 to prove this theorem.

We can write the Q update from Eq. 6.6 using the formula:

$$Q^{POMF}_{t+1}(s^j_t, a^j_t, \tilde{a}^j_t)$$
$$= Q^{POMF}_t(s^j_t, a^j_t, \tilde{a}^j_t) + \alpha_t[r^j_t + \gamma v^{POMF}_t(s^j_{t+1}) - Q^{POMF}_t(s^j_t, a^j_t, \tilde{a}^j_t)] \tag{6.42}$$

Let F be defined as

$$F(Q^{POMF}_t(s^j_{t+1}, a^j_t, \tilde{a}^j_t)) = \mathbb{E}[r^j] + \gamma\,\mathbb{E}[v^{POMF}_t(s^j_{t+1})] \tag{6.43}$$

and the value function from Eq. 6.9 be

$$v^{POMF}_t(s^j_{t+1}) = \sum_{a^j} \pi^j_t(a^j|s', \tilde{a}^j) Q^{POMF}_t(s^j_{t+1}, a^j, \tilde{a}^j) \tag{6.44}$$

Using Eq. 6.43, then Eq. 6.42 can be written as

$$Q_{t+1}^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j)$$

$$= Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) + \alpha_t[F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j)) - Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) + w_t(s_t^j, a_t^j, \tilde{a}_t^j)]$$

where

$$w_t(s_t^j, a_t^j, \tilde{a}_t^j) = r^j - \mathbb{E}[r^j] + \gamma v_t^{POMF}(s_{t+1}^j) - \gamma\, \mathbb{E}[v_t^{POMF}(s_{t+1}^j)]$$

Assumption 6 is satisfied, since we are setting $\tau_j^i(t) = t$ in Eq. 6.42. Our formulation is the same as that in Tsitsiklis [276] where the Q function is not allowed any outdated information from the previous steps in the current update. Assumption 6 only needed to guarantee that, if old information were used, that would be eventually discarded w.p.1.

To satisfy all the measurability conditions of Assumption 2, let $\mathcal{F}_t$ be a σ-field generated by random variables in all history of the stochastic game till the time step t, i.e., $(s_t, \alpha_t, \boldsymbol{a}_t, r_{t-1}, \tau_t, \cdots, s_1, \alpha_1, \boldsymbol{a}_1, \tau_1, Q_0)$. Let Q_t be a random variable from this trajectory and is hence $\mathcal{F}_t$-measurable. Thus, from Eq. 6.43, we can see that F will also be $\mathcal{F}_t$-measurable. This satisfies Assumptions 7(a), (b), and (c). Assumption 7(d) can be directly verified from Eq. 6.43. From Assumption 1 it can be shown that the action-value function and the value function remain bounded and hence we can find arbitrary constants A and B such that Assumption 7(e) will hold.

To satisfy Assumption 9(a), note that the F mapping in Eq. 6.43 will increase or decrease only as the term v^{POMF} increases or decreases for a given reward function. Now v^{POMF} depends on Q^{POMF}. As Q^{POMF} increases or decreases then v^{POMF} also increases or decreases for a stationary policy. Thus the monotonicity condition (Assumption 9(a)) is proved. Because the function is a linear function, the continuity condition is also proved (Assumption 9(b)). Hence, the first two conditions of Assumption 9 are satisfied.

Using Eq. 6.43 we can write:

$$F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j))$$

$$= \mathbb{E}[r^j] + \gamma\, \mathbb{E}[v_t^{MF}(s_{t+1}^j)] + \gamma[\mathbb{E}[v_t^{POMF}(s_{t+1}^j)] - \mathbb{E}[v_t^{MF}(s_{t+1}^j)]]$$

$$= \mathbb{E}[r^j + \gamma[v_t^{Nash}(s_{t+1}^j)]] + \mathbb{E}(\gamma[v_t^{POMF}(s_{t+1}^j) - v_t^{MF}(s_{t+1}^j)]) + \mathbb{E}(\gamma[v_t^{MF}(s_{t+1}^j) - v_t^{Nash}(s_{t+1}^j)])$$

$$\leq Q^*(s_t^j, \boldsymbol{a}_t^j) + D$$

Also we have,

$$F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j))$$

$$= \mathbb{E}[r^j + \gamma[v_t^{Nash}(s_{t+1}^j)]] + \mathbb{E}(\gamma[v_t^{POMF}(s_{t+1}^j) - v_t^{MF}(s_{t+1}^j)]) - \mathbb{E}[\gamma[v_t^{Nash}(s_{t+1}^j) - v_t^{MF}(s_{t+1}^j)]]$$

$$\geq Q^*(s_t^j, \boldsymbol{a}_t^j) - D$$

In the above two equations we use the fact that in the limit ($t \to \infty$) the mean field value function becomes equal to the Nash value function given Assumption 5 (see Theorem 1 in Yang et al. [303] for the proof). Thus, that term can be dropped. This satisfies Assumption 9(c).

Now to satisfy Assumption 9(d), consider,

$$\begin{aligned} &F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) + p) \\ &= \mathbb{E}[r^j] + \gamma[XQ_t^{POMF}(s_{t+1}^j, a^j, \tilde{a}^j) + p] \\ &= \mathbb{E}[r^j] + \gamma[XQ_t^{POMF}(s_{t+1}^j, a^j, \tilde{a}^j)] + \gamma p] \end{aligned} \tag{6.45}$$

where $X = \sum_{a^j} \pi_t^j(a^j | s', \overline{a}^j)$.

Now consider,

$$\begin{aligned} &F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j)) + p \\ &= \mathbb{E}[r^j] + \gamma[XQ_t^{POMF}(s_{t+1}^j, a^j, \tilde{a}^j)] + p \end{aligned} \tag{6.46}$$

Since, $\gamma \leq 1$, from Eqs. 6.45 and 6.46, we find that

$$F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) + p) \leq F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j)) + p \tag{6.47}$$

By a symmetric argument, we can prove

$$F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j)) - p \leq F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) - p)$$

and the condition

$$F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) - p) \leq F(Q_t^{POMF}(s_t^j, a_t^j, \tilde{a}_t^j) + p)$$

will hold by monotonicity.

Thus, all the conditions of Assumption 9 have been proved.

Consider the POMF Q-update in Eq. 6.6. Notice that if the reward function r is guaranteed to be bounded, then the Q-function will also be bounded as there are no other variables that Q depends on, which can make it diverge to infinity. Since Assumption 5 guarantees that the reward function will stay bounded, the value and action-value functions will also stay bounded. This will mean that the entire stochastic process as given in Theorem 12 remains bounded. Now, with all conditions met, we can prove that the partially observable mean field Q function will either converge or oscillate in a small range $(Q^*(s_t, \boldsymbol{a}_t) - 2D, Q^*(s_t, \boldsymbol{a}_t) + 2D)$ in the limit $(t \to \infty)$ according to Theorem 12.

□

It is important to note that we need only three minor assumptions (Assumptions 1, 2, and 5) to hold for Theorem 4, which is our main theoretical result. Theorem 15 shows that the POMFQ updates stay very close to the Nash equilibrium in the limit $(t \to \infty)$. The lower bound on the probability of this is high for a small action space and low for a large action space. In a multi-agent setting, the Q-updates are in the form of POMFQ updates, and do not have the (intuitive) effect of having any fixed point as commonly seen in RL. Theorem 15 proves our update rule is very close to the Nash equilibrium, a stationary point for the stochastic game. Hence, the policy in Eq. 6.5 is approximately close to this stationary point, which guarantees that it becomes (asymptotically) stationary in the limit $(t \to \infty)$.

The distance between the POMF Q-function and the Nash Q-function is inversely proportional to the number of samples from the Dirichlet (n). If the agent chooses to take a large number of samples, the POMF Q-estimate is very close to the Nash Q-estimate, but this may lead to a degradation in performance due to having no additional exploratory noise as discussed in Section 6.3. In MARL, the Nash equilibrium is not a guarantee of optimal performance, but only a fixed point guarantee. The (self-interested) agents would still take finite samples, for better performance. To balance the theory and performance, the value of n should not be too high nor too low.

In Section 6.8 we provide an experimental illustration of Theorem 15 in the Ising model, a mathematical model used to describe the magnetic spins of atomic particles.

6.8 Ising Model

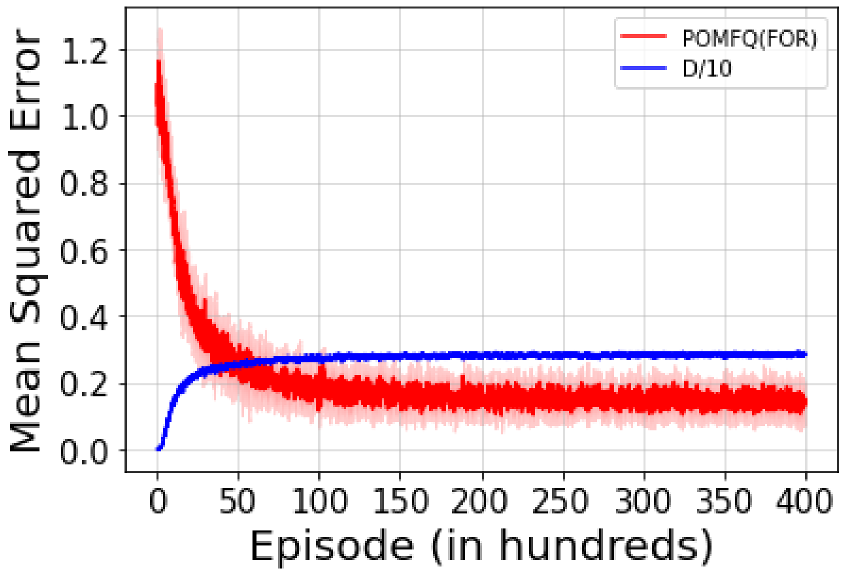


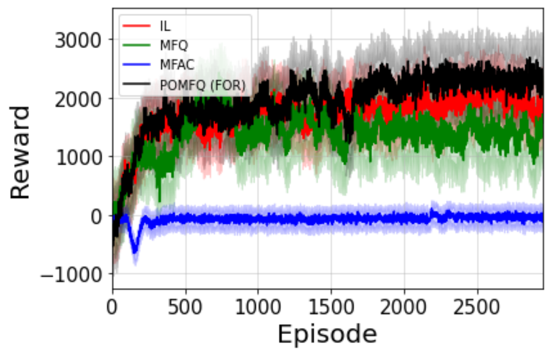
Figure 6.1: Ising model: Mean square distance between the Q-values and Nash Q-values.

The Ising model was introduced as a stochastic game by Yang et al. [303]. The energy function of this model determines that the overall energy of the system stays low when each agent chooses to spin in a direction that is consistent with its neighbours. In this game, each agent has a choice of one of the two actions (spinning up or spinning down) and obtains rewards proportional to the number of agents spinning in the same direction in the neighbourhood. We use the same settings and parameters for the Ising model as in Yang et al. [303]. Particularly, each agent obtains rewards [-2.0, -1.0, 0.0, 1.0, 2.0] based on the number of neighbours [0,1,2,3,4] spinning in the same direction as itself in each stage game. Refer to [303] for more details on this domain. We set the temperature τ of the ising model to 0.8. We use this domain to give a practical illustration of the tabular version of our algorithm on a simple domain and to verify Theorem 15. The Ising model used in our implementation is a stateless system with a total of 100 agents and the Nash Q-function of this stochastic game is exactly obtainable [303]. We implement the POMFQ (FOR) algorithm on this domain and calculate the mean square error (MSE) between the Q-values of each agent's action with the Nash Q-value at each stage. The average of this error across all the 100 agents is plotted against number of episodes in Figure 6.1, along with the 95% confidence interval shaded out around each point. From the plot it can be seen that the MSE steadily reduces and its 95% confidence interval stays bounded by the line representing the value of $D/10$ line after a finite number of episodes. This is a stronger result than Theorem 15 where we had proved that the error will be bounded by $2D$. This shows that there is some scope for a stronger version of Theorem 15 as well. Substituting the value of δ as 0.95 in Theorem 13 we can calculate the value of the constant D used in Theorem 15. Notice that D as given in Eq. 6.41 is the sum of two other constants Z and K. We set the value of the number of samples n drawn from the Dirichlet at each time step to be 10000 and hence from Theorem 14 it can be seen that constant $Z \approx 0$. Now to calculate the Lipschitz constant K, we calculate the difference between the MFQ Q-values between the two actions for the given state and mean action as in Eq 6.37. Since we start all the Q-values with an initialization of 0, the value of K (and hence, D) is 0 at the beginning of Figure 6.1. However, once learning takes place and the Q-values change, we see a finite value of D. As the Q-values move closer to convergence, the value of D also converges to a finite value as seen in Figure 6.1. This value is used to compare the mean squared error of the Q-values in Figure 6.1.

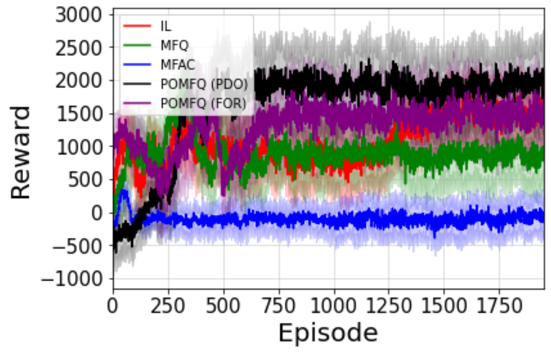
6.9 Experiments And Results

This section empirically demonstrates that using POMFQ updates will result in better performance in a partially observable environment than when using the MFQ updates. All the code for the experiments is open sourced [245].

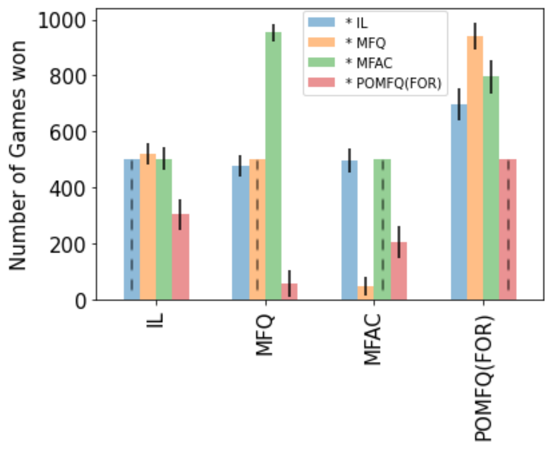
We design three cooperative-competitive games for each of the two problems (FOR and PDO) within the MAgent framework [317] to serve as testbeds. This is the same simulator considered in Chapter 5, however our current domains have several differences as compared to those used in Chapter 5. Here, we wish to highlight the importance of partial observability and not consider types as in the Chapter 5. We will provide the important elements of these experimental domains here, while the comprehensive details (including exact reward functions and hyperparameter values) are deferred to Appendix D.2. For all games, we have a two stage process: training and faceoff (test). We consider four algorithms for all the games: MFQ, MFAC, IL, and POMFQ. In each stage, there are two groups of agents: group A and group B. Since the agents do not know what kind of opponents they will see in the faceoff stage, they train themselves against another group that plays the same algorithm in the training stage. Thus, in the training stage, each algorithm will train two networks (groups A and B). In the faceoff stage, groups trained by different algorithms fight against each other. Our formulation is consistent with past research using the MAgent framework [248, 303]. We plot the rewards obtained by group A in each episode for the



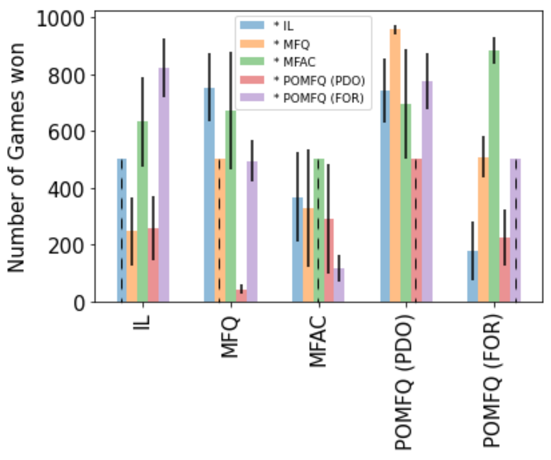
(a) FOR - Train



(b) PDO - Train



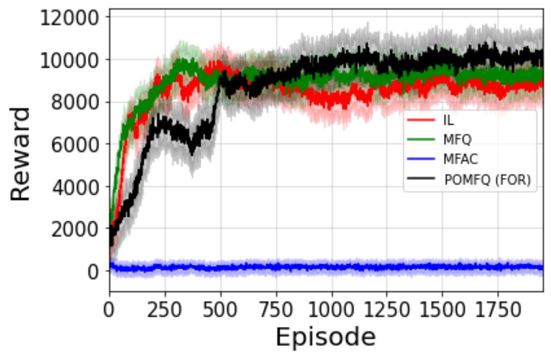
(c) FOR - Test



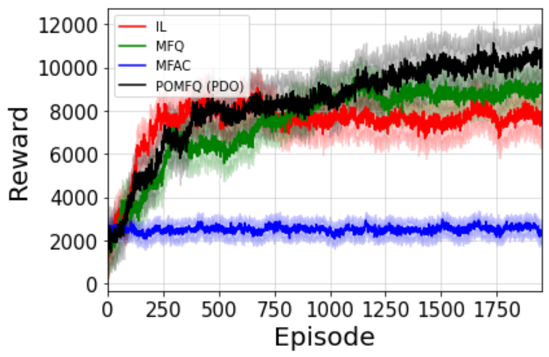
(d) PDO - Test

Figure 6.2: Multibattle results. The * in the legend of test plots denotes the opponent. For example, first orange bar (from the left) in the bar plots is result for IL. vs MFQ. The dashed lines indicate bars that we set for symmetry. We do not run faceoff experiments between the same algorithm.

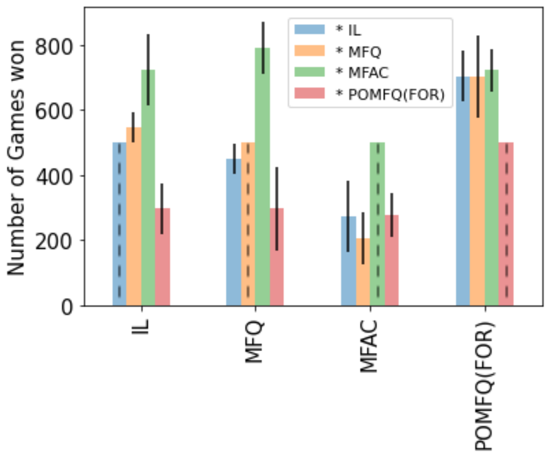
training stage (group B also shows similar trends — our games are not zero sum) and the number of games won by each algorithm in the faceoff stage. For statistical significance, we report p-values of an unpaired 2-sided t test for particular episodes in the training stage and a Fischer's exact test for the average performances in the faceoff stage. We treat p-values of less than 0.05 as statistically significant differences. The tests are usually conducted between POMFQ and next best performing algorithm in the final episode of training for the training results.



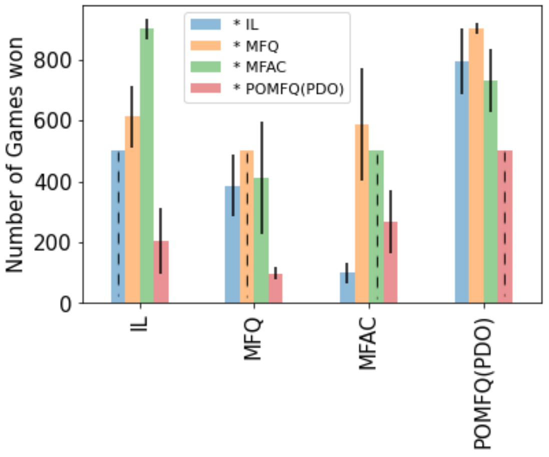
(a) FOR - Train



(b) PDO - Train



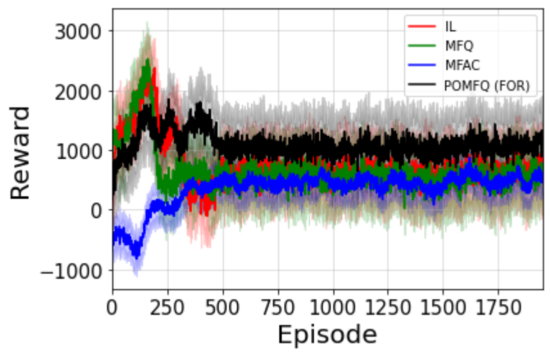
(c) FOR - Test



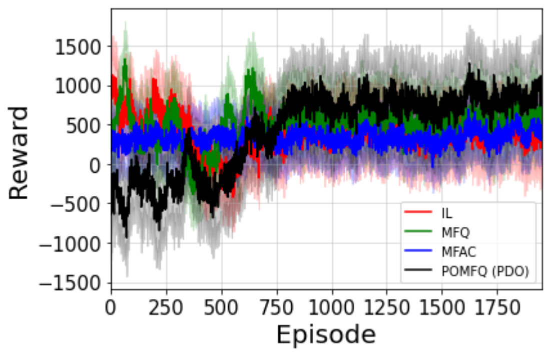
(d) PDO - Test

Figure 6.3: Training and faceoff results of Battle-Gathering game.

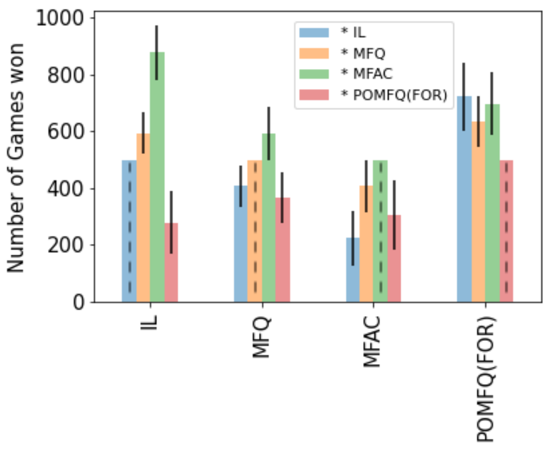
The Multibattle game has two groups of agents fight against each other. There are 25 agents in each group for a total of 50. Agents learn to cooperate within the group and compete across the group to win. We analyse both the FOR and PDO cases. In FOR, information about nearby agents is available, but agents further than 6 units are hidden. In PDO, the game engine maintains a Bernoulli distribution of visibility of each agent from every other agent as discussed in Section 6.4. Based on this probability, each agent in PDO could see different numbers of other agents at each time step. MFQ and MFAC use a frequentist strategy where agents observed at each time step are aggregated (Eq. 6.4) to obtain a mean action. We run 3000 episodes of training in FOR and 2000 episodes in PDO. Each episode has a maximum of 500 steps. For the faceoff, group A trained using the first



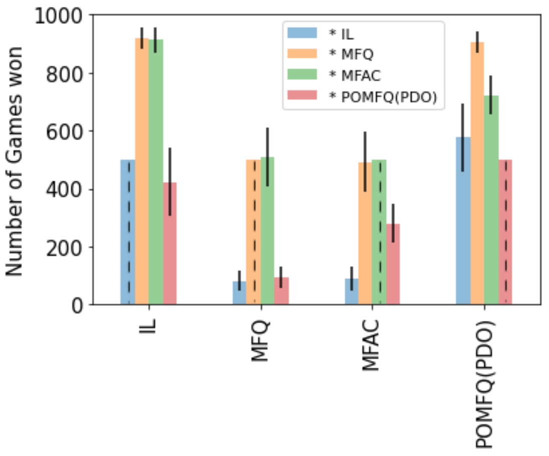
(a) FOR - Train



(b) PDO - Train



(c) FOR - Test



(d) PDO - Test

Figure 6.4: Training and faceoff results of Predator-Prey game.

algorithm and group B trained using the second algorithm fight against each other for 1000 games. We report all results as an average of 20 independent runs for both training and faceoff (with standard deviation). In our experiments, an average of 6 – 8 agents out of 50 agents are visible to the central agent at a given time step (averaged over the length of the game). Note that we use 50 agents per game, more agents are used in previous research [248, 303]. In our case, the ratio of agents seen vs. the total number of agents matters more than the simple absolute number of agents in the competition.

In the FOR setting of the Multibattle domain (Figure 6.2 (a)) the POMFQ algorithm plays the FOR variant (Algorithm 11). POMFQ dominates other baselines from about 1800 episodes ($p < 0.3$) until the end ($p < 0.03$). We see that MFAC quickly falls into a poor local optimum and never recovers. The poor performance of MFAC in the MAgent games, compared to the other baselines, is consistent with previous work [248, 303]. Faceoff in the FOR case (Figure 6.2(c)) shows that POMFQ wins more than 50% of the games against others ($p < 0.01$). An ablation study in Section 6.11 shows that performance improves with increase in viewing distance.

In the PDO setting, we use both the FOR variant of POMFQ algorithm (no $\overline{\lambda}$ parameter) and the PDO variant of POMFQ algorithm (Algorithm 12). We differentiate these two algorithms in the legends of Figures 6.2(b) and 6.2(d). The FOR variant loses out to the PDO algorithm that explicitly tracks the $\overline{\lambda}$ parameter ($p < 0.02$). If an algorithm bases decisions only on $\tilde{a}$, as in Algorithm 11, the agents do not know how risk seeking or risk averse their actions should be (when agents nearby are not visible). In this game, agents can choose to make an attack (risk seeking) or a move (risk averse). The additional parameter $\overline{\lambda}$ helps agents understand the uncertainty in not seeing some agents when making decisions. The PDO algorithm takes a lead over the other algorithms from roughly 900 episodes ($p < 0.04$) and maintains the lead until the end ($p < 0.03$). In faceoff, PDO wins more than 50% (500) of the games against all other algorithms as seen in Figure 6.2(d) ($p < 0.01$).

The second game, Battle-Gathering, is similar to the Multibattle game where a set of two groups of 50 agents are fighting against each other to win a battle, but with an addition of food resources scattered in the environment. All the agents are given an additional reward when capturing food, in addition to killing the competition (as in Multibattle). The training and faceoff are conducted similar to Multibattle game. Figure 6.3(a) shows that the POMFQ algorithm dominates the other three algorithms from about 900 episodes ($p < 0.03$) till the end ($p < 0.01$). In the comparative battles (Figure 6.3(c)), POMFQ has a clear lead over other algorithms ($p < 0.01$). MFQ and IL are similar in performance and MFAC loses to all other algorithms. We also observe this in the PDO domain train ($p < 0.02$, Figure 6.3(b)) and test experiments ($p < 0.01$, Figure 6.3(d)).

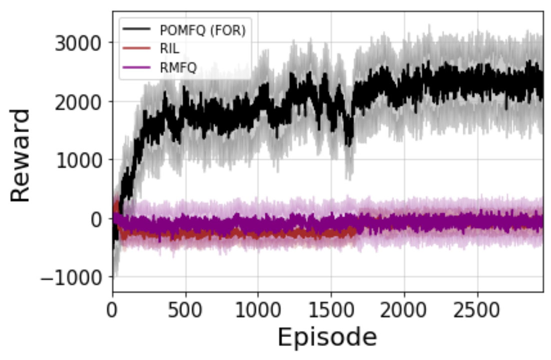
The third game is a type of Predator-Prey domain, where there are two groups — predators and prey. There are a total of 20 predator agents and 40 prey agents in our domain. The predators are stronger than the prey and have an objective of killing the prey. The prey are faster than the predator and try to escape from the predators. The training is conducted and rewards are plotted using the same procedure as in the Multibattle domain. Training performances are in Figures 6.4(a) and 6.4(c). The standard deviation of the performance in this game is considerably higher than the previous two games because we have two completely different groups that are trying to outperform each other in the environment. At different points in training, one team may have a higher performance than the other, and this lead can change over time. In the first setting, Figure 6.4(a), we can see that the POMFQ (FOR) shows a small lead over other baseline algorithms (at the end of training, p $<$ 0.1). In the direct faceoff (Figure 6.4(c)), POMFQ wins more games than the other algorithms ($p < 0.01$), In the PDO setting too, the POMFQ-PDO algorithm has an edge over the others during the training phase ($p < 0.4$) and the testing ($p < 0.01$)(Figure 6.4(b) and 6.4(d)). As the p-values for the training suggest, POMFQ can be seen to have a better performance, but the results are not statistically significant. The faceoff results, on the other hand, are statistically significant ($p < 0.01$). We have run training for 2000 games and faceoff for 1000 games in the last two domains.

In three semantically different domains, we have shown that in the partially observable case, the MFQ and MFAC algorithms using frequentist strategies do not provide good performances. Also, the frequentist strategies (MFQ and MFAC) have worse performance in the harder PDO domain compared to the FOR domain. Sometimes, they also lose out to a simpler algorithm that does not even track the mean field parameter (IL). The FOR and PDO algorithms gives the best performance across both settings, as evidenced by the training and the test results. The training results clearly show that POMFQ never falls into a very poor local optimum like MFAC often does. The test results show that in a direct face-off, POMFQ outperforms all other algorithms. The p-values indicate that our results are statistically significant. Additionally, in Section 6.10, we provide comparisons of the POMFQ - FOR and PDO algorithms with two more baselines, recurrent versions of IL and MFQ, in the same three MAgent domains and the results show that POMFQ has clear advantages compared to these recurrent baselines as well.

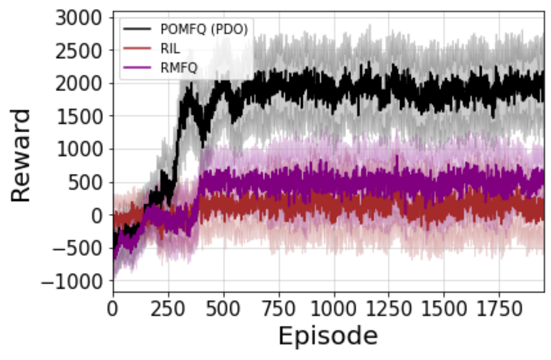
6.10 Recurrent Baselines

This section describes our comparisons of the POMFQ - FOR and PDO algorithms with recurrent versions of IL and MFQ (denoted as RIL and RMFQ respectively). We report all results for 20 independent runs as done for the comparisons in Section 6.9.

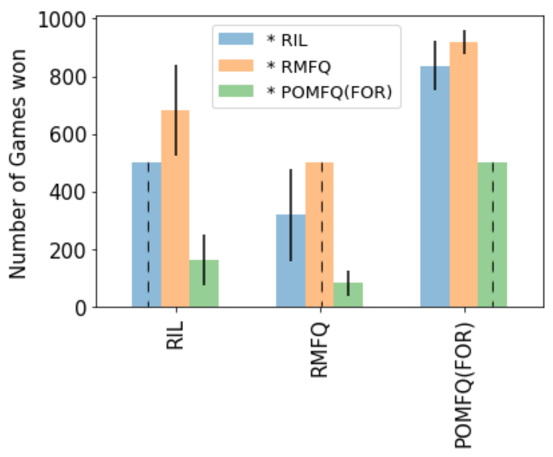
Multibattle game results are given in Figure 6.5. In the FOR domain, Figure 6.5(a), we can see that the recurrent algorithms fall into a local optimum early on and do not improve much, similar to the performance of MFAC. The reason for this could be that the recurrent network provides a larger amount of information to the agent at each time step because the state is recurrently fed back to the network as an input. In a mean field setting, this may not a good idea since there is already an overload of information available to each agent, and the superior performance of POMFQ is due to the fact that it supplies exactly the right amount of information needed to calculate the best response at each time step. In the test battles for FOR (Figure 6.5 (c)), POMFQ (FOR) comfortably beats RIL ($p < 0.01$)



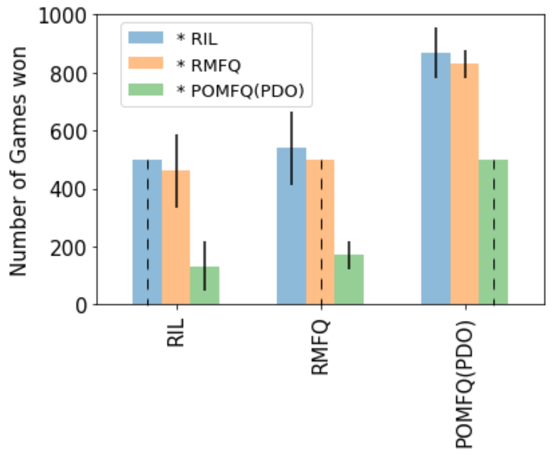
(a) FOR - Train



(b) PDO - Train



(c) FOR - Test

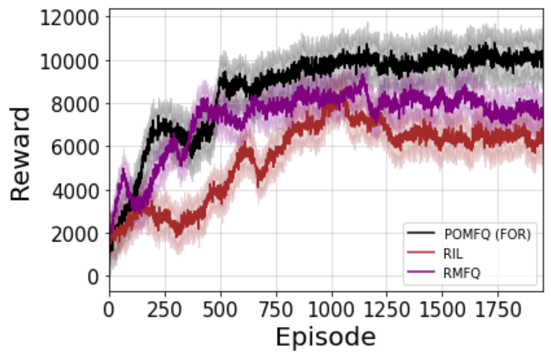


(d) PDO - Test

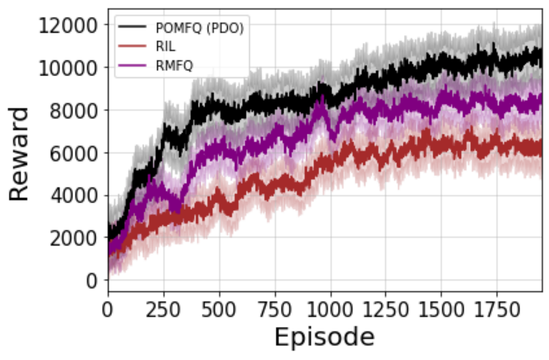
Figure 6.5: Multibattle results with recurrent baselines.

and RMFQ ($p < 0.01$). This performance advantage is also seen in the PDO setting, where the POMFQ (PDO) beats RIL and RMFQ in both the train (Figure 6.5(b), $p < 0.01$) and test experiments (Figure 6.5(d), $p < 0.01$).

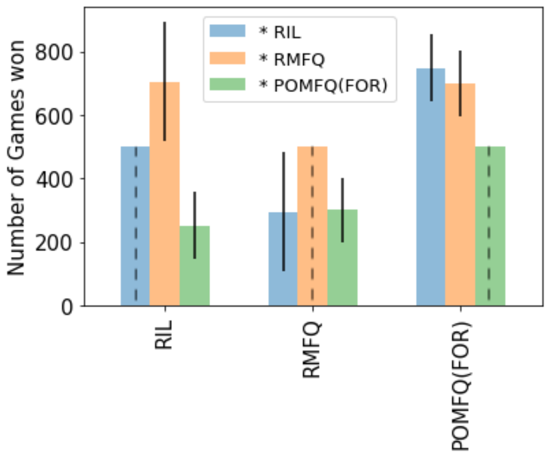
In the Battle-Gathering (Figure 6.6) we still see that the POMFQ beats the recurrent baselines in both training and test performances in both the domains. However, it is interesting to note that in both the FOR (Figure 6.6 (a)) and PDO (Figure 6.6 (b)) experiments, both the recurrent baselines learn to perform reasonably well and do not fall into a trivial local optimum like in the Multibattle game. Recall that this game has two tasks, one is to gather as many food particles as possible and the other is to kill the enemies.



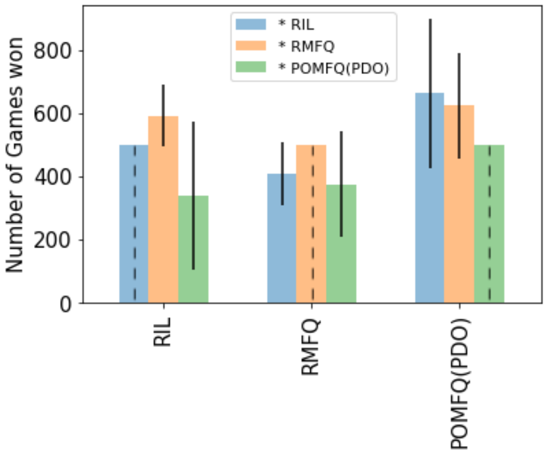
(a) FOR - Train



(b) PDO - Train



(c) FOR - Test

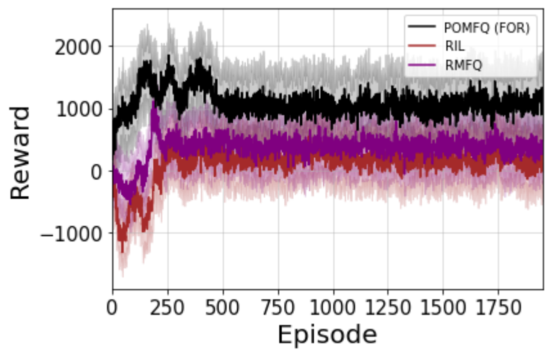


(d) PDO - Test

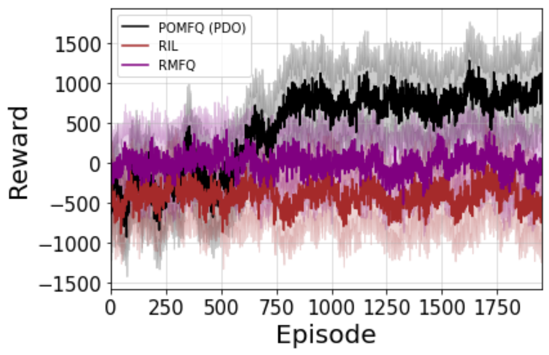
Figure 6.6: Battle-Gathering results with recurrent baselines.

The recurrent baselines seem to learn to perform one of the tasks very well (gathering food), but still lose out to POMFQ, which learns to perform both the tasks reasonably well. The test results in Figure 6.6(c) and (d) also show the performance advantage of POMFQ against the recurrent baselines in this domain. The train results have $p < 0.04$ and test results have $p < 0.02$ for the comparisons between POMFQ and the next best performing recurrent algorithm.

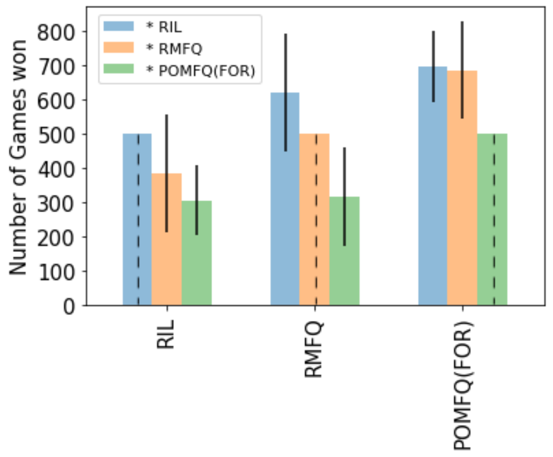
For the Predator-Prey domain, the results are given in Figure 6.7. In this domain too, POMFQ algorithm beats the performance of both the recurrent baselines in the training experiments (Figure 6.7(a) and (b) with $p < 0.06$) as well as the test experiments (Figure



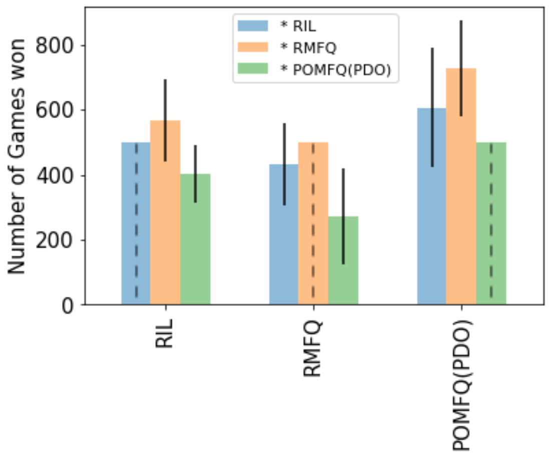
(a) FOR - Train



(b) PDO - Train



(c) FOR - Test



(d) PDO - Test

Figure 6.7: Predator-Prey results with recurrent baselines.

6.7(c) and (d)) with $p < 0.03$). As in comparisons with the other algorithms for the Predator-Prey domain, our training results are not statistically significant, but the test results are statistically significant.

6.11 Ablation Study

In this section, we perform an ablation study by varying the neighbourhood distance and study the performance of the POMFQ (FOR) algorithm.

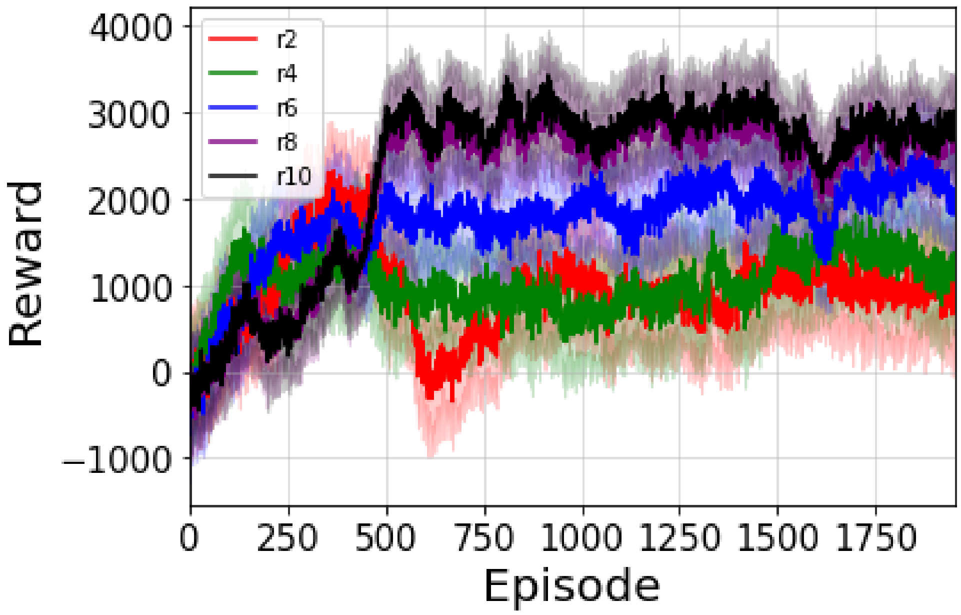


Figure 6.8: Multibattle (FOR case) with varying viewing distance.

Figure 6.8 shows the results of an ablation study where we change the distance of neighbourhood view in the Multibattle game. We study the neighbourhood viewable distances of 2 units, 4 units, 6 units, 8 units, and 10 units. These are denoted as r2, r4, r6, r8, and r10 respectively in Figure 6.8. Our results are averaged over 20 independent trials similar to the other experiments. The experiment is here is similar to our train experiments, where the POMFQ(FOR) algorithm competes against itself. In r2, groups A and B can see a distance of 2, and in r10, groups A and B can all see a distance of 10. When the average number of agents seen is higher (because the observation distance is higher), we expect the reward will be greater than or equal to the case where the average number of agents seen is lower due to the observation distance being lower. The Figure 6.8 shows that performance does steadily increase as we increase the neighbourhood distance. When more agents in the environment are seen by the central agent, more useful information is available about the environment — as expected, this impacts performance positively. As an example, once the agents are able to see a larger distance, they do need to indiscriminately employ the attack action that entails a penalty for usage against an empty neighbour (reward function in Appendix D.2).

It is important to note that these games are not zero sum games (reward functions are in Appendix D.2). We have empirically seen that when agents have access to limited information they quickly fall into sub-optimal performance based on the limited neighbourhood they can see. In this case, they are calculating their actions based on limited information and strategies are sub-optimal. When more information is available to the agents, their performance increases as they take into account a larger context before deciding best response strategies. In Figure 6.8, we see that there is almost no difference in performance between r8 and r10 — when the distance increases, the actions of agents at a large distance does not have a considerable impact on the performance of the central agent. The performance of the agent when the viewing distance is 2 units and 4 units are also not much different but overall, but the setting with distance 4 units just outperforms the setting with distance of 2 units ($p < 0.4$). The performance with distance 6 units is in between the performance with 4 units distance ($p < 0.2$) and 8 units distance ($p < 0.2$). Despite seeing the described performances, it should be noted that these differences are not statistically significant as $p < 0.2$. The difference between r2 and r10 has $p < 0.01$, which is statistically significant. Additionally, we see some unusual behaviour of unlearning in the case of r2. The agents could have started off being aggressive due to higher exploration but switched to more defensive strategies later on due to the availability of very less information. Nonetheless, this does not change the core message of this section.

6.12 Related Work

Mean field games (MFG) [1] were introduced by Lasry and Lions [136], extending mean field theory [205, 237] to the stochastic games framework. The stochastic games formulation was obtained by extending MDPs to MARL environments [104, 150]. Recent research has actively used the mean field games construct in a MARL setting, allowing tractable solutions in environments in which many agents participate. Model-based solutions have also been tried in this setting [127], but the model is specific to the application domain and these methods do not generalize well. Subramanian and Mahajan [243] analyze the problem using a stationary mean field. In contrast to our approach, this paper needs strict assumptions regarding this stationarity, which do not hold in practice. Mguni et al. [162] approaches this problem using the fictitious play technique. They provide strong theoretical properties for their algorithms, but only in the finite time horizon case. These results do not directly hold for infinite horizons. Additionally, strict assumptions on the reward function and fictitious property [19] assumption makes their algorithms less generally applicable. In fictitious play, each agent assumes that its opponents are playing stationary strategies.

[1]MFG is another approach to scaling MARL using the mean field, that is different from the MFRL technique we have focussed on so far. We discuss more about the MFG in the next chapter.

Thus, the response of each agent is a best response to the empirical frequency of their opponents. Another work by the same authors [164] introduces an algorithm and provides theoretical analysis for the mean field learning problem in cooperative environments. Our methods, on the other hand, work for both cooperative and competitive domains. Along the same lines, the work by Elie et al. [59, 60] contributes fictitious play based techniques to solve mean field games with general theoretical properties based on quantifying the errors accumulated at each time step. However, the strict assumptions on the reward function in addition to the fictitious play assumption is also present in the work by Elie et al. In our work, the agents do not make the fictitious play assumption for best responses. Yang et al. [303] do not have the limitations of other works noted here, but it assumes the global state is observable for all agents and a local action is taken from it. This has been relaxed by us.

6.13 Conclusion

This chapter considers many agent RL problems where the exact cumulative metrics regarding the mean field behaviour is not available and only local information is available. We used two variants of this problem and provided practical algorithms that work in both settings. We empirically showed that our approach is better than previous methods that used a simple aggregate of neighbourhood agents to estimate the mean field action. We theoretically showed that POMFQ stays close to the Nash Q under common assumptions.

Chapter 7

Decentralized Mean Field Games

As discussed earlier, using mean field theory to aggregate agents has been proposed as a solution to make MARL algorithms tractable in environments with many agents. However, many previous methods in this area make a strong assumption of a centralized system where all the agents in the environment can share policy parameters. Also, all agents are assumed to be effectively homogeneous and indistinguishable from each other. In this chapter, we relax this assumption about indistinguishable agents and propose a new mean field system known as *Decentralized Mean Field Games*, where each agent can be quite different from others [1]. All agents learn independent policies in a decentralized fashion, based on their local observations. We define a theoretical solution concept for this system and provide a fixed point guarantee for a Q-learning based algorithm in this system. A practical consequence of our approach is that we can address a 'chicken-and-egg' problem in empirical mean field reinforcement learning algorithms. Further, we provide Q-learning and actor-critic algorithms that use the decentralized mean field learning approach and give stronger performances compared to common baselines in this area. In our setting, agents do not need to be clones of each other and learn in a fully decentralized fashion. Hence, for the first time, we show the application of mean field learning methods in fully competitive environments, large-scale continuous action space environments, and other environments with heterogeneous agents. Importantly, we also apply the mean field method in a ride-sharing problem using a real-world dataset. We propose a decentralized solution to this problem, which is more practical than existing centralized training methods.

[1] Note that in Chapter 5 we considered a similar problem of extending mean field methods to heterogeneous agents using type classification. However, that approach required agents to be classified into a finite set of types where homogeneity assumptions are satisfied within types. We will need no such requirement in this chapter as we explain later.

The contribution of this chapter are summarized as follows:

- A new mean field paradigm, known as decentralized mean field games, that relaxes the homogeneity assumption, and subsequently centralized training requirements in prior mean field based settings. Each agent is independent and can be quite different from others.
- A decentralized solution concept for decentralized mean field games, along with fixed-point convergence guarantees for a Q-learning based tabular method in this setting.
- Resolution to a 'chicken-and-egg' problem in empirical mean field reinforcement learning algorithms.
- Practical Q-learning and actor-critic algorithms that use fully decentralized learning and show superior performances (as compared to strong MARL baselines) in a variety of many agent reinforcement learning testbeds along with a real-world ride-pool matching environment.

The core contents of this chapter were published in AAAI-2022 [253] and is available on arXiv [251].

7.1 Introduction

As discussed previously, most MARL algorithms are not tractable when applied to environments with many agents as these algorithms are exponential in the number of agents [37]. One exception is a class of algorithms that use the mean field theory [236] to approximate the many agent setting to a two agent setting, where the second agent is a mean field distribution of all agents representing the average effect of the population. This makes MARL algorithms tractable since, effectively, only two agents are being modelled. Lasry and Lions [136] introduced the framework of a *mean field game* (MFG), which incorporates the mean field theory in MARL. In MARL, the mean field can be a population state distribution [106] or action distribution [303] of all the other agents in the environment.

MFGs have three common assumptions. First, each agent does not have access to the local information of the other agents. However, it has access to accurate global information regarding the mean field of the population. Second, all agents in the environment are independent, homogeneous, and indistinguishable. Third, all agents maintain interactions with others only through the mean field. These assumptions (especially the first two) severely restrict the potential of using mean field methods in real-world environments. The first assumption is impractical, while the second assumption implies that all agents share the same state space, action space, reward function, and have the same objectives. Further, given these assumptions, prior works use centralized learning methods, where all agents learn and update a shared centralized policy. These two assumptions are only applicable to cooperative environments with extremely similar agents. Theoretically, the agent indices are omitted since all agents are interchangeable [136]. We will relax the first two assumptions. In our case, the agents are not interchangeable and can each formulate their own policies during learning that differs from others. Also, we will not assume the availability of the immediate global mean field. Instead, agents only have local information and use modelling techniques (similar to opponent modelling common in MARL [98]) to effectively model the mean field during the training process. We retain the assumption that each agent's impact on the environment is infinitesimal [106], and hence agents calculate best responses only to the mean field. Formulating best responses to each individual agent is intractable and unnecessary [136].

The solution concepts proposed by previous mean field methods have been either the centralized Nash equilibrium [303] or a closely related mean field equilibrium [136]. These solution concepts are centralized as they require knowledge of the current policy of all other agents or other global information, even in non-cooperative environments. Verifying their existence is infeasible in many practical environments [174].

This chapter presents a new kind of mean field system, *Decentralized Mean Field Games* (DMFGs), which uses a decentralized information structure. This new formulation of the mean field system relaxes the assumption of agents' indistinguishability and makes mean field methods applicable to numerous real-world settings. Subsequently, we provide a decentralized solution concept for learning in DMFGs, which will be more practical than the centralized solution concepts considered previously. We also provide a fixed point guarantee for a Q-learning based algorithm in this system.

A 'chicken-and-egg' problem exists in empirical mean field reinforcement learning algorithms where the mean field requires agent policies, yet the policies cannot be learned without the global mean field [304]. We show that our formulation can address this problem, and we provide practical algorithms to learn in DMFGs. We test our algorithms in different types of many agent environments. We also provide an example of a ride-sharing application that simulates demand and supply based on a real-world dataset.

7.2 Background

In this section we describe two different kinds of mean field systems (i.e., mean field games and mean field reinforcement learning) that we will use in this chapter. Though we have described the MFRL framework before, we choose to provide a brief description again since we subtly change notations for the sake of clarity in the context of this chapter. Also, we highlight some limitations in MFRL that pertains to this chapter (mainly the 'chicken-and-egg' problem).

Mean Field Game[2]: MFG was introduced as a framework to solve the stochastic game when the number of agents N is very large [108, 136]. In this setting, calculating the best response to each individual opponent is intractable, so each agent responds to the aggregated state distribution $\mu_t(s) \triangleq \lim_{N\rightarrow\infty} \frac{\sum_{j=1}^{N} \mathbf{1}(s_t^j = s)}{N}$, known as the mean field. Here $\mathbf{1}$ denotes the indicator function, i.e. $\mathbf{1}(x) = 1$, if x is true and 0 otherwise. Let $\boldsymbol{\mu} \triangleq \{\mu_t\}_{t=0}^{\infty}$. MFG assumes that all agents are identical (homogeneous), indistinguishable, and interchangeable [136]. Given this assumption, the environment changes to a single-agent stochastic control problem where all agents share the same policy [211]. Hence, $\pi^1 = \cdots = \pi^N = \boldsymbol{\pi}$. The theoretical formulation focuses on a representative player, and the solution of this player (optimal policy) obtains the solution for the entire system. The value function of a representative agent can be given as $V(s, \boldsymbol{\pi}, \boldsymbol{\mu}) \triangleq \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, \mu_t) | s_0 = s\right]$, where s and a are the state and action of the representative agent, respectively, and γ denotes the discount factor. The transition dynamics is represented as $P(s_t, a_t, \mu_t)$, where the dependence is on the mean field distribution. All agents share the same reward function $r(s_t, a_t, \mu_t)$. The action comes from the policy $\pi_t(a_t | s_t, \mu_t)$.

The solution concept for the MFG is the mean field equilibrium (MFE). A tuple $(\boldsymbol{\pi}^*_{MFG}, \boldsymbol{\mu}^*_{MFG})$ is a mean field equilibrium if, for any policy $\boldsymbol{\pi}$, an initial state $s \in \mathcal{S}$ and given mean field $\boldsymbol{\mu}^*_{MFG}$, $V(s, \boldsymbol{\pi}^*_{MFG}, \boldsymbol{\mu}^*_{MFG}) \geq V(s, \boldsymbol{\pi}, \boldsymbol{\mu}^*_{MFG})$. Additionally, $\boldsymbol{\mu}^*_{MFG}$ is the mean field obtained when all agents play the same $\boldsymbol{\pi}^*_{MFG}$ at each $s \in \mathcal{S}$.

Mean Field Reinforcement Learning (MFRL): As introduced previously, MFRL from Yang et al. [303] is another approach for learning in stochastic games with a large

[2]As noted in this section, MFG is another mean field framework which is different from the MFRL approach used in prior chapters. This dissertation mostly focuses on the MFRL paradigm since it does not contain the interchangeability assumption commonly seen in MFGs. Also, MFRL applies to settings with large but finite numbers of agents, while MFGs require infinite (in the limit) numbers of interchangeable agents, which is hard to satisfy in practical environments. However, in this chapter we touch upon MFG as well, since our proposed framework, DMFG, can be seen as extending both MFG and MFRL to the decentralized setting. Later in Section 7.12 we tabulate the differences between the different mean field settings.

number of agents. Here, the empirical mean action is used as the mean field, which each agent then uses to update its Q-function and Boltzmann policy. Each agent is assumed to have access to the global state, and it takes local action from this state. In MFRL, the Q-function of the agent j is updated as,

$$\begin{aligned} &Q^j(s_t, a_t^j, \mu_t^a) \\ &\quad = (1-\alpha)Q^j(s_t, a_t^j, \mu_t^a) + \alpha[r_t^j + \gamma v^j(s_{t+1})] \end{aligned} \tag{7.1}$$

where

$$v^j(s_{t+1}) = \sum_{a_{t+1}^j} \pi^j(a_{t+1}^j|s_{t+1}, \mu_t^a)Q^j(s_{t+1}, a_{t+1}^j, \mu_t^a) \tag{7.2}$$

$$\mu_t^a = \frac{1}{\mathcal{N}} \sum_j a_t^j, a_t^j \sim \pi^j(\cdot|s_t, \mu_{t-1}^a) \tag{7.3}$$

$$\pi^j(a_t^j|s_t, \mu_{t-1}^a) = \frac{\exp(-\hat{\beta}Q^j(s_t, a_t^j, \mu_{t-1}^a)}{\sum_{a_t^{j'} \in A^j} \exp(-\hat{\beta}Q^j(s_t, a_t^{j'}, \mu_{t-1}^a))} \tag{7.4}$$

where s_t is the global old state, s_{t+1} is the global resulting state, r_t^j is the reward of j at time t, v^j is the value function of j, $\mathcal{N}$ is the total number of agents, and $\hat{\beta}$ represents the Boltzmann parameter. Also, $\gamma \in [0, 1)$ denotes the discount factor. The action a^j is assumed to be discrete and represented using one-hot encoding. Like stochastic games, MFRL uses the NE as the solution concept. The global mean field μ_t^a captures the action distribution of all agents. To address this global limitation, Yang et al. [303] specify mean field calculation over certain neighbourhoods for each agent. However, an update using Eq. 7.3 requires each agent to have access to all other agent policies or the global mean field to use the centralized concept (NE). Also, as highlighted in Chapter 6, the neighbourhoods are not applicable in many practical environments. We omit an expectation in Eq. 7.2 since Yang et al. [303] guaranteed that their updates will be greedy in the limit with infinite exploration (GLIE).

From Eq. 7.3 and Eq. 7.4, it can be seen that the current action depends on the mean field and the mean field depends on the current action. To resolve this 'chicken-and-egg' problem, Yang et al. [303] simply use the previous mean field action to decide the current action, as in Eq. 7.4. This can lead to a loss of performance since the agents are formulating best responses to the previous mean field action μ_{t-1}^a, while they are expected to respond to the current mean field action μ_t^a.

7.3 Related Work

MFGs were first proposed in Huang et al. [107], while a comprehensive development of the system and principled application methods were given later in Lasry and Lions [136]. Subsequently, learning algorithms were proposed for this framework. Subramanian and Mahajan [243] introduce a restrictive form of MFG (known as stationary MFG) and provide a model-free policy-gradient [130, 258] algorithm along with convergence guarantees to a local Nash equilibrium. On similar lines, Guo et al. [88] provide a model-free Q-learning algorithm [291] for solving MFGs, also in the stationary setting. The assumptions in these works are difficult to verify in real-world environments. Particularly, as discussed previously, Guo et al. [88] assume the presence of a game engine (simulator) that accurately provides mean field information to all agents at each time step, which is not practical in many environments. Further, as discussed in the previous chapter, other works depend on fictitious play updates for the mean field parameters [60, 91], which involves the strong assumption that opponents play stationary strategies. All these papers use the centralized setting for the theory and the experiments.

Prior works [2, 84, 211] have established the existence of a (centralized) mean field equilibrium in the discrete-time MFG under a discounted cost criterion, in finite and infinite-horizon settings. Authors have also studied the behaviour of iterative algorithms and provided theoretical analysis for learning of the non-stationary (centralized) mean field equilibrium in infinite-horizon settings [8, 293, 294]. We provide similar guarantees in the decentralized setting with possibly heterogeneous agents.

Mean field games have also been used in the inverse reinforcement learning paradigm, where the objective is to determine a suitable reward function given expert demonstrations [300], and not decision-making as seen in traditional RL [256]. Other methods such as [40] perform a very computationally expensive computation using the full knowledge of the environment, which does not scale to large real-world environments. The work by [74] provides a mean field actor-critic algorithm along with theoretical analysis in the linear function approximation setting, unlike other works which only analyze the tabular setting [88, 303]. However, it is not clear if this algorithm is strong empirically since it does not contain empirical experiments that illustrate its performance. Further, this work considered the linear-quadratic setting that contains restrictions on the type of reward function. Mguni et al. [163] explore the connection between MARL and mean field games in the model-free setting.

The mean field setting under the cooperative case can be handled using a mean field control model [41]. In the linear-quadratic setting, [41] prove that policy-gradient methods converge to local equilibrium. In further work, the same authors prove that Q-learning algorithms also converge [42].

As discussed earlier, Yang et al. [303] introduces MFRL that uses a mean field approximation through the empirical mean action, and provides two practical algorithms that show good performance in large MARL settings. The approach is model-free, and the algorithms do not need strong assumptions regarding the nature of the environment. However, they assume access to global information that needs a centralized setting for both theory and experiments. While we extend MFRL to multiple types in Chapter 5 and partially observable environments in Chapter 6, our prior work in Chapter 5 assumes that agents can be divided into a finite set of types, where agents within a type are homogeneous (mean field approximation holds within types and does not need to hold across types). We do not need the agents to be grouped into a finite set of types in our work in this chapter. Additionally, while our work in Chapter 6 relaxes the assumption of global information in MFRL, it still uses the (centralized) Nash equilibrium as the solution concept. In this chapter, we will provide a decentralized solution concept. Further, that work contains some assumptions regarding the existence of conjugate priors, which is hard to verify in real-world environments. All of these are relaxed in this chapter.

In the experiments, we consider a real-world application of our methods on the Ride-Pool Matching Problem (RMP) as originally defined in Javier et al. [6]. This is the problem considered by top ride-sharing platforms such as UberPool and Lyft-Line. The problem studies the efficiency of accepting ride requests by individual vehicles in such a way that the vehicles make more money (cater to more requests) per trip and the ride-sharing platform improves its ability to serve more orders. Previous approaches to solving the RMP problem have used standard optimization techniques [208]. However, these methods are not scalable to large environments. One example from this class of methods is the zone path construction approach (ZAC) [155]. The ZAC solves for an optimization objective where the environment is abstracted into zones and each zone path represents a set of trips. The available vehicles are assigned to suitable zone paths using the ZAC algorithm. Another proposed solution was to make greedy assignments [6], which considers maximizing the immediate returns and not the long term discounted gains traditionally studied in RL. Prior work has also considered RL approaches for this problem [288, 299]. However, these works consider very restrictive settings, which do not model multi-agent interactions between the different vehicles. The work by Li et al. [146] used a mean field approach for order dispatching, however, it assumes vehicles serve only one order at a time. The recent work by Shah et al. [217] introduced a very general formulation for the RMP where vehicles of arbitrary capacity are designed to serve batches of requests, with the whole solution scalable to thousands of vehicles and requests. They introduced a Deep Q-Network (DQN) [167] based RL approach (called neural approximate dynamic programming or NeurADP) that learns efficient assignment of ride requests. However, the proposed approach assumes a set of identical vehicles and uses centralized training, where all vehicles learn the same policy using centralized updates. This is not practical in real-world environments, where the individual vehicles are typically heterogeneous (many differences in vehicular capacity and preferences). We propose a mean field based decentralized solution to this problem, which is more practical than the approach by Shah et al. [217].

7.4 Decentralized Mean Field Game

The DMFG model is specified by $\langle \boldsymbol{\mathcal{S}}, \boldsymbol{\mathcal{A}}, p, \boldsymbol{R}, \mu_0 \rangle$, where $\boldsymbol{\mathcal{S}} = \mathcal{S}^1 \times \cdots \times \mathcal{S}^N$ represents the state space and $\boldsymbol{\mathcal{A}} = \mathcal{A}^1 \times \cdots \times \mathcal{A}^N$ represents the joint action space. Here, $\mathcal{S}^j$ represents the state space of an agent $j \in \{1, \ldots, N\}$ and $\mathcal{A}^j$ represents the action space of j. As in MFGs, we are considering the infinite population limit of the game, where the set of agents N, satisfy $N \to \infty$. Similar to the MFG formulation in several prior works [108, 136, 211], we will specify that both the state and action spaces are Polish spaces. Particularly, $\mathcal{A}^j$ for all agents j, is a compact subset of a finite dimensional Euclidean space $\Re^d$ with the Euclidean distance norm $|| \cdot ||$. Since all agents share the same environment, for simplicity, we will also assume that the state spaces of all the agents are the same $\mathcal{S}^1 = \cdots = \mathcal{S}^N = \mathcal{S}$ and are locally compact. Since the state space is a complete separable metric space (Polish space), it is endowed with a metric d_X. The transition function $p : \mathcal{S} \times \mathcal{A}^j \times \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{S})$ determines the next state of any j given the current state and action of j, and the probability distribution of the state in the system (represented by the mean field). The reward function is represented as a set $\boldsymbol{R} = \{R^1, \ldots, R^N\}$, where, $R^j : \mathcal{S} \times \mathcal{A}^j \times \mathcal{P}(\mathcal{S}) \to [0, \infty)$ is the reward function of j.

Recall that a DMFG has two major differences as compared to MFG and MFRL. 1) DMFG does not assume that the agents are indistinguishable and homogeneous (agent indices are retained). 2) DMFG does not assume that each agent can access the global mean field of the system. However, each agents' impact on the environment is infinitesimal, and therefore all agents formulate best responses only to the mean field of the system (no per-agent modelling is required).

As specified, in DMFG, the transition and reward functions for each agent depends on the mean field of the environment, represented by $\boldsymbol{\mu} \triangleq (\mu_t)_{t \geq 0}$, with the initial mean field represented as μ_0. For DMFG the mean field can either correspond to the state distribution $\mu_t(s) \triangleq \lim_{N \to \infty} \frac{\sum_{j=1}^{N} \mathbf{1}(s_t^j = s)}{N}$, or the action distribution μ_t^a as in Eq. 7.3 (for discrete settings, or a mixture of Dirac measures for continuous spaces as used in Anahtarci et al. [8]). Without a loss of generality, we use the mean field as the state distribution μ_t (represented by $\mathcal{P}(\mathcal{S})$), as done in prior works [61, 136]. However, our setting and theoretical results will hold for the mean field as action distribution μ_t^a as well.

In the DMFG, each agent j will not have access to the true mean field of the system and instead use appropriate techniques to actively model the mean field through exploration. The agent j holds an estimate of the actual mean field represented by μ_t^j. Let $\boldsymbol{\mu}^j \triangleq (\mu_t^j)_{t\geq 0}$. Let, $\mathcal{M}$ be used to denote the set of mean fields $\{\boldsymbol{\mu^j} \in \mathcal{P}(\mathcal{S})\}$. A Markov policy for an agent j is a stochastic kernel on the action space $\mathcal{A}^j$ given the immediate local state (s_t^j) and the agent's current estimated mean field μ_t^j, i.e. $\pi_t^j : \mathcal{S} \times \mathcal{P}(\mathcal{S}) \to \mathcal{P}(\mathcal{A}^j)$, for each $t \geq 0$. Alternatively, a non-Markov policy will depend on the entire state-action history of game play. We will use Π^j to denote a set of all policies (both Markov and non-Markov) for the agent j. Let s_t^j represent the state of an agent j at time t and a_t^j represent the action of j at t. Then an agent j tries to maximize the objective function given by the following equation (where r^j denotes the immediate reward obtained by the agent j and $\beta \in [0, 1)$ denotes the discount factor),

$$J_{\boldsymbol{\mu}}^j(\pi^j) \triangleq \mathbb{E}^{\pi^j}[\textstyle\sum_{t=0}^{\infty} \beta^t r^j(s_t^j, a_t^j, \mu_t)]. \tag{7.5}$$

In line with prior works [248, 303], we assume that each agent's sphere of influence is restricted by its neighbourhood, where it conducts exploration. Using this assumption, we assert that, after a finite t, the mean field estimate will accurately reflect the true mean field for j, in its neighbourhood denoted as $\mathcal{N}^j$. This assumption specifies that agents have full information in their neighbourhood, and they can use modelling techniques to obtain accurate mean field information within the neighbourhood (also refer to flocking from Perrin et al. [188]).

Assumption 10. *There exists a finite time T and a neighbourhood $\mathcal{N}^j$, such that for all $t > T$, the mean field estimate of an agent $j \in 1, \ldots, N$ satisfies $(\forall s^j \in \mathcal{N}^j)$ $\boldsymbol{\mu}^j(s^j) = \boldsymbol{\mu}(s^j)$. Also, $\forall s^j \in \mathcal{N}^j$, we have, $p(\cdot|s_t^j, a_t^j, \mu_t^j) = p(\cdot|s_t^j, a^j, \mu_t)$ and $r^j(\cdot|s_t^j, a_t^j, \mu_t^j) = r^j(\cdot|s_t^j, a_t^j, \mu_t)$.*

Let us define a set $\Phi : \mathcal{M} \to 2^{\Pi}$ as $\Phi(\boldsymbol{\mu^j}) = \{\pi^j \in \Pi^j : \pi^j \text{ is optimal for } \boldsymbol{\mu^j}\}$. Conversely, for j, we define a mapping $\Psi : \boldsymbol{\Pi} \to \mathcal{M}$ as, given a policy $\pi^j \in \Pi^j$, the mean field state estimate $\boldsymbol{\mu}^j \triangleq \Psi(\pi^j)$ can be constructed as,

$$\mu_{t+1}^j(\cdot) = \int_{\mathcal{S}\times\mathcal{A}^j} p(\cdot|s_t^j, a_t^j, \mu_t)\mathcal{P}^{\pi^j}(a_t^j|s_t^j, \mu_t^j)\mu_t^j(s_t^j). \tag{7.6}$$

Here $\mathcal{P}^{\pi^j}$ is a probability measure induced by π^j. Later (in Theorem 16) we will prove that restricting ourselves to Markov policies is sufficient in a DMFG, and hence $\mathcal{P}^{\pi^j} = \pi^j$.

Now, we can define the *decentralized mean field equilibrium* (DMFE) which is the solution concept for this game.

Definition 13. *The decentralized mean field equilibrium of an agent j is represented as a pair $(\pi^j_*, \mu^j_*) \in \Pi^j \times \mathcal{M}$ if $\pi^j_* \in \Phi(\mu^j_*)$ and $\mu^j_* = \Psi(\pi^j_*)$. Here π^j_* is the best response to μ^j_* and μ^j_* is the mean field estimate of j when it plays π^j_*.*

The important distinction between DMFE and centralized concepts, such as NE and MFE, is that DMFE does not rely on the policy information of other agents. MFE requires all agents to play the same policy, and NE requires all agents to have access to other agents' policies. DMFE has no such constraints. Hence, this decentralized solution concept is more practical than NE and MFE. In Section 7.12, we summarize the major differences between the DMFG, MFG, and MFRL.

7.5 Theoretical Results

We provide a set of theorems that will first guarantee the existence of the DMFE in a DMFG. Further, we will show that a simple Q-learning update will converge to a fixed point representing the DMFE. We will borrow relevant results from prior works in centralized MFGs in our theoretical guarantees. Particularly, we aim to adapt the results and proof techniques in works by Saldi et al. [211], Lasry and Lions [136], and Anahtarci et al. [8] to the decentralized setting. Further, we will state and prove each of our theorems in this section. Our theorems will depend on some assumptions all of which we will formally state as part of the proofs. The proof of theorems will depend on smaller lemmas whose proofs are provided in Appendix E.1.

Similar to an existing result from centralized MFG [136], in the DMFG, restricting policies to only Markov policies would not lead to any loss of optimality. We use Π^j_M to denote the set of Markov policies for the agent j.

Theorem 16. *For any mean field, $\boldsymbol{\mu} \in \mathcal{M}$, and an agent $j \in \{1, \ldots, N\}$, we have,*

$$\sup_{\pi^j \in \Pi^j} J^j_{\boldsymbol{\mu}}(\pi^j) = \sup_{\pi^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\pi^j). \tag{7.7}$$

Proof. For the proof, we will follow the idea of using discounted occupancy measures, as discussed in Putterman [196]. Let us consider an agent $j \in \{1, \dots, N\}$. Let $\boldsymbol{\mu} \in \mathcal{M}$ and $\pi^j \in \Pi^j$ be arbitrary. Then from the Ionescu-Tulcea theorem (see Theorem 3.3 in Lattimore and Szepesvari [137]), a mean field distribution $\boldsymbol{\mu}$ on $\mathcal{S}$ and a policy π^j defines a unique probability measure $\mathcal{P}^{\pi^j}$ on $\mathcal{S} \times \mathcal{A}^j$.

We will begin with a definition of a discounted occupancy measure for some mean field $\boldsymbol{\mu}$. Consider an arbitrary policy $\pi^j \in \Pi^j$. Then, the discounted occupancy measure $\nu_{\boldsymbol{\mu}}^{\pi^j} \in \mathcal{F}(\mathcal{S} \times \mathcal{A})$ induced by this policy on the set $\mathcal{S} \times \mathcal{A}^j$ with mean field $\boldsymbol{\mu}$ is given by the equation

$$\nu^{\pi^j}(s^j, a^j|\mu) = \sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\pi^j}(s_t^j = s^j, a_t^j = a^j | \mu_t = \mu). \tag{7.8}$$

The discounted occupancy measure assigns a measure to the given state-action pair, which is the infinite discounted sum of the process that has a mean field of $\boldsymbol{\mu}$ and follows the policy π^j, with the agent j hitting at various times the state s^j and taking the action a^j at that state.

The discounted measure has a property that the value function can be represented as a product of the immediate reward and the discounted occupancy measure. To prove this, consider (from Eq. 7.5),

$$\begin{aligned}
J_{\boldsymbol{\mu}}^j(\pi^j) &= \mathbb{E}^{\pi^j}[\textstyle\sum_{t=0}^{\infty} \beta^t r^j(s_t^j, a_t^j, \mu_t)] \\
&= \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} \textstyle\sum_{t=0}^{\infty} \beta^t\, \mathbb{E}^{\pi^j}[r^j(s_t^j, a_t^j, \mu_t)\mathcal{I}(s^j = s_t^j, a^j = a_t^j, \mu_t = \mu)] \\
&\overset{1}{=} \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} \textstyle\sum_{t=0}^{\infty} \beta^t\, \mathbb{E}^{\pi^j}[r^j(s^j, a^j, \mu)\mathcal{I}((s_t^j = s^j, a_t^j = a^j, \mu_t = \mu)] \\
&\overset{2}{=} \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j, a^j, \mu) \textstyle\sum_{t=0}^{\infty} \beta^t\, \mathbb{E}^{\pi^j}[\mathcal{I}(s_t^j = s^j, a_t^j = a^j, \mu_t = \mu)] \\
&= \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j, a^j, \mu) \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\pi^j}(s_t^j = s^j, a_t^j = a^j | \mu_t = \mu) \\
&= \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j, a^j, \mu) \nu^{\pi^j}(s^j, a^j|\mu)
\end{aligned} \tag{7.9}$$

Here the notation $\mathcal{I}(x)$ is an indicator function, which equals 1 if the x is true and 0 if x is false. The sum of all indicators for all the state-action pairs at a given t equals 1. The step (1) uses this property of the indicator function, to drop the time indices of the reward function. The step (2) is from Lebesgue's dominated convergence theorem [18].

Now we state a lemma needed for our proof.

Lemma 36. *For an agent $j \in \{1, \dots, N\}$ and policy π^j, given mean field $\boldsymbol{\mu}$, there exists a Markov policy $\hat{\pi}^j \in \Pi^j_M$ such that*

$$\nu^{\hat{\pi}^j}(s^j, a^j|\mu) = \nu^{\pi^j}(s^j, a^j|\mu) \tag{7.10}$$

Now, we aim to show that $\sup_{\pi^j \in \Pi^j} J^j_{\boldsymbol{\mu}}(\pi^j) = \sup_{\pi^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\pi^j)$. Since $\Pi^j_M \subset \Pi^j$, we know that $\sup_{\pi^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\pi^j) \leq \sup_{\pi^j \in \Pi^j} J^j_{\boldsymbol{\mu}}(\pi^j)$. To show the equality we aim to prove $\sup_{\pi^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\pi^j) \geq \sup_{\pi^j \in \Pi^j} J^j_{\boldsymbol{\mu}}(\pi^j)$ as well.

To show this result we use Lemma 36. Consider a policy π^j and a Markov policy $\hat{\pi}^j$, such that $\nu^{\hat{\pi}^j}_{\boldsymbol{\mu}} = \nu^{\pi^j}_{\boldsymbol{\mu}}$. Now using the Eq. 7.9, we have

$$\begin{aligned}
J^j_{\boldsymbol{\mu}}(\pi^j) &= \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j, a^j, \mu)\nu^{\pi^j}(s^j, a^j|\mu) \\
&\overset{1}{=} \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j, a^j, \mu)\nu^{\hat{\pi}^j}(s^j, a^j|\mu) \\
&\leq \sup_{\hat{\pi}^j \in \Pi^j_M} \int_{(s^j \in \mathcal{S}, a^j \in \mathcal{A}^j)} r^j(s^j_t, a^j_t, \mu_t)\nu^{\hat{\pi}^j}(s^j, a^j|\mu) \\
&= \sup_{\hat{\pi}^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\hat{\pi}^j)
\end{aligned} \tag{7.11}$$

Here (1) is from Lemma 36. Now, from Eq. 7.11, and taking supermum on both sides, we get that $\sup_{\pi^j \in \Pi^j} J^j_{\boldsymbol{\mu}}(\pi^j) \leq \sup_{\pi^j \in \Pi^j_M} J^j_{\boldsymbol{\mu}}(\pi^j)$, which concludes our proof.

□

Next, we show the existence of a DMFE under a set of assumptions similar to those previously used in the centralized MFG [136, 211]. The assumptions pertain to bounding the reward function and imposing restrictions on the nature of the mean field (formal provided in the proof of Theorem 17). We do not need stronger assumptions than those previously considered for the MFGs.

Theorem 17. *An agent $j \in \{1, \ldots, N\}$ in the DMFG admits a decentralized mean field equilibrium $(\pi^j_*, \mu^j_*) \in \Pi^j \times \mathcal{M}$.*

Proof. The proof of this theorem follows the Theorem 1 in Saldi et al. [211]. We aim to extend it to the decentralized setting as mentioned in Section 7.5.

Before starting the proof, we will state some assumptions,

Assumption 11. *The reward function for all agents $j \in \{1, \ldots, N\}$ is bounded and continuous.*

Let us define a function $w : \mathcal{S} \to [1, \infty)$, such that there exists an increasing sequence of compact subsets $\{K_n\}_{n \geq 1}$ of $\mathcal{S}$ where

$$\lim_{n \longrightarrow \infty} \inf_{s \in \mathcal{S} \setminus K_n} w(s) = \infty. \tag{7.12}$$

That is, the value of $w(s)$ gets close to infinity in this limit. Here, the w can be regarded as a continuous moment function [211].

Assumption 12. *There exists a positive α such that the following holds (for all agents $j \in \{1, \ldots, N\}$),*

$$\sup_{(a^j, \mu^j) \in \mathcal{A}^j \times P(\mathcal{S})} \int_{\mathcal{S}} w(s^j_{t+1}) p(s^j_{t+1} | s^j_t, a^j_t, \mu_t) \leq \alpha w(s^j_t) \tag{7.13}$$

Also, consider two different moment functions w and v, then α satisfies,

$$\begin{aligned} &\sup\nolimits_{(a^j, \mu^j) \in \mathcal{A}^j \times P(\mathcal{S})} \Big| \int_{\mathcal{S}} w(s^j_{t+1}) p(s^j_{t+1} | s^j_t, a^j_t, \mu_t) - \int_{\mathcal{S}} v(s^j_{t+1}) p(s^j_{t+1} | s^j_t, a^j_t, \mu_t)) \Big| \\ &\leq \alpha |w(s^j_t) - v(s^j_t)| \end{aligned} \tag{7.14}$$

Assumption 13. *The initial mean field μ_0 satisfies*

$$\int_{\mathcal{S}} w(s^j) \mu_0(s^j) \triangleq M < \infty \tag{7.15}$$

Assumption 14. *There exist a $\gamma \geq 1$ and a positive scalar $\mathcal{R}$ such that for all $t \geq 0$, we define $M_t \triangleq \gamma^t \mathcal{R}$, then*

$$\sup_{(a^j, \mu) \in \mathcal{A}^j \times P(\mathcal{S})} r^j(s^j_t, a^j_t, \mu_t) \leq M_t w(s^j_t) \tag{7.16}$$

Now, we will state a few definitions required for our proof. For any function $g : \mathcal{S} \to \Re$, we define the w-norm as:

$$||g||_w \triangleq \sup_{s \in \mathcal{S}} \frac{|g(s)|}{w(s)} \tag{7.17}$$

Let $B_w(\mathcal{S})$ be the Banach space of all real valued measurable functions on $\mathcal{S}$ with a finite w-norm.

Also, for any signed measure μ in the space of $\mathcal{S}$, the w-norm can be defined as

$$||\mu||_w \triangleq \sup_{g \in B_w(\mathcal{S}) : ||g||_w \leq 1} \Big| \int_{\mathcal{S}} g(s)\mu(s) \Big| \tag{7.18}$$

Further, we define another set

$$\mathcal{T}_w(\mathcal{S}) \triangleq \{\mu \in \mathcal{P}(\mathcal{S}) : ||\mu||_w < \infty\} \tag{7.19}$$

Also, for $t \geq 0$, we define

$$\mathcal{T}_w^t(\mathcal{S}) \triangleq \Big\{\mu \in \mathcal{T}_w(\mathcal{S}) : \int_{\mathcal{S}} w(s)\mu(s) \leq \alpha^t M\Big\} \tag{7.20}$$

where the constant M is obtained from Assumption 13. Like our notation $\mathcal{P}(\mathcal{S})$, we are using the notation $\mathcal{P}(\mathcal{S} \times \mathcal{A}^j)$ to denote the probability of a state-action pair. Let us consider an element $\nu \in \mathcal{P}(\mathcal{S} \times \mathcal{A}^j)$, and use the notation $\nu_1 \triangleq \nu(\cdot \times \mathcal{A}^j)$ to denote the state marginal of ν. Now we define a new set,

$$\mathcal{T}_w^t(\mathcal{S} \times \mathcal{A}^j) \triangleq \Big\{\nu \in \mathcal{P}(\mathcal{S} \times \mathcal{A}^j) : \nu_1 \in \mathcal{T}_w^t(\mathcal{S})\Big\}. \tag{7.21}$$

For each $t \geq 0$, we will define

$$L_t \triangleq \sum_{k=t}^{\infty} (\beta\alpha)^{k-t} M_k \tag{7.22}$$

Here the constants α and M_k are obtained from Assumption 12 and Assumption 14 respectively.

It follows that the following equation holds,

$$L_t = M_t + (\beta\alpha)L_{t+1} \tag{7.23}$$

Let $C_w(S)$ denote the Banach space of all real valued bounded measurable functions on $\mathcal{S}$ with a finite w-norm.

Let us define a set

$$C_w^t(\mathcal{S}) \triangleq \{u \in C_w(\mathcal{S}) : ||u||_w \leq L_t\} \tag{7.24}$$

For an agent $j \in \{1, \ldots, N\}$, let us consider an operator

$$T_t^{\boldsymbol{\mu}} u^j(s_t) = \max_{a_t^j \in \mathcal{A}^j} \left[r(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} u^j(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \right] \tag{7.25}$$

where $u^j : \mathcal{S} \to \Re$

We will first provide the following lemma.

Lemma 37. *For all $t \geq 0$ and a given mean field $\boldsymbol{\mu}$, the operator $T_t^{\boldsymbol{\mu}}$ maps $C_w^{t+1}(\mathcal{S})$ into $C_w^t(\mathcal{S})$. Also, this operator will satisfy*

$$||T_t^{\boldsymbol{\mu}} u - T_t^{\boldsymbol{\mu}} x||_w \leq \alpha\beta ||u - x||_w \tag{7.26}$$

for any $u, x \in C_w(\mathcal{S})$.

Let us define a new set $\mathcal{C}$ from $C_w^t(\mathcal{S})$ as follows:

$$\mathcal{C} \triangleq \Pi_{t=0}^{\infty} C_w^t(\mathcal{S}) \tag{7.27}$$

Also, let us assign the following metric to $\mathcal{C}$:

$$\rho(\boldsymbol{u}, \boldsymbol{v}) \triangleq \sum_{t=0}^{\infty} \sigma^{-t} ||u_t - v_t||_w \tag{7.28}$$

where $\sigma > 0$ and the following assumption holds.

Assumption 15. *The variables σ, α, and β satisfy $\alpha\sigma\beta < 1$.*

This assumption guarantees that the metric $\mathcal{C}$ is complete w.r.t ρ, and $\rho(\boldsymbol{u}, \boldsymbol{v}) < \infty$ for all $\boldsymbol{u}, \boldsymbol{v} \in \mathcal{C}$.

Using the operators $\{T_t^{\boldsymbol{\mu}}\}_{t\geq 0}$, let us define a new operator $T^{\boldsymbol{\mu}} : \mathcal{C} \to \mathcal{C}$ as follows (for all $t \geq 0$).

$$(T^{\boldsymbol{\mu}}\boldsymbol{u})_t = T_t^{\boldsymbol{\mu}} u_{t+1} \tag{7.29}$$

From Lemma 37 we know that $T^{\boldsymbol{\mu}}$ is a well-defined operator that maps $\mathcal{C}$ to itself. Also, from Eq. 7.26, $T^{\boldsymbol{\mu}}$ is a contraction operator on $\mathcal{C}$ with constant of contraction $\alpha\beta\sigma < 1$ (from Assumption 15). Now, $T^{\boldsymbol{\mu}}$ will have a unique fixed point by the Banach fixed point theorem in $\mathcal{C}$.

Now, we move to proving another lemma. Consider a mean field $\boldsymbol{\mu}$, the optimal value function for an agent j can be given by

$$J_{*,t}^{j,\boldsymbol{\mu}}(s^j) = \sup_{\pi^j \in \Pi^j} \mathbb{E}^{\pi^j}\Big[\sum_{t=0}^{\infty} \beta^t r^j(s_t^j, a_t^j, \mu_t) | s_0^j = s^j\Big]. \tag{7.30}$$

Let $\boldsymbol{J}_*^{j,\boldsymbol{\mu}} \triangleq (J_{*,t}^{j,\boldsymbol{\mu}})_{t\geq 0}$ denote the optimal value function for an agent $j \in \{1, \dots, N\}$. This function is guaranteed to be continuous by Assumption 11.

Next, we provide another lemma which is a result on $\boldsymbol{\mu}$.

Lemma 38. *For any $\boldsymbol{\mu}$, the optimal point $\boldsymbol{J}_*^{j,\boldsymbol{\mu}}$, for an agent $j \in \{1, \dots, N\}$, belongs to the Banach space $\mathcal{C}$.*

Let us define another set $\mathcal{D}$ as

$$\mathcal{D} \triangleq \Pi_{t=0}^{\infty} \mathcal{T}_w^t(\mathcal{S} \times \mathcal{A}^j) \tag{7.31}$$

Before proving further results, we restate a result proved previously in non-homogeneous stochastic processes [101]. We will adapt the result to pertain to an agent $j \in \{1, \dots, N\}$. For any E-valued random element x, we use the notation $\mathcal{L}(x)$ to denote the distribution of x (i.e. $\mathcal{L}(x) \in P(E)$).

Consider a variable $\boldsymbol{\nu} \in \mathcal{D}$. From the operator $T^{\boldsymbol{\mu}}$, we will define another operator $T^{\boldsymbol{\nu}}$, where the relation is such that $\mu_t = \nu_{t,1}$. Recall that the subscript here refers to the state marginal of ν (refer Eq. 7.21). From the result of $T^{\boldsymbol{\mu}}$, we know that the operator $T^{\boldsymbol{\nu}}$ is well-defined and has a unique fixed point. Now, we can state the following result.

Lemma 39. *For an agent $j \in \{1, \dots, N\}$, for any $\boldsymbol{\nu} \in \mathcal{D}$, the collection of value functions $\boldsymbol{J}_{*}^{j,\boldsymbol{\nu}}$ is the unique fixed point of the operator $T^{\boldsymbol{\nu}}$. Furthermore, $\pi^j \in M$ is optimal if and only if*

$$\nu_t^{\pi^j}(\{(s_t^j, a_t^j) : r^j(s_t^j, a_t^j, \nu_{t,1}) + \beta \int_{\mathcal{S}} J_{*,t+1}^{j,\boldsymbol{\nu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \nu_{t,1}) = T_t^{\nu} J_{*,t+1}^{j,\boldsymbol{\nu}}(s_t^j)\}) = 1 \tag{7.32}$$

where $\nu_t^{\pi^j} = \mathcal{L}(s_t^j, a_t^j)$.

Now, let us define a set-valued mapping $\tau : \mathcal{D} \to 2^{|\mathcal{P}(\mathcal{S}\times\mathcal{A})^{\infty}|}$, for an agent j as follows:

$$\tau(\boldsymbol{\nu}) = C(\boldsymbol{\nu}) \cap B(\boldsymbol{\nu}) \tag{7.33}$$

where

$$\begin{aligned} &C(\boldsymbol{\nu}) \triangleq \{\boldsymbol{\nu}' \in \mathcal{P}(\mathcal{S} \times \mathcal{A}^j) : \nu'_{0,1} = \mu_0 \text{ and} \\ &\nu'_{t+1,1}(\cdot) = \int_{\mathcal{S}\times\mathcal{A}^j} p(\cdot | s^j, a^j, \nu_{t,1}) \nu_t(s^j, a^j)\} \end{aligned} \tag{7.34}$$

and

$$\begin{aligned} &B(\boldsymbol{\nu}) \triangleq \Big\{\boldsymbol{\nu}' \in \mathcal{P}(\mathcal{S} \times \mathcal{A}^j) : \forall t \geq 0, \\ &\nu'_t(\big\{(s_t^j, a_t^j) : r^j(s_t^j, a_t^j, \nu_{t,1}) + \beta \int_{\mathcal{S}} J_{*,t+1}^{j,\boldsymbol{\nu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \nu_{t,1})\big\} = 1\Big\} \end{aligned} \tag{7.35}$$

Now in the next result we show that, the image of $\mathcal{D}$ under τ is contained in $2^{\mathcal{D}}$.

Proposition 1. *For any $\nu \in \mathcal{D}$, we have $\tau(\nu) \subset \mathcal{D}$*

Proof. Fix a $\nu \in \mathcal{D}$. To prove the result, it is sufficient to prove that $C(\boldsymbol{\nu}) \subset \mathcal{D}$. Let $\boldsymbol{\nu}' \in C(\boldsymbol{\nu})$. We aim to prove by induction that $\nu'_{t,1} \in P_v^t(\mathcal{S})$ for all $t \geq 0$. The claim trivially holds for $t = 0$ as $\nu'_{0,1} = \mu_0$. Assume that the claim holds for t and consider $t+1$. We have

$$\begin{aligned}
&\int_{\mathcal{S}} w(s_t^j)\nu'_{t+1,1}(s_{t+1}^j) \\
&= \int_{\mathcal{S}\times\mathcal{A}} \int_{\mathcal{S}} w(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \nu_{t,1}) \nu_t(s_t^j, a_t^j) \\
&\leq \int_{\mathcal{S}} \alpha w(s_t^j) \nu_{t,1}(s_t^j) \\
&\leq \alpha^{t+1} M
\end{aligned} \tag{7.36}$$

The third step is from Assumption 12 and the last step is from the fact that $\nu_{t,1} \in P_v^t(\mathcal{S})$. Hence, we can conclude that $\nu'_{t+1,1} \in P_v^{t+1}(\mathcal{S})$. $\square$

Now we can say that $\boldsymbol{\nu} \in \mathcal{D}$ is a fixed point of τ if $\boldsymbol{\nu} \in \tau(\boldsymbol{\nu})$. The following lemma connects the mean field equilibrium and the fixed points of τ.

Lemma 40. *Suppose the set valued mapping τ has a fixed point $\boldsymbol{\nu^j} = (\nu_t^j)_{t\geq 0}$ for an agent $j \in \{1, \dots, N\}$. Consider a Markov policy for the agent j as $\pi^j = (\pi_t^j)_{t\geq 0}$, which is obtained by factoring as $\nu_t^j(s_t^j, a_t^j) = \nu_{t,1}^j(s_t^j)\pi_t^j(a_t^j|s_t^j)$, and let $\boldsymbol{\nu}_1^j = (\nu_{t,1}^j)_{t\geq 0}$. Then the pair $(\pi^j, \boldsymbol{\nu}_1^j)$ is a decentralized mean field equilibrium.*

From the Lemma 40, it can be seen that the set valued mapping (operator) τ having a fixed point is sufficient to guarantee the existence of the decentralized mean field equilibrium. Like several results in centralized multi-agent systems [172] and centralized mean field games [108, 136, 211], we will use the Kakutani's fixed point theorem [121], to guarantee the existence of a fixed point for the operator τ. This theorem requires the set on which the set valued mapping τ operates to be a non-empty, compact and convex subset of some Euclidean space. Further, the operator $\tau(\boldsymbol{\nu}^j)$ is required to be non-empty and convex for all $\boldsymbol{\nu}^j$. Finally, the operator τ should have a closed graph. Given these three conditions, we can conclude that τ has a fixed point using the Kakutani's fixed point theorem.

For the first condition, we need to show that the set on which $\boldsymbol{\nu}^j$ resides is non-empty, compact and convex, i.e. we need to show that $\mathcal{D}$ is non-empty, compact and convex. First, note that the function w can be expressed as a continuous moment function and hence the corresponding set $\mathcal{T}_v^t(\mathcal{S})$ is guaranteed to be compact [99]. As a consequence, the set $\mathcal{T}_v^t(\mathcal{S} \times \mathcal{A}^j)$ is tight, since the action space $\mathcal{A}^j$ is compact. Also, since the set $\mathcal{T}_v^t(\mathcal{S} \times \mathcal{A}^j)$ is closed, it is compact. Therefore, the set $\mathcal{D}$ is also compact. From Assumption 12 and

Assumption 14 we can show that a line segment between any two points in $\mathcal{D}$ lies in $\mathcal{D}$ and hence the set $\mathcal{D}$ is convex. These assumptions also guarantee that the set $\mathcal{D}$ is non-empty.

For the third condition, we need to show that $\tau(\boldsymbol{\nu}^j)$ is non-empty and convex for any $\boldsymbol{\nu}^j \in \mathcal{D}$. Now, from Lemma 39 we know that $B(\boldsymbol{\nu}^j)$ is non-empty and hence $\tau(\boldsymbol{\nu}^j)$ is non-empty. Also, we can show that each of the sets $C(\boldsymbol{\nu}^j)$ and $B(\boldsymbol{\nu}^j)$ is convex (see Saldi et al. [211]) and hence, their intersection is convex. This makes the set $\tau(\boldsymbol{\nu}^j)$ convex.

Next, we state another result before providing our final result.

Lemma 41. *Using Assumptions 11—15, the graph of τ, i.e. the set*

$$Gr(\tau) \triangleq \{(\boldsymbol{\nu}, \boldsymbol{\mathcal{E}}) \in \mathcal{D} \times \mathcal{D} : \boldsymbol{\mathcal{E}} \in \tau(\boldsymbol{\nu})\} \tag{7.37}$$

is closed.

Now, we are ready to give the final result.

Lemma 42. *Using Assumptions 11—15, for an agent $j \in \{1, \dots, N\}$, there exists a fixed point $\boldsymbol{\nu}^j$ of the set valued mapping $\tau : \mathcal{D} \to 2^{\mathcal{D}}$. Then, the pair $(\pi^j, \boldsymbol{\nu}_1^j)$ is a decentralized mean field equilibrium, where π^j is the policy of agent j and $\boldsymbol{\nu}_1^j$ is its mean field estimate constructed according to Lemma 40.*

The Theorem 17 follows from Lemma 42.

□

We use $\mathcal{C}$ to denote a set containing bounded functions in $\mathcal{S}$. Now, we define a decentralized mean field operator (H),

$$H : \mathcal{C} \times \mathcal{P}(\mathcal{S}) \ni (Q^j, \boldsymbol{\mu}^j) \to (H_1(Q^j, \boldsymbol{\mu}^j), H_2(Q^j, \boldsymbol{\mu}^j)) \in \mathcal{C} \times \mathcal{P}(\mathcal{S}) \tag{7.38}$$

where

$$H_1(Q^j, \boldsymbol{\mu}^j)(s_t^j, a_t^j) \triangleq r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} Q^j_{\max_{a^j}}(s_{t+1}^j, a^j, \mu_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \tag{7.39}$$

$$H_2(Q^j, \boldsymbol{\mu}^j)(\cdot) \triangleq \int_{\mathcal{S} \times \mathcal{A}^j} p\Big(\cdot | s_t^j, \pi^j(s_t^j, Q_t^j, \mu_t^j), \mu_t \Big) \mu_t^j(s) \tag{7.40}$$

for an agent j. Here, π^j is a maximiser of the operator H_1.

For the rest of the theoretical results, we consider a set of assumptions different from those needed for Theorem 17. Here we assume that the reward and transition functions are Lipschitz continuous with a suitable Lipschitz constant. The Lipschitz continuity assumption is quite common in the mean field literature [136, 248, 303]. We also consider some further assumptions regarding the nature of gradients of the value function. The formal statements for all the assumptions are provided as part of the proof of Theorem 18. All assumptions are similar to those considered before for the analysis in the centralized MFGs [8, 106, 136]. First, we provide a theorem regarding the nature of operator H. Then, we provide another theorem showing that H is a contraction.

Theorem 18. *The decentralized mean field operator H is well-defined, i.e., this operator maps $\mathcal{C} \times \mathcal{P}(\mathcal{S})$ to itself.*

Proof. In this and the subsequent theorems, we follow the theoretical results and proofs in the work by Anahtarci et al. [8], and extend them to the decentralized setting.

First, we will start with some definition for the norms, similar to our previous proofs. Consider an agent $j \in \{1, \ldots, N\}$. Let $w : \mathcal{S} \times \mathcal{A}^j \to \Re$ be a continuous weight function. For any measurable function $v : \mathcal{S} \times \mathcal{A}^j \to \Re$, the w-norm of v is defined as,

$$||v||_w \triangleq \sup_{s^j, a^j} \frac{|v(s^j, a^j)|}{w(s^j, a^j)}. \tag{7.41}$$

Also, for any measurable function $u : \mathcal{S} \to \Re$, the $w_{\max}$-norm is defined as

$$||v||_{w_{\max}} \triangleq \sup_{s^j} \frac{u(s^j)}{w_{\max}(s^j)}. \tag{7.42}$$

Now we will state a set of assumptions needed for our results.

Assumption 16. *For all agents $j \in \{1, \ldots, N\}$, the reward function is continuous, and it satisfies the following Lipschitz bounds (for some Lipschitz constants L_1 and L_2):*

$$||r^j(\cdot, \cdot, \mu_t) - r^j(\cdot, \cdot, \hat{\mu}_t)||_w \leq L_1 W_1(\mu_t, \hat{\mu}_t), \quad \forall \mu_t, \hat{\mu}_t \tag{7.43}$$

where W_1 is the Wasserstein distance of order 1. Also,

$$\sup_{(a^j_t, \mu_t) \in \mathcal{A}^j \times P(\mathcal{S})} |r^j(s^j_t, a^j_t, \mu_t) - r(\hat{s}^j_t, a^j_t, \mu_t)| \leq L_2 d_X(s^j_t, \hat{s}^j_t), \quad \forall s^j_t, \hat{s}^j_t \tag{7.44}$$

Assumption 17. *For an agent $j \in \{1, \ldots, N\}$, the transition function $p(\cdot|s^j, a^j, \mu)$ is weakly continuous in (s^j, a^j, μ) and satisfies the following Lipschitz bounds (for some Lipschitz constants K_1 and K_2):*

$$\sup_{s^j \in S} W_1(p(\cdot|s_t^j, a_t^j, \mu_t), p(\cdot|s_t^j, \hat{a}_t^j, \hat{\mu}_t)) \leq K_1(||a_t^j - \hat{a}_t^j|| + W_1(\mu_t, \hat{\mu}_t)), \quad \forall \mu_t, \hat{\mu}_t, \forall a_t^j, \hat{a}_t^j. \tag{7.45}$$

Also, we have,

$$\sup_{\mu \in P(S)} W_1(p(\cdot|s_t^j, a_t^j, \mu_t), p(\cdot|\hat{s}_t^j, \hat{a}_t^j, \mu_t)) \leq K_2(d_X(s_t^j, \hat{s}_t^j) + ||a_t^j - \hat{a}_t^j||), \quad \forall s_t^j, \hat{s}_t^j, \forall a_t^j, \hat{a}_t^j. \tag{7.46}$$

Assumption 18. *The action space $\mathcal{A}^j$ is convex for all agents $j \in \{1, \ldots, N\}$.*

Assumption 19. *For all agents j and all time t, there exist non-negative real numbers M and α such that for each $(s_t^j, a_t^j, \mu) \in \mathcal{S} \times \mathcal{A}^j \times \mathcal{P}(\mathcal{S})$, we have*

$$r^j(s_t^j, a_t^j, \mu) \leq M \tag{7.47}$$

$$\int_S w_{\max}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu) \leq \alpha w(s_t^j, a_t^j) \tag{7.48}$$

Assumption 20. *The product of constants $\beta\alpha < 1$.*

Let $C(\mathcal{S})$ denote the set of real-valued continuous functions on $\mathcal{S}$. Let $Lip(\mathcal{S})$ denote the set of all Lipshitz continuous function on $\mathcal{S}$, i.e.,

$$Lip(\mathcal{S}) \triangleq \{g \in C(\mathcal{S}) : ||g||_{Lip} < \infty\} \tag{7.49}$$

where $||g||_{Lip}$ is defined as

$$||g||_{Lip} \triangleq \sup_{(x,y) \in \mathcal{S} \times \mathcal{S}} \frac{|g(x) - g(y)|}{d_X(x, y)} \tag{7.50}$$

Here $g \in C(\mathcal{S})$. The finiteness of $||g||_{Lip}$ guarantees that g is Lipschitz continuous with constant $||g||_{Lip}$.

Also, let us define $B(\mathcal{S}, K)$ to be the set of all real valued measurable functions in $\mathcal{S}$ with $w_{\max}$-norm less than K. We use $Lip(\mathcal{S}, K)$ to denote the set of all Lipschitz continuous functions with $||g||_{Lip} < \infty$.

Finally, we define an operator $F : \mathcal{S} \times Lip(\mathcal{S}) \times \mathcal{P}(\mathcal{S}) \times \mathcal{A}^j \to \Re$ as

$$\begin{aligned} &F : \mathcal{S} \times Lip(\mathcal{S}) \times \mathcal{P}(\mathcal{S}) \times \mathcal{A}^j \ni (s_t^j, v_t^j, \mu_t, a_t^j) \\ &\to r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} v_t^j(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \in \Re \end{aligned} \tag{7.51}$$

Assumption 21. *Let $\mathcal{F}$ be the set of non-negative functions in*

$$Lip\Big(\mathcal{S}, \frac{L_2}{1-\beta K_2}\Big) \cap B\Big(\mathcal{S}, \frac{M}{1-\beta\alpha}\Big) \tag{7.52}$$

For an agent $j \in \{1, \dots, N\}$, for any $v \in \mathcal{F}$, $\mu \in \mathcal{P}(\mathcal{S})$, and $s^j \in S$, $F(s^j, v^j, \mu, \cdot)$ is ρ-strongly convex; that is $F(s^j, v^j, \mu, \cdot)$ is differentiable and its gradient ∇F satisfies

$$F(s_t^j, v_t^j, \mu_t, a_t^j) \geq F(s_t^j, v_t^j, \mu_t, \hat{a}_t^j) + \nabla F(s_t^j, v_t^j, \mu_t, \hat{a}_t^j)^T \cdot (a_t^j - \hat{a}_t^j) + \tfrac{\rho}{2}||a_t^j - \hat{a}_t^j||^2 \tag{7.53}$$

for some $\rho > 0$ and for all $a_t^j, \hat{a}_t^j \in \mathcal{A}^j$.

Assumption 22. *For an agent $j \in \{1, \dots, N\}$, the gradient $\nabla F(s_t^j, v_t^j, \mu_t, a_t^j) : \mathcal{S} \times \mathcal{F} \times \mathcal{P}(\mathcal{S}) \times \mathcal{A}^j \to \Re$ satisfies the following Lipshitz bound (for some Lipschitz constant K_F):*

$$\begin{aligned} &\sup_{a_t^j \in A^j} ||\nabla F(s_t^j, v_t^j, \mu_t, a_t^j) - \nabla F(\hat{s}_t^j, \hat{v}_t^j, \hat{\mu}_t, a_t^j)|| \\ &\leq K_F(d_X(s_t^j, \hat{s}_t^j) + ||v_t^j - \hat{v}_t^j||_{w_{\max}} + W_1(\mu_t^j, \hat{\mu}_t^j)) \end{aligned} \tag{7.54}$$

for every $s_t^j, \hat{s}_t^j \in \mathcal{S}; v^j, \hat{v}_t^j \in \mathcal{F}, \mu_t, \hat{\mu}_t \in \mathcal{P}(\mathcal{S})$.

Assumption 21 and Assumption 22 present a constraint on the change in the value function during each of the updates. These assumptions are generally considered in literature establishing the existence of Lyapunov functions that guarantee the presence of local and global optimal points (maxima or minima) as the case may be [266].

We also need to assume the following for all the constants,

Assumption 23. *The following relation holds*

$$k \triangleq \max\{\beta\alpha + \tfrac{K_F}{\rho}K_1, \beta\tfrac{L_2}{1-\beta K_2} + \Big(\tfrac{K_F}{\rho} + 1\Big)K_1 + K_2 + \tfrac{K_F}{\rho}\} < 1 \tag{7.55}$$

For the rest of the proof we will also apply Assumption 10. Consider a mean field $\boldsymbol{\mu}$, the optimal value function for an agent j is given by the Eq. 7.30. Using the same procedure as the result in Lemma 39, this optimal value function can be shown to be a unique fixed point of a Bellman operator $T_{\boldsymbol{\mu}}$ that is a contraction on the $w_{\max}$-norm with a constant of contraction $\beta\alpha$ (from Assumption 20). Now we can write the following equation,

$$\begin{aligned} &J_*^{j,\boldsymbol{\mu}}(s_t^j) = \max\nolimits_{a^j \in \mathcal{A}^j} \Big[r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} J_*^{j,\boldsymbol{\mu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \Big] \\ &\triangleq T_{\boldsymbol{\mu}} J_*^{j,\boldsymbol{\mu}}(s_t^j) \end{aligned} \tag{7.56}$$

Let us assume that this maximization is caused by a policy $\pi^j(a^j|s^j, \mu^j)$, which will be the optimal policy. In other words,

$$\begin{aligned} &\max\nolimits_{a^j \in \mathcal{A}^j} \Big[r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} J_*^{j,\boldsymbol{\mu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \Big] \\ &= r^j(s_t^j, \pi^j(a_t^j | s_t^j, \mu_t^j), \mu_t) + \beta \int_{\mathcal{S}} J_*^{j,\boldsymbol{\mu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \end{aligned} \tag{7.57}$$

The objective is to obtain this optimal policy. Towards the same we use the Q-functions, where the optimal Q-function can be defined as

$$Q_{\boldsymbol{\mu}}^{j,*}(s_t^j, a_t^j) = r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} J_*^{j,\boldsymbol{\mu}} p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \tag{7.58}$$

The optimal value function given by $J_*^{j,\boldsymbol{\mu}}$ is the maximum of the Q-function given by $Q_{\boldsymbol{\mu},\max}^{j,*}$. Hence, we can rewrite Eq. 7.58 as follows

$$\begin{aligned} &Q_{\boldsymbol{\mu}}^{j,*}(s_t^j, a_t^j) = r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} Q_{\boldsymbol{\mu},\max_{a^j}}^{j,*}(s_{t+1}^j, a^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \\ &\triangleq H Q_{\boldsymbol{\mu}}^{j,*}(s_t^j, a_t^j) \end{aligned} \tag{7.59}$$

Here, the operator H is the optimality operator for the Q-functions. Our objective is to prove that this operator is a contraction and has a unique fixed point given by Q_*.

First, let us define a set of all bounded Q-functions for an agent $j \in \{1, \ldots, N\}$:

$$\mathcal{C} \triangleq \Big\{ Q^j : \mathcal{S} \times \mathcal{A}^j \to [0,\infty); ||Q^j_{\max}||_w \leq \tfrac{M}{1-\beta\alpha}$$

$$\text{and} \tag{7.60}$$

$$||Q^j_{\max}||_{Lip} \leq \tfrac{L_2}{1-\beta K_2} \Big\}$$

From Assumption 10, Assumption 21 and Assumption 22, we know that there exists a unique maximiser $\pi^j(s^j_t, Q^j_t, \mu^j_t)$ for the function F represented as,

$$\begin{aligned} & r^j(s^j_t, a^j_t, \mu_t) + \beta \int_{\mathcal{S}} Q^j_{\boldsymbol{\mu},\max_{a^j}}(s^j_{t+1}, a^j) p(s^j_{t+1}|s^j_t, a^j_t, \mu_t) \\ & = F(s^j_t, Q^j_{\boldsymbol{\mu},\max}, \mu_t, a^j_t). \end{aligned} \tag{7.61}$$

.

This maximser makes the gradient of F, 0.

$$\nabla F(s^j_t, Q^j_{\boldsymbol{\mu},\max}, \mu_t, \pi^j(s^j_t, Q^j_t, \mu^j_t)) = 0 \tag{7.62}$$

This shows that the maximiser for the operator H_2 in Eq. 7.38 is unique, under the considered assumptions. Now, we are ready to prove the required theorem.

Now we are ready to prove our result. We first apply Assumption 10. Now, consider the decentralized mean field operator as defined in Eq. 7.38. To prove the given theorem, we know that $H_2(Q^j, \boldsymbol{\mu}) \in \mathcal{P}(\mathcal{S})$. We need to prove that $H_1(Q^j, \boldsymbol{\mu}) \in \mathcal{C}$. Let us consider an element $(Q^j, \boldsymbol{\mu}) \in \mathcal{C} \times \mathcal{P}(\mathcal{S})$. Then we have,

$$
\begin{aligned}
&\sup_{s_t^j,a_t^j} \frac{\left|H_1(Q^j,\boldsymbol{\mu})(s_t^j,a_t^j)\right|}{w(s_t^j,a_t^j)} \\
&= \sup_{s_t^j,a_t^j} \frac{\left|r^j(s_t^j,a_t^j,\mu)+\beta\displaystyle\int_{\mathcal{S}} Q^j_{\max}(s_{t+1}^j,a^j,\mu_{t+1}^j)p(s_{t+1}^j|s_t^j,a_t^j,\mu_t)\right|}{w(s_t^j,a_t^j)} \\
&\leq \sup_{s_t^j,a_t^j} \frac{\left|r^j(s_t^j,a_t^j,\mu)\right|}{w(s_t^j,a_t^j)} + \beta \sup_{s_t^j,a_t^j} \frac{\left|\displaystyle\int_{\mathcal{S}} Q^j_{\max}(s_{t+1}^j,a_t^j,\mu_{t+1})p(s_{t+1}^j|s_t^j,a_t^j,\mu_t)\right|}{w(s_t^j,a_t^j)} \\
&\leq M + \beta||Q^j_{\max}||_{w_{\max}} \sup_{s_t^j,a_t^j} \frac{\left|\displaystyle\int_{\mathcal{S}} w_{\max}(s_{t+1}^j)p(s_{t+1}^j|s_t^j,a_t^j,\mu_t)\right|}{w(s_t^j,a_t^j)} \\
&\leq M + \beta\alpha||Q^j_{\max}||_w \\
&\leq M + \beta\alpha\frac{M}{1-\beta\alpha} \\
&= \frac{M}{1-\beta\alpha}
\end{aligned}
\tag{7.63}
$$

We used the Assumption 19 and the fact that $||Q^j||_{w_{\max}} \leq ||Q^j||_w$.

Also, we have,

$$
\begin{aligned}
&|H_1(Q^j,\boldsymbol{\mu})_{\max}(s_t^j) - H_1(Q^j,\boldsymbol{\mu})_{\max}(\hat{s}_t^j)| \\
&= \max_{a^j\in\mathcal{A}^j}[r^j(s_t^j,a_t^j,\mu_t) + \beta\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s_t^j,a_t^j,\mu_t)] \\
&\qquad - \max_{a^j\in\mathcal{A}^j}[r^j(\hat{s}_t^j,a_t^j,\mu_t) + \beta\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|\hat{s}_t^j,a_t^j,\mu_t)] \\
&\leq \sup_{a^j\in\mathcal{A}^j}\Big|r^j(s_t^j,a_t^j,\mu_t) - r^j(\hat{s}_t^j,a_t^j,\mu_t)\Big| + \\
&\beta\sup_{a^j\in A^j}\Big|\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s_t^j,a_t^j,\mu_t) - \int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|\hat{s}_t^j,a_t^j,\mu_t)\Big| \\
&\leq L_2 d_X(s,\hat{s}) + \beta K_2||Q^j_{\max}||_{Lip}d_X(s,\hat{s}) \\
&\leq \tfrac{L_2}{1-\beta K_2}d_X(s,\hat{s})
\end{aligned}
\tag{7.64}
$$

Here we are using the Assumption 16 and Assumption 17. This proves that $H_1(Q^j,\mu)\in\mathcal{C}$ and concludes our proof.

□

Theorem 19. *Let $\mathcal{B}$ represent the space of bounded functions in $\mathcal{S}$. Then the mapping $H:\mathcal{C}\times\mathcal{P}(\mathcal{S})\to\mathcal{C}\times\mathcal{P}(\mathcal{S})$ is a contraction in the norm of $\mathcal{B}(\mathcal{S})$.*

Proof. In this proof, we will continue to use the set of assumptions and definitions we introduced in the proof of Theorem 18.

Fix any (Q^j,μ) and $(\hat{Q}^j,\hat{\mu})$ in $\mathcal{C}\times\mathcal{P}(\mathcal{S})$, let us consider,

$$
\begin{aligned}
&||H_1(Q^j,\mu) - H_1(\hat{Q}^j,\hat{\mu})||_w \\
&= \sup_{s_t^j,a_t^j} \Big[\frac{r^j(s_t^j,a_t^j,\mu_t)+\beta\displaystyle\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)}{w(s_t^j,a_t^j)} \\
&\qquad \frac{-r^j(s_t^j,a_t^j,\hat{\mu}_t)-\beta\displaystyle\int_{\mathcal{S}} \hat{Q}^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\hat{\mu}_t)}{w(s_t^j,a_t^j)}\Big] \\
&\le \sup_{s_t^j,a_t^j} \frac{|r^j(s_t^j,a_t^j,\mu_t)-r^j(s_t^j,a_t^j,\hat{\mu}_t)|}{w(s_t^j,a_t^j)} + \beta \sup_{s_t^j,a_t^j} \frac{\Big|\displaystyle\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)}{w(s_t^j,a_t^j)} \\
&\qquad \frac{-\displaystyle\int_{\mathcal{S}} \hat{Q}^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\hat{\mu}_t)\Big|}{w(s_t^j,a_t^j)} \\
&\le L_1 W_1(\mu,\hat{\mu}) + \beta \sup_{s_t^j,a_t^j} \frac{\Big|\displaystyle\int_{\mathcal{S}} Q^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)}{w(s_t^j,a_t^j)} \\
&\qquad \frac{-\displaystyle\int_{\mathcal{S}} \hat{Q}^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)\Big|}{w(s_t^j,a_t^j)} + \beta \sup_{s_t^j,a_t^j} \frac{\Big|\displaystyle\int_{\mathcal{S}} \hat{Q}^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)\Big|}{w(s_t^j,a_t^j)} \\
&\qquad \frac{-\displaystyle\int_{\mathcal{S}} \hat{Q}^j_{\max}(s^j_{t+1},a^j,\mu^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\hat{\mu}_t)\Big|}{w(s_t^j,a_t^j)} \\
&\le L_1 W_1(\mu,\hat{\mu}) + \beta||Q^j_{\max} - \hat{Q}^j_{\max}||_{w_{\max}} \sup_{s_t^j,a_t^j} \frac{\Big|\displaystyle\int_{\mathcal{S}} w_{\max}(s^j_{t+1})p(s^j_{t+1}|s^j_t,a^j_t,\mu_t)\Big|}{w(s_t^j,a_t^j)} \\
&+\beta||\hat{Q}^j_{\max}||_{Lip} \sup_{s_t^j,a_t^j} \frac{W_1(p(\cdot|s_t^j,a_t^j,\mu_t),p(\cdot|s_t^j,a_t^j,\hat{\mu}_t))}{w(s_t^j,a_t^j)} \\
&\le L_1 W_1(\mu,\hat{\mu}) + \beta\alpha||Q^j - \hat{Q}^j||_w + \beta\frac{L_2}{1-\beta K_2}K_1 W_1(\mu,\hat{\mu})
\end{aligned}
\tag{7.65}
$$

First we apply Assumption 16. In the last step we apply Assumption 17 and Assumption 19.

Now consider the distance between $H_2(Q^j, \boldsymbol{\mu})$ and $H_2(\hat{Q}^j, \hat{\boldsymbol{\mu}})$. First, we will consider the difference between the unique maximiser $\pi^j(s^j, Q^j, \mu^j)$ of $H_1(Q^j, \boldsymbol{\mu})(s^j, a^j)$ and the unique maximiser $\pi^j(s^j, \hat{Q}^j, \hat{\mu}^j)$ of $H_1(\hat{Q}^j, \hat{\boldsymbol{\mu}})(s^j, a^j)$ with respect to the action (using the Assumption 10). Let us consider the function,

$$\begin{aligned} &F : \mathcal{S} \times \mathcal{C} \times \mathcal{P}(\mathcal{S}) \times \mathcal{A}^j \ni (s_t^j, v_t^j, \mu_t, a_t^j) \\ &\to r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} v(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \in \Re \end{aligned} \tag{7.66}$$

This is ρ-strongly convex by Assumption 21, hence it satisfies the following equation [92],

$$[\nabla F(s_t^j, v_t^j, \mu_t, a_t^j + b_t^j) - \nabla F(s_t^j, v_t^j, \mu_t, a_t^j)]^T \cdot r_t^j \geq \rho ||b_t^j||^2 \tag{7.67}$$

For any $a_t^j, b_t^j \in \mathcal{A}^j$ and for any $s_t^j \in \mathcal{S}$, we can use the notation $a_t^j = \pi^j(s_t^j, Q_t^j, \mu_t^j)$ and $b^j = \pi^j(s_{t+1}^j, \hat{Q}_t^j, \hat{\mu}_t^j) - \pi^j(s_t^j, Q_t^j, \mu_t^j)$.

Now, $a_t^j = \pi^j(s_t^j, Q_t^j, \mu_t^j)$ is the unique maximiser of the strongly convex function $F(s_t^j, Q_{max,t}^j, \mu_t, \cdot)$, we have

$$\nabla F(s_t^j, Q_{max,t}^j, \mu_t, \pi^j(s_t^j, Q_t^j, \mu_t^j)) = 0 \tag{7.68}$$

Also, the term $a_t^j + b_t^j = \pi^j(s_{t+1}^j, \hat{Q}_t^j, \hat{\mu}_t^j)$ is the unique maximiser of $F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, \cdot)$. Now, using Assumption 21 and Eq. 7.67 we have,

$$\begin{aligned} &-\nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j)^T \cdot b_t^j \\ &\quad = -\nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j)^T \cdot b_t^j + \nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j + b_t^j)^T \cdot b_t^j \geq \rho ||b_t^j||^2 \end{aligned} \tag{7.69}$$

Similarly, using the Assumption 22 we also have,

$$\begin{aligned} &-\nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j)^T \cdot b_t^j \\ &= -\nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j)^T \cdot b_t^j + \nabla F(s_t^j, Q_{max,t}^j, \mu_t, a_t^j)^T \cdot b_t^j \\ &\leq ||b_t^j|| ||\nabla F(s_t^j, Q_{max,t}^j, \mu_t, a_t^j) - \nabla F(s_{t+1}^j, \hat{Q}_{max,t}^j, \hat{\mu}_t, a_t^j)|| \\ &\leq K_F ||b_t^j|| (d_X(s_t^j, s_{t+1}^j) + ||Q_{max,t}^j - \hat{Q}_{max,t}^j||_{w_{max}} + W_1(\mu_t, \hat{\mu}_t)) \\ &\leq K_F ||b_t^j|| (d_X(s_t^j, s_{t+1}^j) + ||Q_t^j - \hat{Q}_t^j||_w + W_1(\mu_t, \hat{\mu}_t)) \end{aligned} \tag{7.70}$$

Therefore, from the above two equations,

$$\begin{aligned}&\pi^j(s^j_{t+1},\hat{Q}^j_t,\hat{\mu}^j_t)-\pi^j(s^j_t,Q^j_t,\mu^j_t)\\&\leq \tfrac{K_F}{\rho}(d_X(s^j_{t+1},s^j_t)+||Q^j_t-\hat{Q}^j_t||_w+W_1(\mu_t,\hat{\mu}_t))\end{aligned} \tag{7.71}$$

Now, consider the W_1 distance between $H_2(Q^j,\boldsymbol{\mu})$ and $H_2(\hat{Q}^j,\hat{\boldsymbol{\mu}})$.

$$\begin{aligned}&W_1(H_2(Q^j,\boldsymbol{\mu}),H_2(\hat{Q}^j,\hat{\boldsymbol{\mu}}))\\&=\sup\nolimits_{||g||_{Lip}\leq 1}\Big|\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,Q^j_t,\mu^j_t),\mu_t\big)\mu_t(s^j_t)\\&-\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\hat{\mu}_t(s^j_t)\Big|\\&\leq \sup\nolimits_{||g||_{Lip}\leq 1}\Big|\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,Q^j_t,\mu^j_t),\mu_t\big)\mu_t(s^j_t)\\&-\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\mu_t(s^j_t)\Big|\\&+\sup\nolimits_{||g||_{Lip}\leq 1}\Big|\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\mu_t(s^j_t)\\&-\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\hat{\mu}_t(s^j_t)\Big|\\&\overset{(1)}{\leq}\int_{\mathcal{S}\times\mathcal{A}^j}\sup\nolimits_{||g||_{Lip}\leq 1}\Big|\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,Q^j_t,\mu^j_t),\mu_t\big)\mu_t(s^j_t)\\&-\int_{\mathcal{S}\times\mathcal{A}^j}\int_{\mathcal{S}}g(s^j_{t+1})p\big(s^j_{t+1}|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\Big|\mu_t(s^j_t)+(K_2+\tfrac{K_F}{\rho})W_1(\mu_t,\hat{\mu}_t)\\&\leq\int_{\mathcal{S}\times\mathcal{A}^j}W_1\Big(p\big(\cdot\,|s^j_t,\pi^j(s^j_t,Q^j_t,\mu^j_t),\mu_t\big),p\big(\cdot\,|s^j_t,\pi^j(s^j_t,\hat{Q}^j_t,\hat{\mu}^j_t),\hat{\mu}_t\big)\Big)\mu_t(s^j_t)\\&\qquad+\Big(K_2+\tfrac{K_F}{\rho}\Big)W_1(\mu_t,\hat{\mu}_t)\\&\overset{(2)}{\leq}\tfrac{K_F}{\rho}K_1\Big(||Q^j_t-\hat{Q}^j_t||_w+W_1(\mu_t,\hat{\mu}_t)\Big)+K_1W_1(\mu_t,\hat{\mu}_t)+\Big(K_2+\tfrac{K_F}{\rho}\Big)W_1(\mu_t,\hat{\mu}_t)\end{aligned} \tag{7.72}$$

In the above derivation, (1) and (2) are obtained from Assumption 17 and Eq. 7.71 (also refer Anahtarci et al. [8]). Combining Eq. 7.65 and Eq. 7.72, and applying Assumption 23, the required result is proved. The constant of contraction is k defined in Assumption 23.

□

Since H is a contraction, by the Banach fixed point theorem [224], we can obtain that fixed point using the Q iteration algorithm given by Alg. 13. We provide this result in the next theorem.

Algorithm 13 Q-learning for DMFG

```
1: For each agent $j \in \{1, \dots, N\}$, start with initial $Q$-function $Q^j_0$ and the initial mean
   field state estimate $\mu^j_0$
2: while $(Q^j_n, \mu^j_n) \neq (Q^j_{n-1}, \mu^j_{n-1})$ do
3:     $(Q^j_{n+1}, \mu^j_{n+1}) = H(Q^j_n, \mu^j_n)$
4: end while
5: Return the fixed point $(Q^j_*, \mu^j_*)$ of $H$
```

Theorem 20. *Let the Q-updates in Algorithm 13 converge to (Q^j_*, μ^j_*) for an agent $j \in \{1, \dots, N\}$. Then, we can construct a policy π^j_* from Q^j_* using the relation,*

$$\pi^j_*(s^j) = \arg\max_{a^j \in \mathcal{A}^j} Q^j_*(s^j, a^j, \mu^j_*). \tag{7.73}$$

Then the pair (π^j_*, μ^j_*) is a DMFE.

Proof. From the Theorem 19 we know that (Q^j_*, μ^j_*) is a fixed point of H. Using the Assumption 10, we consider a μ_*, where $\mu_*(s^j) = \mu^j_*(s^j)$, for all states $s^j \in \mathcal{N}^j$. Hence, we can construct the following equations.

$$Q^j_*(s^j_t, a^j_t, \mu^j_*) = r^j(s^j_t, a^j_t, \mu_{*,t}) + \beta \int_{\mathcal{S}} Q^j_{*,\max}(s^j_{t+1}, a^j, \mu^j_{*,t+1}) p(s^j_{t+1}|s^j_t, a^j_t, \mu_{*,t}) \tag{7.74}$$

$$\mu^j_{*,t+1}(\cdot) = \int_{\mathcal{S}\times\mathcal{A}^j} p(\cdot|s^j_t, a^j_t, \mu_{*,t}) \pi^j_*(a^j_t|s^j_t, \mu^j_{*,t}) \mu_{*,t}(s^j_t) \tag{7.75}$$

From Eq. 7.75 and Assumption 10, we can construct the mean field estimate of agent$j$, $\boldsymbol{\mu}^j_*$. Now, the above equations imply that $\pi^j_* \in \Phi(\mu^j_*)$ and $\mu^j_* = \Psi(\pi^j_*)$. Hence, (π^j_*, μ^j_*) is a decentralized mean field equilibrium.

□

Hence, from the above results, we have proved that a DMFG admits a DMFE, and an iterative algorithm using Q-updates can arrive at this equilibrium. This provides a fixed point for Algorithm 13, to which the Q-values converge.

7.6 Algorithms

We will apply the idea of decentralized updates to the model-free MFRL framework. We modify the update equations in MFRL (Section 7.2) and make them decentralized, where agents only observe their local state and do not have access to the immediate global mean field. Our new updates are:

$$Q^j(s_t^j, a_t^j, \mu_t^{j,a}) = (1-\alpha)Q^j(s_t^j, a_t^j, \mu_t^{j,a}) + \alpha[r_t^j + \gamma v^j(s_{t+1}^j)] \tag{7.76}$$

where

$$v^j(s_{t+1}^j) = \sum_{a_{t+1}^j} \pi^j(a_{t+1}^j | s_{t+1}^j, \mu_{t+1}^{j,a}) Q^j(s_{t+1}^j, a_{t+1}^j, \mu_{t+1}^{j,a}) \tag{7.77}$$

$$\mu_t^{j,a} = f^j(s_t^j, \hat{\mu}_{t-1}^{j,a}) \tag{7.78}$$

$$\text{and } \pi^j(a_t^j | s_t^j, \mu_t^{j,a}) = \frac{\exp(-\hat{\beta} Q^j(s_t^j, a_t^j, \mu_t^{j,a}))}{\sum_{a_t^{j'} \in A^j} \exp(-\hat{\beta} Q^j(s_t^j, a_t^{j'}, \mu_t^{j,a}))} \tag{7.79}$$

Here, s_t^j is the local state and $\mu_t^{j,a}$ is the mean field action estimate for the agent j at time t and $\hat{\mu}_{t-1}^{j,a}$ is the observed local mean field action of j at $t-1$. Other variables have the same meaning as Eq. 7.1 – Eq. 7.4. In Eq. 7.78, the mean field estimate for j is updated using a function of the current state and the previous local mean field. Opponent modelling techniques commonly used in MARL [98] can be used here. We use the technique of He and Graber [95], that used a neural network to model the opponent agent(s). In our case, we use a fully connected neural network (2 Relu layers of 50 nodes and an output softmax layer) to model the mean field action. The network takes the current state and previous mean field action as inputs and outputs the estimated current mean field. This network is trained using a mean square error between the estimated mean field action from the network (Eq. 7.78) and the observed local mean field (local observation of actions of other agents $\hat{\mu}_t^{j,a}$) after action execution. The policy in Eq. 7.79 does not suffer from the 'chicken-and-egg' problem of Eq. 7.4 since it depends on the current mean field estimate, unlike MFRL which used the previous global mean field in Eq. 7.4.

We provide a neural network-based Q-learning implementation for our update equations, namely *Decentralized Mean Field Game Q-learning* (DMFG-QL), and an actor-critic implementation, *Decentralized Mean Field Game Actor-Critic* (DMFG-AC). In the next section we provide detailed description of the algorithms. We also provide a complexity analysis in Section 7.8. The hyperparameter details are in Appendix E.3 and wall clock times for our main experiments with these algorithms are in Appendix E.4.

7.7 Algorithm Pseudo-Code

Algorithm 14 provides the pseudo-code of the DMFG-QL algorithm. Here, the neural networks are used as the function approximator for the Q-function. In addition, we include the use of a second target network and a replay buffer for training, as done in the popular Deep Q-learning (DQN) algorithm [167]. Also, similar to Yang et al. [303], we specify that the policies satisfy the GLIE assumption and hence we use the max operator over the Q-value of the next state (i.e. choose the action that maximizes the Q value) to calculate the temporal difference (T.D.) error instead of maintaining a distinct value function as in Eq. 7.77.

Algorithm 15 provides the pseudo-code of the DMFG-AC algorithm. This algorithm uses the policy as the actor, and the value function as the critic, as is common in practice [130]. We choose to use a stochastic actor that does not depend on the mean field. This is done to provide an algorithm that can work independent of the mean field during execution. Since the critic (not used during execution) can be dependent on the mean field, we additionally parameterize the value function with the mean field. The modified updates can be seen in Algorithm 15. For both Algorithm 14 and Algorithm 15 we initialize the estimated mean field distribution to a uniform distribution (arbitrarily).

7.8 Complexity Analysis

A tabular version of our DMFG-QL algorithm is guaranteed to be linear in the total number of states, polynomial in the total number of actions, and constant in the number of agents. These guarantees are the same for both space complexity and time complexity. Comparatively, a tabular version of mean field Q-learning (MFQ) algorithm from Yang et al. [303] has a space complexity that is linear in the number of states, polynomial in the number of actions and exponential in the number of agents. This is due to Eq. 7.3 requiring each agent to maintain Q-tables of all other agents. The time complexity is also the same

Algorithm 14 Q-learning for Decentralized Mean Field Games (DMFG-QL)

```
 1: Initialize the $Q$-functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, for all agents $j \in \{1, \ldots, N\}$
 2: Initialize the mean field estimate (parameterized by weights) $\mu^{j,a}_{\eta}$ for each agent $j \in \{1, \ldots, N\}$
 3: Initialize the estimated mean field $\mu^{j,a}_{0,\eta}$ for each $j$ to a uniform distribution
 4: Initialize the total steps (T) and total episodes (E)
 5: Obtain the current state $s^j_t$
 6: while Episode < E do
 7:     while Step < T do
 8:         For each agent $j$, obtain action $a^j_t$ from the policy induced by $Q_{\phi^j}$ with the current estimated mean action $\mu^{j,a}_{t,\eta}$ and the exploration rate $\hat{\beta}$ according to Eq. 7.79
 9:         Execute the joint action $\mathbf{a}_t = [a^1_t, \ldots, a^N_t]$. Observe the rewards $\mathbf{r}_t = [r^1_t, \ldots, r^N_t]$ and the next state $\mathbf{s}_{t+1} = [s^1_{t+1}, \ldots, s^N_{t+1}]$
10:         For each agent $j$, obtain the current observed local mean action $(\hat{\mu}^{j,a}_t)$
11:         Update the parameter $\eta$ of the mean field network using a mean square error between the observed mean action $\hat{\mu}^{j,a}_t$ and the current estimated mean action $\mu^{j,a}_{t,\eta}$ for all $j$
12:         For each $j$, obtain the mean field estimates $\mu^{j,a}_{t+1,\eta}$ for the next state $s^j_{t+1}$ using the mean field network according to Eq. 7.78
13:         For each $j$, store $\langle s^j_t, a^j_t, r^j_t, s^j_{t+1}, \mu^{j,a}_{t,\eta}, \mu^{j,a}_{t+1,\eta} \rangle$ in replay buffer $B$
14:         Set the next mean field estimate as the current mean field estimate $\mu^{j,a}_{t,\eta} = \mu^{j,a}_{t+1,\eta}$ and the next state as the current state $s^j_t = s^j_{t+1}$ for all agents $j \in \{1, \ldots, N\}$
15:     end while
16:     while $j = 1$ to $N$ do
17:         Sample a minibatch of K experiences $\langle s^j_t, a^j_t, r^j_t, s^j_{t+1}, \mu^{j,a}_{t,\eta}, \mu^{j,a}_{t+1,\eta} \rangle$ from $B$
18:         Set $y^j = r^j_t + \gamma \max_{a^j_{t+1}} Q_{\phi^j_-}(s^j_{t+1}, a^j_{t+1}, \mu^{j,a}_{t+1,\eta})$ according to Eq. 7.76
19:         Update the Q network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s^j_t, a^j_t, \mu^{j,a}_{t,\eta}))^2$
20:     end while
21:     Update the parameters of the target network for each agent $j$ with learning rate $\tau$;
                $$\phi^j_- \leftarrow \tau \phi^j + (1 - \tau) \phi^j_-$$
22: end while
```

Algorithm 15 Actor-Critic for Decentralized Mean Field Game (DMFG-AC)

```
1: Initialize the $V$-function or critic (parameterized by weights) $V_{\phi^j}$ and the policy or actor (parameterized by weights) $\pi_{\theta^j}$ for all agents $j \in 1, \dots, N$
2: Initialize the mean field estimate (parameterized by weights) $\mu^{j,a}_{\eta}$ for each agent $j \in \{1, \dots, N\}$
3: Initialize the estimated mean field $\mu^{j,a}_{0,\eta}$ for each $j$ to a uniform distribution
4: Initialize the total episodes (E)
5: Obtain the current state $s^j_t$
6: while Episode < E do
7:     For each agent $j$, obtain action $a^j_t$ from the policy $\pi_{\theta^j}$ at the current state $s^j_t$
8:     Execute the joint action $\mathbf{a}_t = [a^1_t, \dots, a^N_t]$. Observe the rewards $\mathbf{r}_t = [r^1_t, \dots, r^N_t]$ and the next state $\mathbf{s}_{t+1} = [s^1_{t+1}, \dots, s^N_{t+1}]$
9:     For each agent $j$, obtain the current observed local mean field action $(\hat{\mu}^{j,a}_t)$
10:    Update the parameter $\eta$ of the mean field network using a mean square error between the observed mean action $\hat{\mu}^{j,a}_t$ and the current estimated mean action $\mu^{j,a}_{t,\eta}$ for all $j$
11:    For each $j$, obtain the mean field estimates $\mu^{j,a}_{t+1,\eta}$ for the next state $s^j_{t+1}$ using the mean field network according to Eq. 7.78
12:    Set $y^j = r^j_t + \gamma V_{\phi^j}(s^j_{t+1}, \mu^{j,a}_{t+1,\eta})$ as the T.D. target according to Eq. 7.76
13:    For each $j$, update the critic by minimizing the loss $L(\phi^j) = (y^j - V_{\phi^j}(s^j_t, \mu^{j,a}_{t,\eta}))^2$
14:    For each $j$, update the actor using the log loss $\mathcal{J}(\theta^j) = \log \pi_{\theta^j}(a^j_t|s^j_t)L(\phi^j)$
15:    Set the next mean field estimate as the current mean field estimate $\mu^{j,a}_{t,\eta} = \mu^{j,a}_{t+1,\eta}$ and the next state as the current state $s^j_t = s^j_{t+1}$ for all agents $j \in \{1, \dots, N\}$
16: end while
```

as the space complexity in this case, since each entry in the table may need to be accessed in the worst case.

7.9 Experiments And Results

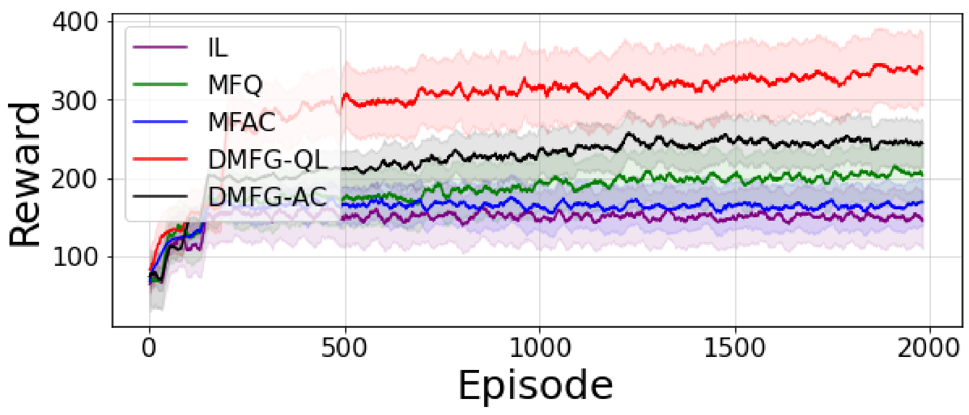
In this section we study the performance of our algorithms. The code for experiments has been open-sourced [246]. We provide the important elements of our domains here, while the complete details are in Appendix E.2. Though we use similar platforms as in prior chapters (MAgent, SISL) in our experiments here, our current settings have some differences as compared to those considered in previous chapters. These are captured in Appendix E.2.

The first five domains belong to the MAgent environment [317], we have used in

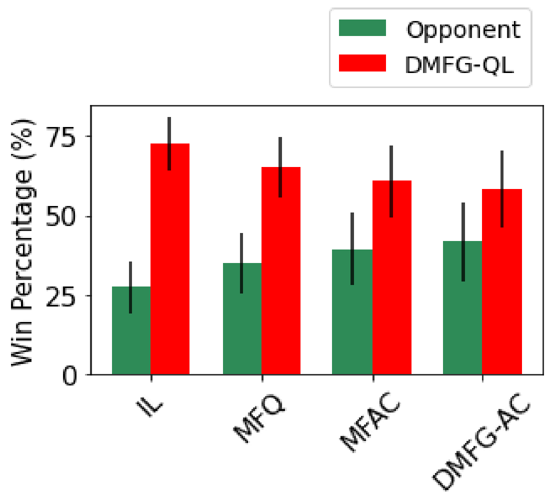
Chapters 5 and 6. We run the experiments in two phases, training and execution. Analogous to experiments conducted in previous mean field studies [248, 303], all agents train against other agents playing the same algorithm for 2000 games. This is similar to multi-agent training using self-play [222]. The trained agents then enter into an execution phase, where the trained policies are simply executed. The execution is run for 100 games, where algorithms may compete against each other. We consider three baselines, independent Q-learning (IL) [264], mean field Q-learning (MFQ) [303], and mean field actor-critic (MFAC) [303]. Importantly, each agent in our implementations learns in a decentralized fashion, where it maintains its own networks and learns from local experiences. This is unlike centralized training in prior works [88, 303]. We repeat experiments 30 times, and report the mean and standard deviation. Wall clock times are given in Appendix E.4.

First, we consider the mixed cooperative-competitive Battle game [317]. This domain consists of two teams of 25 agents each. Each agent is expected to cooperate with agents within the team and compete against agents of the other team to win the battle. For the training phase, we plot the cumulative rewards per episode obtained by the agents of the first team for each algorithm in Fig. 7.1(a). The performance of the second team is also similar (our environment is not zero-sum). From the results, we see that DMFG-QL performs best while the others fall into a local optimum and do not get the high rewards. The DMFG-AC algorithm comes second. It has been noted previously [303] that Q-learning algorithms often perform better compared to their actor-critic counterparts in mean field environments. MFQ and MFAC (using the previous mean field information) performs poorly compared to DMFG-QL and DMFG-AC (using the current estimates). Finally, IL loses out to others due to its independent nature. In execution, one team trained using one algorithm competes against another team from a different algorithm. We plot the percentage of games won by each algorithm in a competition against DMFG-QL and DMFG-AC. A game is won by the team that kills more of its opponents. The performances are in Fig. 7.1(b) and (c), where DMFG-QL performs best. In Section 7.11, we show that DFMG-QL can accurately model the true mean field in the Battle game.

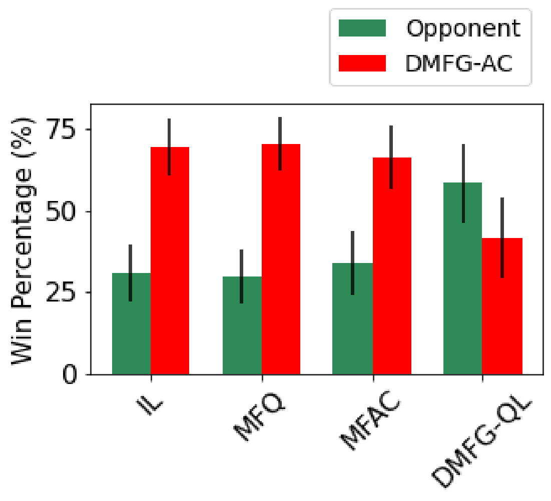
The second domain is the heterogeneous Combined Arms environment. This domain is a mixed setting similar to Battle except that each team consists of two different types of agents, ranged and melee, with distinct action spaces. Each team has 15 ranged and 10 melee agents. This environment is different from those considered in Chapter 5, which formulated each team as a distinct type, where agents within a team are homogeneous. The ranged agents are faster and attack further, but can be killed quickly. The melee agents are slower but are harder to kill. We leave out MFQ and MFAC for this experiment, since both these algorithms require the presence of fully homogeneous agents. The experimental procedure is the same as in Battle. From the results we see that DMFG-QL performs best



(a) Training



(b) Execution vs. DMFG-QL

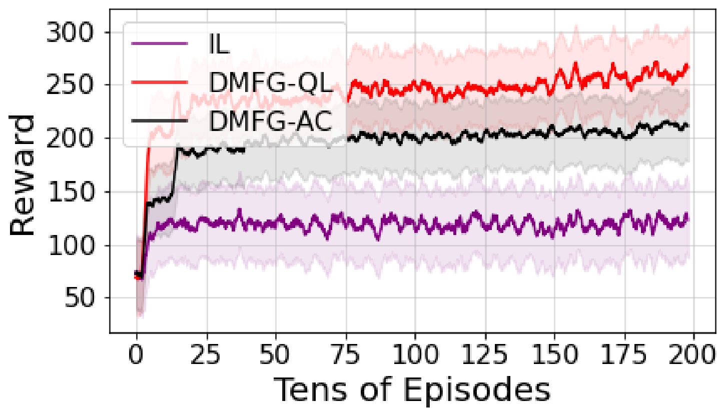


(c) Execution vs. DMFG-AC

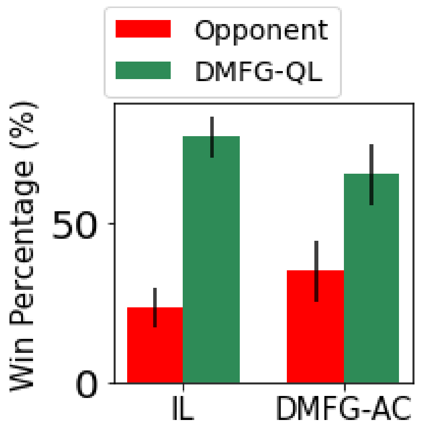
Figure 7.1: Battle results. In (a) solid lines show average and shaded regions represent standard deviation. In (b) and (c) bars are average and black lines represent standard deviation.

in both phases (see Fig. 7.2).

Next is the fully competitive Gather environment. This contains 30 agents trying to capture limited food. All the agents compete against each other for capturing food and could resort to killing others when the food becomes scarce. We plot the average rewards obtained by each of the five algorithms in the training phase (Fig. 7.3(a)). DMFG-QL once

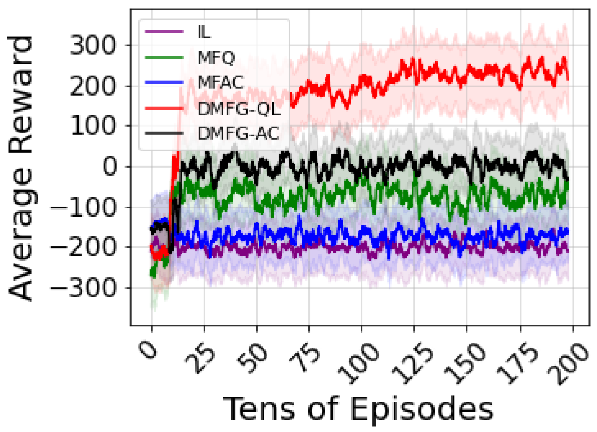


(a) Training

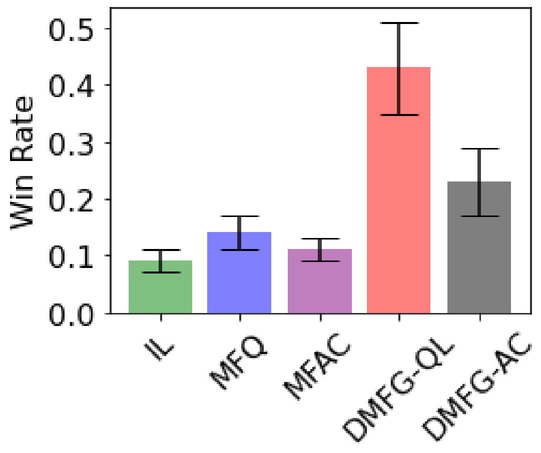


(b) Execution

Figure 7.2: Combined Arms results.



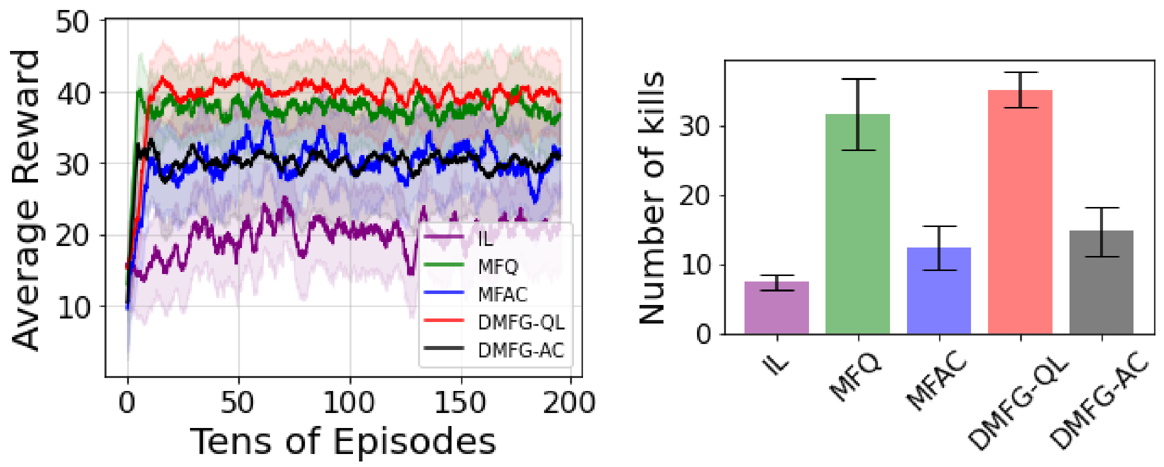
(a) Training



(b) Execution

Figure 7.3: Gather results.

again obtains the maximum performance. In competitive environments, actively formulating the best responses to the current strategies of opponents is crucial for good performances. Predictably, the MFQ and MFAC algorithms (relying on previous information) lose out. For execution, we sample (at random) six agents from each of the five algorithms to make a total of 30. We plot the percentage of games won by each algorithm in a total of 100 games. A game is determined to have been won by the agent obtaining the most rewards. Again, DMFG-QL shows the best performance during execution (Fig. 7.3(b)).

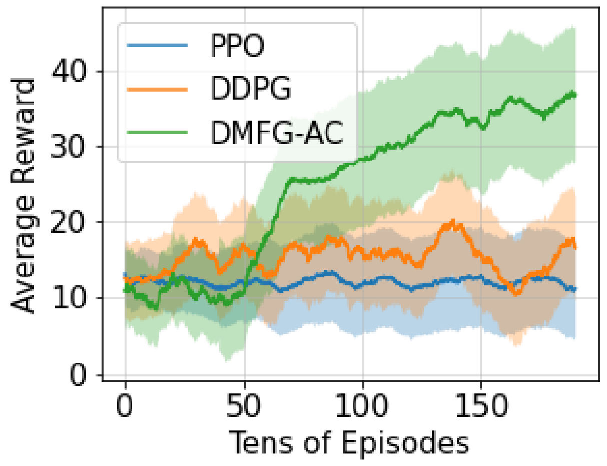


(a) Training (b) Execution

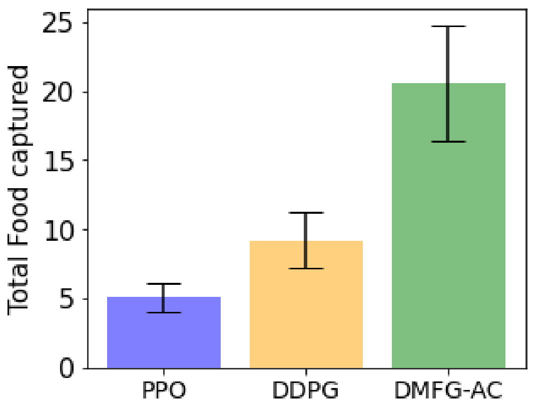
Figure 7.4: Tiger-Deer results.

The next domain is a fully cooperative Tiger-Deer environment. In this environment, a team of tigers aims to kill deer. The deer are assumed to be part of the environment moving randomly, while the tigers are agents that learn to coordinate with each other to kill the deer. At least two tigers need to attack a deer in unison to gain large rewards. Our environment has 20 tigers and 101 deer. In the training phase, we plot the average reward obtained by the tigers (Fig. 7.4(a)). The performance of MFQ almost matches that of DMFG-QL and the performance of DMFG-AC matches MFAC. In a cooperative environment, best responses to actively changing strategies of other agents are not as critical as in competitive environments. Here all agents aid each other and using the previous time information (as done in MFQ and MFAC) does not hurt performance as much. For execution, a set of 20 tigers from each algorithm execute their policy for 100 games. We plot the average number of deer killed by the tigers for each algorithm. DMFG-QL gives the best performance (Fig. 7.4(b)).

The next domain is the continuous action Waterworld domain, first introduced by Gupta et al. [89]. This is also a fully cooperative domain similar to Tiger-Deer, where a group of 25 pursuer agents aim to capture a set of food in the environment while actively avoiding poison. The action space corresponds to a continuous thrust variable. We implement DMFG-AC in this domain, where the mean field is a mixture of Dirac deltas of actions taken by all agents. The experimental procedure is the same as Tiger-Deer. For this



(a) Training

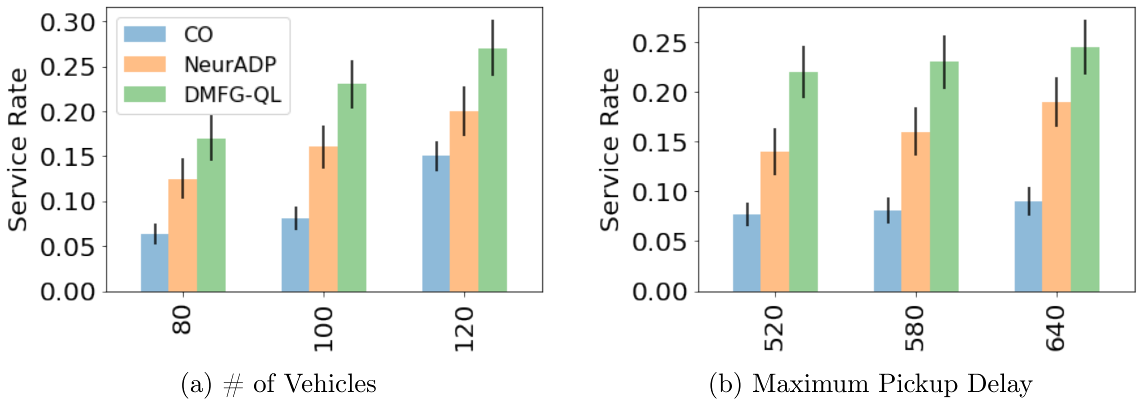


(b) Execution

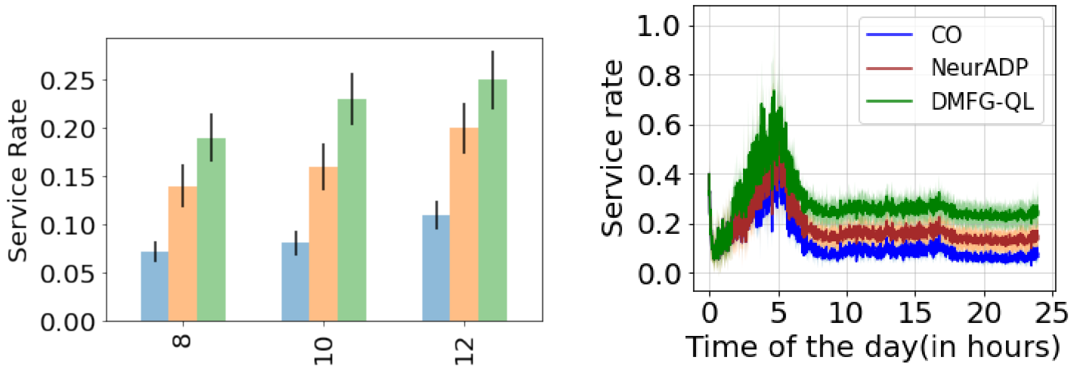
Figure 7.5: Waterworld results.

continuous action space environment, we use proximal policy optimization (PPO) [216] and deep deterministic policy gradient (DDPG) [147], as baselines. We see that DMFG-AC obtains the best performance in both phases (refer to Fig. 7.5(a) and (b)).

Our final environment is a real-world *Ride-pool Matching Problem* (RMP) introduced by Javier et al. [6]. This problem pertains to improving the efficiency of vehicles satisfying ride requests as part of ride-sharing platforms such as UberPool. In our environment, ride requests come from the open source New York Yellow Taxi dataset [181]. The road network (represented as a grid with a finite set of nodes or road intersections) contains a simulated set of vehicles (agents) that aim to serve the user requests. Further details about this domain are in Appendix E.2.6. We consider two baselines in this environment. The first is the method from Javier et al. [6], which used a constrained optimization (CO) approach to match ride requests to vehicles. This approach is hard to scale and is myopic in assigning requests (it does not consider future rewards). The second baseline is the Neural Approximate Dynamic Programming (NeurADP) method from Shah et al. [217], which used a (centralized) DQN algorithm to learn a value function for effective mapping of requests. This approach assumes all agents are homogenous (i.e., having the same capacity and preferences), which is impractical. To keep comparisons fair, we consider a decentralized version of NeurADP as our baseline. Finally, we implement DMFG-QL for this problem where the mean field corresponds to the distribution of ride requests at every node in the environment.



(a) # of Vehicles
(b) Maximum Pickup Delay



(c) Capacity
(d) Single Day Test

Figure 7.6: Results for the ride-sharing experiment. For (a), (b) and (c), we start with a prototypical configuration of c=10, τ = 580, and N = 100, and then vary the different parameters. Figures (a), (b) and (c) share the same legend given in (a).

Similar to prior approaches [155, 217], we use the service rate (total percentage of requests served) as the comparison metric. We train NeurADP and DMFG-QL using a set of eight consecutive days of training data and test all the performances in a previously unseen test set of six days. The test results are reported as an average (per day) of performances in the test set pertaining to three different hyperparameters. The first is the capacity of the vehicle (c) varied from 8 to 12, the second is the maximum allowed waiting time (τ) varied from 520 seconds to 640 seconds, and the last is the number of vehicles (N), varied from 80 to 120. The results in Figs. 7.6(a-c) show that DMFG-QL outperforms the baselines in all our test cases. The mean field estimates in DMFG-QL help predict the distribution of ride requests in the environment, based on which agents can choose ride requests strategically. If agents choose orders that lead to destinations with a high percentage of requests, they will be able to serve more requests in the future. Thus, DMFG-QL outperforms the NeurADP method (which does not maintain a mean field). We note that the service rate for all algorithms is low in this study (10% – 30%), since we are considering fewer vehicles compared to prior works [217] due to the computational requirements of being decentralized. In practice our training is completely parallelizable and this is not a limitation of our approach. Also, from Fig. 7.6(c), an increase in the number of vehicles, increases the service rate. In Fig. 7.6(d) we plot the performance of the three algorithms for a single test day (24 hours — midnight to midnight). During certain times of the day (e.g., 5 am), the ride demand is low, and all approaches satisfy a large proportion of requests. However, during the other times of the day, when the demand is high, the DMFG-QL satisfies more requests than the baselines, showing its relative superiority.

7.10 Statistical Significance Tests

We run a statistical significance test for all the main results in Section 7.9. In the MAgent environments, for the training results we conduct an unpaired two-sided t test at the last episode (2000) of training. For the execution results, we conduct a Fischer's exact test for the average performances. The tests are between the best performing algorithm (DMFG-QL or DMFG-AC) and the best performing baseline (IL, MFQ, MFAC). In the ride-sharing experiments we conduct an unpaired two-sided t test for the average and standard deviation of performances in Figure 7.6. The test is conducted between DMFG-QL and NeurADP (second best performing algorithm). We report the p-values for all the tests. As convention, we treat all p-values less than 0.05 as statistically significant outcomes.

From the results in Table 7.1, we see that the better performance given by our algorithms (DMFG-QL or DMFG-AC) is statistically significant in all domains except the Tiger-Deer domain. As noted in Section 7.9, the MFQ and MFAC algorithms perform as well as the DMFG-QL and DMFG-AC algorithms in this cooperative domain. Though we observe that DMFG-QL gives the best overall average performance in both training and execution in the Tiger-Deer environment, the results are not statistically significant as noted from the p-values in Table 7.1.

The statistical significance results for the ride-sharing experiment in Table 7.2 shows that our observations regarding the superior performance of DMFG-QL are statistically significant.

Experimental Domain	Training (p-value)	Testing (p-value)	Statistically Significant
Battle	$p < 0.01$	$p < 0.01$	Yes
Combined Arms	$p < 0.01$	$p < 0.01$	Yes
Gather	$p < 0.01$	$p < 0.01$	Yes
Tiger-Deer	$p < 0.5$	$p < 0.7$	No
Waterworld	$p < 0.01$	$p < 0.01$	Yes

Table 7.1: Statistical significance of our results in simulated experiments.

# of Vehicles (N)	Maximum Pickup Delay (τ) (sec)	Capa-city (c)	p-value	Statistically Significant
80	580	10	$p < 0.01$	Yes
100	580	10	$p < 0.01$	Yes
120	580	10	$p < 0.01$	Yes
100	520	10	$p < 0.01$	Yes
100	640	10	$p < 0.01$	Yes
100	580	8	$p < 0.01$	Yes
100	580	12	$p < 0.02$	Yes

Table 7.2: Statistical significance of our results in the ride-sharing domain.

7.11 Mean Field Modelling In DMFG-QL

In this section, we plot the mean square error between the estimated mean field action ($\mu_t^{j,a}$) (from the neural network representing Eq. 7.78) and the true observed local mean field action ($\hat{\mu}_t^{j,a}$) to show that the true local mean field action can be accurately modelled by the DMFG-QL algorithm. We will use the Battle game for this illustration.

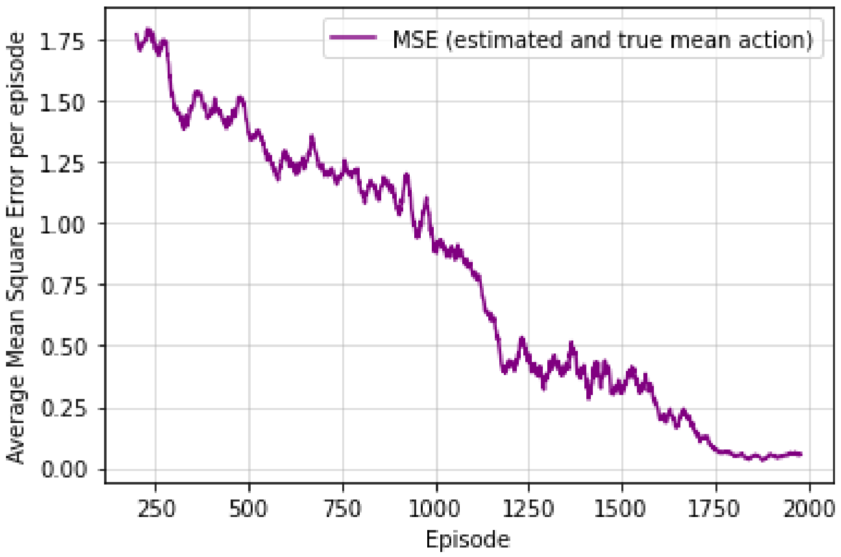


Figure 7.7: This figure shows the average (per agent) mean square error (MSE) between the true local mean field action and the estimated mean field action from the neural network in each episode (sum of MSE in the 500 steps) of the Battle game training experiment. The results show that the MSE steadily reduces, and hence DMFG-QL is able to accurately model the true mean field. The result shows the average of 10 runs (negligible standard deviation).

Fig. 7.7 shows the mean square errors (MSE) between the true mean field action and the estimated mean field action, averaged per agent, for each episode of the Battle game. The MSE converges to a small value close to 0 during training. This shows that the DMFG-QL algorithm can accurately model the mean field, which contributes to its superior performance in many of our experimental domains. As stated in Section 7.6, DMFG-QL algorithm formulates best responses to the current mean field action. This contrasts with approaches such as MFQ that formulate best responses to the previous mean field, which leads to a loss in performance. We start the plot in Fig 7.7 from episode 200 (instead of

0), since the first few episodes have a very high MSE, that simply skews the axis of the resulting graph.

7.12 Differences Between Mean Field Settings

Method	MFRL [303]	MFG [136]	DMFG (ours)
Game formulation	Stochastic Game	Mean Field Game	Decentralized Mean Field Game
Reward function	Same for all agents	Same for all agents	Can be different
Action space	Discrete only	Can be continuous	Can be continuous
Action space	Same for all agents	Same for all agents	Can be different
Solution Concept	Nash Equilibrium (centralized)	Mean Field Equilibrium (centralized)	Decentralized Mean Field Equilibrium
Complexity of obtaining the mean field	Exponential in agents	Exponential in agents	Constant in agents
Theoretical Guarantees	Needs very strong assumptions	Needs weak assumptions	Needs weak assumptions
Number of agents	Should be large but finite	Can be infinite in the limit	Can be infinite in the limit
Mean Field Information	Global	Global	Local
Agents	Identical and homogeneous	Identical and homogeneous	Non-identical
Policy	Stationary	Can be non-stationary	Can be non-stationary
State Observable	Global	Local	Local

Table 7.3: Table to capture the differences between Mean Field Reinforcement Learning (MFRL), Mean Field Games (MFG), and Decentralized Mean Field Games (DMFG).

Table 7.3 captures the differences between the three mean field frameworks, MFRL, MFG, and DMFG. Particularly, we show that the DMFG is different from other frameworks introduced previously, and it is more practical than prior methods as it relaxes some strong assumptions in them.

There are several disadvantages in using a centralized solution concept like Nash equilibrium (or, by extension, the mean field equilibrium) in multi-agent systems. These are listed as follows: 1) Centralized nature of the equilibrium, though practical systems have a decentralized information structure. 2) The equilibrium computation is almost intractable for more than two agents [174]. 3) Needs strong assumptions to give theoretical guarantees of convergence of learning systems in general sum games [104], even in the stationary case. 4) The equilibrium requires agreement between agents even in the competitive case. 5) It is hard to verify this point in practice. Our decentralized solution concept (DMFE) is more practical than the Nash equilibrium and the mean field equilibrium, since it does not suffer these limitations noted for the centralized methods.

7.13 Conclusion

In this chapter we relaxed two strong assumptions in prior work on using mean field methods in RL. We introduced the DMFG framework, where agents are not assumed to have global information and are not homogeneous. All agents learn in a decentralized fashion, which contrasts with centralized procedures in prior work. Theoretically, we proved that the DMFG will have a suitable solution concept, DMFE. Also, we proved that a Q-learning based algorithm will find the DMFE. Further, we provided a principled method to address the 'chicken-and-egg' problem in MFRL, and demonstrated performances in a variety of environments (including RMP).

Chapter 8

Mean Field Reinforcement Learning Using Attention and Action Advising

In this chapter we address the last set of limitations of mean field methods highlighted in Chapter 1. Specifically, we will focus on two limitations of mean field methods not addressed so far. First, using the mean field requires an assumption that all agents in a many-agent environment are extremely similar which results in each of them having the same level of importance on all other agents. However, in practice, some agents always matter more than the others for the learning of an individual agent. Second, similar to any reinforcement learning algorithm, mean field reinforcement learning approaches learn from scratch which is highly sample inefficient. In this chapter, we address both these limitations. We present a new framework, called mean field attention advising (MFAA), which is a mean field reinforcement learning technique that includes two novelties. First, this approach uses the recently introduced attention mechanism to dynamically attend to different agents based on their relative importance to the decision-making of a representative learning agent. Second, we use the previously introduced notion of action advising from external knowledge sources (aka 'advisors') to accelerate the training of mean field reinforcement learning approaches. We present a Q-learning and an actor-critic algorithm based on this framework. Empirically, we test these algorithms on several environments that support many agent learning and show that our algorithms provide superior performances compared to previous approaches, in addition to having other desirable properties like learning faster and being easier to scale to environments with larger numbers of agents.

The core contributions of this chapter are summarized as follows:

- Practical mean field reinforcement learning approach that relaxes the assumption of equal (or similar) other agent importance in prior works that use mean field based solutions. The approach uses a transformer architecture to determine relative importance of other agents nearby (local responses), and continues to use the mean field for global responses.

- Combining advantages of learning from advisors (Chapter 3) with mean field reinforcement learning to provide mean field based algorithms that can accelerate training using other sources of knowledge (instead of learning from scratch, only with environmental experiences).
- Extending the provided framework to heterogeneous agent environments using types as introduced in Chapter 5.
- Empirical illustrations of superior performances (as compared to strong MARL baselines) in a suite of many agent 'MAgent' testbeds and a massively multiplayer online (MMO) game environment.

8.1 Introduction

As mentioned previously, Yang et al. [303] introduced the mean field reinforcement learning (MFRL) framework that integrates the mean field approximation in stochastic games by using the empirical action distribution of all agents in the environment as the mean field. The MFRL framework allows tractable MARL solutions in environments with many agents. However, the MFRL framework contains two significant limitations that prevent its application in real-world environments.

First, MFRL requires all agents in the environment to have the same level of importance in the learning of a representative agent, called *central agent*. This requirement is justified using the fact that MFRL considers environments with very large numbers of homogeneous agents (possibly infinite in the limit), such that each agent's impact on the environment is negligible. However, in practice, environments only contain finite agents and the central agent is always impacted more by some agents than others. In this context, the mean field (that captures global information about the environment) is useful for long term planning, however, agents may be required to give higher importance to local information for short term decision-making. For example, in autonomous driving, a driving agent can plan ahead using the mean field, however, if faced with collision avoidance scenarios, it must provide more importance to nearby vehicles (as compared to vehicles further away). This difference in importance is not restricted to some notion of distance. An alternate example is that in business applications, product based companies plan longer term strategies based on global market dynamics, however, in the short term they may need to give more importance to the strategies of a particular set of close competitors (i.e., near in the sense of competition). The prior mean field based methods are incapable of reasoning at the agent level (in-fact it is assumed to be unnecessary) and hence are not applicable to many practical real-world environments.

Second, MFRL algorithms learn from scratch which is highly sample inefficient [225]. In general, RL (and MARL) algorithms have very poor sample efficiency [17, 267], and this problem is likely to be exacerbated in large environments containing many agents in them. Hence, it is essential to find alternate approaches that can accelerate the training of MFRL algorithms especially in the earlier stages of training.

In this chapter, we address both of these critical limitations that prevent MFRL applications in complex real-world problems. We introduce a novel MFRL framework, *mean field attention advising* (MFAA) that uses an attention mechanism to dynamically attend to different agents within a fixed importance set (finite set of agents close to the central agent), which addresses the first limitation. To address the second limitation, MFAA will use an approach that integrates knowledge from existing (potentially sub-optimal) policies which we broadly refer to as *advisors* as in Chapter 3 and Chapter 4. We will extend our approach for action advising in Chapter 3 (using a single online advisor) to the mean field setting. To recall, the advisor provides action advice that the agent can use for learning, which is helpful particularly at the earlier stages of learning. Further, using the MFAA framework, we provide two practical algorithms, one using a Q-learning [291] method and the other using an actor-critic [130] method. Additionally, using our prior work that extends MFRL methods to environments with heterogeneous agents using type classification (in Chapter 5), we extend MFAA to a Multi-Type MFAA (MTMFAA) framework that applies to environments with heterogeneous agents. Experimentally, we test our methods on several environments with a large number of agents, and show that our methods provide better performances than standard MARL baselines for many agent learning, in addition to having other advantages such as showing faster training and being robust to scaling to environments with larger numbers of agents. Particularly, using the Neural MMO simulator [242], we show an application of our method in massively multi-player online games (MMOs) which are currently a multi-billion dollar industry [238].

8.2 Background

The work in this chapter we will depend upon the MFRL approach from Yang et al. [303] and the MTMFRL approach from Chapter 5. We will restate the update equations again for easy referencing. Importantly, in this chapter we do away with the idea of neighbourhoods, seen in Yang et al. [303] and in Chapter 5. As discussed in Chapter 6 and Chapter 7, for many practical environments, the neighbourhood should correspond to the entire environment in the case of MFRL, hence, this term loses its significance.

Mean field reinforcement learning: Yang et al. introduce the mean field Q-learning (MFQ) method that is recursively updated using Eqs. 8.1 – 8.4:

$$Q^j(s_t, a^j_t, \overline{a}^j_t) = (1-\alpha)Q^j(s_t, a^j_t, \overline{a}^j_t) + \alpha[r^j_t + \gamma v^j(s_{t+1})] \tag{8.1}$$

$$\text{where } v^j(s_{t+1}) = \sum_{a^j_{t+1}} \pi^j(a^j_{t+1}|s_{t+1}, \overline{a}^j_t)Q^j(s_{t+1}, a^j_{t+1}, \overline{a}^j_t) \tag{8.2}$$

$$\overline{a}^j_t = \frac{1}{N}\sum_{k\neq j} a^k_t, a^k_t \sim \pi^k(\cdot|s_t, \overline{a}^k_{t-1}) \tag{8.3}$$

$$\text{and } \pi^j(a^j_t|s_t, \overline{a}^j_{t-1}) = \frac{\exp(-\hat{\beta}Q^j(s_t, a^j_t, \overline{a}^j_{t-1}))}{\sum_{a^{j'}_t \in A^j} \exp(-\hat{\beta}Q^j(s_t, a^{j'}_t, \overline{a}^j_{t-1}))}. \tag{8.4}$$

All notations have the same meaning as provided in Chapter 6. To recall, r^j_t represents the reward for the agent j, s_t and s_{t+1} represents the state at time steps t and $t+1$ respectively, α is the learning rate, v^j represents the value function of the agent j, $\gamma \in [0, 1)$ is the discount factor, and $\hat{\beta}$ is the Boltzmann parameter. Notably, a^j_t denotes the (discrete) action of the agent j represented as a one-hot encoding whose components are one of the actions in the action space. The notation $\overline{a}^j_t$ represents the mean action of all other agents apart from j (i.e., mean field) and π denotes the Boltzmann policy. Further, Yang et al. provide two practical algorithms, mean field Q-learning (MFQ) and mean field actor-critic (MFAC) that use function approximation for the Q-functions and a replay buffer for training as introduced in the well known Deep Q-learning (DQN) algorithm [167].

Multi-Type Mean Field Reinforcement Learning: The homogeneity assumptions in MFRL are strong and is not very applicable in real-world environments as these environments usually contain diverse sets of agents that are different in objectives and abilities. In this context, in Chapter 5 we extend the MFRL framework to environments that may contain heterogeneous agents. The approach groups agents into a finite set of *types*, where these types are constructed in such a way that the mean field assumption of homogeneity is valid within types but not applicable across types. Assuming the presence of M agent types, the mean field update equations in Eqs. 8.1 – 8.4 are replaced with the multi-type mean field (MTMF) updates, given as follows:

$$Q^j(s_t, a^j_t, \overline{a}^j_{1,t}, \ldots, \overline{a}^j_{M,t}) = (1-\alpha)Q^j(s_t, a^j_t, \overline{a}^j_{1,t}, \ldots, \overline{a}^j_{M,t}) + \alpha[r^j_t + \gamma v^j(s_{t+1})] \tag{8.5}$$

$$\text{where } v^j(s_{t+1}) = \sum_{a^j_{t+1}} \pi^j(a^j_{t+1}|s_{t+1}, \overline{a}^j_{1,t}, \ldots, \overline{a}^j_{M,t})Q^j(s_{t+1}, a^j_{t+1}, \overline{a}^j_{1,t}, \ldots, \overline{a}^j_{M,t}) \tag{8.6}$$

$$\overline{a}^j_{i,t} = \frac{1}{N_i}\sum_{k\neq j} a^k_{i,t}, a^k_{i,t} \sim \pi^k(\cdot|s_t, \overline{a}^k_{1,t-1}, \ldots, \overline{a}^k_{M,t-1}) \tag{8.7}$$

$$\text{and } \pi^j(a^j_t|s_t, \overline{a}^j_{1,t-1}, \ldots, \overline{a}^j_{M,t-1}) = \frac{\exp(-\hat{\beta}Q^j(s_t, a^j_t, \overline{a}^j_{1,t-1}, \ldots, \overline{a}^j_{M,t-1}))}{\sum_{a^{j'}_t \in A^j} \exp(-\hat{\beta}Q^j(s_t, a^{j'}_t, \overline{a}^j_{1,t-1}, \ldots, \overline{a}^j_{M,t-1}))}. \tag{8.8}$$

Here, the notation $\overline{a}^j_{i,t}$ denotes the mean field of the agent j belonging to the type i at time t. The variable N_i denotes the number of agents belonging to type i. All the other variables have the same meaning as in Eqs. 8.1 – 8.4. Further, we provided a practical algorithm, multi-type mean field Q-learning (MTMFQ), that uses function approximation (similar to the approach in MFQ), and show that this algorithm out-performs MFQ in different many agent heterogeneous environments.

8.3 Related Work

In this section, we provide a brief survey of literature in mean field learning, attention mechanism, and action advising in RL (and MARL) that is most relevant to our work. Regarding mean field methods and action advising, we will keep the discussion brief since we have already provided exhaustive details in the previous chapters.

The mean field reinforcement learning (MFRL) paradigm introduced by Yang et al. [303] applies MARL to environments with many agents by using the empirical action distribution as the mean field. However, all MFRL based approaches in the literature are sample inefficient (since they train from scratch) and are not widely applicable due to an assumption that requires all agents to have equal importance in the learning of the central agent. We address both these limitations in this chapter. Notably, Yang et al. [303] requires fully homogeneous agents (having same state space, action space, and reward functions). In Chapter 5, we extended [303] to multiple types which is applicable to environments with heterogeneous agents.

The attention mechanism [282] mimics the differentiable key-value memory model typically used in database retrieval [85, 182]. Attention models help in selecting relevant information for decision-making and have been successful in a variety of applications including vision [12] and natural language processing [13]. They have also been used in single-agent RL to enable a training agent to attend to different views in the observation space [16] and for explainability (interpretation) of the learned policies [168]. In MARL, the attention mechanism has been explored in the context of fully centralized training, for learning to attend to different agents [47] and learning communication in cooperative settings [117]. Attention models have been used with policy gradient approaches in MARL, where Shariq and Fei [110] propose a centralized training and decentralized execution (CTDE) method [156], called multi-actor-attention-critic (MAAC), where the central critic uses an attention mechanism to select relevant information about other agents at each time step. While powerful, this approach still has a linear dependence on the number of agents, which contrasts with mean field approaches that provide a constant dependence on the number of agents (hence, are easier to scale to large environments with millions of agents). Also, since the MAAC attention module includes inputs from all other agents in the environment for decision-making, it is computationally too demanding in many-agent environments. To the best of our knowledge, attention mechanism has not been used in the context of mean field learning previously. We provide a method that extends the use of attention mechanism in the mean field setting.

It is well acknowledged that MARL algorithms have poor sample efficiency and hence it is critical to accelerate the training of MARL approaches using external sources of knowledge to make MARL applicable in complex real-world problems [225]. In this context, the most popular approach is that of reinforcement learning from expert demonstration (RLED) [191], where RL training is accelerated using demonstrations from experts [100] (including humans [268]). RLED has also been explored in MARL, however most prior works require near-optimal experts [202, 292] or are only applicable to restrictive settings [226, 227]. Our approach in Chapter 3 proposes the advising multiple intelligent reinforcement agents - decision making (ADMIRAL-DM) algorithm for learning from sub-optimal advisors in non-restrictive general sum stochastic games. We will adapt this approach to the mean field setting in this chapter.

8.4 Mean Field Attention Advising

In this section we introduce the mean field attention advising algorithm. As described in Section 8.1, MFAA brings two novelties to mean field learning, 1) incorporating the attention mechanism for determining different levels of importance for other agents and 2) accelerating training using advisors. As in Chapter 3, we assume that each agent can have access to at-most one advisor in our setting. First, we focus on the attention module and then explain the action advising part of the algorithm.

Before training, the MFAA requires a specification of a fixed number, K, which denotes the size of the importance set, where an importance set consists of "near-by" agents that influence the decision-making of the central agent. We will use the variable N to denote the total number of agents (where N is large since this is a mean field setting). Every agent considers a set of K other agents nearby as part of its importance set (indexed as $i \in [1, \ldots, K]$), where $K < N$. K is a hyper-parameter which could be picked using domain knowledge.

The objective is to enable the central agent to attend to different agents in the importance set based on their relative importance. Towards this objective, we incorporate an attention layer [282] as part of the neural network that serves as the function approximator for the Q-function in Eq. 8.1. The architecture of this neural network is provided in Figure 8.1. The architecture follows a similar scheme as used in Shariq and Fei [110]. Let the central agent be indexed by j. The inputs to the network are the state of the stochastic game (which includes the local observation of the agent j and the current available observations and actions of other agents nearby) and the mean field action for the agent j, i.e., $\overline{a}^j$. The function approximation of the mean field Q function of the agent j considers the agents' own observation, action and mean field, in addition to having another component that reflects the contributions of other agents in the importance set. This is expressed as,

$$Q^j_\psi(s, \overline{a}^j) = f^j(g^j(o^j), h^j(\overline{a}^j), x^j) \tag{8.9}$$

where $Q^j_\psi(s, \overline{a}^j)$ is a vector that contains the Q-values of all the actions in the action space of the agent j. From this, the value of $Q^j_\psi(s, \overline{a}^j, a^j)$ can be obtained using the action a^j. Also, as in Shariq and Fei [110], f^j is two-layer multi-layer perceptron (MLP), g^j and h^j are one-layer MLP embeddings. The x^j captures the contribution from other agents in the importance set, expressed as:

$$x^j = \sum_i \alpha^i v^i = \sum_i \alpha^i n(Vl^i(o^i, a^i)) \tag{8.10}$$

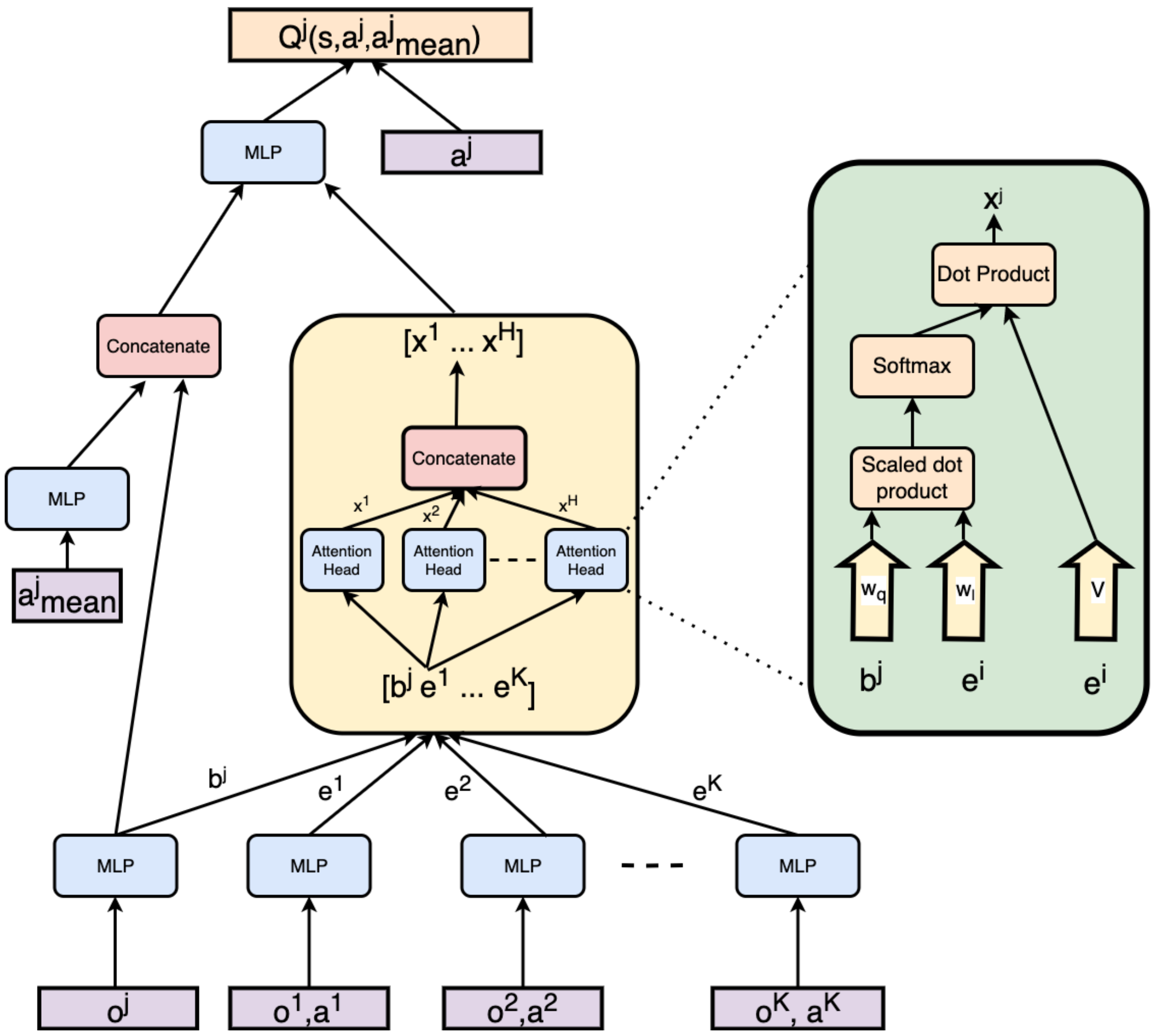


Figure 8.1: Network architecture for calculating (function approximation based value) $Q^j(s, a^j, \overline{a}^j)$ for each agent j (in the figure $a^j_{mean} = \overline{a}^j$). Each agent uses its observation of other agents' observation and the other agents' action in addition to the mean field to calculate the Q-value for each action in its action space.

where l^i is an embedding function for the observation and action of the agent i, which is further linearly transformed by the value matrix V. Also, n is a non-linear activation function. Hence, v^i represents an embedding of the agent i. α^i is the attention weight that compares two embeddings, one associated with the agent j's own observation, i.e. $b^j = g^j(o^j)$, and the other associated with an agent i in the importance set, i.e. $e^i = l^i(o^i, a^i)$, to calculate a similarity value. This similarity value is then passed into a softmax as done in Vaswani et al. [282]. Let W_q and W_k represent embeddings that transform b^j into a "query" and e^i into a "key" respectively. Then the attention weights can be expressed as (where the T represents transpose),

$$\alpha^i \propto \exp(e^{i,T} W_k^T W_q b^j). \tag{8.11}$$

As in Shariq and Fei [110] we use a multi-head attention in our implementations, where the network has a total of H heads. The objective is for each head to focus on a different weighted mixture of other agents in the importance set. Each head uses a separate set of parameters (W_k, W_q, V), and the final output from each head is concatenated as a single vector.

Finally, by combining all the components discussed so far, we obtain the function approximation described in Eq. 8.9.

Now, we explain the action advising component of our algorithm. MFAA follows the same scheme as in our work in Chapter 3, where the probabilistic policy reuse (PPR) [66] technique is used to incorporate a finite amount of action advice from an online advisor during training. As specified earlier, in our setting, we assume that every agent has access to at-most one advisor. The PPR technique uses a hyper-parameter (ϵ') to control the probability of receiving an action advice from an advisor. This hyper-parameter is decayed (linearly) in such a way that the agent receives maximum advisor inputs at the beginning of training and subsequently, the frequency of advisor inputs is gradually reduced. Whenever the action advice from the advisor is available, the agent performs this action as opposed to relying on its own policy. After a fixed number of training episodes, the agent will not receive any further inputs from the advisor and must rely on its own policy for action selection.

Using the update equations provided in Eqs. 8.1 – 8.4, along with the attention mechanism and action advising components, we provide a practical Q-learning algorithm called Mean Field Attention Advising Q-Learning (MFAA-QL). The complete pseudo code for MFAA-QL is provided in Algorithm 16. Here, we use a separate target network for training along with a replay buffer, as introduced by Mnih et al. [167] for DQN. Further, we extend

MFAA-QL to an actor-critic method called Mean Field Attention Advising Actor-Critic (MFAA-AC). Here the Q-function is used as the critic and the policy derived from the Q-value serves as the actor. The complete pseudo-code for MFAA-AC is provided in Algorithm 17.

Algorithm 16 Mean Field Attention Advising Q-Learning (MFAA-QL)

```
 1: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, for all agents $j$
 2: Initialize the mean action $\overline{a}^j$ for each agent $j \in 1, \dots, N$
 3: Initialize a value for the hyper-parameter $\epsilon'$ (i.e. value for $\epsilon'_0$)
 4: Initialize the total number of steps (T) and total number of episodes (E)
 5: while Episode < E do
 6:     while Step < T do
 7:         For each $j$, let $u$ be a uniform random number between 0 and 1
 8:         if $u < \epsilon'_t$ then
 9:             Obtain action $a^j$ from the advisor (using current state $s$ and mean action $\overline{a}^j$)
10:         else
11:             Choose action $a^j$ from $Q_{\phi^j}$ according to Eq. 8.4 using the mean action $\overline{a}^j$
    and the exploration rate $\hat{\beta}$
12:         end if
13:         For each agent j, compute the new mean action $\overline{a}^j$ according to Eq. 8.3
14:         Execute the joint action $\boldsymbol{a} = [a^1, \dots, a^N]$. Observe the rewards $\boldsymbol{r} = [r^1, \dots, r^N]$
    and the next state $s'$
15:         Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}} \rangle$ in replay buffer $D$, where $\overline{\boldsymbol{a}} = [\overline{a}^1, \dots, \overline{a}^N]$ is the joint mean
    action. The $\boldsymbol{a}$ captures all the $N$ agents
16:     end while
17:     while $j = 1$ to $N$ do
18:         Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}} \rangle$ from $D$
19:         Set $y^j = r^j + \gamma v^{MF}_{\phi^j_-}(s')$ according to Eq. 8.1
20:         Update the $Q$ network by minimizing the loss $L(\phi^j) = \frac{1}{K} \sum (y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j))^2$
21:     end while
22:     Update the parameters of the target network for each agent $j$ with learning rate $\tau$;

            $$\phi^j_- \leftarrow \tau \phi^j + (1 - \tau) \phi^j_-$$

23:     At the end of each episode linearly decay $\epsilon'_t$
24: end while
```

Algorithm 17 Mean Field Attention Advising Actor-Critic (MFAA-AC)

```
1: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, and the policies $\pi_{\theta^j}$, $\pi_{\theta^j_-}$, for all agents $j$
2: Initialize the mean action $\overline{a}^j$ for each agent $j \in 1, \ldots, N$
3: Initialize a value for the hyper-parameter $\epsilon'$ (i.e. value for $\epsilon'_0$)
4: Initialize the total number of steps (T) and total number of episodes (E)
5: while Episode < E do
6:     while Step < T do
7:         For each $j$, let $u$ be a uniform random number between 0 and 1
8:         if $u < \epsilon'_t$ then
9:             Obtain action $a^j$ from the advisor (using current state $s$ and mean action $\overline{a}^j$)
10:        else
11:            Choose action $a^j$ from $\pi_{\theta^j}(s^j)$ according to Eq. 8.4 using the exploration rate $\hat{\beta}$
12:        end if
13:        For each agent j, compute the new mean action $\overline{a}^j$ according to Eq. 8.3
14:        Execute the joint action $\boldsymbol{a} = [a^1, \ldots, a^N]$. Observe the rewards $\boldsymbol{r} = [r^1, \ldots, r^N]$ and the next state $s'$
15:        Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}} \rangle$ in replay buffer $D$, where $\overline{\boldsymbol{a}} = [\overline{a}^1, \ldots, \overline{a}^N]$ is the joint mean action. The $\boldsymbol{a}$ captures all the $N$ agents
16:    end while
17:    while $j = 1$ to $N$ do
18:        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}} \rangle$ from $D$
19:        Set $y^j = r^j + \gamma v^{MF}_{\phi^j_-}(s')$ according to Eq. 8.1
20:        Update the $Q$ network (critic) by minimizing the loss $L(\phi^j) = \frac{1}{K}\sum(y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j))^2$
21:        Update the actor using the sample policy gradient:
               $$\nabla_{\theta^j} J(\theta^j) \approx \frac{1}{K}\sum \nabla_{\theta^j} \log \pi_{\theta^j}(s'^j) Q_{\phi^j_-}(s', a^j_-, \overline{a}^j)|_{a^j_- = \pi_{\theta^j_-}(s'^j)}$$
22:    end while
23:    Update the parameters of the target network for each agent $j$ with learning rate $\tau_\phi$ and $\tau_\theta$;
               $$\phi^j_- \leftarrow \tau_\phi \phi^j + (1 - \tau_\phi)\phi^j_-$$
               $$\theta^j_- = \tau_\theta \theta^j + (1 - \tau_\theta)\theta^j_-$$
24:    At the end of each episode linearly decay $\epsilon'_t$
25: end while
```

8.5 Multi-Type Mean Field Attention Advising

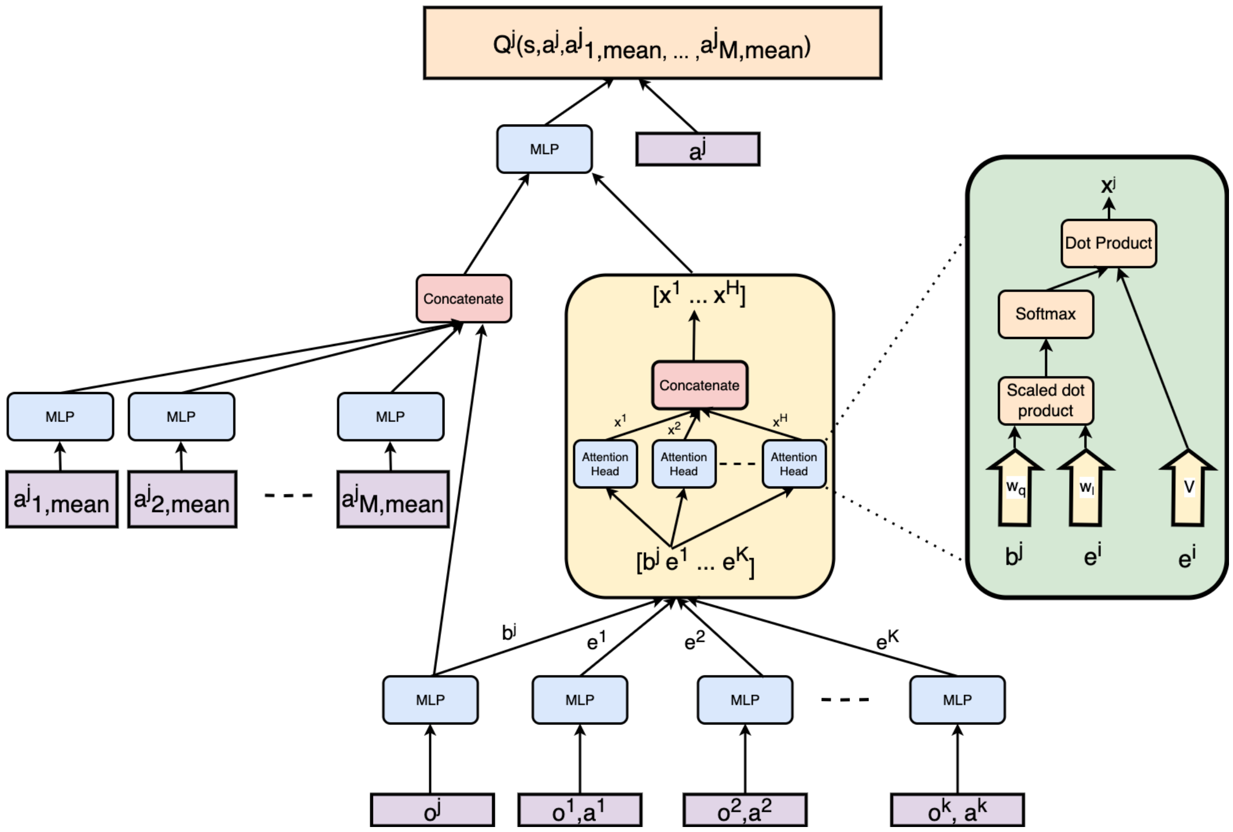


Figure 8.2: Multi-type network architecture for calculating (function approximation based value) $Q^j(s, a^j, \overline{a}^j_{1,t}, \dots, \overline{a}^j_{M,t})$ for each agent j. As compared to Figure 8.1, this version of our algorithm considers different types, so the mean field of each type is explicitly handled (as opposed to just a single mean field in Figure 8.1).

In this section, we extend the MFAA framework to include multiple types using the approach we introduced in Chapter 5. Assuming a total of M types in the environment (indexed by m, i.e. $m \in [1, \dots, M]$), we use the update equations provided in Eqs. 8.5 – 8.8. Recall, in Chapter 5, we analyze two different kinds of type based mean field environments, i.e. known types and unknown types. In this chapter, we restrict our work to the case of known types where the type classification of an agent in the environment is known prior to the commencement of the game (public knowledge).

The only change we need to make in our MFAA algorithm is to include the mean field action of each type separately in the neural network architecture provided in Figure 8.1. The updated architecture is provided in Figure 8.2, where the mean fields of the different types are maintained separately.

Now, we provide a practical Q-learning algorithm for the multiple type based updates called Multi-Type Mean Field Attention Advising Q-Learning (MTMFAA-QL). The complete pseudo code for MTMFAA-QL is provided in Algorithm 18. We also extend this method to an actor-critic technique in Algorithm 19. This algorithm is called Multi-Type Mean Field Attention Advising Actor-Critic (MTMFAA-AC).

Algorithm 18 Multi-Type Mean Field Attention Advising Q-Learning (MTMFAA-QL)

```
1: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, for all agents $j$
2: Let the environment have $M$ types. Initialize the mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ for each agent $j \in 1, \ldots, N$
3: Initialize a value for the hyper-parameter $\epsilon'$ (i.e. value for $\epsilon'_0$)
4: Initialize the total number of steps (T) and total number of episodes (E)
5: while Episode < E do
6:     while Step < T do
7:         For each $j$, let $u$ be a uniform random number between 0 and 1
8:         if $u < \epsilon'_t$ then
9:             Obtain action $a^j$ from the advisor (using current state $s$ and mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$)
10:        else
11:            Choose action $a^j$ from $Q_{\phi^j}$ according to Eq. 8.8 using the mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ and the exploration rate $\hat{\beta}$
12:        end if
13:        For each agent j, compute the new mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ according to Eq. 8.7
14:        Execute the joint action $\boldsymbol{a} = [a^1, \ldots, a^N]$. Observe the rewards $\boldsymbol{r} = [r^1, \ldots, r^N]$ and the next state $s'$
15:        Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}}_1, \ldots, \overline{\boldsymbol{a}}_M \rangle$ in replay buffer $D$, where $\overline{\boldsymbol{a}}_i = [\overline{a}^1_i, \ldots, \overline{a}^N_i]$ is the joint mean action for type $i$. The $\boldsymbol{a}$ captures all the $N$ agents
16:    end while
17:    while $j = 1$ to $N$ do
18:        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}}_1, \ldots, \overline{\boldsymbol{a}}_M \rangle$ from $D$
19:        Set $y^j = r^j + \gamma v^{MTMF}_{\phi^j_-}(s')$ according to Eq. 8.5
20:        Update the $Q$ network by minimizing the loss $L(\phi^j) = \frac{1}{K}\sum(y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M))^2$
21:    end while
22:    Update the parameters of the target network for each agent $j$ with learning rate $\tau$;
           $$\phi^j_- \leftarrow \tau\phi^j + (1-\tau)\phi^j_-$$
23:    At the end of each episode linearly decay $\epsilon'_t$
24: end while
```

Algorithm 19 Multi-Type Mean Field Attention Advising Actor-Critic (MTMFAA-AC)

```
1: Initialize the $Q$ functions (parameterized by weights) $Q_{\phi^j}, Q_{\phi^j_-}$, and the policies $\pi_{\theta^j}$, $\pi_{\theta^j_-}$, for all agents $j$
2: Let the environment have $M$ types. Initialize the mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ for each agent $j \in 1, \ldots, N$
3: Initialize a value for the hyper-parameter $\epsilon'$ (i.e. value for $\epsilon'_0$)
4: Initialize the total number of steps (T) and total number of episodes (E)
5: while Episode < E do
6:     while Step < T do
7:         For each $j$, let $u$ be a uniform random number between 0 and 1
8:         if $u < \epsilon'_t$ then
9:             Obtain action $a^j$ from the advisor (using current state $s$ and mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$)
10:        else
11:            Choose action $a^j$ from $\pi_{\theta^j}(s^j)$ according to Eq. 8.8 using the exploration rate $\hat{\beta}$
12:        end if
13:        For each agent j, compute the new mean action for each type $\overline{a}^j_1, \ldots, \overline{a}^j_M$ according to Eq. 8.7
14:        Execute the joint action $\boldsymbol{a} = [a^1, \ldots, a^N]$. Observe the rewards $\boldsymbol{r} = [r^1, \ldots, r^N]$ and the next state $s'$
15:        Store $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}}_1, \ldots, \overline{\boldsymbol{a}}_M \rangle$ in replay buffer $D$, where $\overline{\boldsymbol{a}}_i = [\overline{a}^1_i, \ldots, \overline{a}^N_i]$ is the joint mean action for type $i$. The $\boldsymbol{a}$ captures all the $N$ agents
16:    end while
17:    while $j = 1$ to $N$ do
18:        Sample a minibatch of $K$ experiences $\langle s, \boldsymbol{a}, \boldsymbol{r}, s', \overline{\boldsymbol{a}}_1, \ldots, \overline{\boldsymbol{a}}_M \rangle$ from $D$
19:        Set $y^j = r^j + \gamma v^{MTMF}_{\phi^j_-}(s')$ according to Eq. 8.5
20:        Update the $Q$ network (critic) by minimizing the loss $L(\phi^j) = \frac{1}{K}\sum(y^j - Q_{\phi^j}(s^j, a^j, \overline{a}^j_1, \ldots, \overline{a}^j_M))^2$
21:        Update the actor using the sample policy gradient:
               $$\nabla_{\theta^j} J(\theta^j) \approx \frac{1}{K}\sum \nabla_{\theta^j} \log \pi_{\theta^j}(s'^j) Q_{\phi^j_-}(s', a^j_-, \overline{a}^j_1, \ldots, \overline{a}^j_M)|_{a^j_- = \pi_{\theta^j_-}(s'^j)}$$
22:    end while
23:    Update the parameters of the target network for each agent $j$ with learning rate $\tau_\phi$ and $\tau_\theta$;
               $$\phi^j_- \leftarrow \tau_\phi \phi^j + (1 - \tau_\phi)\phi^j_-$$
               $$\theta^j_- = \tau_\theta \theta^j + (1 - \tau_\theta)\theta^j_-$$
24:    At the end of each episode linearly decay $\epsilon'_t$
25: end while
```

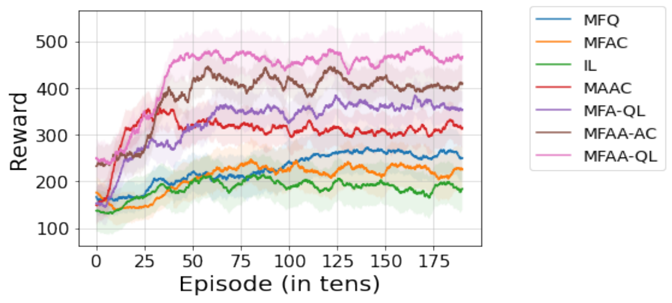
8.6 Experiments and Results

In this section, we run a series of experiments that illustrates the performance of our algorithms. The important elements of our experimental domains are provided here, while the complete details (including specifications of the reward function) are available in Appendix F.1. We conduct our experiments across two test beds. The first is the MAgent environment introduced by Zheng et al. [317] (which we have used in prior chapters), which supports the learning of a large number of pixel based agents interacting in different kinds of multi-agent scenarios. The second is the Neural MMO simulator introduced by Suarez et al. [242] which simulates the learning of large number of agents in virtual worlds mimicking the classic massively multiplayer online role playing games [53]. The per-agent complexity in Neural MMO is much larger than that of MAgents [242]. All environments contain a discrete action space to be applicable to the MFRL setting.

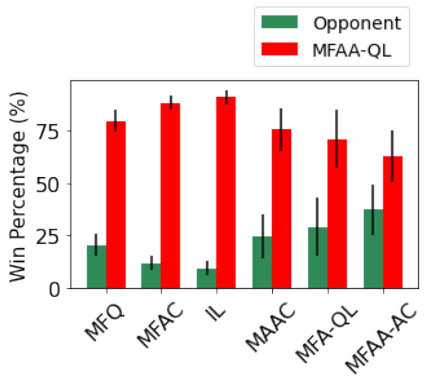
As in many prior works in mean field learning [88, 248, 303], and as done in previous chapters, we run experiments in two phases — training and execution. Each of our environments contains agents belonging to specific teams. In the training phase, every agent of the same team trains using the same network and the same algorithm, as done in prior works [248, 303]. In environments containing multiple teams, our training procedure is similar to the self-play training scheme [222], where the different teams train separate networks using the same algorithm. The training is conducted for 2000 episodes. After training, agents enter into an execution phase where the trained policies are executed with no further training. The execution competitions are run for 1000 games, where agents trained using one algorithm may compete against agents trained using a different algorithm in a face-off style competition. We consider seven algorithms in our experiments. These are independent Q-learning (IL) [264], mean field Q-learning (MFQ) [303], mean field actor-critic (MFAC) [303], multi-actor-attention-critic (MAAC) [110], MFAA-QL and MFAA-AC. We also consider an ablated version of our algorithm, MFA-QL (Mean Field Attention Q-Learning), which is an implementation of MFAA-QL but without the action advising component (cannot learn from advisors). For the algorithms that use advisors (MFAA-QL and MFAA-AC), the hyper-parameter ϵ' starts at 1 at the beginning of training and decays to 0 within 500 episodes of training. There is no influence of advisors for the remaining training episodes. Also, there is no influence from advisors during the execution phase. The hyper-parameter K in MFAA-QL and MFAA-AC is set to 10 throughout, i.e., we consider 10 closest (based on distance to the learning agent) agents as part of the importance set for each agent. Since our domains support a variable number of agents (includes killing), if the number of agents in the environment goes less than K, we use a placeholder value (all zeroes) for the observation and actions of the remaining agents. This is also done for our implementation of MAAC, since MAAC does not naturally support learning in environments with a variable number of agents.

We repeat all experiments 30 times and report the average and standard deviation of performances. The hyper-parameters used in our experiments are reported in Appendix F.2 and wall clock times are provided in Appendix F.3. For all the results in this section, we run statistical significance tests. For the training experiments we conduct an unpaired two-sided t test, typically at the last episode of training. For the execution experiments we conduct a Fischer's exact test for the final average performances. We report the p-values for each of these tests and consider results with $p < 0.05$ to be statistically significant differences, as is standard in practice [285].

Now we consider a few domains from the MAgents environment [317]. The first domain we consider is the mixed cooperative-competitive Battle game, that we used in previous chapters. Recall, this domain contains two teams of homogeneous agents. The game starts with the same number of agents in each team. Each agent is expected to cooperate with other agents of the same team and compete against the agents of the other team to win the battle. The agents will need to kill other agents belonging to the opposite team to win. The action space contains a total of 21 actions, with 12 action pertaining to moving, 8 actions pertaining to attacking and one action pertaining to doing nothing. Our current setting contains some subtle changes from the Battle setting in previous chapters (details in Appendix F.1).



(a) Training

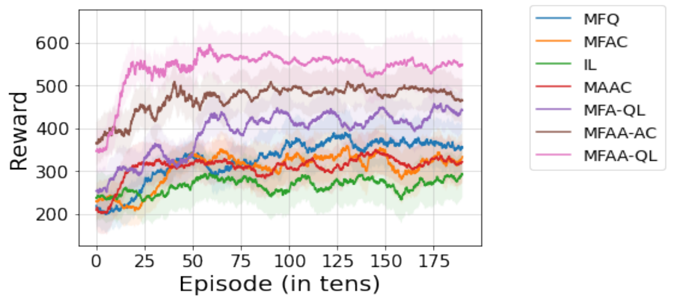


(b) Execution

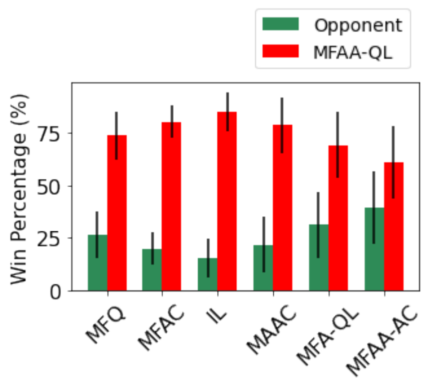
Figure 8.3: Battle game with 64 agents in each team.

We consider three different settings with the Battle game, with each team containing 64, 100 and 144 agents respectively. For the training phase we plot the cumulative rewards obtained by all agents belonging to one team (the performance of the other team is also similar, our domains are not zero-sum). For the execution phase, we plot the percentage of games won by each algorithm in a face-off competition with MFAA-QL. In the execution phase, a game is determined to be won by the team that kills more of its opponents. If both teams kill the same number of opponents, the team obtaining higher cumulative rewards is determined to be the winner. The MFAA-QL and MFAA-AC methods use a pre-trained policy of IL trained for 10000 episodes as the advisor. Figure 8.3(a) plots the training performance in the Battle game setting with 64 agents in each team. We make several observations. First, we note that the independent method (IL) that does not use either the mean field or the attention mechanism, loses out to all other algorithms. The mean field based methods that do not use attention (MFQ, MFAC), lose out to an algorithm that does not use mean field but uses the attention mechanism (MAAC) ($p < 0.05$). In this game, though the mean field can help in a relatively longer term strategic planning, an agent will need to escape enemies and kill off competition in its immediate vicinity to survive and hence the attention mechanism is more useful to obtain larger rewards (i.e., per agent modelling is more useful than global information). However, MAAC attends to every other agent in the environment which includes a lot of noise (agents further away may not pose any immediate threat, and are unnecessary information in the attention mechanism). Recall that the attention mechanism learns a similarity score between the central agent and each of the other agents in the importance set (query-key-value based system). If there are many agents in the importance set, it may take the attention mechanism a relatively long training period to determine the relative importance amongst all the agents. Hence, MAAC loses out to MFA-QL ($p < 0.06$) which uses an attention mechanism to attend to nearby agents (small number of agents in the importance set) and uses the mean field to reason about agents further away. MFA-QL is outperformed by MFAA-AC ($p < 0.04$) and MFAA-QL ($p < 0.01$), since both these algorithms are capable of training acceleration using action advising. We also notice that MFAA-QL obtains a better performance than MFAA-AC ($p < 0.02$) and outperforms all the other algorithms comfortable ($p < 0.01$). Prior works [303] note that Q-learning based approaches induce a positive bias through the max action and often out-perform actor-critic based approaches, which explains the better performance of MFAA-QL relative to MFAA-AC. Since MFAA-QL uses an attention mechanism, learns from an advisor, and also considers the mean field, it provides better performances than other baselines ($p < 0.01$). The execution results in Figure 8.3(b) also show that MFAA-QL wins more games than all other algorithms (including MFAA-AC, $p < 0.01$), demonstrating its superiority.

Figure 8.4(a) shows the training performances in the same Battle game setting with 100 agents in each team. Again we note that MFAA-QL provides the best performance compared

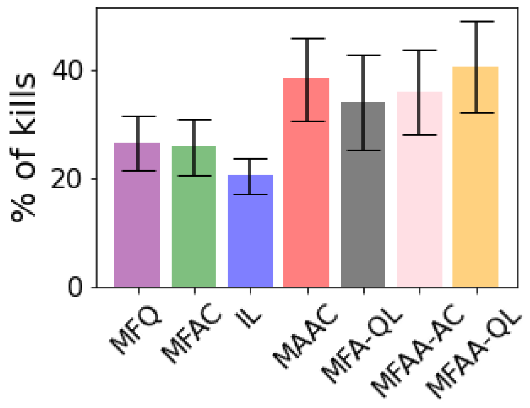


(a) Training

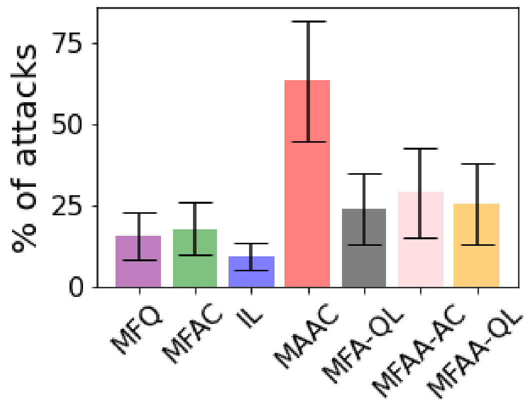


(b) Execution

Figure 8.4: Battle game with 100 agents in each team.



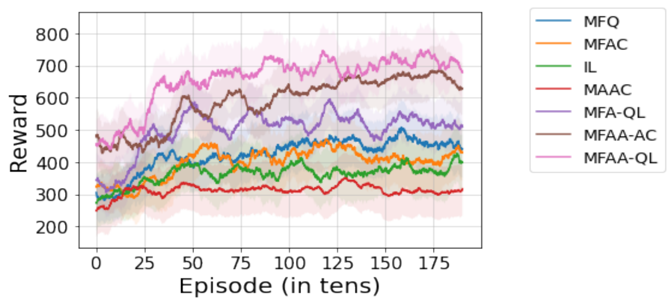
(a) % of agents killed by each algorithm during training.



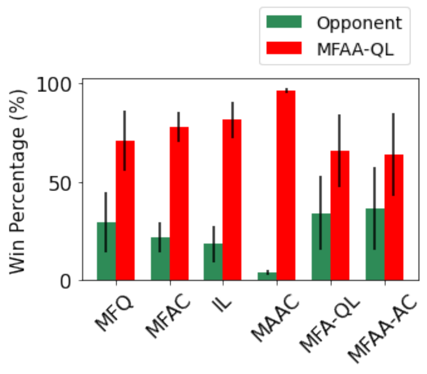
(b) % of attack actions performed by each algorithm during training.

Figure 8.5: Battle game training with 100 agents in each team, killing and attacking statistics.

to all other algorithms ($p < 0.01$). We draw similar inferences for the performances of all other algorithms as in Figure 8.3. However, one notable observation is that the performance of MAAC sees a relative drop in this setting as compared to the setting with 64 agents. In fact, a purely mean field approach (MFQ) overtakes the performance of MAAC in this setting ($p < 0.08$). This demonstrates two facts, 1) mean field based methods are most well-suited to scaling MARL to environments with larger agents and 2) purely attention based methods that consider all other agents in the attention mechanism do not scale well, since more information noise is being included (agents further away are not likely to have any considerable impact in large environments). The approaches that combine mean field with the attention mechanism (MFAA-QL, MFAA-AC, MFA-QL) outperform MFQ ($p < 0.03$). Learning from action advising (MFAA-QL, MFAA-AC) provides an edge over methods that do not learn from action advising (MFA-QL) ($p < 0.01$). Our execution results in Figure 8.4(b) also reinforces the superior performance of MFAA-QL as compared to all baselines and MFAA-AC ($p < 0.01$). To further analyze the performance of MAAC, we plot the percentage of attack actions executed by each method and the percentage of opponents killed by each method during training in Figure 8.5. Figure 8.5(a) shows that MAAC kills almost the same number of opponents on average as MFAA-QL (which is more than MFQ), however, it executes a very large percentage of attack actions as seen in Figure 8.5(b) (much larger than any of the other methods). In the Battle game, agents are penalized for making wrong attack moves, like attacking empty grids or attacking within the same team (details in Appendix F.1). This leads to MAAC performance being poor compared to other methods as seen in Figure 8.4(a). We infer that MAAC learns a very aggressive and a relatively naive strategy of attacking the nearest agent (and not seeking opponents to attack), while purely mean field based methods (MFQ, MFAC) are more conservative (very few attack actions). The methods that combine both attention and mean field (MFAA-AC, MFAA-QL, MFA-QL) seem to balance the aggressive and defensive strategies which is the best approach to win the Battle game. This is reflected in their superior performances in Figure 8.4.



(a) Training



(b) Execution

Figure 8.6: Battle game with 144 agents in each team.

In the third Battle setting, with 144 agents in each team (see Figure 8.6), we note that MAAC performance suffers further and overall even achieves a lower performance than the independent method (IL). However, purely mean field methods (MFQ, MFAC) show a better performance than IL. This observation is consistent with our explanations in the previous setting with 100 agents in each team. In this setting too, MFAA-QL provides the best performance as compared to other baselines in both training and execution ($p < 0.01$).

Next we perform an ablation study using this Battle domain. In this study we vary the number of agents in the importance set K for the MFAA-QL algorithm. We consider the Battle game setting with 64 agents in each team. The performances are plotted in Figure 8.7 where we note that, for a very low value of $K = 4$, the performances are quite poor. This is because a sufficient number of other agents that can impact the performance of the central agent are not being considered in the importance set. When K is increased to 10, we see a large improvement in performance. Increasing K further (20 and 30) shows only a small improvement in performance. In fact, when the value of $K = 40$ the performance drops. This is due to inclusion of information noise while considering more than the sufficient number of agents. This study shows the importance of choosing the right value of K for the performance of MFAA-QL. If the value of K is too small, the performances are poor, and if the value of K is too large, there may not be too much gain in performance relative to increase in computational demands or there may even be a drop in performance.

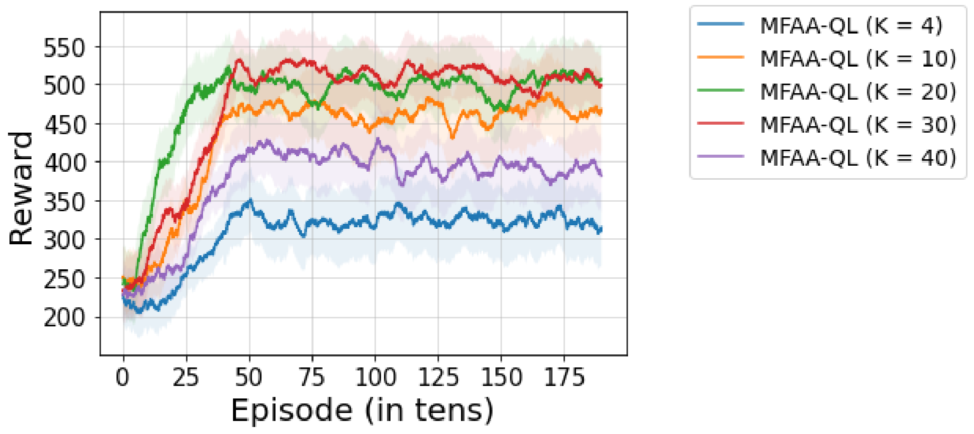
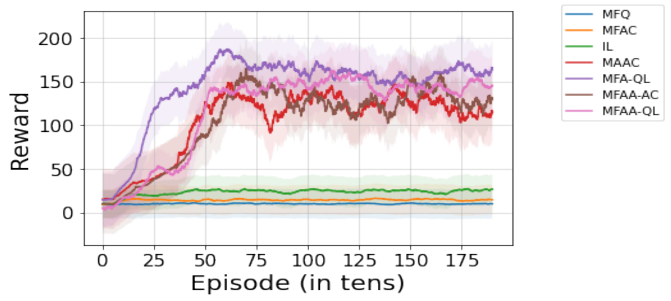
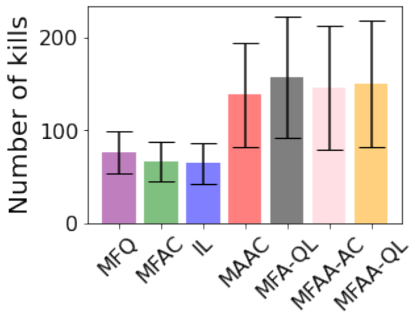


Figure 8.7: Ablation on MAgent Battle game varying the value of K (64 agents in each team).

Now, we move to our second MAgent environment, which is the fully cooperative Tiger game. This setting is the same as that considered in Chapter 7, though there are some subtle changes in the conditions (details in Appendix F.1). This environment contains a team of tigers that need to work together to kill the deer in the environment. The deer are part of the environment (moving randomly) while tigers are the learning agents. The tigers are rewarded for attacking and killing deer, while being penalized for making wrong attacks (attacking other tigers or an empty grid). At-least two tigers need to attack a deer together to obtain rewards. The action space is the same as the Battle game. More details are provided in Appendix F.1. We consider two settings with this domain, the first has 54 tigers and 273 deer and the second has 100 tigers and 500 deer. In the training phase, we plot the cumulative rewards obtained by the team of tigers using each of the seven algorithms. In the execution phase, we plot the average number of deer killed (per episode) by the trained policies pertaining to each of the algorithms (no further training). Similar to the Battle game, the MFAA-QL and MFAA-AC algorithms use IL trained for 10000 episodes as the advisor.



(a) Training

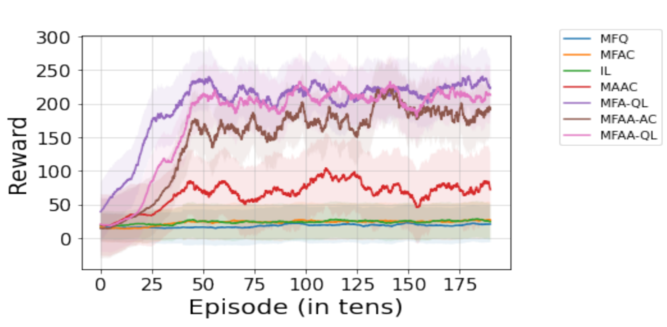


(b) Execution

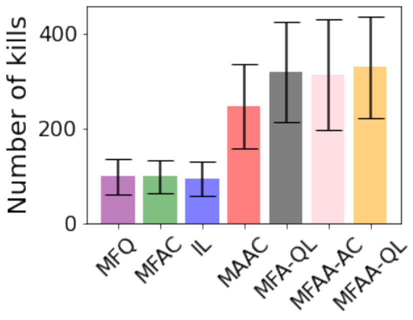
Figure 8.8: Tiger domain with 54 tigers.

First, we consider the setting with 54 tigers. The training results in are Figure 8.8(a) which show that the purely mean field methods (MFQ, MFAC) and independent method (IL) lose out completely in this game. This shows that the mean field is not sufficient for learning in this game (since learning to cooperate with nearby agents is required to gain any rewards, per agent modelling is critical and local information is more important than global information). The algorithms that use the attention mechanism (MFAA-QL, MFAA-AC, MFA-QL, MAAC) show good learning behavior in this domain. Since the advisor used in this setting (IL) does not perform well, the MFA-QL algorithm without action advising shows marginally better performance than MFAA-QL and MFAA-AC ($p < 0.21$). Though MFAA-QL and MFAA-AC initially lose out due to bad advising, they ultimately overcome this and are able to perform almost as well as MFA-QL. The MFA-QL algorithm outperforms MAAC ($p < 0.01$). The reason is in line with our discussion in the

Battle game (MAAC has more information noise, using mean field for reasoning about far away agents is sufficient and even superior as compared to considering each individual agent in the attention network). The performances in the execution phase (Figure 8.8(b)) are also consistent with our findings in the training phase, where MFA-QL obtains the maximum performance outperforming MFQ, MFAC, IL, and MAAC (with $p < 0.4$) and MFAA-QL and MFAA-AC (with $p < 0.8$). Despite observing such performances, the differences are not statistically significant as reflected in the p-values.



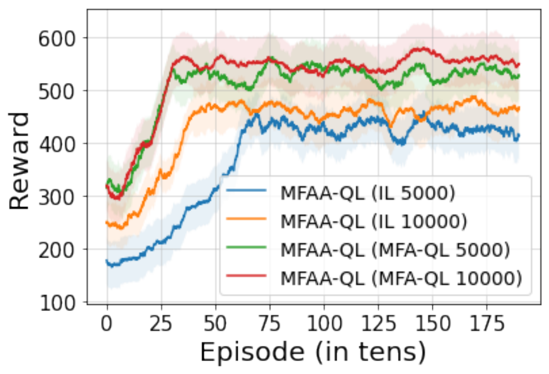
(a) Training



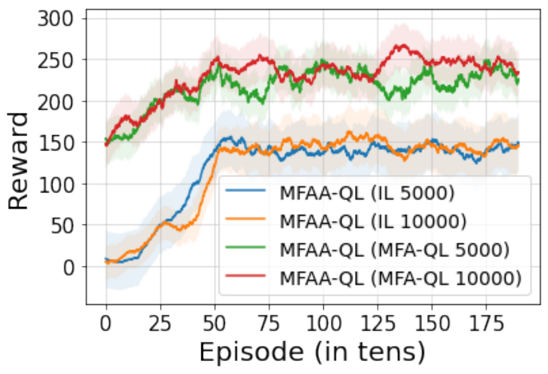
(b) Execution

Figure 8.9: Tiger domain with 100 tigers.

Scaling the same setting to 100 tiger agents we plot the training and execution performance in Figure 8.9. Most of our observations are the same as the setting with 54 tiger agents (purely mean field methods fail and attention based methods perform better). In addition, we note that the performance of MAAC undergoes a significant drop as compared to the previous setting. Again, we note that MAAC does not scale well to hundreds of agents.



(a) Battle

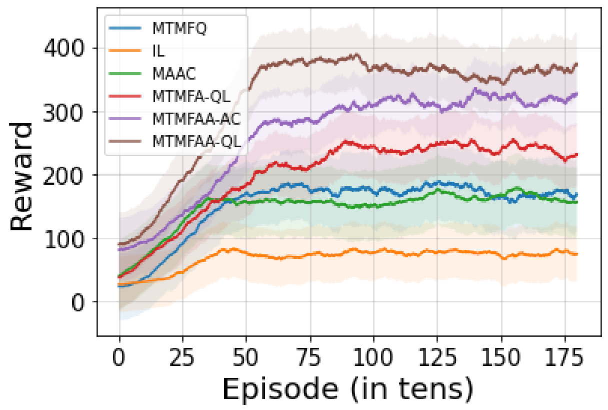


(b) Tiger

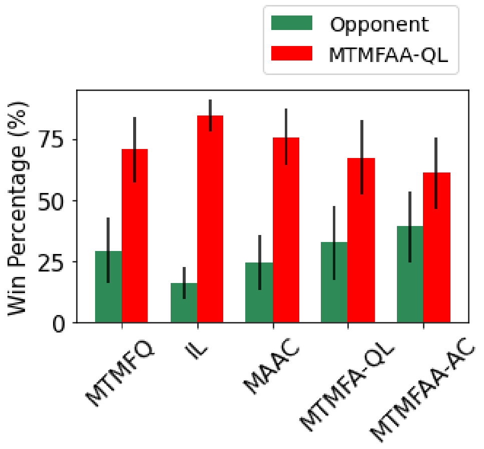
Figure 8.10: Ablation that shows the influence of advisors on the MAgent Battle and Tiger games. The Battle game in this study uses 64 agents in each team while the Tiger domain uses 54 tiger agents.

Now, using the Battle and Tiger domains we perform an ablation study, that considers different types of advisors to analyze the effect of these different advisors. First, we consider the Battle domain with 64 agents in each team (total of 128 agents). In this domain, we plot the performances of MFAA-QL using 4 different advisors: 1) IL pretrained for 5000 episodes, 2) IL pretrained for 10000 episodes, 3) MFA-QL pretrained for 5000 episodes and 4) MFA-QL pretrained for 10000 episodes. As noted in our experimental performances in Figure 8.3(a) MFA-QL performance is far better than IL. Also, within the same algorithm training for more episodes leads to (at-least) a better performance. Hence, the ranking of these advisors are MFA-QL (10000) $>$ MFA-QL (5000) $>$ IL (10000) $>$ IL (5000), where the number in the brackets denotes the number of pre-training episodes (based on performances, MFA-QL trained for 5000 episodes is better than IL trained for 10000 episodes). The performances of MFAA-QL in Figure 8.10(a) show that MFA-QL (10000) provides the best performance and IL(5000) provides the least performance. This shows that comparatively better advisors lead to better performances and vice-versa. Next, we conduct the same study (with the same set of advisors) in the Tiger domain with 54 agents. As noted in Figure 8.8(a), the IL algorithm hardly learns anything in this domain and the MFA-QL method reaches a relatively very stable equilibrium point after 2000 episodes of training. Hence, the order of advisors here are MFA-QL (10000) $\approx$ MFA-QL (5000) $>$ IL (10000) $\approx$ IL (5000). The performances of MFAA-QL in Figure 8.10(b) show a relatively better performance with better advisors (and vice-versa) as noted in the Battle game.

For the final MAgent domain we consider the Combined Arms heterogeneous environment.



(a) Combined Arms - Training

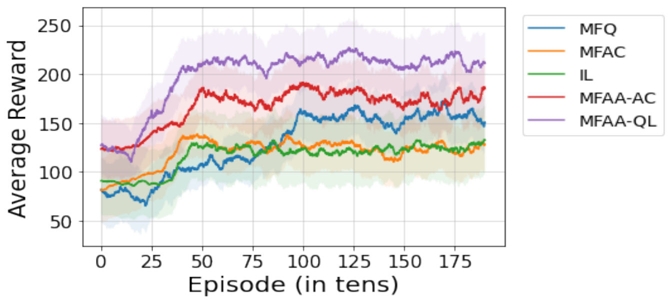


(b) Combined Arms - Execution

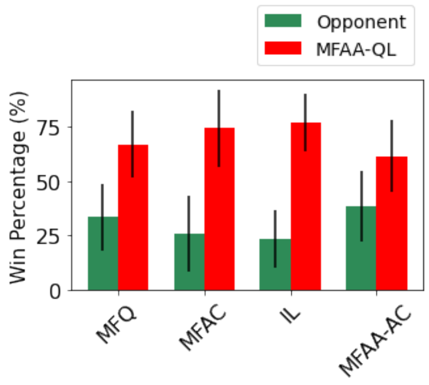
Figure 8.11: Combined Arms heterogeneous agents experiment. Each team contains 100 agents of which 60 are ranged and 40 are melee.

The domain is similar to the Battle game, where we have two teams of agents trying to win a Battle by killing members of the opponent team. However, unlike the Battle game, this environment consists of teams of heterogeneous agents. In each team, there are two types of agents, ranged and melee with different action spaces. Ranged and melee agents have different abilities and objectives. The ranged agents are faster and can attack further, but lack stamina (can be killed quickly). The melee agents are slower but have higher stamina (harder to kill). This environment is the same as that considered in Chapter 7. More details are provided in Appendix F.1. In this experiment, we model the ranged and melee as distinct types and use the multi-type version of our algorithms (MTMFAA-QL and MTMFAA-AC). Like the previous experiments, we will consider a ablated version of MTMFAA-QL without action advising. This algorithm is called MTMFA-QL (Multi-Type Mean Field Attention Q-Learning). We will omit MFQ and MFAC for this experiment since they require fully homogeneous agents. Instead, we consider the MTMFQ algorithm from Chapter 5 as a baseline. Similar to the previous experiments, the advisor for MTMFAA-QL and MTMFAA-AC is a pre-trained IL agent (trained for 10000 episodes). In this experiment, each team has a total of 100 agents (50 ranged and 50 melee). The importance set for MTMFAA-QL, MTMFAA-AC, MTMFA-QL contains the nearest 10 agents (5 ranged and 5 melee) based on distances to the central agent. All other conditions of training and execution are the same as the Battle game.

The training results in Figure 8.11(a) show that MTMFAA-QL obtains the best performance across all baselines ($p < 0.01$). The execution results in Figure 8.11(b) also provides the same observation ($p < 0.01$). This experiment shows that our algorithms can be extended to environments with multiple types and still provide the best performance as compared to previous approaches.



(a) Training



(b) Execution

Figure 8.12: Neural MMO Battle game with each team containing 40 agents.

Our last experiment uses the Neural MMO simulator [242]. We design a Battle game similar to the setting with MAgent, with 40 agents in each team. The important difference is that the per-agent complexity is more in this setting as compared to the MAgent setting. We are performing this experiment to test the generalizability of our conclusions in MAgent, i.e., we want to show that our methods work well on different (possibly harder) platforms as well. In this domain, each agent needs to explore a larger environmental space while seeking opponents and killing them (which may require learning cooperative strategies within the team). Unlike the MAgent Battle setting, an agent in this setting will need to search and gather food and water resources in the environment, before its existing resources are depleted, in order to survive. Refer to Appendix F.1 and Suarez et al. [242] for more details. For this experiment we will omit MAAC (since it performed poorly in previously experiments and is highly computationally intensive) and MFA-QL (we are not studying the effect of action advising). Our experiment follows the same procedure as in the MAgent Battle game, where MFAA-QL and MFAA-AC use IL pre-trained for 10000 episodes as the advisor. We plot the training and execution performances of the algorithms in Figure 8.12. For training, we plot the average performance (averaged per agent) of a team of agents (as opposed to the cumulative performances considered in the other experiments) for each algorithm. The training performances in Figure 8.12(a) show that MFAA-QL obtains the best performance across all the other algorithms ($p < 0.01$). The execution phase considers the face-off performance of each algorithm against MFAA-QL, where MFAA-QL outperforms all baselines including MFAA-AC ($p < 0.01$) (see Figure 8.12(b)).

From all the experiments, we note that the mean field has the advantage of scaling to environments with more agents, but cannot reason at the agent level and hence loses out when reasoning about individual agents nearby is critical. Alternatively, the attention mechanism has the advantage of explicitly reasoning about other agents to provide best responses, but cannot scale easily. Our algorithms combine the best of both worlds and are able to reason at the agent level and at the same time scale to environments with more agents. This shows the advantage of our method. The p-values show that the majority of our observations are statistically significant.

8.7 Conclusion

In this chapter we introduced a new mean field reinforcement learning paradigm, MFAA, and provided two practical algorithms, one based on Q-learning (MFAA-QL) and the other based on actor-critic (MFAA-AC). This method has two important novelties over prior approaches for mean field learning, 1) using an attention module to attend to different (nearby) agents differently based on their relative level of importance and 2) using action advises from an advisor to accelerate training. Further, we extended MFAA to the MT-MFAA framework (relying on our our prior work in Chapter 5) which can be used in heterogeneous agent environments. Experimentally, we showed that these algorithms outperform prior approaches in many agent environments, and that scaling the number of agents in the environment does not affect its performance.

Chapter 9

Conclusion and Future Work

In this chapter we provide a summary of our major contributions and discuss avenues for future work.

9.1 Summary of Contributions

There has been significant progress in multi-agent reinforcement learning (MARL) in the past decade, but there still remain critical challenges which limit its use in many large-scale real world environments. In this dissertation we address two of these challenges, namely sample complexity and scaling to many agents, and present our work in these areas.

First, we argue that knowledge provided by *advisors* can be leveraged in a principled manner, and can reduce the sample complexity in MARL. To this end, we propose a general framework for learning from external advisors in MARL and provide practical MARL algorithms. Our first set of algorithms restricted each agent to obtain guidance from a single advisor. Subsequently, we relaxed this restriction and provided algorithms that apply to settings that contain agents having access to multiple different advisors. Further, we show that these algorithms have desirable theoretical properties such as convergence to a unique solution concept. We also provide a finite time analysis (sample complexity guarantees), proving superior guarantees as compared to prior work. Empirically, we show that our proposed algorithms compare favourably to other baselines on a set of well-known MARL testbeds, which demonstrates the effectiveness of our approach.

Second, we explore the use of mean-field theory in the context of scaling MARL. In particular, we extend previous work by making mean-field reinforcement learning (MFRL) algorithms applicable to a set of heterogeneous agents, partially observable environments, and decentralized learning paradigms. We provide novel MFRL algorithms that outperform previous methods on a set of large games with many agents. Theoretically, we prove that each of our algorithms are principled since they contain appropriate fixed point guarantees while learning in large environments. We also provide bounds on the information loss experienced as a result of agents responding to the mean field as opposed to each of the other agents in the environment.

Finally, we combine the merits of our approaches to improving MARL sample complexity and scaling MARL algorithms using the mean field. We present the mean field attention advising (MFAA) algorithm that uses a multi-head attention [282] layer to attend differently to other agents nearby. By performing a small amount of per-agent modelling, this mean field approach provides a large improvement in performance. Experimentally, we demonstrate superior performances in several large-scale games including the massively multi-player online (MMO) games.

9.2 Future Work

There are several possibilities for future directions of research. We first discuss some avenues from the empirical perspective, and then provide possibilities of extending our theoretical contributions.

Regarding our contributions to action advising in MARL, in Chapter 3 we provided a relatively simple technique of making use of the evaluation of advisors in a learning algorithm by setting the value of ϵ'_0. More sophisticated ways of analyzing the performance of ADMIRAL-AE and using the results for learning faster and more effective decision-making policies is left to future work. An observation about ADMIRAL-AE is that, in MARL, the advisors can be used as a way to predict the behaviour of other agents as well, which is not relevant in single-agent settings. In MARL, each agent needs to have the ability to perform accurate opponent modelling, based on its observations, to obtain strong performances [98]. This is because the reward function and the transition dynamics depends on the joint action at each state. Previous methods have used several techniques, such as using a separate neural network for learning opponent behaviour [95], learning policy features from raw observations [103], and using the agent's own policy to predict opponent actions [197]. However, many of these methods are computationally expensive and scale poorly with the number of states, actions, and agents. Another possibility for opponent modelling is leveraging an external advisor that can possibly predict opponent behaviour, as done in ADMIRAL-AE, which could be relatively computationally friendly given the availability of an appropriate advisor. This could open up a very interesting research direction in learning from advisors in MARL. A natural extension of our work in Chapter 3 and Chapter 4 is to explore these algorithms in the context of human advisors and apply them to real world settings like autonomous driving and wildfire fighting.

Regarding our work on mean field learning, our work on different types in Chapter 5 can be extended to completely heterogeneous environments with agents having different capacities and objectives. StarCraft [206] is one example of such a domain. Our work would be well suited for this scenario as clustering would be even easier. Another approach would be to consider sub types, further dividing types. As future work for our work on partial observations in Chapter 6, we would like to relax some assumptions about the Bayesian approach using conjugate priors and make our analysis more generally applicable. Additionally, different observation distributions could allow the direction of view to determine the "viewable" agents, such as when agents in front of another agent are more likely to be seen than agents behind it. Our work on decentralized mean field learning in Chapter 7 can be readily applied to other real world environments such as autonomous driving and problems on demand and supply optimization. Hierarchical learning [260] approaches can be also be investigated in the context of both of our approaches in Chapter 5 and Chapter 6.

Our final chapter on combining action advising and mean field learning (Chapter 8) has the potential to spark an entire field of research exploring sample efficient MFRL algorithms. This is a very exciting avenue of research since there are countless application possibilities in large scale real world systems. Previous researchers have identified a list of critical real world applications where machine learning/RL could be useful [207]. This dissertation can contribute to almost all of these applications. As the world continues to become more globalized, mean field modelling would need to employed to study the wide ranging economic, political, and social effects of decisions that affect large populations. Particularly, from our side, as future work, we would like to extend the MFAA framework to allow agents to have access to more than one advisor. The multi-type version of our method (MT-MFAA) was restricted to known types. Extending to unknown types is left to future work.

From the theoretical perspective, some of the theoretical assumptions we considered in this dissertation may seem restrictive. However, we argue that this dissertation is the first to provide a theoretical foundation for MARL under action advising and mean field learning, and that these assumptions are useful in understanding the theoretical underpinnings of our approaches. Furthermore, we note that the assumptions we make are also made by other works, such as Hu and Wellman [104] and Yang et al. [303] (in fact, for some algorithms we relaxed assumptions from prior work as seen in Chapter 3). In future work, we wish to explore the ramifications of fully relaxing some of these assumptions. The theoretical understanding of MARL, in general, is still in its infancy and much more research into MARL theory is required to enhance our understanding of this area [313]. In this dissertation, we restrict the theoretical analysis to tabular settings, which is in line with the state-of-the-art in theoretical analysis of learning in general-sum stochastic games [313]. The objective is to provide a theoretical guarantee in the most basic (baseline) setting possible. Using a similar approach to single-agent RL methods that extend the tabular results to the function approximation setting [43], it would be possible to extend our theoretical results in this dissertation to the function approximation setting as well. An elaborate theoretical study of this is left to future work. Additionally, we restricted theoretical analysis to Q-learning based approaches in this dissertation. While we empirically studied policy gradient based approaches, it would be useful to perform an elaborate theoretical study of these methods. We leave this to future work.

References

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference of Machine Learning (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69, pages 1–9. ACM, 2004.

[2] Sachin Adlakha, Ramesh Johari, and Gabriel Y Weintraub. Equilibria of dynamic games with many players: Existence, approximation, and market structure. *Journal of Economic Theory*, 156:269–316, 2015.

[3] US Fire Administration. Fire statistics. https://www.usfa.fema.gov/data/statistics/, 2018. Accessed: 2018-01-01.

[4] Alan A Ager, Nicole M Vaillant, and Mark A Finney. Integrating fire behavior models and geospatial analysis for wildland fire risk assessment and fuel management planning. *Journal of Combustion*, 2011:60–75, 2011.

[5] Asma Al-Tamimi, Murad Abu-Khalaf, and Frank L Lewis. Adaptive critic designs for discrete-time zero-sum games with application to hcontrol. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(1):240–247, 2007.

[6] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli, and Daniela Rus. On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment. *Proceedings of National Academy of Science USA*, 114(3):462–467, 2017.

[7] Ofra Amir, Ece Kamar, Andrey Kolobov, and Barbara J. Grosz. Interactive Teaching Strategies for Agent Training. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI, New York, NY, USA, 9-15 July 2016*, pages 804–811. IJCAI/AAAI Press, 2016.

[8] Berkay Anahtarci, Can Deha Kariksiz, and Naci Saldi. Value iteration algorithm for mean-field games. *arXiv preprint arXiv:1909.01758*, 2019.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, Nov 2017.

[10] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[11] Christopher G. Atkeson and Stefan Schaal. Robot Learning From Demonstration. In *Proceedings of the Fourteenth International Conference on Machine Learning (ICML 1997), Nashville, Tennessee, USA, July 8-12, 1997*, pages 12–20. Morgan Kaufmann, 1997.

[12] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[14] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[15] Albert-László Barabási, Réka Albert, and Hawoong Jeong. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications*, 272(1-2):173–187, 1999.

[16] Elaheh Barati and Xuewen Chen. An actor-critic-attention mechanism for deep reinforcement learning in multi-view environments. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 2002–2008. ijcai.org, 2019.

[17] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

[18] Robert G Bartle. *The elements of integration and Lebesgue measure*. John Wiley & Sons, 2014.

[19] Ulrich Berger. Brown's original fictitious play. *Journal of Economic Theory*, 135(1):572–578, 2007.

[20] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[21] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

[22] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-dynamic programming*, volume 3 of *Optimization and neural computation series*. Athena Scientific, 1996.

[23] Sushrut Bhalla, Sriram Ganapathi Subramanian, and Mark Crowley. Deep Multi Agent Reinforcement Learning for Autonomous Driving. In *Canadian Conference on Artificial Intelligence*, volume LNAI 12109, page 17. Springer, Lecture Notes in Artificial Intelligence, 2020.

[24] Adam Bignold, Francisco Cruz, Matthew E. Taylor, Tim Brys, Richard Dazeley, Peter Vamplew, and Cameron Foale. A Conceptual Framework for Externally-influenced Agents: An Assisted Reinforcement Learning Review. *Journal of Ambient Intelligence and Humanized Computing*, 2021.

[25] Daan Bloembergen, Karl Tuyls, Daniel Hennes, and Michael Kaisers. Evolutionary dynamics of multi-agent learning: A survey. *Journal of Artificial Intelligence Research*, 53:659–697, 2015.

[26] Kenneth D. Bogert and Prashant Doshi. Multi-robot inverse reinforcement learning under occlusion with interactions. In *International conference on Autonomous Agents and Multi-Agent Systems, (AAMAS 2014), Paris, France, May 5-9, 2014*, pages 173–180. IFAAMAS, 2014.

[27] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative Entropy Inverse Reinforcement Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, (AISTATS 2011), Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 182–189. JMLR.org, 2011.

[28] Michael Bowling. Convergence problems of general-sum multiagent reinforcement learning. In *ICML*, pages 89–94, 2000.

[29] Michael Bowling. Convergence and no-regret in multiagent learning. In *Advances in neural information processing systems*, pages 209–216, 2005.

[30] Michael Bowling and Manuela Veloso. Rational and convergent learning in stochastic games. In *International joint conference on artificial intelligence*, volume 17, pages 1021–1026. Lawrence Erlbaum Associates Ltd, 2001.

[31] Michael Bowling and Manuela Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[32] Justin A Boyan and Andrew W Moore. Generalization in reinforcement learning: Safely approximating the value function. In *Advances in neural information processing systems*, pages 369–376, 1995.

[33] Emma Brunskill and Lihong Li. Sample complexity of multi-task reinforcement learning. *arXiv preprint arXiv:1309.6821*, 2013.

[34] Tim Brys, Anna Harutyunyan, Halit Bener Suay, Sonia Chernova, Matthew E. Taylor, and Ann Nowé. Reinforcement Learning from Demonstration through Shaping. In *IJCAI, Buenos Aires, Argentina, July 25-31, 2015*, pages 3352–3358. AAAI Press, 2015.

[35] Tim Brys, Anna Harutyunyan, Peter Vrancx, Matthew E. Taylor, Daniel Kudenko, and Ann Nowé. Multi-objectivization of reinforcement learning problems by reward shaping. In *2014 International Joint Conference on Neural Networks, IJCNN 2014, Beijing, China, July 6-11, 2014*, pages 2315–2322. IEEE, 2014.

[36] Lucian Bu, Robert Babu, Bart De Schutter, et al. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[37] Lucian Busoniu, Robert Babuska, and Bart De Schutter. Multi-agent reinforcement learning: A survey. In *2006 9th International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE, 2006.

[38] Lucian Busoniu, Robert Babuska, and Bart De Schutter. *Multi-agent Reinforcement Learning: An Overview*, volume 310, pages 183–221. Delft University of Technology, 07 2010.

[39] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part C*, 38(2):156–172, 2008.

[40] Pierre Cardaliaguet and Saeed Hadikhanloo. Learning in mean field games: the fictitious play. *ESAIM: Control, Optimisation and Calculus of Variations*, 23(2):569–591, 2017.

[41] René Carmona, Mathieu Laurière, and Zongjun Tan. Linear-quadratic mean-field reinforcement learning: convergence of policy gradient methods. *arXiv preprint arXiv:1910.04295*, 2019.

[42] René Carmona, Mathieu Laurière, and Zongjun Tan. Model-free mean-field reinforcement learning: mean-field mdp and mean-field q-learning. *arXiv preprint arXiv:1910.12802*, 2019.

[43] Diogo Carvalho, Francisco S. Melo, and Pedro Santos. A new convergent variant of Q-learning with linear function approximation. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems (NeurIPS 2020), December 6-12, 2020, virtual*, 2020.

[44] Nicolò Cesa-Bianchi, Alex Conconi, and Claudio Gentile. On the Generalization Ability of On-Line Learning Algorithms. In *Advances in Neural Information Processing Systems, (NeurIPS 2001), Vancouver, British Columbia, Canada, December 3-8, 2001*, pages 359–366. MIT Press, 2001.

[45] Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 709–716, 2003.

[46] Jessica Chemali and Alessandro Lazaric. Direct Policy Iteration with Demonstrations. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, (IJCAI 2015), Buenos Aires, Argentina, July 25-31, 2015*, pages 3380–3386. AAAI Press, 2015.

[47] Jinyoung Choi, Beom-Jin Lee, and Byoung-Tak Zhang. Multi-focus attention network for efficient deep reinforcement learning. In *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence, Saturday, February 4-9, 2017, San Francisco, California, USA*, volume WS-17 of *AAAI Technical Report*. AAAI Press, 2017.

[48] Felipe Codevilla, Matthias Müller, Antonio M. López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-End Driving via Conditional Imitation Learning. In *International Conference on Robotics and Automation, (ICRA 2018), Brisbane, Australia, May 21-25, 2018*, pages 1–9. IEEE, 2018.

[49] Felipe Codevilla, Eder Santana, Antonio M. López, and Adrien Gaidon. Exploring the Limitations of Behavior Cloning for Autonomous Driving. In *International Conference on Computer Vision, (ICCV 2019) Seoul, Korea (South), October 27 - November 2, 2019*, pages 9328–9337. IEEE, 2019.

[50] Vincent Conitzer and Tuomas Sandholm. AWESOME: A General Multiagent Learning Algorithm that Converges in Self-Play and Learns a Best Response Against Stationary Opponents. In *Proceedings of the Twentieth International Conference of Machine Learning, (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 83–90. AAAI Press, 2003.

[51] Miguel Cruz and Martin Alexander. Assessing Crown Fire Potential in Coniferous Forests of Western North America: A Critique of Current Approaches and Recent Simulation Studies. *The Bark Beetles, Fuels, and Fire Bibliography*, 19, 01 2010.

[52] Miguel Cruz and Martin Alexander. Assessing crown fire potential in coniferous forests of western north america: A critique of current approaches and recent simulation studies. *The Bark Beetles, Fuels, and Fire Bibliography*, 19, 01 2010.

[53] Aaron Delwiche. Massively multiplayer online games (mmos) in the new media classroom. *Journal of Education Technology and Society*, 9(3):160–172, 2006.

[54] Sam Devlin, Daniel Kudenko, and Marek Grześ. An empirical study of potential-based reward shaping and advice in complex, multi-agent systems. *Advances in Complex Systems*, 14(02):251–278, 2011.

[55] Thomas G. Dietterich. Ensemble Methods in Machine Learning. In *Multiple Classifier Systems, First International Workshop, (MCS 2000), Cagliari, Italy, June 21-23, 2000, Proceedings*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.

[56] Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

[57] Bruno Dufay and Jean-Claude Latombe. An approach to automatic robot programming based on inductive learning. *The International journal of robotics research*, 3(4):3–20, 1984.

[58] Gabriel Dulac-Arnold, Nir Levine, Daniel J Mankowitz, Jerry Li, Cosmin Paduraru, Sven Gowal, and Todd Hester. Challenges of real-world reinforcement learning: Definitions, Benchmarks and Analysis. *Machine Learning*, 1:1–50, 2021.

[59] Romuald Elie, Julien Pérolat, Mathieu Laurière, Matthieu Geist, and Olivier Pietquin. Approximate fictitious play for mean field games. *arXiv preprint arXiv:1907.02633*, 2019.

[60] Romuald Elie, Julien Perolat, Mathieu Laurière, Matthieu Geist, and Olivier Pietquin. On the convergence of model free learning in mean field games. In *AAAI Conference on Artificial Intelligence (AAAI 2020)*, 2020.

[61] Robert Elliott, Xun Li, and Yuan-Hua Ni. Discrete time mean-field stochastic linear-quadratic optimal control problems. *Automatica*, 49(11):3222–3233, 2013.

[62] Canan Eryigit. Marketing Models: A Review of the Literature. *International Journal of Market Research*, 59(3):355–381, 2017.

[63] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *International Conference of Machine Learning*, 2018.

[64] Eyal Even-Dar and Yishay Mansour. Learning Rates for Q-learning. *Journal of Machine Learning Research*, 5:1–25, 2003.

[65] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 720–727, 2006.

[66] Fernando Fernández and Manuela M. Veloso. Probabilistic policy reuse in a reinforcement learning agent. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss, and Peter Stone, editors, *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, pages 720–727. ACM, 2006.

[67] Arlington M Fink et al. Equilibrium in a stochastic n-person game. *Journal of Science of the Hiroshima University, Series AI (Mathematics)*, 28(1):89–93, 1964.

[68] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A Connection between Generative Adversarial Networks, Inverse Reinforcement Learning, and Energy-Based Models. *arXiv preprint arXiv:1611.03852*, 2016.

[69] Mark A Finney. *FARSITE, Fire Area Simulator–model development and evaluation*. US Department of Agriculture, Forest Service, Rocky Mountain Research Station, 1998.

[70] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2137–2145. Curran Associates, Inc., 2016.

[71] Jakob Foerster, Ioannis Alexandros Assael, Nando De Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in neural information processing systems*, pages 2137–2145, 2016.

[72] Jakob Foerster, Richard Y Chen, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, and Igor Mordatch. Learning with opponent-learning awareness. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 122–130, Stockholm, Sweden, 2018. International Foundation for Autonomous Agents and Multiagent Systems, Multi Robot Systems.

[73] Justin Fu, Katie Luo, and Sergey Levine. Learning Robust Rewards with Adverserial Inverse Reinforcement Learning. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018*. OpenReview.net, 2018.

[74] Zuyue Fu, Zhuoran Yang, Yongxin Chen, and Zhaoran Wang. Actor-critic provably finds nash equilibria of linear-quadratic mean-field games. In *ICLR*, 2019.

[75] Drew Fudenberg, Fudenberg Drew, David K Levine, and David K Levine. *The theory of learning in games*, volume 2. MIT press, 1998.

[76] Sumitra Ganesh, Nelson Vadori, Mengda Xu, Hua Zheng, Prashant Reddy, and Manuela Veloso. Reinforcement learning for market making in a multi-agent dealer market. *arXiv preprint arXiv:1911.05892*, 2019.

[77] Tanmay Gangwani, Yuan Zhou, and Jian Peng. Learning Guidance Rewards with Trajectory-space Smoothing. In *Advances in Neural Information Processing Systems (NeurIPS 2020), December 6-12, 2020, virtual*, 2020.

[78] Chao Gao, Bilal Kartal, Pablo Hernandez-Leal, and Matthew E. Taylor. On Hard Exploration for Reinforcement Learning: A Case Study in Pommerman. In *Proceedings of the Fifteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, (AIIDE 2019), October 8-12, 2019, Atlanta, Georgia, USA*, pages 24–30. AAAI Press, 2019.

[79] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement Learning from Imperfect Demonstrations. In *6th International Conference on Learning Representations, (ICLR 2018), Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018.

[80] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. Reinforcement learning with multiple experts: A bayesian model combination approach. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 9549–9559, 2018.

[81] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. Contextual policy transfer in reinforcement learning domains via deep mixtures-of-experts. In Cassio P. de Campos, Marloes H. Maathuis, and Erik Quaeghebeur, editors, *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence, UAI 2021, Virtual Event, 27-30 July 2021*, volume 161 of *Proceedings of Machine Learning Research*, pages 1787–1797. AUAI Press, 2021.

[82] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters*, 1(2):661–667, 2015.

[83] Vinicius G. Goecks, Gregory M. Gremillion, Vernon J. Lawhern, John Valasek, and Nicholas R. Waytowich. Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, (AAMAS 2020), Auckland, New Zealand, May 9-13, 2020*, pages 465–473. IFAAMAS, 2020.

[84] Diogo A Gomes, Joana Mohr, and Rafael Rigao Souza. Discrete time, finite state space mean field games. *Journal de mathématiques pures et appliquées*, 93(3):308–328, 2010.

[85] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014.

[86] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L. Isbell Jr., and Andrea Lockerd Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 2625–2633, 2013.

[87] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *International Conference on Machine Learning*, pages 2829–2838, 2016.

[88] Xin Guo, Anran Hu, Renyuan Xu, and Junzi Zhang. Learning mean-field games. In *Advances in Neural Information Processing Systems*, pages 4967–4977, 2019.

[89] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

[90] Dylan Hadfield-Menell, Stuart J. Russell, Pieter Abbeel, and Anca D. Dragan. Cooperative Inverse Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, December 5-10, 2016*, pages 3909–3917, 2016.

[91] Saeed Hadikhanloo and Francisco J Silva. Finite mean field games: fictitious play and convergence to a first order continuous mean field game. *Journal de Mathématiques Pures et Appliquées*, 132:369–397, 2019.

[92] Bruce Hajek and Maxim Raginsky. Statistical learning theory. *Lecture Notes*, 387, 2019.

[93] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015.

[94] Elad Hazan, Amit Agarwal, and Satyen Kale. Logarithmic regret algorithms for online convex optimization. *Machine Learning*, 69(2-3):169–192, 2007.

[95] He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daum. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pages 1804–1813, New York City, USA, 2016.

[96] Bassam Helou, Aditya Dusi, Anne Collin, Noushin Mehdipour, Zhiliang Chen, Cristhian Lizarazo, Calin Belta, Tichakorn Wongpiromsarn, Radboud Duintjer Tebbens, and Oscar Beijbom. The Reasonable Crowd: Towards evidence-based and interpretable models of driving behavior. In *IROS*, pages 6708–6715. IEEE, 2021.

[97] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[98] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E. Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33(6):750–797, 2019.

[99] Onésimo Hernández-Lerma and Jean B Lasserre. *Discrete-time Markov control processes: basic optimality criteria*, volume 30. Springer Science & Business Media, 2012.

[100] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[101] Karl Hinderer. Decision models. In *Foundations of Non-stationary Dynamic Programming with Discrete Time Parameter*, pages 78–83. Springer, 1970.

[102] Jonathan Ho and Stefano Ermon. Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2016), Barcelona, Spain, December 5-10, 2016*, pages 4565–4573, 2016.

[103] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. A Deep Policy Inference Q-Network for Multi-Agent Systems. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018 Stockholm, Sweden, July 10-15, 2018*, pages 1388–1396. IFAAMAS, 2018.

[104] Junling Hu and Michael P Wellman. Nash q-learning for general-sum stochastic games. *Journal of Machine Learning Research*, 4(Nov):1039–1069, 2003.

[105] Yue Hu, Juntao Li, Xi Li, Gang Pan, and Mingliang Xu. Knowledge-Guided Agent-Tactic-Aware Learning for Starcraft Micromanagement. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, (IJCAI 2018), July 13-19, 2018, Stockholm, Sweden*, pages 1471–1477. ijcai.org, 2018.

[106] Minyi Huang. Large-population lqg games involving a major player: the nash certainty equivalence principle. *SIAM Journal on Control and Optimization*, 48(5):3318–3353, 2010.

[107] Minyi Huang, Peter E Caines, and Roland P Malhamé. Individual and mass behaviour in large population stochastic wireless power control problems: centralized and nash equilibrium solutions. In *42nd IEEE International Conference on Decision and Control*. IEEE, 2003.

[108] Minyi Huang, Roland P Malhamé, Peter E Caines, et al. Large population stochastic dynamic games: closed-loop mckean-vlasov systems and the nash certainty equivalence principle. *Communications in Information & Systems*, 6(3):221–252, 2006.

[109] Mostafa Hussein, Brendan Crowe, Marek Petrik, and Momotaz Begum. Robust Maximum Entropy Behavior Cloning. *arXiv preprint arXiv:2101.01251*, 2021.

[110] Shariq Iqbal and Fei Sha. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970. PMLR, 2019.

[111] Tommi Jaakkola, Michael I. Jordan, and Satinder P. Singh. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, 6(6):1185–1201, 1994.

[112] Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[113] Roghayeh Jahdi, Michele Salis, Ali A Darvishsefat, Fermin Alcasena, Mir A Mostafavi, V Etemad, Olga M Lozano, and Donatella Spano. Evaluating fire modelling systems in recent wildfires of the golestan national park, iran. *Forestry*, 89(2):136–149, 2015.

[114] Sebastian Jaimungal, Mojtaba Nourian, and Xuancheng Huang. Mean-field game strategies for a major-minor agent optimal execution problem. *Available at SSRN*, 2578733, 2015.

[115] Piyush Jain, Sean CP Coogan, Sriram Ganapathi Subramanian, Mark Crowley, Steve Taylor, and Mike D Flannigan. A review of machine learning applications in wildfire science and management. *arXiv preprint arXiv:2003.00646*, 2020.

[116] Piyush Jain, Sean C.P. Coogan, Sriram Ganapathi Subramanian, Mark Crowley, Steve Taylor, and Mike D. Flannigan. A review of machine learning applications in wildfire science and management. *Environmental Reviews*, 28(4):478–505, 2020.

[117] Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montreal, Canada*, pages 7265–7275, 2018.

[118] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. *arXiv preprint arXiv:1907.05388*, 2019.

[119] Mingxuan Jing, Xiaojian Ma, Wenbing Huang, Fuchun Sun, Chao Yang, Bin Fang, and Huaping Liu. Reinforcement Learning from Imperfect Demonstrations under Soft Expert Guidance. In *The Thirty-Fourth Conference of Association for the Advancement of Artificial Intelligence (AAAI 2020), New York, NY, USA, February 7-12, 2020*, pages 5109–5116. AAAI Press, 2020.

[120] Sham Machandranath Kakade. *On the sample complexity of reinforcement learning.* PhD thesis, UCL (University College London), 2003.

[121] Shizuo Kakutani. A generalization of brouwer's fixed point theorem. *Duke mathematical journal*, 8(3):457–459, 1941.

[122] Ehud Kalai and Ehud Lehrer. Rational learning leads to nash equilibrium. *Econometrica: Journal of the Econometric Society*, pages 1019–1045, 1993.

[123] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[124] Peter Karkus, David Hsu, and Wee Sun Lee. QMDP-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4694–4704, 2017.

[125] Dong-Ki Kim, Miao Liu, Shayegan Omidshafiei, Sebastian Lopez-Cot, Matthew Riemer, Golnaz Habibi, Gerald Tesauro, Sami Mourad, Murray Campbell, and Jonathan P. How. Learning Hierarchical Teaching Policies for Cooperative Agents. In *AAMAS, Auckland, New Zealand, May 9-13, 2020*, pages 620–628. IFAAMAS, 2020.

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

[127] Arman C Kizilkale and Peter E Caines. Mean field stochastic adaptive control. *IEEE Transactions on Automatic Control*, 58(4):905–920, 2012.

[128] Arman C. Kizilkale and Peter E. Caines. Mean field stochastic adaptive control. *IEEE Trans. Autom. Control.*, 58(4):905–920, 2013.

[129] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[130] Vijay R. Konda and John N. Tsitsiklis. Actor-Critic Algorithms. In *NeurIPS*. The MIT Press, 1999.

[131] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[132] George Konidaris and Andrew Barto. Autonomous shaping: Knowledge transfer in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 489–496, 2006.

[133] Alex Kuefler, Jeremy Morton, Tim Wheeler, and Mykel Kochenderfer. Imitating Driver Behavior with Generative Adversarial Networks. In *IEEE Intelligent Vehicles Symposium (IV 2017)*, pages 204–211. IEEE, 2017.

[134] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Julien Perolat, David Silver, Thore Graepel, et al. A unified game-theoretic approach to multiagent reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 4190–4203, 2017.

[135] Michael Laskey, Jonathan Lee, Roy Fox, Anca D. Dragan, and Ken Goldberg. DART: Noise Injection for Robust Imitation Learning. In *1st Annual Conference on Robot Learning, (CoRL 2017), Mountain View, California, USA, 2017*, volume 78 of *Proceedings of Machine Learning Research*, pages 143–156. PMLR, 2017.

[136] Jean-Michel Lasry and Pierre-Louis Lions. Mean field games. *Japanese journal of mathematics*, 2(1):229–260, 2007.

[137] Tor Lattimore and Csaba Szepesvari. *Bandit algorithms*. Cambridge University Press, 2020.

[138] Adam Daniel Laud. *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.

[139] Hoang Minh Le, Yisong Yue, Peter Carr, and Patrick Lucey. Coordinated Multi-Agent Imitation Learning. In *Proceedings of the 34th International Conference on Machine Learning, (ICML 2017), Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1995–2003. PMLR, 2017.

[140] Brigitte Leblon, Laura Bourgeau-Chavez, and Jesús San-Miguel-Ayanz. Use of remote sensing in wildfire management. In *Sustainable Development-Authoritative and Leading Edge Content for Environmental Management*. IntechOpen, 2012.

[141] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[142] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

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

[144] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear Inverse Reinforcement Learning with Gaussian Processes. In *Advances in Neural Information Processing Systems (NeurIPS 2011) Granada, Spain, December 2011*, pages 19–27, 2011.

[145] Mao Li, Yi Wei, and Daniel Kudenko. Two-level q-learning: learning from conflict demonstrations. *The Knowledge Engineering Review*, 34, 2019.

[146] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning. In *The World Wide Web Conference, WWW 2019*. ACM, 2019.

[147] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations*, 2016.

[148] Xiaomin Lin, Stephen C Adams, and Peter A Beling. Multi-agent inverse reinforcement learning for certain general-sum stochastic games. *Journal of Artificial Intelligence Research (JAIR)*, 66:473–502, 2019.

[149] Xiaomin Lin, Peter A Beling, and Randy Cogill. Multiagent inverse reinforcement learning for two-person zero-sum games. *IEEE Transactions on Games*, 10(1):56–68, 2017.

[150] Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In William W. Cohen and Haym Hirsh, editors, *Machine Learning, Proceedings of the Eleventh International Conference, Rutgers University, New Brunswick, NJ, USA, July 10-13, 1994*, pages 157–163. Morgan Kaufmann, 1994.

[151] Michael L. Littman. Friend-or-Foe Q-learning in General-Sum Games. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), Williams College, Williamstown, MA, USA, June 28 - July 1, 2001*, pages 322–328. Morgan Kaufmann, 2001.

[152] Michael L Littman. Value-function reinforcement learning in markov games. *Cognitive Systems Research*, 2(1):55–66, 2001.

[153] ML Littman and AW Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

[154] Qinghua Liu, Tiancheng Yu, Yu Bai, and Chi Jin. A Sharp Analysis of Model-based Reinforcement Learning with Self-play. In *ICML, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 7001–7010. PMLR, 2021.

[155] Meghna Lowalekar, Pradeep Varakantham, and Patrick Jaillet. ZAC: A zone path construction approach for effective real-time ridesharing. In *ICAPS*, pages 528–538. AAAI Press, 2019.

[156] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in neural information processing systems*, pages 6379–6390, 2017.

[157] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56, 2016.

[158] Ofir Marom and Benjamin Rosman. Belief Reward Shaping in Reinforcement Learning. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 3762–3769. AAAI Press, 2018.

[159] Eric Maskin. Nash equilibrium and welfare optimality. *The Review of Economic Studies*, 66(1):23–38, 1999.

[160] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems. *The Knowledge Engineering Review*, 27(1):1–31, 2012.

[161] Hardik Meisheri, Omkar Shelke, Richa Verma, and Harshad Khadilkar. Accelerating training in Pommerman with imitation and reinforcement learning. *arXiv preprint arXiv:1911.04947*, 2019.

[162] David Mguni, Joel Jennings, and Enrique Munoz de Cote. Decentralised learning in systems with many, many strategic agents. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[163] David Mguni, Joel Jennings, and Enrique Munoz de Cote. Decentralised learning in systems with many, many strategic agents. In *AAAI*, pages 4686–4693. AAAI Press, 2018.

[164] David Mguni, Joel Jennings, Emilio Sison, Sergio Valcarcel Macua, Sofia Ceppi, and Enrique Munoz de Cote. Coordinating the crowd: Inducing desirable equilibria in non-cooperative systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 386–394. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[165] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral Normalization for Generative Adversarial Networks. In *6th International Conference on Learning Representations, (ICLR 2018), Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[166] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937, 2016.

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

[168] Alexander Mott, Daniel Zoran, Mike Chrzanowski, Daan Wierstra, and Danilo Jimenez Rezende. Towards interpretable reinforcement learning using attention augmented agents. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12329–12338, 2019.

[169] Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. *arXiv preprint arXiv:1812.07671*, 2018.

[170] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming Exploration in Reinforcement Learning with Demonstrations. In *2018 IEEE International Conference on Robotics and Automation, (ICRA 2018) Brisbane, Australia, May 21-25, 2018*, pages 6292–6299. IEEE, 2018.

[171] John Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[172] John F Nash. *Non-Cooperative Games.* Princeton University Press, 1951.

[173] Sriraam Natarajan, Gautam Kunapuli, Kshitij Judah, Prasad Tadepalli, Kristian Kersting, and Jude W. Shavlik. Multi-Agent Inverse Reinforcement Learning. In *The Ninth International Conference on Machine Learning and Applications, (ICMLA 2010), Washington, DC, USA, 12-14 December 2010*, pages 395–400. IEEE Computer Society, 2010.

[174] J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische annalen*, 100(1):295–320, 1928.

[175] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*, pages 278–287. Morgan Kaufmann, 1999.

[176] Andrew Y. Ng and Stuart J. Russell. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 663–670. Morgan Kaufmann, 2000.

[177] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multi-agent systems: a review of challenges, solutions and applications. *arXiv preprint arXiv:1812.11794*, 2018.

[178] Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE transactions on cybernetics*, 2020.

[179] V Nikitin, S Golubin, R Belov, V Gusev, and N Andrianov. Development of a robotic vehicle complex for wildfire-fighting by means of fire-protection roll screens. In *IOP Conference Series: Earth and Environmental Science*, volume 226, page 012003. IOP Publishing, 2019.

[180] Ann Nowé, Peter Vrancx, and Yann-Michaël De Hauwere. Game theory and multi-agent reinforcement learning. In *Reinforcement Learning*, pages 441–470. Springer, 2012.

[181] NYYellowTaxi. New York yellow taxi dataset. `http://www.nyc.gov/html/tlc/html/about/triprecorddata.shtml`, 2016. [Online; accessed 19-June-2021].

[182] Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 2790–2799. JMLR.org, 2016.

[183] Shayegan Omidshafiei, Dong-Ki Kim, Miao Liu, Gerald Tesauro, Matthew Riemer, Christopher Amato, Murray Campbell, and Jonathan P. How. Learning to Teach

in Cooperative Multiagent Reinforcement Learning. In *Thirty-Third Conference of Association for the Advancement of Artificial Intelligence (AAAI-19), Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 6128–6136. AAAI Press, 2019.

[184] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[185] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[186] Sindhu Padakandla. A survey of reinforcement learning algorithms for dynamically varying environments. *ACM Comput. Surv.*, 54(6):127:1–127:25, 2021.

[187] Peixi Peng, Junliang Xing, and Lili Cao. Hybrid Learning for Multi-agent Cooperation with Sub-optimal Demonstrations. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, (IJCAI-2020)*, pages 3037–3043. International Joint Conferences on Artificial Intelligence Organization, 7 2020.

[188] Sarah Perrin, Mathieu Laurière, Julien Pérolat, Matthieu Geist, Romuald Élie, and Olivier Pietquin. Mean field games flock! the reinforcement learning way. In *IJCAI*, pages 356–362, 2021.

[189] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[190] Connie Phan and Hugh HT Liu. A cooperative UAV/UGV platform for wildfire detection and fighting. In *Asia Simulation Conference-7th International Conference on System Simulation and Scientific Computing*, pages 494–498. IEEE, 2008.

[191] Bilal Piot, Matthieu Geist, and Olivier Pietquin. Boosted and reward-regularized classification for apprenticeship learning. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '14, Paris, France, May 5-9, 2014*, pages 1249–1256. IFAAMAS/ACM, 2014.

[192] Dean Pomerleau. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Advances in Neural Information Processing Systems (NeurIPS 1988) Denver, Colorado, USA, 1988*, pages 305–313. Morgan Kaufmann, 1988.

[193] Dean A Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

[194] Bob Price and Craig Boutilier. Accelerating Reinforcement Learning through Implicit Imitation. *Journal of Artificial Intelligence Research*, 19:569–629, 2003.

[195] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[196] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

[197] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. Modeling Others using Oneself in Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4254–4263. PMLR, 2018.

[198] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

[199] Carl Edward Rasmussen. Gaussian Processes in Machine Learning. In *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 63–71. Springer, 2003.

[200] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum Margin Planning. In *Proceedings of the Twenty-Third International Conference on Machine Learning (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*, volume 148 of *ACM International Conference Proceeding Series*, pages 729–736. ACM, 2006.

[201] Nathan D. Ratliff, David M. Bradley, J. Andrew Bagnell, and Joel E. Chestnutt. Boosting Structured Prediction for Imitation Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2006) Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 1153–1160. MIT Press, 2006.

[202] Tummalapalli Sudhamsh Reddy, Vamsikrishna Gopikrishna, Gergely V. Záruba, and Manfred Huber. Inverse reinforcement learning for decentralized non-cooperative multiagent systems. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC 2012), Seoul, Korea (South), October 14-17, 2012*, pages 1930–1935. IEEE, 2012.

[203] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A Multi-Agent Playground. In *arXiv preprint arXiv:1809.07124*, 2018.

[204] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *CoRR*, abs/1809.07124, 2018.

[205] P Ring. Relativistic mean field theory in finite nuclei. *Progress in Particle and Nuclear Physics*, 37:193–263, 1996.

[206] Sebastian Risi and Mike Preuss. Behind deepmind's alphastar ai that reached grandmaster level in starcraft ii. *KI-Künstliche Intelligenz*, 34(1):85–86, 2020.

[207] David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *arXiv preprint arXiv:1906.05433*, 2019.

[208] Stefan Ropke and Jean-François Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[209] Stéphane Ross, Geoffrey J. Gordon, and Drew Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011), Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 627–635. JMLR.org, 2011.

[210] Richard C Rothermel. *A mathematical model for predicting fire spread in wildland fuels*, volume 115. Intermountain Forest & Range Experiment Station, Forest Service, US, 1972.

[211] Naci Saldi, Tamer Basar, and Maxim Raginsky. Discrete-time risk-sensitive mean-field games. *arXiv preprint arXiv:1808.03929*, 2018.

[212] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[213] Stefan Schaal. Learning from Demonstration. In *Advances in Neural Information Processing Systems (NeurIPS 1996), Denver, CO, USA, December 2-5, 1996*, pages 1040–1046. MIT Press, 1996.

[214] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[215] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

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

[217] Sanket Shah, Meghna Lowalekar, and Pradeep Varakantham. Neural Approximate Dynamic Programming for On-Demand Ride-Pooling. In *AAAI*, pages 507–515. AAAI Press, 2020.

[218] Shai Shalev-Shwartz and Sham M. Kakade. Mind the Duality Gap: Logarithmic regret algorithms for online optimization. In *Advances in Neural Information Processing Systems (NeurIPS 2008), Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 1457–1464. Curran Associates, Inc., 2008.

[219] Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

[220] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[221] Yoav Shoham and Kevin Leyton-Brown. *Multiagent Systems - Algorithmic, Game-Theoretic, and Logical Foundations.* Cambridge University Press, 2009.

[222] Yoav Shoham, Rob Powers, and Trond Grenager. Multi-agent reinforcement learning: a critical survey. *Web manuscript*, 2003.

[223] Yoav Shoham, Rob Powers, and Trond Grenager. If multi-agent learning is the answer, what is the question? *Artificial intelligence*, 171(7):365–377, 2007.

[224] Satish Shukla, Sriram Balasubramanian, and Mirjana Pavlović. A generalized Banach fixed point theorem. *Bulletin of the Malaysian Mathematical Sciences Society*, 39(4):1529–1539, 2016.

[225] Felipe Leno Da Silva and Anna Helena Reali Costa. A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems. *Journal of Artificial Intelligence Research (JAIR)*, 64:645–703, 2019.

[226] Felipe Leno Da Silva, Ruben Glatt, and Anna Helena Reali Costa. Simultaneously Learning and Advising in Multiagent Reinforcement Learning. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, (AAMAS 2017), São Paulo, Brazil, May 8-12, 2017*, pages 1100–1108. ACM, 2017.

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

[228] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[229] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38(3):287–308, 2000.

[230] Satinder P Singh, Michael J Kearns, and Yishay Mansour. Nash convergence of gradient dynamics in general-sum games. In *UAI*, pages 541–548, 2000.

[231] Tomah Sogabe, Dinesh Bahadur Malla, Shota Takayama, Seiichi Shin, Katsuyoshi Sakamoto, Koichi Yamaguchi, Thakur Praveen Singh, Masaru Sogabe, Tomohiro Hirata, and Yoshitaka Okada. Smart grid optimization by deep reinforcement learning over discrete and continuous action space. In *2018 IEEE 7th World Conference on Photovoltaic Energy Conversion (WCPEC)(A Joint Conference of 45th IEEE PVSC, 28th PVSEC & 34th EU PVSEC)*, pages 3794–3796. IEEE, 2018.

[232] Aaron Sonabend, Junwei Lu, Leo Anthony Celi, Tianxi Cai, and Peter Szolovits. Expert-Supervised Reinforcement Learning for Offline Policy Learning and Evaluation. In *Advances in Neural Information Processing Systems (NeurIPS 2020), virtual, December 6-12, 2020*, 2020.

[233] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-Agent Generative Adversarial Imitation Learning. In *Advances in Neural Information Processing Systems (NeurIPS 2018), Montreal, Canada, December 3-8, 2018,*, pages 7472–7483, 2018.

[234] Ziang Song, Song Mei, and Yu Bai. When Can We Learn General-Sum Markov Games with a Large Number of Players Sample-Efficiently? *arXiv preprint arXiv:2110.04184*, 2021.

[235] Sriram Srinivasan, Marc Lanctot, Vinicius Zambaldi, Julien Pérolat, Karl Tuyls, Rémi Munos, and Michael Bowling. Actor-critic policy optimization in partially observable multiagent environments. In *Advances in neural information processing systems*, pages 3422–3435, 2018.

[236] H Eugene Stanley. Phase transitions and critical phenomena. clarendon, 1971.

[237] H Eugene Stanley and Guenter Ahlers. Introduction to phase transitions and critical phenomena. *Physics Today*, 26:71, 1973.

[238] Statista. Mmo Gaming - Statistics & Facts. https://www.statista.com/topics/2290/mmo-gaming, 2021. Online; accessed 29 January 2022.

[239] Peter Stone. Multiagent learning is not the answer. it is the question. *Artificial Intelligence*, 171(7):402–405, 2007.

[240] Peter Stone and Manuela M. Veloso. Multiagent systems: A survey from a machine learning perspective. *Auton. Robots*, 8(3):345–383, 2000.

[241] Kaj Storbacka and Ted Moser. The changing role of marketing: transformed propositions, processes and partnerships. *AMS Review*, 10(3):299–310, 2020.

[242] Joseph Suarez, Yilun Du, Clare Zhu, Igor Mordatch, and Phillip Isola. The Neural MMO platform for Massively Multiagent Research. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[243] Jayakumar Subramanian and Aditya Mahajan. Reinforcement learning in stationary mean-field games. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 251–259. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[244] Sriram Ganapathi Subramanian. Multi Type Mean Field Reinforcement Learning. https://github.com/BorealisAI/mtmfrl, 2020.

[245] Sriram Ganapathi Subramanian. Partially Observable Mean Field Reinforcement Learning. https://github.com/Sriram94/pomfrl, 2020.

[246] Sriram Ganapathi Subramanian. Decentralized Mean Field Games. https://github.com/Sriram94/DMFG, 2021.

[247] Sriram Ganapathi Subramanian. Multi-Agent Advisor Q-Learning. https://github.com/Sriram94/multiagentadvisorqlearning, 2022.

[248] Sriram Ganapathi Subramanian, Pascal Poupart, Matthew E. Taylor, and Nidhi Hegde. Multi Type Mean Field Reinforcement Learning. In *Proceedings of the Autonomous Agents and Multi Agent Systems (AAMAS 2020)*, Auckland, New Zealand, 9–13 May 2020. IFAAMAS.

[249] Sriram Ganapathi Subramanian, Pascal Poupart, Matthew E. Taylor, and Nidhi Hegde. Multi Type Mean Field Reinforcement Learning. *arXiv preprint arXiv:2002.02513*, 2020.

[250] Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, and Pascal Poupart. Partially Observable Mean Field Reinforcement Learning. *arXiv preprint arXiv:2012.15791*, 2020.

[251] Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, and Pascal Poupart. Decentralized Mean Field Games. *arXiv preprint arXiv:2112.09099*, 2021.

[252] Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, and Pascal Poupart. Partially Observable Mean Field Reinforcement Learning. In *AAMAS*. ACM, 2021.

[253] Sriram Ganapathi Subramanian, Matthew E. Taylor, Mark Crowley, and Pascal Poupart. Decentralized Mean Field Games. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 36, pages 9439–9447, Vancouver, Canada, 2022. AAAI Press.

[254] Sriram Ganapathi Subramanian, Matthew E. Taylor, Kate Larson, and Mark Crowley. Multi-Agent Advisor Q-Learning. *arXiv preprint arXiv:2111.00345*, 2021.

[255] Sriram Ganapathi Subramanian, Matthew E. Taylor, Kate Larson, and Mark Crowley. Multi-Agent Advisor Q-Learning. *Journal of Artificial Intelligence Research*, 74:1–74, 2022.

[256] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[257] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *Advances in Neural Information Processing Systems 12, [NuerIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*, pages 1057–1063. The MIT Press, 1999.

[258] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, pages 1057–1063. The MIT Press, 1999.

[259] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[260] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[261] Umar Syed and Robert E. Schapire. A Reduction from Apprenticeship Learning to Classification. In *Advances in Neural Information Processing Systems (NeurIPS 2010) Vancouver, British Columbia, Canada, December 6-9, 2010*, pages 2253–2261. Curran Associates, Inc., 2010.

[262] Csaba Szepesvári and Michael L Littman. A unified analysis of value-function-based reinforcement-learning algorithms. *Neural Computation*, 11(8):2017–2060, 1999.

[263] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PloS one*, 12(4), 2017.

[264] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[265] Ming Tan. Multi-agent reinforcement learning: Independent vs Cooperative learning. *Readings in Agents*, pages 487–494, 1997.

[266] Kazuo Tanaka, Tsuyoshi Hori, and Hua O Wang. A multiple Lyapunov function approach to stabilization of fuzzy control systems. *IEEE Transactions on fuzzy systems*, 11(4):582–589, 2003.

[267] Matthew E. Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research (JMLR)*, 10(7), 2009.

[268] Matthew E. Taylor, Halit Bener Suay, and Sonia Chernova. Integrating reinforcement learning with human demonstrations of varying ability. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 617–624. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[269] Justin K Terry, Benjamin Black, Mario Jayakumar, Ananth Hari, Luis Santos, Clemens Dieffendahl, Niall L Williams, Yashas Lokesh, Ryan Sullivan, Caroline Horsch, and Praveen Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning. In *arXiv preprint arXiv:2009.14471*, 2020.

[270] Evangelos A. Theodorou, Jonas Buchli, and Stefan Schaal. Reinforcement learning of motor skills in high dimensions: A path integral approach. In *International Conference on Robotics and Automation, (ICRA 2010), Anchorage, Alaska, USA, 3-7 May 2010*, pages 2397–2403. IEEE, 2010.

[271] Matthew P. Thompson, Christopher J. Lauer, David E. Calkin, Jon D. Rieck, Crystal S. Stonesifer, and Michael S. Hand. Wildfire Response Performance Measurement: Current and Future Directions. *Fire*, 1(2), 2018.

[272] Matthew P Thompson, Yu Wei, David E Calkin, Christopher D O'Connor, Christopher J Dunn, Nathaniel M Anderson, and John S Hogland. Risk management and analytics in wildfire response. *Current Forestry Reports*, 5(4):226–239, 2019.

[273] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[274] Lisa Torrey and Matthew E. Taylor. Teaching on a budget: agents advising agents in reinforcement learning. In *International conference on Autonomous Agents and Multi-Agent Systems, AAMAS '13, Saint Paul, MN, USA, May 6-10, 2013*, pages 1053–1060. IFAAMAS, 2013.

[275] Alan Tsang, Kate Larson, and Rob McAlpine. Resource sharing for control of wildland fires. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[276] John N Tsitsiklis. Asynchronous stochastic approximation and q-learning. *Machine learning*, 16(3):185–202, 1994.

[277] Kagan Tumer and Adrian K. Agogino. Distributed agent-based air traffic flow management. In Edmund H. Durfee, Makoto Yokoo, Michael N. Huhns, and Onn Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 255. IFAAMAS, 2007.

[278] Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, and prospects. *Ai Magazine*, 33(3):41–41, 2012.

[279] Cordy Tymstra, Brian J Stocks, Xinli Cai, and Mike D Flannigan. Wildfire management in canada: Review, challenges and opportunities. *Progress in Disaster Science*, 5:100045, 2020.

[280] Hado van Hasselt. Double Q-learning. In *NIPS, Vancouver, British Columbia, Canada*, pages 2613–2621. Curran Associates, Inc., 2010.

[281] Robert J Vanderbei et al. *Linear programming*. Springer, 2020.

[282] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[283] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[284] John Von Neumann and Oskar Morgenstern. Theory of games and economic behavior, 2nd rev. 1947.

[285] Kiri Wagstaff. Machine learning that matters. *arXiv preprint arXiv:1206.4656*, 2012.

[286] Xingyu Wang and Diego Klabjan. Competitive Multi-agent Inverse Reinforcement Learning with Sub-optimal Demonstrations. In *Proceedings of the Thirty-fifth International Conference on Machine Learning (ICML 2018), Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 5130–5138. PMLR, 2018.

[287] Yixi Wang, Wenhuan Lu, Jianye Hao, Jianguo Wei, and Ho-fung Leung. Efficient Convention Emergence through Decoupled Reinforcement Social Learning with Teacher-Student Mechanism. In *AAMAS, Stockholm, Sweden, July 10-15, 2018*, pages 795–803. IFAAMAS / ACM, 2018.

[288] Zhaodong Wang, Zhiwei (Tony) Qin, Xiaocheng Tang, Jieping Ye, and Hongtu Zhu. Deep Reinforcement Learning with Knowledge Transfer for Online Rides Order Dispatching. In *ICDM*, pages 617–626. IEEE Computer Society, 2018.

[289] Zhaodong Wang and Matthew E. Taylor. Improving reinforcement learning with confidence-based demonstrations. In *IJCAI*, pages 3027–3033, 2017.

[290] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *International Conference on Learning Representations*, 2017.

[291] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[292] Kevin Waugh, Brian D. Ziebart, and Drew Bagnell. Computational Rationalization: The Inverse Equilibrium Problem. In *Proceedings of the Twenty-Eighth International Conference on Machine Learning, (ICML 2011), Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 1169–1176. Omnipress, 2011.

[293] Piotr Więcek. Discrete-time ergodic mean-field games with average reward on compact spaces. *Dynamic Games and Applications*, 10(1):222–256, 2020.

[294] Piotr Więcek and Eitan Altman. Stationary anonymous sequential games with undiscounted rewards. *Journal of optimization theory and applications*, 166(2):686–710, 2015.

[295] Eric Wiewiora, Garrison W. Cottrell, and Charles Elkan. Principled Methods for Advising Reinforcement Learning Agents. In *Proceedings of the Twentieth International Conference of Machine Learning (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 792–799. AAAI Press, 2003.

[296] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[297] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum Entropy Deep Inverse Reinforcement Learning. *arXiv preprint arXiv:1507.04888*, 2015.

[298] Baicen Xiao, Bhaskar Ramasubramanian, and Radha Poovendran. Shaping Advice in Deep Multi-Agent Reinforcement Learning. *CoRR*, abs/2103.15941, 2021.

[299] Zhe Xu, Zhixin Li, Qingwen Guan, Dingshui Zhang, Qiang Li, Junxiao Nan, Chunyang Liu, Wei Bian, and Jieping Ye. Large-scale order dispatch in on-demand ride-hailing platforms: A learning and planning approach. In *KDD*. ACM, 2018.

[300] Jiachen Yang, Xiaojing Ye, Rakshit Trivedi, Huan Xu, and Hongyuan Zha. Deep Mean Field Games for Learning Optimal Behavior Policy of Large Populations. *CoRR*, abs/1711.03156, 2017.

[301] Jiachen Yang, Xiaojing Ye, Rakshit Trivedi, Huan Xu, and Hongyuan Zha. Learning deep mean field games for modeling large population behavior. In *International Conference on Learning Representations (ICLR)*, 2018.

[302] Tianpei Yang, Weixun Wang, Hongyao Tang, Jianye Hao, Zhaopeng Meng, Hangyu Mao, Dong Li, Wulong Liu, Yingfeng Chen, Yujing Hu, et al. An Efficient Transfer Learning Framework for Multiagent Reinforcement Learning. *NeurIPS, Virtual Event*, 34, 2021.

[303] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. Mean field multi-agent reinforcement learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5567–5576, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

[304] Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

[305] Dayong Ye, Tianqing Zhu, Zishuo Cheng, Wanlei Zhou, and Philip S. Yu. Differential Advising in Multi-Agent Reinforcement Learning. *CoRR*, abs/2011.03640, 2020.

[306] Huibing Yin, Prashant G. Mehta, Sean P. Meyn, and Uday V. Shanbhag. Learning in Mean-Field Games. *IEEE Trans. Autom. Control.*, 59(3):629–644, 2014.

[307] Mohan Yogeswaran and SG Ponnambalam. Reinforcement learning: Exploration–exploitation dilemma in multi-agent foraging task. *Opsearch*, 49(3):223–236, 2012.

[308] Chris Yoon. Deriving policy gradients and implementing reinforce. https://medium.com/@thechrisyoon/deriving-policy-gradients-and-implementing-reinforce-f887949bd63. Accessed: 2022-03-30.

[309] Huizhen Yu and Dimitri P Bertsekas. Q-learning algorithms for optimal stopping based on least squares. In *2007 European Control Conference (ECC)*, pages 2368–2375. IEEE, 2007.

[310] Lantao Yu, Jiaming Song, and Stefano Ermon. Multi-Agent Adversarial Inverse Reinforcement Learning. In *Proceedings of the Thirty-Sixth International Conference on Machine Learning, (ICML 2019), Long Beach, California, USA, 9-15 June 2019*, volume 97 of *Proceedings of Machine Learning Research*, pages 7194–7201. PMLR, 2019.

[311] Yang Yu. Towards sample efficient reinforcement learning. In *IJCAI*, pages 5739–5743, 2018.

[312] Yusen Zhan, Haitham Bou-Ammar, and Matthew E. Taylor. Theoretically-Grounded Policy Advice from Multiple Teachers in Reinforcement Learning Settings with Applications to Negative Transfer. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 2315–2321. IJCAI/AAAI Press, 2016.

[313] Kaiqing Zhang, Zhuoran Yang, and Tamer Bashar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 1, 2019.

[314] Jun Zhao, Guang Qiu, Ziyu Guan, Wei Zhao, and Xiaofei He. Deep reinforcement learning for sponsored search real-time bidding. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1021–1030, 2018.

[315] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. Drn: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*, pages 167–176, 2018.

[316] Jiangchuan Zheng, Siyuan Liu, and Lionel M. Ni. Robust Bayesian Inverse Reinforcement Learning with Sparse Behavior Noise. In *Proceedings of the Twenty-Eighth Conference of Association of Artificial Intelligence (AAAI-2014), Quebec City, Quebec, Canada, July 27 -31, 2014*, pages 2198–2205. AAAI Press, 2014.

[317] Lianmin Zheng, Jiacheng Yang, Han Cai, Ming Zhou, Weinan Zhang, Jun Wang, and Yong Yu. Magent: A many-agent reinforcement learning platform for artificial

collective intelligence. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[318] Hongwei Zhou, Yichen Gong, Luvneesh Mugrai, Ahmed Khalifa, Andy Nealen, and Julian Togelius. A hybrid search agent in Pommerman. In *Proceedings of the 13th International Conference on the Foundations of Digital Games (FDG 2018), Malmo, Sweden, August 07-10, 2018*, pages 46:1–46:4. ACM, 2018.

[319] Ming Zhou, Jun Luo, Julian Villela, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, Aurora Chongxi Huang, Ying Wen, Kimia Hassanzadeh, Daniel Graves, Dong Chen, Zhengbang Zhu, Nhat M. Nguyen, Mohamed Elsayed, Kun Shao, Sanjeevan Ahilan, Baokuan Zhang, Jiannan Wu, Zhengang Fu, Kasra Rezaee, Peyman Yadmellat, Mohsen Rohani, Nicolas Perez Nieves, Yihan Ni, Seyedershad Banijamali, Alexander Imani Cowen-Rivers, Zheng Tian, Daniel Palenicek, Haitham Bou-Ammar, Hongbo Zhang, Wulong Liu, Jianye Hao, and Jun Wang. SMARTS: Scalable Multi-agent Reinforcement Learning Training School for Autonomous Driving. *CoRR*, abs/2010.09776, 2020.

[320] Pengfei Zhu, Xin Li, Pascal Poupart, and Guanghui Miao. On improving deep reinforcement learning for POMDPs. *arXiv preprint arXiv:1704.07978*, 2017.

[321] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei A. Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, and Nicolas Heess. Reinforcement and Imitation Learning for Diverse Visuomotor Skills. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*, 2018.

[322] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the Twenty-Third Conference of Association for the Advancement of Artificial Intelligence (AAAI-2008), Chicago, Illinois, USA, July 13-17, 2008*, pages 1433–1438. AAAI Press, 2008.

[323] Katelyn Zigner, Leila Carvalho, Seth Peterson, Francis Fujioka, Gert-Jan Duine, Charles Jones, Dar Roberts, and Max Moritz. Evaluating the ability of FARSITE to simulate wildfires influenced by extreme, downslope winds in Santa Barbara, California. *Fire*, 3(3):29, 2020.

[324] Martin Zinkevich, Amy Greenwald, and Michael Littman. Cyclic equilibria in markov games. *Advances in neural information processing systems*, 18, 2005.

APPENDICES

Appendix A

Appendix For Chapter 3

This appendix includes the proofs, additional experiments, and comprehensive details regarding the domains used in Chapter 3.

A.1 Proof For The Lemmas In Chapter 3

In this section, we will restate all the lemmas in Chapter 3 with detailed proofs. No new lemmas are given in this section.

Lemma 1. *Let us fix an arbitrary positive constant C, an arbitrary w_0, and a sequence ϵ. Then provided that*

(i) $y_t(w_0, \epsilon)$ converges to some point (independent of t) $\mathcal{D}$

(ii) The sequence ϵ converges to 0 in the limit ($t \to \infty$)

The homogeneous process $x_t(w_0, \epsilon)$ converges to a point $\frac{1}{\hat{\beta}}\mathcal{D}$ w. p. 1, where $\hat{\beta}$, satisfying $0 < \hat{\beta} \leq 1$, is the scaling factor applied.

Proof. We state that

$$y_t(w, \epsilon) = x_t(d_t w, c_t \epsilon_t) \tag{A.1}$$

for some sequences $\{c_t\}$ and $\{d_t\}$, where $c_t = (c_{t0}, c_{t1}, \ldots, c_{ti}, \ldots)$, $\{c_t\}$ and $\{d_t\}$ satisfy $0, d_t, c_{ti} \leq 1$, and $c_{ti} = 1$ if $i \geq t$. Here, the product of the sequences c_t and ϵ_t is component wise: $(c_t \epsilon_t)_i = c_{ti}\epsilon_i$. Note that $y_t(w, \epsilon_t)$ and $x_t(w, \epsilon_t)$ depend only on $\epsilon_0, \cdots, \epsilon_{t-1}$. Thus, it is possible to prove Eq. A.1 by constructing the appropriate sequence c_t and d_t.

Set $c_{0i} = d_i = 1$ for all $i = 0, 1, 2, \ldots$. Then Eq. A.1 holds for $t = 0$. Let us assume that (c_i, d_i) is defined in a way that Eq. A.1 holds for t. Let B_t be the "scaling coefficient" of y_t at step $t+1$ ($B_t = 1$ if there is no scaling, otherwise $0 < B_t < 1$ with $B_t = C/||G_t(y_t, \epsilon_t||)$. Now we have:

$$\begin{aligned} y_t(w, \epsilon_t) &= B_t G_t(y_t(w, \epsilon_t), \epsilon_t) \\ &= G_t(B_t y_t(w, \epsilon_t), B_t \epsilon_t) \\ &= G_t(B_t x_t(d_t w, c_t \epsilon_t), B_t \epsilon_t). \end{aligned} \tag{A.2}$$

We claim that

$$B x_t(w, \epsilon_t) = x_t(Bw, B\epsilon_t) \tag{A.3}$$

holds for all w, ϵ_t and $B > 0$. For $t = 0$, this obviously holds. Assume that it holds for some time t. Then, from Eq. 3.9,

$$\begin{aligned} &B x_{t+1}(w, \epsilon_t) = B G_t(x_t(w, \epsilon_t), \epsilon_t) \\ &= G_t(B x_t(w, \epsilon_t), B\epsilon_t) = x_{t+1}(Bw, B\epsilon_t) \end{aligned} \tag{A.4}$$

Thus,

$$y_{t+1}(w, \epsilon_t) = G_t(x_t(B_t d_t w, B_t c_t \epsilon_t), B_t \epsilon_t), \tag{A.5}$$

and we see that Eq. A.1 holds if we define $c_{t+1,i}$ as $c_{t+1,i} = B_t c_{ti}$ if $0 \le i \le t$, $c_{t+1,i} = 1$ if $i > t$ and $d_{t+1} = B_t d_t$.

Thus, we get that with the sequences

$$\begin{aligned} &c_{t,i} = \Pi_{j=i}^{t-1} B_j, \text{ if } i < t; \\ &c_{t,i} = 1, otherwise; \end{aligned} \tag{A.6}$$

$d_0 = 1$ and

$$d_{t+1} = \Pi_{i=0}^{t} B_i, \tag{A.7}$$

Eq. A.1 is satisfied for all $t \ge 0$.

We know that $y_t(w, \epsilon_t) \to \mathcal{D}$ w. p. 1. Since the process y_t has been constructed by bounding the process with $||y_t|| \le C$, it follows that $\mathcal{D} \le C$. Then, there exists a finite index M such that if $t > M$ then

$$Pr(||y_t(w, \epsilon_t)|| < C) > 1 - \delta \tag{A.8}$$

without applying any more rescaling. Now, let us restrict our attention to those events ω for which $||y_t(w, \epsilon_t(\omega))|| < C$ for all $t > M$ without rescaling. Thus, we have no rescaling for all t, such that $t > M$. Thus, $c_{t,i} = c_{M+1,i}$ for all $t \geq M + 1$ and i, and specifically $c_{ti} = 1$ if $i, t \geq M + 1$. Similarly, if $t > M$ then $d_{t+1}(\omega) = \Pi_{i=0}^{M} B_i(\omega) = d_{M+1}(\omega)$. Let $A_\omega = \{\omega : ||y_t(w, \epsilon)(\omega)|| < C\}$ without rescaling. By Eq. A.1, we have that if $t > M$ then,

$$y_t(w, \epsilon_t(\omega)) = x_t(d_{M+1}(\omega)w, c_{M+1}(\omega)\epsilon_t(\omega)). \tag{A.9}$$

Thus, it follows from our assumption concerning y_t that $x_t(d_{M+1}(\omega)w, c_{M+1}\epsilon_t(\omega))$ converges to $\mathcal{D}$ almost everywhere (a. e.) on A_ω and, consequently, by Eq. A.3, we get that the expression $x_t(w, c_{M+1}\epsilon_t(\omega)/d_{M+1}(\omega))$ converges to $\mathcal{D}' = \frac{1}{d_{M+1}}\mathcal{D}$ a. e. on A_ω. Since $c_{M+1} = 1$ in the limit and we are analyzing in the space of A_ω, $x_t(w, \epsilon_t(\omega)/d_{M+1}(\omega))$ converges to $\mathcal{D}'$ too. Now, since ϵ converges to 0 in the limit, and we are analyzing values in the space of A_ω, we can write $x_t(w, \epsilon_t(\omega)) \approx x_t(w, \epsilon_t(\omega)/d_{M+1}(\omega)))$ which also converges to $\mathcal{D}'$. All these hold with probability at least $1 - \delta$, since, by Eq. A.8 $Pr(A_\omega > 1 - \delta)$. Since δ was arbitrary, the lemma follows.

□

Lemma 2. *Let X and Y be normed vector spaces, $U_t : X \times Y \rightarrow X (t = 0, 1, 2, \ldots)$ be a sequence of mappings, and $\theta_t \in Y$ be an arbitrary sequence. Let $\theta_\infty \in Y$ and $x_\infty \in X$. Consider the sequences $x_{t+1} = U_t(x_t, \theta_\infty)$, and $y_{t+1} = U_t(y - t, \theta_t)$ and suppose that x_t and θ_t converge to x_∞ and θ_∞ respectively, in the norm of the appropriate spaces.*

Let L_k^θ be the uniform Lipschitz index of $U_k(x, \theta)$ with respect to θ at θ_∞ and, similarly, let $L_t^{\mathscr{X}}$ and L_t^θ satisfy the relations $L_t^\theta \leq C(1 - L_t^{\mathscr{X}})$, and $\Pi_{m=t}^{\infty} L_m^{\mathscr{X}} = 0$ where $C > 0$ is some constant and $t = 0, 1, 2, \ldots$, then $\lim_{t \rightarrow \infty} ||y_t - x_\infty|| = 0$.

Proof. See Theorem 15 in Szepesvari and Littman [262] for detailed proof. □

Lemma 3. *Let $\mathcal{Z}$ be an arbitrary set and consider the process*

$$x_{t+1}(z) = G_t(z)x_t(z) + F_t(z)(C + k_t(z)C)$$

where $x_1, F_t, G_t \geq 0$ are random processes, $||x_1|| < C < \infty$ w. p. 1 for some $C > 0$, and z is an element in $\mathcal{Z}$. Assume that for all k, $\lim_{n \rightarrow \infty} \Pi_{t=k}^{n} G_t(z) = 0$ uniformly in z w. p. 1 and $F_t(z) = \gamma(1 - G_t(z))$, for some $0 \leq \gamma < 1$, and $\forall z \in \mathcal{Z}$, w. p. 1. Also, $k_t(z)$ converges to $K(z)$ in the limit. Then, $x_t(z)$ converges to a point $D(z) = \gamma(C + K(z)C)$ w. p. 1.

Proof. Consider the process that is obtained from substituting the value of $F_t(z)$ in terms of $G_t(z)$ in Eq. 3.12,

$$x_{t+1}(z) = G_t(z)x_t(z) + \gamma(1 - G_t(z))(C + k_t(z)C). \tag{A.10}$$

Now, subtracting $\gamma(C + k_t(z)C)$ on both sides of the equation we get

$$x_{t+1}(z) - \gamma(C + k_t(z)C) = G_t(z)(x_t(z) - \gamma(C + k_t(z)C)). \tag{A.11}$$

The above equation converges to 0 in the limit as $\lim_{n\to\infty} \Pi_{t=k}^{n} G_t(z) = 0$. Thus, we have that $x_{t+1}(z) - \gamma(C + k_t(z)C)$ converges to 0 in the limit. Hence, x_t converges to $\gamma(C + K(z)C)$, since $k_t(z)$ converges to $K(z)$ in the limit.

□

Lemma 4. *Consider an equation of the form*

$$x_{t+1}(z) = G_t(z)x_t(z) + F_t(z)(||x_t|| + \epsilon_t + k_t(z)||x_t||)$$

where the sequence ϵ_t converges to zero w. p. 1. Assume that for all k, $\lim_{n\to\infty} \Pi_{t=k}^{n} G_t(z) = 0$ uniformly in z w. p. 1 and $F_t(z) = \gamma(1 - G_t(z))$, for some $0 \leq \gamma < 1$, and $\forall z \in \mathcal{Z}$, w. p. 1. Assume further that $k_t(z)$ is finite, and it converges to $K(z)$ in the limit ($t \to \infty$). Then $x_t(z)$ converges to a point represented by $S'(z) = \frac{1}{\hat{\beta}}(\gamma C_1 + K(z)C_1)$, where C_1 is a small positive constant, w. p. 1 uniformly over $\mathcal{Z}$. Here $\hat{\beta}$ is a scaling factor satisfying $0 < \hat{\beta} \leq 1$.

Proof. Let us consider a process y_t that is obtained from keeping the original process $||x_t||$ bounded by a constant C_1. This is an arbitrary bound, with C_1 specified to be a small positive constant. Since, $||x_t||$ is guaranteed to be positive, we can find such a positive C_1. Now we get,

$$y_{t+1}(z) = G_t(z)y_t(z) + \gamma(1 - G_t(z))(C_1 + \epsilon_t + k_t(z)C_1). \tag{A.12}$$

By Lemma 2, y_t converges to $\gamma C_1 + K(z)C_1$ as the following bindings show $X, Y := \mathbf{R}, \theta_t := \epsilon_t, U_t(x, \theta) := G_t(z)x + \gamma(1 - G_t(z))(C_1 + k_t(z)C_1 + \theta)$, where $z \in Z$ is arbitrary. Then, $L_t^X = G_t(z)$ and $L_t^\theta = \gamma(1 - G_t(z))$ satisfying the conditions of Theorem 2. We know that x in the expression $U_t(x, \theta)$ converges to $\gamma C_1 + K(z)C_1$ as proved in Lemma 3.

In Lemma 1, we proved that the original process will converge to a point $\frac{1}{\hat{\beta}}\mathcal{D}$, if the bounded process converges to $\mathcal{D}$. Here $\hat{\beta}$ is the scaling factor applied, satisfying the relation $0 < \hat{\beta} \leq 1$. Using this result, now we get that the process represented by the Eq. 3.13 converges to a point $S'(z) = \frac{1}{\hat{\beta}}(\gamma C_1 + K(z)C_1)$ which is a constant for a given z. This gives an expression for the point S in Theorem 1.

□

Lemma 5. *Let $\mathcal{F}_t$ be an increasing sequence of σ-fields, let $0 \leq \alpha_t$ and w_t be random variables such that α_t and w_{t-1} are $\mathcal{F}_t$ measurable. Assume that the following hold w. p. 1: $E[w_t|\mathcal{F}_t, \alpha_t \neq 0] = A$, $E[w_t^2|\mathcal{F}_t] < B < \infty$, $\sum_{t=1}^{\infty} \alpha_t = \infty$ and $\sum_{t=1}^{\infty} \alpha_t^2 < C < \infty$ for some $B, C > 0$. Then the process*

$$Q_{t+1} = (1 - \alpha_t)Q_t + \alpha_t w_t$$

converges to A w. p. 1.

Proof. Refer to Lemma 4 in Szepesvari and Littman [262] for detailed proof. □

Lemma 6. *Assume that α_t satisfies Assumption 2 and the mapping $P_t : \mathcal{Q} \to \mathcal{Q}$ satisfies the condition that, there exists a scalar γ satisfying $0 \leq \gamma < 1$, a sequence $\lambda_t \geq 0$ converging to zero w. p. 1, and a finite sequence $k_t(s)$ such that $||P_tQ - P_tQ_*|| = \beta||Q - Q_*|| + \lambda_t + \beta k_t(s)||Q - Q_*||$ for all Q, and all $s \in \mathcal{S}$. Assume further that, $k_t(s)$ converges to a finite point $K(s)$ in the limit. Additionally, $Q_*(s, \boldsymbol{a}) = E[P_tQ_*(s, \boldsymbol{a})]$, then the iteration defined by*

$$Q_{t+1}(s, \boldsymbol{a}) = (1 - \alpha_t)Q_t(s, \boldsymbol{a}) + \alpha_t[P_tQ_t(s, \boldsymbol{a})]$$

converges to $(Q_* - S)$ w. p. 1, where S is as given in Theorem 1.

Proof. This lemma directly follows from Corollary 1 and Lemma 5. □

Lemma 7. *For a n-player stochastic game, $E[P_tQ_*] = Q_*$ where $Q_* = (Q_*^1, \ldots, Q_*^n)$.*

Proof. Refer to Lemma 11 in Hu and Wellman [104] for proof. □

Lemma 8. *A random iterative process*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x)$$

where $x \in X$, $t = 0, 1, \ldots, \infty$, converges to zero with probability one (w. p. 1) if the following properties hold:

1. The set of possible states X is finite.

2. $0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t^2(x) < \infty$ w. p. 1, where the probability is over the learning rates α_t.

3. $||\mathbb{E}\{F_t(x)|\mathscr{P}_t\}||_W \leq \mathscr{K}||\Delta_t||_W + c_t$, where $\mathscr{K} \in [0, 1)$ and c_t converges to zero w. p. 1.

4. $\textbf{\textit{var}}\{F_t(x)|\mathscr{P}_t\} \leq K(1 + ||\Delta_t||_W)^2$, where K is some constant.

Here $\mathscr{P}_t$ is an increasing sequence of σ-fields that includes the past of the process. In particular, we assume that $\alpha_t, \Delta_t, F_{t-1} \in \mathscr{P}_t$. The notation $|| \cdot ||_W$ refers to some (fixed) weighted maximum norm.

Proof. Refer to Theorem 1 in Jaakkola et al. [111] for proof.

□

Lemma 9. *Under Assumption 5, the Nash operator as defined in Eq. 3.26 forms a contraction mapping with the fixed point being the Nash Q-value of the game.*

Proof. The detailed proof is given in Theorem 17 of Hu and Wellman [104]. □

A.2 Additional Definitions For Theorem 1

We restate some definitions from [262], needed for us in Theorem 1, to stay self-contained.

Let us consider an arbitrary operator, $T : \mathcal{B} \to \mathcal{B}$, where $\mathcal{B}$ is a normed vector space with norm $||.||$. Let $\mathcal{T} = (T_0, T_1, \cdots, T_t, \cdots)$ be a sequence of random operators, T_t mapping $\mathcal{B} \times \mathcal{B}$ to $\mathcal{B}$.

Definition 14. *Let $F \subseteq \mathcal{B}$ be a subset of $\mathcal{B}$ and let $\mathcal{F}_0 : F \to 2^{\mathcal{B}}$ be a mapping that associates subsets of $\mathcal{B}$ with the elements of F. If, for all $f \in F$ and all $m_0 \in \mathcal{F}_0(f)$, the sequence generated by the recursion $m_{t+1} = T_t(m_t, f)$ converges to Tf in the norm of $\mathcal{B}$ with probability 1, then we say that $\mathcal{T}$ approximates T for initial values from $\mathcal{F}_0(f)$ and on the set $F \subseteq \mathcal{B}$. Further, we say that $\mathcal{T}$ approximates T on the singleton set f and the initial value mapping $\mathcal{F}_0 : F \to B$ defined by $\mathcal{F}_0(f) = F_0$.*

Definition 15. *The subset $F \subseteq \mathcal{B}$ is invariant under $T : \mathcal{B} \times \mathcal{B} \to \mathcal{B}$ if, for all $f, g \in F, T(f, g) \in F$. If $\mathcal{T}$ is an operator sequence as above, then F is said to be invariant under $\mathcal{T}$ if for all $i \geq 0$, F is invariant under T_i.*

A.3 ADMIRAL-AE Using An Adaptive Advisor

In this sub-section, we aim to provide an illustration of the behaviour of ADMIRAL-AE in the presence of an adaptive advisor. This advisor would actively change and adapt its strategies according to the changing opponent. The objective is to show that the ADMIRAL-AE algorithm will be able to capture the strength of the adaptive advisor, and hence there is merit in using a principled approach to evaluate an advisor. Also, we would like to discuss the advantages of keeping this evaluation method separate from another approach that aims to learn from the advisor. In this section, we clarify that we are only considering the 'pre-learning' phase introduced in Chapter 3, since our objective is to analyze the performance of ADMIRAL-AE with two different advisors. Results in this section use the average and standard deviation of 30 runs.

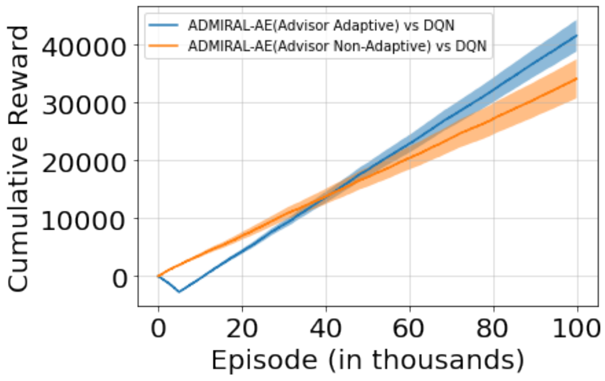


Figure A.1: Two instances of ADMIRAL-AE along with an adaptive advisor and a non-adaptive advisor on Pommerman Domain OneVsOne. The plot shows the performance of ADMIRAL-AE using the Advisor Non-Adaptive against the common opponent of DQN (orange line) and the performance of ADMIRAL-AE using the Advisor Adaptive against the common opponent of DQN (blue line). This plot shows that the ADMIRAL-AE using an adaptive advisor starts off week, but eventually surpasses the performance of the non-adaptive advisor. Thus, a principled method like ADMIRAL-AE can evaluate an adaptive advisor appropriately, while other non-principled approaches may have a high percentage of failure.

We will continue to use the two-agent version (Domain OneVsOne) of Pommerman for this experiment. We consider two different advisors, namely, Advisor Non-Adaptive and Advisor Adaptive. The Advisor Non-Adaptive does not actively track opponent strategies or adapt to them. This advisor plays a balanced strategy — choosing to be risk-seeking (actively laying bombs to kill the opponent) while the opponent is in proximity and choosing to be risk-averse (escaping from the enemy) otherwise. This advisor is reactive and only responds to the present position of the opponent, and does not attempt to model the opponent's nature actively. It has been observed that this is a relatively good strategy in Pommerman, particularly in the early stages of training [161]. At this stage, an agent playing a relatively conservative strategy could wait for the opponent to make a mistake and kill itself. However, this strategy is not very strong and could lose out once the opponent is well-trained. A well-trained opponent is less likely to make the mistake of killing itself, and there is the added possibility of the non-adaptive strategy becoming predictable, which could be figured out by the opponent. Hence, the Advisor Non-Adaptive will find it hard to win games in the middle and later stages of training. On the other hand, the Advisor Adaptive uses an adaptive strategy that plays a risk-averse strategy when the enemy is risk-seeking, and a risk-seeking strategy when the enemy is risk-averse. This is a very strong strategy for winning in Pommerman [318] but requires active modelling of the opponent. The Advisor Adaptive tracks the percentage of bombs played by the opponent to determine the nature of the opponent. In the initial few episodes (about 5000) the Advisor Adaptive's behaviour is close to random since it still does not have enough information to learn the nature of the opponent.

We implement a separate instance of the ADMIRAL-AE algorithm with both the advisors and plot the performance against a common baseline agent using DQN for learning. We run all the training for 100,000 episodes and plot the performances of both algorithms. The results are captured in Figure A.1. The results show that ADMIRAL-AE using the Advisor Adaptive loses out in the beginning while it is still figuring out the nature of the opponent. However, it soon shows a much stronger performance that surpasses the performance of the ADMIRAL-AE using the Advisor Non-Adaptive. ADMIRAL-AE using the Advisor Non-Adaptive, while showing good overall performance, does not quite reach the levels of the performance of ADMIRAL-AE using the Advisor Adaptive, due to the non-adaptive strategy possibly becoming predictable and prone to exploitation by the opponent, after the opponent has trained for a sufficient number of episodes. This performance shows that an agent during learning should listen more to the Advisor Adaptive as compared to the Advisor Non-Adaptive for the best outcome. If the advisors were directly used for learning without being evaluated separately, then a learning agent would be prone to discarding Advisor Adaptive quickly due to its initial weak performance, while the agent would listen more to the Advisor Non-Adaptive. Yet we have seen that the opposite behaviour would actually be better using an implementation of ADMIRAL-AE with each of these advisors. This demonstrates that the evaluation would be prone to inaccuracy and inconsistency if it is combined with learning a policy. The analysis in this sub-section shows the advantage of using a principled evaluation algorithm (ADMIRAL-AE) for evaluating an advisor, especially when they have adaptive characteristics.

A.4 Experimental Details

In this section, we provide the experimental details for all the experiments. We have given detailed information about the advisors and reward functions for all the experiments. Hyperparameters for all algorithms are also provided.

A.4.1 Grid Maze Domain

The reward function is defined in such a way that the agents get a +1 if any one of two agents reaches the goal, and they get a -1 if any one of the two gets to the pitfall. Both the agents get a +2 if both reach the goal at the same instant, and both the agents get a -2 if they reach a pitfall at the same instant. If one of the two gets to the goal and the other gets to the pitfall, they both still get a +1. In this environment, each agent obtains a local state (observation) which corresponds to the coordinates of the grid cell the agent is currently located. We use the joint observations of the two agents as the state in the environment for determining the Q-values in the experiments showing convergence in Section 3.4.1. This state is available to both agents. The other experiments simply use the observation of the concerned agent in the Q-updates, in case of both ADMIRAL-DM and ADMIRAL-AE, to make the challenge harder.

The actions that the agents can take in this game are one of moving up, down, left or right. If the wall obstructs an action, then the agent will remain at the same spot.

All four advisors used are rule-based agents, where Advisor 1 follows high-quality rules at each grid cell, which enables the agent to move to the goal state and avoid the pitfalls. Advisor 2 on the other hand can only suggest the correct actions for the agents to reach the goal and escape the pitfall if the agent is right next to the goal or pitfall (within one step). It is not capable of giving the right action in the other parts of the grid. Thus, Advisor 2 cannot teach perfect coordination to obtain the large positive reward (+2). Advisor 3 can only suggest actions that make the agents move closer to the goal state, but cannot give accurate actions that avoid pitfalls. Advisor 4 is a random advisor which only suggests random actions.

The ADMIRAL-AE implementation for this experiment chooses to do the advisor actions with a probability of 50% ($\eta' = 0.5$). The action that maximizes the Q-value (greedy action) is chosen with probability 45% and a random action with probability 5% ($\eta = 0.05$) to satisfy Assumption 1. We use the previous actions of the other agents while determining actions at the current time step t as done in the other algorithms. The ADMIRAL-DM algorithm simply follows the scheme in Algorithm 1 where the advisor suggestions are performed with decreasing probabilities and the algorithm becomes completely greedy (dependent only on the agent's own policy) after a finite number of episodes.

A.4.2 Pommerman Domain

All the advisors are based on the rule-based agent (called the simple agent) already provided in the Pommerman framework. It is well documented that the simple agent is hard to beat even for complex RL algorithms in the Pommerman environment [203]. We use four advisors for the ADMIRAL-AE experiments in Section 3.4.2 and 3.4.4, of which the first advisor alone is used in the experiments that study the performance of ADMIRAL-DM and ADMIRAL-DM(AC) in Section 3.4.3. The first advisor (Advisor 1) is the best of all the four advisors and has rules for all the different aspects of the game (escaping from enemies, collecting bombs, laying bombs, etc.). The second advisor (Advisor 2) has rules for moving away from danger and collecting powerups like life and bombs. This is a relatively conservative advisor which relies on staying safe and hoping for the opponent to make a mistake. Actually, this is a very effective strategy in the Pommerman game and therefore Advisor 2 is also a useful one. Advisor 3 only has rules for collecting powerups, but cannot teach any of the other strategies needed to win the Pommerman game. When there is no possibility of teaching any useful strategy based on the situation of the game (no nearly enemies, powerups, etc.), the first three advisors provide pseudo-random strategies that encourage exploration of states which are relatively less visited. The fourth advisor (Advisor 4) is the weakest advisor of the four. It suggests random actions to the agent and does not contribute to the learning of the agent (rather, it harms learning).

Each new Pommerman game gives a randomized game map and there is a maximum of 800 steps for each game.

A.4.3 Pursuit Domain

The Pursuit domain was initially defined in SISL [89]. We use the canonical domain implemented in the petting zoo environment [269]. All the environmental parameters including the rewards are left as the same as the defaults in [89]. For the training phase, the game has 30 evader agents and 8 pursuer agents, where the evader agents move randomly, and the pursuer agents are controlled by learning algorithms. The game is cooperative, with the pursuers receiving a reward of 5 for fully surrounding an evader (the evader is removed from the environment). The pursuers receive a reward of 0.01 for each time they touch an evader. The actions space is discrete with 5 actions, each action corresponding to moving to a neighbouring grid (including stay). Each episode is a full game with a maximum of 500 steps. Each agent in this environment is able to observe a grid of 7×7 centered around itself (the whole grid is 16×16). Since this environment is fully cooperative with homogeneous learning agents, all training (for each algorithm) is completely centralized (all agents train the same network).

A.4.4 Waterworld Domain

Waterworld is also a SISL environment with a set of 5 pursuers tasked with collecting food and avoiding poison. We use the corresponding petting zoo environment [269]. The environmental parameters are the same as that in [89]. This is also a cooperative environment where multiple pursuers need to work together to capture food. The environment is a two-dimensional continuous space, where the action space is a continuous two-dimensional value representing the horizontal and vertical thrust that makes the agents move in particular directions with the desired speed. The local state (observation) of each agent consists of multiple sensor features and two other elements that indicate the collision of the agent with a food or poison respectively. In this environment, at least two agents need to attack a food together to capture it. There are a total of 5 food particles (not destroyed upon capture) and 10 poison particles in the environment. The agents get a reward of 10.0 for capturing food and a 0.01 for encountering food. Further, the agents have a thrust penalty of -0.5, and a penalty of -1.0 for encountering poison. The poison particles and food particles move in the environment with a speed of 0.01. Since this environment is also fully cooperative with homogenous learning agents, all training is completely centralized for this domain too.

A.4.5 Hyperparameters And Implementation Details

The hyperparameters for the baselines were chosen to be the same as those recommended by the respective papers. Some minor modifications were made due to performance and computational efficiency considerations.

Regarding the hyperparameters of DQfD, we set 1×10^6 as the demo buffer size and perform 50,000 mini-batch updates for pretraining. The replay buffer size is twice the size of the demo buffer. The N-step return weight is 1.0, the supervised loss weight is 1.0 and the L2 regularization weight is 10^{-5}. The epsilon greedy exploration is 0.9. The discount factor is 0.99 and the learning rate is 0.002. The pretraining for DQfD comes from a data buffer related to a series of games where two rule-based agents (advisors) compete against each other. All other values are similar to that used in [100].

The CHAT [289] implementation uses a neural network for confidence measurement (termed NNHAT in [289]). The learning rate is 0.01, we use a discount factor of 0.9 and a fixed exploration constant (ϵ-greedy) of 0.9. We use the extra action variant of HAT [268] in the CHAT implementation, as this gave the best performance in most of our comparative experiments. A neural network is used as the function approximator, as described in [167]. The target net is replaced every 10 learning iterations. The confidence threshold is set as 0.6 and the default action as "action-0". The mini-batch size is 32 and learning rate is $\alpha = 0.01$. The CHAT algorithm can directly use advisors in an online fashion similar to ADMIRAL-DM. We omit the rule summarization step of CHAT, and directly use the advisor, to make the performance as good as possible. The replay buffer size is 2×10^6.

The DQN and ADMIRAL-DM hyperparameters are the same as those mentioned for CHAT (as relevant). These algorithms also perform ϵ-greedy exploration with a constant value of 0.9 for ϵ. The advisor influence parameter (ϵ'_t in Algorithm 1) for ADMIRAL-DM and ADMIRAL-DM(AC) starts at a high value of 0.8 at the beginning of the training and linearly decays to 0.01 during training. The face-off and execution phases have this parameter to be 0 (no advisor influence). All other hyperparameters for DQN, ADMIRAL-DM and CHAT are similar to that used in Mnih et al. [167]. The ADMIRAL-DM(AC) has the critic learning rate set at 10^{-3} and the actor learning rate to be 10^{-5}. The discount factor for ADMIRAL-DM(AC) is 0.9.

The ADMIRAL-AE algorithm in Section 3.4.2 chooses to do the advisor action with a probability of 50%. The action that maximizes the Q-value (the greedy action) is chosen with probability 45% and a random action with probability 5% to satisfy Assumption 1. This is the same as that in the tabular domain (Section 3.4.1). The hyperparameters of ADMIRAL-AE are the same as ADMIRAL-DM.

For the function approximation experiments, both ADMIRAL-DM and ADMIRAL-DM(AC) simply use the observed current actions of other agents for action selection instead of maintaining copies of policies of the other agents as specified in Algorithm 3 and Algorithm 5, since the opponents could possibly be using different algorithms (all agents are not always using the same algorithmic steps). This is also computationally more efficient. The current actions of all other agents are either directly observed or provided by the game engine to each agent in all our experiments, to perform a joint action update.

The DDPG uses the learning rate of the actor as 0.001 and the critic as 0.002. The discount factor is 0.9. We use the soft replacement strategy with a learning rate of 0.01. The batch size is 32. The PPO implementation also uses the same batch size and actor and critic learning rates. All other values are similar to those used in Schulman et al. [216] (PPO) and Lillicrap et al. [147] (DDPG). For PPO, we used a single-thread implementation, which we found to be as good as the multi-threaded implementation for our experiments, and more computationally efficient. This could be because the data correlations are already being broken by the multi-agent (non-stationary) nature of the domains.

For the Deep Sarsa implementation used in Chapter 3, we follow almost the same steps as done in Algorithm 4, except that we do not have any advisors and hence the algorithm does not have the term ϵ'_t in the implementation. As a consequence, the greedy action is selected with probability $(1 - \epsilon_t)$. Also, we are using an independent implementation, where the actions of the other agents are not considered during action selection by the algorithms playing Deep Sarsa. The action is chosen only based on the current state. Deep Sarsa uses the same hyperparameters as ADMIRAL-DM. The values are either the same or closely match those considered in previous research [167].

We use a set of 30 random seeds (1–30) for all training experiments and a new set of 30 random seeds (31–60) for all execution experiments.

A.4.6 Wall Clock Times

The experiments on the Grid Maze domain can just be run on the CPU and takes less than 20 minutes to complete.

The training for all the experiments on the Pommerman domain in Section 3.4.2 and Section 3.4.3 was run on a 2 GPU virtual machine with 16 GB GPU memory per GPU. The experiments take an average of 18 hours wall clock time to complete. We use Nvidia Volta-100 (V100) GPUs for all these experiments. The CPUs use Skylake as the processor microarchitecture.

The experiments on Pursuit and Waterworld domains in Section 3.4.3 were run on a virtual machine having the same configuration containing 2 GPUs. These experiments take an average of 12 hours wall clock time to complete. The experiments on Pommeran in Section 3.4.4 took an average of 15 hours wall clock time to complete.

Appendix B

Appendix For Chapter 4

B.1 Proof Of Lemmas In Chapter 4

Lemma 10. *A random iterative process*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)F_t(x) \tag{B.1}$$

where $x \in X$, $t = 0, 1, \dots, \infty$, converges to zero with probability one (w. p. 1) if the following properties hold:

1. *The set of possible states X is finite.*

2. *$0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, $\sum_t \alpha_t^2(x) < \infty$ w. p. 1, where the probability is over the learning rates α_t.*

3. *$||\,\mathbb{E}\{F_t(x)|\mathscr{P}_t\}||_W \leq \mathscr{K}\,||\Delta_t||_W + c_t$, where $\mathscr{K} \in [0, 1)$ and c_t converges to zero w. p. 1.*

4. *$\boldsymbol{var}\{F_t(x)|\mathscr{P}_t\} \leq K(1 + ||\Delta_t||_W)^2$, where K is some constant.*

*Here $\mathscr{P}_t$ is an increasing sequence of σ-fields that includes the past of the process. In particular, we assume that $\alpha_t, \Delta_t, F_{t-1} \in \mathscr{P}_t$. The notation $||\cdot||_W$ refers to some (fixed) weighted maximum norm and the notation **var** refers to the variance.*

Proof. Refer to Theorem 1 in Jaakola et al. [111] for proof.

□

Lemma 11. *Under Assumption 5, the Nash operator as defined in Eq. 4.5 forms a contraction mapping with the fixed point being the Nash Q-value of the game.*

Proof. See Theorem 17 of Hu and Wellman [104]. □

Lemma 12. *For every k, there exists a time τ_k such that, for any $t \geq \tau_k$ we have $||X_t^j|| \leq D_k^j$.*

Proof. See Theorem 8 in Even-Dar and Mansour [64]. □

Lemma 13. *For $m \geq \frac{1}{\beta}\ln(Q_{\max}^j/\epsilon)$ we have $D_m^j \leq \epsilon$.*

Proof. Since we have that $D_1^j = Q_{\max}^j$ and $D_i^j = (1-\beta)D_{i-1}^j$, we will need a m that satisfies $D_m^j = Q_{\max}^j(1-\beta)^m \leq \epsilon$. By taking a logarithm on both sides, we get

$$\begin{aligned} &\ln\Big((Q_{\max}^j)(1-\beta)^m\Big) \leq \ln(\epsilon) \\ &\ln(Q_{\max}^j) + m\ln(1-\beta) \leq \ln(\epsilon) \\ &\implies \ln(Q_{\max}^j) - \ln(\epsilon) \leq -m\ln(1-\beta) \\ &\implies \ln(Q_{\max}^j/\epsilon) \leq m\textstyle\sum_{k=0}^{\infty}\beta^k/k \\ &\implies 1/\beta\ln(Q_{\max}^j/\epsilon) \leq m. \end{aligned} \tag{B.2}$$

In the last step we omit the higher powers since β is a small fraction. This proves the result. □

Lemma 14. *For every state s and joint action $\boldsymbol{a}$ and time τ_k, we have*

$$\begin{aligned} &-Y_{t;\tau_k}^j(s,\boldsymbol{a}) + W_{t;\tau_k}^j(s,\boldsymbol{a}) \\ &\leq Q_*^j(s,\boldsymbol{a}) - Q_t^j(s,\boldsymbol{a}) \\ &\leq Y_{t;\tau_k}^j(s,\boldsymbol{a}) + W_{t;\tau_k}^j(s,\boldsymbol{a}) \end{aligned} \tag{B.3}$$

Proof. See Lemma 4.4 in Bertsekas and Tsitsiklis [22]. □

Lemma 15. *The sequence $Y_{t;\tau_k}^j$ is a monotonically decreasing sequence.*

Proof. From Eq. 4.18 we can write (subtract γD_k^j from both sides),

$$\begin{aligned}
&(Y_{t+1;\tau_k}^j(s,\boldsymbol{a}) - \gamma D_k^j) \\
&= (1-\alpha_t^\omega(i))Y_{t;\tau_k}^j(s,\boldsymbol{a}) + (1-\alpha_t^\omega(s,\boldsymbol{a}))\gamma D_k^j \\
&= (1-\alpha_t^\omega(i))(Y_{t;\tau_k}^j(s,\boldsymbol{a}) + \gamma D_k^j).
\end{aligned} \tag{B.4}$$

Now, the convergence of $||Y_{t+1;\tau_k}^j(s,\boldsymbol{a}) - \gamma D_k^j||$ follows since $\lim_{n\to\infty}\Pi_{t=k}^n(1-\alpha_t^\omega(s,\boldsymbol{a})) = 0$. This shows that the sequence $(Y_{t+1;\tau_k}^j(s,\boldsymbol{a}) - \gamma D_k^j)$ monotonically decreases to 0 and hence the sequence $Y_{t;\tau_k}^j$ monotonically decreases to γD_k. This proves our result.

□

Lemma 16. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate and assume that for any $t \geq \tau_k$ we have $Y_{t;\tau_k}^j(s,\boldsymbol{a}) \leq D_k$. Then for any $t \geq \tau_k + L\tau_k^\omega = \tau_{k+1}$ we have $Y_{t;\tau_k}^j(s,\boldsymbol{a}) \leq D^j(\gamma + \frac{2}{e}\beta)$.*

Proof. For each state-joint action pair $s, \boldsymbol{a}$ we are assured that $n(s,a,\tau_k,\tau_{k+1}) \geq \tau_k^\omega$, since the covering time is L and the underlying policy has made $L\tau_k^\omega$ steps (since we have the relation $\tau_{k+1} = \tau_k + L\tau_k^\omega$).

Let $Y_{\tau_k,\tau_k}^j(s,\boldsymbol{a}) = \gamma D_k^j + \rho_{\tau_k}^j$, where $\rho_{\tau_k} = (1-\gamma)D_k^j$. Now we have the following expression,

$$Y_{t+1,\tau_k}^j(s,\boldsymbol{a}) = (1-\alpha_t^\omega)Y_{t;\tau_k}^j(s,\boldsymbol{a}) + \alpha_t^\omega\gamma D_k^j = \gamma D_k^j + (1-\alpha_t^\omega)\rho_t^j \tag{B.5}$$

where $\rho_{t+1}^j = \rho_t^j(1-\alpha_t^\omega)$. We aim to show that after time $\tau_{k+1} = \tau_k + \tau_k^\omega$, for any $t \geq \tau_k^\omega$ for any $t \geq \tau_{k+1}$ we have $\rho_t^j \leq \frac{2}{e}\beta D_k^j$. By definition, we can rewrite ρ_t^j as

$$\begin{aligned}
\rho_t^j &= (1-\gamma)D_k^j\Pi_{l=1}^{t-\tau_k}(1-\alpha_{l+\tau_k}^\omega) \\
&= 2\beta D_k^j\Pi_{l=1}^{t-\tau_k}(1-\alpha_{l+\tau_k}^\omega) \\
&= 2\beta D_k^j\Pi_{l=1}^{t-\tau_k}(1-\tfrac{1}{|n(s,\boldsymbol{a},\tau_{k+1},l)|^\omega})
\end{aligned} \tag{B.6}$$

the last identity follows from the definition of α_t^ω.

Since the τ_k^ω's are monotonically decreasing,

$$\rho_t^j \leq 2\beta D_k^j (1 - \frac{1}{\tau_k^\omega})^{t-\tau_k}. \tag{B.7}$$

For $t \geq \tau_k + \tau_k^\omega$ we have,

$$\rho_t^j \leq 2\beta D_k^j \Big(1 - \frac{1}{\tau_k^\omega}\Big)^{\tau_k^\omega} \leq \frac{2}{e}\beta D_k^j. \tag{B.8}$$

The last step is obtained from the fact that $\lim_{x \longrightarrow \infty}(1 - (1/x))^x = 1/e$.

Hence, $Y_{t;\tau_k}(s, \boldsymbol{a}) \leq (\gamma + \frac{2}{e}\beta)D_k$.

□

Lemma 17. *Let* $\tilde{w}_{i+\tau_k}^{j,t}(s, \boldsymbol{a}) = \eta_{i+\tau_k}^{k,t,j}(s, \boldsymbol{a}) w_{i+\tau_k}^j(s, \boldsymbol{a})$*, then for any* $t \in [\tau_{k+1}, \tau_{k+2}]$ *the random variable* $\tilde{w}_{i+\tau_k}^j(s, \boldsymbol{a})$ *has zero mean and bounded by* $(L/\tau_k)^\omega Q_{\max}^j$

Proof. Since $\eta_{i+\tau_k}^{k,t,j}((s, \boldsymbol{a})) = \alpha_{i+\tau_k}^\omega(s, \boldsymbol{a})\Pi_{l=\tau_k+i+1}^t(1 - \alpha_l^\omega(s, \boldsymbol{a}))$, we can divide $\eta_{i+\tau_k}^{k,t,j}$ into two parts, the first $\alpha_{i+\tau_k}^\omega$ and the second $\mu = \Pi_{l=\tau_k+i+1}^t(1 - \alpha_l^\omega)$. Since, μ is bounded from above by 1, we have,

$$\begin{aligned} &\eta_{i+\tau_k}^{k,t,j}(s, \boldsymbol{a}) \leq \alpha_{i+\tau_k}^\omega(s, \boldsymbol{a}) \\ &= \tfrac{1}{|\#(s,\boldsymbol{a},i+\tau_k)|} \overset{*}{\leq} (\tfrac{L}{i+\tau_k})^\omega \leq (\tfrac{L}{\tau_k})^\omega. \end{aligned} \tag{B.9}$$

Here, the $*$ is from the fact that in a time interval of τ, each state-joint action pair is performed at-least τ/L times by the definition of covering time.

Hence, we get the relation that $\eta_{i+\tau_k}^{k,t,j}(s, \boldsymbol{a}) \leq (L/\tau_k)^\omega$.

Now, consider the expectation of $\tilde{w}_{i+\tau_k}(s, \boldsymbol{a})$. By definition, $w_{\tau_k+i}^j(s, \boldsymbol{a})$ has zero mean and is bounded by $Q_{\max}^j$ for any history and state-joint action pair, hence,

$$\begin{aligned} &\mathbb{E}[\tilde{w}_{i+\tau_k}(s, \boldsymbol{a})] \\ &= \mathbb{E}[\eta_{i+\tau_k}^{k,t,j}(s, \boldsymbol{a}) w_{i+\tau_k}(s, \boldsymbol{a})] \\ &= \eta_{i+\tau_k}^{k,t,j}(s, \boldsymbol{a})\, \mathbb{E}[w_{i+\tau_k}(s, \boldsymbol{a})] = 0. \end{aligned} \tag{B.10}$$

Next we can prove that it is bounded as well,

$$
\begin{aligned}
&|\tilde{w}_{i+\tau_k}(s, \boldsymbol{a})| = |\eta_i^{k,t,j}(s, \boldsymbol{a}) w_{i+\tau_k}(s, \boldsymbol{a})| \\
&\leq |\eta_i^{k,t,j}(s, \boldsymbol{a}) Q^j_{\max}| \\
&\leq (L/\tau_k)^{\omega} Q^j_{\max}
\end{aligned}
\tag{B.11}
$$

□

Now, let us define $W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) = \sum_{i=1}^{l} \tilde{w}^t_{i+\tau_k}(s, \boldsymbol{a})$. The objective is to prove that this is a martingale difference sequence having bounded differences.

Lemma 18. *For any $t \in [\tau_{k+1}, \tau_{k+2}]$ and $1 \leq l \leq t$ we have that $W^{l,j}_{t;\tau_k}(s, \boldsymbol{a})$ is a martingale sequence that satisfies*

$$
|W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s, \boldsymbol{a})| \leq (L/\tau_k)^{\omega} Q^j_{\max} \tag{B.12}
$$

Proof. We first note that the term $W^{l,j}(s, \boldsymbol{a})$ is a martingale sequence, since

$$
\mathbb{E}[W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s, \boldsymbol{a}) | F_{\tau_k + l - 1}] = \mathbb{E}[\tilde{w}^{j,t}_{l+\tau_k}(s, \boldsymbol{a}) | F_{\tau_k + l - 1}] = 0 \tag{B.13}
$$

where the variable $F_{\tau_k + l - 1}$ denotes all previous values of W^l.

Also, by Lemma 17 we can show that $\tilde{w}^j_{l+\tau_k}(s, \boldsymbol{a})$ is bounded by $(L/\tau_k)^{\omega} Q^j_{\max}$, thus

$$
|W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s, \boldsymbol{a})| = \tilde{w}^j_{l+\tau_k}(s, \boldsymbol{a}) \leq (L/\tau_k)^{\omega} Q^j_{\max} \tag{B.14}
$$

□

Lemma 19. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate. With probability at least $1 - \frac{\delta}{m}$ we have that, for every state-joint action pair $|W^j_{t;\tau_k}(s, \boldsymbol{a})| \leq (1 - \frac{2}{e}\gamma D_k)$ for any $t \in [\tau_{k+1}, \tau_{k+2}]$, i.e.*

$$
Pr\Big[\forall s, \boldsymbol{a} \forall t \in [\tau_{k+1}, \tau_{k+2}] : |W_{t;\tau_k}(s, \boldsymbol{a})| \leq (1 - \tfrac{2}{e})\beta D_k\Big] \geq 1 - \tfrac{\delta}{m} \tag{B.15}
$$

given that

$$
\tau_k = \Theta\Big(\big(\tfrac{L^{1+3\omega} Q^{2,j}_{\max} \ln(Q^j_{\max} |S| |\Pi_i| |A|_i m / (\delta \beta D_k))}{\beta^2 D_k^2} \big)^{1/\omega} \Big) \tag{B.16}
$$

Proof. For each state-joint action pair comparing $W^{l,j}_{t;\tau_k}(s,\boldsymbol{a})$ and $W^{j}_{t;\tau_k}(s,\boldsymbol{a})$ we note that $W^{j}_{t;\tau_k}(s,\boldsymbol{a}) = W^{t-\tau_k+1,j}_{t;\tau_k}(s,\boldsymbol{a})$.

Let $l = n(s,\boldsymbol{a},\tau_k,t)$, then for any $t \in [\tau_{k+1},\tau_{k+2}]$ we have that

$$\begin{aligned} l &\le \tau_{k+2} - \tau_k = \tau_{k+1} + L\tau^{\omega}_{k+1} - \tau_k \\ &= \tau_k + L\tau^{\omega}_k + L(\tau_k + L\tau^{\omega}_k) - \tau_k \\ &= L\tau^{\omega}_k + L\tau_k + L^2\tau^{\omega}_k \le \Theta(L^{1+\omega}\tau^{\omega}_k). \end{aligned} \tag{B.17}$$

By Lemma 18 we can apply Azuma's inequality to $W^{t-\tau_k+1,j}_{t;\tau_k}(s,\boldsymbol{a})$ with $c_i = (L/\tau_k)^{\omega} Q^{j}_{\max}$. Therefore, we can derive that

$$\begin{aligned} &Pr\Big[|W_{t;\tau_k}(s,\boldsymbol{a})| \ge \tilde{\epsilon} | t \in [\tau_{k+1},\tau_{k+2}]\Big] \\ &\le 2e^{\frac{-\tilde{\epsilon}^2}{2\sum_{i=\tau_k+1}^{t} c_i^2}} \le 2e^{\frac{-\tilde{\epsilon}^2}{2lc_i^2}} \\ &\le 2e^{-c\frac{\tilde{\epsilon}^2\tau_k^{2\omega}}{lL^{2\omega}Q^{2,j}_{\max}}} \\ &\le 2e^{-c\frac{\tilde{\epsilon}^2\tau_k^{\omega}}{L^{1+3\omega}Q^{2,j}_{\max}}} \end{aligned} \tag{B.18}$$

for some constant $c > 0$. We can set $\tilde{\delta}_k = 2e^{\frac{-c\tau_k^{\omega}\tilde{\epsilon}^2}{L^{1+3\omega}Q^{2,j}_{\max}}}$, which holds for $\tau^{\omega}_k = \Theta(\ln(1/\tilde{\delta}_k)L^{1+3\omega}Q^{2,j}_{\max}/\tilde{\epsilon}^2)$.

Using the union bound we get,

$$\begin{aligned} &Pr[\forall s, \forall \boldsymbol{a}, \forall t \in [\tau_{k+1},\tau_{k+2}] : W^{j}_{t,\tau_k}(s,\boldsymbol{a}) \le \tilde{\epsilon}] \\ &\ge 1 - \textstyle\sum_{t=\tau_{k+1}}^{\tau_{k+2}} Pr[\forall s, \forall \boldsymbol{a}, W^{j}_{t;\tau_k}(s,\boldsymbol{a}) \ge \tilde{\epsilon}] \\ &\ge 1 - \textstyle\sum_{t=\tau_{k+1}}^{\tau_{k+2}} Pr[\forall s, \forall \boldsymbol{a}, |W^{j}_{t;\tau_k}(s,\boldsymbol{a})| \ge \tilde{\epsilon}] \\ &\ge 1 - \textstyle\sum_{t=\tau_{k+1}}^{\tau_{k+2}} \tilde{\delta}_k |S|\Pi_i|A|_i \\ &\ge 1 - (\tau_{k+2} - \tau_{k+1})\tilde{\delta}_k |S|\Pi_i|A|_i \end{aligned} \tag{B.19}$$

We would like to set a level of probability of $1-\frac{\delta}{m}$, for each state-joint action pair. From the above equation we get,

$$\begin{aligned}
&1-(\tau_{k+2}-\tau_{k+1})\tilde{\delta}_k|S|\Pi_i|A|_i = 1-\delta/m \\
&\implies (\tau_{k+2}-\tau_{k+1})\tilde{\delta}_k|S|\Pi_i|A|_i = \delta/m. \qquad \text{(B.20)} \\
&\implies \tilde{\delta}_k = \delta/(\tau_{k+2}-\tau_{k+1})m|S|\Pi_i|A|_i.
\end{aligned}$$

Thus taking $\tilde{\delta}_k = \frac{\delta}{m(\tau_{k+2}-\tau_{k+1})|S||A|}$ assures $1-\frac{\delta}{m}$ for each state-joint action pair. As a result we have,

$$\begin{aligned}
\tau_k^\omega &= \Theta\Big(\tfrac{L^{1+3\omega}Q_{\max}^{2,j}\ln(|S|\Pi_i|A|_i m\tau_k^\omega/\delta)}{\tilde{\epsilon}^2}\Big) \\
&= \Theta\Big(\tfrac{L^{1+3\omega}Q_{\max}^{2,j}\ln(|S|\Pi_i|A|_i mQ_{\max}^j/(\delta\tilde{\epsilon}))}{\tilde{\epsilon}^2}\Big)
\end{aligned} \qquad \text{(B.21)}$$

Setting $\tilde{\epsilon} = (1-2/e)\beta D_k$ gives the desired bound.

□

Lemma 20. *Consider the low-Q update given in Eq. 4.10, with a polynomial learning rate. With probability $1-\delta$, for every iteration $k \in [1,m]$ and time $t \in [\tau_{k+1},\tau_{k+2}]$ we have $|W_{t;\tau_k}^j(s,\boldsymbol{a})| \le (1-\frac{2}{e})\beta D_k^j$, i.e.,*

$$Pr\Big[\forall k \in [1,m], \forall t \in [\tau_{k+1},\tau_{k+2}], \forall s,\boldsymbol{a} : |W_{t;\tau_k}^j(s,\boldsymbol{a})| \le (1-\tfrac{2}{e})\beta D_k^j\Big] \ge 1-\delta \qquad \text{(B.22)}$$

given that

$$\tau_0 = \Theta\Big(\big(\frac{L^{1+3\omega}Q_{\max}^{2,j}\ln(Q_{\max}^j|S|\Pi_i|A|_i m/(\delta\beta\epsilon))}{\beta^2\epsilon^2}\big)^{1/\omega}\Big) \qquad \text{(B.23)}$$

Proof. From Lemma 19 we have that

$$Pr\Big[\forall t \in [\tau_{k+1},\tau_{k+2}] : |W_{t;\tau_k}^j| \ge (1-\tfrac{2}{e})\beta D_k^j\Big] \le \tfrac{\delta}{m} \qquad \text{(B.24)}$$

Using the union bound we have that

$$\begin{aligned}&Pr[\forall k \leq m, \forall t \in [\tau_{k+1}, \tau_{k+2}] |W^j_{t;\tau_k}| \geq \tilde{\epsilon}] \\ &\leq \textstyle\sum_{k=1}^{m} Pr[\forall t \in [\tau_{k+1}, \tau_{k+2}] |W^j_{t;\tau_k}| \geq \tilde{\epsilon}] \leq \delta\end{aligned} \tag{B.25}$$

where $\tilde{\epsilon} = (1 - \frac{2}{e})\beta D_k$.

□

Lemma 21. *Let*

$$a_{k+1} = a_k + La_k^{\omega} = a_0 + \sum_{i=0}^{k} La_i^{\omega} \tag{B.26}$$

Then for any constant $\omega \in (0, 1)$, $a_k = \Omega(a_0^{1-\omega}/L + L^{\frac{1}{1-\omega}}((k+1)/2)^{\frac{1}{1-\omega}})$.

Proof. Let us define the following series

$$b_{k+1} = \textstyle\sum_{i=0}^{k} Lb_i^{\omega} + b_0 \tag{B.27}$$

with an initial condition $b_0 = L^{\frac{1}{1-\omega}}$.

Now we lower bound b_k by $(L(k+1)/2)^{\frac{1}{1-\omega}}$. We use induction to prove this hypothesis. For $k = 0$,

$$b_0 = L^{\frac{1}{1-\omega}} \geq (\frac{L}{2})^{\frac{1}{1-\omega}} \tag{B.28}$$

Assume that the induction hypothesis holds for $k - 1$ and prove for k,

$$\begin{aligned}&b_k = b_{k-1} + Lb_{k-1}^{\omega} \\ &= (Lk/2)^{\frac{1}{1-\omega}} + L(Lk/2)^{\frac{\omega}{1-\omega}} \\ &= L^{\frac{1}{1-\omega}}((k/2)^{\frac{1}{1-\omega}} + (k/2)^{\frac{\omega}{1-\omega}}) \\ &\geq L^{\frac{1}{1-\omega}}((k+1)/2)^{\frac{1}{1-\omega}}.\end{aligned} \tag{B.29}$$

For $a_0 \geq L^{\frac{1}{1-\omega}}$ we can view the series as starting at $b_k = a_0$. Since the first time step is 1, we can see that the start point has moved $\Theta(a_0^{1-\omega}/L)$. Therefore, we have a total complexity of $\Omega(a_0^{1-\omega}/L + L^{\frac{1}{1-\omega}}((k+1)/2)^{\frac{1}{1-\omega}})$.

□

Lemma 22. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. Assume that for $t \geq \tau_k$ we have that $Y^j_{t;\tau_k}(s, \boldsymbol{a}) \leq D^j_k$. Then for any $t \geq \tau_k + (1+\psi)L\tau_k = \tau_{k+1}$ we have that $Y^j_{t;\tau_k}(s, \boldsymbol{a}) \leq (\gamma + \frac{2}{2+\psi}\beta)D^j_k$*

Proof. For each state-joint action pair, we are assured that $n(s, \boldsymbol{a}, \tau_k, \tau_{k+1}) \geq (1+\psi)\tau_k$, since in an interval of $(1+\psi)L\tau_k$ steps, each state-joint action pair is visited at least $(1+\psi)\tau_k$ times by the definition of covering time.

Let $Y^j_{\tau_k,\tau_k}(s, \boldsymbol{a}) = \gamma D^j_k + \rho^j_{\tau_k}$, where $\rho^j_{\tau_k} = (1-\gamma)D^j_k$. We now have,

$$\begin{aligned}
\rho^j_t &= (1-\gamma)\Pi^{t-\tau_k}_{l=1}(1-\alpha_{l+\tau_k}) \\
&= 2\beta D^j_k \Pi^{t-\tau_k}_{l=1}(1-\alpha_{l+\tau_k}) \\
&= 2\beta D^j_k \Pi^{t-\tau_k}_{l=1}(1-\tfrac{1}{|n(s,\boldsymbol{a},\tau_{k+1},l)|}),
\end{aligned} \tag{B.30}$$

where the last identity follows from the fact that $\alpha_t = \frac{1}{n(s,\boldsymbol{a},0,t)}$. Since the τ_k's are monotonically decreasing, using $t = \tau_k + (1+\psi)L\tau_k$, we get (using $\psi < 0.712$),

$$\begin{aligned}
\rho_t &\leq 2D^j_k\beta(1-\tfrac{1}{(1+\psi)\tau_k})^{t-\tau_k} \\
&\leq 2D^j_k\beta(1-\tfrac{1}{(1+\psi)\tau_k})^{1+\psi L\tau_k} \\
&\leq 2D^j_k\beta(1-\tfrac{1}{(1+\psi)\tau_k})^{1+\psi\tau_k} \\
&\leq \tfrac{2D^j_k\beta}{e} \\
&\leq \tfrac{2D^j_k\beta}{2+\psi}
\end{aligned} \tag{B.31}$$

Hence, $Y^j_{t;\tau_k}(s, \boldsymbol{a}) \leq (\gamma + \frac{2}{2+\psi}\beta)D^j_k$.

□

Lemma 23. *For any $t \geq \tau_k$ and $1 \leq l \leq t$ we have that $W^{l,j}_{t;\tau_k}(s, \boldsymbol{a})$ is a martingale sequence, which satisfies,*

$$|W^{l,j}_{t;\tau_k}(s, \boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s, \boldsymbol{a})| \leq \frac{Q^j_{\max}}{n(s, \boldsymbol{a}, 0, t)} \tag{B.32}$$

Proof. $W^{l,j}_{t;\tau_k}(s,\boldsymbol{a})$ is a martingale difference sequence since,

$$\begin{aligned}&\mathbb{E}[W^{l,j}_{t;\tau_k}(s,\boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s,\boldsymbol{a})|F_{\tau_k+l-1}]\\ &= \mathbb{E}[\eta^{k,t,j}_{\tau_k+l}(s,\boldsymbol{a})w^{j}_{\tau_k+l}(s,\boldsymbol{a})|F_{\tau_k+l-1}]\\ &= \eta^{k,t,j}_{\tau_k+l}(s,\boldsymbol{a})\,\mathbb{E}[w^{j}_{\tau_k+l}(s,\boldsymbol{a})|F_{\tau_k+l-1}] = 0\end{aligned} \tag{B.33}$$

For linear learning rate we have that $\eta^{k,t,j}_{\tau_k+l}(s,\boldsymbol{a}) \leq \alpha_{l+\tau_k} = 1/n(s,\boldsymbol{a},0,t)$, thus

$$\begin{aligned}&|W^{l,j}_{t;\tau_k}(s,\boldsymbol{a}) - W^{l-1,j}_{t;\tau_k}(s,\boldsymbol{a})|\\ &= \eta^{k,t,j}_{\tau_k+l}(s,\boldsymbol{a})|w^{j}_{\tau_k+l}(s,\boldsymbol{a})|\\ &\leq \tfrac{Q^{j}_{\max}}{n(s,\boldsymbol{a},0,t)}.\end{aligned} \tag{B.34}$$

□

Lemma 24. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. With probability at least* $1-\frac{\delta}{m}$ *we have that for every state-joint action pair* $|W^{j}_{t;\tau_k}(s,\boldsymbol{a})| \leq \frac{\psi}{2+\psi}\beta D^{j}_{k}$*, for any* $t > \tau_{k+1}$ *and any positive constant* $\psi \leq 0.712$*, i.e.,*

$$Pr\Big[\forall t \in [\tau_{k+1},\tau_{k+2}] : W^{j}_{t;\tau_k}(s,\boldsymbol{a}) \leq \tfrac{\psi}{2+\psi}\beta D^{j}_{k}\Big] \geq 1 - \tfrac{\delta}{m} \tag{B.35}$$

given that $\tau_k \geq \Theta(\frac{Q^{2,j}_{\max}\ln(Q^{j}_{\max}|S|\Pi_i|A|_i m)/(\psi\delta\beta D_k)}{\psi^2\beta^2 D^2_k})$.

Proof. By Lemma 23 we can apply Azuma's inequality on $W^{t-\tau_k+1,j}_{t;\tau_k}$ (note that $W^{t-\tau_k+1,j}_{t;\tau_k} = W_{t;\tau_k}$) with $c_i = \Theta\Big(\frac{Q^{j}_{\max}}{n(s,a,0,t)}\Big)$ for any $t \geq \tau_{k+1}$. Therefore, we derive that

$$Pr[|W^{j}_{t;\tau_k} \geq \tilde{\epsilon}|] \leq 2e^{\frac{-2e\tilde{\epsilon}^2}{\Sigma^{t}_{i=\tau_k} c^2_i}} \leq 2e^{-c\frac{\tilde{\epsilon}^2 n(s,\boldsymbol{a},\tau_k,t)}{Q^{2,j}_{\max}}} \tag{B.36}$$

for some positive constant c.

Let us define,

$$\begin{aligned}&\zeta_t(s,\boldsymbol{a}) = 1, \text{ if } \alpha_t(s,\boldsymbol{a}) \neq 0\\ &\zeta_t(s,\boldsymbol{a}) = 0, \text{ otherwise .}\end{aligned} \tag{B.37}$$

Using the union bound and the property that, in an interval of length $(1+\psi)L\tau_k$, each state-joint action pair is visited at least $(1+\psi)\tau_k$ times, we get

$$\begin{aligned}
&Pr\Big[\forall t \in [\tau_{k+1}, \tau_{k+2}] : |W^j_{t;\tau_k}(s,\boldsymbol{a})| \geq \tilde{\epsilon}\Big]\\
&\leq Pr[\forall t \geq ((1+\psi)L+1)\tau_k : |W^j_{t;\tau_k}(s,\boldsymbol{a})| \geq \tilde{\epsilon}]\\
&\leq \textstyle\sum_{t=((1+\psi)L+1)\tau_k}^{\infty} Pr\Big[|W^j_{t;\tau_k}(s,\boldsymbol{a})| \geq \tilde{\epsilon}\Big]\\
&\leq \textstyle\sum_{t=((1+\psi)L+1)\tau_k}^{\infty} \zeta_t(s,\boldsymbol{a}) 2e^{-c\frac{\tilde{\epsilon}^2 n(s,\boldsymbol{a},0,t)}{Q^{2,j}_{\max}}}\\
&\leq 2e^{-c\frac{\tilde{\epsilon}^2((1+\psi)\tau_k)}{Q^{2,j}_{\max}}} \textstyle\sum_{t=0}^{\infty} e^{-\frac{t\tilde{\epsilon}^2}{Q^{2,j}_{\max}}}\\
&= \frac{2e^{-c\frac{\tilde{\epsilon}^2((1+\psi)\tau_k)}{Q^{2,j}_{\max}}}}{1-e^{\frac{-\tilde{\epsilon}^2}{Q^{2,j}_{\max}}}}\\
&= \Theta\Big(\frac{Q^{2,j}_{\max} e^{\frac{-c'\tau_k\tilde{\epsilon}^2}{Q^{2,j}_{\max}}}}{\tilde{\epsilon}^2}\Big)
\end{aligned} \tag{B.38}$$

for some positive constant c'. Setting $\frac{\delta}{m|S|\Pi_i|A|_i} = \Theta\Bigg(\frac{e^{-\frac{c'\tau_k\tilde{\epsilon}^2}{Q^{2,j}_{\max}}} Q^{2,j}_{\max}}{\tilde{\epsilon}^2}\Bigg)$, which holds for $\tau_k = \Theta(\frac{Q^{2,j}_{\max}\ln(Q^j_{\max}|S|\Pi_i|A|_i m/(\delta\tilde{\epsilon}))}{\tilde{\epsilon}^2})$, and $\tilde{\epsilon} = \frac{\psi}{2+\psi}\beta D^j_k$ assures us that for every $t \geq \tau_{k+1}$ (and as a result for any $t \in [\tau_{k+1}, \tau_{k+2}]$) with probability at least $1 - \frac{\delta}{m}$ the statement holds at every state-joint action pair.

$\square$

Lemma 25. *Consider the low-Q update given in Eq. 4.10, with a linear learning rate. With probability* $1-\delta$*, for every iteration* $k \in [1,m]$*, time* $t \in [\tau_{k+1}, \tau_{k+2}]$*, and any positive constant* $\psi \leq 0.712$*, we have* $|W^j_{t;\tau_k}| \leq \frac{\psi\beta D^j_k}{2+\psi}$*, i.e.,*

$$Pr\Big[\forall k \in [1,m], \forall t \in [\tau_{k+1}, \tau_{k+2}] : |W^j_{t;\tau_k}| \leq \frac{\psi\beta D^j_k}{2+\psi}\Big] \geq 1-\delta \tag{B.39}$$

given that $\tau_0 = \Theta\Big(\frac{Q^{2;j}_{\max} \ln(Q^j_{\max}|S||A|m/(\delta\beta\epsilon\psi))}{\psi^2\beta^2\epsilon^2}\Big)$

Proof. From Lemma 24, we know that

$$Pr\Big[\forall t \in [\tau_{k+1}, \tau_{k+2}] : |W^j_{t;\tau_k}| \geq \tfrac{\psi\beta D^j_k}{2+\psi}\Big] \leq \tfrac{\delta}{m}. \tag{B.40}$$

Using the union bound, we have that,

$$\begin{aligned}
&Pr\Big[\forall k \leq m, \forall t \in [\tau_{k+1}, \tau_{k+2}] : |W^j_{t;\tau_k}| \geq \tfrac{\psi\beta D^j_k}{2+\psi}\Big] \\
&\leq \textstyle\sum_{k=1}^m Pr\Big[\forall t \in [\tau_{k+1}, \tau_{k+2}] |W^j_{t;\tau_k}| \geq \tfrac{\psi\beta D^j_k}{2+\psi}\Big] \\
&\leq \delta.
\end{aligned} \tag{B.41}$$

□

B.2 Experimental Details

This section provides the complete details of all of our experimental domains, including details about the reward function and the advisors used. For the Pommerman and Pursuit environments, we assume that all the actions of other agents are either directly observable (fully observable), shared amongst agents, or provided by the game engine to perform centralized updates. For the MPE environment, the actions of other agents are observable only during training and not during execution (CTDE).

B.2.1 Pommerman

In Pommerman, the complete set of skills needed to be learned in order to win games include 1) escaping from the enemy, 2) obtaining power ups (bombs/life), 3) killing the enemy, 4) blasting walls to open routes, and 5) coordinating with a teammate (in the team version) [203].

In our experiments, we consider two Pommerman domains. The first four experiments use the two-agent version of Pommerman and the fifth experiment uses the four-agent team version of Pommerman. Each episode in our training and execution experiments corresponds to a full Pommerman game with a randomized board and a maximum of 800 steps before completion. The game ends either when all the steps are completed or when one of the two Pommerman agents dies (two-agent version). In the team version, the game ends either when all the steps are complete (800 steps maximum) or when one of the two teams completely dies.

The first experiment (Experiment 1) uses four sufficient advisors of varying quality. The first advisor (Advisor 1) can teach all the strategies (i.e., various Pommerman skills as mentioned in Section 4.7) needed to win the Pommerman game. Advisor 2 can teach moves associated with killing the opponent if the opponent is very close to the agent and defensive strategies that help avoid the enemy. The third advisor (Advisor 3) can only teach defensive strategies that help in escaping the enemy, and cannot teach aggressive strategies needed to kill the enemy. Finally, the last advisor (Advisor 4) only provides random actions. Hence, all the agents should follow the first advisor as far as possible, and the fourth advisor must be avoided entirely.

In the second experiment (Experiment 2), we use a new set of four advisors. Here, the first advisor teaches only defensive skills (escaping the enemy), the second advisor can only teach aggressive skills (killing the enemy), the third advisor can only teach strategies that enable obtaining the power ups, and the fourth advisor teaches ways to seek and blast open wooden walls which opens up various paths in the game. It can be seen here that no one advisor is can teach all the strategies needed to win in Pommerman. However, together, all four advisors can teach the requisite strategies, and they need to be leveraged appropriately.

In Experiment 3, we use a set of four advisors of decreasing quality as in the first experiment; however, none of the advisors are can teach strategies that seek enemies and kill them (insufficient set). Also, none of the advisors are capable of teaching the skills needed to seek wooden walls to blast open. Hence, this set of advisors is insufficient for winning the Pommerman game. The first advisor (Advisor 1) can teach strategies to escape from an enemy, kill the enemy if the enemy is right next to the agent, and obtaining the power-ups. The second advisor (Advisor 2) can only teach strategies that help in escaping the enemy or killing an enemy close to the agent. Advisor 3 can only teach strategies that pertain to escaping the enemy, and Advisor 4 only provides random suggestions.

In Experiment 4, we have a set of advisors similar to the second experiment; however, the advisors are incapable of teaching sufficient skills needed to win in Pommerman. The first advisor (Advisor 1) teaches only defensive skills to escape an enemy. The second advisor (Advisor 2) helps in learning a strategy that can kill an enemy right next to the agent. The third advisor (Advisor 3) can teach strategies that lead to obtaining the power-ups, and the fourth advisor (Advisor 4) can teach skills needed to blast open wooden walls, if the agent is very close to the wall.

The fifth experiment (Experiment 5) with the team domain uses the same set of advisors as Experiment 1.

The Pommerman environment was released by Resnick et al. [203] under the Apache2 license.

B.2.2 Pursuit domain

The pursuit domain was first introduced by [89], and we use the implementation provided by the Petting Zoo environment [269] (released under MIT license). The game has a set of 8 pursuer agents cooperating with each other to capture a set of 30 evaders in the environment. We use the same reward function and environmental parameters as in [269] with some minor modifications. In our setting, the agents get a reward of $+1$ for hitting (tagging) the evaders and a reward of $+30$ for catching an evader. An evader is decided to be caught if it is surrounded by a group of at least two pursuer agents. The captured evaders are removed from the environment. All agents get an urgency penalty of -0.1 at each time step of the game. Each episode has a maximum of 500 steps. The episode terminates either when all the evaders are captured or when all the steps are completed. Each pursuer observes a 7×7 grid around itself (the entire grid is 16×16), which means that the pursuer can get full information about other pursuers and evaders within this observable grid and no information outside this grid.

B.2.3 Multi Particle Environment (Predator-Prey)

The Multi Particle Environment (MPE) was originally released by [156] as a set of testbeds for the purpose of testing algorithms that pertain to cooperative and mixed cooperative-competitive settings with characteristics of communication and obstacle interactions. From this suite of testbeds we use the Simple Tag environment that pertains to a Predator-Prey setting where a set of three predators try to capture a single prey. We use the environment defined as a part of the Petting Zoo library [269] (released under MIT license). Here all agents have a discrete action space with a set of 5 actions (four cardinal directions and one action that signifies no movement). Both the predators and prey have a continuous observation space that corresponds to the velocity and position of all agents including the agent itself. The predators get a reward of $+10$ for hitting (colliding/tagging) prey and the prey get a punishment of -10 for being hit by any predator (the prey is not removed from the environment). The prey also receives a small additional penalty for exiting the field of play (see [269] for more details). All the predators receive the same reward at all time steps (global reward structure). Our environment contains eight predators and eight prey, in addition to five obstacles that block the path of the predators and prey. Each game contains 500 steps of training or execution. We model this domain as a CTDE setting, where the actions taken, and the rewards obtained by all agents are available to each agent during training, but not available during execution.

B.3 Hyperparameters And Implementation Details

All the hyperparameters in our implementation of baselines are either the same or closely match the values recommended by the respective papers that introduced these algorithms. Our algorithms also use similar hyperparameters as the baseline algorithms, with a few exceptions (for performance and computational efficiency reasons).

The DQN [167], CHAT [289], TLQL [145], ADMIRAL-DM [254], and MA-TLQL implementations use almost the same hyperparameters. These algorithms use a learning rate of 0.01, a discount factor of 0.9, a replay memory size of 2×10^6, and a fixed exploration rate of 0.9. The target network is replaced every 10 learning iterations using the hard replacement strategy. The evaluation and target networks use 3 fully connected layers (2 ReLU layers of 50 neurons and an output layer). We use a batch size of 32.

For the CHAT implementation, we use the neural network based confidence variant (NNHAT) from Wang and Taylor [289]. We set a confidence threshold of 0.6 and use 3 fully connected layers (2 layers using ReLU as the activation function with 50 neurons and an additional output layer). The advisors are directly used in CHAT instead of preparing decision based rules from classifier models as done in [289] due to performance reasons and also because the advisors used in our experiments are either rule-based agents or pretrained networks and not human advisors as designed in CHAT. Since these advisors are considered to be extracted rules in CHAT (not actual demonstrators/advisors), we allow CHAT dependence on advisors in the execution phase as well.

We use the PPR technique in TLQL for our experiments, though this is not used in Li et al. [145]. This is done due to two reasons. First, unlike in single-agent settings, independent Q-learning methods do not have a policy improvement guarantee in multi-agent settings (as discussed in Section 4.4) [264], hence, the vanilla TLQL is not guaranteed to stop depending on advisors as motivated by Li et al. [145]. Second, without PPR, the experimental TLQL training performance is very good (since it has unlimited dependence on advisors), while the execution performances are very poor (since advisors are not available during execution). The best execution performances for TLQL is obtained while using PPR.

Regarding the implementation of the PPR technique, the TLQL, ADMIRAL-DM, MA-TLAC, and MA-TLQL implementations start with a value of $\epsilon' = 1$, which is linearly decayed to 0 at the end of training. There is no influence of advisors during execution, and hence, $\epsilon' = 0$ during execution.

To stay consistent with the description in [145], the TLQL implementation uses a total of 3 networks (evaluation and target networks for low-Q in addition to high-Q). The high-Q and low-Q networks use fully connected layers with the same architecture as described for the DQN. The MA-TLQL implementation uses four networks (two evaluation and target networks for the low-Q and high-Q respectively), also having the same configuration as described for the DQN. Further, the MA-TLQL uses a second replay buffer for the high-Q in addition to the replay buffer of the low-Q. Both buffers use the same memory size of 2×10^6.

Regarding DQfD [100], we set 1×10^6 as the demo buffer size and perform 50,000 mini-batch updates for pretraining. The replay buffer size is twice the size of the demo buffer. The N-step return weight is 1.0, the supervised loss weight is 1.0 and the L2 regularization weight is 10^{-5}. The epsilon greedy exploration is 0.9. The discount factor is 0.99 and the learning rate is 0.002. The network architecture uses 3 fully connected layers (2 ReLU layers of 24 neurons and an output layer). The pretraining for DQfD comes from a data buffer related to a series of games where the advisors compete against each other.

The MA-TLAC uses two actor networks and two critic networks. The network architecture is the same as described for MA-TLQL. The actor networks use a learning rate of 10^{-6}, and the critic networks use a learning rate of 10^{-3}.

For all training experiments, we use a set of 30 random seeds $(1 - 30)$. We use a new set of 30 random seeds $(31 - 60)$ for the execution experiments.

B.4 Wall Clock Times

All the training for the experiments were conducted on a virtual machine having 2 Nvidia A100 GPUs with a GPU memory of 40 GB. The CPUs use the AMD EPYC processors with a memory of 125 GB. The Pommerman experiments took an average of 12 hours wall clock time to complete, and the Pursuit experiments took an average of 15 hours wall clock time to complete.

B.5 Illustrative Example

In this section we would like to show a toy example where the TLQL updates as provided by Li et al. [145] takes a longer time to figure out the best advisor from a set of advisors, as compared to the MA-TLQL updates, we introduced in Chapter 4. We will just use a single agent based grid-world environment as given in Figure B.1, instead of multi-agent environments. The TLQL updates will use the Bellman update for the low-Q updates and a subsequent synchronization step to update its high-Q values as proposed in Li et al. [145]. In MA-TLQL, the low-Q values will use the control update as given in Eq. 4.3 albeit in a single-agent fashion (joint actions need not be considered), since this environment only contains one agent. The high-Q will use the evaluation update as given in Eq. 4.1 in a single-agent fashion. Also, for simplicity, we do not use the ensemble selection strategy in Eq. 4.2 for the MA-TLQL action selection in this example. Rather we only use the high-Q and low-Q updates for advisor and action evaluation at the given state. Here we have a set of 6 states $\{S1, S2, S3, S4, S5, G\}$ with the agent starting at state $S1$ and trying to reach the goal state G. The states $S3, S4$ and G are the terminal states where the agent receives a reward of -1 in states $S3$ and $S4$, and a reward of $+1$ in state G. The state $S5$ only exists for symmetry (cannot be reached in practice). The agent can take one of the two actions $\{R, D\}$ (to denote right and down respectively) at each state. In this environment, it can be seen that the agent needs to take action R in states $S1$ and $S2$ to obtain the maximum rewards. The agent has access to two advisors $A1$ and $A2$, where $A1$ is an optimal advisor providing the correct action (right) at every state and $A2$ is a sub-optimal advisor which provides action R with probability 0.5 and the action D with probability 0.5. All transitions in this domain are deterministic. Also, we specify that the learning rate (α) is 0.1 and the discount factor (γ) is 0.9.

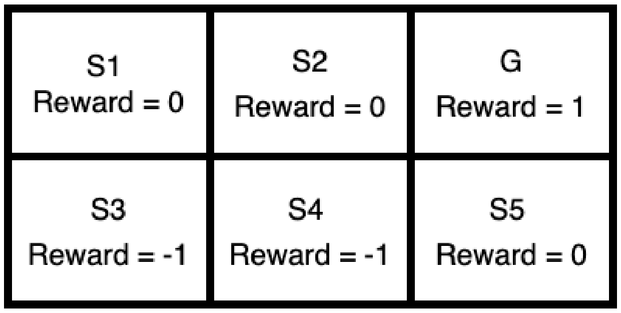


Figure B.1: Toy environment to compare updates of TLQL and MA-TLQL.

Now we consider the Q-updates (high-level and low-level) pertaining to both MA-TLQL

(ST./AC.)	Right	Down
S1	0	0
S2	0	0

(a) Low-Q values at time $t = 1$

(ST./AC.)	Adv. 1	Adv. 2
S1	0	0
S2	0	0

(b) TLQL high-Q at time $t = 1$

(ST./AC.)	Adv. 1	Adv. 2
S1	0	0
S2	0	0

(c) MA-TLQL high-Q at time $t = 1$

Table B.1: TLQL and MA-TLQL updates at time $t = 1$. The columns refer to the actions and the rows refer to the states.

and TLQL. At the beginning, we initialize all the Q-values to 0 arbitrarily.

At the initial time step ($t = 0$) let us assume that the agent starts at the initial state $S1$. Let both the advisors suggest action R. Then the agent takes this action and first updates its low-Q using the equation,

$$\begin{aligned}
&lowQ_1(S1, R) = lowQ_0(S1, R) + \alpha\Big(r + \gamma \max_a lowQ_0(S2, a) - lowQ_0(S1, R)\Big) \\
&lowQ_1(S1, R) = 0 + 0.1\Big(0 + 0.9 \times 0 - 0\Big) \\
&lowQ_1(S1, R) = 0.
\end{aligned} \tag{B.42}$$

Now, TLQL will set the value of the high-Q of both advisors to be 0 (synchronization). The MA-TLQL updates will also yield a value of 0 for both the advisors. In Table B.1 we tabulate the Q-values for the non-terminal states ($S1$ and $S2$).

At the next time step $t = 2$, the agent is at state $S2$. Again let both advisors suggest the right action. Now the low-Q is updated as,

(ST./AC.)	Right	Down
S1	0	0
S2	0.1	0

(a) Low-Q values at time $t = 2$

(ST./AC.)	Adv. 1	Adv. 2
S1	0	0
S2	0.1	0.1

(b) TLQL high-Q at time $t = 2$

(ST./AC.)	Adv. 1	Adv. 2
S1	0	0
S2	0.1	0.1

(c) MA-TLQL high-Q at time $t = 2$

Table B.2: TLQL and MA-TLQL updates at time $t = 2$.

$$\begin{aligned}
&lowQ_2(S2, R) = lowQ_1(S2, R) + \alpha\Big(r + \gamma \max_a lowQ_1(G, a) - lowQ_1(S2, R)\Big)\\
&lowQ_2(S2, R) = 0 + 0.1\Big(1 + 0.9 \times 0 - 0\Big)\\
&lowQ_2(S2, R) = 0.1
\end{aligned} \tag{B.43}$$

We specify that when the next state is terminal, the temporal difference (T.D.) target is the reward itself. Both the algorithms, TLQL and MA-TLQL, will have the same high-Q values in this case as well. The Q-values for time $t = 2$ are tabulated in Table B.2.

Now, we move to time $t = 3$. Since the goal has been reached at the previous time step, the agent resets back to the initial state $S1$. Now, let us assume that both advisors specify the right action again at state $S1$. The low-Q values are updated as,

$$\begin{aligned}
&lowQ_3(S1, R) = lowQ_2(S1, R) + \alpha\Big(r + \gamma \max_a lowQ_2(S2, a) - lowQ_2(S2, R)\Big)\\
&lowQ_3(S1, R) = 0 + 0.1\Big(0 + 0.9 \times 0.1 - 0\Big)\\
&lowQ_3(S1, R) = 0.009
\end{aligned} \tag{B.44}$$

Again, high-Q values for MA-TLQL and TLQL will be the same as the low-Q values and are tabulated in Table B.3.

(ST./AC.)	Right	Down
S1	0.009	0
S2	0.1	0

(a) Low-Q values at time $t = 3$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.009	0.009
S2	0.1	0.1

(b) TLQL high-Q at time $t = 3$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.009	0.009
S2	0.1	0.1

(c) MA-TLQL high-Q at time $t = 3$

Table B.3: TLQL and MA-TLQL updates at time $t = 3$.

Next, the agent moves to state $S2$. We are at time $t = 4$. Let the advisor $A2$ specify action D at this state (Advisor $A1$ always specifies R). Also let us assume that the agent chooses to listen to $A2$ at this state (Q-values of both advisors are the same, so the agent is indifferent between the two advisors) and hence it performs action D (down) from $A2$. Now the Q-value for the low-Q can be updated as,

$$\begin{aligned} &lowQ_4(S2, D) = lowQ_4(S2, D) + \alpha\Big(r + \gamma \max\nolimits_a lowQ_4(S4, a) - lowQ_4(S2, D)\Big) \\ &lowQ_4(S2, D) = -0.1 \end{aligned} \tag{B.45}$$

The high-Q values for the MA-TLQL update is given by,

$$\begin{aligned} &highQ_4(S2, A2) = highQ_3(S2, A2) + \alpha\Big(r + \gamma highQ_3(S4, A2) - highQ_3(S2, A2)\Big) \\ &highQ_4(S2, A2) = 0.1 + 0.1(-1 - 0.1) \\ &highQ_4(S2, A2) = -0.01 \end{aligned} \tag{B.46}$$

All Q-values for time $t = 4$ are tabulated in Table B.4.

Since the state $S4$ was a terminal state, the agent is back to state $S1$. We are at time $t = 5$. At this state, let us assume that both the advisors specify action R. Now the agent chooses to perform this action and updates its low-Q using,

(ST./AC.)	Right	Down
S1	0.009	0
S2	0.1	-0.1

(a) Low-Q values at time $t = 4$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.009	0.009
S2	0.1	-0.1

(b) TLQL high-Q at time $t = 4$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.009	0.009
S2	0.1	-0.01

(c) MA-TLQL high-Q at time $t = 4$

Table B.4: TLQL and MA-TLQL updates at time $t = 4$.

$$\begin{aligned}
&lowQ_5(S1, R) = lowQ_4(S1, R) + \alpha\Big(r + \gamma \max\nolimits_a lowQ_4(S2, a) - lowQ_4(S2, R)\Big)\\
&lowQ_5(S1, R) = 0.009 + 0.1\Big(0 + 0.9 \times 0.1 - 0.009\Big)\\
&lowQ_5(S1, R) = 0.0171.
\end{aligned} \tag{B.47}$$

Since both the advisors specified action R, both advisors are assigned the same Q-values in the high-Q in the TLQL update. Now, the high-Q value of advisor $A1$ will be the same as the low-Q value in the MA-TLQL update as well. However, the high-Q value of the advisor $A2$ of the MA-TLQL update will be as follows,

$$\begin{aligned}
&highQ_5(S1, A2) = highQ_4(S1, A2) + \alpha\Big(r + \gamma highQ_4(S2, A2) - highQ_4(S1, A2)\Big)\\
&highQ_5(S1, A2) = 0.009 + 0.1\Big(0 + 0.9 \times -0.01 - 0.009\Big)\\
&highQ_5(S1, A2) = 0.009 - 0.0018 = 0.0072
\end{aligned} \tag{B.48}$$

At this stage (time $t = 5$) all the Q-values are tabulated in Table B.5. Comparing the high-Q value of TLQL and MA-TLQL, we see that in the TLQL updates (Table B.5b), the agent is indifferent between following advisor $A1$ or advisor $A2$ in state $S1$ (same Q-values), while it would decide to choose advisor $A1$ at the state $S2$. Even after 5 update steps

(ST./AC.)	Right	Down
S1	0.0171	0
S2	0.1	-0.1

(a) Low-Q values at time $t = 5$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.0171	0.0171
S2	0.1	-0.1

(b) TLQL high-Q at time $t = 5$

(ST./AC.)	Adv. 1	Adv. 2
S1	0.0171	0.0072
S2	0.1	-0.01

(c) MA-TLQL high-Q at time $t = 5$

Table B.5: TLQL and MA-TLQL updates at time $t = 5$.

TLQL has not been able to determine the right advisor (as advisor $A1$ is better than $A2$) for both states. In contrast, the MA-TLQL updates found in Table B.5c clearly show a higher Q-value for the advisor $A1$ than the advisor $A2$ for both the states. This example presents a situation where the MA-TLQL distinguishes between a good and a bad advisor faster than the vanilla TLQL update as introduced by Li et al. [145].

Appendix C

Appendix For Chapter 5

C.1 Illustrative Example

This section provides an example of a simulated scenario where the Multi Type Mean Field algorithm is more useful than the simple mean field algorithm.

Consider a game in which the central agent has to decide the direction of spin. The domain is stateless. The spin direction is influenced by the direction of spin of its A,B,C and D neighbours (four neighbours in total). Here we denote A as the neighbour to the left of the agent, B as the neighbour to the top of the agent, C as the neighbour to the right of the agent, and D as the neighbour to the bottom of the agent. The neighbour agents spin in one direction at random. If the agent spins in the same direction as both of its C and D neighbours, the agent gets a reward of -2 regardless of what the A and B are doing. If the spin is in the same direction as both of its A and B neighbours, the agent gets a +2 reward unless the direction is not the same as the one used by both of its C and D neighbours. All other scenarios result in a reward of 0. So the agent in Grid A in Figure C.1 will get a -2 for the spin down (since the C and D neighbours are spinning down) and a +2 for a spin up (since the A and B neighbours are spinning up). In Grid B of Figure C.1 the agent will get a -2 for spin up and a +2 for spin down. It is clear that the best action in Grid A is to spin up and the best action in Grid B is to spin down.

Here we have a notion of multi stage game with many individual stages. In each stage of a multi stage game, one or more players take one action each, simultaneously and obtain rewards.

Consider a sequence of stage games in which the agent gets Grid A for every 2 consecutive stages and then the Grid B for the third stage. The goal of the agent is to take the best possible action at all stages. Let us assume that the agent continues to learn at all stages, and it starts with Q values of 0. We apply MFQ (Equation 5.2) and MTMFQ (Equation 5.20) and show why MFQ fails, but MTMFQ succeeds in this situation. We are going to calculate the Q values for the 3 stages using both the MFQ (from [303]) and the MTMFQ update rules. We approximate the average action using the number of times the neighbourhood agents spin up. In the MTMFQ we use the A, B neighbours as the type 1 and the C, D neighbours as the type 2.

Applying MFQ:

In the first stage,

$$Q_1^j(\uparrow, \overline{a}^j = 2) = 0 + 0.1(2 - 0) = 0.2$$

$$Q_1^j(\downarrow, \overline{a}^j = 2) = 0 + 0.1(-2 - 0) = -0.2$$

Thus, the agent will choose to spin up in the first stage (correct action).

For the second stage,

$$Q_2^j(\uparrow, \overline{a}^j = 2) = 0.38$$

$$Q_2^j(\downarrow, \overline{a}^j = 2) = -0.38$$

Again the agent will choose to spin up in the second stage (correct action).

For the third stage,

$$Q_3^j(\uparrow, \overline{a}^j = 2) = 0.38 + 0.1(-2 - 0.38) = 0.142$$

$$Q_3^j(\downarrow, \overline{a}^j = 2) = -0.38 + 0.1(2 + 0.38) = -0.142$$

Here again the agent will choose to make the spin up (wrong action).

Now coming to MTMFQ updates, for the first stage,

$$Q_1^j(\uparrow, \overline{a}_1^j = 2, \overline{a}_2^j = 0) = 0 + 0.1(2 - 0) = 0.2$$

$$Q_1^j(\downarrow, \overline{a}_1^j = 2, \overline{a}_2^j = 0) = 0 + 0.1(-2 - 0) = -0.2$$

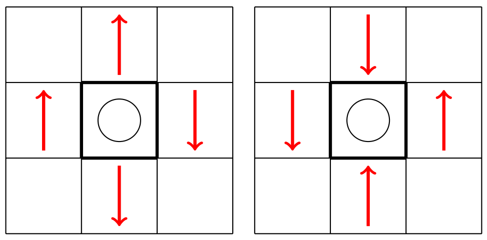


Figure C.1: A counter example to show the failure of MFQ and success of MTMFQ.

Here the agent will spin up (correct action).

For the second stage,

$$Q_2^j(\uparrow, \overline{a}_1^j = 2, \overline{a}_2^j = 0) = 0.38$$

$$Q_2^j(\downarrow, \overline{a}_1^j = 2, \overline{a}_2^j = 0) = -0.38$$

Again the agent will spin up (correct action).

For the third stage,

$$Q_3^j(\uparrow, \overline{a}_1^j = 0, \overline{a}_2^j = 2) = -0.2$$

$$Q_3^j(\downarrow, \overline{a}_1^j = 0, \overline{a}_2^j = 2) = 0.2$$

Now it can be seen that the agent will spin down in this case (correct action).

Thus, the MFQ agent will make one wrong move out of 3 moves whereas the MTMFQ agent will make the right move all the time. In situations like these, where the relationship of the agent with different neighbour agents is different, the MFQ algorithm would fail. The differences would be captured by MTMFQ which would take more efficient actions. This shows an example where MTMFQ outperforms MFQ.

C.2 Experimental Details

This section gives more details about the experimental conditions, especially the reward function.

For the Multi Team Battle domain, the agents in all of these groups get a reward of -0.01 for each move action, and a reward of -1 for dying. Attacking an empty grid has a reward of -0.1. Each of these members has a power of 10 and a damage of 2. When an agent loses all its powers, it dies. The power is like health that the agents maintain which gets depleted on being attacked. The agents can also recover (regain health points) from the attack using a step recovery rate. This is set to be 0.1. The positive rewards for attacking another agent is cyclic in nature with each group preferring to attack and kill a particular opponent group more than others. Group A has a positive reward of 0.2, 0.3, and 0.4 for attacking an agent of group B, C and D respectively, and it gets a reward of 80, 90, and 100 for killing a member of group B, C, and D respectively. Here we will call Group D as the favourable opponent of Group A as A gets more returns for fighting or killing D. The group B has a positive reward of 0.2, 0.3, and 0.4 for attacking a member of group C, D, and A respectively and a positive reward of 80, 90, and 100 for killing a member of Group C, D, and A respectively. Thus, Group A is the favourable opponent of Group B. Group C has a reward of 0.2, 0.3, and 0.4 for attacking Groups D, A, and B respectively. It gets a kill reward of 80, 90, and 100 for killing agents of Group D, A and B respectively. For Group D the order is Group A, B and C with a attack reward of 0.2, 0.3, and 0.4 and kill reward of 80, 90, and 100.

For the Battle-Gathering game, the reward function is similar with an addition of each agent getting a +80 for collecting a food resource. The food resources are stationary objects, but each food resource has a power similar to agents. This power has to be reduced by damage before the food can be captured. Capturing a food will constitute making repeated efforts to gain the food resource by attacking the grid containing food. The agents would get a +0.5 for making such an attack.

For the Predator Prey domain, the predators have a power of 10 and a speed of 2 with an attack range of 2 (they can attack within a distance of 2 units) and they get a dead penalty of -0.1 and an attack penalty of -0.2. The step recovery rate is 0.1. The prey have a faster speed of 2.5 and an attack range of 0. All agents get a reward of +5 for killing agents belonging to other groups. Additionally, Group A gets a reward of +0.5 for attacking a member of Group B (since Group B is also a predator, Group A does not prefer to attack that), +1 for attacking a member of Group C, +3 for attacking a member of Group D (though both Groups C and D are prey, A prefers D to C). Group B gets 0.5 for attacking

A, but gets +1 for attacking D and +3 for attacking C. Group B prefers Group C to Group D. Every attack action entails a punishment for the (attack) receiver which is equal in value to the reward for the attacker.

C.3 Proofs For Lemmas In Chapter 5

Lemma 26. *Under Assumption 5, the Nash operator $\mathscr{H}^{Nash}$ forms a contraction mapping under the complete metric space from $\mathcal{Q}$ to $\mathcal{Q}$ with the fixed point being the Nash Q value of the entire game ($\boldsymbol{Q}_*$), i.e., $\mathscr{H}^{Nash}\boldsymbol{Q}_* = \boldsymbol{Q}_*$.*

Proof. Refer to Theorem 17 in [104] for a detailed proof. □

Lemma 27. *If we have a stochastic process of the form*

$$y_{n+1} - py_n = d \tag{C.1}$$

where n goes from 0 to ∞, then the general solution of this process can be given by

$$\begin{aligned}&\text{if } p = 1:\\ &y_n = dn + a;\\ &\text{if } p \neq 1:\\ &y_n = \tfrac{d}{1-p} + (a - \tfrac{d}{1-p})p^n\end{aligned} \tag{C.2}$$

where $y_0 = a$.

Proof. Consider the expression,

$$y_{n+1} - py_n = d. \tag{C.3}$$

Z transforms will be applied to solve this expression. Z transform is a generalized version of Discrete Time Fourier Transform that transforms the variables into a new subspace where solving the equation is easier. Once we obtain a solution, we can apply inverse Z transforms to get the solution in terms of the original variables.

$$zY(z) - zy_0 - pY(z) = \tfrac{dz}{z-1}$$

$$(z-p)Y(z) = \tfrac{dz}{z-1} + za \tag{C.4}$$

$$Y(z) = \tfrac{dz}{(z-1)(z-p)} + \tfrac{az}{z-p}.$$

Considering $p \neq 1$ we can rewrite the Equation C.4 as

$$Y(z) = dz[\tfrac{1}{(1-p)(z-1)} - \tfrac{1}{(1-p)(z-p)}] + \tfrac{az}{z-p}$$

$$Y(z) = \tfrac{d}{1-p}\tfrac{1}{1-z^{-1}} - \tfrac{d}{1-p}\tfrac{1}{1-pz^{-1}} + \tfrac{a}{1-pz^{-1}} \tag{C.5}$$

$$Y(z) = \tfrac{d}{1-p}\tfrac{1}{1-z^{-1}} + (a - \tfrac{d}{1-p})(\tfrac{1}{1-pz^{-1}}).$$

Now taking the inverse Z transform

$$y_n = [\tfrac{d}{1-p} + (a - \tfrac{d}{(1-p)})p^n]. \tag{C.6}$$

The above result is for the case where $p \neq 1$. If $p = 1$, see that the Equation 5.30 forms an arithmetic progression whose general term is $a + nd$.

□

Lemma 28. *A random process*

$$w_{n+1}(x) = (1 - \alpha_n(x))w_n(x) + \beta_n(x)r_n(x) \tag{C.7}$$

converges to zero w.p.1, if the following conditions are satisfied:

$$\begin{aligned} &1) \qquad \textstyle\sum_n \alpha_n(x) = \infty, \sum_n \alpha_n^2(x) < \infty, \\ &\qquad \textstyle\sum_n \beta_n(x) = \infty \textit{ and } \sum_n \beta_n^2(x) < \infty \\ &\textit{uniformly over } x \textit{ w.p.1} \\ &2) \qquad \mathbb{E}\{r_n(x)|P_n, \beta_n\} = 0 \\ &\qquad \textit{and } \mathbb{E}\{r_n^2(x)|P_n, \beta_n\} \leq C \end{aligned} \tag{C.8}$$

w.p.1, where
$P_n = \{w_n, w_{n-1}, \ldots, r_{n-1}, r_{n-1}, \ldots,$
$\alpha_{n-1}, \alpha_{n-2}, \ldots, \beta_{n-1}, \beta_{n-2}, \ldots\}$

All the random variables are allowed to depend on the past P_n. $\alpha_n(x)$ and $\beta_n(x)$ are assumed to be non-negative and mutually independent given P_n.

Proof. This is the same as Lemma 1 in [111]. The proof is based on that fact that we can divide the process w_{n+1} (both sides of Equation 5.32) by a large value $W(x)$ such that $r_n(x) \ll W(x)$. Now the Equation 5.32 is effectively reduced to

$$w_{n+1}(x) = (1 - \alpha_n(x))w_n(x). \tag{C.9}$$

This is because of the conditions 1 and 2 which guarantees that β_n is a fraction and that the variance of the process r_n is finite.

Now if you consider Equation C.9, the update is in such a way that the process is equal to a fraction of its previous value. Thus, this process converges to 0 w.p.1.

□

Lemma 29. *Consider the stochastic iteration*

$$X_{n+1}(x) = G_n(X_n, Y_n, x) \tag{C.10}$$

where G_n is a sequence of functions and Y_n is a random process. Let $(\Omega, \mathscr{F}, \mathscr{P})$ be a probability space. If the following are satisfied:

1) The process is scale invariant. That is w.p.1 for all $\omega \in \Omega$

$$G(\beta X_n, Y_n(\omega), x) = \beta G(X_n, Y_n(\omega), x). \tag{C.11}$$

2) If we can keep $||X_n||$ bounded by scaling the process then X_n would converge to a constant D w.p.1 under condition 1.

The original process will converge to a constant $D_1 = \frac{D}{\beta_0}$ where β_0 is the scaling factor that was applied to $||X_n||$.

Proof. The intuition of the proof is that we have a process that starts at a value and its first difference reduces with time till the value stabilizes at a point. Now, if this process is invariant to scaling, we can start the process with a small value (after scaling by a small fraction β_0) and then we can select a constant over which the $||X_n||$ should not increase (we can scale the whole process if it goes above that constant). Now according to the second condition the bounded process should converge to a constant D. Here the relation is $D \leq C$. To show that the net effect of the corrections must stay finite w.p.1, note that if $||X_n||$ converges then for any $\epsilon > 0$ there exists M_ϵ such that $||X_n|| \leq D \leq C$ for all $n > M_\epsilon$ with probability at least $1 - \epsilon$. This implies that the norm of the original process does not go above C after M_ϵ. Thus, convergence of $||X_n||$ to constant D would then imply that the scaled version of the original process converges to the same constant D w.p.1 under the bound. Now if we remove the scaling factor the convergence point of the original process is $\frac{D}{\beta_0}$. □

Lemma 30. *A stochastic process X_n which is bounded by the relation*

$$|X_{n+1}(x)| = (1-\alpha)|X_n(x)| + \gamma\beta C_1 + K \tag{C.12}$$

converges to a constant D w.p.1 provided

1) $x \in S$, where S is a finite set.

2) $\sum_n \alpha = \infty, \sum_n \alpha^2 < \infty$, $\sum_n \beta = \infty$, $\sum_n \beta^2 < \infty$, $\mathbb{E}\{\beta|P_n\} \leq E\{\alpha|P_n\}$, uniformly over x w.p.1,
where

$$P_n = \{w_n, w_{n-1}, \ldots, r_{n-1}, r_{n-1}, \ldots, \alpha, \beta\}$$

and α, β and γ are assumed to be non negative.

3) K is finite.

4) $\gamma \in (0, 1)$

5) The original process X_n is scale invariant.

The convergence point of the original process will then be $D = \frac{K+\gamma\beta C_1}{\alpha\beta_0}$, where β_0 is the scaling factor applied to the original process.

Proof. This is simply an application of Lemma 29.

According to condition 5, we have a process that is scale invariant. Let us assume that we applied a scaling factor of β_0 to that process to get the bound as in Equation 5.36. Now, let us consider the iterative process

$$|X_{n+1}(x)| = (1-\alpha)|X_n(x)| + \gamma\beta C_1 + K. \tag{C.13}$$

Equation C.13 is linear in $|X_n(x)|$ and will converge to a point by Lemma 27 w.p.1, to some $X^*(x)$, where $||X^*|| \leq E_1$, where E_1 is a finite arbitrary constant. From Lemma 27 we can see that Equation C.13 converges to a constant and hence, Lemma 29 can be applied. The convergence point will be $\frac{K+\gamma\beta C_1}{\alpha}$ from Lemma 27 for Equation C.13. If we change the value of the bound C_1 then the convergence point will change accordingly. To get the convergence point of the original process, we reapply the scaling factor. Thus, we get that point to be $\frac{K+\gamma\beta C_1}{\alpha\beta_0}$.

□

Lemma 31. *A random iterative process*

$$\Delta_{n+1}(x) = (1-\alpha)\Delta_n(x) + \beta F_n(x)$$

converges to a constant D w.p.1 under the following conditions:

1) $x \in S$, where S is a finite set.

2) $\sum_n \alpha = \infty, \sum_n \alpha^2 < \infty$, and $\sum_n \beta = \infty$, $\sum_n \beta^2 < \infty$, and $\mathbb{E}\{\beta|P_n\} \leq E\{\alpha|P_n\}$, uniformly over x w.p.1.

3) $||\,\mathbb{E}\{F_n(x)|P_n, \beta\}||_w \leq \gamma||\Delta_n||_w + K$, where $\gamma \in (0,1)$ and K is finite.

4) $\boldsymbol{var}\{F_n(x)|P_n, \beta\} \leq C(1 + ||\Delta_n||_W)^2$, where C is some constant. Here

$$P_n = \{X_n, X_{n-1}, \ldots, F_{n-1}, \ldots, \alpha, \beta\}$$

stands for the past at step n. $F_n(x)$ is allowed to depend on the past. α and β are assumed to be non negative. The notation $||.||$ refers to some weighted maximum norm.

The value of this constant $D = \frac{\psi C_1 + \beta|K|}{\alpha\beta_0}$ where $\psi \in (0,1)$ and C_1 is the constant with which the iterative process is bounded. β_0 is the scaling factor that was applied to the original process.

Proof. Be defining $r_n(x) = F_n(x) - \mathbb{E}\{F_n(x)|P_n, \beta\}$ we can decompose the iterative process into two parallel processes given by

$$\begin{aligned} \delta_{n+1}(x) &= (1-\alpha)\delta_n(x) + \beta\,\mathbb{E}\{F_n(x)|P_n, \beta\} \\ w_{n+1}(x) &= (1-\alpha)w_n(x) + \beta r_n(x) \end{aligned} \tag{C.14}$$

where $\Delta_n(x) = \delta_n(x) + w_n(x)$. Dividing both the sides of Equation C.14 by β_0 for each x and denoting $\delta'_n(x) = \delta_n(x)/\beta_0$, $w'_n(x) = w_n(x)/\beta_0$ and $r'_n(x) = r_n(x)/\beta_0$ we can bound the δ'_n process by condition 3.

Now we can rewrite the equation pair from condition 3 as

$$\begin{aligned} |\delta'_{n+1}| &\leq (1-\alpha)|\delta'_n(x)| + \gamma\beta||\ |\delta'| + w'_n|| + \beta|K| \\ w'_{n+1}(x) &= (1-\alpha)w'_n(x) + \gamma\beta r'_n(x). \end{aligned} \tag{C.15}$$

Let us assume that the Δ_n process stays bounded. Then the variance of $r'_n(x)$ is bounded by some constant C and thereby w'_n converges to zero w.p.1 according to Lemma 28. Hence, there exists M such that for all $n > m$, $||w'_n|| < \epsilon$ with probability at least $1-\epsilon$. This implies that the δ'_n process can be further bounded by

$$|\delta'_{n+1}| \leq (1-\alpha)|\delta'_n(x)| + \gamma\beta||\delta'_n - \epsilon|| + \beta|K| \tag{C.16}$$

with probability $> 1-\epsilon$. If we choose C such that $\gamma(C+1)/C \leq 1$ then for $||\delta'_n|| > C\epsilon$

$$\gamma||\delta'_n + \epsilon|| \leq \gamma(C+1)/C||\delta'_n||. \tag{C.17}$$

Note that in the above relation we do not need the term $\frac{C+1}{C}$ to be less than 1. We only need the product of this term with γ to be less that one. Let us represent $F = (C+1)/C$. Now rewriting Equation C.16 we get the following bound,

$$|\delta'_{n+1}| \leq (1-\alpha)|\delta'_n(x)| + \gamma\beta F||\delta'_n|| + \beta|K|. \tag{C.18}$$

Let us bound the norm by C_1. Then we get the bound as

$$|\delta'_{n+1}| \leq (1-\alpha)|\delta'_n(x)| + \gamma\beta F C_1 + \beta|K|. \tag{C.19}$$

Let us assume that δ_n is scale invariant (we prove that below). Now we can apply Lemma 30 as this satisfies all the conditions of Lemma 30. The original process converges to a constant D w.p.1. Again according to Lemma 30 this constant value is $D = \frac{\gamma\beta F C_1 + \beta|K|}{\alpha\beta_0}$ where β_0 is the factor with which the original process was scaled. Let us denote a new fraction $\psi = \gamma\beta F$. Thus, the value of $D = \frac{\psi C_1 + \beta|K|}{\alpha\beta_0}$. Now, this guarantees w.p.1 convergence of the original process under the boundedness assumption if that process is scale invariant. Let the constant $D_1 = \frac{\gamma\beta F C_1 + \beta|K|}{\alpha}$ be the point Equation C.19 converges to according to Theorem 27. Since, the value of D_1 is very small for a small K (we will show that K is small in the application) and a small C_1 (since C_1 is arbitrary, we can choose a small C_1), we can get $|\delta_n| \approx \delta_n$.

Now we prove the scale invariance condition same as Jaakola et al. [111]. By Condition 4, $r'_n(x)$ can be written as $(1 + ||\delta_n + w_n||)s_n(x)$, where $E\{s_n^2(x)|P_n\} \leq C$. Let us now decompose w_n as $u_n + v_n$ with

$$\begin{aligned} u_{n+1}(x) &= (1-\alpha)u_n(x) + \gamma\beta||\delta'_n + u_n + v_n||s_n(x) \\ v_{n+1}(x) &= (1-\alpha)v_n(x) + \gamma\beta s_n(x) \end{aligned} \tag{C.20}$$

and v_n converges to zero w.p.1 by Lemma 28. Again by choosing C such that $\gamma(C+1)/C < 1$ we can bound the δ'_n and u_n processes for $||\delta'_n + u_n|| > C\epsilon$. The pair (δ'_n, u_n) is then a scale invariant process whose bounded version was proven earlier to converge to D w.p.1. This proves the w.p.1 convergence of the triple δ'_n, u_n, v_n bounding the original process.

$\square$

Appendix D

Appendix For Chapter 6

D.1 Proof Of Lemmas In Chapter 6

Lemma 32. *For every $k \geq 0$, we have*

$$F(A^k) \leq A^{k+1} \leq A^k + D, \tag{D.1}$$

and

$$F(L^k) \geq L^{k+1} \geq L^k - D \tag{D.2}$$

Proof. The proof is by induction on k.

Consider,

$$\begin{aligned} F(A^0) &= F(x^* + p) \\ &\leq F(x^*) + p \\ &\leq x^* + p + D \\ &\leq A^0 + D \end{aligned} \tag{D.3}$$

The second step is from Assumption 9(d) and the third step is from Assumption 9(c).

Now,

$$A^1 = \tfrac{A^0+F(A^0)+D}{2}$$

$$\leq \tfrac{A^0+A^0+D+D}{2}$$

$$= A^0 + D$$

In the above equation, we have applied the inequality for $F(A^0)$ obtained in Eq. D.3.

Let us assume that

$$A^{k+1} \leq A^k + D$$

This will imply, due to the monotonicity assumption, that

$$F(A^{k+1}) \leq F(A^k + D) \leq F(A^k) + D$$

Now consider,

$$A^{k+2} = \frac{A^{k+1} + F(A^{k+1}) + D}{2} \tag{D.4}$$

and

$$A^{k+1} = \tfrac{A^k+F(A^k)+D}{2}$$

$$A^{k+1} + D = \tfrac{A^k+D+F(A^k)+D+D}{2}$$

$$A^{k+1} + D \geq \tfrac{A^{k+1}+F(A^{k+1})+D}{2}$$

$$A^{k+1} + D \geq A^{k+2}$$

This proves that $A^{k+1} \leq A^k + D$.

From the definition of A^{k+1}, we find that

$$A^{k+1} = \tfrac{A^k+F(A^k)+D}{2}$$

$$2A^{k+1} \geq A^{k+1} + F(A^k)$$

$$A^{k+1} \geq F(A^k)$$

Hence, we find that $F(A^k) \leq A^{k+1}$, proving Eq. 6.28.

Using an entirely symmetrical argument, we can also prove that Eq. 6.29 is true. $\square$

Lemma 33. *The sequence A^k will converge to a point upper bounded by $x^* + 2D$ and the sequence L^k will converge to a point lower bounded by $x^* - 2D$.*

Proof. We will first show that the sequence A^k remains bounded from below by $x^* - D$. That is, we will show that $A^k \geq x^* - D$ for all k. This is true for A^0, by definition. Suppose that $A^k \geq x^* - D$. Then, by monotonicity and Assumption 9(c), $F(A^k) \geq F(x^* - D) \geq x^* - D$, from which the inequality $A^{k+1} \geq F(A^k) \geq x^* - D$ follows (from Lemma 32).

Let us consider a $\mathcal{B}$ such that, $\mathcal{B} \gg D$ and $\mathcal{B} > p$. Now, since D is the bound of the F mapping, but p is arbitrary, we can find such a $\mathcal{B}$ for a small D (we will show that D is small in our application in Theorem 15). Our objective is to prove that the sequence A^k is upper bounded by $x^* + \mathcal{B}$. This is true for A^0 by definition. Let us assume that $A^k \leq x^* + \mathcal{B}$. Then the inequality $A^k + D \leq x^* + \mathcal{B} + D \approx x^* + \mathcal{B}$ follows. This implies that $A^{k+1} \leq A^k + \mathcal{B}$ from Lemma 32. Hence the sequence A^k also has an upper bound.

Now we have from Lemma 32,

$$A^{k+1} \leq A^k + D$$

$$A^{k+2} \leq A^{k+1} + D$$

Subtracting we get,

$$A^{k+2} - A^{k+1} \leq A^{k+1} - A^k$$

This shows that A^k sequence has its first difference reducing. So either this sequence should converge to a point or it should diverge to infinity (it cannot oscillate). Now, since we proved that A^k is upper bounded and lower bounded by some value, it has to converge to a point. Let that point be A^*.

We have from Eq. 6.27:

$$A^* = \tfrac{A^* + F(A^*) + D}{2}$$

$$2A^* \leq A^* + x^* + D + D$$

$$A^* \leq x^* + 2D$$

In the above equation, we used the fact that F is upper bounded by $x^* + D$.

Thus, the fixed point for the sequence A^k is upper bounded by $x^* + 2D$.

Using a completely symmetrical argument, we can prove that the fixed point of L^k is lower bounded by $x^* - 2D$.

□

Lemma 34.

$$x_i(t) \leq X_i(t) + W_i(t; t'_k), \forall t \geq t'_k$$

Proof. Refer to Lemma 6 in Tsitsiklis [276] for proof. □

Lemma 35. *We have $x_i(t) \leq A_i^{k+1}$, for all i and $t \geq t''_k$*

Proof. Eq. 6.32 and the relation $X_i(t'_k) = A_i^k$ makes the process X_i a convex combination of A_i^k and $F_i(A^k)$. The coefficient of A_i^k is equal to $\Pi_{\tau=t'_k}^{t-1}(1-\alpha(\tau))$, whose maximum value is $\frac{1}{4}$. It follows that

$$X_i(t) \leq \tfrac{1}{4}A_i^k + \tfrac{3}{4}F_i(A^k) = \tfrac{1}{2}A_i^k + \tfrac{1}{2}F_i(A^k) + \tfrac{D}{2} - \tfrac{1}{4}(A_i^k - F_i(A^k)) - \tfrac{D}{2}$$

$$X_i(t) \leq \tfrac{1}{2}A_i^k + \tfrac{1}{2}F_i(A^k) + \tfrac{D}{2} - \tfrac{1}{4}(A_i^k - F_i(A^k) + 2D)$$

$$X_i(t) \leq A_i^{k+1} - \delta_k$$

$$X_i(t) \leq A_i^{k+1} - W_i(t; t'_k)$$

Now, using the result in Lemma 34,

$$x_i(t) \leq X_i(t) + W_i(t; t'_k)$$

$$x_i(t) \leq A_i^{k+1} - W_i(t; t'_k) + W_i(t; t'_k)$$

$$x_i(t) \leq A_i^{k+1}$$

This implies that $x_i(t) \leq A_i^{k+1}$ for all $t \geq t''_k$.

□

D.2 Experimental Details

In this section, we provide more details regarding the game domains and hyperparameters of our algorithms.

D.2.1 MAgent Games

The state space in our MAgent games do not have layers containing spatial information, as in the default state space in MAgents. We modify the state space to contain complete information (position, health, group information) about the viewable agents along with the individual agent features (already available in MAgent). All the agents at a distance of the view range from the central agent are visible in the FOR setting and for the PDO domains the viewable agents are sampled from the Bernoulli distribution, in all the games. Additionally, for the Battle-Gathering game, the position of all the food in the environment is available to all agents in their observation of the state. For all the games, we assume that at any given time step any agent cannot process more than 20 other agents and hence at most the 20 closest agents are considered. The action space for all the games includes move and attack actions. There are a total of 21 actions, with 13 move actions (move to one of the 13 cells in a unit circle, refer to [317] for more details) and 8 attack actions (attack one of the 8 nearest cells in a unit circle).

Note that training is decentralized at the group level. Agents are independent in terms of the information that they act on: if an agent were to try to account for the group's full observation space, they would quickly become overwhelmed in this setting. Agents do not use separate neural networks for each agent as the complexity would be linear in the number of agents. Instead, agents in a group train on, and use, a single neural network. This is consistent with Yang et al.'s battle games [303] and the MAgent baselines [317].

In the faceoff experiments for the Multibattle and the Battle-Gathering games, for every 1000 game set, group A from the first algorithm and group B trained using the second algorithm fight against each other for the first 500 games and group B from the first algorithm and group A from the second algorithm fights for the second 500 games. The team that has the highest number of agents alive at the end of a game (500 steps) wins the battle. If both teams have the same number of agents alive, then the team that gets the highest reward at the end is the winner. For the Predator-Prey game, the faceoff contest is conducted similar to the Multibattle game, but with a small change in how the winners are determined. The winner is the group with the most agents alive at the end of the game. If two groups have the same agents alive, then the game is considered to be a draw. Since we start with more prey (40) than predators (20), we have a fair contest in the faceoff as the predators have to kill many more prey to win the game and the prey have to attempt to escape the predators.

The reward function for the Multibattle domain gives every agent a -0.005 reward for each step and a -0.1 for attacking an empty grid (a needless attack). The agents get a +200 for killing opponents and a +0.2 for successfully attacking. Each agent has a health of 10 which must be exhausted by damage before the agent dies. All the agents are of size 1 unit width and 1 unit height and have a speed of 2 units per turn. In the Battle-Gathering domain, agents get a +80 for capturing food and a +5 for killing opponents with all other rewards being similar to the Multibattle domain. For both the Multibattle and Battle-Gathering domains, the view range is 6 units. In the Predator-Prey domain, the predators and prey have different reward functions. The predators have sizes of width 2 units and height 2 units with a maximum health point of 10 units. They have a speed of 2 units. The prey have sizes of width 1 unit and height 1 unit and a maximum health point of 2 units. The speed of prey is 2.5 units. The speed determines the size of the circle where the cells containing a valid move direction lies (refer to [317] for more details). The view range of predators are 7 units and the view range of prey are 6 units. The predators get a +1 for attacking prey and the prey get a -1 for being attacked. The predators receive a reward of +100 for killing prey. The predators get a -0.3 for making a needless attack. The prey get a -0.5 for dying (dead penalty). The reward function for all the three domains are same for the FOR and PDO settings. The rewards in all the MAgent games is per agent. That is, each agent gets an individual reward based on its actions in the environment. However, we sum all the rewards obtained by the agents in a team for our training plots in the Chapter 6.

Across our three domains there is more useful information available in the local observation, which makes it easier for algorithms that do not model partial information. Unlike Multibattle, Battle-Gathering has food available in the local observation that agents can capture. Predator-Prey has more agents (60) with distinct roles (predators must only attack and prey must escape) that makes modelling partial information less important. Consequently, as seen in the experimental results, the performance gain in using POMFQ instead of MFQ is maximized in Multibattle, and decreases for Battle-Gathering and Predator-Prey. Yet, POMFQ beats MFQ in all games. This can be observed across the two settings (FOR and PDO) and in both the train and test experiments. IL loses out in Battle-Gathering compared to POMFQ as the independent strategy finds it difficult to balance the twin goals (capturing food and killing opponents).

D.2.2 Hyperparameters

The hyperparameters of IL, RIL, MFQ, RMFQ and POMFQ are almost the same. The learning rate is $\alpha = 10^{-4}$ and the exploration rate β decays from 1 to 0 linearly during the 2000 (or 3000 as the case may be) rounds of training. The discount factor γ is 0.95, the size of replay buffer is 2^{10}, and the mini batch size is 64. POMFQ always takes 100 samples for all sampling steps in both the algorithms. The recurrent baselines (RIL and RMFQ) contains a GRU (gated recurrent unit) layer in addition to the fully connected layers. We take 100 samples in all sampling steps (i.e. from Dirichlet and Gamma distributions).

MFAC has the same learning rate and batch size as the other three algorithms, the temperature of soft-max layer of actor is $\tau = 0.1$, the coefficient of entropy in the total loss is 0.08, and the coefficient of value in the total loss is 0.1.

Most hyperparameters are same as those maintained by Yang et al. [303] in their battle game experiments.

Each training of 2000 rounds takes a wall clock time of 18 – 24 hours to complete on a virtual machine with 2 GPUs and 50 GB of memory. The test experiments take approximately 12 hours to complete 1000 rounds on a similar virtual machine.

Appendix E

Appendix For Chapter 7

E.1 Proof Of Lemmas In Chapter 7

Lemma 36. *For an agent $j \in \{1, \dots, N\}$ and policy π^j, given mean field $\boldsymbol{\mu}$, there exists a Markov policy $\hat{\pi}^j \in \Pi^j_M$ such that*

$$\nu^{\hat{\pi}^j}(s^j, a^j|\mu) = \nu^{\pi^j}(s^j, a^j|\mu) \tag{E.1}$$

Proof. For a mean field $\boldsymbol{\mu}$, we define an occupancy measure over the state space $\tilde{\nu}^{\pi^j}(s^j|\mu) \triangleq \int_{a^j} \nu^{\pi^j}(s^j, a^j|\mu)$.

Now consider a Markov policy $\hat{\pi}^j$ as follows,

$$\begin{aligned} &\hat{\pi}^j(a^j|s^j, \mu) = \tfrac{\nu^{\pi^j}(s^j, a^j|\mu)}{\tilde{\nu}^{\pi^j}(s^j|\mu)}, \quad \text{if } \tilde{\nu}^{\pi^j}(s^j, \mu) \neq 0 \\ &\text{and} \\ &\hat{\pi}^j(a^j|s^j, \mu) = \pi^j_0(a^j), \quad \text{if } \tilde{\nu}^{\pi^j}(s^j, \mu) = 0 \end{aligned} \tag{E.2}$$

Here, $\pi^j_0(a^j) \in \Pi^j_M$ is an arbitrary policy.

Consider,

$$\tilde{\nu}^{\hat{\pi}^j}(s^j|\mu) = \int_{a^j} \nu^{\hat{\pi}^j}(s^j, a^j|\mu)$$

$$= \int_{a^j} \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j, a_t^j = a^j|\mu_t = \mu)$$

$$\stackrel{1}{=} \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu)$$

$$= \nu_0(s^j) + \textstyle\sum_{t=1}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu)$$

$$= \nu_0(s^j) + \beta \textstyle\sum_{t=1}^{\infty} \beta^{t-1} \displaystyle\int_{s'^j \in \mathcal{S}} \int_{a'^j \in \mathcal{A}^j} \mathcal{P}^{\hat{\pi}^j}[s_{t-1}^j = s'^j, a_{t-1}^j = a'^j|\mu_{t-1} = \mu'] p(s^j|s'^j, a'^j, \mu')$$

$$= \nu_0(s^j) + \beta \textstyle\sum_{t=1}^{\infty} \beta^{t-1} \displaystyle\int_{s'^j \in \mathcal{S}} \mathcal{P}^{\hat{\pi}^j}[s_{t-1} = s'|\mu_{t-1} = \mu'] \int_{a'^j \in \mathcal{A}^j} \hat{\pi}^j(a'^j|s'^j, \mu') p(s^j|s'^j, a'^j, \mu')$$

$$= \nu_0(s^j)$$

$$+\beta \int_{s'^j \in \mathcal{S}} \textstyle\sum_{t=1}^{\infty} \beta^{t-1} \mathcal{P}^{\hat{\pi}^j}[s_{t-1}^j = s'^j|\mu_{t-1} = \mu'] \displaystyle\int_{a'^j \in \mathcal{A}^j} \hat{\pi}^j(a'^j|s'^j, \mu') p(s^j|s'^j, a'^j, \mu_{t-1} = \mu')$$

$$\stackrel{2}{=} \nu_0(s^j) + \beta \int_{s'^j \in \mathcal{S}} \tilde{\nu}^{\hat{\pi}^j}(s'^j|\mu') \int_{a'^j \in \mathcal{A}^j} \hat{\pi}^j(a'^j|s'^j, \mu') p(s^j|s'^j, a'^j, \mu') \tag{E.3}$$

Here $\nu_0(s^j)$ is the initial distribution of the state of the agent j. To obtain expression (2), see that this follows from expression (1).

Now consider,

$$\begin{aligned}
&\tilde{\nu}^{\pi^j}(s^j|\mu) = \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\pi^j}(s_t^j = s^j|\mu_t = \mu)\\
&= \nu_0(s^j) + \textstyle\sum_{t=1}^{\infty} \beta^t \mathcal{P}^{\pi^j}(s_t^j = s^j|\mu_t = \mu)\\
&= \nu_0(s^j) + \beta \textstyle\sum_{t=1}^{\infty} \beta^{t-1} \displaystyle\int_{s'^j \in \mathcal{S}} \int_{a'^j \in \mathcal{A}^j} \mathcal{P}^{\pi^j}[s_{t-1}^j = s'^j, a_{t-1}^j = a'^j|\mu_{t-1} = \mu'] p(s^j|s'^j, a'^j, \mu')\\
&= \nu_0(s^j) + \beta \int_{s'^j \in \mathcal{S}} \int_{a'^j \in \mathcal{A}^j} \textstyle\sum_{t=1}^{\infty} \beta^{t-1} \mathcal{P}^{\pi^j}[s_{t-1}^j = s'^j, a_{t-1}^j = a'^j|\mu_{t-1} = \mu'] p(s^j|s'^j, a'^j, \mu')\\
&\overset{1}{=} \nu_0(s^j) + \beta \int_{s'^j \in \mathcal{S}} \int_{a'^j \in \mathcal{A}^j} \nu^{\pi^j}(s'^j, a'^j|\mu') p(s^j|s'^j, a'^j, \mu')\\
&\overset{2}{=} \nu_0(s^j) + \beta \int_{s'^j \in \mathcal{S}} \int_{a'^j \in \mathcal{A}^j} \hat{\pi}^j(a'^j|s'^j, \mu') \tilde{\nu}^{\pi^j}(s'^j|\mu') p(s^j|s'^j, a'^j, \mu')\\
&= \nu_0(s^j) + \beta \int_{s'^j \in \mathcal{S}} \tilde{\nu}^{\pi^j}(s'^j|\mu') \int_{a'^j \in \mathcal{A}^j} \hat{\pi}^j(a'^j|s'^j, \mu') p(s^j|s'^j, a'^j, \mu')
\end{aligned} \tag{E.4}$$

The (1) is from Eq. 7.8 and (2) is from Eq. E.2. From both Eq. E.4 and Eq. E.3, we find that both the discounted state occupation frequency can be recursively expressed as a term depending on the state occupation frequency at the previous time step and a few other terms that are the same for both the Eq. E.3 and Eq. E.4. Since the initial state distribution is the same for both policies, we can conclude that $\tilde{\nu}^{\hat{\pi}^j}(s^j|\mu) = \tilde{\nu}^{\pi^j}(s^j|\mu)$.

Now, consider

$$\begin{aligned}
&\nu^{\hat{\pi}^j}(s^j, a^j|\mu) = \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j, a_t^j = a^j|\mu_t = \mu)\\
&= \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(a_t^j = a^j|s_t^j = s^j, \mu_t = \mu) \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu)\\
&= \textstyle\sum_{t=0}^{\infty} \beta^t \hat{\pi}^j(a_t^j = a^j|s_t^j = s^j, \mu_t = \mu) \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu)
\end{aligned} \tag{E.5}$$

Also, from Eq. E.2, we have

$$\begin{aligned} &\nu^{\pi^j}(s^j, a^j|\mu) = \hat{\pi}^j(a^j|s^j,\mu) \times \tilde{\nu}^{\pi^j}(s^j|\mu) \\ &= \hat{\pi}^j(a^j|s^j,\mu) \times \tilde{\nu}^{\hat{\pi}^j}(s^j|\mu) \\ &= \hat{\pi}^j(a^j|s^j,\mu) \textstyle\sum_{t=0}^{\infty} \beta^t \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu) \\ &= \textstyle\sum_{t=0}^{\infty} \beta^t \hat{\pi}^j(a^j|s^j,\mu) \mathcal{P}^{\hat{\pi}^j}(s_t^j = s^j|\mu_t = \mu) \end{aligned} \tag{E.6}$$

From Eq. E.5 and Eq. E.6 we find that $\nu^{\hat{\pi}^j}(s^j, a^j|\mu) = \nu^{\pi^j}(s^j, a^j|\mu)$, due to the property of the Markov policy $\hat{\pi}^j$.

□

Lemma 37. *For all $t \geq 0$ and a given mean field $\boldsymbol{\mu}$, the operator $T_t^{\boldsymbol{\mu}}$ maps $C_w^{t+1}(\mathcal{S})$ into $C_w^t(\mathcal{S})$. Also, this operator will satisfy*

$$||T_t^{\boldsymbol{\mu}} u - T_t^{\boldsymbol{\mu}} x||_w \leq \alpha\beta ||u - x||_w \tag{E.7}$$

for any $u, x \in C_w(\mathcal{S})$.

Proof. Let $u \in C_w^{t+1}(\mathcal{S})$. We have the following relation,

$$\begin{aligned} &||T_t^{\boldsymbol{\mu}} u||_w \leq \sup_{(s_t^j, a_t^j) \in \mathcal{S} \times \mathcal{A}^j} \frac{\left| r^j(s_t^j, a_t^j, \mu_t) + \beta \displaystyle\int_S u(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \right|}{w(s_t^j)} \\ &\leq \sup_{(s_t^j, a_t^j) \in \mathcal{S} \times \mathcal{A}^j} \frac{M_t w(s_t^j) + \beta\alpha L_{t+1} w(s_t^j)}{w(s_t^j)} \\ &= M_t + \beta\alpha L_{t+1} \\ &= L_t \end{aligned} \tag{E.8}$$

The second last step is from Assumption 14 and Assumption 12. Also, we use the bound on the w-norm from Eq. 7.24. The last step is from Eq. 7.23. This proves the first statement.

To prove the second statement, without loss of generality, let us assume that $u_0 \geq x_0$. Now consider,

$$
\begin{aligned}
&||T_t^{\boldsymbol{\mu}} u - T_t^{\boldsymbol{\mu}} x||_w \\
&= \sup_{(s_t^j)\in\mathcal{S}} \frac{\max_{a_t^j} \left| r^j(s_t^j, a_t^j, \mu_t) + \beta \int_{\mathcal{S}} u(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t)}{w(s_t^j)} \frac{- r^j(s_t^j, a_t^j, \mu_t) - \beta \int_{\mathcal{S}} x(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \right|}{w(s_t^j)} \\
&= \sup_{(s_t^j)\in\mathcal{S}} \frac{\max_{a_t^j} \left| \beta \Big(\int_{\mathcal{S}} u(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t)}{w(s_t^j)} \frac{- \int_{\mathcal{S}} x(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) \Big) \right|}{w(s_t^j)} \\
&= \sup_{(s_t^j)\in\mathcal{S}} \frac{\max_{a_t^j} \left| \beta \int_{\mathcal{S}} p(s_{t+1}^j | s_t^j, a_t^j, \mu_t) (u(s_{t+1}^j) - x(s_{t+1}^j)) \right|}{w(s_t^j)} \\
&\le \sup_{(s_t^j)\in\mathcal{S}} \frac{\alpha\beta \left| (u(s_t^j) - x(s_t^j)) \right|}{w(s_t^j)} \\
&= \alpha\beta ||u - x||_w
\end{aligned}
\tag{E.9}
$$

We apply Assumption 12 and the last step is from Eq. 7.17. This proves the second part of the lemma as well.

$\square$

Lemma 38. *For any $\boldsymbol{\mu}$, the optimal point $\boldsymbol{J}_*^{j,\boldsymbol{\mu}}$, for an agent $j \in \{1, \ldots, N\}$, belongs to the Banach space $\mathcal{C}$.*

Proof. Let π be a Markov policy. At any time $t \ge 0$, we have

$$
\begin{aligned}
&J_{*,t}^{j,\boldsymbol{\mu}}(s) \\
&= \textstyle\sum_{k=t}^{\infty} \beta^{k-t} \, \mathbb{E}^{\pi}(r^j(s_k^j, a_k^j, \mu_k) | s_t^j = s^j) \\
&\le \textstyle\sum_{k=t}^{\infty} \beta^{k-t} M_k \, \mathbb{E}^{\pi}(v(s_k) | s_t^j = s^j) \\
&\le \textstyle\sum_{k=t}^{\infty} \beta^{k-t} M_k \alpha^{k-t} v(s^j) \\
&= L_t v(s^j)
\end{aligned}
\tag{E.10}
$$

The second step is from Assumption 14. The third step is from Assumption 12. Hence, the function $\boldsymbol{J}_{*,t}^{j,\boldsymbol{\mu}}$ belongs to the set $C_w^t(\mathcal{S})$ from the Eq. 7.24.

□

Lemma 39. *For an agent $j \in \{1, \dots, N\}$, for any $\boldsymbol{\nu} \in \mathcal{D}$, the collection of value functions $\boldsymbol{J}_*^{j,\boldsymbol{\nu}}$ is the unique fixed point of the operator $T^{\boldsymbol{\nu}}$. Furthermore, $\pi^j \in M$ is optimal if and only if*

$$\nu_t^{\pi^j}(\{(s_t^j, a_t^j) : r^j(s_t^j, a_t^j, \nu_{t,1}) + \beta \int_{\mathcal{S}} J_{*,t+1}^{j,\boldsymbol{\nu}}(s_{t+1}^j) p(s_{t+1}^j | s_t^j, a_t^j, \nu_{t,1}) = T_t^{\nu} J_{*,t+1}^{j,\boldsymbol{\nu}}(s_t^j)\}) = 1 \tag{E.11}$$

where $\nu_t^{\pi^j} = \mathcal{L}(s_t^j, a_t^j)$.

Proof. See Hinderer [101].

□

Lemma 40. *Suppose the set valued mapping τ has a fixed point $\boldsymbol{\nu^j} = (\nu_t^j)_{t \geq 0}$ for an agent $j \in \{1, \dots, N\}$. Consider a Markov policy for the agent j as $\pi^j = (\pi_t^j)_{t \geq 0}$, which is obtained by factoring as $\nu_t^j(s_t^j, a_t^j) = \nu_{t,1}^j(s_t^j)\pi_t^j(a_t^j | s_t^j)$, and let $\boldsymbol{\nu}_1^j = (\nu_{t,1}^j)_{t \geq 0}$. Then the pair $(\pi^j, \boldsymbol{\nu}_1^j)$ is a decentralized mean field equilibrium.*

Proof. If $\boldsymbol{\nu}^j \in \tau(\boldsymbol{\nu}^j)$, then the corresponding Markov policy π^j satisfies the Eq. 7.32 for $\boldsymbol{\nu}^j$. Thus, by Lemma 39, $\pi^j \in \Phi(\boldsymbol{\nu}_1^j)$. Further, since $\boldsymbol{\nu}^j \in C(\boldsymbol{\nu}^j)$, we have $\Psi(\pi^j) = \boldsymbol{\nu}_1^j$, which completes the proof. □

Lemma 41. *Using Assumptions 11—15, the graph of τ, i.e. the set*

$$Gr(\tau) \triangleq \{(\boldsymbol{\nu}, \boldsymbol{\mathcal{E}}) \in \mathcal{D} \times \mathcal{D} : \boldsymbol{\mathcal{E}} \in \tau(\boldsymbol{\nu})\} \tag{E.12}$$

is closed.

Proof. See Proposition 3.9 in Saldi et al. [211] for complete proof. □

Lemma 42. *Using Assumptions 11—15, for an agent $j \in \{1, \dots, N\}$, there exists a fixed point $\boldsymbol{\nu}^j$ of the set valued mapping $\tau : \mathcal{D} \to 2^{\mathcal{D}}$. Then, the pair $(\pi^j, \boldsymbol{\nu}_1^j)$ is a decentralized mean field equilibrium, where π^j is the policy of agent j and $\boldsymbol{\nu}_1^j$ is its mean field estimate constructed according to Lemma 40.*

Proof. Using the Lemma 41 and our previous results, we have proved that $\mathcal{D}$ is non-empty, compact and convex. Further, the set valued mapping τ has a closed graph, is non-empty and convex. Hence, by Kakutani's fixed point theorem [121], τ has a fixed point. Thus, from Lemma 40 this fixed point is the decentralized mean field equilibrium. □

E.2 Experimental Details

In this section, we describe each of our game domains in detail, including the reward functions. We also provide the implementation details of our algorithms, especially the hyperparameters. We also discuss the wall-clock times of our algorithmic implementations. Each episode for the Petting Zoo environments has a maximum of 500 steps. In our implementation of MFQ and MFAC, the agents learn in a decentralized fashion with independent networks and use the previous mean field of the local neighbourhood instead of the global mean field.

The first 5 domains are obtained from the Petting Zoo environment [269], and the game parameters are mostly left unchanged from those given in Terry et al. [269]. In these games, as agents can die during game play, we use the agent networks saved in the last available episode during training for the execution experiments. Complete details of these domains are given in each of the sub-sections below.

E.2.1 Battle Domain

This is the first domain from the Petting Zoo environment [269] that is mixed cooperative-competitive. This domain was originally defined in the MAgents simulator [317]. We have two teams of 25 agents, each learning to cooperate against the members of the same team and compete against the members of its opponent team. The agent gets rewarded for attacking and killing agents of the opposing team. At the same time, the agent is penalized for being attacked. In our implementation, each agent learns using its local observation and cannot get global information. Each agent has a view range of a circular radius of 6 units around it. Most rewards are left as defaults, as given in Terry et al. [269]. The agents get a penalty of -0.005 for each step (step reward) and a penalty of -0.1 for being killed. The agents get a reward of 5 for attacking an opponent and a reward of 10 for killing an opponent. There is an attack penalty of -0.1 (penalty for attacking anything). Agents start with a specific number of hitpoints (HP) that are damaged upon being attacked. Agents lose 2 HP when attacked and recover an HP of 0.1 for every step that they are not attacked. They start with 10 HP, and when this HP is completely lost, the agent dies. Each agent can view a circular range of 6 units around it (view range) and can attack in a circular range of 1.5 units around it (attack range). The action space of each agent is a discrete set of 21 values, which corresponds to taking moving and attacking actions in the environment. In this game, we assume that agents can get perfect information about actions of other agents at a distance of 6 units from themselves and no information beyond this point. In the execution phase, a game is considered to be won by a team that kills more opponent agents. If both teams kill the same number of agents, the team having the higher cumulative reward is determined as the winner.

E.2.2 Gather Domain

In this environment, all agents try to capture limited food available in the environment and gain rewards. Each food needs to be attacked before it can be captured (takes 5 attacks to capture food). This is a fully competitive game, where all agents try to outcompete others in the environment and gain more food for themselves. Agents can also kill other agents in the environment by attacking them (just a single attack). Each agent gets a step penalty of -0.01, an attack penalty (penalty for attack) of -2 and a death penalty (punishment for dying) of -20. Also, each agent gets a reward of 20 for attacking food and a reward of 60 for capturing the food. There are a total of 30 agents learning in our environment. The action space of each agent is a set of 33 values. The agents have a view range of 7 and an attack range of 1. All other conditions and rewards are the same as the Battle game.

E.2.3 Combined Arms Domain

The Combined Arms environment is a heterogeneous large-scale team game with two types of agents in each team. The first type is a ranged agent, which can move fast and attack agents situated further away but has fewer HP. The second type is the melee agent that can only attack close-by agents and move slowly; however, they have more HP. The reward function for the agents is the same as that given in the battle game. The action space of the melee agents is a discrete set of 9 values, while the action space of the ranged agents is a discrete set of 25 values. The actions correspond to moving in the environment and attacking neighbouring agents. The ranged agents have a maximum HP of 3, and the melee agents have a maximum HP of 10. The maximum HP is the limit after which the HP of any agent cannot increase in this environment. Like the battle domain, agents lose 2 HP for each time they are attacked and gain 0.1 HP for each step without being attacked. In our experiments, each team consists of 25 agents, with 15 ranged and 10 melee agents at the start. The view range is 6 and the attack range is 1 for melee agents. The view range is 6 and attack range is 2 for ranged agents. All other conditions and rewards are the same as the Battle game. To create the mean field we choose to use the action space of the type which has the larger number of actions (i.e. we use the action space of the ranged agents).

E.2.4 Tiger-Deer Domain

In the tiger-deer environment, tigers are the learning agents, cooperating with each other to kill deer in the environment. At least two tigers need to attack a deer together to get high rewards. The tigers start with an HP of 10 and gain an HP of 8 whenever they kill deer. The tigers lose an HP of 0.021 at each step they do not eat a deer and die when they lose all the HP. In this game, the deer move randomly in the environment, and start with an HP of 5 and lose 1 HP upon attack. The tiger gets a reward of +1 for attacking a deer alongside another tiger. In this game, each tiger is assumed to get perfect information about the actions of other tigers at a distance of 4 units from itself and no information beyond this point. The view range of the tiger is 4 and the attack range is 1. The tigers also get a shaping reward of 0.2 for attacking a deer alone. All other rewards and conditions are the same as the Battle game.

E.2.5 Waterworld Domain

The Waterworld domain was first introduced as a part of the Stanford Intelligent Systems Laboratory (SISL) environments by Gupta et al. [89]. We use the same domain adapted by the Petting Zoo environment [269]. This is a continuous action space environment, where a group of pursuer agents aim to capture food and avoid poison. These pursuers are the learning agents, while both food and poison move randomly in the environment. This is a cooperative environment where pursuer agents need to work together to capture food. At least two agents need to attack a food particle together to be able to capture it. The action is a two-element vector, where the first element corresponds to horizontal thrust and the second element corresponds to vertical thrust. The agents choose to use a thrust to make themselves move in a particular direction with a desired speed. The action values are in the range [-1, +1]. Our domain contains 25 pursuer agents, 25 food particles and 10 poison particles. The food is not destroyed but respawned upon capture. The agents get a reward of +10 upon capturing food and a penalty of -1 for encountering poison. If a single agent encounters food alone, it gets a shaping reward of +1. Each time an agent applies thrust, the agent obtains a penalty of thrust penalty $\times$ ||action||, where the value of thrust penalty is -0.5. Since this is a cooperative environment, each agent is allowed access to the global mean field action which is composed of actions of all agents in the environment, at each time step. This is done to simplify this complex domain.

E.2.6 Ride-Sharing Domain

In this domain, our problem formulation and environment are the same as that described in Shah et al. [217]. The demand distribution is obtained from the publicly available New York Yellow Taxi Dataset [181]. The overall approach follows six steps. First, the user requests are obtained from the dataset. Second, sets of feasible trips are generated (by an oracle) using the approach in Javier et al. [6]. These feasible trips keep the action space from exploding. Third, the feasible actions are scored by the individual agents using their respective value functions. Fourth, a mapping of requests takes place by checking different constraints. Fifth, the final mapping is used to update the rewards for the individual agents. Sixth, the motion of vehicles is simulated until the next decision epoch.

We consider a maximum of 120 vehicles in our experiments. Since we are learning in a decentralized fashion, each vehicle maintains its own network and is computationally intensive. However, in practical applications, the training can be parallelized across agents and the computational demands will not be a limitation of our proposed setting.

Our goal in this experiment is to implement the mean field algorithm on a real-world problem and compare the performance to other state-of-the-art approaches. The experimental setup is along the lines of [6, 217], where we restrict ourselves to the street networks of Manhattan, where the vast majority of requests are contained. The New York Yellow Taxi dataset contains information about ride requests at different times of the day during a given week. Similar to Shah et al. [217], we use the pickup and drop-off locations, pickup times and travel times from the dataset. The dataset has about 330,000 requests per day. In our experiments, the taxis are assigned a random location at the start and react to incoming ride requests. We train the networks using data pertaining to 8 weekdays and validate it on a single day as done in Shah et al. [217]. We assume all vehicles have similar capacities for simplicity, although our decentralized set-up is completely applicable to environments with very different types of vehicles as against prior work that relied on centralized training [155, 217].

In the experiments, a single day of training corresponds to one episode. Each episode has 1440 steps, where each step (decision unit) is considered to span one minute. The initial location of vehicles at the beginning of an episode (single day of training) is random. The state, action space and reward function in our system is the same as that in Shah et al. [217]. The state of the system can be described as a tuple (c_t, u_t), where c_t is the location of each vehicle j denoted by $c_t^j = (p^j, t, L^j)$ representing its trajectory. This captures the current location and a list of future locations that each vehicle visits. The user request i at the time t is represented as $u_t^i = (o^i, e^i)$ which corresponds to the origin and destination of the requests. The action for each agent is to provide a score for all the requests with the objective of assigning the user requests to itself. The user request should satisfy the constraints at the vehicle level (maximum allowed wait time and capacity limits) and the constraints at the system level (each request is only assigned to a single agent). Since these constraints are environmental, we continue to use a central agent that performs the constraints check before assigning requests to individual agents. Each agent gets a reward based on the proportion of requests in its feasible action set that it is able to satisfy, as done in Shah et al. [217].

E.3 Hyperparameters and Implementation Details

The implementation of DMFG-QL, IL and MFQ almost use the same hyperparameters, with the learning rate set as $\alpha = 10^{-2}$. The temperature for the Boltzmann policy is set as 0.1. Additionally, we also conduct epsilon greedy exploration which is decayed from 20% to 1% during the training process. The discount factor γ is equal to 0.9. The replay buffer size is 2×10^5, and the agents use a mini-batch size of 64. The target network is updated at the end of every episode.

Our implementations of DMFG-AC and MFAC almost use the same hyperparameters, where the learning rate of the critic is 10^{-2} and the learning rate of the actor is 10^{-4}. The mean field network of the DMFG-AC uses a learning rate of 10^{-2}. The discount factor is the same as the other three algorithms. In our implementation of MFAC, we do not use replay buffers unlike the implementation of Yang et al. [303]. We found this version of the actor-critic algorithm using the current updates (instead of delayed updates through the replay buffer) is more stable and performs better than the implementation of Yang et al. [303] in our experiments. Additionally, our implementations of both MFAC and DMFG-AC use complete decentralization during execution where the actors only need to use their local states (mean field does not need to be maintained anymore). Also since a separate network is being maintained for the stochastic policy (actor) we do not use the Boltzmann policy for the actor-critic based methods (MFAC and DMFG-AC).

For the continuous action space Waterworld environment, every component of the obtained estimated mean field is normalized to be in the range $[-1, 1]$ (the range of the action values) and we do not use a softmax for the output layer (since this a mixture of Dirac deltas as discussed in Section 7.9).

Most hyperparameters are the same or closely match those used by prior works [303, 248, 88, 252] in many agent environments.

For the ride-sharing experiments, DMFG-QL uses the same network architecture as given in [217] for the NeurADP algorithm. The implementation of CO and NeurADP uses the implementation provided by [217] except that all agents are fully decentralized as mentioned before. Unlike the approach in [217], each of our agents train their independent neural network using their local experiences. This network learns a suitable value function that can assign an appropriate score to each of the ride requests. This is done for both our implementations of NeurADP and DMFG-QL. For our baseline of CO [6], we used the implementation from Shah et al. [217], which used the immediate rewards as the score for the given requests along with a small bonus term pertaining to the time taken to process a request (faster processing of requests is encouraged). The mean field for the DMFG-QL implementation is obtained by processing a distribution of the ride requests across every node in the environment at each step. This mean field is made available to all agents during both training and testing. Also, we use a slightly different architecture for estimating the mean field in this domain (3 Relu layers of 50 nodes and an output softmax layer).

The DDPG hyperparameters are based on Lillicrap et al. [147] and the PPO hyperparameters are based on Schulman et al. [216]. DDPG uses the learning rate of the actor as 0.001 and that of the critic as 0.002 and a discount factor of 0.9. We use the soft replacement strategy with learning rate 0.01. The batch size is 32. The PPO implementation uses the same batch size and discount factor. The actor learning rate is 0.0001 and critic learning rate is 0.0002. Independent PPO uses a single thread implementation, since the data correlations are already being broken by the non-stationary nature of the environment induced by the multiple agents. This is also computationally efficient.

We use a set of 30 random seeds (1 – 30) for all training experiments and a new set of 30 random seeds (31 – 60) for all execution experiments.

E.4 Wall Clock Times

The training for all the experiments on the simulated Petting Zoo domains was run on a 2 GPU virtual machine with 16 GB GPU memory per GPU. We use Nvidia Volta-100 (V100) GPUs for all these experiments. The CPUs use Skylake as the processor microarchitecture. We have a CPU memory of 178 GB. The Battle, Combined Arms and Gather experiments take an average of 5 days wall clock time to complete for all the considered algorithms. The Tiger-Deer experiments take an average of 4 days wall clock time to complete and the Waterworld experiments take an average of 2 days wall clock time to complete.

The majority of our experiments on the RMP problem were run on a virtual machine with 4 Ampere-100 GPUs with a GPU memory of 40 GB each. The CPUs use Skylake as the processor microarchitecture. Each training takes an average of 5 days to complete execution.

Appendix F

Appendix For Chapter 8

F.1 Experimental Details

In this work, we used two experimental testbeds — MAgents and Neural MMO. More details about each of these domains are given in this section.

F.1.1 MAgents

We consider three domains in the MAgents testbed. These are the Battle, Tiger and the Combined Arms environments.

The Battle domain is a mixed cooperative-competitive domain that contains two teams of agents, where an agent is trying to kill the members of the opponent team by cooperating with the members of the same team. In this game, each agent starts with a maximum horse power (HP) of 10 units. The HP denotes the energy of the agent. This HP gets depleted upon being attacked by opponents. The HP is gradually replenished over time, and the agent dies when the HP becomes 0. Each agent loses 2 HP upon being attacked by an opponent and is able to recover 0.1 HP at every step. Each agent gets a small penalty of -0.005 at every step, a punishment of -0.1 for dying (dead penalty), and a punishment of -0.01 for a wrong attack (attacking agents of the same team or an empty grid). An agent's attack on another agent in its own team is not registered. The agents gain a reward of +0.5 upon attacking opponents and a reward of +10 upon killing an opponent. Each episode in this setting constitutes a full battle of a maximum of 1000 action steps (or cycles). The game terminates if either all the steps are done or if one of the two teams completely dies.

The observation space of each agent is a 13×13 map around the agent where the agent gets full observation (and no observation outside this radius). The entire state space is a grid of dimensions ranging from 40×40 (for environments with 64 agents in each team) to 60×60 (for environments with 144 agents in each team). Each agent can perform a total of 21 discrete actions, of which one action pertains to doing nothing, 12 actions pertain to moving and 8 actions pertaining to attacking another agent nearby. The conditions in this game are almost the same as the defaults in the Petting Zoo library [269].

Another domain we consider is the Combined Arms heterogeneous mixed cooperative-competitive environment [269]. The conditions of this game are similar to the Battle game, except that each team consists of 2 heterogeneous groups of agents — ranged and melee. The ranged agents can move faster and attack further, but have a lesser maximum HP of 3 units (can be killed easily as compared to melee agents). The melee agents can only attack other agents very close to itself but have more energy (more maximum HP of 10 units as compared to ranged agents). The melee agents have a total of 9 actions with consists of doing nothing, 4 attack actions and 4 move actions. The ranged agents have a total of 21 actions involving doing nothing, 12 move actions and 12 attack actions. The entire state space is a grid of dimensions 50×50. All other conditions of this game are the same as the Battle game.

Another MAgent environment we consider is the fully cooperative Tiger game. In this game, the tigers start with a maximum HP of 18 units where they lose 0.02 HP at every step they do not eat a deer. If they do eat a deer, they gain an HP of 8. When the HP becomes 0, the tiger dies. The deer have a maximum HP of 5 and lose 1 HP when attacked and regain 0.1 HP at every step. So it is better to kill a deer quickly. The tigers have an action space of 9, where there is one action to signify doing nothing, 4 attack actions and 4 move actions. The action space of the deer consists of 5 actions (includes doing nothing and 4 move actions, the deer cannot attack). The tigers receive a reward of $+2$ for attacking a deer alongside other tiger(s). At least two tigers have to attack a deer together to obtain a reward. The tigers get a penalty of -0.01 in case of a wrong attack (attacking empty grids, other tigers, or deer alone). The state is a grid of dimensions ranging from 74×74 (environment containing 54 tigers) to 100×100 (environment containing 100 tigers), with a tiger getting an observation of dimension 9×9 around itself. Same as the Battle domain, each episode in this setting constitutes a full game of a maximum of 1000 action steps (or cycles). An episode ends either if all the deer are killed or if all the steps are completed.

F.1.2 Neural MMO

Neural MMO, originally released by Suarez et al. [242], is a simulation of the popular massively multiplayer online (MMO) [238] games. This environment supports learning with many concurrently learning agents as in the MAgent platform. However, the per-agent complexity of agents in Neural MMO is much larger than that of MAgents. We consider a mixed cooperative-competitive setting where we have two groups of agents with 40 agents on each team in a 128×128 grid (medium configuration from Suarez et al. [242]). The objective of an agent is the same as the MAgent Battle game, where each agent tries to kill agents belonging to the opponent team by cooperating with agents in the same team. At the start of the game, each agent spawns randomly at any one of the grids with a maximum HP of 100. At each step, an agent loses 20 units of HP upon being attacked and gains 1 HP every step at which it is not attacked. The agent also loses different quantities of HP based on its current stock of food and water resources (see Suarez et al. [242] for more details). The agent dies when the HP drops to 0 or lower. The observation of each agent contains two components, where the first component contains entity information (i.e., information about the agent itself including previous action and HP) and the second component includes a local view of 15×15 around an agent. The action space is a discrete set of 108 actions, where 103 actions pertain to attack (including direction and style of attack) and 5 actions pertain to moving (into one of the 4 cardinal directions and doing nothing), see Suarez et al. [242] for more details. Agents get a reward of $+150$ upon killing an opponent and $+5$ for attacking an opponent. The agents lose -5 for dying and -2 for a wrong attack. All other conditions are the same as the defaults in Suarez et al. [242].

F.2 Hyper-parameter and Implementation Details

Most hyper-parameters are the same or closely match prior works in mean field learning [303]. We make small changes for performance and computational efficiency reasons.

For the MAgent environment experiments, the Q-learning based methods (IL, MFQ, MFA-QL, MFAA-QL, MTMFA-QL MTMFAA-QL) almost use the same hyper-parameters, where the learning rate is $\alpha = 10^{-4}$, with an ϵ-greedy exploration rate that decays linearly, starting from 20% to 1% at the end of training. The discount factor $\gamma = 0.95$, and we use a mini-batch size of 64. The replay buffer memory size is 2×10^4. IL and MFQ use an evaluation and target network that has 3 fully connected layers (2 ReLU layers of 50 neurons and an output layer). The target network is replaced after every 10 learning iterations using the hard replacement strategy. The architecture of MFAA-QL and MTMFAA-QL are as specified in Section 8.4 and Section 8.5 respectively. The temperature for Boltzmann's policy is set to 0.1.

For the MAgent environment experiments, the actor-critic based algorithms (MFAC, MAAC, MFAA-AC) almost use the same hyper-parameters. The learning rate is $\alpha = 10^{-4}$ and the temperature for Boltzmann's policy is 0.1. MFAC uses a critic and actor network with an architecture of 3 fully connected layers (2 ReLU layers of 50 neurons and an output layer). The actor networks of MFAA and MFAA-AC also use the same configuration. The critic of MFAA uses the configuration given in Shariq and Fei [110]. The critic of MFAA-AC uses the configuration given in Section 8.4. The target networks are replaced every 10 learning iterations using the hard replacement strategy.

For the Neural MMO experiments, all hyper-parameters and network architectures are the same as mentioned for the MAgent experiments. The exceptions are: 1) the neural network configurations used in all algorithms contain one additional fully-connected layer, and 2) all network layers contain 200 neurons.

All algorithms that use the attention mechanism (MAAC, MFA-QL, MFAA-AC, MFAA-QL, MTMFA-QL, MTMFAA-QL) use a total of four attention heads in all of our environments. Leaky ReLU is used as the activation function in all of these networks involving the attention mechanism.

F.3 Wall Clock Times

The majority of training for all the experiments on the MAgent domains was run on a 4 GPU virtual machine with 40 GB GPU memory per GPU. We use Nvidia Ampere-100 (A100) GPUs for all these experiments. The CPUs use AMD Milan 7413 as the microprocessor architecture. We have a CPU memory of 200 GB. The Battle and Combined Arms experiments took an average of 5 days wall clock time to complete for all the considered algorithms. The Tiger experiments took an average of 3 days wall clock time to complete.

The majority of training for the Neural MMO domain was done on a virtual machine of a similar configuration, except that we use a CPU memory of 300 GB. The algorithms took an average of 7 days wall clock time to complete.