THESIS FOR THE DEGREE OF DOCTOR IN PHILOSOPHY

MACHINE LEARNING

IMPROVED REPRESENTATIONS FOR COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

JACOPO CASTELLINI

University of Liverpool

SUPERVISED BY
PROF. FRANS A. OLIEHOEK,
PROF. RAHUL SAVANI

JUNE 20, 2022

This thesis is primarily my own work. The sources of other materials are identified. This work has not been submitted for any other degree or professional qualification except as specified.
This document was typeset using the typographical look-and-feel `smart-thesis` developed by Jan Philip Göpfert and Andreas Stöckel. `smart-thesis` is available here:
`https://github.com/astoeckel/smart-thesis`.

It's a dangerous business, Frodo, going out your door. You step onto the road, and if you don't keep your feet, there's no knowing where you might be swept off to.

— JOHN R.R. TOLKIEN, THE LORD OF THE RINGS - THE FELLOWSHIP OF THE RING

ACKNOWLEDGEMENTS

Taking on a PhD is an amazing but also very difficult journey. You are faced with the great challenge of learning how to investigate and possibly push a bit forward the limits of human knowledge, and, although it is an extremely exciting and stimulating adventure, there may be moments in which you feel daunted or even overwhelmed by the task. Personally, I have been lucky enough to not be alone during this emotional roller-coaster, and count on a multitude of amazing people that always supported (and sometimes withstood) me. All of my biggest and sincerest gratitude goes to them all.

I could start with no one else than my two supervisors, **Frans** and **Rahul**. They did not only teach me how to be a researcher, both intellectually and personally, but also have been the first and foremost source of support and help for whatever kind of issue I had during these years. I could not even imagine being the person I am now without the two of you, and I will never stop to thank you for this. Also, a huge thanks to all the people from the smARTLab (or gravitating around it): **Shan**, **James**, **Daniel**, **Greg**, **Tom**, **Guan**, **Jiaqi** and all of the others that have been part of my University life, thanks for having been amazing colleagues and friends. I would also like to mention **Benjamin**, a brilliant colleague that tragically passed away some years ago: you have been the first to crack my shell when I was new and and felt alone in the lab. I will never forget you. A special thank goes to **Sam Devlin** from Microsoft Research Cambridge: working with you has been a huge pleasure, and with your kindness and help you taught me a lot on how to do research and be a scientist.

I met a lot of other people outside the University that made my time in Liverpool the great time it has been: **Tomaso**, **Alessandro**, **Jonathan**, **Andrea**, **Carmelo**, **Luca**, **Salvo**, **Alberto**, **Michael**, **Shah**, **Matilde** and all of the others, thanks for having been the perfect fellowship to countless memories and moments. This whole experience would not have been the same without you. A special mention goes to my girlfriend **Laura**: you have been an amazing friend and flatmate before, and one of the most important persons in my life now. Very few others know more than her how this whole adventure has been for me: I cannot even count the hours spent talking about this or that, and you always provided me your with undisputed attention and help (even when you were not understanding a single word of what I was talking about). For this I am extremely grateful to you, I could not have asked for more.

Being a student abroad is exciting, but it is also difficult to relocate and adapt to a completely new reality without seeing the people you love for months. I wish to thank all of my friends in Italy: **Lorenzo**, **Leonardo**, **Michael**, **Mattia**, **Gianluca**, **Giacomo**, **Leone**, **Tommaso**, **Andrea**, **Daniele**, **Flavio**, **Michele**, **Pierpaolo**, **Marco**, **Ramazan**, **Stefano**, **Francesco**, **Diego**, **Valerio**, **Lorenzo**, **Carlo** and **Stefania** just to name a few (but equal gratitude goes to all of the others that have not been explicitly named here). Every time I came back home, you made me felt as if no day passed since I left the previous time. Thanks for having always made me felt a part of your everyday life even when I was kilometres away. Roots are important, and with roots like you no storm can ever worry me.

Finally and mainly, my most sincere and biggest gratitude goes to my family: my mom **Paola**, my dad **Stefano**, my sister **Lucrezia**, my aunt **Silvia** and uncle **Simone** with their kids **Gerardo** and **Celeste**, my two grandmas **Teresa** and **Mara**. You always believed in me, even before I believed in myself. You always gave me your complete and unconditioned support for every aspect of this experience, even when you were not understanding my decisions or problems. You are the pillars that keep my life up, and every day I wake up knowing that there is someone, somewhere, that loves me. Nothing has helped me more than being aware of this. This work is above all dedicated to you.

Post Scrittum: after having passed my VIVA, it would be impossible for me to not thank the internal examiner **John Fearnley** and external **Ann Nowé**. You made the final step of this long journey an unexpectedly pleasuring one.

ABSTRACT

Multi-agent systems [33, 136] are an ubiquitous presence in our everyday life: our entire society could be seen as a huge multi-agent system in which each individual has to perform in an environment populated by other entities, each motivated by its own goals and objectives. In a cooperative system, all of these entities act towards a common goal. This setting has gained a lot of popularity in the AI research community, as many real-world situations can naturally be modelled as such [150, 177, 151, 71, 109, 23]. Still, optimally solving such systems remains a challenging problem.

Multi-agent reinforcement learning (MARL) is one of the most employed techniques used to tackle these: agents learn how to behave by repeatedly interacting with their own environment. Although major key steps have been taken in this direction, some fundamental issues arising from the presence of multiple agents that learn and act together remain. In this work, two such aspects are addressed: team representation and the multi-agent credit assignment problem.

The former is about the way in which a system designer has to represent and learn the team of agents: on one hand, representing the whole team as a single centralized entity may seem compelling, but this solution is not adaptable to scale to larger systems. On the other hand, learning each agent independently from the others solves that issue [146, 83, 24, 171], but introduces non-stationarity in the agent learning experiences due to the now ignored presence of the other agents. In this work, the focus is on factorization techniques [50, 53, 52], as a middle ground between these two extremes: this idea has recently gained a major interest and served as the basis for many recent deep MARL algorithms [139, 115, 135, 158]. Although well performing in practice, all of these methods only focused on single-agent decompositions, leaving the idea of "higher-order" factorizations almost unexplored. Moreover, although factorizations are widely considered capable of improving performances over the two approaches detailed above, no wide investigation of the real merits or general applicability of factored techniques has been conducted so far. This work fills the gap by investigating a wide array of factored methods on a diverse set of cooperative scenarios, to assess their performance both in terms of accuracy of the represented functions and action selection.

About the multi-agent credit assignment problem [20, 94, 178, 168] instead, many techniques have been proposed, including difference rewards [169, 168] (one of the most popular family of algorithms used to tackle such problems), but few have been extended to the deep MARL framework. One of such applications is Counterfactual Multi-Agent Policy Gradients (COMA) [40], that employs difference rewards to provide each agent with an individual signal to learn from. This algorithm have however proved to perform poorly in practice [160, 181, 63, 82]: the reason for such poor performances has to be identified in the centralized critic used by COMA to estimate such values. Such a critic is difficult to learn because of compounding factors, and thus may provide inaccurate or wrong values to the agents. For this reason, in this work two novel algorithms, named Dr.Reinforce and Dr.ReinforceR, are proposed. These avoid the above difficulties by applying difference rewards on the system reward function, either by accessing it directly or by learning it with a centralized network. The results show the improvements over COMA, pointing out how learning a more accurate representation is a key factor towards a wider applicability of difference rewards to solve the multi-agent credit assignment problem.

The original contributions presented in this thesis could deepen the understanding of multi-agent reinforcement learning, by providing evidences that carefully designed alternative representations are indeed useful in improving the learning of multiple agents and help in contrasting some fundamental problems

that characterize this kind of systems. Moreover, novel techniques could build upon the solutions investigated in this work, and further improve performance or allow us to tackle increasingly complex problems.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

Since its beginning, research on *artificial intelligence* (AI) [122] has investigated building machines capable of solving different and complex tasks that were considered sole prerogative of the human intelligence [44]. This desire of a machine capable of replicating human behaviours has driven this novel and flourishing field, that is now widely considered capable of reshaping our everyday life and provide effective and working solutions to a variety of problems that we consider crucial [64, 147, 62].

Reinforcement learning (RL) [141, 65] is perhaps considered one of the most promising set of techniques to achieve such goal: an AI agent autonomously learns how to interact with its environment and how to achieve the desired behaviour through repeated trials and errors, relying only on a numerical signal to assess its own performance with respect to the desired goal. Moreover, the recent increase in the available computing power (for example, the availability of fast computational resources like GPUs or TPUs) has allowed the reinforcement learning community to move a step further: *deep learning* (DL) [45, 113, 28] and *neural networks* [58, 48, 22] can be efficiently used as function approximators to represent reinforcement learning structures and tables, in what is called *deep reinforcement learning* (DRL) [41, 3, 12, 72]. Deep neural networks are capable of identifying and extracting compact features from large-scale, highly complex inputs and exhibit good generalization capabilities beyond their training examples [21, 9, 153, 29, 92], thus allowing deep reinforcement learning to tackle problems that seemed too difficult or too big to be solved with standard reinforcement learning techniques.

The brief history of deep reinforcement learning is paved with big successes: as an example, AlphaGo [133, 134] has been the first autonomous agent capable of defeating the human world champion Lee Sedol at Go, a game known for being extremely complex for machines to master due to a very large number of possible board configurations and the careful planning required to play a good game. Other successful reinforcement learning applications range from super-human scores in playing Atari games [85, 86] to autonomous navigation of unmanned aerial vehicles [78, 88, 6], to robot locomotion [173, 176, 61] and manipulation [75, 154, 37].

1.1 MOTIVATION

In many situations, an autonomous agent has to behave in an environment populated also by other acting agents (or even human beings), whose actions can affect the environment own dynamics in turn. These settings are known as *multi-agent systems* (MASs) [33, 136]. In such systems, the agents can have different types of interactions

and interests [144, 145]: they could have totally opposing goals, for example when playing a game one against the other, as in competitive settings [143]; they can share a common objective to achieve as in *cooperative settings* [15, 74, 107], thus requiring cooperation and coordination among the team; or more generally they can be self interested but not strictly competing against nor cooperating with any of the other agents in the system, as in self-interested settings [116]. For the remainder of this work, the focus is going to be purely on cooperative multi-agent systems.

The study of cooperative multi-agent systems has a long-standing history across a multitude of different fields: both natural, cognitive and behavioural sciences has investigated the problem of how cooperation has arisen and helped the shaping of animal and human communities [4, 148]. Also concerned with the study of rational behaviours of multiple agents (called players in this context) is the related field of *game theory* (GT) [105]: although in general not restricted to such settings only, abundant work has also been made on how to solve and identify optimal behaviours for cooperating players [19, 128, 49]. The concept of a suitable (i.e. rational) behaviour for the players capable of maximizing the team common outcome, called an equilibrium, is a commonly employed solution concept, and many algorithms have been proposed to find these, although the problem of identifying an optimal equilibrium is known to be highly difficult [105]. Such a difficulty is even exacerbated when the number of agents increases, or when the system dynamics become more complex and a careful and differentiate decision process in necessary according to the evolving situation of such a system.

Also AI has been interested in the field of multi-agent systems, especially in the cooperative setting [70, 98, 43]. Different solution techniques has been proposed so far: for example, for problems whose dynamics are known in advance, planning algorithms [43, 98] can be used to find optimal execution strategies for the team of agents. Also, techniques based on dynamic programming [56, 8], like value or policy iteration, or existing AI algorithms, like MAA* [95], has been adapted to multiple agents. However, accurately knowing the system dynamics is not often practical (for example in complex real-world applications), or it may still be difficult to exploit such a knowledge because of the size of the environment itself.

Reinforcement learning (and its deep incarnation) has been widely applied and investigated in the multi-agent setting too, resulting in the *multi-agent reinforcement learning* (MARL) field [14, 70, 54, 83, 59]. As for single-agent reinforcement learning, also MARL has achieved significant breakthroughs: AlphaStar [156], combining techniques from competitive MARL and game theory, has been capable of achieving the grandmaster rank in StarCraft II, a highly complex and temporally extended resource management video-game, playing in an online league against professional human players. It is therefore natural the growing interest that this novel field is gaining in the research community, with an increasing number of papers on the subject accepted at top level AI conferences such as AAAI, IJCAI, ICML, NeurIPS

and AAMAS (that is specifically dedicated to this topic) every year.

Cooperative MARL, where agents have to cooperate and coordinate to solve a common problem by learning only from a shared feedback signal, has seen an amazing number of real-world applications: examples are air traffic management [150], packet routing in sensor networks [177], traffic light control [151, 71], emergency rescue [109] and warehouse management [23], just to name a few. To show its importance, let's consider the application of MARL to the traffic light control problem: the coordination and coexistence of multiple traffic lights that impact on the same system (the flow at a set of traffic intersections) is intrinsically of a multi-agent nature. Such a problem is capable of shaping the way in which people use and even feel driving, and thus it has a great impact of our everyday life. Although a human operator may have clear ideas on how to regulate traffic lights such as to allow safe yet fast travel to the vehicles (for example, with intuitive rules such as "no green lights on empty lanes, as no one would benefit from these"), it is in general impractical to regulate such a system manually. Also, describing the traffic light control through a series of hard-coded rules may be difficult: there may be hundreds of good rules to implement, and some may be contrasting, or perhaps not clearly applicable or designable. Thus, having a solution that is capable of regulating such delicate situations by autonomously optimizing their behaviour to allow both fast and safe travelling, as well as being adaptive to unseen situations (like a different configuration of the traffic junctions or the number of vehicles), is a great step towards a suitable solution that may have a positive impact on our society.

1.2 LEARNING IN MULTI-AGENT SYSTEMS

Designing cooperative multi-agent systems presents a set of choices and challenges that stem from the unique property of having multiple autonomous agents and that are not usually faced nor addressed by standard reinforcement learning algorithms, therefore requiring novel and specific solutions. One of the first and most fundamental decisions that have to be made is that of how to control the team of agents: the two most common choices are using a *centralized controller*, that models the entire team of agents as a single centralized unit, and *independent learners* [146, 83, 24, 171], where each agent in the team is modelled and learned independently from the others. Although being both popular approaches, these are also well-known for their inherent limitations that may hinder or even impede the learning of proper cooperative behaviours [83, 24, 171, 108].

Another fundamental difficulty faced by learning agents in cooperative settings is that of *multi-agent credit assignment* [20, 94, 178, 168]: with only a shared numerical feedback to assess the quality of the team performance in the environment, agents do not have a clear way to figure out their own specific contribution towards the attained result. This way, each agent can be tricked into thinking that its performance is good,

while in reality the obtained feedback is mainly or solely due to the performances of the other acting agents, and this overall result could be further improved if that specific agent changes its behaviour towards a more performing or favourable one.

Both the choice of a suitable team representation and the tackling of the multi-agent credit assignment are challenges that are strictly inherent to cooperative multi-agent systems, and thus not directly addressed by standard single-agent reinforcement learning algorithms. The multi-agent community has already provided possible solutions to these, but these have been scarcely investigated in the field of multi-agent deep reinforcement learning yet. This is where the contributions of this work lay, trying to tackle two fundamental problems that could impede the learning of proper behaviours for the agents. This work aims at investigating the benefits of such multi-agent techniques in the field of MARL, and possibly extend these into novel algorithms that can improve upon current state-of-the-art solutions, and does so by learning additional or novel representations that could be used to directly address such limitations.

1.3 PROBLEM STATEMENT

In this work, the aim is to explore and design algorithms capable of learning representations that can help tackling the aforementioned challenges. The use of informative and correct representations can greatly benefit MARL algorithms: providing structures that can be easily learned and that are useful to contrast these difficulties have potential to provide the research community with better tools for designing new algorithms that can scale more gracefully with the number of agents present in the system, while at the same time achieving good performances. MARL is a new and flourishing field, and thus the research community may greatly benefit from such advances and improvements.

Given the benefits, but also the limitations, showed by both the centralized controller and the independent learners approaches, researchers have tried to identify a middle ground that could retain most of their benefits while at the same time reduce the number of possible drawbacks. One of such solutions is *factorization* [50, 53, 52] (also known as *value-decomposition*), in which the centralized controller is broken down into multiple different components, each only comprising a small subset of the whole team. This way, although the number of components that need to be learned is greater than a single centralized one, now each of these is only affected by the information of few agents, and thus is easier to learn and does not suffer the combinatorial explosion in size when the number of total agents increases. In contrast with independent learners instead, each of such components is still explicitly representing multiple agents together, and so it should partly ameliorate the non-stationarity issue.

Such an approach is now the underlying idea of a growing number of modern multi-agent deep reinforcement learning algorithms [139, 115, 135, 158]. However,

most of these algorithms (with some recent exceptions [10, 76, 89], that are indeed inspired by the original contribution presented in this work) focus only on a *single-agent decomposition*, where the centralized controller is divided into components representing only a single agent, leaving almost unexplored the benefits that using "higher-order" decompositions may have. Moreover, theoretical analyses in which general criteria and conditions for these algorithms to work well (i.e. result in selecting the same actions of a single centralized controller) are usually provided, but it is not always easy to tell if a given problem respects such conditions and more in general these analyses does not say anything about the quality of the resulting learned representations (as opposed to action selection only). With so much interest in this technique, a more rigorous and systemic empirical analysis, in which different factorizations are tested and compared on a number of different problems with varying properties, would be of a great benefit to assess its real strengths and eventual limitations and to inform future directions in its application.

Also the multi-agent credit assignment has been tackled with a variety of techniques: many value-decomposition methods [139, 115, 135, 158, 175] are indeed implicitly addressing such a problem [157], but these lack interpretability and it is in general not clear how the agents' contributions are computed. Instead, an explicit technique of noticeable interest and that found a wide application is that of *difference rewards* [169, 168]. The idea of difference rewards is to provide each autonomous agent with an individual *shaped reward* signal rather than using the shared feedback provided by the cooperative environment, and many different algorithms have been proposed to compute different expressions for such a shaped value [178, 169, 114, 26, 25]. With an individual signal to drive their learning process, agents should be better able to assess how their actions have been really contributing to the overall performance, and could improve their own behaviour in a way such that the whole team can benefit from this. There is however a limitation to the applicability of such methods: complete access to the system reward function is indeed required in order to compute the shaped signals. This is a strict condition in a reinforcement learning setting, where usually the reward function is not known in advance and the agents have access to its samples only through their repeated interactions with the environment.

As well as "higher-order" factorizations, difference rewards algorithms have not been widely applied in the field of multi-agent deep reinforcement learning, especially because of the aforementioned limitation, where the problem of multi-agent credit assignment has only recently gained a major attention [20, 160, 40, 182, 130]. However, among other deep MARL algorithms specifically designed to explicitly address such a problem [159, 174], there is only one that has been proposed that applies the idea of difference rewards, *Counterfactual Multi-Agent Policy Gradients* (COMA) [40]. Although the algorithm is cleverly applying the difference rewards mechanism on a learned representation of action-value function of the entire team, thus allowing

each agent to explicitly reason about the contribution of its own actions, its results in practice are often poor [160, 181, 63, 82], rendering COMA not practical to use in larger multi-agent systems. The reasons for these poor performances are probably to be identified in its centralized representation itself: learning the Q-function is a difficult problem due to compounding factors such as bootstrapping, the moving target problem (as target values used in the update rule change over time), and the Q-function dependence on the joint actions. This makes the approach difficult to apply with more than a few agents.

1.4 CONTRIBUTIONS & RESEARCH QUESTIONS

This work aims at making a step further in tackling these two MARL problems, as considered crucial for the development of the field. First, an in-depth and accurate analysis and comparison of different factorizations on a wide set of diverse cooperative scenarios, each with its own difficulties and peculiarities, is provided. Justified by the growing advent of deep reinforcement learning, neural networks are used to represent and learn these factored methods. The purpose is to foster a clear and solid understanding of the various benefits and limitations of such techniques, assessing how various instantiations of this can represent the behaviour of the agents and how does these scale with the size of the system. In order to assess this, the proposed analysis has been restricted to the one-shot decision setting (i.e. the team perform a joint action and the environment immediately reset), as the simplest setting capturing many of the cooperative multi-agent-related issues, such as the exponential explosion of the number of actions and the strict coordination requirements, without hindering the comparison with additional details like sequentiality of decisions or the representation of states. The hope is to inform future research on the real benefits and capabilities of using "higher-order" decompositions, and also to provide a practical guide with results on different settings, on which system designers can rely to choose a suitable technique for their own problem at hand.

At a high level, the prominent research question (RQ) that this work aims at addressing is the following:

RQ1: Is the general assumption on "higher-order" factorizations being helpful in learning improved approximations in the multi-agent setting justified by practical results?

Chapter 3

Then, multi-agent credit assignment and its application to sequential deep MARL is considered, by proposing *Dr.Reinforce* and *Dr.ReinforceR*, two novel algorithms that overcome most of the difficulties that limit COMA by combining a difference rewards mechanism computed on the reward function, either accessible from the environment

itself or learned with a centralized network, with policy gradient learners, being able to outperform this state-of-the-art algorithm. Also, an analysis of the learning problem is performed, in which the benefits of learning the reward function rather than a centralized action-value critic as in COMA are shown.

RQ2: Can difference rewards applied to the reward function (possibly by learning a representation of it) provide better learning signals to distributed agents compared to the same idea applied on a centralized action-value critic?

Chapter 4

RQ3: Can the same idea be extended to partially observable settings and still improve performances?

Chapter 5

The remainder of this work is structured as follows: in Chapter 2 preliminary notions about game theory, single-agent reinforcement learning and its extension to cooperative multi-agent systems are provided. This chapter is based on research made by other people, and relevant citations to identify the original authors and works are provided. In Chapter 3 we provide a thoughtful comparison and analysis of different factorizations in representing action-value functions arising from a diverse set of one-shot multi-agent problems. In Chapter 4 we combine difference rewards with policy gradients to provide novel deep MARL algorithms that are capable of addressing the multi-agent credit assignment problem while avoiding many of the issues encountered with action-value function learning, and in Chapter 5 such techniques are extended to partially observable settings. Finally, conclusions and further remarks are provided in Chapter 6.

1.5 PUBLISHED WORKS

The work in Chapter 3 is based on the following paper:

- Jacopo Castellini, Frans A. Oliehoek, Rahul Savani and Shimon Whiteson. The Representational Capacity of Action-Value Networks for Multi-Agent Reinforcement Learning - Extended Abstract. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'19, pages 1862–1864. International Foundation for Autonomous Agents and MultiAgent Systems, 2019,

which has been subsequently extended into:

- Jacopo Castellini, Frans A. Oliehoek, Rahul Savani and Shimon Whiteson. Analysing Factorizations of Action-Value Networks for Cooperative Multi-Agent Reinforcement Learning. *Autonomous Agents and Multi-Agent Systems*, 35(25):53 pages, Springer Nature, 2021.

The work in Chapter 4 is instead based on:

- Jacopo Castellini, Sam Devlin, Frans A. Oliehoek and Rahul Savani. Difference Rewards Policy Gradients - Extended Abstract. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'21, pages 1475–1477. International Foundation for Autonomous Agents and Multi-Agent Systems, 2021,

- Jacopo Castellini, Sam Devlin, Frans A. Oliehoek and Rahul Savani. Difference Rewards Policy Gradients. In *AAMAS'21 Autonomous and Learning Agents Workshop* (Best Paper Award Winner), ALA'21, 2021,

Finally, the content of Chapter 5, extending the span of the above works, has been published into:

- Jacopo Castellini, Sam Devlin, Frans A. Oliehoek and Rahul Savani. Difference Rewards Policy Gradients. *Neural Computing and Applications (ALA'21 Special Issue)* (under review), 2022

2 BACKGROUND

In this chapter preliminary notions about single-agent reinforcement learning, game theory and cooperative multi-agent reinforcement learning are provided, as used throughout the remainder of this work.

2.1 REINFORCEMENT LEARNING

Reinforcement learning is one of the most popular family of machine learning techniques. It is used to learn a (near-)optimal behaviour for an agent to follow in an unknown environment, only through repeated interactions with such an environment. In this section, after a preliminary introduction to the setting, focus is going to be on its combination with deep learning (called deep reinforcement learning), introducing both value-based and policy-based algorithms. Finally, a brief excursion on partially observable settings is provided.

2.1.1 *The RL Problem*

Single-agent reinforcement learning is a class of problems based on trial and error interactions of an autonomous agent with its environment. The process proceeds in a discrete fashion, at each time step, the agent perceives the state of the environment, and based on this perception it internally decides the action that it wants to perform. When the agent has performed its action, the environment responds to this by transitioning to a new state based on the previous one and the action chosen by the agent itself and by providing the agent with a numerical signal, that expresses the value of its choice in that specific situation, and then the entire process repeats from the new situation. The aim of the autonomous agent is, through multiple such interactions with the environment, to learn a mapping from environment states to actions, called a policy, so that the overall reward accumulated over a sequence of these is as high as possible. This iterative process is schematically depicted in Figure 2.1.

Formally, a reinforcement learning problem can be defined as a *Markov decision process* (MDP) [141, 65, 41, 3, 12] $\mathcal{M} = \langle S,A,T,R,\gamma \rangle$, where S is the (usually finite) set of environment states, A is the set of actions available to the agent, that may be finite and discrete (as considered in the remainder of this work unless explicitly stated) or infinite and continuous, $T : S \times A \times S \to [0,1]$ is called the transition function (also known as transition dynamics, or transition kernel) and expresses, given the current state $s \in S$ and selected action $a \in A$ and a candidate future state $s' \in S$,

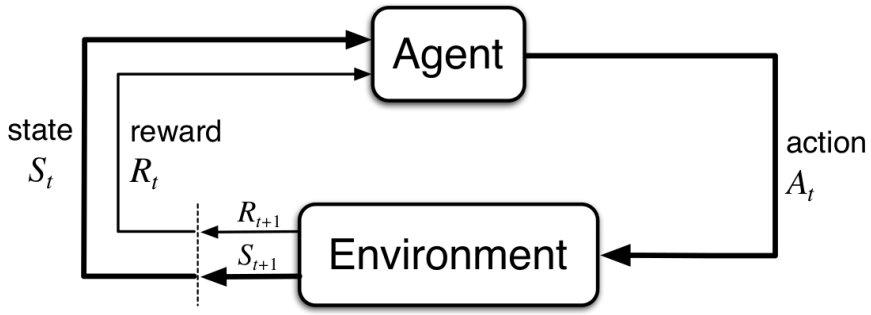


Figure 2.1: A schematic view of the RL process: the agent repeatedly interacts with the environment. Source: Sutton and Barto 2018.

the probability of the environment transitioning to s' when action a is performed in state s. Finally, $R : S \times A \to \mathbb{R}$ is the reward function, used to give feedback to the agent when action a is performed in state s, and $\gamma \in (0,1]$ is a discount factor used to balance the importance of current and future performances.

A fundamental property of MDPs is the so-called *Markov property*: the transition dynamics do not depend on the entire history of the environment, but these can be determined solely based on the current information. This property allows the agent to be oblivious about the history of the environment, and it is at the basis of many algorithms' convergence proofs [162, 142, 69].

At a given time step t, the agent has to maximize its discounted future return over a T-steps (with T being possibly infinite when $\gamma \neq 1$) trajectory $R_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$, where r_t is the reward obtained at time step t. To select its actions, an agent maintains and learns either a *stochastic policy* $\pi(s,a)$, mapping environment states to the corresponding probability value of selecting each available action (when the action set A is discrete and finite), or a *deterministic policy* $a = \pi(s)$, where a fixed action is selected for each state (usually used in continuous action problems).

An agent policy π induces a *value-function* $V^\pi(s) = \mathbb{E}[R_t|s_t = s]$, expressing how good it is for the agent to be in state s and follow its policy, and an *action-value function* $Q^\pi(s,a) = \mathbb{E}[R_t|s_t = s, a_t = a]$ (also called *Q-function*), that expresses how good it is for the agent to choose action a in state s and then follow its policy. The aim of the agent is to learn an optimal policy $\pi^* = \max_\pi V^\pi(s), \forall s \in S$.

Both the value function $V^\pi(s)$ and the action-value function $Q^\pi(s,a)$ can be recursively defined via the Bellman equation [7, 141, 65] as:

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) \left(R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s') \right), \tag{2.1}$$

$$Q^\pi(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^\pi(s'), \tag{2.2}$$

$$V^\pi(s) = \sum_{a \in A} \pi(a|s) Q^\pi(s,a). \tag{2.3}$$

These definitions also entail that, if the transition dynamics $T(s,a,s')$ and the reward function $R(s,a)$ are known, we can find the optimal value function V^{π^*} induced by π^* using the value iteration algorithm:

$$V^{\pi}(s) \leftarrow \max_{a^* \in A} \left(R(s,a^*) + \gamma \sum_{s' \in S} T(s,a^*,s') V^{\pi}(s') \right). \tag{2.4}$$

However these quantities are usually unknown in the reinforcement learning setting, as opposed to planning [43] in which these are instead known in advance, and so the policy has to be learned by interacting with the environment rather than using dynamic programming.

One of the intrinsic challenges of reinforcement learning is that of the *exploration vs. exploitation trade-off* [141, 18]: on one hand, an agent wants to maximize its accrued reward, thus choosing at each state the action that seems to maximize such a value based on its imperfect estimates, but on the other hand it cannot be sure that these estimates are really accurate enough and that the selected action is really optimal, and thus need to continue exploring to further refine them. This trade-off is crucial in allowing the agent to learn proper estimates of the real values of its actions, and needs to be carefully considered and each action to be sufficiently explored in order to achieve a really optimal policy.

2.1.2 *Partial Observability*

The MDP framework allows us to model situations in which the agent has a perfect sensing of the surrounding environment, being able to capture everything it needs to choose an action. In many real-world situations however this is not the case, as the agent may have limited or imperfect sensing [35, 6, 141].

A framework capable of modelling such situations is that of *partially observable MDPs* (POMDPs) [34, 98]. A POMDP can be mathematically defined as $\mathcal{M} = \langle S,A,T,O,Z,R,\gamma \rangle$, where S,A,T,R and γ are as in a standard MDP. The main difference is that now the agent does not perceive the environment state $s \in S$ directly, but it is rather provided with an observation $o \sim Z(s) \in O$, where O is the observation set, drawn from the observation function $Z : S \times O \to [0,1]$ based on what the real state is.

In a POMDP, the environment loses the Markov property from the perspective of the agent: not being able to observe the full state, now the observation alone is not enough for the agent to determine the complete system dynamics, that thus seems non-stationary from the agent's perspective. This property loss disrupts all of the convergence properties of standard reinforcement learning algorithms, as now the agent has to condition on the *action-observation history* $h_t = (o_0,a_0,o_1,a_1,\ldots,o_{t-1},a_{t-1},o_t) \in (O \times A)^* \times O$ reached until the current time step t (rather than on the real state s) to try and figure out the underlying configuration of the environment. However, this

history can grow unbounded, and this can be problematic when the policy has to react differently based on each such a history. Dedicated techniques are required to address this issue, such as truncating the histories to a certain length or using recurrent neural networks, with their internal representation, as function approximators instead of feed-forward ones [57, 166].

2.1.3 *Deep Learning*

Neural networks (NNs) [58] are used to extract salient information from high-dimensional inputs and process these to obtain the desired output. These have the capability to generalize beyond the training examples used to learn them, and therefore can be used as *function approximators* [45, 72] in many different problems, from classification to regression and time series analysis [113, 28]. The most basic type of neural networks are *feed-forward neural networks*, in which each input i_t is passed through a series of L stacked layers, each with its own weights W^l and bias b^l, so that the corresponding output o_t is computed as:

$$o_t = \sigma(W^L \cdot \sigma(W^{L-1} \cdot \ldots \sigma(W^2 \cdot \sigma(W^1 \cdot i_t + b^1) + b^2) \ldots + b^{L-1}) + b^L), \tag{2.5}$$

where σ is a linear activation function, for example the ReLU activation function [90]. Networks parameters $\Pi = \{W^l, b^l\}_{l=1}^{L}$ are then adjusted to reduce the difference between the output o_t and a desired target value y_t for that specific input under a given *loss function* $S(o_t, y_t)$ (usually the categorical cross-entropy for classification problems or the mean square error for regression) [45]. Such an adjustment is performed by following the direction of the *gradient* of the loss function (the set of first-order derivatives with respect to each network parameter) [11], computed using the *backpropagation algorithm* [119], and parameters Π are updated using (a variant of) the *stochastic gradient descent* (SGD) algorithm [11, 117]:

$$\Pi \leftarrow \Pi + \eta \nabla_{\Pi} S(o_t, y_t), \tag{2.6}$$

where $\eta \in (0,1)$ is a suitable learning rate value. If update steps are sufficiently small, the neural network should iteratively adjust its parameters is a direction that minimize its error, thus predicting correct outputs for the given inputs.

Another very popular type of neural networks are *recurrent neural networks* (RNNs) [120, 72], that have the capability to deal with data of a sequential nature by using a kind of memory mechanism that allows them to correlate consecutive pieces of information, rendering them particularly important in tasks such as time series analysis or natural language processing [113, 28]. This capability is implemented by a set of self-directed connections W_R^l inside each recurrent layer, allowing it to compute its output h_t^l (called its *internal state*) based on the current input x_t (that may be the

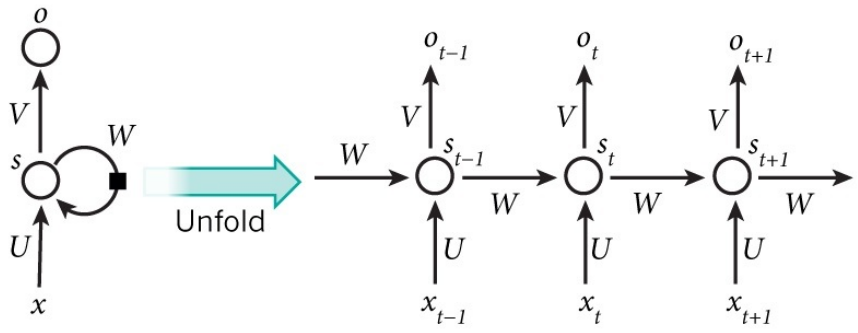


Figure 2.2: A schematic representation of a recurrent neural network and its unfolding through time. Source: LeCun et al. 2015.

output of the previous layer in the network h_t^{l-1} or the initial input i_t) and its own previous output h_{t-1}^l as:

$$h_t^l = W^l \cdot x_t + W_R^l \cdot h_{t-1}^l + b^l. \tag{2.7}$$

Recurrent neural networks can in principle update their own parameters in a similar way to that of feed-forward ones, but these now have to also take into account their own previous outputs when computing the gradient of the loss function, that in turn depends on the network parameters at the previous steps and so on. Algorithms such as *backpropagation through time* [164] have been proposed to learn these networks by unfolding them along the temporal dimension, although problems like the vanishing or the exploding gradient are common with the training of vanilla RNNs (but are ameliorated by the gated mechanisms in extensions such as LSTM [48] or GRU [22]).

2.1.4 *Q-Learning & SARSA*

One of the most famous reinforcement learning algorithm is Q-learning [162, 65, 141]. Q-learning is a *value-based method*, i.e. a method that learns the action-value function $Q(s,a)$ and then uses this to shape the agent policy, for example by acting greedily with respect to the action-values or using some probability distribution like the Boltzmann distribution [65]. As the action-values (also called Q-values) are an estimate of the expected discounted future reward for the agent starting in a certain situation and choosing a certain action, acting probabilistically according to these would make the agent focus on actions that seem more valuable and capable of yielding a high overall reward. However, this does not entirely discard exploration of suboptimal actions, thus tackling the exploration vs. exploitation trade-off and allowing the agent to keep refining its estimates.

Q-learning works by maintaining a table in which the entries are all of the state-action pairs, and for each it stores the estimated action-value that the algorithm learns.

The algorithm uses bootstrapping to compute the updates: for a given transition (s,a,r,s') that the agent experiences, this update is computed using the so-called *temporal-difference* (TD-) error δ:

$$Q(s,a) = Q(s,a) + \alpha \left(\underbrace{r + \gamma \max_{a' \in A} Q(s',a') - Q(s,a)}_{\delta} \right), \tag{2.8}$$

where $\alpha \in (0,1)$ is the learning rate of the algorithm.

This is an *off-policy algorithm* [141], as it does not consider the real action performed by the agent at the next time step (when the environment is in state s'), but considering instead the action that gives the maximum action-value, as if the agent were going to behave greedily from the next step on.

The biggest limitation of Q-learning is its use of a table to store the action-values: if the size of the environment is large (with a large state space for example), the table can easily become too big to fit in memory. For this reason, *function approximation* has been proposed, with techniques such as tile coding [141, 165] or neural networks [86]. With function approximation, the entire table is approximated by a fixed-size set of parameters θ, that are learned to reproduce as accurately as possible the same function learned by tabular Q-learning:

$$\theta = \theta + \alpha \left(r + \gamma \max_{a' \in A} Q_\theta(s',a') - Q_\theta(s,a) \right) \nabla_\theta Q_\theta(s,a). \tag{2.9}$$

Another popular value-based algorithm is SARSA [121, 65, 141]. This algorithm is similar to Q-learning in the way it updates the Q-function, but differently from the former it is an *on-policy algorithm* [141]: it uses the action a' that the agent really chose at the next time step while interacting with the environment to compute the TD-error δ:

$$Q(s,a) = Q(s,a) + \alpha \left(\underbrace{r + \gamma Q(s',a') - Q(s,a)}_{\delta} \right). \tag{2.10}$$

2.1.5 *Deep Q-Network*

The use of neural networks as function approximators [149], with their compact representations and generalization capabilities, has fostered the field of deep reinforcement learning in gaining the status of a breakthrough in modern AI. One of the first and most popular algorithms is deep Q-network (DQN) [86, 152, 161]. DQN works similarly to the standard version of Q-learning with function approximation [149], but it uses a couple of tricks to improve the stability of the learning process, that have been a major issue when using neural networks.

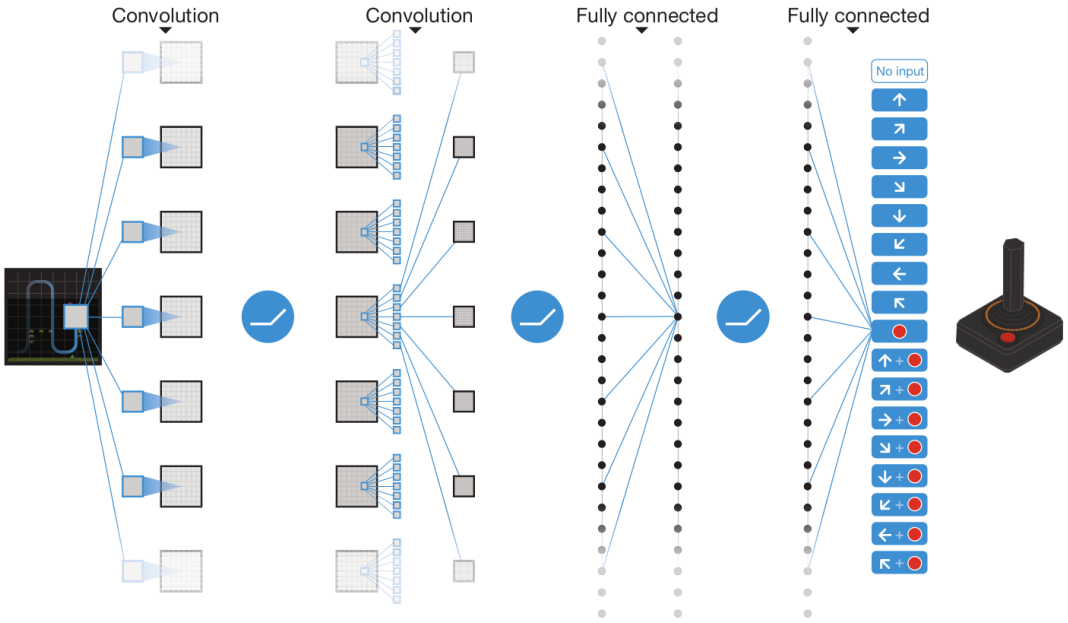


Figure 2.3: A schematic representation of the DQN architecture. Source: Mnih et al., 2015.

This first trick is the use of a large buffer $\mathcal{D}$, called *experience replay* [86, 124], to store transitions $\langle s,a,r,s' \rangle$ encountered during execution. These transitions are then uniformly sampled and used for *offline learning*. This way, the statistical correlation between subsequent samples, that could lead to poor learning performances [86], is broken. The second trick is the use of a *target network*, a copy of the trained neural network whose parameters θ^- are periodically copied from those of the trained network θ, when computing the updates target, contrasting the *moving target problem* [86, 152] (a network update will also change the value of the targets that it is trying to match, therefore resulting in difficulties in converging to a stable representation).

In DQN the input is the state s of the environment, that is then passed through a series on neural network layers until the output units, one for each available action $a \in A$ and representing the corresponding Q-value for that action. The architecture is schematically represented in Figure 2.3.

The updates are computed similarly to Q-learning with function approximation, but sampling a batch of transitions to use for training from the experience replay $\mathcal{D}$ and using the target network θ^- to compute the updates target. The overall loss function then is:

$$\mathcal{L}(\theta) = \mathbb{E}_{\langle s,a,r,s' \rangle \sim U(\mathcal{D})} \left[\left(r + \gamma Q_{\theta^-}(s',a') - Q_\theta(s,a) \right)^2 \right]. \tag{2.11}$$

Different variants of DQN have been proposed [152, 161], and the algorithm has in general shown the promises of deep reinforcement learning, for example by learning how to play Atari games at a superhuman level by only learning from raw input images [85, 86].

2.1.6 *Policy Gradients*

A different approach to that of value-based methods are *policy gradient* (PG) methods [142, 141], a family of algorithms that, instead of learning the action-value function and then designing a policy on top of that, aim at learning a stochastic parametrized policy π_θ directly. Because the objective of an agent policy is to maximize the expected future discounted return (the value-function) $V^\pi(s)$, policy gradient methods improve the parameters θ of the policy π_θ by maximizing a parametrized version of such a value-function $V(\theta) = \mathbb{E}_{s_0}[V^{\pi_\theta}(s_0)]$, where $s_0 \in S$ is the initial state of the environment. This can be done by performing gradient descent [11, 117] on the following gradient:

$$\begin{aligned}\nabla_\theta V(\theta) &= \nabla_\theta \sum_{s\in S} d^{\pi_\theta}(s) V^\pi(s) \\ &= \nabla_\theta \sum_{s\in S} d^{\pi_\theta}(s) \sum_{a\in A} \pi_\theta(a|s) Q^{\pi_\theta}(s,a),\end{aligned} \tag{2.12}$$

where $d^{\pi_\theta}(s)$ is the ergodic state occupancy measure as defined in [142, 141], and the equality comes from the equivalence in Equation 2.3.

Computing the derivative of this quantity is however problematic, as this measure is usually unknown and complex to compute directly. However, the *policy gradient theorem* [142, 141, 132] reformulates this expression in a way such that the derivative of $d^{\pi_\theta}(s)$ can be avoided. Furthermore, because the gradient with respect to this term is not further required, the entire expression can be reduced to a practical version that approximates the required quantities by sampling from the environment, thus avoiding the expectations in Equation 2.12. The parameters of the policy θ are thus updated as:

$$\theta \leftarrow \theta + \alpha \underbrace{\sum_{t=0}^{T-1} Q^{\pi_\theta}(s_t,a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)}_{g}, \tag{2.13}$$

where α is the learning rate and g indicates the approximated gradient. Another problem with policy gradients is that also the action-value function $Q^{\pi_\theta}(s,a)$ is unknown, and thus a practical way to approximate that is required. One of the earliest policy gradient algorithm, called REINFORCE [167, 141], proposes to approximate the Q-function using the *Monte-Carlo return* $G_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$ as an unbiased estimate of that. This quantity is simple to compute, as it only involves the reward values obtained by the environment while interacting, and so it is the corresponding instantiation of the policy gradient:

$$\theta \leftarrow \theta + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t G_t \nabla_\theta \log \pi_\theta(a_t|s_t)}_{g}. \tag{2.14}$$

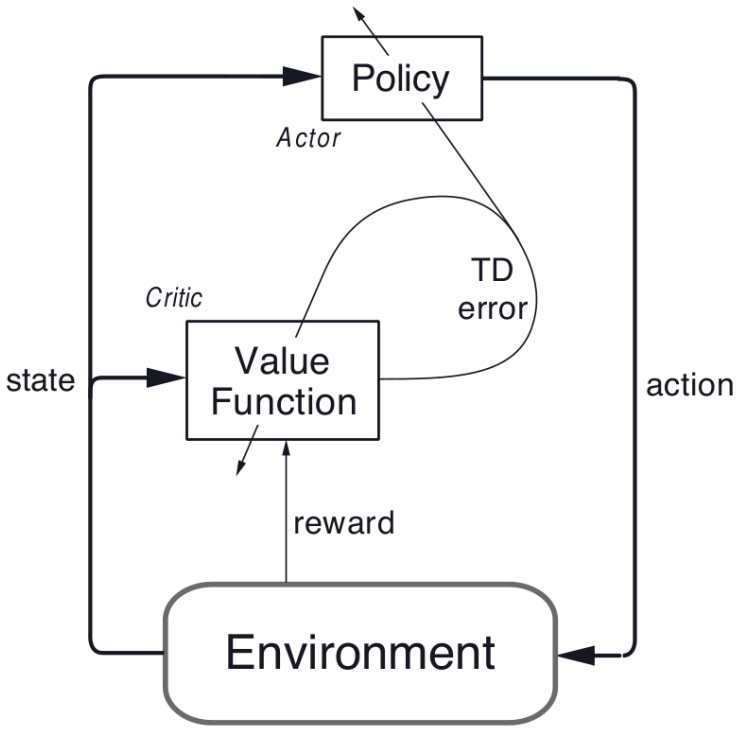


Figure 2.4: A schematic representation of the actor-critic algorithm: the actor selects actions to interact with the environment, while the critic informs the actor about the quality of such actions. Source: Sutton and Barto, 2018.

However, the return has a high variance because of the stochasticity of the environment, and this may deteriorate the policy gradients and slow down the algorithm convergence. For this reason, the use of an *unbiased baseline* $b(s)$ [47, 141], not depending on the agent policy, has been proposed to reduce such a variance:

$$\theta \leftarrow \theta + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t (G_t - b(s)) \nabla_\theta \log \pi_\theta(a_t|s_t)}_{g}. \tag{2.15}$$

A different class of algorithms, called actor-critic [69, 87, 77], proposes to learn an approximation of the Q-function (or the value function, for which the following equations are similar) named the *critic* and parametrized by ω, together with the policy π_θ. This can be learned by minimizing the on-policy TD-error (but also off-policy versions exist [77]):

$$\delta = r + \gamma Q_\omega(s',a') - Q_\omega(s,a). \tag{2.16}$$

It can then be used to compute the policy gradient of the actor π_θ as:

$$\theta \leftarrow \theta + \alpha \underbrace{\sum_{t=0}^{T-1} Q_\omega(s_t,a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)}_{g}. \tag{2.17}$$

The critic, by learning an approximation of the expected return, is less prone to variance than a single Monte-Carlo sample (as used in REINFORCE) and thus can improve the convergence of the policy gradients, at the cost of introducing some bias. It is also possible to subtract an unbiased baseline from the critic value $Q_\omega(s,a)$ to further reduce variance. If such a baseline corresponds to the value function, the result is the *advantage function* [47, 87, 141]:

$$A(s,a) = Q(s,a) - V(s) = r + \gamma V(s') - V(s). \tag{2.18}$$

2.2 GAME THEORY

In many settings [150, 73, 129], the interest is on finding a suitable (i.e. rational) strategy for a team of n agents so that each of them is capable of achieving the highest possible outcome from a given situation. Such is the context that is studied by classic game theory. In the following section, a brief introduction to some of its basic concepts is provided, as serving as a foundation stone for the the study of multi-agent systems. Also, details on some more specific game-theoretical settings mentioned in the remainder of this work, such as Bayesian games or partially observable stochastic games, are given.

2.2.1 *(Cooperative) One-Shot Games*

The simplest possible setting to investigate such rational interactions is that of a *one-shot game* [105] (also known as a stateless problem or an n-players normal/strategic form game in the game theory language), which can be modelled as a tuple $\mathcal{M} = \langle D,\{A^i\}_{i\in D},\{R^i\}_{i\in D}\rangle$, where $D = \{1,\dots,n\}$ is the set of agents (called players) and A^i is the action set for player i. Each player chooses what to do in the game based on its own strategy π^i, which can be either a *pure strategy*, if it deterministically defines an action a^i to select for the game $\pi^i = a^i$ (so that each player has a set of available pure strategies corresponding to its action set A^i), or it can be a *mixed strategy*, if it defines a probability distribution over the player set of pure strategies $\Delta^i(A^i)$. A strategy profile $\pi = \langle \pi^1,\dots,\pi^n\rangle$ is a set comprising a suitable strategy (either pure or mixed) for each of the players. Finally, $R^i : A \to \mathbb{R}$ is the individual reward function (or individual payoff function in this setting) for player i, dependent on the *joint action* $a \in A = \times_{i\in D} A^i$ of the players (a realization of the players strategy profile). In cooperative one-shot games [105, 19], the payoff function is common to all players, $R^i = R : A \to \mathbb{R}, \forall i \in D$.

It is to note that in such a stateless setting, the shared reward function $R(a)$ corresponds to the team action-value function $Q(a)$, as there is no possible future outcome that has to be taken into account. This (and the other reinforcement learning-

based nomenclature) is the notation used in Chapter 3 to further highlight the similarities with standard sequential MARL.

Closely related settings are that of repeated games [105], in which the players are faced with the same one-shot game for a series of multiple stages and alternate their actions in turns, conditioning on the joint action history $h_t = (a_1, a_2, \dots, a_t)$ to choose a suitable strategy at every turn, and that of multi-agent bandit problems [129], in which the agents have to agree on which arm to pull in a classical multi-armed bandit problem [141] and whose main focus is on minimizing regret [129, 105].

2.2.2 *Nash Equilibria & Optimal Solutions*

In a one-shot game, the aim of each player is to choose a suitable strategy such that it maximizes its own expected accrued payoff $\pi^{i,*} = \arg\max_{\pi^i} \mathbb{E}_{a\sim\pi}\left[R^i(a)\right]$, given the other players strategy profile. Given the common goal of all the players to find such a strategy for themselves, each player has to consider that the others may switch to a different strategy to increase their own payoff, so that the player selected strategy $\pi^{i,*}$ is not optimal anymore. A very popular solution concept that takes this idea into account and has been widely investigated in game theory is that of a *Nash equilibrium* (NE) [91, 105]: a Nash equilibrium is a strategy profile π^* from which none of the players has the incentive (i.e. can increase its own payoff) to unilaterally deviate. This means that, given a Nash equilibrium $\pi^* = \langle \pi^{1,*}, \dots, \pi^{n,*} \rangle$:

$$\nexists \pi' = \langle \pi^{1,*}, \dots, \pi^i, \dots, \pi^{n,*} \rangle : \mathbb{E}_{a'\sim\pi'}\left[R^i(a')\right] > \mathbb{E}_{a\sim\pi^*}\left[R^i(a)\right], \forall i \in D. \tag{2.19}$$

Such an equilibrium strategy is guaranteed to exist for finite games [91, 105], and in general may not be unique. Also, such equilibria are not in general guaranteed to be the strategy that gives the highest payoff to all of the players (a concept known as the *Pareto optimal solution*), as it happens for example in the popular Prisoner Dilemma [105]. However, in a cooperative one-shot game, where the aim of the players is to identify and coordinate on a strategy profile that gives the maximum common payoff, such an optimal solution is also guaranteed to be a Nash equilibrium [19]. Each strategy $\pi^{i,*}$ is called a *best response* to the other players' strategies in π^*, and indeed a popular algorithm to identify a Nash equilibrium is the Iterated Best Response method [105]: iteratively each player computes its best response against the fixed strategies of the other players (usually by solving a linear program [27]), until none of the players changes its strategy anymore.

2.2.3 *Graphical Games & Coordination Graphs*

A particular type of one-shot games is *graphical games* [100]: in a graphical game the individual payoff of each player R^i is affected only by the actions a^e of a subset of other players $e \subseteq D$. Thus, in order to maximize its own payoff, the player does not have to take into account the entire strategy profile π, but has only to consider the strategies $\pi^e = \langle \pi^j \rangle_{j \in e}$ to find an equilibrium strategy.

In cooperative settings as well, a player's decision can only be affected by the decisions of a subset of all the players, rather than the entire team. Such influences can be represented as a (hyper-)graph called *coordination graph* (CG) [53, 50, 52, 67], in which the nodes represent the players in the team and (hyper-)edges $e \in \mathcal{E}$ connect players who are influencing each other. Such a locality of interaction allows us to break down the team action-value function $Q(a)$ (i.e., the shared payoff function $R(a)$ using the game theoretic nomenclature) into a sum of smaller *local payoff functions* Q^e, one for each edge in the coordination graph, called a *factorization*:

$$Q(a) = \sum_{e \in \mathcal{E}} Q^e(a^e), \tag{2.20}$$

where $a^e = \langle a^i \rangle_{i \in e}$ is called *local joint action*, formed by the joint action of only the players comprised in such a local term e, named a *factor*. Coordination graphs are a useful tool to represent influences among the players, and message-passing algorithms, such as variable elimination [50] and max-sum [67, 118], work over coordination graphs and require good representations of the payoff function to allow the team to choose an optimal and coordinated joint action a^*.

Such an idea also has been applied to problems that do not present a decomposable structure on their own. For these cases, it is possible to resort to an *approximate factorization*: a set of factors $\mathcal{E}$ is designed beforehand and superimposed over the problem, that is then decomposed following the resulting factorization as:

$$Q(a) \approx \hat{Q}(a) = \sum_{e \in \mathcal{E}} \hat{Q}^e(a^e). \tag{2.21}$$

2.2.4 *Bayesian Games*

Another type of games is Bayesian games [100, 101], in which each player i is given some private information θ^i that it can use to condition its strategy $\pi^i(\theta^i)$ on. Thus, the actions selected by a player strategy (or the probability to select each action for a mixed strategy) now depend on the value of its private information, and a player has to consider what kind of private information the other players may have when evaluating its own strategy.

2.2.5 *(Partially Observable) Stochastic Games, MMDPs and Dec-POMDPs*

The most general form of games, called *stochastic games* (SG) [131, 98], also includes environment transitions among different states, determined by the players' actions. Formally, a stochastic game is defined as a tuple $\mathcal{P} = \langle D,S,\{A^i\}_{i\in D},T,\{R^i\}_{i\in D}\rangle$, where D,A^i and R^i are the same as in a one-shot game, S is the environment state set and $T : S \times A \times S \to [0,1]$ is the transition function of the environment, depending on the joint action of the players $a \in A = \times_{i\in D} A^i$. Similarly to repeated games, players can select their actions either simultaneously or in turns. In these games, each player observes the environment state $s \in S$ and aims at maximizing the discounted total payoff over a T-step series of stages (where T could possibly be infinite) $v^i = \sum_{t=0}^{T-1} \gamma^t R^i(s_t,a_t)$, where $\gamma \in (0,1]$ is a discount factor.

The above model can be adapted to represent cooperative settings: the resulting *multi-agent MDP* (MMDP) [98, 13] is defined as a tuple $\mathcal{M} = \langle D,S,\{A^i\}_{i\in D},T,R,\gamma\rangle$, where all the components are the same as in a stochastic game, but the reward function is common to all agents in the team, $R^i = R : S \times A \to \mathbb{R}, \forall i \in D$. In a MMDP, all the agents usually act simultaneously. Similarly to single-agent reinforcement learning, the team aims at maximizing the (now shared) discounted future return $R_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$: each agent can use the environment state s to condition its own (stochastic or deterministic) *local policy* $\pi^i(s,a^i)$, and the *joint policy* $\pi(s,a) = \langle \pi^1,\dots,\pi^n\rangle$ can be defined for the entire team. Also, the *joint value-function* $V^\pi(s)$ and *joint action-value function* $Q^\pi(s,a)$ can be defined similarly to the single-agent case. Each agent i cannot in general access information from the other agents, such as their own actions a^{-i} they perform: the other agents acting and learning together with the protagonist one are influencing the transitions and reward values given by the environment, therefore the agent itself perceives this environment as highly non-stationary [83, 24, 108].

Stochastic games can also be extended to allow for partial observability. A *partially observable stochastic game* (POSG) [56, 98] is defined as a tuple $\mathcal{P} = \langle D,S,\{A^i\}_{i\in D},T,$ $\{R^i\}_{i\in D},\{O^i\}_{i\in D},Z\rangle$, where everything is as in a standard stochastic game, but now each player only observes a local observation $o^i \in O^i$, where O^i is its *local observation set*, and the *joint observation* $o \sim Z(s) \in \times_{i\in D} O^i = O$ is taken from the observation function $Z : S \times O \to [0,1]$. If a (partially observable) stochastic game is finite, it is also guaranteed to have at least one (possibly mixed) Nash equilibrium [105].

Adapting a partially observable stochastic game to the cooperative setting results in a *decentralized partially observable MDP* (Dec-POMDP) [98, 14]. This is defined as a tuple $\mathcal{M} = \langle D,S,\{A^i\}_{i\in D},T,\{O^i\}_{i\in D},Z,R,\gamma\rangle$, where everything is defined as in a POSG, but the reward function $R : S \times A \to \mathbb{R}$ is now shared as in MMDPs. For the remainder of this work, we assume the action set A^i for each agent i to be discrete and finite. As in a POMDP, agents do not perceive the full state of the environment, but are instead provided each with a local observation o^i that can be noisy and not enough informative on the true underlying state of the environment, and so they have to rely

on their own *local action-observation history* $h_t^i = (o_0^i, a_0^i, o_1^i, a_1^i, \dots, o_{t-1}^i, a_{t-1}^i, o_t^i) \in (O^i \times A^i)^* \times O^i$ up to the current time step t to try and capture what the true environment state is (a *join action-observation history* $h_t = \times_{i \in D} h_t^i$ can be defined similarly to other quantities). These are used to condition a joint policy $\pi(h,a) = \langle \pi^1, \dots, \pi^n \rangle$, where $\pi^i(h^i, a^i)$ is the local policy for agent i. The joint policy induces a joint value-function $V^\pi(h)$ and a joint action-value function $Q^\pi(h,a)$ that are defined similarly to the MMDP case. The combination of other learning agents and partially observability only exacerbates the non-stationarity problem already faced in the fully observable multi-agent case.

2.3 COOPERATIVE MULTI-AGENT REINFORCEMENT LEARNING

Finally, in the following section, details on multi-agent reinforcement learning and its deep instantiation are given, together with an explanation of some more advanced concepts (like coordination graphs or difference rewards) that are investigated in further detail in this work. Also, a brief survey of some popular and known deep MARL algorithms mentioned in the following is provided.

2.3.1 *Centralized Controller vs. Independent Learners*

As already discussed in Section 1.2, there are two different ways of representing and learning the team of agents: on one hand, the multi-agent system can be seen as a large, single-agent problem, where a single *centralized controller* learns either the joint policy $\pi(h,a)$ or the joint action-value function $Q^\pi(h,a)$ and uses it to control all of the agents in the team. This solution is however often impractical, as communication in the environment could be limited or entirely unavailable, rendering difficult (or even impossible) to broadcast the actions chosen by the centralized controller to the agents that have to execute them. Also, tolerance against possible faults in the system is scarce: if the centralized controller fails, none of the agents is capable of doing anything on its own. Moreover, this solution does not scale well with the number of agents in the multi-agent system [24, 171]: when introducing more agents, the size of the joint action set $|A|$ increases exponentially with respect to the number of these, thus complicating the learning of an optimal behaviour; and in some situations the number of agents present in the system may even dynamically change over time, therefore not allowing for a fixed controller structure. All of the above limitations drastically reduce the applicability of a centralized controller in practise, although it may seem compelling because of the convergence guarantees of many standard single-agent reinforcement learning algorithms that could be applied straight out of the box [162, 167, 142, 69].

On the other hand, each autonomous agent could be modelled independently from the others, in the so-called *independent learners* (IL, also known as *decentralized*

learners) approach [146, 83, 24, 171], and learned using a single-agent reinforcement learning algorithm on each agent individually. This formulation of the problem looks compelling and offers various benefits: learning the behaviour of each single agent individually, either in the form a local policy π^i or by defining a local action-value function $Q^{\pi^i}(h^i,a^i)$ (or $Q^{\pi^i}(s,a^i)$ if learning in an MMDP), is a simpler problem than learning the global behaviour of the entire team all at once, as its size does not scale exponentially with the number of agents but is instead fixed; also, each agent can now autonomously take its own decisions without having to rely on any centralized component, thus being highly resistant against possible faults in the system. Moreover, adding new agents in the team would just require to learn their own behaviours, rather than having to learn the entire centralized controller from scratch.

However, these benefits does not come for free: now each autonomous agent is completely oblivious to what the other agents are doing or perceiving, and treats them simply as part of the environment. This problem is even exacerbated by the fact that the numerical feedback used to inform the agents are usually not informative enough to extrapolate this information, thus leading to a *non-stationarity* of the environment from the perspective of each agent [83, 24, 171, 108]: the other agents are learning as well, possibly changing the way they interact with the environment in a certain situation over time. Therefore, an agent performing the same action in the same situations may experience completely different outcomes depending on what the others agents did in turn. The loss of this stationarity breaks down all of the convergence guarantees of single-agent reinforcement learning algorithms, so the resulting behaviours may be arbitrarily bad.

2.3.2 *IL Pathologies*

Although compelling and easy to implement, independent learners have to face a series of limitations, mainly stemming from the fact that basically each agent learns its own behaviour totally ignoring the other learning agents, thus rendering the environment non-stationary from the protagonist agent perspective.

This non-stationarity induces a series of *learning pathologies* [106, 163] that independent learners have to overcome in order to achieve good coordinated behaviours:

- Miscoordination: this happens when two or more agents fail at coordinating on one of the many available optimal joint actions, thus practically resulting in a lower reward than expected;
- Relative overgeneralization: this pathology occurs when agents are pushed towards favouring a suboptimal behaviour because this gives, on average, a higher reward than the others, although they could have obtained an higher value if perfectly coordinating on a different behaviour. This happens because the concurrent learning and the exploration process of the other agents shadows

the optimal strategy (equilibrium shadowing), rendering it less attractive than a suboptimal one;

- Alter-exploration: the probability of at least one agent exploring the environment rather than acting optimally can fool the other agents into underestimating its optimal actions because of the noise coming from such an exploration;
- Moving target: updating the agents behaviour in parallel renders the used transitions deprecated, and thus undermines the used single-agent reinforcement learning algorithm convergence properties;
- Deception: in sequential settings, errors coming from the above pathologies can propagate through the agents' updates, thus reducing the chance of transitioning to higher-return situations.

2.3.3 *Centralized Training-Decentralized Execution*

A technique to mediate the gap between the benefits of a centralized controller and independent learners is that of the *centralized training-decentralized execution* (CTDE) framework [70, 108]. Although having agents that can act in a decentralized fashion is useful (an often required) because of their many benefits, learning a centralized controller that is thus capable of coordinating the agent is a great improvement in many cooperative setting. The idea of CTDE is to get the best of both worlds: training is often carried out in simulation, even when training for real world problems, where data and simulations are available at a low cost. Therefore the agents are allowed to access all of the information they need from the rest of the team during this training phase, being able to learn and use any required centralized component. During execution instead, when the agents have to interact with the environment and choose an action to perform, they are restricted to rely solely on local information, thus achieving independent execution.

CTDE has gained a lot of success in modern MARL algorithms [139, 115, 135, 158, 40, 10, 160, 181] because the expectation is that, although acting independently and thus still suffering from the non-stationarity introduced by conditioning only on local information, each agent can benefit from the centralized information that it has been exposed to during the training phase and use these to learn better coordinated policies (although some studies prove that this is not always the case [80]).

2.3.4 *Factorizations*

The factorization framework presented in Section 2.2.3 has been adapted to sequential settings where the environment has a state s [10, 179]. Similarly to the stateless setting, a (hyper-)graph can be defined to model the influence of each agent on the others, in

which the nodes represent the agents in the team and (hyper-)edges $e \in \mathcal{E}$ connect agents who are influencing each other. The joint action-value function $Q(s,a)$ can thus be decomposed as:

$$Q(s,a) = \sum_{e \in \mathcal{E}} Q^e(s^e,a^e), \tag{2.22}$$

where s^e is the portion of the entire environment state that affects the decision of the agents in factor e (similar to a set of local observations $o^e = \langle o^i \rangle_{i \in e}$, but containing all of the useful information from the true state), and $a^e = \langle a^i \rangle_{i \in e}$ is the local joint action as in the stateless case. Decomposing a problem in such a way allows us to learn smaller terms compared to a centralized controllers, as each factor comprises only a subset of all the agents.

Unfortunately, not all the problems exhibit such a locality of interaction, and indeed there exist many [177, 151, 123] in which the choice of each agent is directly influenced by the entire team, and thus a direct application of factorization is not possible. However, it is still possible to resort to an approximate factorization, defined similarly to the stateless case by superimposing a set of factors $\mathcal{E}$ over the team and use it as:

$$Q(s,a) \approx \hat{Q}(s,a) = \sum_{e \in \mathcal{E}} \hat{Q}^e(s^e,a^e). \tag{2.23}$$

This way, an approximation of the true Q-function is learned based on approximated local terms $\hat{Q}^e$, trading off some accuracy with an easier learning problem. However, such an approximation may be arbitrarily bad, although it is common belief in the research community that applying an approximate factorization is usually useful and results in a good representation [10, 179]. At the current stage however, a systematic study of this has never been performed.

The idea of using factorization is not new, and it has been proposed to model those problems that presented such a decomposed structure intrinsically [50, 53, 51, 125, 118]: if an agent can only be affected by the decisions of a small subset of other agents, it is enough to model these together into one component rather than modelling the entire team at once. Moreover, this idea has gone further, and has been extended also to settings that are not presenting such properties, for example in settings with sparse agents interactions depending on the system state [67, 68, 84, 99, 102, 101, 97, 103, 104]. In such, a factorization works by representing an approximation of the centralized structure, that can thus be learned efficiently at the expense of some introduced error. This idea has enabled application of factored approaches to more complex and general problems [151, 71]. Deep multi-agent reinforcement learning has also widely applied this idea during the last years, resulting in a number of different algorithms [139, 115, 135, 158, 175] that proved capable of achieving better performances on many difficult multi-agent problems.

It is to note that both centralized controllers and independent learners are limit cases of (approximate) factorizations, namely:

- A centralized controller corresponds to the coordination graph with a single hyper-edge connecting all of the agents,
- Independent learners, on the other hand, can be seen as a coordination graph with no edges at all, and thus each agent is not directly coordinating with any other.

2.3.5 *Multi-Agent Credit Assignment & Difference Rewards*

In a cooperative multi-agent system, at each time step all of the agents are rewarded with the same numerical feedback by the environment in response to their actions. This way, however, none of the agent has a clear and easy way to assess how its own local decisions had an impact on the overall team performance, and how much of the feedback depended on these. This issue is called *multi-agent credit assignment* (MACA) [20, 94, 178, 168], and it is a key problem unique to this kind of settings. If not carefully considered, this problem could lead agents towards learning sub-optimal behaviours, being misled into thinking that their actions are useful and contribute towards the team objective while instead they could adopt a different strategy and improve the performance even more. As an example, let's consider a football match, in which one of the two teams just won a trophy final with a score of 4-3. All of the players in the team are rewarded with the trophy medal and part of the prize money, but from the score it is clear that, while the strikers played a really good game and scored 4 goals, the defenders and the goalkeeper have not really been able to do their job properly, allowing the opponents to score 3 goals in turn. However, they all won the game and got awarded together, so that the whole team could be tricked into thinking that each of them has played at his best. However, if the defenders and the goalkeeper were capable of figuring out their own mistakes and play differently, preventing their opponents from scoring so many goals, their team could have reached a much easier and less tight victory.

A family of algorithms that has been designed to tackle multi-agent credit assignment is that of *difference rewards* (DR) methods [169, 168]: the idea of these methods is to provide each agent with an individual shaped reward value, rather than using the shared one provided by the environment, to use as the learning signal. This shaped value is computed in a way such that the contribution of each agent to the overall performance of the team is marginalized out in some way, so that the agent is capable of assessing the specific merit of its chosen action against some fixed baseline value.

As for the way in which such a marginalization is computed, various algorithms have been proposed so far [178, 169, 114, 26, 25]. One of the most popular is called *wonderful life utility* (WLU) [169, 94], which is based on the idea of using a *default*

action a_d^i for each agent i. Such a default action replaces the action a^i in the joint action a that has really been selected at state s, thus resulting in a different reward value than the perceived one r. This way, each agent can now compute an individual reward value as:

$$\Delta R^i(s,a) = r - R(s,\langle a^{-i},a_d^i\rangle), \tag{2.24}$$

where a^{-i} is the joint action of all the other agents but agent i and $R(s,a)$ is the reward function. Each agent is going to compute a different ΔR^i. Also, for each agent the choice of a different local action a^i would result is different sampled r, but the same baseline value $R(s,\langle a^{-i},a_d^i\rangle)$, as this does not depend on the agent's selected action a^i. Therefore, if an action a^i increases r, this is not increasing the baseline value, and thus results in higher shaped values (the two values are aligned) [1]. However, one key difficulty of such a method is the choice of the default action: no clear way of selecting one local action over the others exists, and the meaning of such a choice is usually unclear but still rather fundamental for the method to work well, as different actions would result in different difference rewards values and influence the learning process.

To avoid such a choice, the *aristocrat utility* (AU) method has been proposed [169, 94]. Instead of selecting a default action to marginalize out the agent contribution, now the policy $\pi^i(s,a^i)$ of the agent is taken into account, so that the baseline is computed as the expected reward value obtained by sampling an action from this policy:

$$\Delta R^i(s,a) = r - \mathbb{E}_{b^i\sim\pi^i(\cdot|s)}\left[R(s,\langle a^{-i},b^i\rangle)\right]. \tag{2.25}$$

This method entirely avoids such an ambiguous choice by using the agent's current policy, but this dependence also comes at a cost: using such a method to learn with value-based algorithms (for example using independent Q-learning agents [146]) may lead to instabilities, as changes in the Q-values used to define the policies corresponds in turn to changes to the target values computed by the aristocrat utility itself and the selected action a [169].

Nonetheless, both aristocrat utility and wonderful life utility require complete knowledge of the reward function $R(s,a)$ in order to compute the shaped values. However, in classical MARL problems, this is a limiting requirement, as the reward function is usually unknown beforehand (as well as the transition function), and thus such methods found a barrier to a possible broader application in practical problems (although some extensions propose to approximate such a function locally [26, 25]).

2.3.6 *Multi-Agent Policy Gradients*

One of the most popular family of algorithms in multi-agent systems is that of *multi-agent policy gradient* (MAPG) methods [112], that naturally allow us to learn decentralized agents that can act independently. The core idea is akin to that of standard, single-agent policy gradient methods: each agent optimizes its own policy π_{θ^i}, parametrized by θ^i, by following the gradient $\nabla_{\theta^i} V(\theta)$ that maximizes the expected team future return $V^{\pi_\theta}(s)$ (or $V^{\pi_\theta}(h)$ in a Dec-POMDP setting):

$$\nabla_{\theta^i} V(\theta) = \nabla_{\theta^i} \sum_{s \in S} d^{\pi_\theta}(s) \sum_{a \in A} \pi_{\theta^i}(a^i|s) Q^{\pi_\theta}(s,a), \tag{2.26}$$

where $Q^{\pi_\theta}(s,a)$ ($Q^{\pi_\theta}(h,a)$ when the state is not observable) is the joint action-value function induced by the joint (parametrized) policy π_θ, as in a cooperative setting the outcome of the environment is not solely determined by the actions of the considered agent i, but also by those of the other agents.

With a similar result to that of the policy gradient theorem [142, 141], such gradients can be approximated via sampling rather than estimating the true expectation [142, 112], so that the update to the policy parameters θ^i for agent i can be computed as:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{\sum_{t=0}^{T-1} Q^{\pi_\theta}(s_t,a_t) \nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i|s_t)}_{g^i}. \tag{2.27}$$

As in the single-agent case, the Q-function is unknown to the agents, even more so now that it depends on the actions of the other agents as well, therefore each agent needs a way to approximate such values before being able to compute its own policy gradients. The easiest and most straightforward way is to use the cooperative Monte-Carlo return $G_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$, that is common to all agents. Such an approach is called *distributed policy gradient* [112]:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t G_t \nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i|s_t)}_{g^i}. \tag{2.28}$$

However, the same limitations already faced by the single-agent version of this algorithm (REINFORCE [167]) are also present (and strengthen) in the multi-agent version: the return suffers from high variance and thus can hinder the policy learning process, although the use of a baseline function $b(s)$ can help [47].

Another possibility is for each agent i to approximate the joint Q-function by learning a *local critic* $Q_{\omega^i}(s,a^i)$ (or $Q_{\omega^i}(h^i,a^i)$) [40, 80, 126], with parameters ω_i, that is solely conditioned on local actions a^i (and local histories h^i if the true state s is not observable) and does not require any information from the other agents. It is easy

to note that this approach corresponds to having independent learners each using (any variant of) the actor-critic algorithm [69, 87], and it is thus named *independent actor-critics* (IAC). As well as in single-agent actor-critic methods, local value-functions $V_{\omega^i}(h^i)$ can be used to replace local action-value functions.

Finally, the CTDE paradigm can be adopted to retain the decentralized execution but learn a *centralized critic* $Q_\omega(s,a)$ or $Q_{\omega^i}(h,a)$, only required during training, to inform the policy gradients. This approach has found a solid interest in recent MARL literature [40, 63, 79], although the learning of the centralized component may be difficult and in general using a centralized critic does not guarantee an improvement in performance [126, 80]. Also in this case, a centralized value-function $V_\omega(s)$ (or $V_\omega(h)$) is a suitable choice as a critic to replace the Q-function, and the advantage function $A(s,a)$ can be used as well to further reduce variance in gradient estimates.

In any case however, the critic(s) is learned by minimizing the same TD-error used for the single-agent algorithm (presented again for the Q-function in the fully observable case, but similar expressions hold for the value-function or under partial observability as well):

$$\delta = r + \gamma Q_\omega(s',a') - Q_\omega(s,a), \tag{2.29}$$

$$\delta^i = r + \gamma Q_{\omega^i}(s',a^{i'}) - Q_{\omega^i}(s,a^i). \tag{2.30}$$

2.3.7 *Deep Multi-Agent Reinforcement Learning*

In this section some of the most popular deep MARL algorithms are introduced, with a particular focus on those algorithms that are related to the research topics presented in the remained of this work. Deep MARL has been used in a variety of different contexts, and different kinds of techniques have been investigated so far. This section is not intended to provide a complete survey on the field (for which [59, 108] are two recent and extremely valid options), and many interesting and flourishing topics, like the use of agents communication [38, 138] or opponent modelling [39], are completely overlooked (although these are not less important nor interesting that the algorithms presented here). The aim of this brief selection is rather to provide the basis on some commonly known algorithms, as well as showing the connections of these to the problems and research questions that this work aims at addressing.

VDN. Inspired by [50, 53, 67], one of the first and most popular value-decomposition method are *Value-Decomposition Networks* (VDN) [139]. In this, each agent i independently learns a local action-value function $Q^i_{\theta^i}(h^i,a^i)$, that it uses to select actions. These independent networks are however all optimized together to minimize the off-policy TD-error [162, 7] of the reconstructed joint Q-function, that is obtained as the sum of such components:

$$Q(h,a) = \sum_{i \in D} Q^i_{\theta^i}(h^i,a^i). \tag{2.31}$$

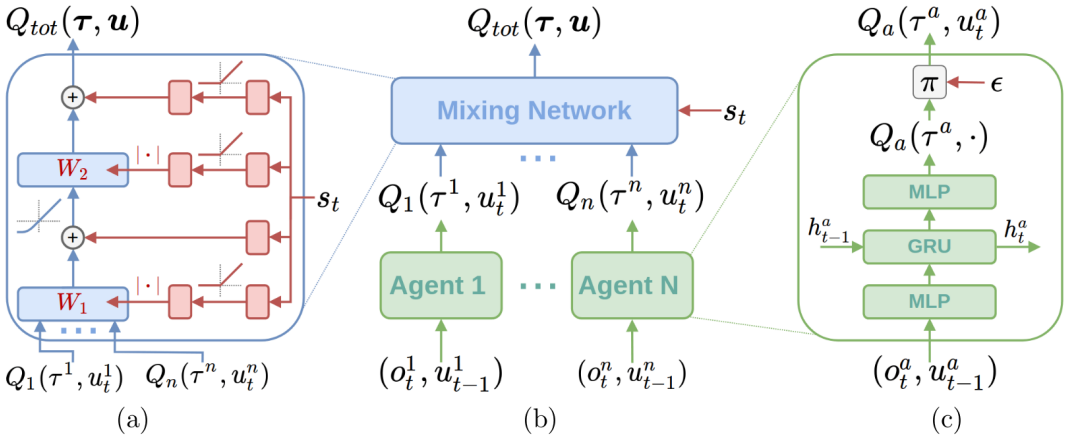


Figure 2.5: Schematic representation of the QMIX algorithm, with enlarged details of the mixing network (left) and an agent Q-network. Source: Rashid et al. 2018.

Additivity of local terms is a sufficient condition (but not strictly required) to ensure the so-called *individual-global maximum* (IGM) property [135, 158], under which the joint action that maximizes the joint Q-function is the same that is obtained by individually maximizing each local term:

$$\arg\max_{a \in A} Q(s,a) = \begin{pmatrix} \arg\max_{a^1 \in A^1} Q^1(s,a^1) \\ \vdots \\ \arg\max_{a^n \in A^n} Q^n(s,a^n) \end{pmatrix}, \forall s \in S, \tag{2.32}$$

As the maximization step over the joint action-value function may be computationally expensive (as it is over an exponential number of joint actions), respecting the IGM property means that the same maximization can be solved individually over some smaller sets (the local action sets of the agents), and thus allow for faster computation.

QMIX. The linear decomposition imposed by VDN is simple but limited in the class of problems that it can accurately represent. To extend such a class, QMIX [115] proposes a new decomposition based on the concept of monotonicity of the local terms:

$$\frac{\partial Q}{\partial Q^i} > 0, \forall i \in D. \tag{2.33}$$

This condition is more general than that of VDN to ensure the IGM property (sufficient although not strictly required), and allows for a non-linear combination of the agents' Q-values. In QMIX, this non-linear combination is done with a learnable *mixing network* f_ω, that is additionally conditioned on the full state s following the CTDE paradigm:

$$Q(h,a) = f_\omega(s, Q_{\theta^1}^1(h^1,a^1), \ldots, Q_{\theta^n}^n(h^n,a^n)). \tag{2.34}$$

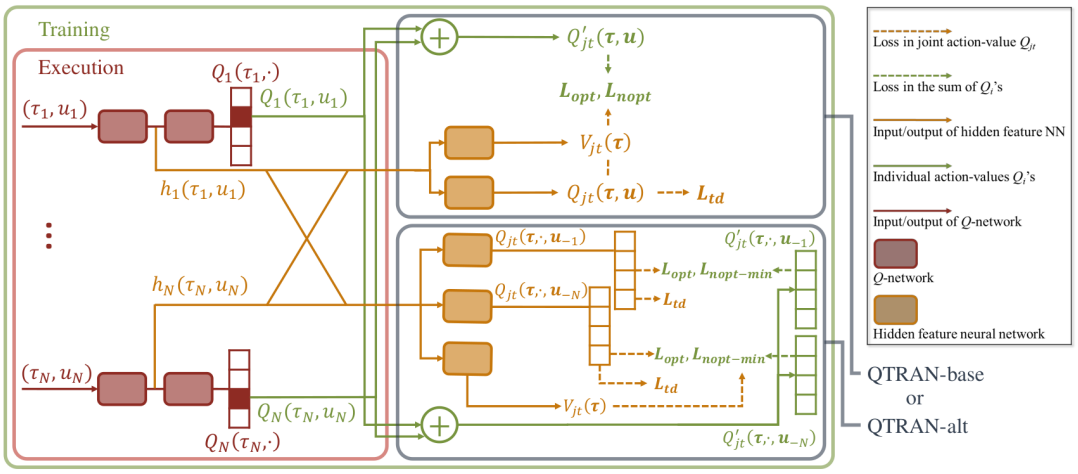


Figure 2.6: Schematic representation of the QTRAN algorithm. Source: Son et al. 2019.

In order to ensure that the monotonicity constraint is always satisfied, the weights and biases for each layer of the mixing network $\omega = \{W_l, b_l\}_{l=1}^{L}$ have to be non-negative, and thus are produced by a set of hyper-networks [55], one for each layer.

QTRAN. Although more general than VDN, also QMIX is limited in its representational capacity. QTRAN [135] aims at addressing this limitation by representing the entire class of factorizable functions so that the necessary condition for the IGM property is respected:

$$\sum_{i \in D} Q^i(s,a^i) - Q(s,a) + V(s) = \begin{cases} = 0, & a = \bar{a}, \\ \geq 0, & a \neq \bar{a}, \end{cases} \tag{2.35}$$

where $V(s) = \max_{a \in A} Q(s,a) - \sum_{i \in D} Q^i(s,\bar{a}^i)$ is the value function and $\bar{a}^i$ is the locally optimal action for agent i $\bar{a}^i = \arg\max_{a^i \in A^i} Q^i(s,a^i)$.

QTRAN enforces such a constraint by defining a transformed joint action-value function $Q'(h,a) = \sum_{i \in D} Q^i(h^i,a^i)$, so that the IGM property holds for that because of additivity, and noting that this function, corrected with the value function $V(h)$, corresponds to an affine transformation of the original $Q(h,a)$, and thus that the optimal actions of the two are the same. The algorithm works by learning the individual terms $Q^i_{\theta^i}(h^i,a^i)$, maintaining a joint Q-function $Q_\omega(h,a)$ learned with the above observation, and a value function $V_\psi(h)$. Two different variants of this algorithm are proposed, differing in the way in which the loss function for the whole architecture is done.

DCG. All of the algorithms introduced above are based on an agent-wise decomposition of the joint Q-function. *Deep Coordination Graphs* (DCG) [10] instead, by heavily relying on the idea of "higher-order" factorizations as studied and analysed in the original work in Chapter 3 [16, 17], goes a step further. Here the joint Q-function is divided into a set of individual utilities $Q^i_{\theta^i}(h^i,a^i)$, one for each agent, and a set of pairwise payoff terms $Q^{ij}_{\omega^{ij}}(h^i,h^j,a^i,a^j)$, defined by a given coordination graph [53, 52].

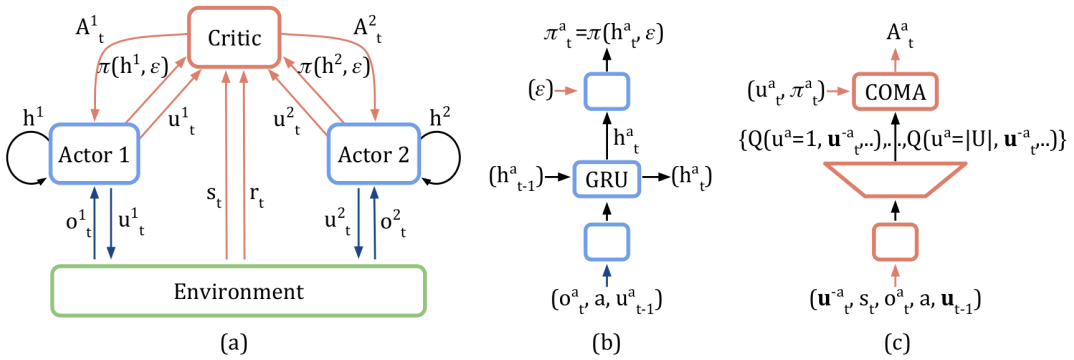


Figure 2.7: Schematic representation of the COMA algorithm: (a) the overall architecture, (b) the actor network, (c) the centralized critic network. Source: Foerster et al. 2018.

Moreover, to further improve sample efficiency, parameter sharing [54] is used, i.e. all the neural networks have the same parameter set. Furthermore, all of the agents utilities and payoff terms are conditioned on a shared embedding function, represented as a recurrent neural network [48, 22] $e_t^i = \text{RNN}_\psi(\cdot|e_{t-1}^i, o_t^i, a_{t-1}^i)$, rather than on agents local histories h_t^i.

The joint Q-function is finally reconstructed as a sum of the agents utilities and the payoff terms:

$$Q(h,a) = \frac{1}{|D|}\sum_{i\in D} Q_\theta^i(e^i,a^i) + \frac{1}{2|\mathcal{E}|}\sum_{ij\in\mathcal{E}}\left(Q_\omega^{ij}(e^i,e^j,a^i,a^j) + Q_\omega^{ji}(e^j,e^i,a^j,a^i)\right). \tag{2.36}$$

COMA. *Counterfactual Multi-Agent Policy Gradients* (COMA) [40] addresses the problem of multi-agent credit assignment by combining decentralized policy gradients with a particular centralized critic, following the CTDE paradigm, that allows for an easy calculation of a difference rewards mechanism.

Such a centralized critic Q_ω receives as an input the state of the environment s and the join action a_t^{-i} for all the agents except agent i, and outputs the Q-values for each of its possible local action $a^i \in A^i$. With such values available in a single pass through the neural network, agent i policy π_{θ^i} is updated by following the individual gradient:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{A^i(s,a)\nabla_{\theta^i}\log\pi_{\theta^i}(a^i|h^i)}_{g^i}, \tag{2.37}$$

where $A^i(s,a) = Q_\omega(s,a) - \sum_{b^i\in A^i}\pi_{\theta^i}(b^i|h^i)Q_\omega(s,\langle a^{-i},b^i\rangle)$ is a *counterfactual advantage* function, that resembles the idea of aristocrat utility [169], but applies the marginalization on the centralized Q-function rather than on the reward function as usual.

The algorithm uses parameters sharing [54], and also optimizes the critic at each time step t using a different formulation of the TD-error based on the TD(λ) algorithm [141, 140]:

$$\delta = y^{(\lambda)} - Q_\omega(s_t,a_t), \tag{2.38}$$

$$y^{(\lambda)} = (1-\lambda)\sum_{k=1}^{\infty}\lambda^{k-1}G_t^{(k)}, \tag{2.39}$$

$$G_t^{(k)} = \sum_{l=1}^{k}\gamma^{l-1}r_{t+l} + \gamma^k Q_\omega(s_{t+k},a_{t+k}). \tag{2.40}$$

This optimization technique, although not strictly related to the underlying idea in COMA, is a crucial implementation detail and even the authors of [40] states its importance and that of a well-tuned parameter λ.

MADDPG. MAPG methods have been also applied to continuous action problems (the set of actions for each agent A^i is continuous and infinite rather than being discrete and finite), and *Multi-Agent Deep Deterministic Policy Gradients* (MADDPG) [79] has been proposed specifically for this kind of problems. Moreover, MADDPG allows us to solve also non-cooperative problems, as it does not require a shared reward signal in order to work.

The idea of this algorithm is that DDPG [132, 77] can be applied to each agent independently to train a deterministic policy $\mu_{\theta^i}(o^i)$ (policies in [79] are reactive and condition on the agents' observations o^i rather than on their complete histories h^i as usual) by using a individual centralized critic $Q_{\omega_i}(o,a)$, so that the resulting policy gradients are:

$$\nabla_{\theta^i}V(\theta) = \mathbb{E}_{o,a\sim\mathcal{D}}\left[\nabla_{\theta^i}\mu_{\theta^i}(o^i)\nabla_{a^i}Q_{\omega^i}(o,a)|_{a^i=\mu_{\theta^i}(o^i)}\right], \tag{2.41}$$

where $\mathcal{D}$ is a replay buffer to store previously encountered transitions [86, 77]. Each centralized critic $Q_{\omega^i}(o,a)$ is trained to minimize the off-policy TD-error:

$$\delta^i = r^i + \gamma Q_{\omega^{i-}}(o',a')|_{a'=\mu_{\theta^-}(o')} - Q_{\omega^i}(o,a), \tag{2.42}$$

where $\mu_\theta(o) = \langle\mu_{\theta^1}(o^1),\ldots,\mu_{\theta^n}(o^n)\rangle$ is the joint policy, and ${\omega_i}^-$ and θ^- are the parameters of the target networks [86, 132] for the agent individual critic and the joint policy respectively. The centralized critics should ensure a stable learning to the decentralized agents, that should thus converge to an equilibrium.

3 ANALYSING FACTORIZATIONS OF ACTION-VALUE NETWORKS FOR MULTI-AGENT REINFORCEMENT LEARNING

It is a widespread idea in the multi-agent community that learning the action-value function with a factored method should in general result in a good approximation (or it can be improved by introducing additional components, similarly to what happens when expanding a function approximation via the Taylor series), although this idea has never specifically been tested in a systematic way. Recently, and with the advent of multi-agent deep reinforcement learning, a novel interest flourished around the idea of factorization: many algorithms have been proposed that exploit this approach [139, 115, 135, 158, 175, 111, 10, 76, 137, 160, 181, 89], achieving robust performances in many difficult multi-agent problems and rendering factorization one of the most used and interesting ideas in modern MARL. As already discussed, most of these algorithms actually employ only a *single-agent decomposition* of the action-value function, where the centralized controller is divided into components representing only a single agent, while the idea of "higher-order" decompositions has remained mostly unexplored. With the growing interest in this technique from deep MARL, it is of the foremost importance to understand and assess the true benefits of factorizations on a wide set of different cooperative multi-agent settings, to investigate their general applicability, as well as to highlight the potential pitfalls that these may have on certain problems. Such an analysis could potentially inform and direct future research on what aspects of factorizations could be exploited to improve learning in multi-agent systems, and foster a wider application of such a technique to a broader set of problems.

In this chapter, centralized learning of factored value-based MARL approaches for cooperative multi-agent systems is investigated. Although the usual focus of MARL is on decentralized execution [139, 115, 40, 82], proper centralized learning is still useful in cases in which centralized execution is also available or as a mean to inform decentralized agents under the centralized training-decentralized execution (CTDE) learning framework [70], for example when learning a centralized critic to inform decentralized policies [40] or in methods that involve bootstrapping [162] or message passing schemes [67]. The learning capacity of these various factored approaches is examined by studying the accuracy of their learned Q-function approximations $\hat{Q}$. Motivated by the deep MARL popularity of factored approaches, neural networks are used to represent the different components of the investigated methods. The main contribution consists of a wide set of diverse experiments across different axes, and an in-depth analysis of the results to assess the potential benefits of these methods, as well as their possible drawbacks. Amongst the others, this chapter aims

at investigating the following points:

- How do factored methods compare to baseline algorithms,
- Impact of factors size on the learned representations,
- Scalability to larger systems,
- Sample efficiency,
- Effects of using an exploratory policy.

To minimise confounding factors, the investigation is restricted to *one-shot* (i.e., non-sequential) problems [105] and a completely exploratory policy (i.e., uniform sampling of the actions). Specifically, the learning power of various network architectures is investigated on a series of one-shot games that require a high level of coordination. Some of these games have an underlying factored structure (that is not assumed to be known in advance) and some do not. Despite their simplicity, these games capture many of the crucial problems that arise in the multi-agent setting, such as an exponential number of joint actions.

Problem Statement: Given the original action-value function $Q(a)$ and a learned representation $\hat{Q}(a)$, the interest is on investigating the quality of this learned representation, both in terms of action ranking, i.e.,

$$\sigma(\mathfrak{R}(Q),\mathfrak{R}(\hat{Q})),$$

where σ is a similarity measure and $\mathfrak{R}$ is a partial ordering of the joint actions according to their action-values, so that the learned function can reliably be used for decision making; and in terms of reconstruction error of the representation, computed using the mean squared error (MSE):

$$MSE = \frac{1}{|A|}\sum_{a\in A}(Q(a) - \hat{Q}(a))^2.$$

As the results show, factored methods prove extremely effective on a variety of such games, achieving correct reconstructions even on those games that do not present a true underlying factored structure, and outperform both independent learners and centralized approaches in terms of learning speed. These benefits are even more apparent when the size of such systems grow larger, and a completely centralized solution proves impractical or even infeasible. Thus, an empirical evaluation to assess the accuracy of various representations in one-shot problems is key to understand and improve deep MARL techniques, and the provided takeaways can help the community in taking informed decisions when developing solutions for multi-agent systems. Additional links to standard sequential MARL are also discussed in Section 3.3, as well as depicting some possible future directions to further clarify the understanding of action-value function factorizations in this setting.

3.1 INVESTIGATED ACTION-VALUE FACTORIZATIONS

Most current value-based deep MARL approaches (of which a noticeable exception is [10]) are either based on the assumption that the joint action-value function $Q(s,a)$ can be represented efficiently by a single neural network (when, in fact, the exponential number of joint actions can certainly make a good approximation hard to learn), or that it suffices to represent (approximated) individual action-values $Q^i(s,a^i)$ [83]. The aim is to investigate to what degree these assumptions are valid by exploring them in the one-shot case, as well as assessing if higher-order factorizations may result in an improved representation of such functions, while speeding up learning (as only small factors need to be learned). When a problem presents an underlying factored structure, knowing such a structure beforehand and being able to exploit it properly can be of the greatest benefit both in terms of learning speed and accuracy, but it is argued that resorting to an approximate factorization can still be beneficial in many cases.

Neural networks are used as function approximators to represent the various components of these factorizations. In this analysis, two distinct aspects of the problem are varied. Firstly, two learning algorithms are studied, which are described in Section 3.1.1. Secondly, different coordination graph structures are proposed, which capture how the team of agents is modelled, presented in Section 3.1.2. Finally, the full set of investigated games is presented in Section 3.1.3.

3.1.1 *Learning Algorithms*

Here the two different learning algorithms that are investigated in the proposed experiments are introduced. The specific choice of these two was taken because these are highly related to many standard sequential MARL algorithms: the mixture of experts learning rule follows the same idea of standard independent learning approach used by early works [146], while the factored Q-function rule uses a joint optimization process resembling that of value decomposition networks [139], but it is also similar to the QMIX algorithm [115], using a linear constant mixing instead of an additional mixing network.

- *Mixture of experts* [2]: each factor network optimizes its own output $\hat{Q}^e$ individually to predict the global reward, thus becoming an "expert" on its own field of action. The loss for the network representing factor $e \in \mathcal{E}$ is defined as:

$$\mathcal{L}^e(a^e) = \frac{1}{2}\left(Q(a) - \hat{Q}^e(a^e)\right)^2, \tag{3.1}$$

  where $Q(a)$ is the common reward signal received after selecting joint action a and $\hat{Q}^e(a^e)$ is the output of the network for local joint action a^e. As the aim is

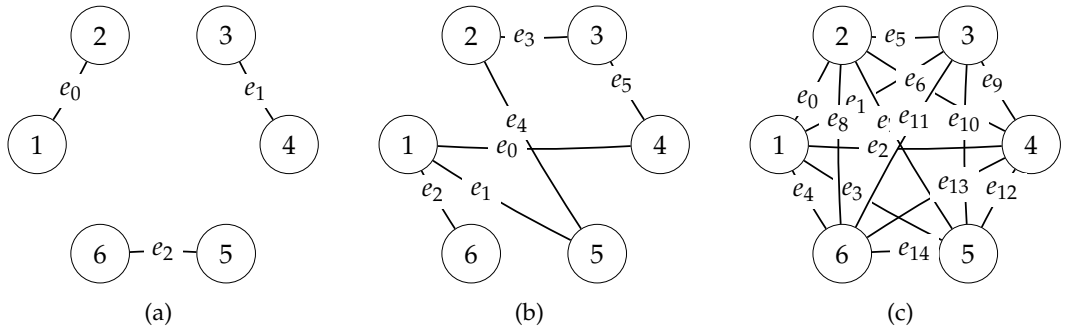


Figure 3.1: Example coordination graphs for: 3.1(a) random partition, 3.1(b) overlapping factors, 3.1(c) complete factorization.

to assess how good the approximated action-value function $\hat{Q}$ is, *after* training the reconstruction obtained from the factors is computed as the mean over the appropriate local Q-values (the "opinion" of each expert is weighted equally):

$$\hat{Q}(a) = \frac{1}{|\mathcal{E}|} \sum_{e \in \mathcal{E}} \hat{Q}^e(a^e), \forall a \in A. \tag{3.2}$$

- *Factored Q-function* [51, 139]: the algorithm jointly optimizes the factor networks to predict the global reward as a sum of their local Q-values $\hat{Q}^e$. The loss for a given interaction is identical for all factor networks:

$$\mathcal{L}(a) = \frac{1}{2} \left(Q(a) - \sum_{e \in \mathcal{E}} \hat{Q}^e(a^e) \right)^2. \tag{3.3}$$

  It is to note that, rather than learning proper action-value functions, the optimization problem in Equation 3.3 learns an utility function for each factor, that does not really represent the values of actions on their own, while Equation 3.1 learns individual Q-values for each factor. After learning, the approximated joint action-value function $\hat{Q}$ is reconstructed by summing the appropriate local Q-values (the components collectively reconstruct the approximation):

$$\hat{Q}(a) = \sum_{e \in \mathcal{E}} \hat{Q}^e(a^e), \forall a \in A. \tag{3.4}$$

3.1.2 *Coordination Graphs*

Four different coordination graphs are proposed and analysed here. Their structures differ both in the number of components and the degree of connection for each agent. This empirical study considers all eight combinations of the two learning rules described above and the four coordination graphs described below.

- *Single agent decomposition*: each agent i is represented by an individual neural network and computes its own individual action-values $\hat{Q}^i(a^i)$, based on its local action a^i. Under the mixture of experts learning rule, this corresponds to the standard independent Q-learning (IQL) approach in MARL [146], in which local agent-wise components are learned, while under the factored Q-function approach this corresponds to value decomposition networks (VDN) [139].

- *Random partition*: agents are randomly partitioned to form factors of size f, with each agent i involved in only one factor[1]. Each of the $|\mathcal{E}| = \frac{n}{f}$ factors has a different neural network that represents local action-values $\hat{Q}^e(a^e)$ for that factor.

- *Overlapping factors*: a fixed number of factors $|\mathcal{E}|$ is picked at random from the set of all possible factors of size f. The sampled set does not include duplicate factors (only distinct components are used) and that every agent i appears in at least one factor. Every factor $e \in \mathcal{E}$ is represented by a different neural network learning local action-values $\hat{Q}^e(a^e)$ for the local joint action a_e. In the proposed experiments $|\mathcal{E}| = n$, to keep the number of networks to be the same than that of the single agent decomposition.

- *Complete factorization*: each agent i is grouped with every possible combination of the other agents in the team $D \setminus i$ to form factors of size f, resulting in $|\mathcal{E}| = \binom{n}{f}$ factors, each represented by a network. Each of these networks learns local action-values $\hat{Q}^e(a^e)$.

A fundamental problem in MARL is that there is currently no method capable of predicting the accuracy of a factored representation on a certain problem in advance. Therefore, assessing the performance and eventual advantages of different structures and approaches is a fundamental step for MARL research, as it can further improve the understanding of these settings and existing algorithms. In this empirical study, factors of size $f \in \{2,3\}$ are primarily considered. The small size of these factors allows us to effectively explore the improvements in the complexity of learning; if the size of each factor is similar to the size of the full team of agents, no significant improvement over a full joint learner in terms of sample complexity and scalability is likely to be achieved (although some experiments on this aspect are also reported in Section 3.2.3).

3.1.3 *Investigated Games*

The proposed methods are investigated on a number of cooperative one-shot games that require a high degree of coordination. Some of these games do not present an

1 If f is not a divisor of n, the random partition factorization would partition into factors that are all of size close to f.

underlying factored structure, while others are truly factored games. For the latter, none of the methods exploits prior knowledge of their true factored structure (but results for the true underlying factorization are also reported, to show the possible benefits when that is known beforehand).

Non-Factored Games

Dispersion Games: In the Dispersion Game, also known as Anti-Coordination Game, the team of agents must divide as evenly as possible between the two local actions that each agent can perform [49]. Think of a town with two different pubs: the inhabitants like both the same, but the two are quite small and cannot contain all the people in the town at once, so the customers have to split up across the two pubs in order to enjoy the situation and not overcrowd them. This game requires explicit coordination, as none of the local actions is good per se, but the obtained reward depends on the decision of the whole team. Two versions of this game are investigated: in the first one the agents obtain reward proportional to their *dispersion coefficient* (i.e., the difference in the number of agents selecting an action rather than the other). The reward function $Q(a)$ for this game with n agents, each with a local action set $A^i = \{a_0,a_1\}$ is:

$$Q(a) = n - \max\{\#a_0, \#a_1\}. \tag{3.5}$$

In the second version, which is dubbed Sparse Dispersion Game, the agents receive a reward (which is set to the maximum dispersion coefficient with n agents: $\frac{n}{2}$) only if they are perfectly split:

$$Q(a) = \begin{cases} \frac{n}{2} & \text{if } \#a_0 = \#a_1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.6}$$

Platonia Dilemma: In the original Platonia Dilemma [60], an eccentric trillionaire gathers 20 people together and tells them that if one and only one of them sends him a telegram by noon the next day, that person will receive a billion dollars. In this cooperative version the reward is set to the number of agents n and is received by the whole team, not just a single agent. Thus, the reward function for n agents with local action sets $A^i = \{send,idle\}$ is:

$$Q(a) = \begin{cases} n & \text{if } \#send = 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.7}$$

Climb Game: In the Climb Game [163], each agent has three local actions $A^i = \{a_0,a_1,a_2\}$. Action a_0 yields a high reward if all the agents choose it, but no reward if only some do. The other two are suboptimal actions that give lower reward but do not require precise coordination. This game enforces a phenomenon called *relative overgeneralization*, [163] that pushes the agents to underestimate a certain action (in

the example, a_0) because of the low rewards they usually receive, while they could get a higher reward by perfectly coordinating on it. The reward function $Q(a)$ is:

$$Q(a) = \begin{cases} n & \text{if } \#a_0 = n, \\ \frac{n}{2} & \#a_1 + \#a_2 = n, \\ 0 & \text{otherwise.} \end{cases} \tag{3.8}$$

Penalty Game: Similarly to the Climb Game, in the Penalty Game [163] each agent has three local actions $A^i = \{a_0,a_1,a_2\}$. In this game, two local actions (for example, action a_0 and a_2) give a high reward if the agents perfectly coordinate on one of them, but also give a negative penalty if they mix them together. The third action a_1 is suboptimal and gives a lower reward when the team coordinates on it, but also no penalty if at least one of the agents uses it. This game could also lead to relative overgeneralization, as the suboptimal action is perceived as giving a higher reward than the optimal ones on average. The following reward function is used:

$$Q(a) = \begin{cases} n & \text{if } \#a_0 = n, \text{ or } \#a_2 = n, \\ \frac{n}{2} & \text{if } \#a_1 = n, \\ 0 & \text{if } 0 < \#a_1 < n, \\ -n & \text{otherwise.} \end{cases} \tag{3.9}$$

Factored Games

Generalized Firefighting: The Generalized Firefighting problem [100] is an extension of the standard two-agent firefighting problem to n agents. This is a cooperative graphical Bayesian game, so each agent i has some private information, called its local type $\theta^i \in \Theta^i$, on which it can condition its decisions. The combination of the various agents types $\theta = \langle \theta^1, \ldots, \theta^n \rangle$ determines the value of the reward function $Q(a,\theta)$. The team is composed of n firefighters that have to fight possible fires at N_h different houses. Each house j can be burning, F_j, or not, N_j. Each agent i has a limited observation and action field: it can observe only N_o houses (so its local type is $\theta^i \in \{F_j,N_j\}^{N_o}$) and can fight the fire only at N_a houses (the sets of the observed and reachable houses are fixed beforehand and are part of the problem specification, with N_o and N_a being their cardinality respectively). Each house h yields a reward component q^h: if one and only one agent fights the fire at a burning house, that house gives a positive reward $q^h = 2$; if the house is not burning (or if it is burning but no-one is fighting the fire at it) it does not provide any reward $q^h = 0$. The reward function is sub-additive: if two agents fight the fire at the same burning house, this gives a reward $q^h = 3 < 2 \cdot 2$. The overall value of the reward function $Q(a,\theta)$ experienced by agents for a given joint type θ and joint action a is the sum of the rewards given by each house q^h:

$$Q(a,\theta) = \sum_{h \in N_h} q^h. \tag{3.10}$$

Therefore, the optimal strategy for the n agents is to split as evenly as possible across all the burning houses $F_j \in \theta$. If the number of burning houses is more than that of the agents, each agent should attend at a different house and fight the fire there, while if there are less burning houses than agents, the remaining agents should exploit sub-additivity and help their colleagues at already attended houses.

In the experiments, the local types θ^i are not given as an input to the neural networks, but are instead used to increase the size of the local action sets (and the joint one thereby) by considering the cardinal product $A^i \times \Theta^i$ as the new action set for agent i, where the agent choose the action $a^i \in A^i$ and the problem chooses the local type $\theta^i \in \Theta^i$. In practice, this corresponds to individually consider each local action for each possible local type, as if the agents are playing a different game (a different joint type) chosen by the environment every time, and they model the values of their actions on each game separately.

Aloha: In Aloha [97] there is a set of nearby islands, each provided with a radio station, trying to send messages to their inhabitants. A slightly altered one-shot version is instead presented here, in which the ruler of each island wants to send a radio message to its inhabitants, but, given that some of the islands are near one to another, if they all send the message the radio frequencies interfere and the messages are not correctly received by the respective populations. Given that all the rulers are living in peace and they want to maximize the number of received messages by their populations, the reward signal is shared and thus the game is cooperative. It is a graphical game, as the result of each island transmission is affected only by the transmissions of nearby islands. Every ruler i has two possible actions: send a message or not. If he does not send a message, he does not contribute to the total reward. If he sends one and the message is correctly received by his population (no interference occurs) he gets a reward $q^i = 2$, but if he interferes with someone else, he gets a penalty of $q^i = -1$. The common reward that all the rulers receive at the end is the sum of their local contributions:

$$Q(a) = \sum_{i \in n} q^i. \tag{3.11}$$

3.2 EXPERIMENTS

The aim of the proposed analysis is investigating the following research questions:

RQ1: Comparison to baselines: how well can the investigated methods represent the action-value function of different cooperative multi-agent systems (both truly

factored or not)? How do these compare to both independent learners and joint learners?

RQ2: Impact of factors size: how small can the factors of these methods be with respect to the team size? How is the factor size affecting the learned representations?

RQ3: Scalability: how do the compared methods scale in the number of agents?

RQ4: Sample efficiency: how is the sample efficiency of these methods compared to both independent learners and joint learners?

RQ5: Exploratory policy: how do the same investigated methods behave with a non-uniform, time-varying policy used to select actions?

The remainder of this Section is organized as follows: RQ1 is addressed in Section 3.2.2 by comparing the methods against both independent learners and a joint learner on a variety of different games, RQ2 is investigated by selecting one of the games and comparing the effect of using small factors versus larger ones in Section 3.2.3, a couple of games with an increasing number of agents is then investigated in Section 3.2.4 to address RQ3, while RQ4 is tackled in Section 3.2.5. An initial step towards RQ5 is made in Section 3.2.6 and finally, a summary of the results and general takeaways are given in Section 3.2.7.

3.2.1 *Experimental Setup*

Table 3.1 defines the abbreviations and acronyms of the combinations of learning approach, coordination graph structure, and factor size used throughout the analysis (other than where differently stated). In the empirical evaluation, these combinations are investigated on the one-shot coordination games presented above.

	Mix. of Experts	Factored Q
Single agent	M1(=IQL [146])	F1(=VDN [139])
Random partition ($f = 2,3$)	M2R, M3R	F2R, F3R
Complete factorization ($f = 2,3$)	M2C, M3C	F2C, F3C
Overlapping factors ($f = 2,3$)	M2O, M3O	F2O, F3O
True (Factored games only)	MTF	FTF

Table 3.1: Combinations of coordination graphs and learning rules.

A hypothesis is that factored representations, by avoiding the combinatorial explosion in the number of joint actions and allowing for some internal coordination inside each factor, are going to produce representations closer to the original action-value function for these multi-agent problems. These are also expected to be sample efficient due to this small size of the factors, speeding up the required training time and learning good representations faster than the other approaches.

The neural networks of the factored representations are trained to reproduce action-value functions for the detailed cooperative one-shot games, using the loss functions and coordination graph structures described in Section 3.1 as combined in Table 3.1. Although a tabular representation for the agents factors would have worked the same in such a one-shot setting (no large state space to make learning impractical), the focus has been kept on neural network representations to strengthen the connection with modern deep MARL value-decomposition methods [139, 115, 135, 158]. After training, the representation $\hat{Q}(a)$ is reconstructed from the factor components' outputs and compared with the original action-value function $Q(a)$ (complete knowledge of this function is withheld from the networks during training, but only samples corresponding to the selected joint action a are provided at every step) to assess the quality of such a representation, both in terms of action ranking and reconstruction error, as defined in the Problem Statement at the beginning of this chapter.

The same hyperparameters are kept for all the investigated representations to favour a fair comparison of the learned representations: using the same learning rates ensures that no method can learn faster than the others, while using the same structure for all the neural network guarantees that none is given with more representational power. Every neural network has a single hidden layer with 16 hidden units[2] using the leaky ReLU activation function [172], while all output units are linear and output local action-values $\hat{Q}^e(a^e)$ for every local joint action a^e. Given the absence of an environment state to feed to the networks as an input, at every time step these just receive a constant scalar value. The mean square error (MSE) is used as the loss function and RMSprop [46] as the training algorithm with a learning rate of $\eta = 10^{-5}$. For every game, the networks are trained with 100,000 examples (to ensure proper and stable convergence of the learned representations) by sampling a joint action a uniformly at random[3]. Then, the gradient update is propagated through each network e from the output unit $\hat{Q}^e(a^e)$. The loss function minimizes the squared difference between the collected reward $Q(a)$ at each training step and the approximation computed by the networks. After training, the learned action-value function $\hat{Q}$ is compared to the original Q. Also, a baseline joint learner (a single neural network with an exponential number $|A| = |A^i|^n$ of output units) is considered. Every experiment was repeated 10 times with random initialization of weights, each time sampling different factors for the random partition and the overlapping factors factorizations; the averages of these 10 runs are reported.

2 Also, some preliminary experiments were performed with deeper networks with 2 and 3 hidden layers, but did not provide improvements for the considered problems.

3 The choice not to use ϵ-greedy is because the interest is on representing the whole action-value function Q and not just the best performing action at every training step and collecting its reward $Q(a)$, as noted in the Problem Statement.

3.2.2 *Comparison to Baselines*

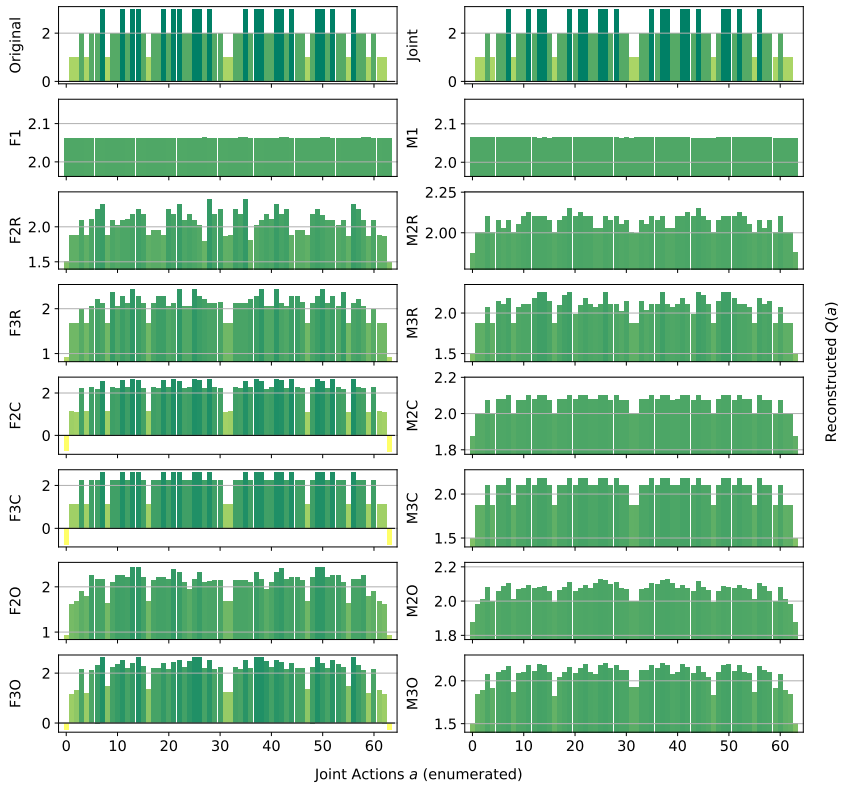
The aim here is to show that factored methods are suitable to represent a wide variety of games, including many that do not present any real underlying decomposition, and that these can perform better than both independent learners and a joint learner baseline. This section starts by discussing the approximate action-value functions obtained by the investigated representations, with the following Table 3.2 summarizing the games that are used and their associated parameters.

Game	n	$\lvert A^i \rvert$	$\lvert A \rvert$	Optimal	Factored
Dispersion Game	6	2	64	20	No
Platonia Dilemma	6	2	64	6	No
Climb Game	6	3	729	1	No
Penalty Game	6	3	729	2	No
Generalized FF	6	2	64 (8192 total)	779	Yes
Aloha	6	2	64	2	Yes

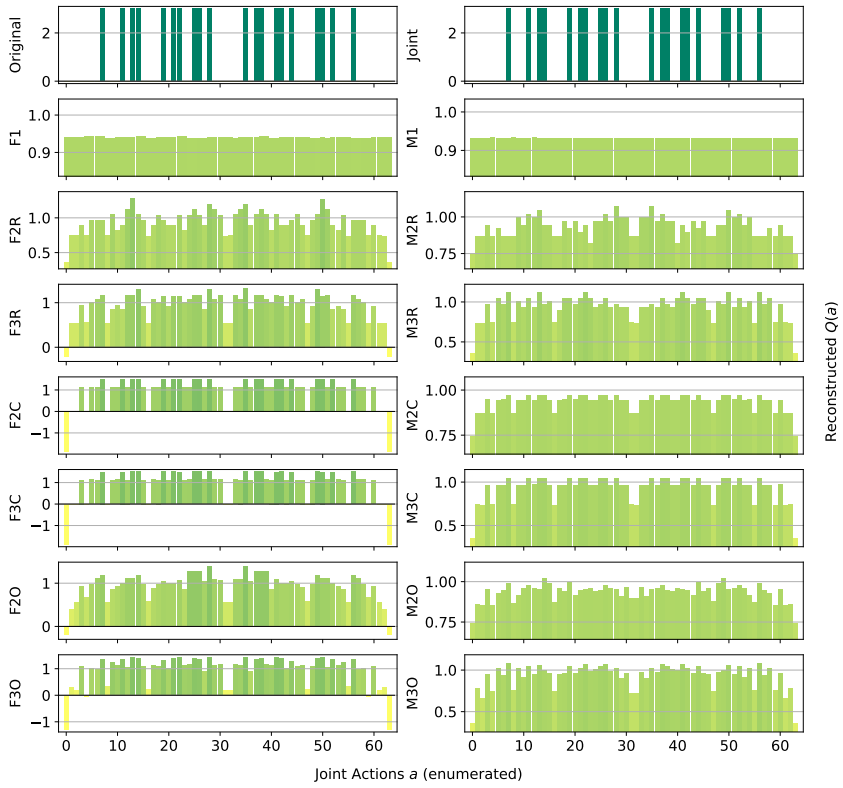
Table 3.2: Details of the investigated games in this section.

In the following plots, the x-axis enumerates the joint actions $a \in A$ and the y-axis shows the corresponding values $\hat{Q}(a)$ for the reconstructed functions, with the heights of the bars encoding the magnitude of the action-values $\hat{Q}(a)$. As defined in the Problem Statement, the quality of the computed reconstructions are analysed considering two aspects: the total reconstruction error of $\hat{Q}(a)$ with respect to the true reward function $Q(a)$, and whether a reconstruction produces a correct ranking of the joint actions. For a good reconstruction, the bars have to have the same relative heights, indicating that the representation correctly ranks the joint actions with respect to their values, and to be of a similar magnitude to those in the original one (the representation can reconstruct a correct value for that joint action). However, reconstruction error alone is not a good accuracy measure because lower reconstruction error does not imply better decision making, as a model could lower the total error by over- or underestimating the value of certain joint actions.

Dispersion Games: Figure 3.2 shows the approximations reconstructed by the proposed coordination graphs and learning approaches for the two variants of the Dispersion Game. Figure 3.2(a) shows that the proposed complete factorizations are able to almost perfectly reconstruct the relative ranking between the joint actions, meaning that these architectures can be reliably used for decision making in this kind of problems. Moreover, the ones using the factored Q-function (F2C and F3C in the plot) are also able to produce a generally good approximation of the various values (expressed by the height of the bars), while those based on the mixture of experts produce a less precise reconstruction: the joint optimization of the former gives an advantage in this kind of extremely coordinated problems.



(a)



(b)

Figure 3.2: Reconstructed $Q(a)$ for 3.2(a) the Dispersion Game, and 3.2(b) its sparse variant.

Smaller factorizations, like the random pairings, are not sufficient to correctly represent this function, probably because a higher degree of connection is required to achieve good coordination. Figure 3.2(b) is similar but in this case the reconstructions are less accurate, with reconstructed action-values that are quite different from the original ones. This is possibly due to the sparsity of the original function, requiring the networks to correctly approximate quite different values with the same output components. In this case, the sparsity of the function to represent fools the approximations into being similar to those of the non-sparse version.

Table 3.3 (as well as the similar ones for the other games) reports the best and worst performing methods on the two variants of this game against a set of different measures that explore the results in terms of the Problem Statement at the beginning of this chapter: the MSE tells us how far is each reconstructed action-value function with respect to the original one, while the number of optimal joint actions found (Opt. Found, i.e. how many of the true optimal actions are also ranked as optimal by such a reconstruction), and the total number of correctly ranked actions (Ranked) points out how reliably these reconstructions can be used for decision making. Methods performing not too bad but also not the best are left out for compactness, for more data and measures on this game (and similarly for the following ones) please see the Appendix.

Model	MSE	Opt. Found	Ranked
Dispersion Game			
Joint	0.00 ± 0.0	20 ± 0	64 ± 0
F1	0.62 ± 0.0	5 ± 2	21 ± 2
F2R	0.52 ± 0.0	8 ± 0	24 ± 1
F2C	0.09 ± 0.0	20 ± 0	64 ± 0
F3C	0.09 ± 0.0	20 ± 0	64 ± 0
F3O	0.19 ± 0.0	13 ± 1	47 ± 3
M1	0.62 ± 0.0	6 ± 1	24 ± 2
M2R	0.56 ± 0.0	8 ± 0	24 ± 1
M2C	0.55 ± 0.0	20 ± 0	64 ± 0
M3C	0.43 ± 0.0	20 ± 0	64 ± 0
M2O	0.56 ± 0.0	10 ± 2	36 ± 4
Dispersion Game (sparse)			
Joint	0.00 ± 0.0	20 ± 0	64 ± 0
F1	1.93 ± 0.0	7 ± 1	37 ± 1
F2R	1.83 ± 0.0	8 ± 0	40 ± 0
F2C	1.41 ± 0.0	20 ± 0	64 ± 0
F3C	1.41 ± 0.0	20 ± 0	64 ± 0
M1	1.93 ± 0.0	6 ± 1	37 ± 2
M2R	1.88 ± 0.0	8 ± 0	40 ± 0
M2C	1.87 ± 0.0	20 ± 0	64 ± 0
M3C	1.74 ± 0.0	20 ± 0	64 ± 0
M2O	1.87 ± 0.0	10 ± 1	44 ± 2

Table 3.3: Best (green) and worst (red) performing methods on the two variants of the Dispersion Game.

It can be observed how the joint learner can easily learn and represent the entire

action-value function for this small setting, resulting in a perfect ranking and a very small error. However, methods using the complete factorization are also able to do so, with the mixture of experts achieving a larger reconstruction error but still a correct ranking of actions, including identifying all of the optimal ones, on both variants of this game. Independent learners instead do not seem able to correctly identify all of the optimal actions, also achieving a very large reconstruction error.

Platonia Dilemma: Figure 3.3 shows the reconstructed action-value functions for the Platonia Dilemma. For this problem, none of the proposed factorizations seems able to correctly represent the action-value function. In fact, while these are perfectly able to correctly learn all the optimal actions (the ones in which only a single agent sends the telegram) at the same level, they all fail to correctly rank and reconstruct the same joint action (the one in which none of the agents sends the telegram). In fact, the unique symmetric equilibrium for the team in this game is that each of them sends the telegram with probability $\frac{1}{n}$, so the agents usually gather more reward by not sending it themselves, but relying on someone else to do so. This results in an 'imbalanced' action-value function in which the high reward is more often obtained, from an agent perspective, by choosing a certain action instead of the other, thus resulting in overestimating one of the joint actions (the one in which all the agents perform the same action, i.e., not sending the telegram).

This imbalance in the reward given by the two actions is probably the cause of the incorrect reconstructions. Thus, for this kind of tightly coupled coordination problems, none of the techniques to approximate action-values currently employed in deep MARL suffices to guarantee a good action is taken, even if the coordination problem is conceptually simple. Table 3.4 reports the best and worst performing methods on this game.

Model	MSE	Opt. Found	Ranked
Joint	0.00 ± 0.0	6 ± 0	64 ± 0
M1	2.80 ± 0.0	5 ± 0	62 ± 0
M2O	2.54 ± 0.0	4 ± 0	61 ± 1
M3O	2.28 ± 0.0	4 ± 0	61 ± 1

Table 3.4: Best (green) and worst (red) performing methods on the Platonia Dilemma.

All of the methods using the factored Q-function learning approach are left out, as these achieve average performances. As already showed by Figure 3.3, here only the joint learner is able to correctly identify all of the optimal actions. The mixture of experts with the overlapping factors are the ones that perform the worst, possibly because the connectivity of this structure is too sparse to help in such an imbalanced reward game.

Climb Game: Figure 3.4 shows the results obtained on the Climb Game. The joint network is not able to learn the correct action-value function in the given training time, due to the large number of joint actions. This highlights again how joint learners

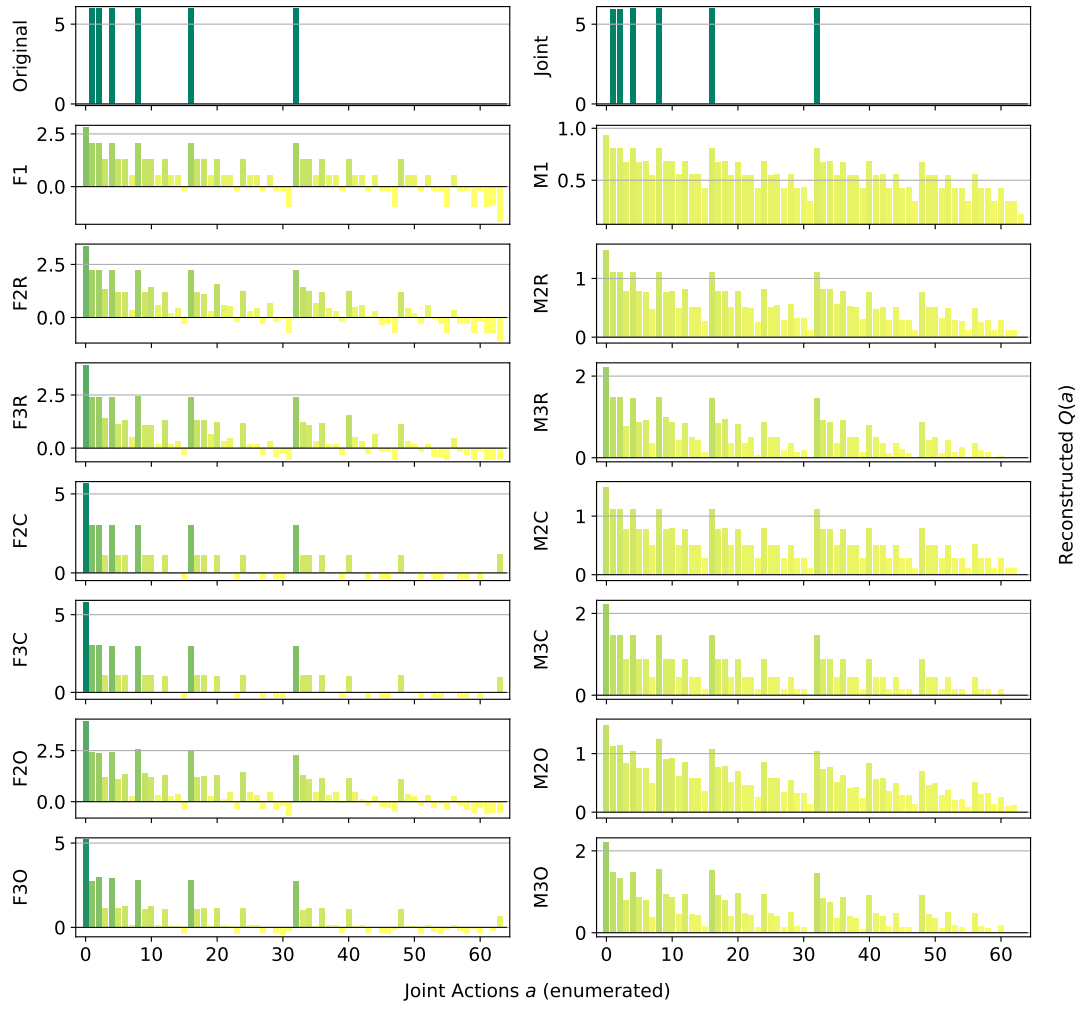
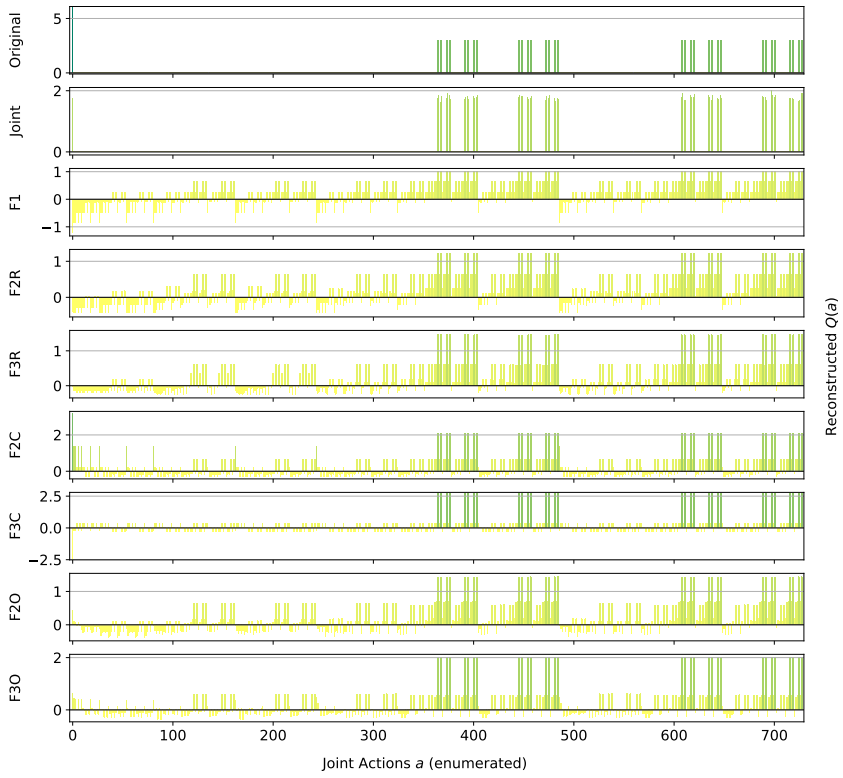


Figure 3.3: Reconstructed $Q(a)$ for the Platonia Dilemma.

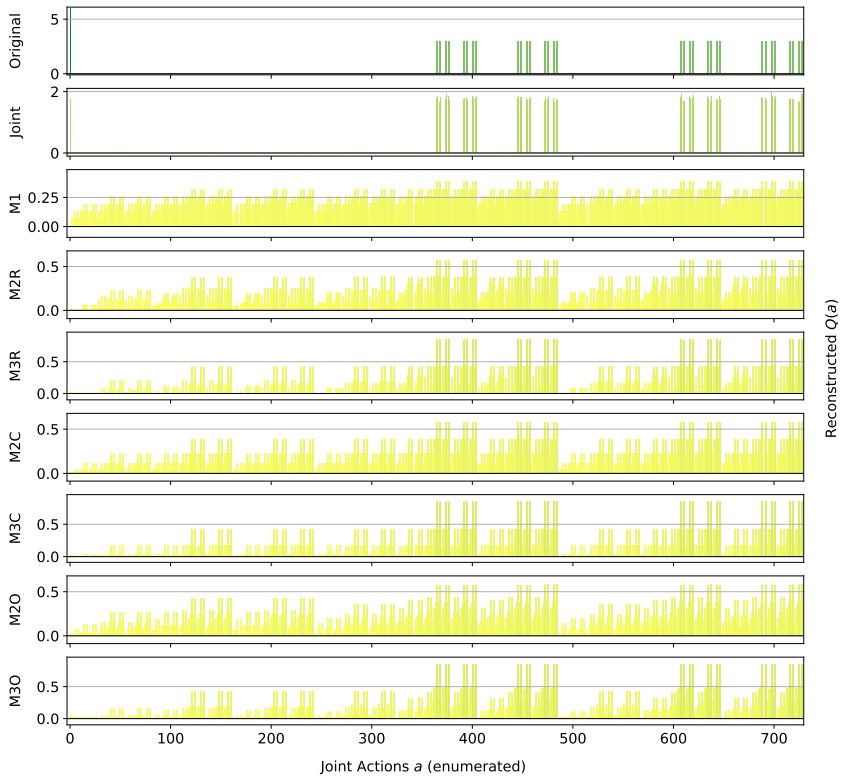
are not suited for this kind of even moderately large multi-agent systems. By contrast, all the other architectures correctly rank the suboptimal actions.

The methods using the factored Q-function and a complete factorization are also able to correctly reconstruct the values for most of the joint actions, as can be seen from the bars. However, only F2C correctly ranks and reconstructs the optimal action (the coordinated one), while even F3C fails to do so and gives it a large negative value. A likely cause for this effect is that, when optimizing the loss function, assigning a negative value to the components forming the optimal joint action reduces the overall mean squared error, even if then one of the reconstructed values gets totally wrong. It can also be observed how the mixture of experts plot looks somewhat comparable to the one for the factored Q-function learning approach, but more 'compressed' and noisy. Table 3.5 reports the best and worst performing methods on the Climb Game.

With a larger joint action space, the joint learner begins to struggle and achieves a larger reconstruction error than some of the factored methods. Interestingly, F2C is the only method capable of identifying the optimal action of this game, where



(a)



(b)

Figure 3.4: Reconstructed $Q(a)$ for the Climb Game: 3.4(a) factored Q-function learning approach, and 3.4(b) mixture of experts learning approach.

Model	MSE	Opt. Found	Ranked
Joint	0.17 ± 0.1	0 ± 0	727 ± 1
F2C	0.25 ± 0.0	1 ± 0	729 ± 0
F3C	0.17 ± 0.0	0 ± 0	726 ± 0
M1	0.71 ± 0.0	0 ± 0	726 ± 0

Table 3.5: Best (green) and worst (red) performing methods on the Climb Game.

also F3C fails. An hypothesis is that this happens because the larger factors push the overall representation to further improve the reconstructed values for the other local joint actions at the expense of these forming the optimal action itself. This points out how, although generally a larger factor size entails a better representation, it may not always be so. On the other hand however, it also shows how small factors can result in a good representation that is also easier and faster to learn.

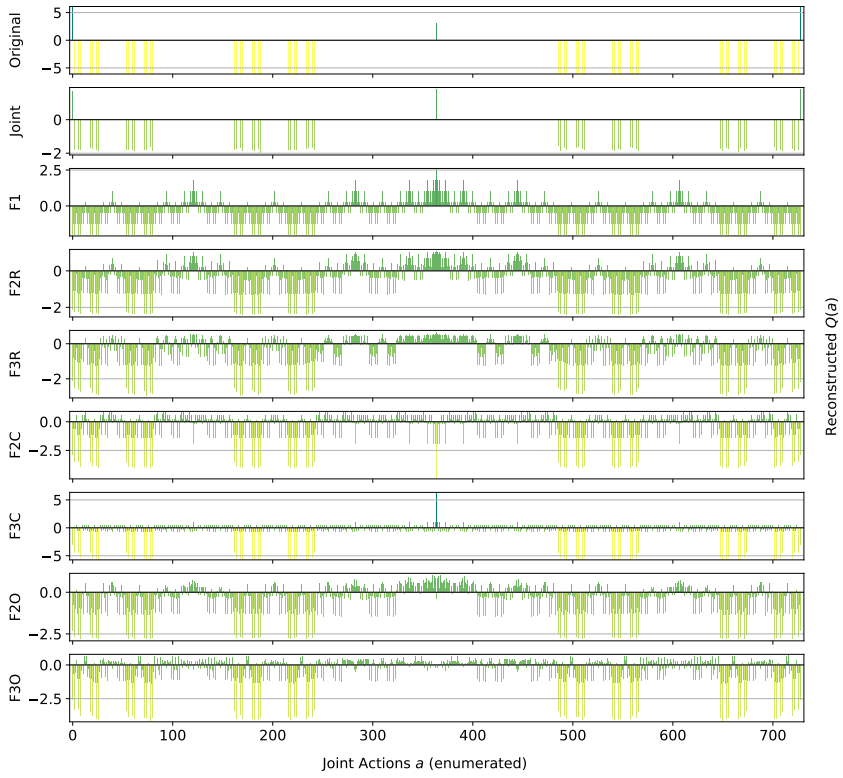
Penalty Game: Figure 3.5 presents the representations obtained by the investigated approximations on this problem. Given the high level of coordination required, all of the architectures using the mixture of experts learn a totally incorrect approximation, biased by the larger number of joint actions that yield a penalty rather than a positive reward.

For this game, none of the architectures can correctly reconstruct the whole structure of the action-value function, and they all fail at the two optimal joint actions (at the two sides of the bar plots). This is probably due to the large gap in the reward values that the agents can receive when choosing one of their coordinated actions: they can get a high reward if all the agents perfectly coordinate, but it is more common for them to miscoordinate and receive a negative penalty, resulting in an approximation that ranks those two joint actions as bad in order to correctly reconstruct the other cases. Furthermore, the suboptimal action is hard to correctly approximate because, similarly to the optimal ones, it also usually results in a smaller reward than the one it gives when all the agents coordinate on it. Only F1 and F3C rank it as better than the other, but surprisingly only F1 is also able to reconstruct the correct value. Table 3.6 reports the best and worst performing methods on this game.

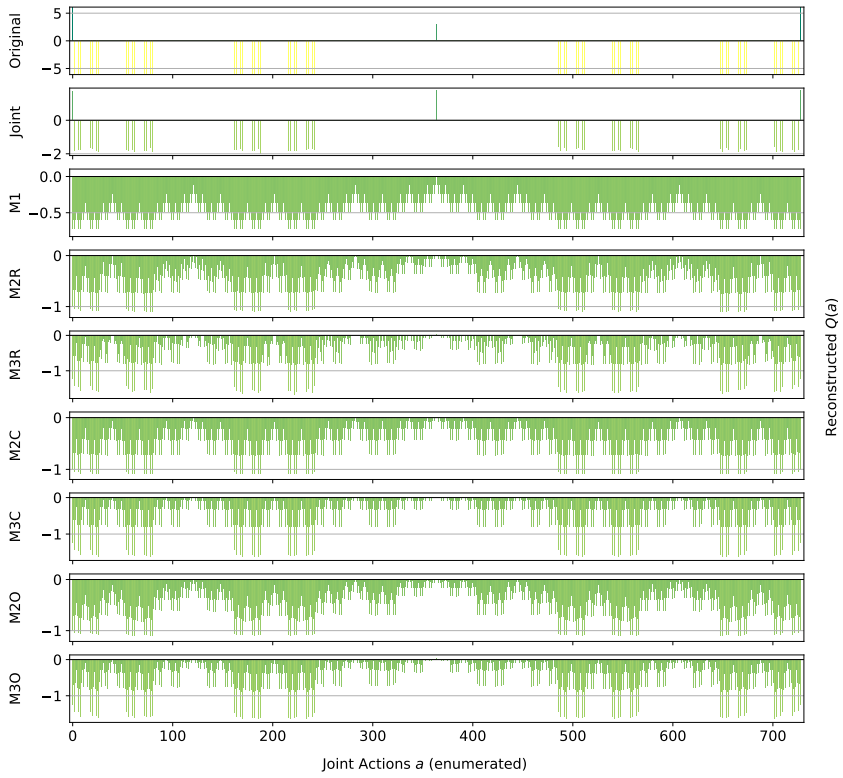
Model	MSE	Opt. Found	Ranked
Joint	1.60 ± 0.4	1 ± 0	727 ± 1
F1	2.18 ± 0.0	0 ± 0	722 ± 0
F2C	1.29 ± 0.0	0 ± 0	722 ± 0
F3C	0.54 ± 0.0	0 ± 0	724 ± 0
F3O	1.27 ± 0.0	0 ± 0	723 ± 0
M1	2.71 ± 0.0	0 ± 0	722 ± 0

Table 3.6: Best (green) and worst (red) performing methods on the Penalty Game.

For this setting as well, the joint learner is struggling to represent the entire action-value function, although it is the only method capable of correctly identify one of the optimal joint actions. All the other methods fail in doing so, even though some



(a)



(b)

Figure 3.5: Reconstructed $Q(a)$ for the Penalty Game: 3.5(a) factored Q-function learning approach, and 3.5(b) mixture of experts learning approach.

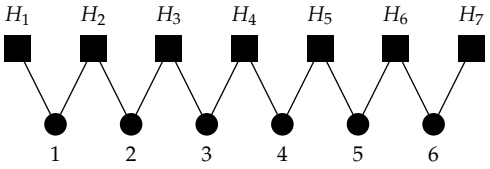


Figure 3.6: Firefighters formation with $n = 6$ agents and $N_h = 7$ houses.

of those that use the factored Q-function learning approach achieve a very small MSE.

Generalized Firefighting: In this experiment, a team of $n = 6$ agents have to fight fire at $N_h = 7$ houses. Each agent can observe $N_o = 2$ houses and can fight fire at the same set of locations ($N_a = 2$), disposed as shown in Figure 3.6. Figure 3.7 shows the representations learned for the single joint type $\theta = \{N_1,F_2,N_3,F_4,N_5,N_6,F_7\}$.

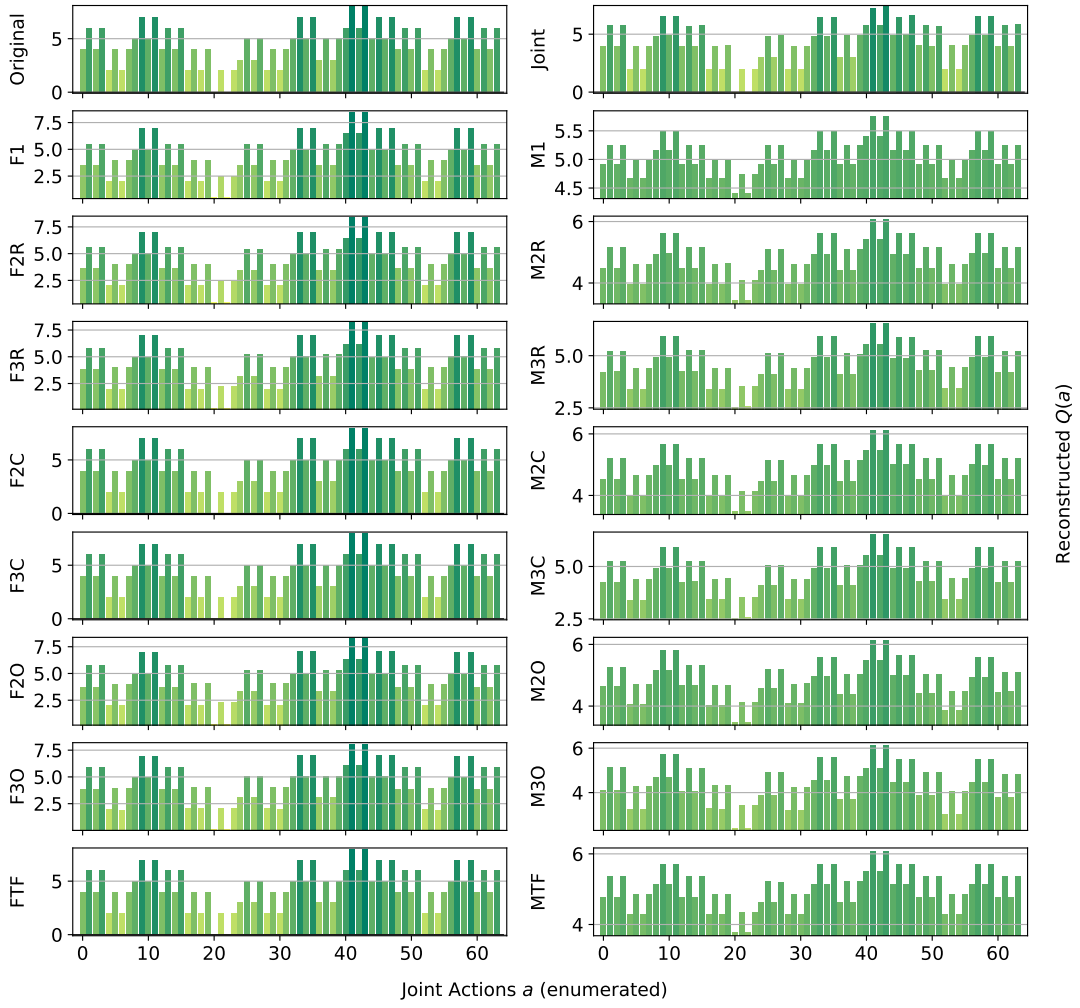


Figure 3.7: Reconstructed $Q(a)$ for a single joint type of the Generalized Firefighting problem.

This game requires less coordination than those studied earlier (agents have to coordinate only with other agents that can fight fire at the same locations), and every

investigated architecture correctly ranks all the joint actions, even the single agent factorizations F1 and M1. However, while those using the factored Q-function can also correctly reconstruct the value of each action, those using the mixture of experts are less precise in their approximations. Overall, this experiment demonstrates that there exist non-trivial coordination problems that can effectively be tackled using small factors, including even individual learning approaches. Also, it is to note how both learning approaches, when coupled with the true underlying factorization, are achieving very good reconstructions and can rank all of the joint actions correctly.

Figure 3.8 shows the results for a different joint type, $\theta = \{F_1,F_2,F_3,\ F_4,F_5N_6,F_7\}$.

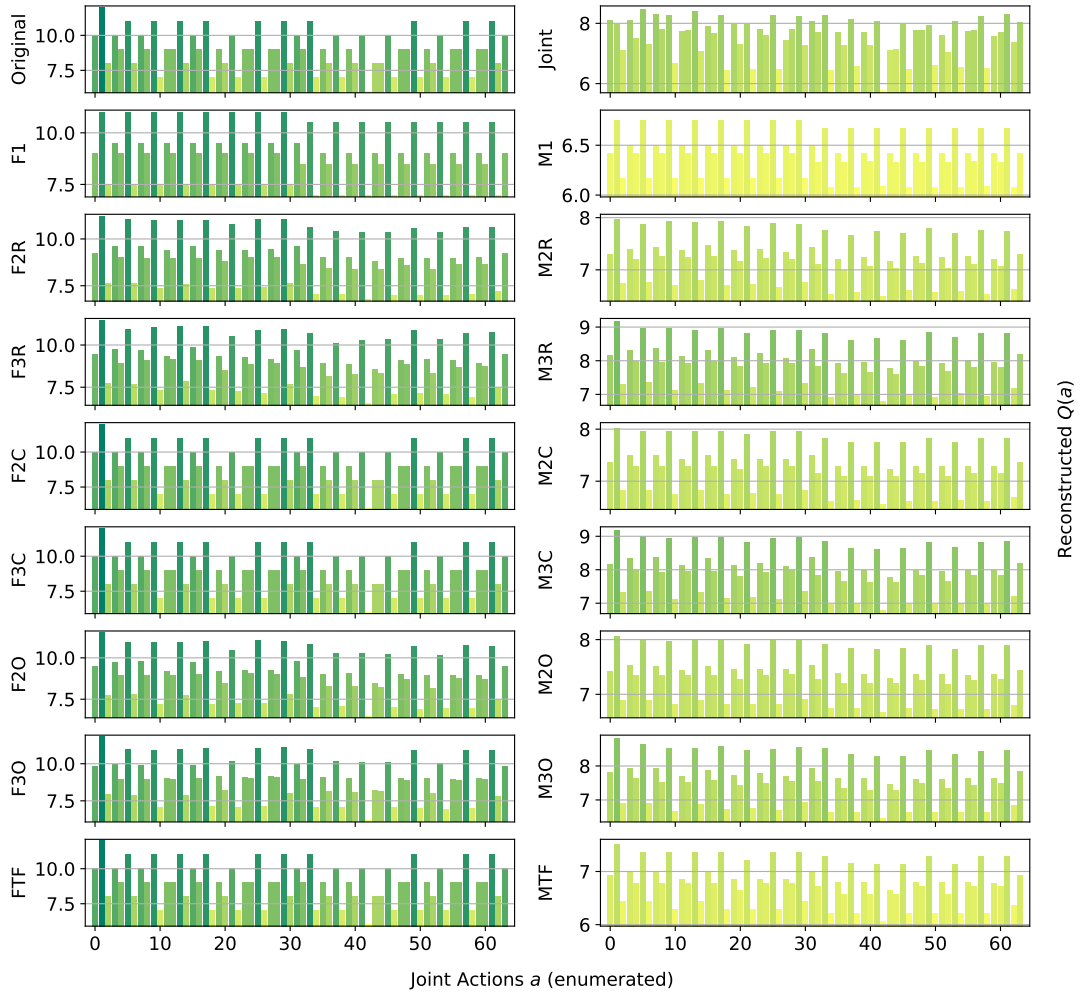


Figure 3.8: Reconstructed $Q(a)$ for a different joint type of the Generalized Firefighting problem.

This type presents multiple adjacent houses burning at the same time, so the agents have to correctly estimate the value of fighting fire at a certain location both on their own or collaborating with other agents. The joint learner is not able to correctly learn the values for this type in the given training time, ranking as optimal actions that are not. Simpler factorizations like F1 or M1 on the other hand fail as well, ranking suboptimal actions as optimal. However, the other factored representations are quite

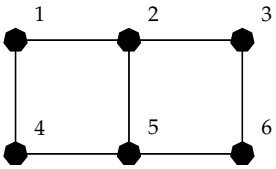


Figure 3.9: Islands configuration with $n = 6$ agents.

accurate and correctly represent the value of coordination: even simpler factorizations using overlapping factors with both learning approaches or random pairing coupled with the factored Q-function learning approach, can correctly identify the optimal joint action. Again, the representations obtained with the factored Q-function learning approach are more accurate in terms of values of the actions. Table 3.7 is showing the best and worst performing methods on this game. On this type as well, both FTF and MTF are achieving very good reconstructions, with FTF also approximating the values of the joint actions correctly.

Model	MSE	Opt. Found	Ranked
Joint	1.29 ± 2.5	656 ± 123	$6{,}893 \pm 1{,}475$
F3R	0.09 ± 0.0	743 ± 25	$7{,}288 \pm 558$
F2C	0.00 ± 0.0	779 ± 0	$8{,}192 \pm 0$
F3C	0.00 ± 0.0	779 ± 0	$8{,}192 \pm 0$
F2O	0.09 ± 0.0	747 ± 18	$7{,}333 \pm 382$
F3O	0.03 ± 0.0	778 ± 4	$8{,}149 \pm 130$
FTF	0.00 ± 0.0	779 ± 0	$8{,}192 \pm 0$
M1	3.55 ± 0.0	700 ± 6	$6{,}220 \pm 30$
M2C	1.82 ± 0.0	777 ± 0	$7{,}826 \pm 0$
M3C	0.85 ± 0.0	778 ± 1	$8{,}151 \pm 4$
M2O	1.97 ± 0.1	741 ± 13	$5{,}628 \pm 249$
M3O	1.73 ± 1.2	738 ± 55	$5{,}867 \pm 682$
MTF	2.60 ± 0.0	779 ± 0	$8{,}177 \pm 2$

Table 3.7: Best (green) and worst (red) performing methods on the Generalized Firefighting problem.

Although the joint action space is very large here (more than 8000 joint actions), most of the factored methods achieve very good performances both in terms of MSE (factored Q-function learning approach methods) and action ranking. Also smaller factorizations like the overlapping factors ones are able to identify almost all of the optimal actions and produce a very good ranking. Both methods using the true underlying factorization are doing very well, with also the mixture of experts one identifying all of the optimal actions. On the other hand, the joint learner is failing in this task, being outperformed even by M1 (that has a higher MSE but a better ranking).

Aloha: This experiment uses a set of $n = 6$ islands disposed in a 2×3 grid as in Figure 3.9, with each island affected only by the transmissions of the islands on their sides and in front of them (islands on the corner of the grid miss one of their side neighbours). Representations learned for this game are reported in Figure 3.10.

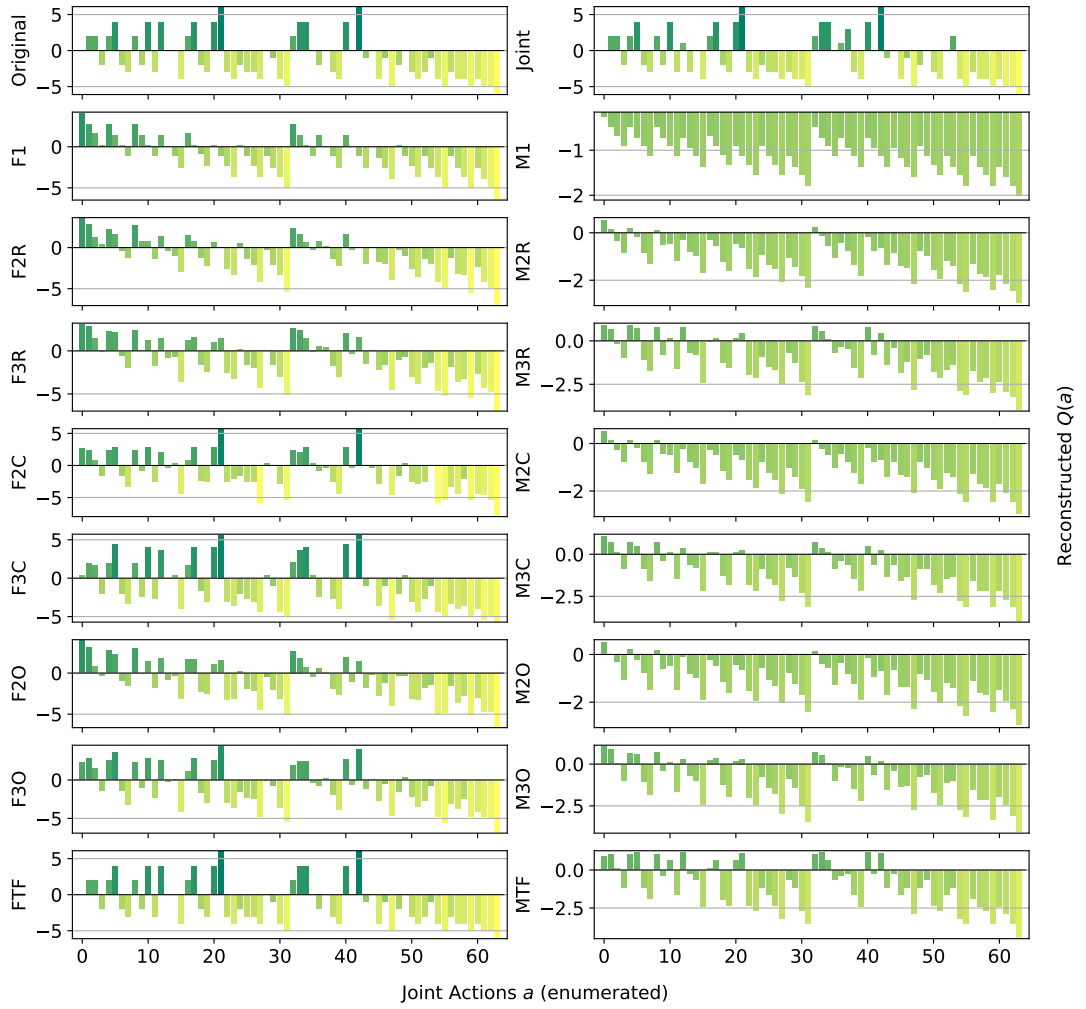


Figure 3.10: Reconstructed $Q(a)$ for Aloha.

The plot shows clearly how this game is challenging for the proposed factorizations to learn, with only three of them (plus the joint learner) able to correctly represent the action-value function. The structure of the game is similar to that of Generalized Firefighting, with an agent depending directly only on a small subset of the others, but the different properties of its Q-function make it more challenging to correctly represent. This is possibly due to the large difference between the two rewards an agent can get when transmitting the radio message, depending on a potential interference. Observing only the total reward, this action looks neutral per se, similarly to what happens for the two actions in the Dispersion Game, its outcome depending on the action of the neighbouring agents, thus possibly fooling many of the proposed factorizations, especially those using the mixture of experts approach. Table 3.8 is showing the best and worst performing methods on this game.

On this more difficult game, all of the mixture of experts methods are not able to identify the optimal actions and achieve a very large MSE. However, the complete factorizations using the factored Q-function learning approach are able to do so, with

Model	MSE	Opt. Found	Ranked
Joint	1.13 ± 0.0	2 ± 0	51 ± 1
F2R	4.05 ± 0.4	0 ± 0	22 ± 4
F3R	3.16 ± 0.5	0 ± 0	26 ± 4
F2C	0.91 ± 0.0	2 ± 0	42 ± 0
F3C	0.07 ± 0.0	2 ± 0	64 ± 0
F2O	3.27 ± 0.3	0 ± 0	23 ± 4
F3O	1.46 ± 0.3	1 ± 1	29 ± 5
FTF	0.00 ± 0.0	2 ± 0	64 ± 0
M1	8.26 ± 0.0	0 ± 0	27 ± 1
M2R	6.52 ± 0.2	0 ± 0	25 ± 4
M2O	6.63 ± 0.4	0 ± 0	22 ± 5
M3O	4.71 ± 0.3	0 ± 0	25 ± 4

Table 3.8: Best (green) and worst (red) performing methods on Aloha.

F3C also ranking correctly all of the other joint actions. Again, FTF is performing the best, with a perfect ranking and a very low reconstruction error, outperforming even the joint learner. This once more shows how beneficial would it be to exploit an appropriate factored structure when that is known beforehand.

3.2.3 *Impact of Factors Size*

Although this chapter mainly focuses on factors of small size, it is also interesting to investigate how the size of the factors is affecting the final representation, and if using factors of larger size can help to overcome some of the issues encountered with small factors. To investigate this, the methods defined in Table 3.9 are tested on the Platonia Dilemma and Penalty Game with $n = 6$ agents, two of the games that proved more problematic to correctly represent.

	Mix. of Experts	Factored Q
Random partition ($f = 4,5$)	M4R, M5R	F4R, F5R
Complete factorization ($f = 4,5$)	M4C, M5C	F4C, F5C
Overlapping factors ($f = 4,5$)	M4O, M5O	F4O, F5O

Table 3.9: Combinations of factorizations and learning rules with larger factors.

Platonia Dilemma: Figure 3.11 shows the reconstructed action-value functions for the Platonia Dilemma. It can be seen how this game, that none of the factored methods in Figure 3.3 was able to correctly approximate, remains very challenging even with factors comprising more agents. Indeed, only methods with a factor size $f = 5$, thus very close to the entire team size $n = 6$, and using the factored Q-function learning approach (F5C and F5O), are able to correctly reconstruct the action-value function. The same factorizations using the mixture of experts learning approach are instead consistently ranking one of the suboptimal actions (the one in which none of

the agents is sending the telegram) as an optimal one, the same as with factors of smaller size.

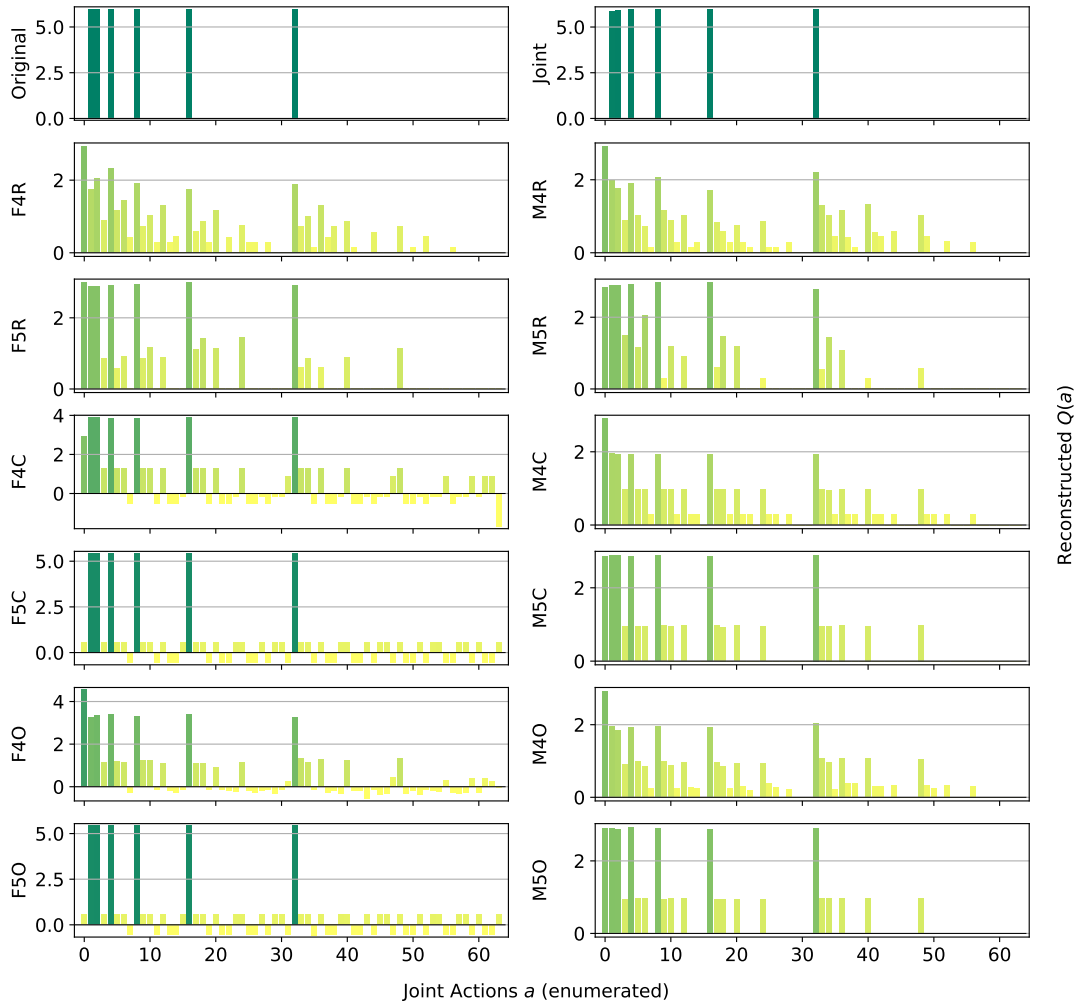
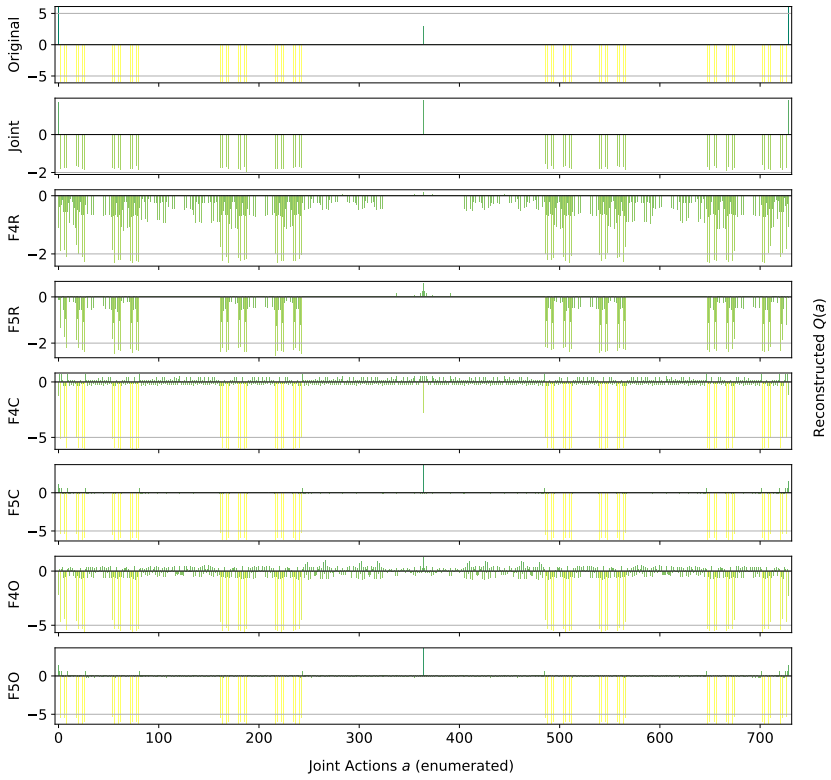
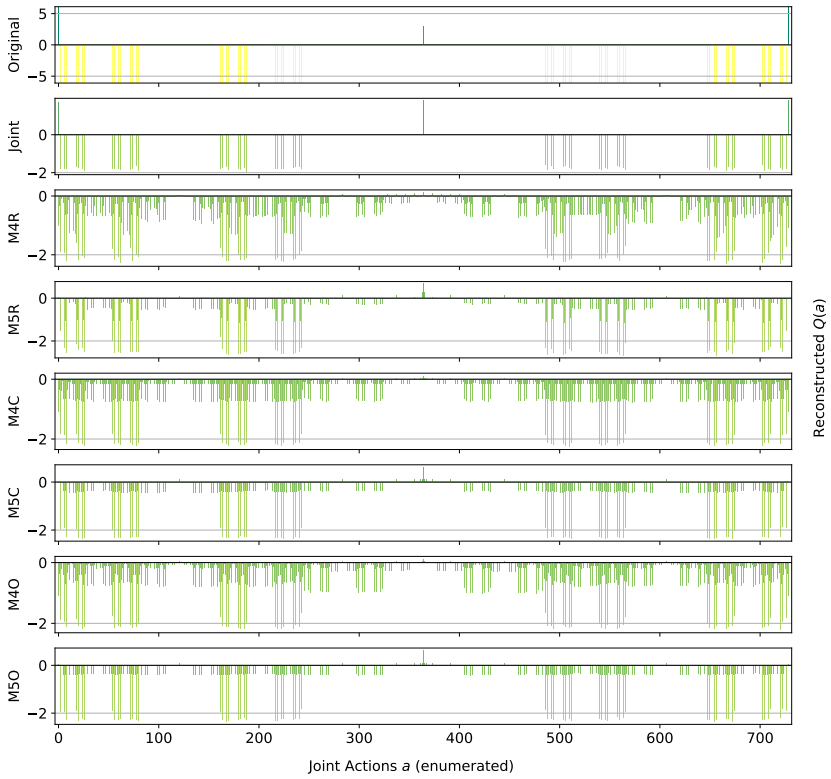


Figure 3.11: Reconstructed $Q(a)$ for the Platonia Dilemma.

Penalty Game: Figure 3.12 presents the representations obtained by the new investigated approximations. Even with larger factors, none of the methods is able to reconstruct any of the optimal actions, but they only are able to discern the value of the sub-optimal one like F1 and F3C in Figure 3.5 (that is seen as optimal). The same kind of problems that arose with smaller factors are also present here, with the mixture of experts methods tending to underestimate values for all the joint actions and generally none of the methods being able to represent the true value of coordination for this problem. However, the methods using the factored Q-function learning approach and $f = 5$ agents into each factor are reconstructing small yet positive values for these optimal actions, meaning that the resulting reconstruction is at least identifying these as good actions that the agents may desire to perform.



(a)



(b)

Figure 3.12: Reconstructed $Q(a)$ for the Penalty Game 3.12(a) factored Q-function learning approach, and 3.12(b) the mixture of experts learning approach.

3.2.4 *Scalability*

A fundamental aspect for a multi-agent algorithm is how well it can scale with the size of the system, i.e. when more agents are introduced and therefore the size of the joint action set exponentially increases. In this section it is investigated how using a factored representation helps when such systems get larger, as well as analyse how this affects the performance of both independent learners and joint learners. Table 3.10 illustrates the games used to investigate this aspect.

Game	n	$\lvert A^i \rvert$	$\lvert A \rvert$	Optimal	Factored
Dispersion Game	9	2	512	252	No
Dispersion Game	12	2	4096	924	No
Generalized FF	9	2	512 (524.288 total)	17.682	Yes
Aloha	9	2	512	1	Yes
Aloha	12	2	4096	2	Yes

Table 3.10: Details of the investigated games in this section.

Dispersion Games: Figure 3.13 shows the action-value function reconstructed by the proposed factorizations and learning approaches for the Dispersion Game with $n = 9$ agents (a similar figure for the case when $n = 12$ would have rendered unreadable and is thereby not included). It can be observed that the complete factorizations are able to almost perfectly reconstruct the relative ranking between the joint actions even in this larger setting, showing how reliable and general can this kind of approach be. As usual, the ones using the factored Q-function are also able to produce a generally good approximation of the various components, while those based on the mixture of experts produce a less precise reconstruction: the joint optimization of the former seems to have an even bigger benefit when more agents are present.

It is interesting to note how both independent learners and the joint learner are failing here, but for different reasons: both types of independent learners seem not able to correctly learn the value of coordination with the others (something already appearing on the smaller instance shown in Figure 3.2), while the latter is struggling because of the increased number of agents that makes the function that has to be represented too big to be reliably learned in the given training time. The other factored approaches instead are capturing the value of such a coordination up to some extent (especially these using the overlapping factors), but the small number of factors is probably not sufficient to completely represent such a function. However, the resulting MSE is still lower than the joint learner and some of the optimal actions are still ranked correctly, making these approaches still viable for decision making. Table 3.11 reports the best and worst performing methods on the two instances of this game, both in terms of action ranking and reconstruction error.

As already stated, when the size of the system increase both independent learners and the joint learner struggle in representing the corresponding action-value function

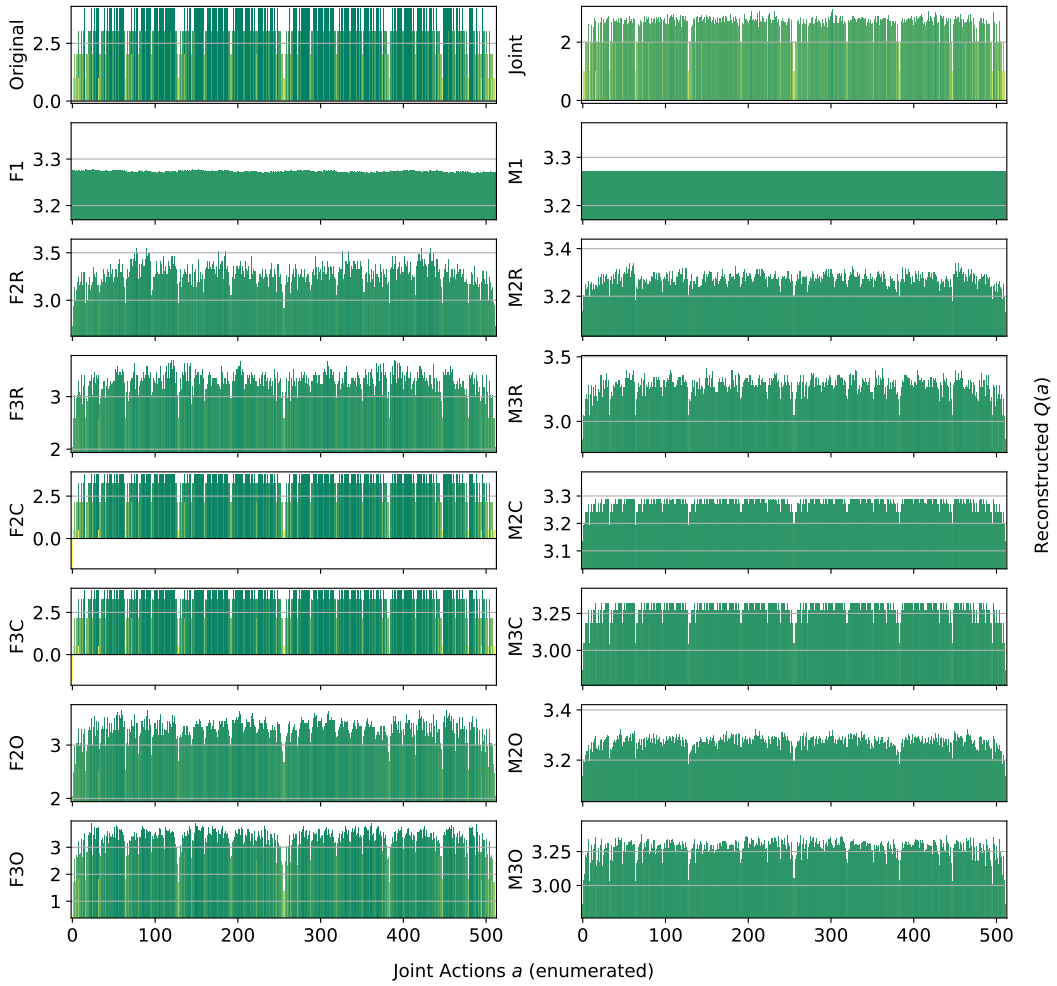


Figure 3.13: Reconstructed $Q(a)$ for the Dispersion Game with $n = 9$ agents.

Model	MSE	Opt. Found	Ranked
Dispersion Game $n = 9$			
Joint	0.93 ± 0.5	162 ± 30	307 ± 77
F1	0.74 ± 0.0	124 ± 1	191 ± 3
F2C	0.06 ± 0.0	252 ± 0	512 ± 0
F3C	0.06 ± 0.0	252 ± 0	512 ± 0
M1	0.74 ± 0.0	124 ± 1	192 ± 4
M2C	0.70 ± 0.0	252 ± 0	512 ± 0
M3C	0.63 ± 0.0	252 ± 0	512 ± 0
Dispersion Game $n = 12$			
Joint	19.48 ± 0.7	210 ± 8	$1{,}108 \pm 34$
F1	1.17 ± 0.0	186 ± 39	$1{,}137 \pm 37$
F3R	0.99 ± 0.0	351 ± 7	$1{,}364 \pm 20$
F2C	0.17 ± 0.0	924 ± 0	$4{,}096 \pm 0$
F3C	0.20 ± 0.0	774 ± 194	$3{,}711 \pm 510$
M1	1.17 ± 0.0	187 ± 30	$1{,}134 \pm 36$
M3R	1.09 ± 0.0	350 ± 9	$1{,}363 \pm 24$
M2C	1.14 ± 0.0	813 ± 69	$3{,}831 \pm 246$
M3C	1.08 ± 0.0	920 ± 5	$4{,}089 \pm 10$

Table 3.11: Best (green) and worst (red) performing methods on the two larger instances of the Dispersion Game.

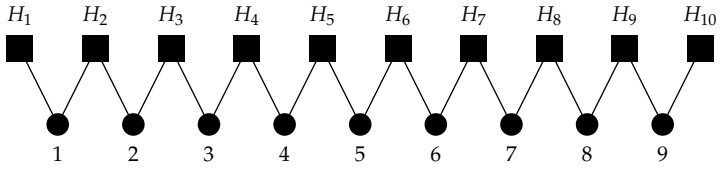


Figure 3.14: Firefighters formation with $n = 9$ agents and $N_h = 10$ houses.

correctly. The latter especially, that was achieving a perfect reconstruction for the same game with only $n = 6$ agents, is now resulting in a higher reconstruction error and fail in identifying all of the optimal joint actions. Methods using a complete factorization with both learning approaches instead are still able to identify most of them (all, when $n = 9$), while at the same time reducing the MSE considerably. Smaller factorizations are not reported because these are not achieving such good performances (as in the smaller case with $n = 6$), showing that on this kind of very tightly coordinated problems these may not suffice for a completely correct representation.

Generalized Firefighting: In this larger experiments, a team of $n = 9$ agents is fighting fire at $N_h = 10$ houses. As in the previous setting, each agent can observe $N_o = 2$ houses and can fight fire at the same set of locations ($N_a = 2$), as shown in Figure 3.14. Reconstruction results for the single joint type $\theta = \{N_1,F_2,F_3,N_4,F_5,F_6,N_7,N_8,F_9,F_{10}\}$ are reported in Figure 3.15.

From these results, it can be observed how, although the problem is very large (with more than half a million total joint actions with the described formulation) most of the factored methods are perfectly representing the corresponding Q-function. While methods using the complete factorization or exploiting the true underlying structure with both learning approaches are capable of achieving a perfect reconstruction, even simpler methods like random pairing with the factored Q-function learning approach are capable or almost perfectly reconstruct the values for this joint type. Conversely, the joint learner seems not capable of doing so, resulting in a totally wrong representation that is not close to the original function. Things are similar for a second joint type, $\theta = \{F_1,F_2,N_3,N_4,N_5,N_6,N_7,F_8,F_9,N_{10}\}$, whose resulting learned representations are shown in Figure 3.16:

Again, the joint learner is not capable of achieving a good representation, but also some of the simpler factorizations are not resulting in a perfect reconstruction, although still capable of correctly identifying the optimal joint actions. Complete factorizations are instead perfectly representing the original Q-function for this joint type as well, even with the mixture of experts learning approach. General metrics and results for the best and worst performing methods on this problem are reported in Table 3.12.

As expected, the methods provided with the true underlying factorization are performing best, with that using the factored Q-function learning approach capable of

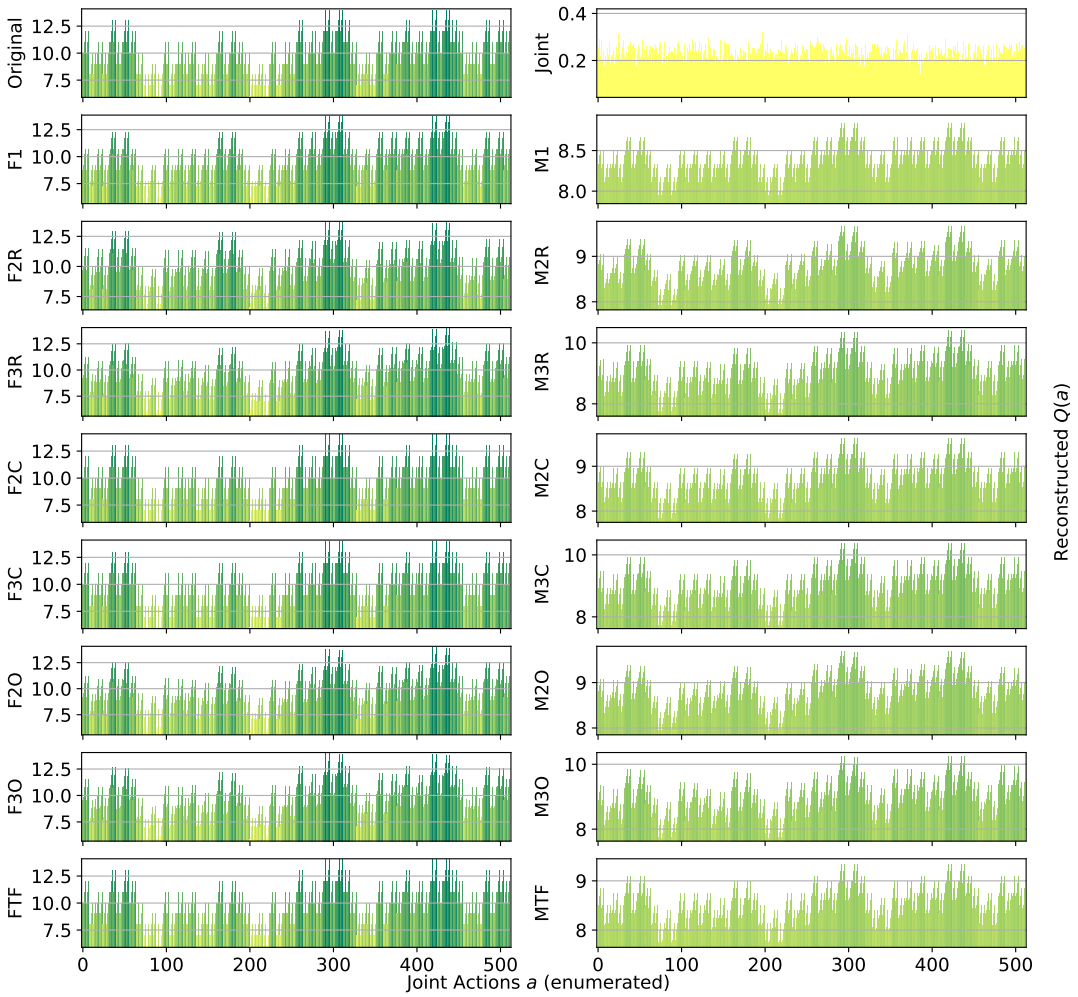


Figure 3.15: Reconstructed $Q(a)$ for a single joint type of the Generalized Firefighting problem with $n = 9$ agents.

Model	MSE	Opt. Found	Ranked
Joint	69.98 ± 0.9	$2{,}556 \pm 50$	$94{,}559 \pm 522$
F2C	0.00 ± 0.0	$17{,}682 \pm 0$	$524{,}288 \pm 0$
F3C	0.00 ± 0.0	$17{,}682 \pm 0$	$524{,}288 \pm 0$
F3O	0.09 ± 0.0	$16{,}939 \pm 415$	$464{,}893 \pm 26{,}119$
FTF	0.00 ± 0.0	$17{,}682 \pm 0$	$524{,}288 \pm 0$
M2C	4.30 ± 0.0	$17{,}680 \pm 0$	$465{,}019 \pm 2$
M3C	2.71 ± 0.0	$17{,}680 \pm 0$	$465{,}090 \pm 2$
M3O	2.96 ± 0.1	$16{,}783 \pm 303$	$298{,}182 \pm 26{,}364$
MTF	5.30 ± 0.0	$17{,}682 \pm 0$	$512{,}022 \pm 810$

Table 3.12: Best (green) and worst (red) performing methods on the larger instance of the Generalized Firefighting problem.

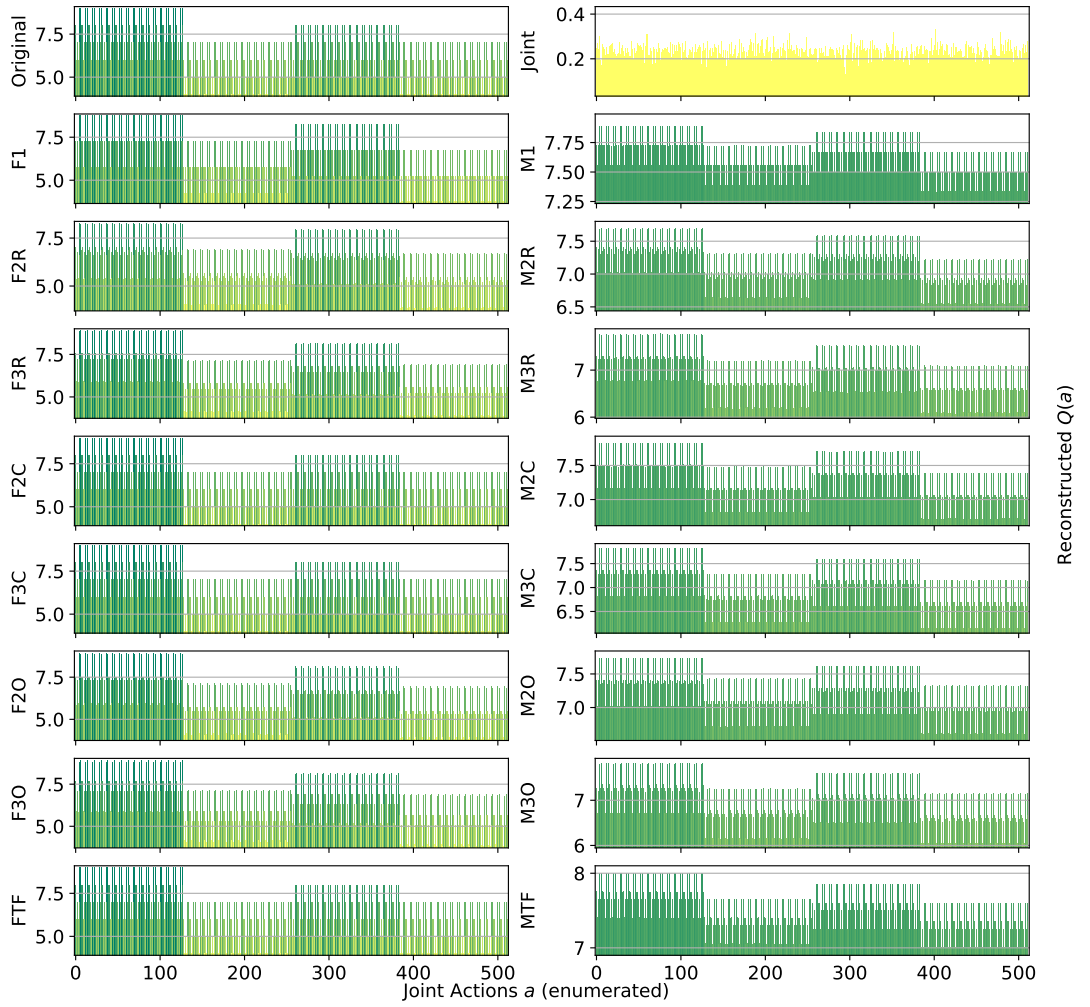
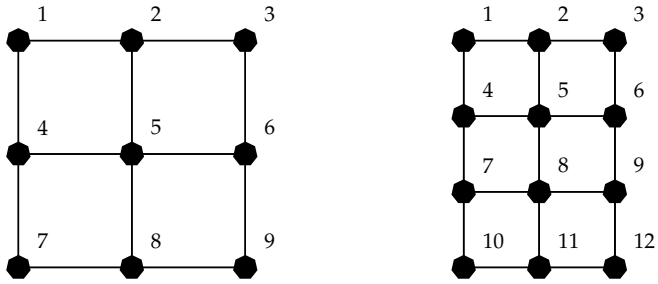


Figure 3.16: Reconstructed $Q(a)$ for a different joint type of the Generalized Firefighting problem with $n = 9$ agents.

achieving a perfect reconstruction and ranking of the actions even on this very large problem. Also complete factorizations are always identifying all of the optimal joint actions and producing correct ranking (perfect for those using the factored Q-function learning approach). It is interesting to note that even methods using overlapping factors, when coupled with larger factors, are performing very well, and can produce a good ranking of the actions. As expected, the mixture of experts methods are resulting in a larger MSE, although being comparable on the other metrics with their counterparts, but are still capable of learning more accurate representations than the joint learner, that is instead achieving the highest MSE and worst ranking among all the compared methods.

Aloha: The experiments reported here use $n = 9$ and $n = 12$ islands disposed in a 3×3 and 4×3 grid respectively, as shown in Figure 3.17. Representations learned for this game with $n = 9$ islands are reported in Figure 3.18 (for identical reasons to those of the Dispersion Game, the figure with $n = 12$ agents is not included).



(a) $n = 9$ (b) $n = 12$

Figure 3.17: Islands configuration for the two larger instances of Aloha.

Again, this game proves to be challenging for almost all of the proposed factorizations. Indeed, other than the true underlying factorization coupled with the factored Q-function learning approach (that achieve a perfect reconstruction, showing how beneficial would it be to know and exploit such an underlying factorization in advance), only the complete factorizations seems able to learn something useful. All the other methods struggle to correctly identify the optimal action, probably because not enough coordination is achieved in order to discriminate between the two local actions for each agent (that seems similar from the agent's perspective). Also on this game, the joint learner is not capable of correctly approximating the action-value function because of the increasing number of agents. Table 3.13 is showing the best and worst performing methods on this game.

Model	MSE	Opt. Found	Ranked
	Aloha $n = 9$		
F2R	6.97 ± 0.4	0 ± 0	100 ± 11
F2C	2.25 ± 0.0	1 ± 0	187 ± 1
FTF	0.00 ± 0.0	1 ± 0	512 ± 0
M1	11.93 ± 0.0	0 ± 0	126 ± 1
M2R	10.45 ± 0.2	0 ± 0	105 ± 11
M2C	10.31 ± 0.0	0 ± 0	140 ± 1
M2O	10.48 ± 0.2	0 ± 0	114 ± 10
	Aloha $n = 12$		
Joint	23.98 ± 0.6	0 ± 0	700 ± 13
F2C	2.18 ± 0.0	2 ± 0	$1{,}380 \pm 5$
F3C	0.81 ± 0.0	1 ± 0	$2{,}488 \pm 10$
FTF	0.00 ± 0.0	2 ± 0	$4{,}096 \pm 0$
M1	15.37 ± 0.0	0 ± 0	845 ± 26
M2O	13.94 ± 0.2	0 ± 0	671 ± 58
M3O	12.56 ± 0.3	0 ± 0	752 ± 87

Table 3.13: Best (green) and worst (red) performing methods on the two larger instances of Aloha.

The table shows how, except for FTF (always capable of correctly representing the entire Q-function), all the methods start deteriorating their performance when the system size increases on this particular problem. Particularly, the joint learner

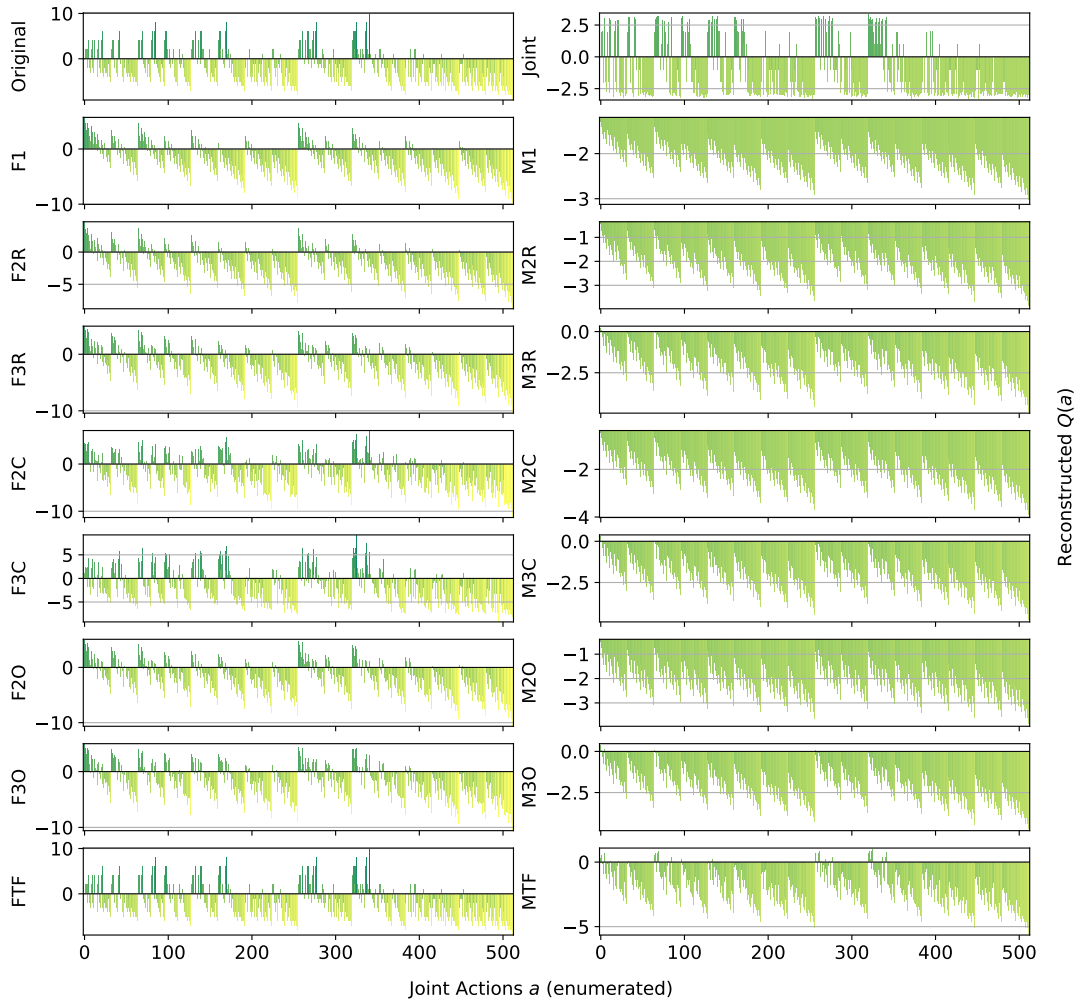


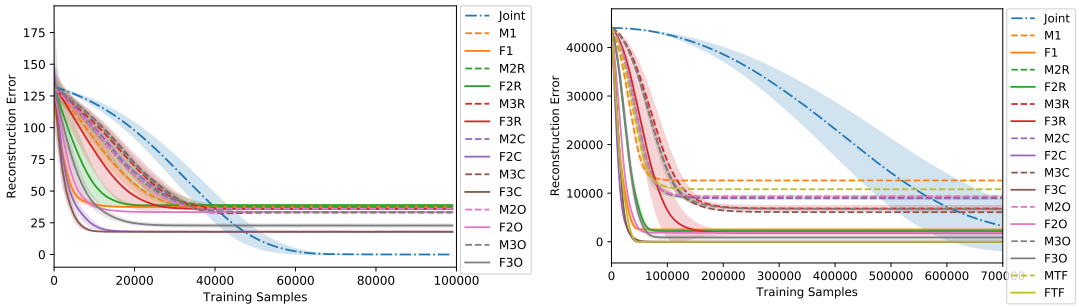
Figure 3.18: Reconstructed $Q(a)$ for Aloha with $n = 9$ agents.

achieves a very high reconstruction error and is not able to identify any of the optimal joint actions. On the other hand, although the corresponding ranking is not perfect, complete factorizations using the factored Q-function learning approach are able to identify such optimal actions. The mixture of experts instead are performing worse here, probably because the benefits of a coordinated optimization is crucial to correctly represent this problem.

3.2.5 *Sample Complexity*

Another important consideration in multi-agent learning is sample complexity, as for example training data could be limited or expensive to obtain. Therefore it is a crucial aspect how efficiently such data can be used and how long does it take for a given representation to converge, especially when the system grows larger in the number of agents. The expectation is that factored representations can improve training efficiency, reducing the number of samples required to learn a good representation,

as the size of the multiple components that have to be learned is small compared to that of the overall problem. To show the benefits of using a factored representation, here in Figure 3.19 the training curves for two of the proposed games, the Dispersion Game and the Generalized Firefighting, both with $n = 6$ agents, are reported.

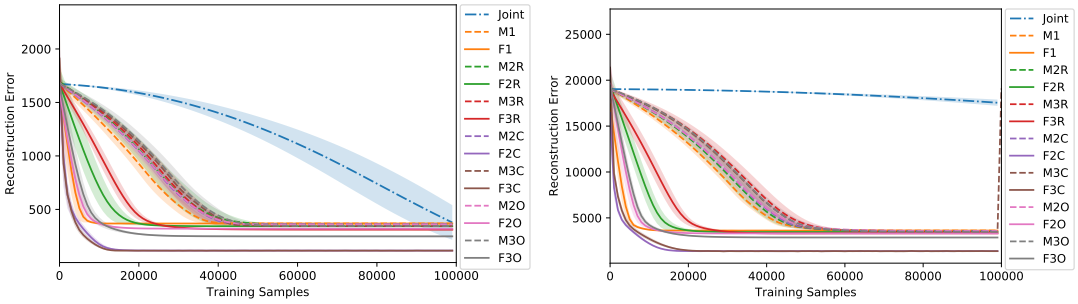


(a) Dispersion Game, $n = 6$ (b) Generalized Firefighting, $n = 6$

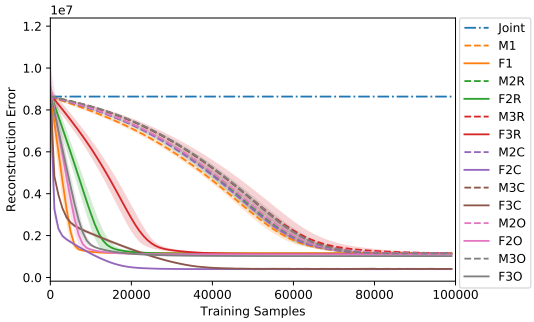
Figure 3.19: Training curves for the investigated architectures on the two proposed problems.

Even for these problems of moderate size, factored approaches achieve a stable approximation of the action-value function with just a fraction of the given training time, while a full joint learner requires many more samples to get the same result. Especially, for the Generalized Firefighting problem (that has got more than 8000 overall joint actions), the joint learner achieves an accurate representation only after a much longer training time, while almost every factored architecture achieves a nearly perfect approximation with few samples, showing how the size of the joint action space is a critical problem that factored representations can help to tackle. On one hand the mixture of experts approaches learn slower than the factored Q-function ones: each factor acts as an expert on its own, thus experiencing higher variance in the received rewards when performing a certain action. On the other hand, larger models learn more quickly, achieving the same final result as the smaller representations but with fewer samples. This could be due to the internal coordination happening inside each factor, helping the agents figure out their own contribution to the global reward, so that a stable representation is learned more easily. When the number of agents is larger, this benefit is even more apparent. Figure 3.20 shows the reconstruction error during the training process obtained on instances of the Dispersion Game with $n = \{9,12,20\}$ agents respectively.

It can be observed how the joint learner is struggling to achieve a good representation in the given training time when the size of the system increases, resulting in a higher reconstruction error. The increasingly large number of joint actions (more than 1 million with $n = 20$ agents) prevents it from converging in a reasonable time, while the factored representations, although only approximating the original function, converge faster, as the size of each factor is small compared to that of the overall problem, and result in a lower reconstruction error.



(a) $n = 9$ (b) $n = 12$



(c) $n = 20$

Figure 3.20: Training curves for the investigated architectures on the Dispersion Game with an increasing number of agents.

3.2.6 *Exploratory Policy*

Although the focus is on a stationary uniform sampling of the actions throughout most of the chapter, some preliminary results with a different, non-stationary action selection mechanism are also provided, more closely resembling those used in sequential MARL [110, 96]. The choice is to use a Boltzmann policy [141, 65] that, given a reconstruction of the action-value function $\hat{Q}(a)$, defines the probability for each joint action $a \in A$ to be selected as:

$$\pi(a) = \frac{e^{\hat{Q}(a)/\tau}}{\sum_{b \in A} e^{\hat{Q}(b)/\tau}}, \tag{3.12}$$

where τ is a temperature parameter governing the exploration rate. In this experiment, $\tau = 1$ for all methods. These are tested on the Dispersion Game with $n = 6$ agents, as many of the methods (including the joint learner) are doing reasonably well and thus any decrease in performance would be solely due to the new exploratory policy. For the factored methods and the independent learners, $\hat{Q}(a)$ is reconstructed at every step and then the Boltzmann policy is applied on this reconstruction. Figure 3.21 shows the learned reconstructions on this game.

If the above is compared to Figure 3.2(a), it can be observed how the results are very much in line with those obtained under a uniform sampling. Even if the joint action space is small enough to select each action a reasonable number of time, the

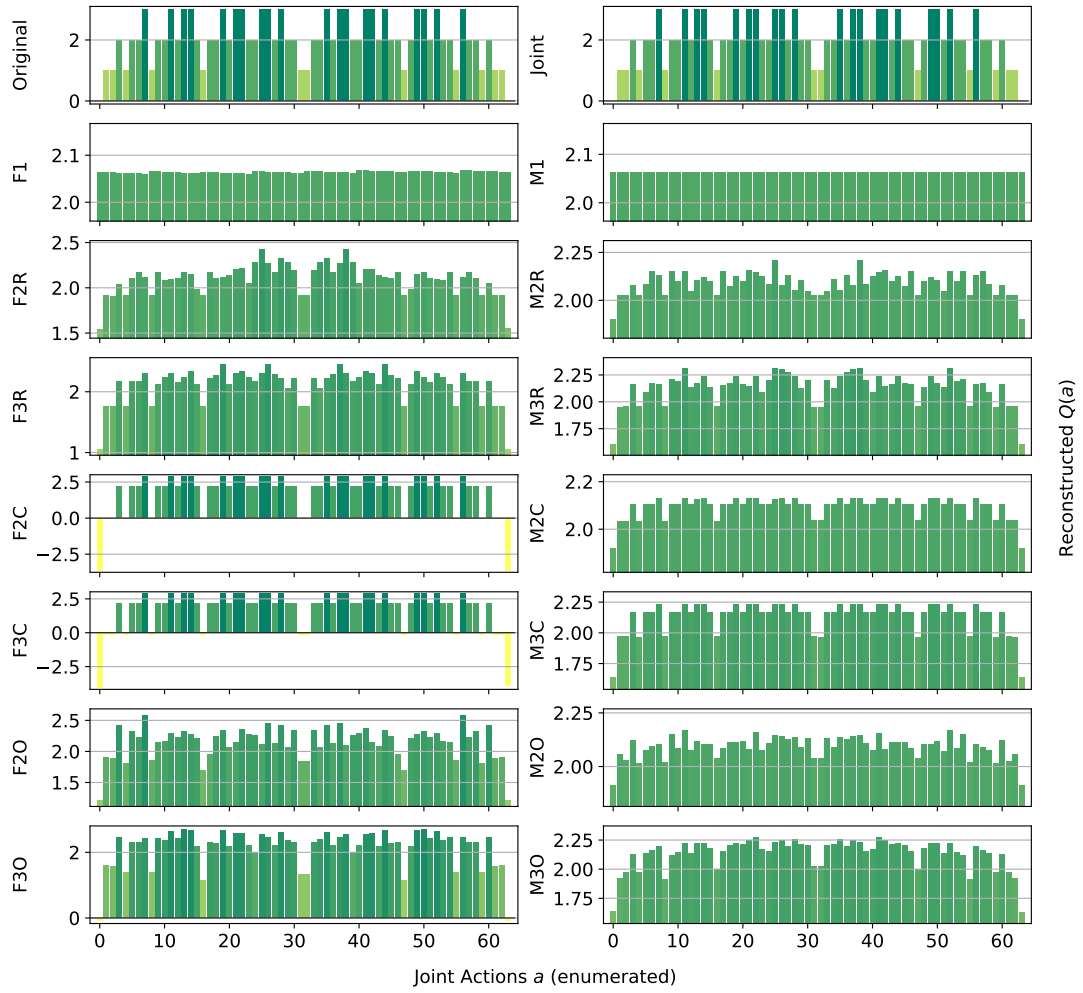


Figure 3.21: Reconstructed $Q(a)$ for the Dispersion Game using the Boltzmann exploratory policy.

fact that the policy is more frequently selecting the actions that look better is not providing any benefits in term of accuracy of the final representations, especially for the independent learners, that are still not able to clearly identify any of the optimal joint actions. The joint learner and the complete factorizations are still able to correctly rank all of the optimal actions, and those using the factored Q-function learning approach are achieving a smaller reconstruction error as under the uniform sampling. Although supported just by a preliminary result, this is an important observation, as it gives more value to the previous results: if a method is not learning an accurate representation under the uniform sampling of actions (thus experiencing enough samples for each such actions), it is unlikely that it can do better with a non-stationary, time-varying sampling mechanism like this.

3.2.7 *Summary of Results*

Many useful insights can be gained from the analysed results: first, it can be observed that the factorizations using mixture of experts learning approach, although generally achieving higher reconstruction errors than their factored Q-function counterparts, in many cases still result in a good approximation in terms of ranking of actions, therefore being a reliable choice for decision making. For example, on the two variants of the Dispersion Game, both M2C and M3C are able to correctly rank all of the joint actions, achieving better accuracy than some smaller factorizations like F2O or F3O, even with a higher mean square error of the reconstruction. This is probably due to the higher number of factors involved in their coordination graphs, allowing for better approximation of the true action-value function and coordination among the agents. Therefore, it can be deduced how the number of factors used to learn an approximation is playing a major role in achieving accurate representations in terms of coordination and action ranking.

Also, the size of these factors is an important aspect: as expected, with more agents comprised into each factor, the resulting approximation is more reliable because the agents into each factor are able to share information and thus better coordinate. This is reflected by both learning approaches, but it is even more apparent with the mixture of experts one, with the factorizations with 3 agents per factor usually achieving smaller reconstruction error and a better ranking of actions than their counterparts with only 2 agents in each. However, factors that are too large (with a size very similar to that of the entire team) do not always result in a better representation, but instead can present some of the difficulties associated with joint learners. This suggests that one can find an optimal trade-off between the totally independent learners and the full joint learner extremes that is capable of achieving a reliable representation in a reasonable training time. Of course, depending on the intended use of such a learned representation, such a trade-off may differ: for example, if one is only interested in selecting an optimal joint action from this reconstruction after the training process (i.e. with a factored centralized joint Q-function agent for the entire team of agents) a smaller factorization with fewer factors and agents per factor, that is faster to train and still able to correctly rank some of these optimal joint actions, may suffice. Conversely, if one is to approximate the critic of an actor-critic method, in which the values of the selected actions are in turn influencing the policies of the agents, he may prefer a bigger factorization with a lower reconstruction error.

Generally speaking, it can be observed how the value of coordination is well captured by the factored Q-function learning approach, that usually produces good approximations and reduces the training time with respect to a joint learner, allowing to learn even in games in which there is no underlying factorization and complete team coordination is required, like the Dispersion Game. This becomes even more important with larger systems, as now more agents have to coordinate and thus a

more accurate representation of such coordination requirements is needed. However, when these requirements are less tight, also the mixture of experts learning approach is showing benefits, being faster to train and not requiring inter-factor communication during the training phase to optimize its objective. Overall, the results showed how the choice of a suitable factorization to efficiently learn in a multi-agent system is a difficult decision that needs to be taken considering the problem structure and requirements. While many aspects can influence the learning outcome, the provided results have five main takeaways:

- There are some problematic examples, like the Platonia Dilemma, where all types of factorization with small factors result in selecting the worst possible joint action. Given that only joint learners (and certain factorizations with larger factors to some extent) seem to be able to address such problems, currently no scalable deep reinforcement learning methods for dealing with such systems seems to exist. A hypothesis is that this is due to an imbalance in the frequency with which each local action for the agents leads to the optimal reward value (i.e., is part of an equilibrium strategy). In the Platonia Dilemma, each agent is more frequently experiencing the positive reward if it does not send the telegram itself and leave this action to someone else. However, if all the agents do this kind of reasoning, no-one is sending the telegram, and the resulting reward is not the optimal one. Breaking this tie is possible when behaving greedily (as the first agent correctly sending the telegram will keep sending it more frequently), but learning a complete and correct representation of the entire Q-function remains challenging.

- Beyond those, "complete factorizations" of modest factor size coupled with the factored Q-function learning approach yield near-perfect reconstructions and rankings of the actions, also for non-factored action-value functions. Moreover, these methods scale much better than joint learners: for a given training time, it can be seen that these complete factorizations already outperform fully joint learners on modestly sized problems, resulting in a correct ranking of the actions and a low reconstruction error. This is a compelling property that renders these methods more suited for large multi-agent systems with a large joint action space and justify the recent interest in factored methods from the research community.

- For many problems with less tight coordination requirements such as Aloha and the Generalized Firefighting problem, random overlapping factors also achieve excellent performances, comparable to those of more computationally complex methods like joint learners and complete factorizations. This suggests that such approaches are a promising direction forward for scalable deep MARL in many problem settings.

- Factorizations with the mixture of experts learning approach usually perform somewhat worse than the corresponding factored Q-function approaches, as these are less able to capture the coordination requirements of certain problems. However, in some cases they perform better or comparably (Dispersion Game, Generalized Firefighting), in which M2R and M3R still outperform F1 (i.e., VDNs). This is promising, because the mixture of experts learning approach does not require to share the learning gradients among the neural networks, thus potentially facilitating learning in settings with communication constraints, and making it easier to parallelize across on multiple CPUs/GPUs.
- The provided results show that, when facing larger multi-agent systems, factored representations retain many of their benefits and can still represent the action-value function correctly in many settings (or at least identify most if not all of the optimal joint actions), while both independent learners and centralized approaches tend to quickly deteriorate, resulting in wrong or incomplete representations. The reasons however are different: while independent learners still decompose the entire function into small components that can easily be learned, these fail in representing the value of coordinated decisions when more agents are comprised into the system. On the other hand instead, a joint learner that is representing the centralized action-value function as a single component (and thus does not introduce any approximation), is now struggling to learn a correct representation because of the exponential number of joint actions. Factored representations instead learn small components easily, but take into account the value of coordination into each component, allowing for better final representations. This is an extremely desirable property that once more points out how these methods deserve attention as holding a great potential.

These observations also shed some light on the performance of independent learners in MARL, as used by many modern deep MARL algorithms [139, 115]: while these can outperform joint learners on large problems, the degree of independence and the final outcome is hard to predict and is affected by different factors. Designing algorithms that are able to overcome these difficulties should be a primary focus of MARL research.

3.3 DISCUSSION

The aim of this analysis was to investigate the learning capabilities of factored representations and compare them to both independent learners and joint learners, to assess eventual benefits of such techniques both in terms on learning speed and final accuracy of the reconstructed approximations. In order to do so, different aspects of these learned representations $\hat{Q}$ have been considered: the optimality of the greedy joint action, which is important when using $\hat{Q}$ to select actions. Also, the distance to

the optimal value $\Delta Q = |Q - \hat{Q}|$, since verifying the optimality of the greedy action requires to have a limited ΔQ. Although the proposed approach focus on centralized learning, a correct representation of the joint action-value function can be used in the CTDE framework [70] to learn improved decentralized policies. Indeed, minimising ΔQ is important for deriving good policy gradients in actor-critic architectures (for example when computing the counterfactual baseline in [40], requiring very accurate estimates of many sub-optimal Q-values) and for sequential value estimation in any approach that relies on bootstrapping (such as Q-learning [162]) or message passing of local payoff values (like the max-sum algorithm [67]), where such values are used to update other values and thus need to be as accurate as possible.

This analysis is focused on cooperative one-shot games. These have been chosen because, although they are a simpler setting than standard sequential problems, they still capture many of the aspects that can be problematic in MARL, like the exponentially large number of joint actions. Also, the shared reward observed by all the agents depends on the joint action of the whole team, and thus seems non-stationary from an agent perspective. Therefore, all of the problems that arise from these conditions are directly translated into this setting, and by removing the hindering effect of states, one is better able to analyse how the different methods can tackle these issues. Moreover, the main focus is on using a stationary uniform sampling mechanism to select joint actions, thus not directly considering the exploration-exploitation trade-off usually faced in MARL problems. There are multiple reasons for this: on one hand, using such a simple mechanism makes the comparison easier, as the interest is on the reconstruction of the values for the entire action-value function (and not only on identifying the greedy action). On the other hand, if a network architecture is not suited to learn and represent accurate action-values under the stationary uniform sampling, it is extremely unlikely that the same is going to perform better under a more complex, time-changing policy as in sequential MARL. This is validated by the preliminary results with a Boltzmann policy, in which none of the methods is achieving better accuracy than under the stationary uniform mechanism. However, while good performances in such one-shot settings does not necessarily imply good performances in the sequential setting, value-based approaches aim to transform the sequential MARL problem to precisely the investigated one-shot decision making problem, as the sequential problem simply becomes a one-shot maximization over the action-values. This implies that any found limitation is likely to directly transfer to such approaches.

Obviously, the opposite is not immediately true: good performances on this setting does not directly imply also good performances on general MARL problems. However, if a given representation is able to correctly capture the value of coordination and deal with the joint action set size in one-shot games, it can be hypothesized that these benefits are likely to hold in sequential problems as well, where the same requirements usually bring similar challenges to the investigated setting and

thus can be tackled with similar solutions. Neural networks are known to be very good at dealing with large input spaces, but it is not so well known how these can deal with large output ones, like those resulting from exponential joint action sets. Therefore, this analysis moves an important step in this direction in the field of MARL, and has to be considered as an initial step towards a proper understanding of action-value functions in multi-agent settings. Of course, a direct investigation in the full sequential setting is an interesting future direction, with key questions that are orthogonal to those addressed here, like the presence of an input state of the exploration-exploitation trade-off. Nonetheless, these results can still help taking informed decisions in such problems as well, as they give interesting insights and takeaways that practitioners of the field may take into account when taking informed decisions in designing multi-agent systems.

4 DIFFERENCE REWARDS POLICY GRADIENTS

One key problem that agents face that is not directly tackled by many MAPG methods is *multi-agent credit assignment* [20, 94, 178, 168]. With a shared reward signal, an agent cannot readily tell how its own actions affect the overall performance. This can lead to sub-optimal policies even with just a few agents. *Difference rewards* [169, 114, 26, 25] were proposed to tackle this problem: agents learn from a shaped reward that allows them to infer how their actions contributed to the shared reward value.

Only one MAPG method has incorporated this idea so far: Counterfactual Multiagent Policy Gradients (COMA) [40] is a state-of-the-art algorithm that does the differencing with a learned action-value function $Q_\omega(s,a)$. However, there are potential disadvantages to this approach: learning a proper representation that accurately represents the true approximated function is crucial, and it is especially so when the critic is used to predict values that goes beyond its training set as done in COMA to compute the difference rewards values. The learning process of a centralized action-value function is a difficult problem in general, because of some compounding factors:

- As with every other centralized controller, it scales poorly in the number of agents. When more agents are introduced, the complexity of the function that has to be represented grows exponentially with this number, rendering the learning problem more difficult (for example, requiring a larger a representation and more training samples) [24],
- Because of its learning target, *bootstrapping* may be problematic: in order to update a certain value estimates of other values are required, and if these are not accurate in turn the update to the considered value may be arbitrarily bad. This problem is even exacerbated by the use of neural networks as function approximators, as convergence guarantees of single-agent reinforcement learning algorithms does not generally hold for non-linear function approximation such as these [162, 86],
- Even if the values are indeed accurate, the *moving target problem* may still hinder the learning process. With neural networks as function approximators, a small update to the network weights may result in a totally different action to be selected by the agent, thus constantly changing the learning target of the action-value function. This problem can be partially eased by using a *target network* [86], but careful choice of its parameters may still be a difficulty.

To overcome these potential difficulties, this chapter takes inspiration from [25] and incorporates the *differencing of the reward function* into MAPG. *Difference rewards*

REINFORCE (Dr.Reinforce), a new MARL algorithm that combines decentralized policies learned with policy gradients via difference rewards that are used to provide gradients with information on each agent's individual contribution to the overall performance, is proposed. Additionally, Dr.ReinforceR, a version for settings where the reward function is not known upfront, is also presented here. In contrast to [25], Dr.ReinforceR exploits the CTDE paradigm and learns a centralized reward network to estimate difference rewards. Although the dimensionality of the reward function is the same as the Q-function, and similarly depends on joint actions, learning the reward function is a simple regression problem. It does not suffer from the moving target problem, which allows for faster training and improved performance. The obtained empirical results show that the proposed approaches can significantly outperform other MAPG methods, particularly with more agents.

4.1 METHODS

COMA learns a centralized action-value function critic $Q_\omega(s,a)$ to do the differencing and drive agents' policy gradients. However, learning such a critic using the TD-error in Equation 2.30 presents a series of challenges that may dramatically hinder final performance if they are not carefully tackled. The Q-values' updates rely on bootstrapping that can lead to inaccurate updates. Moreover, the target values for these updates are constantly changing because the other estimates used to compute them are also updated, leading to a moving target problem. This is exacerbated when function approximation is used, as these estimates can be indirectly modified by the updates of other Q-values. Target networks are used to try and tackle this problem [86], but these require careful tuning of additional parameters and may slow down convergence with more agents.

The proposed algorithm, named Dr.Reinforce, combines the REINFORCE [167] policy gradient method with a difference rewards mechanism to deal with credit assignment in cooperative multi-agent systems, thus avoiding the need of learning a critic.

4.1.1 *Dr.Reinforce*

If the shared reward function $R(s,a)$ is known, difference rewards can directly be used with policy gradients. Let us define the *difference return* ΔG_t^i for agent i as the discounted sum of the difference rewards $\Delta R^i(a_t^i|s_t,a_t^{-i})$ from time step t onward as:

$$\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i}) \triangleq \sum_{l=0}^{T-t-1} \gamma^l \Delta R^i(a_{t+l}^i|s_{t+l},a_{t+l}^{-i}), \tag{4.1}$$

where T is the length of the sampled trajectory and $\Delta R^i(a_t^i|s_t,a_t^{-i})$ is the difference rewards for agent i, computed using the aristocrat utility [169] as in Equation 2.25.

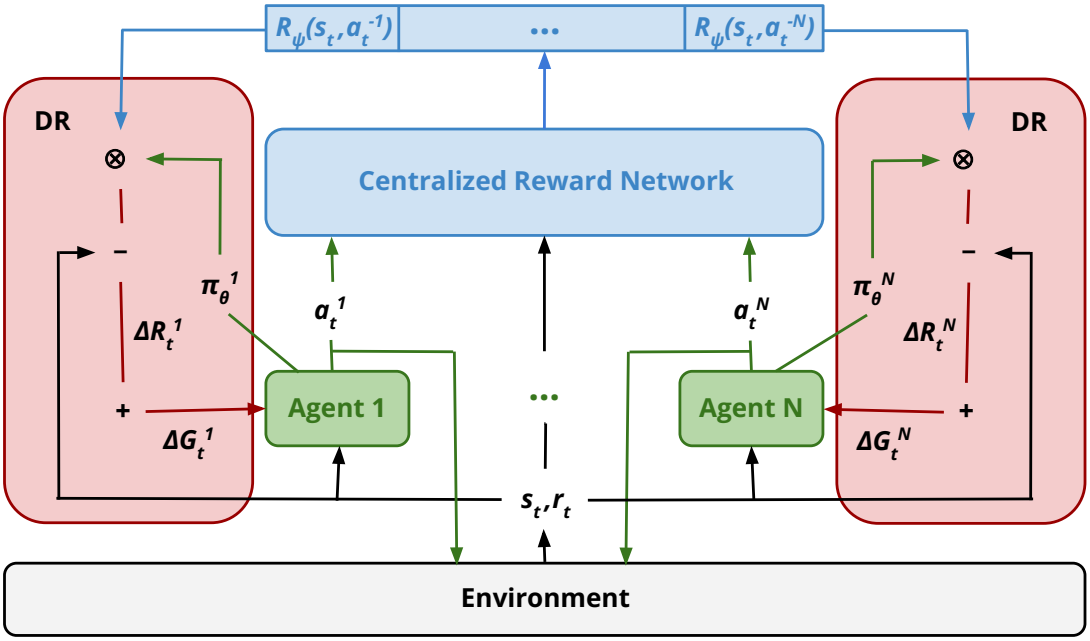


Figure 4.1: Schematic representation of the Dr.ReinforceR algorithm.

Please note that the subscript $t:T$ in the notation is a shorthand used to identify the sequence of values of given quantity from time step t up to (but not including) time step T.

To learn the decentralized policies π_{θ^i}, the proposed algorithm follows a modified version of the distributed policy gradient in Equation 2.28 that uses the difference return, optimizing each policy by using the update target:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t \Delta G_t^i(a_{t:T}^i | s_{t:T}, a_{t:T}^{-i}) \nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i | s_t)}_{g^{DR,i}}, \tag{4.2}$$

where ΔG_t^i is the difference return defined in Equation 4.1. This way, each policy is guided by an update that takes into account its individual contribution to the shared reward, and thus an agent takes into account the real value of its own actions. This signal is expected to drive the policies towards regions in which individual contributions are higher, and thus also the shared reward, since a sequence of actions improving ΔG_t^i also improves the global return G_t [1].

4.1.2 *Online Reward Estimation*

In many settings, complete access to the reward function to compute the difference rewards is not available. Thus, Dr.ReinforceR is proposed, which is similar to Dr.Reinforce but additionally learns a *centralized reward network* R_ψ, with parameters ψ, that is used to estimate the value $R(s, \langle a^i, a^{-i} \rangle)$ for every local action $a^i \in A^i$ for

agent i. Following the CTDE paradigm, this centralized network is only used during training to provide policies with learning signals, and is not needed during execution, when only the decentralized policies are used. The reward network receives as input the environment state s_t and the joint action of the agents a_t at time t, and is trained to reproduce the corresponding reward value $r_t \sim R(s_t,a_t)$ by minimizing a standard MSE regression loss:

$$\mathcal{L}_t(\psi) = \frac{1}{2}\left(r_t - R_\psi(s_t,a_t)\right)^2. \tag{4.3}$$

Although the dimensionality of the function $R(s,a)$ that is learned with the reward network is the same as that of $Q(s,a)$ learned by the COMA critic, growing exponentially with the number of agents as both depend of the joint action $a \in A = \times_{i=1}^{|D|} A^i$, learning R_ψ is a regression problem that does not involve bootstrapping or moving targets, thus avoiding many of the problems faced with an action-value function critic. Moreover, alternative representations of the reward function can be used to further improve learning speed and accuracy, e.g., by using factorizations [53].

The learned R_ψ can then be used to compute the difference rewards ΔR^i_ψ using the aristocrat utility [169] as:

$$\Delta R^i_\psi(a^i_t|s_t,a^{-i}_t) \triangleq r_t - \sum_{c^i \in A^i} \pi_{\theta^i}(c^i|s_t) R_\psi(s_t,\langle c^i,a^{-i}_t\rangle). \tag{4.4}$$

The second term of the r.h.s. of Equation 4.4 can be estimated for each agent i with a number of network evaluations that is linear in the size of the local action set A^i, as the actions of the other agents a^{-i}_t remains fixed, avoiding an exponential cost.

Let now redefine the difference return ΔG^i_t from Equation 4.1 as the discounted sum of the estimated difference rewards ΔR^i_ψ:

$$\Delta G^i_t(a^i_{t:T}|s_{t:T},a^{-i}_{t:T}) \triangleq \sum_{l=0}^{T-t-1} \gamma^l \Delta R^i_\psi(a^i_{t+l}|s_{t+l},a^{-i}_{t+l}). \tag{4.5}$$

4.2 THEORETICAL RESULTS

REINFORCE [167] suffers from high variance of gradients estimates because of the sampled estimation of the return. This can be accentuated in the multi-agent setting. Using an unbiased baseline is crucial to reducing this variance and improving learning [47, 141]. Here these concerns are addressed by showing that using difference rewards in policy gradient methods corresponds to subtracting an unbiased baseline from the policy gradient of each individual agent. Since this unbiased baseline does not alter the expected value of the update targets, applying difference rewards policy gradients to a common-reward MARL problem turns out to be same in expectation as using distributed policy gradient update targets. Such gradients' updates have

been shown to be equivalent to those of a joint gradient [112], which under some technical conditions is known to converge to a local optimum [142, 69].

Lemma 1. *In an MMDP, using difference return $\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i})$ as the learning signal for policy gradient in Equation 4.2 is equivalent to subtracting an unbiased baseline $B^i(s_{t:T},a_{t:T}^{-i})$ from the distributed policy gradients in Equation 2.28.*

Proof. Let start by rewriting $\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i})$ from Equation 4.1 as:

$$\begin{aligned}\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i}) = &\sum_{l=0}^{T-t-1}\gamma^l r_{t+l} \\ &- \sum_{l=0}^{T-t-1}\gamma^l \sum_{c^i\in A^i}\pi_{\theta^i}(c^i|s_{t+l})R(s_{t+l},\langle c^i,a_{t+l}^{-i}\rangle).\end{aligned} \tag{4.6}$$

Note that the first term on the r.h.s. of Equation 4.6 is the return G_t used in Equation 2.28. The second term on the r.h.s. of Equation 4.6 is defined as the baseline $B^i(s_{t:T},a_{t:T}^{-i})$:

$$B^i(s_{t:T},a_{t:T}^{-i}) = \sum_{l=0}^{T-t-1}\gamma^l \sum_{c^i\in A^i}\pi_{\theta^i}(c^i|s_{t+l})\cdot R(s_{t+l},\langle c^i,a_{t+l}^{-i}\rangle). \tag{4.7}$$

The total expected update target for agent i can thus be rewritten as:

$$\mathbb{E}_{\pi_\theta}\left[\hat{g}^{DR,i}\right] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i})\right] \tag{4.8}$$

(by definition of ΔG_t^i)

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)\left(G_t - B^i(s_{t:T},a_{t:T}^{-i})\right)\right] \tag{4.9}$$

(distributing the product)

$$\begin{aligned}&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)G_t\right. \\ &\left.- \left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)B^i(s_{t:T},a_{t:T}^{-i})\right]\end{aligned} \tag{4.10}$$

(by linearity of the expectation)

$$\begin{aligned}&= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)G_t\right] \\ &- \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right)B^i(s_{t:T},a_{t:T}^{-i})\right]\end{aligned}$$

$$= \mathbb{E}_{\pi_\theta}\left[\hat{g}^i\right] + \mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right]. \tag{4.11}$$

In order for the baseline to be unbiased, it has to be shown that the expected value of its update $\mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right]$ with respect to the policy π_θ is 0. Let $P_t^{\pi_\theta}(s_t) = \sum_{s_{t-1}\in S} P_{t-1}^{\pi_\theta}(s_{t-1}) \sum_{a_{t-1}\in A} \pi_\theta(a_{t-1}|s_{t-1})\ T(s_t|a_{t-1},s_{t-1})$ be the probability of the state at time step t to be s_t under the joint policy π_θ (with $P_0^{\pi_\theta}(s_0) = \rho(s_0)$ and ρ is the initial state distribution), it follows that:

$$\mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right] \triangleq -\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right) B^i(s_{t:T},a_{t:T}^{-i})\right] \tag{4.12}$$

(by expanding the expectation)

$$\begin{aligned} &= -\sum_{t=0}^{T-1}\sum_{s_t\in S} P_t^{\pi_\theta}(s_t) \sum_{a_t^{-i}\in A^{-i}} \pi_{\theta^{-i}}(a_t^{-i}|s_t) \sum_{a_t^i\in A^i} \pi_{\theta^i}(a_t^i|s_t) \\ &\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|s_t)\right) \sum_{s_{t+1:T},a_{t+1:T}} \\ &\prod_{l=1}^{T-t-1} T(s_{t+l}|a_{t+l-1},s_{t+l-1}) \cdot \pi_\theta(a_{t+l}|s_{t+l})\ B^i(s_{t:T},a_{t:T}^{-i}) \end{aligned} \tag{4.13}$$

(by applying the inverse log trick)

$$\begin{aligned} &= -\sum_{t=0}^{T-1}\sum_{s_t\in S} P_t^{\pi_\theta}(s_t) \sum_{a_t^{-i}\in A^{-i}} \pi_{\theta^{-i}}(a_t^{-i}|s_t) \sum_{a_t^i\in A^i} \left(\nabla_{\theta^i}\pi_{\theta^i}(a_t^i|s_t)\right) \\ &\sum_{s_{t+1:T},a_{t+1:T}} \prod_{l=1}^{T-t-1} T(s_{t+l}|a_{t+l-1},s_{t+l-1}) \cdot \pi_\theta(a_{t+l}|s_{t+l})\ B^i(s_{t:T},a_{t:T}^{-i}) \end{aligned} \tag{4.14}$$

(by moving the gradient outside the policy sum)

$$\begin{aligned} &= -\sum_{t=0}^{T-1}\sum_{s_t\in S} P_t^{\pi_\theta}(s_t) \sum_{a_t^{-i}\in A^{-i}} \pi_{\theta^{-i}}(a_t^{-i}|s_t) \left(\nabla_{\theta^i} \sum_{a_t^i\in A^i} \pi_{\theta^i}(a_t^i|s_t)\right) \\ &\sum_{s_{t+1:T},a_{t+1:T}} \prod_{l=1}^{T-t-1} T(s_{t+l}|a_{t+l-1},s_{t+l-1}) \cdot \pi_\theta(a_{t+l}|s_{t+l})\ B^i(s_{t:T},a_{t:T}^{-i}) \end{aligned} \tag{4.15}$$

(policy probabilities sum up to 1)

$$\begin{aligned} &= -\sum_{t=0}^{T-1}\sum_{s_t\in S} P_t^{\pi_\theta}(s_t) \sum_{a_t^{-i}\in A^{-i}} \pi_{\theta^{-i}}(a_t^{-i}|s_t)\ \nabla_{\theta^i} 1 \\ &\sum_{s_{t+1:T},a_{t+1:T}} \prod_{l=1}^{T-t-1} T(s_{t+l}|a_{t+l-1},s_{t+l-1}) \cdot \pi_\theta(a_{t+l}|s_{t+l})\ B^i(s_{t:T},a_{t:T}^{-i}) \\ &= 0. \end{aligned} \tag{4.16}$$

Therefore, using the baseline in Equation 4.7 reduces the variance of the updates [47] but does not change their expected value, as it is unbiased and its expected update target $\mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right] = 0$. □

Corollary. *Using the estimated reward network R_ψ to compute the baseline in Equation 4.7 still results in an unbiased baseline.*

Proof. Let rewrite $\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i})$ from Equation 4.5 as:

$$\begin{aligned}\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i}) = &\sum_{l=0}^{T-t-1}\gamma^l r_{t+l} \\ &- \sum_{l=0}^{T-t-1}\gamma^l \sum_{c^i\in A^i}\pi_{\theta^i}(c^i|s_{t+l})R_\psi(s_{t+l},\langle c^i,a_{t+l}^{-i}\rangle).\end{aligned} \tag{4.17}$$

for which the second term on the r.h.s. of Equation 4.17 is defined as the baseline $B_\psi^i(s_{t:T},a_{t:T}^{-i})$:

$$B_\psi^i(s_{t:T},a_{t:T}^{-i}) = \sum_{l=0}^{T-t-1}\gamma^l \sum_{c^i\in A^i}\pi_{\theta^i}(c^i|s_{t+l})\cdot R_\psi(s_{t+l},\langle c^i,a_{t+l}^{-i}\rangle). \tag{4.18}$$

It can be observed that the derivation of Equation 4.16 still holds, as the use of the reward network R_ψ rather than the true reward function $R(s,a)$ does not alter it. Therefore, the baseline $B_\psi^i(s_{t:T},a_{t:T}^{-i})$ is again unbiased and does not alter the expected value of the updates. □

Theorem 1. *In an MMDP, given the conditions on function approximation detailed in [142], using Dr.Reinforce update target as in Equation 4.2, the series of parameters $\{\theta_t = \langle\theta_t^1,\dots,\theta_t^N\rangle\}_{t=0}^k$ converges in the limit such that the corresponding joint policy π_{θ_t} is a local optimum:*

$$\lim_{k\to\infty}\inf_{\{\theta_t\}_{t=0}^k}||\hat{g}^{DR}|| = 0 \qquad w.p.\ 1.$$

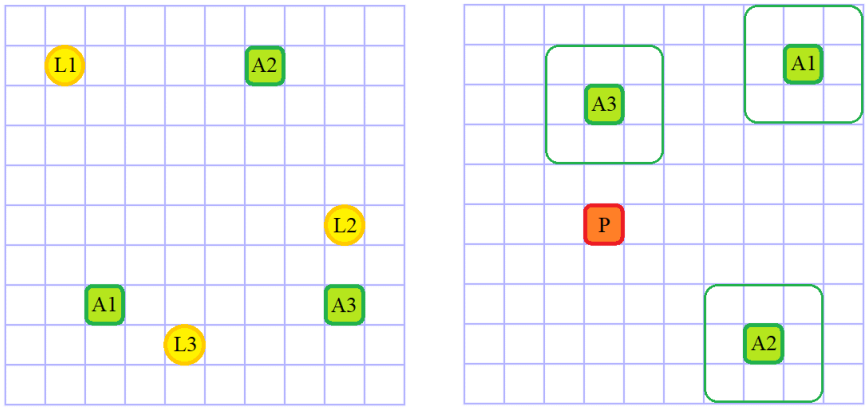
Proof. To prove convergence, it needs to be proved that:

$$\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{DR}\right] = \mathbb{E}_{\pi_{\theta_t}}\left[\sum_{i=0}^{N}\hat{g}^{DR,i}\right] = \nabla_{\theta_t}V(\theta_t). \tag{4.19}$$

Let rewrite the total expected update target as:

$$\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{DR,i}\right] = \mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{i}\right] + \mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{B,i}\right] \tag{4.20}$$

as in Equation 4.11, and by Lemma 1 it results that $\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{B,i}\right] = 0$. Therefore, the overall expected update $\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{DR,i}\right]$ for agent i reduces to $\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}^{i}\right]$, that is equal to the distributed policy gradient update target in Equation 2.28. These updates for all the agents have been proved to be equal to these of a centralized policy gradient agent $\mathbb{E}_{\pi_{\theta_t}}\left[\hat{g}\right]$ by Theorem 1 in [112], and therefore converge to a local optimum of $\nabla_{\theta_t}V(\theta_t)$ by Theorem 3 in [142]. □



(a) Multi-Rover (b) Predator-Prey

Figure 4.2: Schematic representation of the two gridworld domains. Agents are green, landmarks are yellow, and the prey is red.

4.3 GRIDWORLD EXPERIMENTS

The interest of this work is on investigating the following research questions:

RQ1: How does Dr.Reinforce compare to existing approaches?

RQ2: How does the use of a learned reward network R_ψ instead of a known reward function affect performance?

RQ3: Is learning the Q-function (as in COMA) more difficult than learning the reward function $R(s,a)$ (as in Dr.ReinforceR)?

To investigate these questions, the proposed methods are tested on two gridworld environments with shared reward: the multi-rover domain, an established multi-agent cooperative domain [26], in which agents have to spread across a series of landmarks, and a variant of the classical predator-prey problem with a randomly moving prey [146].

4.3.1 *Comparison to Baselines*

The proposed methods are compared to a range of other policy gradient methods: independent learners using REINFORCE to assess the benefits of using a difference rewards mechanism, labelled PG. These also compared against a standard actor-critic algorithm [69] with decentralized actors and a centralized action-value function critic to show that their improvements are not only due to the centralized information provided to the agents during training, denoted as CentralQ here. The main comparison is with COMA [40], a state-of-the-art difference rewards method using the Q-function

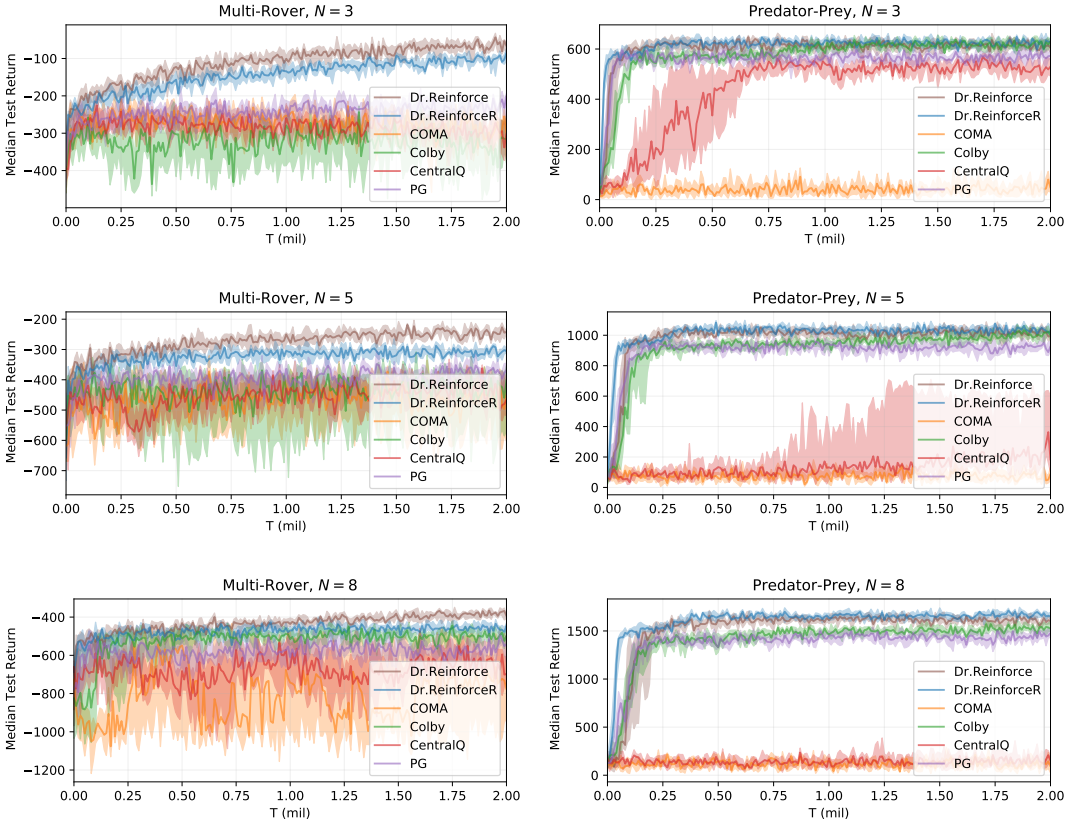


Figure 4.3: Training curves on the multi-rover domain (left) and the predator-prey problem (right), showing the median return and $25-75\%$ percentiles across seeds.

for computing the differences. Finally, these methods are compared against the algorithm proposed in [25], to show the benefit of learning a centralized reward network to estimate the difference rewards as in Dr.ReinforceR. This method has been adapted to use policy gradients instead of evolutionary algorithms to optimise the policies to not conflate the comparisons with the choice of a policy optimizer where possible, and only focus on the effect of using difference rewards during learning.

Multi-Rover Domain

In this domain, a team of N agents is placed into a 10×10 gridworld with a set of N landmarks. The aim of the team is to spread over all the landmarks and cover them (which agent covers which landmark is not important): the reward received by the team depends on the distance of each landmark to its closest agent, and a penalty if two agents collide (reach the same cell simultaneously) during their movements is also applied. Each agent observes its relative position with respect to all the other agents and landmarks, and can move in the four cardinal directions or stand still (actions are deterministic and do no fail). Figure 4.3 (left) reports the median return and $25-75\%$ percentiles (shaded area in the plots) across 10 independent runs obtained by the compared methods on a team of increasing size, to investigate scaling

to larger multi-agent systems.

It can be observed that both Dr.Reinforce and Dr.ReinforceR are always outperforming all of the other compared baselines on this domain. Also, Dr.ReinforceR is generally matching the upper bound given by Dr.Reinforce (that represents a limit case when the centralized reward network R_ψ has perfectly converged to the true reward function). However, the wide gap between these two algorithms and the other baselines when $N = 3$ reduces when more agents are introduced in the system, possibly pointing out that also these methods start to struggle in achieving optimal and coordinated behaviours on larger instances of this domain. When more agents are present, the gridworld becomes quite crowded: an explanation for this loss in performance is that the difference rewards signal pushes each agent towards the landmark that is furthest from all of the agents, and thus contributing the most to the negative reward value, in an attempt to mitigate this problem, but letting another landmark increase its negative contribution in turn. Coordination is key to efficiently solve this domain, and achieving such a coordination may be difficult in larger settings.

Moreover, even if the reward network learns a good representation, the synergy between this and the agents' policies has to be carefully considered: the reward network has to converge properly before the policies got stuck into a local optimum, or it could be the case that these will not be able to escape it even if the gradients signals are then accurate enough. However, the simpler learning problem used to provide signals to the agents' policies, as opposed to the very complex learning of the action-value function critic used by COMA, proves effective in speeding up learning and achieve higher returns, even in difficult settings with many agents where all the other policy gradient methods seem to fail as well. Computing the difference rewards requires very accurate reward estimates, so if the reward network does not exhibit appropriate generalization capabilities it may end up overfitting on the reward values encountered during training but not being able to give correct predictions beyond those. It is true however that also difference rewards methods using the action-value function have the same requirements.

Predator-Prey

In this version of the classical predator-prey problem, a team of N predators has to pursue a single prey for as long as possible in a 10×10 gridworld. Each predator has a range of sight of one cell in each direction from its current position: if the prey is into this range, the whole team receives a positive reward bonus, otherwise they do not receive any reward. Each agent observes its relative position with respect to the other agents and the prey itself and can move in the four cardinal directions or stand still. The prey selects actions uniformly at random (actions for both the agents and the prey are again guaranteed to succeed). Figures 4.3 (right) shows median

return and $25-75\%$ percentiles across 10 independent runs with teams comprising an increasing number of predators.

Also in this environment, Dr.ReinforceR is outperforming all the other compared methods, achieving performance that is equal or close to these of the Dr.Reinforce upper bound. On one hand, some of the other baselines are also performing well: PG and Colby are almost performing or-par with the two above algorithms, even on larger instances of the problem. This is probably due to the less strict coordination requirements of the predator-prey problem compared to the previous multi-rover domain: each agent is independently contributing towards the common goal, and thus simply needs to optimize its own behaviour by learning how to reach and stay on the prey in order to improve global performances.

On the other hand, COMA is performing extremely poorly, being outperformed even by the simple CentralQ (that has slowly learned something in the simpler case with $N = 3$). This points out how accurately learning an optimal Q-function may be problematic in many settings, even more so on a sparse setting such as this, in which the agents are only perceiving rewards if some of them are effectively on the prey. If the Q-function converges to a sub-optimal solution and keeps pushing the agents towards a local optimum, the policies may struggle to escape from it afterwards and in turn push the action-value function towards a worst approximation. Moreover, to compute the counterfactual baseline in COMA, estimates of Q-values need to be accurate even on state-action pairs that the policies do not visit often, further exacerbating this problem. From this side, learning the reward function to compute the difference rewards is an easier learning problem, cast as a regression task and not involving bootstrapped estimates or a moving target, and thus can improve policy gradient performance providing them with better learning signals in achieving high return behaviours with no further drawback.

4.3.2 *Analysis*

The results of the proposed experiments show the benefits of learning the reward function over the more complex Q-function, leading to faster policy training and improved final performances, but also that this is not always an easy task and it can present issues on its own that can hinder the learning of an optimal joint policy. Indeed, although not suffering from the moving target problem and no bootstrapping is involved, learning the reward function online together with the policies of the agents can lead to biases of the learned function due to the agents behaviours. These biases could push the training samples towards a specific region of the true reward function, hindering the generalization capacity of the learned reward network and in turn leading to worst learning signal for the policies themselves, that can get stuck into a sub-optimal region. Similarly, this problem can appear when a centralized action-value critic is used to drive the policy gradients.

To investigate the claimed benefits of learning the reward function rather the Q-function, let us now analyse the accuracy of the learned representations on the two proposed gridworld domains by sampling a set of different trajectories from the execution of the corresponding policies and comparing the predicted values from the reward network $R_\psi(s,a)$ of Dr.ReinforceR and the $Q_\omega(s,a)$ critic from COMA to the real ground-truth values of the reward function and the Q-function respectively. This has been called the *on-policy dataset*, representing how correctly can the reward network and the critic represent the values of state-action pairs encountered during their training phase. Moreover, both Dr.ReinforceR and COMA rely on a difference rewards mechanism and thus need to estimate values for state-action pairs that are only encountered infrequently (or not at all) in order to compute correct values to drive the policy gradients. To investigate the generalization performances of the learned representations, let also analyse the prediction error on a *off-policy dataset*, by sampling uniformly across the entire action-state space $S \times A$ and again comparing the predicted values from the learned reward function $R_\psi(s,a)$ of Dr.ReinforceR and the $Q_\omega(s,a)$ critic from COMA to their corresponding ground-truth values. Please note that, not knowing the true Q-function for the proposed problems to compare against, these have been approximated that via 100 rollouts sampled starting from the current state-action sample and following the corresponding learned policies afterwards. Figure 4.4 shows the mean and standard deviation of the prediction error (PE) distribution of these networks. All the prediction errors have been normalized by the value of $r_{max} - r_{min}$ (respectively $q_{max} - q_{min}$ for COMA critic) for each environment and number of agents individually, so that the resulting values are comparable across the two different methodologies and across different settings.

These plots give us some insights on the performance reported in Section 4.3.1. Dr.ReinforceR is in general achieving improved performances with respect to the compared baselines, and the low prediction error of its reward network on the two problems may be an explanation for this: with correct value estimates, the learning signals provided to the policy gradients are better in turn, and thus lead to higher-return behaviours. Also the variance is low, meaning that most of the sampled values are consistently predicted correctly and the network exhibits good generalization performances across the increasing number of agents on both datasets. This generalization capacity of the learned approximation also explains why Dr.ReinforceR is in general matching the Dr.Reinforce upper bound: the difference rewards mechanism requires multiple predictions to compute the agents' signals and, if these are not accurate enough, the resulting values may be completely wrong and push the agents towards sub-optimal policies in turn.

The prediction errors for COMA action-value critic instead are higher, especially on the multi-rover domain, where the errors do not scale so gracefully in the number of agents even on the on-policy dataset. It can be observed that the critic network is biased towards overestimating most of the samples for the multi-rover domain, while

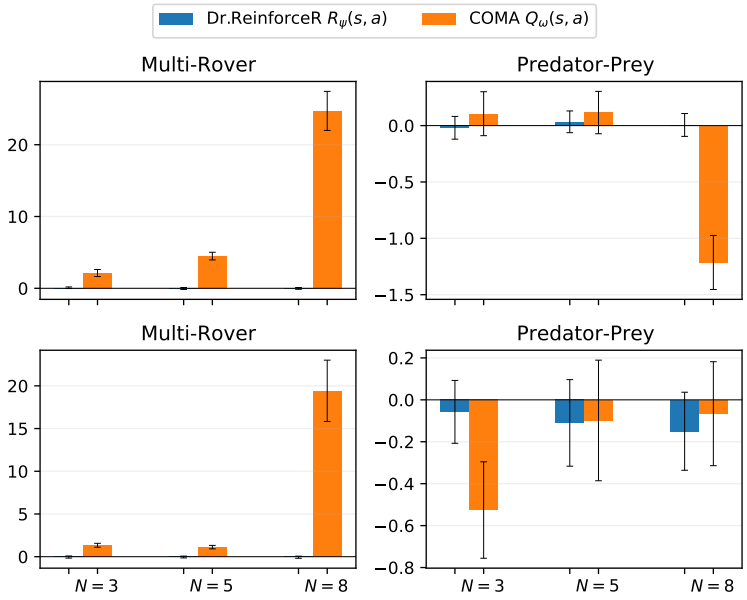


Figure 4.4: Normalized mean prediction error and standard deviation for Dr.ReinforceR reward network R_ψ and COMA critic Q_ω on the on-policy dataset (first row) and the off-policy dataset (second row), for the two environments.

instead underestimates them for predator-prey (especially more so on the off-policy dataset, where non-encountered state-action pairs may be sampled), thus resulting in bad estimations of the counterfactual baseline. On the predator-prey environment, it seems that COMA critic quickly overfits to the Q-function of a sub-optimal joint policy, resulting in a very low prediction error on the off-policy dataset when the number of agents increases (and most of the samples indeed lead to no rewards trajectories), that does not seem able to give good signals to the agents' policies and leads them to get stuck into this poor local optimum in turn. These results can also explain why COMA is performing worse than CentralQ on this domain: if the critic is not accurate or is representing the value of a poor policy (as it can be hypothesized for the above results), COMA requirement of more estimations from it in order to compute the counterfactual baseline only exacerbates this problem and further hinders the final performance.

Finally, the effect of noise on computation of the difference rewards are investigated. Generally, an accurate reward value for every agent's action is needed to compute correct difference rewards. The reward network R_ψ is an approximation of the true reward function $R(s,a)$ and can therefore give noisy estimates that could dramatically affect the resulting computation. To investigate this, noise sampled from different processes is added to the reward values of the agent's different actions that are obtained from the environment. These are used to compute the baseline (the second term of the r.h.s. in Equation 2.25, as this is the only term for which R_ψ is used in Equation 4.4), and the resulting difference rewards are compared with the true ones for a generic agent i under a uniform policy $\pi_{\theta^i}(a^i|s) = \frac{1}{|A^i|}$. Figure 4.5 reports

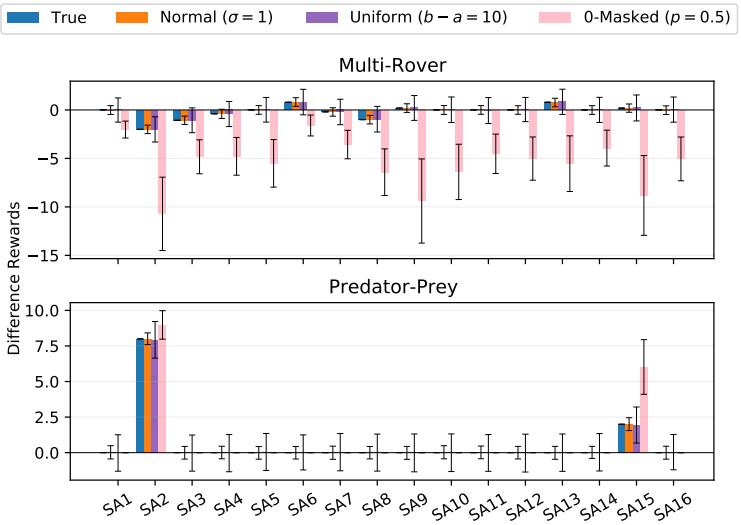


Figure 4.5: Mean and variance of difference rewards for a set of samples under different noise profiles.

the mean value and variance over 1000 noise samples of a set of sampled state-action pairs from the reward function of the two investigated domains with $N = 3$ agents.

It can be observed how different noise processes differently affect the resulting difference rewards. For example, in both environments, the difference rewards mechanism is quite resistant against noise from a normal or a uniform distribution. This is probably due to the symmetricity of these noises, that tends to cancel out with each other. However, a masking kind of noise seems to be more detrimental for difference rewards evaluation: cancelling out some of the reward values definitely changes the computation and gives wrong estimates. This is worse in the multi-rover domain, in which the reward function is dense, while for the predator-prey environment and its sparse reward function it seems to be less harming.

These two observations together help explain why Dr.ReinforceR outperforms COMA on the two proposed environments: learning the reward function $R(s,a)$ is easier than learning the Q-function and, although function approximation introduces noise, the difference rewards mechanism is resistant against common types of noise and still provides useful signals to policy gradients. Therefore, if one is able to learn a good approximation of the reward, the proposed algorithm learns better and more reliable policies than other policy gradient algorithms, without the difficulties of learning the Q-function.

5 EXTENDING DR.REINFORCE TO PARTIALLY OBSERVABLE SETTINGS

Full observability of the environment as in MMDPs is a desirable property, but in many real-world situations [177, 151, 123] such a strong assumption is often unrealistic. The complexity of the environment itself or the limited sensing or communication capabilities available are usually transforming such problems into partially observable settings from the perspective of the agents. In these, the agents cannot directly observe the state of the environment, but instead are provided with a local and possibly noisy observation that represents only a limited amount of information about the underlying environmental state itself. Policy gradient methods are one of the most used family of algorithms to tackle partially observable settings, because of their straightforward applicability and little changes required in order to work [166, 40, 79, 126, 180, 170].

Chapter 4 introduced Dr.Reinforce, an algorithm that tackles the multi-agent credit assignment problem by combining policy gradients and a difference rewards mechanism and computing such a differencing on the true reward function of the system (as opposed to COMA, that uses the Q-function instead), and Dr.ReinforceR, that additionally learns a centralized reward network to estimate the values required to compute the differencing when complete knowledge of the reward function is not available. Theoretical results showed the convergence properties of these algorithms to an optimal solution in the fully observable case, as in practice the difference rewards mechanism corresponds to subtracting an unbiased baseline to the standard distributed policy gradient.

In this chapter the two proposed algorithms are extended to address the partially observable case, and similar theoretical guarantees are proved to be lost when the agents' policies condition on their local action-observation histories rather than on the environment state. Nonetheless, improved empirical performances over COMA and other common policy gradient baselines are shown on the popular SMAC domain [123], highlighting the applicability of such methods even when the convergence guarantees are lost.

5.1 METHODS

Policy gradient algorithms can easily be adapted to work under partial observability by simply replacing the environment state s used by the agents policies π_{θ^i} with the corresponding agent's local action-observation history h_t^i. The distributed policy

gradient in Equation 2.28 thus becomes:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t G_t \nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i|h_t^i)}_{\hat{g}^i}. \tag{5.1}$$

Similarly, it is straightforward to also adapt Dr.Reinforce to work in Dec-POMDPs by simply adjusting the policy terms that appear in Equation 4.1 and Equation 4.2 to condition on the agents' local action-observation history h_t^i. The difference return ΔG_t^i in thus defined as:

$$\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i},h_{t:T}^i) \triangleq \sum_{l=0}^{T-t-1} \gamma^l \Delta R^i(a_{t+l}^i|s_{t+l},a_{t+l}^{-i},h_{t+l}^i), \tag{5.2}$$

while the decentralized policies are learned by using the update target:

$$\theta^i \leftarrow \theta^i + \alpha \underbrace{\sum_{t=0}^{T-1} \gamma^t \Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i},h_{t:T}^i) \nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i|h_t^i)}_{g^{DR,i}}. \tag{5.3}$$

When complete access to the reward function is not available, a modified version of Dr.ReinforceR can be applied. The centralized reward network R_ψ, by following the CTDE paradigm, can still be learned in the same way as in Equation 4.3 and condition on the environment state $s \in S$, as it is not required during execution. While centralized state-based critics like COMA one have been proved to not always improve performances and introduce bias or variance in the policy gradients [80, 5, 81], learning a centralized reward network that conditions on the true state of the environment even in a partially observable setting is theoretically sound, as the true reward function of the environment does indeed depend on the state and the joint action of the agents.

To use the centralized reward network for estimates, it is enough to adapt Equation 4.4 as done before, thus obtaining:

$$\Delta R_\psi^i(a_t^i|s_t,a_t^{-i},h_t^i) \triangleq r_t - \sum_{c^i \in A^i} \pi_{\theta^i}(c^i|h_t^i) R_\psi(s_t,\langle c^i,a_t^{-i}\rangle), \tag{5.4}$$

and consequently adjust Equation 4.5 as:

$$\Delta G_t^i(a_{t:T}^i|s_{t:T},a_{t:T}^{-i},h_{t:T}^i) \triangleq \sum_{l=0}^{T-t-1} \gamma^l \Delta R_\psi^i(a_{t+l}^i|s_{t+l},a_{t+l}^{-i},h_t^i). \tag{5.5}$$

5.2 THEORETICAL RESULTS

Above, Dr.Reinforce has been adapted to partially observable settings, and intuitively it can improve learning by providing individual agents with a better learning signal.

In these settings, using difference rewards as the agents' learning signal induces a partially observable stochastic game [56, 98] $\hat{\mathcal{P}} = \langle D, S, \{A^i\}_{i=1}^{|D|}, T, \{\Delta R^i\}_{i=1}^{|D|}, \{O^i\}_{i=1}^{|D|}, Z\rangle$ in which the cooperating agents do not receive the same reward after each time step. Even though difference rewards are aligned with the true reward values [1, 93], for these games convergence to an optimal solution is not immediate.

When agents are required to base their decisions on their local action-observation history h_t^i, the same result on an unbiased baseline derived in Section 4.2 for the fully observable case does not hold anymore. Generally speaking, this is due to the Monte-Carlo nature of the difference return ΔG_t^i, that requires future quantities in order to compute the value of the baseline. The local histories for the episode time steps (used to compute the aristocrat utility values in the r.h.s. of Equation 5.2) are now strictly depending on the actions selected at the previous time steps, and thus break this independence of the baseline from the currently selected action.

Observation. *In a Dec-POMDP setting, using difference return* $\Delta G_t^i(a_{t:T}^i | s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)$ *as the learning signal for policy gradients in Equation 5.3 is in general not equivalent to subtracting an unbiased baseline* $B^i(s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)$ *from the distributed policy gradient in Equation 2.28.*

Proof. Let start by rewriting $\Delta G_t^i(a_{t:T}^i | s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)$ from Equation 5.2 as:

$$\begin{aligned}\Delta G_t^i(a_{t:T}^i | s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i) = &\sum_{l=0}^{T-t-1} \gamma^l r_{t+l} \\ &- \sum_{l=0}^{T-t-1} \gamma^l \sum_{c^i \in A^i} \pi_{\theta^i}(c^i | h_{t+l}^i) R(s_{t+l}, \langle c^i, a_{t+l}^{-i} \rangle).\end{aligned} \tag{5.6}$$

Note that the first term on the r.h.s. of Equation 5.6 is the return G_t used in Equation 2.28. Let us then define the second term on the r.h.s. of Equation 5.6 as the baseline $B^i(s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)$:

$$B^i(s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i) = \sum_{l=0}^{T-t-1} \gamma^l \sum_{c^i \in A^i} \pi_{\theta^i}(c^i | h_{t+l}^i) \cdot R(s_{t+l}, \langle c^i, a_{t+l}^{-i} \rangle). \tag{5.7}$$

The total expected update target for agent i can thus be rewritten as:

$$\mathbb{E}_{\pi_\theta}\left[\hat{g}^{DR,i}\right] = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \left(\nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i | h_t^i)\right) \Delta G_t^i(a_{t:T}^i | s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)\right] \tag{5.8}$$

(by definition of ΔG_t^i)

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1} \left(\nabla_{\theta^i} \log \pi_{\theta^i}(a_t^i | h_t^i)\right) \left(G_t - B^i(s_{t:T}, a_{t:T}^{-i}, h_{t:T}^i)\right)\right] \tag{5.9}$$

(distributing the product)

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)G_t\right.$$
$$\left.-\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i)\right] \tag{5.10}$$

(by linearity of the expectation)

$$= \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)G_t\right]$$
$$-\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i)\right]$$
$$= \mathbb{E}_{\pi_\theta}\left[\hat{g}^i\right] + \mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right]. \tag{5.11}$$

In order to show that the baseline is unbiased the expected value of its update $\mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right]$ with respect to the policy π_θ should be 0. Let $P^{\pi_\theta}(h_t) = P^{\pi_\theta}(h_{t-1}) \cdot \pi_\theta(a_{t-1}|h_{t-1})\sum_{s_t\in S}P_t^{\pi_\theta}(s_t) \cdot Z(o_t,s_t)$ (with $P^{\pi_\theta}(h_0) = \sum_{s_0\in S}Z(o_0|s_0)\rho(s_0)$ and $\rho(s_0)$ the initial state distribution) be the joint action-observation history distribution. Let also define the *complete system history* $\hat{h}_t = \langle h_t,a_t,s_{0:t}\rangle \in \hat{\mathcal{H}}_t$, so that $P^{\pi_\theta}(\hat{h}_t) = P^{\pi_\theta}(h_t)\cdot\pi_\theta(a_t|h_t)\cdot\prod_{l=0}^{t}P_l^{\pi_\theta}(s_l)$, it follows that:

$$\mathbb{E}_{\pi_\theta}\left[\hat{g}^{B,i}\right] \triangleq -\mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{T-1}\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i)\right] \tag{5.12}$$

(by expanding the expectation)

$$= -\sum_{t=0}^{T-1}\sum_{\hat{h}_t\in\hat{\mathcal{H}}_t}P^{\pi_\theta}(\hat{h}_t)\left(\nabla_{\theta^i}\log\pi_{\theta^i}(a_t^i|h_t^i)\right)$$
$$\sum_{\hat{h}_T\in\hat{\mathcal{H}}_T}P^{\pi_\theta}(\hat{h}_T|\hat{h}_t)\;B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i) \tag{5.13}$$

(by applying the inverse log trick)

$$= -\sum_{t=0}^{T-1}\sum_{h_t\in\mathcal{H}_t}P^{\pi_\theta}(h_t)\sum_{a_t^{-i}\in A^{-i}}\pi_{\theta^{-i}}(a_t^{-i}|h_t^{-i})$$
$$\sum_{a_t^i\in A^i}\left(\nabla_{\theta^i}\pi_{\theta^i}(a_t^i|h_t^i)\right)\sum_{\hat{h}_T\in\hat{\mathcal{H}}_T}P^{\pi_\theta}(\hat{h}_T|\hat{h}_t)\;B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i) \tag{5.14}$$

(by moving the gradient outside the policy sum)

$$\neq -\sum_{t=0}^{T-1}\sum_{h_t\in\mathcal{H}_t} P^{\pi_\theta}(h_t)\sum_{a_t^{-i}\in A^{-i}}\pi_{\theta^{-i}}(a_t^{-i}|h_t^{-i})$$

$$\left(\nabla_{\theta^i}\sum_{a_t^i\in A^i}\pi_{\theta^i}(a_t^i|h_t^i)\right)\sum_{\hat{h}_T\in\hat{\mathcal{H}}_T} P^{\pi_\theta}(\hat{h}_T|\hat{h}_t)\; B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i)) \tag{5.15}$$

$$\tag{5.16}$$

The gradient cannot be moved outside of the sum now (as done in Equation 4.16), because of the baseline B^i depending on the policy parameters via the agent action a_t^i included in the histories $h_{t+1:T}^i$. The sum over the policy term is therefore a weighted summation over different baseline values, and these in general do not sum up to 0, and thus the baseline is in general not unbiased (although problems for which the summation is 0 in any case may exist, and in these special cases the baseline is still unbiased). □

The result in the above Lemma shows that using the baseline in Equation 5.7 alter the expected value of the overall gradient, as the baseline $B^i(s_{t:T},a_{t:T}^{-i},h_{t:T}^i)$ is not unbiased, and thus the policy gradients are not guaranteed to converge to the same local optimum of the distributed policy gradient [112].

5.3 STARCRAFT II EXPERIMENTS

Although there is no theoretical guarantee on the convergence of the proposed methods under partial observability, these might still work well in practice. Therefore, application of the proposed methods on the StarCraft II multi-agent challenge (SMAC[1]) [123] has been investigated to show good empirical performances. StarCraft II[2] is an highly complex strategic game from Blizzard[3] in which two opposing factions (either player vs. player or player vs. game AI) compete to gather resources, build structures and fight against the opponent army to win the game. The game can be divided into two different aspects:

1. Macromanagement: the set of choices that involves resource management and the game economy;

2. Micromanagement: the fine-grained control of army units during a fight.

SMAC focuses on the micromanagement aspect of the game, by providing a wide set of diverse scenarios, each with a different number and different types of units fighting against an opposing team controlled by the game AI. Each of the player's units is controlled by an individual agent (no centralized controller is allowed), and

1 SMAC is available at `https://github.com/oxwhirl/smac`
2 StarCraft II official website: `https://starcraft2.com/`
3 Blizzard official website: `https://www.blizzard.com/`

(a) 2c_vs_64zg

(b) so_many_baneling

Figure 5.1: Two example SMAC scenarios with different types of units and their remaining health. Source: Samvelyan et al. 2019.

has to select one of its available actions based only on its local view of the battlefield. Such a view is composed of information on the ally and enemy units that are within a limited observation radius, that is the range of sight for a given unit. The other units that are outside such a range are not observed directly, and thus the environment is partially observable from each agent's own perspective. Units have a given amount of health (and some types also have an additional shied) that is reduced when hit by opposing fire, and after which a unit is considered dead and removed from the game. Agents are rewarded with a small positive value for killing an enemy unit, and with a larger one for winning a battle (although the reward function can be configured to also give penalties for a dead ally or even be entirely sparse and only reward agents at the end of the episode). All the agents share the same reward value, as the environment is cooperative. Agents are required to learn highly complex and coordinated strategies in order to maximize their damage to opposing units and reduce the incoming damage, in order to kill all of these (before the opposing team

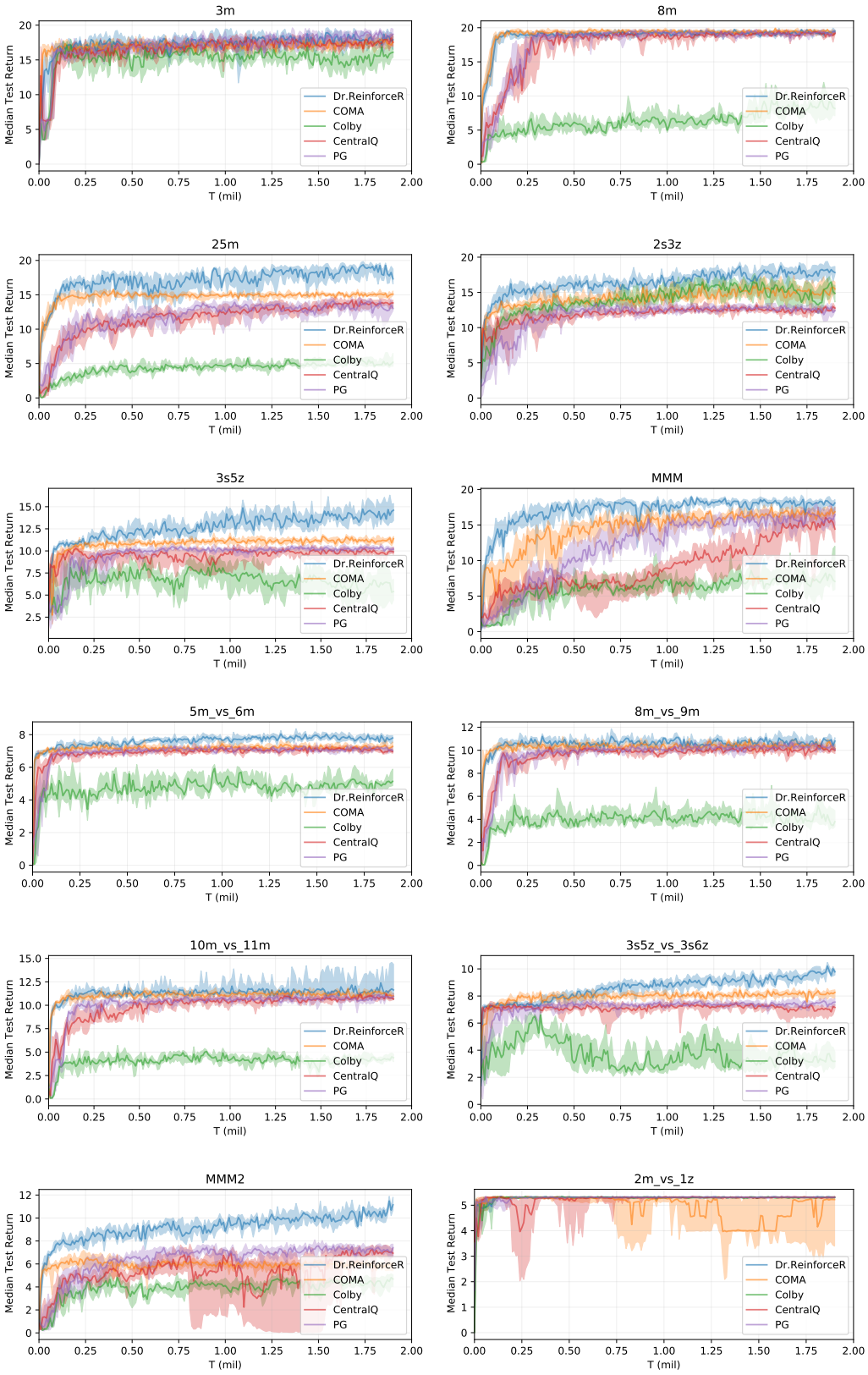


Figure 5.2: Training curves on the entire set of SMAC scenarios, showing the median return and $25-75\%$ percentiles across seeds.

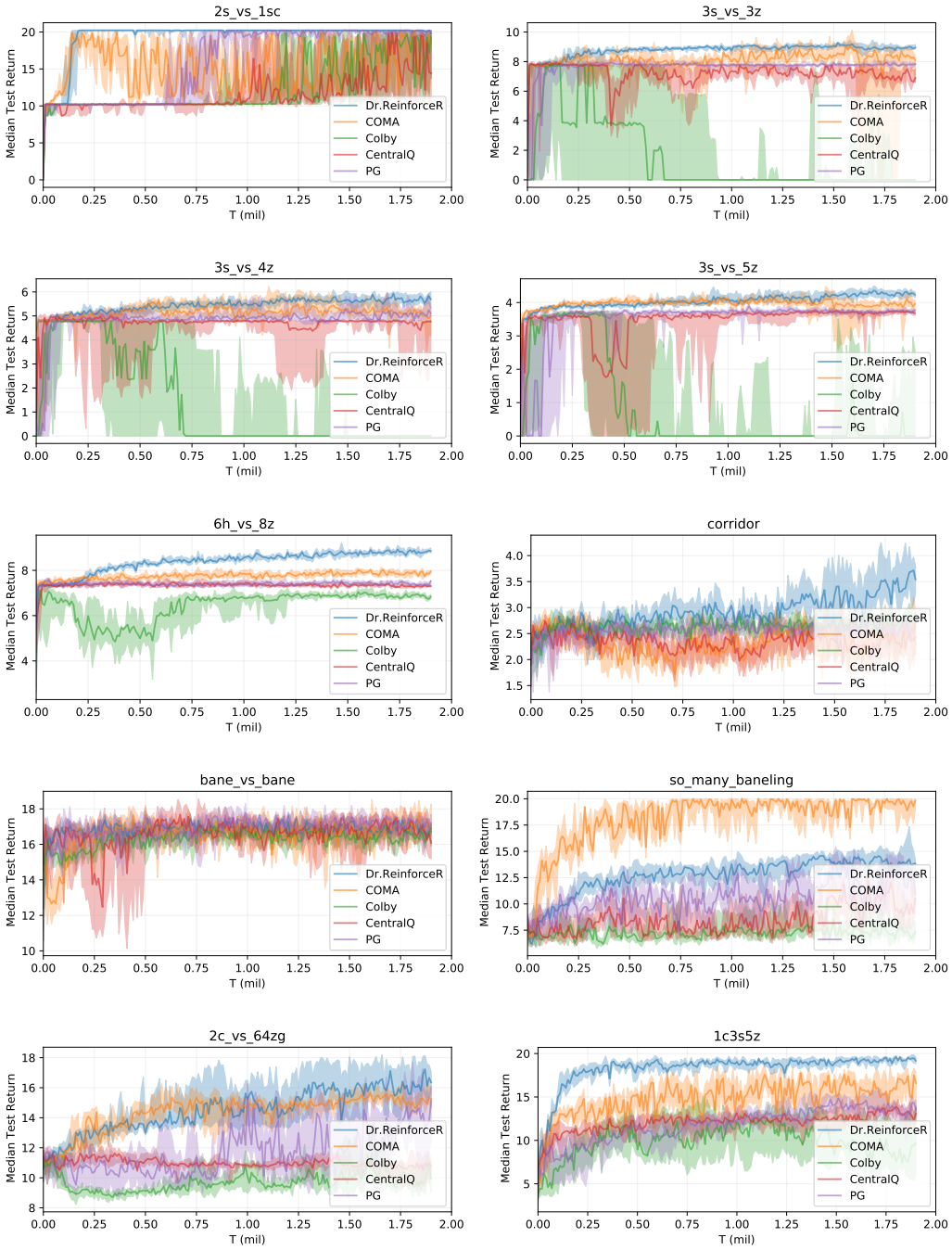


Figure 5.2: Training curves on the entire set of SMAC scenarios, showing the median return and $25-75\%$ percentiles across seeds.

is able to do it in turn). Some of these scenarios present an heterogeneous team (comprising different types of units), require to fight against an asymmetric team of opponents or to learn specific and coordinated strategies in order to succeed.

As with the current game back-end [155] it has not been possible to obtain all the reward values for the possible agents' actions, Dr.Reinforce has not been investigated here, and thus only Dr.ReinforceR is investigated. Figure 5.2 shows median return and $25-75\%$ percentiles across 10 independent runs on the whole set of available scenarios, with the difficulty level of the opponent team set to Very Hard.

In this setting, Dr.ReinforceR is almost never underperforming with respect to all the other baselines, with significant improvements over COMA on heterogeneous scenarios like `3s5z`, `1c3s5z` or `MMM`. This shows how learning the Q-function may be difficult in complex settings with large state and action spaces and multiple agents, while the centralized reward network can be learned more easily and in turn produces better policies. Also, it is worth mentioning that the severe partial observability of this setting is well addressed in practice by the use of the CTDE paradigm, with the reward network conditioned on the true state s: these results show how advantageous it is to resort to centralized training of the reward representation over a local approximation as in the algorithm from [25]. In particular, the good performance on the `25m` scenario, involving a large number of agents, shows again the better scalability of the proposed centralized reward network with respect to a centralized Q-function critic, where the effects of bootstrapping and the moving target problem become even more severe.

A noticeable exception is represented by the `so_many_baneling` scenario, where COMA is achieving a good result while neither Dr.ReinforceR nor any of the other baselines are capable of outperforming it. An hypothesis for this is that the difference return ΔG_t^i is driving each agent into performing the more rewarding actions at each step (for example, hit an opponent if possible), but in the long run this strategy is not a winning one on this particular scenario, with the agents never experiencing the high reward for winning and thus never being able to change their learned behaviours. Reasoning on the more complex Q-function here could be helpful to drive the policies towards a winning situation at the cost of performing actions that seem suboptimal at the current step.

Another commonly used metric to evaluate agents in SMAC is the win rate, i.e. how many battles the learned team of agents is capable of winning against the opposing team. This metric gives a proper understanding of the behaviour of the agents, and in general an increase in the win rate also entails an increase in the overall return (as the agents are receiving a very high reward for winning a battle). Policy gradient methods are known to perform poorly on the most challenging SMAC scenarios, where value-based methods are generally preferred [115, 158] (although some more recent and complex multi-agent policy gradient algorithms are also achieving significant performances [180, 126]). Dr.ReinforceR, although in general

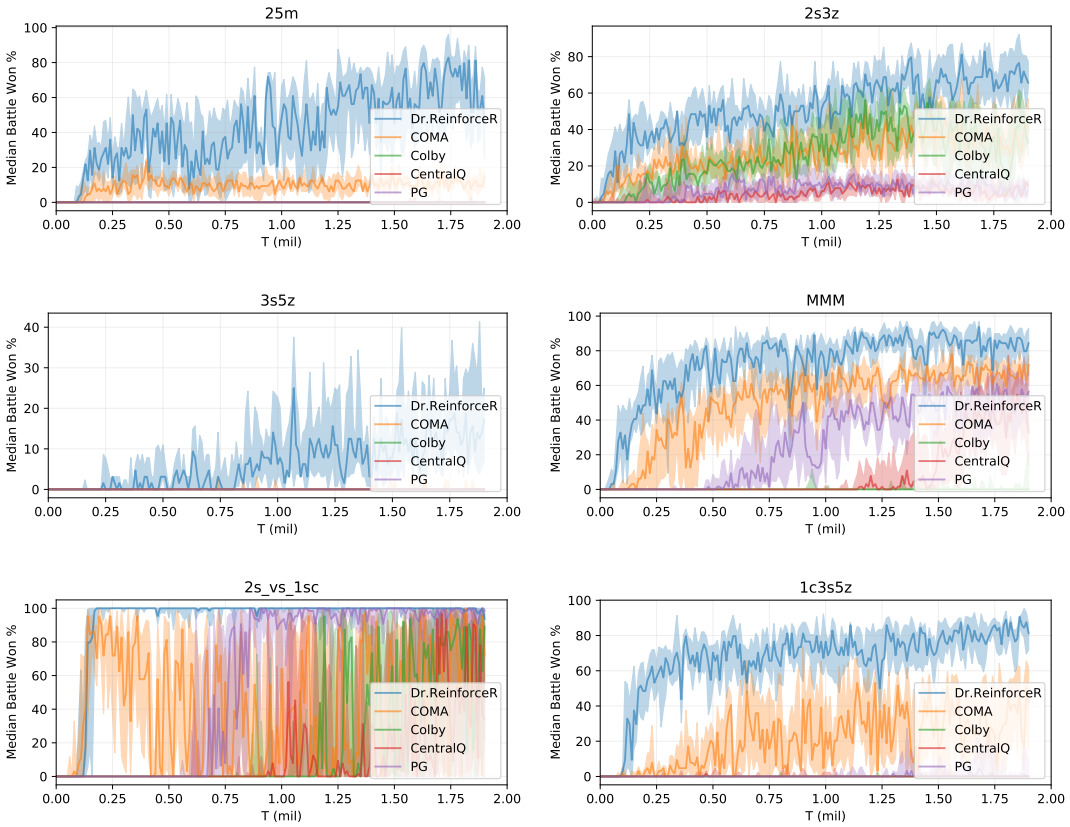


Figure 5.3: Training curves on a set of SMAC scenarios, showing the median victory rate and $25-75\%$ percentiles across seeds.

improving the win rate over COMA on some of the simpler scenarios, still struggles to achieve significant victories on most of the hardest ones, like `6h_vs_8z` or `MMM2`, even though the median return for these is significantly higher than that of the other baselines. Complete results for this metric are reported in Appendix B.3, while only some of the settings in which a significant improvement is achieved are shown in Figure 5.3. Similarly to above, the plots show the median win rate and $25-75\%$ percentiles across 10 independent runs.

Dr.ReinforceR is greatly improving performances over COMA on some scenarios, including heterogeneous ones like `1c3s5z` or `MMM` and the asymmetric `2s_vs_1sc`, where it is achieving a perfect winning strategy. Interesting is the result on the `25m` scenario, one of the larger instances provided with SMAC: this once more demonstrates the improved scalability of the reward representation, and how this is positively affecting agents' learned behaviours.

5.4 DISCUSSION

Despite the good empirical results obtained by Dr.ReinforceR in the experiments detailed above, Lemma 5.2 clearly shows that the combination of difference rewards

and policy gradients in a partially observable setting has in general no theoretical guarantees of convergence, as the baseline that is subtracted from the distributed policy gradient is not unbiased. This means that experimental performances could be unstable or arbitrarily bad.

Here possible alternatives to the currently investigated formulation that are capable of restoring the theoretical convergence guarantees are eventually identified. This could be ensured by replacing the current baseline $B^i(s_{t:T},a^{-i}_{t:T},h^i_{t:T})$ in Equation 5.7 with a new $\tilde{B}^i(s_{t:T},a^{-i}_{t:T},h^i_t)$ that does not depend on the currently selected action a^i_t via the local histories $h^i_{t+1:T}$. A couple of possible solutions are proposed, that are not however investigated in the current work:

1. Replace the current agent policy $\pi_{\theta^i}(a^i_t|h^i_t)$ with a fixed policy $\mu(a^i_t)$ (a type of difference rewards also proposed in [169]):

$$\tilde{B}^i(s_{t:T},a^{-i}_{t:T}) = \sum_{l=0}^{T-t-1} \gamma^l \sum_{c^i \in A^i} \mu(c^i) \cdot R(s_{t+l},\langle c^i,a^{-i}_{t+l}\rangle).$$

   This idea however would require to fix beforehand a policy $\mu(a^i_t)$ to use, a choice similar to that of the default action a^i_d in Equation 2.24 [169, 114].

2. Use the current agent policy $\pi_{\theta^i}(a^i_t|h^i_t)$, but do not condition on the local histories for the episode time steps $h^i_{t+1:T}$, but only on the current local history h^i_t:

$$\tilde{B}^i(s_{t:T},a^{-i}_{t:T},h^i_t) = \sum_{l=0}^{T-t-1} \gamma^l \sum_{c^i \in A^i} \pi_{\theta^i}(c^i|h^i_t) \cdot R(s_{t+l},\langle c^i,a^{-i}_{t+l}\rangle).$$

3. Use a potential-based reward shaping mechanism to replace the baseline. These are known to retain policy invariance in single-agent reinforcement learning, both under full observability [93] as well as partial one [36], while in multi-agent systems converge to the same set of Nash Equilibria of the policies learned with the shared reward alone [30, 31], while improve learning performance. In general, a potential-based reward shaping mechanism provides the agents with a shaped reward $\hat{r}$:

$$\hat{r} \triangleq r_t + \underbrace{F(s_t,s_{t+1})}_{\tilde{B}^i}, \tag{5.17}$$

   where $F(s_t,s_{t+1}) = \gamma\phi(s_{t+1}) - \phi(s_t)$, and $\phi(s)$ is a suitable function that provides additional information on the state s, so that $F(s_t,s_{t+1})$ is unbiased in expectation with respect to the policy gradients, and thus keep the convergence guarantees.

   A particular form of potential-based reward shaping, that combines its benefits with those of difference rewards, is Counterfactual as Potential [32], in which the potential-based reward shaping function is:

$$\phi(s) = R(s^{-i}), \tag{5.18}$$

and $R(s^{-i})$ is a reward term that marginalizes out the presence of agent i. It is to note that, while in general such a term needs to be provided by the environment itself via the use of a simulator (as with difference rewards), with a learned reward network that issue could be overcome.

Another crucial aspect of Dr.ReinforceR is that it resorts to the CTDE framework [70, 108] to learn its centralized reward network. Although CTDE is a widely used and accepted methodology [108, 40, 79], it indeed restricts the training procedure to be carried out offline and in a separate step from the agents execution. There are settings however in which being able to retain decentralized execution while being able to learn during real interactions with the environment may be required. In such cases, it may be appropriate to replace the centralized reward network R_ψ with a set of individual reward networks $R_{\psi^i}(s,a^i)$ (or $R_{\psi^i}(h_t^i,a^i)$ when learning in a Dec-POMDP), one for each agent i, to approximate the difference rewards computation. These local networks are learning the expected value of the reward for each agent when performing a certain action in a given situation, independently of what the others are doing:

$$R_{\psi^i}(s,a^i) \approx \mathbb{E}_{\pi_{\theta^{-i}}}\left[R_\psi(s,\langle a^i,a^{-i}\rangle)\right]. \tag{5.19}$$

This additional approximation is suitable to break the dependence from the CTDE paradigm, although it may introduce approximation error in the local reward terms via the expectation over the other agents policies (while the centralized reward network R_ψ is in principle capable of perfectly approximate the reward function $R(s,a)$ and thus provide the policy gradients with perfect difference rewards values).

6 CONCLUSIONS

In this final chapter the research questions introduced in Section 1.4 are finally answered, relating them with the various aspects of the original work presented here. Following, a summary of the original contributions of this thesis is provided, and finally some possible future directions and current limitations of the presented work are outlined.

6.1 ANSWERING THE RESEARCH QUESTIONS

RQ1: Is the general assumption on "higher-order" factorizations being helpful in learning improved approximations in the multi-agent setting justified by practical results?

Chapter 3

Although it is widely assumed in literature that factored approaches can improve the learning accuracy and efficiency in cooperative multi-agent systems, no systematic study with a broad investigation of their applicability and performance has ever been carried out before this work. In Chapter 3 such a lack of investigation has been addressed in the deep reinforcement learning setting, by comparing different factorizations, represented by neural networks, and learning approaches on a diverse set of highly-coordinated one-shot problems. These have been chosen as the minimum setting presenting most of the issues related to the presence of multiple learning agents, as an exponential number of joint actions. Different degrees of interconnection among the agents in the coordination graph has been investigated, to assess how the structure of such a graph is affecting the final performance, as well as trying to identify common structures that perform well across different problems. Also, two learning rules, one in which the agents jointly optimize their objective, the other in which each component learns independently from the others, have been used. The obtained results support the widely spread intuition that factorizations are really of an help when learning in multi-agent systems: factorizations seem to be in general capable of learning a good approximation of the action-value function in many different coordination problems, even in situations where no such a decomposable structure is suggested by the problem itself, and thus in principle each agent needs to explicitly coordinate with all of the others in the team to learn a correct behaviour. Moreover, when the problem can indeed be decomposed into smaller components, the use of factorizations is of a great benefit even when the coordination graph structure does not resemble the

true underlying decomposition of such a problem (as done in the proposed experiments, where the factored games were indeed treated as black-boxes with unknown properties). Action-value functions approximated via a factored representation based on neural networks are generally close to their original counterparts both in terms of mean prediction error and ranking of the actions.

Both independent learners and centralized controllers can be seen as the two opposite extreme cases of a factorization: the first one resembles the case where each agent is not explicitly coordinating with any of the others, while the latter is the case where all the agents can explicitly coordinate and learn jointly. Both approaches however are known to have issues: while independent learners are easy to learn because each agent only has to rely on local information, the environment becomes non-stationary from each agent's own perspective, and thus achieving complex coordination becomes difficult. On the other hand, a centralized controller allows all the agent to learn the value of their joint actions, but it scales exponentially in the number of agents, and thus quickly becomes impractical to be applied to growingly large multi-agent systems. Factorizations offer a suitable middle ground between these two approaches: on one hand these allow for some degree of explicit coordination among the subset of agents comprised into each factor, while on the other hand these approximate the joint action-value function by mean of a set of such factors, each one being considerably smaller than a centralized controller itself. This results in improved approximations over both the extremes: the coordination inside each component allows a factored approach to learn the value of coordination more easily than independent learners, while the reduced size of each such components speeds up their learning and eases the exponential explosion of the joint action set.

As many problems do not present a decomposable structure, it is often useful to resort to an approximate factorization to try and learn an approximated action-value function. This approximation however can be done in many different ways, and the choice of a particular coordination graph used to connect the agents is going to affect also the overall learned approximation. In Chapter 3 an array of different possibilities has been proposed (although many other can be imagined and tested), with the aim of general applicability as one of the main goals. The results highlight some fundamental aspects that system designers can keep in mind when resorting to this kind of techniques:

1. The number of factors seem to be a choice of the primary importance. As expected, introducing more components in which the agents can explicitly coordinate with a subset of the others is helpful in reducing the approximation error and identify the optimal joint actions. Although the use of a complete coordination graph has proved effective on almost every problem, there are situations in which such a solution may be redundant: it

may suffice to use a smaller coordination graph, where fewer components are used and learned, to still obtain a useful action-value function, while at the same time making the learning faster. The number of factors can thus be seen as a trade-off value between accuracy and speed, and a careful tuning of such a parameter may be crucial, and still tied to the problem at hand to some extent.

2. On the other hand, the experiments show that the largeness of these factors is less crucial a choice. Although it may be useful to have more agents into each component, thus allowing them to coordinate explicitly, good results can be obtained in many situations even with just two agents into each factor. This intuition is in accordance to the one above: if agents are comprised into multiple factors, such factors can be of a small size, as the agents are still going to adjust the values for their actions considering those of the others.
3. The choice of the learning approach places somewhere in-between these two aspects: a joint optimization of the objective is beneficial in many situations, and in general reduces the mean square error of the learned approximation. This setting is for example a common choice under the CTDE paradigm [139, 115, 135, 158], where agents are trained centrally in simulation before being deployed. But, especially if the main interest is on the ranking of the actions rather than a perfect representation, it may simply suffice to optimize each component independently from the others. This situation is not unrealistic: for example, if the overall representation is used in couple with an (ϵ-)greedy policy to drive the behaviours of the agents, recognizing the optimal actions is enough to achieve good policies.

However, there are some problem-specific aspects that none of the investigated representations seems able to deal with, such as an unbalance in the reward function (as in the Platonia Dilemma). For these, factored approaches do not seem to offer the same level of improvement achieved on other problems, and could indeed lead to wrong representations.

Another crucial aspect that has been investigated is how these approaches scale in the number of agents: in many situations the system may comprise a significant number of agents, and this in turn may exacerbate the problem of both independent learners (more ignored agents that render the environment even more noisy) and centralized controller (because of the exponential explosion of the number of joint actions). Factorizations however are less prone to suffer from this: the proposed coordination graph structures seem to scale gracefully when more agents are present in the system, retaining their good accuracy and speed-up in learning. This is an extremely desirable property, and partly justifies the recent interest in value-decomposition methods from the research

community. Because of the small number of agents comprised into each factor with respect to the overall team size, also sample efficiency scales better than that of a centralized controller: when simulators are not available and training samples are difficult to gather, this is a crucial benefit.

RQ2: Can difference rewards applied to the reward function (possibly by learning a representation of it) provide better learning signals to distributed agents compared to the same idea applied on a centralized action-value critic?

Chapter 4

In cooperative multi-agent settings, the agents usually share the same reward value given by the environment after each interaction. Although this signal is useful to represent problems with a common goal, it does not tell anything to the agents about their own contribution to the team performance: indeed, with an high reward signal, an agent may be fooled into thinking that its policy is good and it is performing well, although there may be more suitable behaviours leading to even higher rewards. Difference rewards are a family of algorithms that allow each agent to learn from a shaped reward signal that reflects its own actual contribution to the shared reward. This way, an agent can change its own behaviour towards regions that improves its own shaped reward signal, and in turn the overall reward of the whole team. In Chapter 4 difference rewards have been combined with distributed policy gradients, a multi-agent algorithm that allows us to learn decentralized policies via policy gradients. The proposed algorithm, Dr.Reinforce, that has full access to the reward function and thus may retrieve all the required reward values needed to compute the difference rewards, has proved effective on a variety of multi-agent problems. In comparison to standard distributed policy gradients, the difference rewards mechanism proves extremely effective in driving the learning of the agents, and allows them to achieve higher return policies with just little computational burden. Also, such a mechanism seem capable of dealing with a sparse reward function (as in the predator-prey problem), and thus could possibly be adapt to tackle delayed rewards in a realistic control setting. The presented theoretical results show how the use of difference rewards corresponds to subtracting an unbiased baseline from these policy gradients, that in fully observable settings has been proved to converge. Such a baseline thus reduces the variance of the gradients without altering the expected value of the gradients themselves. Counterfactual Multi-Agent Policy Gradients (COMA) [40] is a state-of-the-art multi-agent algorithm that applies the idea of difference rewards to the Q-function by providing each agent with a counterfactual advantage function that marginalizes out the agent own contribution. This idea seems to be widely applicable and should ease the multi-agent credit assignment problem that arises in cooperative settings, but empirical performance of COMA has been

shown to be quite weak in a number of problems. In contrast, the proposed Dr.Reinforce seems to be applicable to a variety of settings with good results when the reward function is known.

The second proposed algorithm, Dr.ReinforceR, overcomes the dependence of Dr.Reinforce on the capability of fully accessing the environment reward function in order to compute the difference rewards values. To do so, an additional centralized reward network is learned following the CTDE paradigm, together with the agents' policies, to represent such a reward function, and is then used to estimate the required values. Empirical performances show that Dr.ReinforceR is performing well, and it is generally matching the performances of Dr.Reinforce (that represents an upper bound for the case in which the reward function has perfectly converged to the real one). The reasons for these performances have been investigated by comparing the learned reward approximation with COMA action-value function critic. Both functions depend on the joint action of the team, and thus suffer from the exponential explosion of such actions with the number of agents. However, the centralized reward network of Dr.ReinforceR achieves better representations that COMA critic, resulting in a lower normalized prediction error and lower variance. This is due to a number of reasons: first, the reward function can be learned via supervised learning, and does not involve bootstrapping as with the action-value function, so that wrong or inaccurate predictions cannot influence the value of other predictions via updates. Moreover, no moving target problem arises: the reward function is fixed and can be correctly learned since the very beginning, while the action-value function that needs to be learned changes depending on the agents policies and the learned approximation itself. Although target networks are used to ease this problem, this still remains a significant challenge in multi-agent settings.

RQ3: Can the same idea be extended to partially observable settings and still improve performances?

Chapter 4 & 5

Policy gradient methods can be straightforwardly extended to address partially observable settings [166], and thus are a popular choice for these problems. Both Dr.Reinforce and Dr.ReinforceR can be similarly extended without too many algorithmic changes, although this way the theoretical guarantees that were derived for the fully observable case are not valid anymore. Nonetheless, on the challenging SMAC domain, an highly-complex partially observable setting where no knowledge of the reward function is available, Dr.ReinforceR outperforms COMA on almost all the available maps in terms of achieved return (and in some cases, also in terms of win rate). This shows how the idea of applying a difference rewards mechanism to tackle multi-agent credit

assignment is indeed still effective also under partial observability, and how using it with the reward function as proposed seems to solve a number of weaknesses that deem COMA.

6.2 SUMMARY OF CONTRIBUTIONS

This thesis work investigated the effects that learning different representations may have on performance in cooperative multi-agent systems. The focus has been on two fundamental aspects: the team representation and the problem of multi-agent credit assignment. About the former, the research community has long gained interest in factored representations [50, 53, 52], a middle ground between the two more classical approaches of independent learners and centralized controllers. While these have been assumed to result in improved approximations of the action-value function (with a multitude of recent deep MARL algorithms relying on this idea [139, 115, 135, 158, 175, 111, 10, 76, 137, 160, 181, 89]), no proper investigation of the real merits and limitations of factored methods has been carried out so far. This work somehow bridges this gap, by comparing different factorizations, with a special focus on "higher-order" decompositions, on a wide set of one-shot multi-agent problems requiring coordination, investigating both the approximation accuracy and the action ranking produced by different instantiations of these methods, to provide a clearer understanding of which situations can be tackled with factored approaches and which cannot, as well as trying to provide common guidelines for system designers for their implementation choices. As for multi-agent credit assignment, difference rewards methods [169, 168, 178, 114, 26, 25] have been proposed to tackle it. Although these have been also applied to deep MARL algorithms, with COMA [40] as the main example, the focus has been of applying the differencing mechanism to the Q-function (learned with a centralized critic), with empirical performances that usually do not match the expectations. This is possibly due to the inherent difficulties of learning the Q-function in a multi-agent setting, such as the dependence on the exponentially many joint actions or the use of bootstrapping. In this work two novel algorithms have been proposed to tackle multi-agent credit assignment while at the same time avoid the pitfalls of learning an action-value function critic. The core idea is to combine decentralized policy gradients with the use of the true reward function to compute the differencing (or to learn it through a centralized reward network in case that is not known beforehand), and the proposed results and analysis show that this is a viable approach to learn an accurate representation and apply difference rewards.

6.3 LIMITATIONS AND FUTURE WORK

While the analysis carried out in Chapter 3 has shed some light on a proper understanding of factorizations applicability and general performance, still a lot could

be done. For example, only one-shot settings has been considered, to try and limit possible sources of interference with the main difficulties that arise in the cooperative multi-agent setting (namely, the exponential number of joint actions leading to a very large function to approximate and the tight coordination requirements). However, many problems are indeed of a sequential nature, and thus agents need to coordinate differently on each state. Although our general results are still valid in these settings (every sequential setting can be reduced to a one-shot problem where the agents have to select their optimal policy), a deeper analysis to understand the impact of the environment state would be beneficial for the research community. Algorithms like DCG [10] already showed an empirical applicability of "higher-order" factorizations in practice, but what kind of approximations these are really learning and how accurate these are is still unclear. Moreover, the proposed results provided some general guidelines and trends that can be followed when implementing a factored approach, but also showed how there is no general "suitable-for-all-needs" configuration that can always be used and provides good results. This points out that the choice of a suitable factorization is a delicate problem, and although these are shown to help in general, such a choice requires careful thinking and depends on the problem at hand. Finally, the use of centralized training could be seen as somehow limiting: while the proposed factored approaches can be useful under the CTDE paradigm, for example to represent a centralized action-value critic in a multi-agent actor critic method, these would require also centralized execution if used for decision making, a possibility that is usually not available and may result problematic in any case (in cases of a failure of the centralized component or when more agents need to enter the environment). A possible future direction to extend such a work could be a broader investigation of other coordination graph structures, or even investigating adaptive coordination graphs that can be learned together with the action-values. Coordination graph learning algorithm (like that proposed in [76], that uses graph networks) can be used to remove the burden of the graph architectural choice, possibly expanding the applicability of "higher-order" factorizations also to unseen situations and to more general problems.

Also the novel algorithms proposed in Chapter 4 and 5 have some possible limitations that could be tackled: for example, although the empirical performances on partially observable settings (as those on the StarCraft II challenge) are satisfactory, the issue of the proposed baseline not being unbiased in such settings may potentially lead to completely wrong behaviours on other problems. As it is a common assumption to model a cooperative multi-agent problem as a partially observable environment, a potential solution to this issue could result in a wider applicability of such algorithms. Possible alternative baselines has been proposed in Section 5.4, and in future work a comparison of these may lead to improved performances and theoretical guarantees. Also, the proposed centralized reward network used by Dr.ReinforceR, although it has been shown to be easier to learn than a centralized critic, has still the same

dependence of the exponentially large joint action set, and thus may be problematic to learn as well in some settings. Possible solutions to this would be to resort to either factorization techniques (as these have been shown to help in Chapter 3) or to learn an individual reward network for each agent, a solution that also break the dependence of the proposed algorithm on the CTDE paradigm. Single-agent decompositions like those used in VDN [139] or QMIX [115] could result in better approximations of the reward function, and in turn to better difference rewards signals (as the computation of difference rewards values strongly depends on the accuracy of the approximated values, even those of unseen state-action pairs). Finally, it is to note that the two algorithms could also benefit from a series of improvements that have been proposed in literature and are orthogonal to the issue addressed here: as an example, multi-agent policy gradients based on the proximal policy optimization (PPO) [127] algorithm, with its clipped objective for the policy gradients that is known to limit policy distance between consecutive updates and improve performances, are gaining increasing attention in the cooperative multi-agent field [126, 180, 170]. It is not difficult to apply such techniques to the two algorithms proposed here, that could possibly lead to even better results.

A ADDITIONAL POLICY GRADIENTS TRAINING DETAILS

A.1 HYPERPARAMETERS AND TRAINING PROCEDURE

For the implementation, the `pymarl` [123] framework[1] has been extended, as already providing many useful tools and the official implementation of COMA to compare against. The policy networks are either feed-forward networks for the two gridworld problems or GRU [22] to deal with partial observability on SMAC, and both use parameter sharing across agents [54] to reduce training time, while the critics and reward networks always use feed-forward networks instead.

On each of the three problems independently, the optimal values for the policy learning rate α_θ and the critic or reward network one $\alpha_{\omega/\psi}$ [42] have been found for each method through a gridsearch over a common set of standard values. The setting with $N = 3$ agents for the two gridworld environments and the map `2s3z` on SMAC has been used, and the values obtained this way have been subsequently used for the other instances of the same domain respectively. Table A.1 reports the value of the used learning rates α_θ and $\alpha_{\omega/\psi}$ for each compared method on each problem.

	Multi-Rover		Predator-Prey		SMAC	
Method	α_θ	$\alpha_{\omega/\psi}$	α_θ	$\alpha_{\omega/\psi}$	α_θ	$\alpha_{\omega/\psi}$
Dr.Reinforce	$25 \cdot 10^{-4}$	N.A.	$25 \cdot 10^{-4}$	N.A.	N.A.	N.A.
Dr.ReinforceR	$25 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$25 \cdot 10^{-4}$
COMA	$1 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
[25]	$5 \cdot 10^{-3}$	$25 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$1 \cdot 10^{-2}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
CentralQ	$5 \cdot 10^{-4}$	$25 \cdot 10^{-4}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-3}$	$5 \cdot 10^{-4}$	$5 \cdot 10^{-4}$
PG	$5 \cdot 10^{-4}$	N.A.	$5 \cdot 10^{-4}$	N.A.	$5 \cdot 10^{-4}$	N.A.

Table A.1: Value of the learning rates for each method.

COMA [40] and CentralQ critics have been trained using the TD(λ) [140, 141] variant presented in [40]. For these, the optimal value for the parameter λ with the learning rates already found by the gridsearches has also been identified following the same procedure detailed above, resulting in the values in Table A.2:

Method	Multi-Rover	Predator-Prey	SMAC
COMA	0.4	0.8	0.8
CentralQ	0.2	0.8	0.8

Table A.2: Value of λ for each method.

1 `pymarl` is available at `https://github.com/oxwhirl/pymarl`

All the methods have been trained for the same amount of steps and all their other hyperparameters are set to the corresponding default values provided by the `pymarl` framework, without being optimized: the reward network R_ψ and the critic network Q_ω for CentralQ and COMA all have the same structure, that is a two-layer feed-forward neural network with 128 hidden units using the ReLU activation function [90] before the final linear layer, as the size of the functions these have to represent is analogous. Every experiment has been repeated 10 times with different random seeds to assess variance across multiple runs, and in each episode the initial configuration has been randomly reset to avoid the policies to overfit.

B ADDITIONAL RESULTS AND PLOTS

B.1 FACTORIZATIONS COMPLETE RESULTS

Table B.1 presents the accuracy of all the investigated representations using various measures, both in terms of action ranking and reconstruction error, as well as evaluating the action selection that these representations result in:

- To evaluate the reconstruction error, the mean square error is computed over all the joint actions (1st column);
- Also, the same measure but restricted only to those actions that are optimal in the original action-value function in analysed (2nd column);
- It is assessed how many optimal actions are correctly identified by the reconstructions (3rd column);
- The value loss (regret) obtained by following the represented value functions when doing decision making (4th column);
- It is also provided a different version of this regret, that is dubbed *Boltzmann value loss* which expresses the value loss obtained by the expected reward accrued by defining a softmax distribution over all the joint actions (5th column). This gives an indication of value loss among all good actions; and
- Finally, the number of correctly ranked actions (accounting for ties where needed) is computed and the corresponding Kendall τ-b coefficient [66] between the computed ranking and the original one (6th and 7th columns respectively).

A low value on the first two columns indicates that a representation is close to the original action value function: especially when the value of on second column is low in combination with the value of the third one, the learned representation tends to correctly reconstruct and identify the optimal joint actions. The fourth and fifth columns tell how reliably these representations can be used to perform decision making (i.e. when applying a greedy policy with respect to the joint action-value function during the evaluation phase) under two different kinds of policy. In fact, if the regret is low, even if the representation is not accurate, the model can still be used to pick actions for the team resulting in high rewards. Finally, the last two columns point out if the reconstructed values of the joint actions match the same hierarchy that these have in the original action-value function: even if the representation is not accurate in terms of mean squared error, a correct ranking of the actions points out that it has the same structure of the original one, and thus the two are similar except

for the magnitude of the values themselves. All of these measures are interlinked and express how similar a learned representation is to the original, but these all highlight different aspects that can help in analysing the benefits and drawbacks of these methods.

For every measure, mean value and standard error across 10 runs are reported. For every game, the best (green) and worst (red) performances of the investigated methods on some of the proposed measures are highlighted.

Model	Mean square error	MSE on optimal actions	Optimal actions found	Value loss	Boltzmann value loss	Correctly ranked	Kendall τ
Dispersion Game $n = 6$ (64 joint actions, 20 optimal)							
Joint	0.00 ± 0.0	0.00 ± 0.0	20 ± 0	0.00 ± 0.0	0.00 ± 0.0	64 ± 0	1.00 ± 0.0
F1	0.62 ± 0.0	0.88 ± 0.0	5 ± 2	1.70 ± 1.0	0.49 ± 0.0	21 ± 2	0.00 ± 0.0
F2R	0.52 ± 0.0	0.82 ± 0.0	8 ± 0	0.00 ± 0.0	0.38 ± 0.0	24 ± 1	0.27 ± 0.0
F3R	0.41 ± 0.0	0.72 ± 0.0	10 ± 1	0.60 ± 0.5	0.31 ± 0.0	31 ± 3	0.39 ± 0.0
F2C	0.09 ± 0.0	0.14 ± 0.0	20 ± 0	0.00 ± 0.0	0.16 ± 0.0	64 ± 0	1.00 ± 0.0
F3C	0.09 ± 0.0	0.14 ± 0.0	20 ± 0	0.00 ± 0.0	0.16 ± 0.0	64 ± 0	1.00 ± 0.0
F2O	0.41 ± 0.0	0.64 ± 0.0	10 ± 1	0.60 ± 0.5	0.32 ± 0.0	36 ± 2	0.44 ± 0.0
F3O	0.19 ± 0.0	0.30 ± 0.0	13 ± 1	0.20 ± 0.4	0.20 ± 0.0	47 ± 3	0.70 ± 0.0
M1	0.62 ± 0.0	0.88 ± 0.0	6 ± 1	1.30 ± 0.8	0.49 ± 0.0	24 ± 2	0.02 ± 0.0
M2R	0.56 ± 0.0	0.82 ± 0.0	8 ± 0	0.00 ± 0.0	0.46 ± 0.0	24 ± 1	0.27 ± 0.0
M3R	0.46 ± 0.0	0.73 ± 0.0	10 ± 1	0.40 ± 0.5	0.39 ± 0.0	31 ± 2	0.39 ± 0.0
M2C	0.55 ± 0.0	0.81 ± 0.0	20 ± 0	0.00 ± 0.0	0.46 ± 0.0	64 ± 0	1.00 ± 0.0
M3C	0.43 ± 0.0	0.68 ± 0.0	20 ± 0	0.00 ± 0.0	0.40 ± 0.0	64 ± 0	1.00 ± 0.0
M2O	0.56 ± 0.0	0.81 ± 0.0	10 ± 2	0.50 ± 0.5	0.46 ± 0.0	36 ± 4	0.46 ± 0.0
M3O	0.44 ± 0.0	0.69 ± 0.0	11 ± 1	0.30 ± 0.5	0.40 ± 0.0	40 ± 3	0.58 ± 0.1
Dispersion Game (sparse) $n = 6$ (64 joint actions, 20 optimal)							
Joint	0.00 ± 0.0	0.00 ± 0.0	20 ± 0	0.00 ± 0.0	0.00 ± 0.0	64 ± 0	1.00 ± 0.0
F1	1.93 ± 0.0	4.25 ± 0.0	7 ± 1	1.50 ± 1.5	1.77 ± 0.0	37 ± 1	0.02 ± 0.0
F2R	1.83 ± 0.0	3.95 ± 0.0	8 ± 0	0.00 ± 0.0	1.64 ± 0.0	40 ± 0	0.13 ± 0.0
F3R	1.72 ± 0.0	3.60 ± 0.0	11 ± 1	1.80 ± 1.5	1.55 ± 0.0	45 ± 1	0.31 ± 0.0
F2C	1.41 ± 0.0	2.25 ± 0.0	20 ± 0	0.00 ± 0.0	1.32 ± 0.0	64 ± 0	1.00 ± 0.0
F3C	1.41 ± 0.0	2.26 ± 0.0	20 ± 0	0.00 ± 0.0	1.32 ± 0.0	64 ± 0	1.00 ± 0.0
F2O	1.72 ± 0.0	3.53 ± 0.0	10 ± 1	1.50 ± 1.5	1.56 ± 0.0	45 ± 3	0.30 ± 0.1
F3O	1.52 ± 0.0	2.71 ± 0.1	12 ± 1	0.60 ± 1.2	1.41 ± 0.0	49 ± 2	0.45 ± 0.1
M1	1.93 ± 0.0	4.27 ± 0.0	6 ± 1	2.10 ± 1.4	1.77 ± 0.0	37 ± 2	0.01 ± 0.1
M2R	1.88 ± 0.0	4.13 ± 0.0	8 ± 0	0.00 ± 0.0	1.73 ± 0.0	40 ± 0	0.13 ± 0.0
M3R	1.77 ± 0.0	3.88 ± 0.0	10 ± 1	1.50 ± 1.5	1.66 ± 0.0	44 ± 3	0.27 ± 0.1
M2C	1.87 ± 0.0	4.11 ± 0.0	20 ± 0	0.00 ± 0.0	1.73 ± 0.0	64 ± 0	1.00 ± 0.0
M3C	1.74 ± 0.0	3.83 ± 0.0	20 ± 0	0.00 ± 0.0	1.66 ± 0.0	64 ± 0	1.00 ± 0.0
M2O	1.87 ± 0.0	4.13 ± 0.0	10 ± 1	1.20 ± 1.5	1.73 ± 0.0	44 ± 2	0.26 ± 0.1
M3O	1.75 ± 0.0	3.84 ± 0.0	12 ± 1	0.60 ± 1.2	1.67 ± 0.0	47 ± 2	0.38 ± 0.1
Platonia Dilemma $n = 6$ (64 joint actions, 6 optimal)							
Joint	0.00 ± 0.0	0.03 ± 0.1	6 ± 0	0.00 ± 0.0	0.01 ± 0.0	64 ± 0	1.00 ± 0.0
F1	2.22 ± 0.0	15.55 ± 0.1	5 ± 0	6.00 ± 0.0	4.19 ± 0.0	62 ± 0	0.82 ± 0.0
F2R	2.11 ± 0.0	14.17 ± 0.1	5 ± 0	6.00 ± 0.0	4.09 ± 0.0	62 ± 0	0.82 ± 0.0
F3R	2.00 ± 0.0	12.90 ± 0.1	5 ± 0	6.00 ± 0.0	4.04 ± 0.0	62 ± 0	0.82 ± 0.0
F2C	1.69 ± 0.0	8.92 ± 0.1	5 ± 0	6.00 ± 0.0	4.39 ± 0.0	62 ± 0	0.82 ± 0.0
F3C	1.69 ± 0.0	8.97 ± 0.1	5 ± 0	6.00 ± 0.0	4.52 ± 0.0	62 ± 0	0.82 ± 0.0
F2O	2.00 ± 0.0	12.83 ± 0.1	5 ± 0	6.00 ± 0.0	3.94 ± 0.0	62 ± 1	0.78 ± 0.1

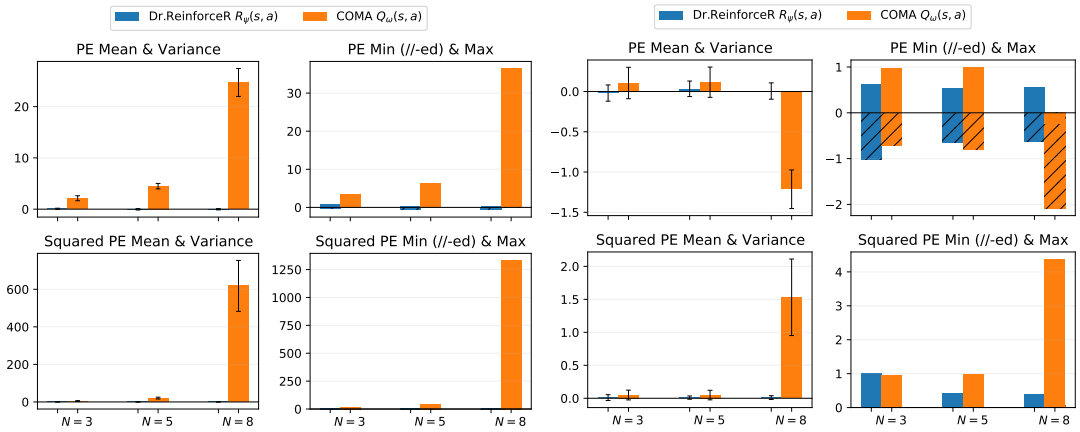
Model	Mean square error	MSE on optimal actions	Optimal actions found	Value loss	Boltzmann value loss	Correctly ranked	Kendall τ
F3O	1.78 ± 0.0	10.10 ± 0.4	5 ± 0	6.00 ± 0.0	4.15 ± 0.1	62 ± 0	0.82 ± 0.0
M1	2.80 ± 0.0	27.01 ± 0.0	5 ± 0	6.00 ± 0.0	5.15 ± 0.0	62 ± 0	0.82 ± 0.0
M2R	2.53 ± 0.0	23.93 ± 0.0	5 ± 0	6.00 ± 0.0	4.93 ± 0.0	62 ± 0	0.82 ± 0.0
M3R	2.28 ± 0.0	20.51 ± 0.1	5 ± 0	6.00 ± 0.0	4.64 ± 0.0	62 ± 0	0.82 ± 0.0
M2C	2.52 ± 0.0	23.90 ± 0.0	5 ± 0	6.00 ± 0.0	4.92 ± 0.0	62 ± 0	0.82 ± 0.0
M3C	2.25 ± 0.0	20.54 ± 0.0	5 ± 0	6.00 ± 0.0	4.62 ± 0.0	62 ± 0	0.82 ± 0.0
M2O	2.54 ± 0.0	23.93 ± 0.1	4 ± 0	6.00 ± 0.0	4.92 ± 0.0	61 ± 1	0.69 ± 0.1
M3O	2.28 ± 0.0	20.60 ± 0.1	4 ± 0	6.00 ± 0.0	4.61 ± 0.0	61 ± 1	0.71 ± 0.1
Climb Game $n = 6$ (729 joint actions, 1 optimal)							
Joint	0.17 ± 0.1	18.45 ± 4.9	0 ± 0	2.70 ± 0.9	1.52 ± 0.3	727 ± 1	1.00 ± 0.0
F1	0.58 ± 0.0	52.29 ± 0.1	0 ± 0	3.00 ± 0.0	2.16 ± 0.0	726 ± 0	0.98 ± 0.0
F2R	0.52 ± 0.0	40.95 ± 0.0	0 ± 0	3.00 ± 0.0	2.06 ± 0.0	726 ± 0	0.98 ± 0.0
F3R	0.44 ± 0.0	36.51 ± 0.2	0 ± 0	3.00 ± 0.0	1.92 ± 0.0	726 ± 0	0.98 ± 0.0
F2C	0.25 ± 0.0	7.86 ± 0.1	1 ± 0	0.00 ± 0.0	1.40 ± 0.0	729 ± 0	1.00 ± 0.0
F3C	0.17 ± 0.0	70.77 ± 0.7	0 ± 0	3.00 ± 0.0	0.96 ± 0.0	726 ± 0	0.98 ± 0.0
F2O	0.45 ± 0.0	30.83 ± 0.1	0 ± 0	3.00 ± 0.0	1.94 ± 0.0	726 ± 0	0.98 ± 0.0
F3O	0.30 ± 0.0	28.89 ± 1.9	0 ± 0	3.00 ± 0.0	1.54 ± 0.0	726 ± 0	0.98 ± 0.0
M1	0.71 ± 0.0	35.91 ± 0.0	0 ± 0	3.00 ± 0.0	2.36 ± 0.0	726 ± 0	0.98 ± 0.0
M2R	0.63 ± 0.0	35.77 ± 0.0	0 ± 0	3.00 ± 0.0	2.30 ± 0.0	726 ± 0	0.98 ± 0.0
M3R	0.53 ± 0.0	35.34 ± 0.1	0 ± 0	3.00 ± 0.0	2.20 ± 0.0	726 ± 0	0.98 ± 0.0
M2C	0.62 ± 0.0	35.77 ± 0.0	0 ± 0	3.00 ± 0.0	2.30 ± 0.0	726 ± 0	0.98 ± 0.0
M3C	0.51 ± 0.0	35.30 ± 0.1	0 ± 0	3.00 ± 0.0	2.20 ± 0.0	726 ± 0	0.98 ± 0.0
M2O	0.63 ± 0.0	35.74 ± 0.0	0 ± 0	3.00 ± 0.0	2.30 ± 0.0	726 ± 0	0.98 ± 0.0
M3O	0.52 ± 0.0	35.31 ± 0.1	0 ± 0	3.00 ± 0.0	2.20 ± 0.0	726 ± 0	0.98 ± 0.0
Penalty Game $n = 6$ (729 joint actions, 2 optimal)							
Joint	1.60 ± 0.4	18.21 ± 4.3	1 ± 0	0.90 ± 1.4	3.24 ± 0.1	727 ± 1	1.00 ± 0.0
F1	2.18 ± 0.0	63.71 ± 0.1	0 ± 0	3.00 ± 0.0	3.31 ± 0.0	722 ± 0	0.91 ± 0.0
F2R	2.00 ± 0.0	65.95 ± 0.3	0 ± 0	3.00 ± 0.0	3.33 ± 0.0	723 ± 0	0.92 ± 0.0
F3R	1.75 ± 0.0	66.27 ± 1.0	0 ± 0	3.00 ± 0.0	3.31 ± 0.0	723 ± 0	0.94 ± 0.0
F2C	1.29 ± 0.0	79.66 ± 0.0	0 ± 0	6.00 ± 0.0	3.30 ± 0.0	722 ± 0	0.92 ± 0.0
F3C	0.54 ± 0.0	82.77 ± 0.0	0 ± 0	3.00 ± 0.0	2.06 ± 0.0	724 ± 0	0.97 ± 0.0
F2O	1.82 ± 0.0	68.81 ± 0.3	0 ± 0	6.00 ± 0.0	3.32 ± 0.0	723 ± 0	0.95 ± 0.0
F3O	1.27 ± 0.0	73.48 ± 1.4	0 ± 0	6.00 ± 0.0	3.29 ± 0.0	723 ± 0	0.95 ± 0.0
M1	2.71 ± 0.0	45.27 ± 0.1	0 ± 0	3.00 ± 0.0	3.66 ± 0.0	722 ± 0	0.91 ± 0.0
M2R	2.43 ± 0.0	49.11 ± 0.2	0 ± 0	3.00 ± 0.0	3.54 ± 0.0	723 ± 0	0.93 ± 0.0
M3R	2.09 ± 0.0	52.23 ± 0.8	0 ± 0	3.00 ± 0.0	3.43 ± 0.0	723 ± 0	0.94 ± 0.0
M2C	2.41 ± 0.0	49.12 ± 0.1	0 ± 0	3.00 ± 0.0	3.54 ± 0.0	724 ± 0	0.97 ± 0.0
M3C	2.02 ± 0.0	52.45 ± 0.2	0 ± 0	3.00 ± 0.0	3.43 ± 0.0	724 ± 0	0.97 ± 0.0
M2O	2.43 ± 0.0	49.11 ± 0.1	0 ± 0	3.00 ± 0.0	3.54 ± 0.0	724 ± 0	0.97 ± 0.0
M3O	2.06 ± 0.0	52.56 ± 0.5	0 ± 0	3.00 ± 0.0	3.43 ± 0.0	723 ± 2	0.96 ± 0.0
Generalized FF $n = 6$ (8192 joint actions, 779 optimal)							
Joint	1.29 ± 2.5	4.96 ± 7.2	656 ± 123	61.60 ± 68.2	46.42 ± 48.4	$6{,}893 \pm 1{,}475$	0.85 ± 0.2
F1	0.16 ± 0.0	0.20 ± 0.0	700 ± 7	26.20 ± 7.3	6.38 ± 0.1	$6{,}236 \pm 38$	0.88 ± 0.0
F2R	0.12 ± 0.0	0.15 ± 0.0	722 ± 19	16.80 ± 8.5	4.78 ± 1.4	$6{,}777 \pm 383$	0.91 ± 0.0
F3R	0.09 ± 0.0	0.11 ± 0.1	743 ± 25	11.10 ± 10.2	3.42 ± 1.7	$7{,}288 \pm 558$	0.94 ± 0.0
F2C	0.00 ± 0.0	0.00 ± 0.0	779 ± 0	0.00 ± 0.0	0.00 ± 0.0	$8{,}192 \pm 0$	1.00 ± 0.0
F3C	0.00 ± 0.0	0.00 ± 0.0	779 ± 0	0.00 ± 0.0	0.00 ± 0.0	$8{,}192 \pm 0$	1.00 ± 0.0
F2O	0.09 ± 0.0	0.10 ± 0.0	747 ± 18	8.00 ± 7.1	3.75 ± 1.1	$7{,}333 \pm 382$	0.95 ± 0.0
F3O	0.03 ± 0.0	0.03 ± 0.0	778 ± 4	0.20 ± 0.6	1.14 ± 0.9	$8{,}149 \pm 130$	1.00 ± 0.0
FTF	0.00 ± 0.0	0.00 ± 0.0	779 ± 0	0.00 ± 0.0	0.00 ± 0.0	$8{,}192 \pm 0$	1.00 ± 0.0
M1	3.55 ± 0.0	8.35 ± 0.0	700 ± 6	27.80 ± 6.4	163.84 ± 0.1	$6{,}220 \pm 30$	0.88 ± 0.0
M2R	1.85 ± 0.1	4.92 ± 0.2	718 ± 12	20.60 ± 5.4	124.61 ± 0.6	$6{,}602 \pm 301$	0.90 ± 0.0

Model	Mean square error	MSE on optimal actions	Optimal actions found	Value loss	Boltzmann value loss	Correctly ranked	Kendall τ
M3R	1.09 ± 0.2	2.81 ± 0.2	739 ± 33	14.60 ± 13.8	88.58 ± 2.3	$7{,}097 \pm 703$	0.93 ± 0.0
M2C	1.82 ± 0.0	4.90 ± 0.0	777 ± 0	0.00 ± 0.0	124.42 ± 0.1	$7{,}826 \pm 0$	0.97 ± 0.0
M3C	0.85 ± 0.0	2.58 ± 0.0	778 ± 1	0.00 ± 0.0	88.09 ± 0.1	$8{,}151 \pm 4$	1.00 ± 0.0
M2O	1.97 ± 0.1	5.08 ± 0.1	741 ± 13	11.70 ± 5.0	127.21 ± 1.9	$5{,}628 \pm 249$	0.84 ± 0.0
M3O	1.73 ± 1.2	4.96 ± 3.3	738 ± 55	8.70 ± 14.0	97.67 ± 10.9	$5{,}867 \pm 682$	0.85 ± 0.1
MTF	2.60 ± 0.0	6.14 ± 0.0	779 ± 0	0.00 ± 0.0	133.79 ± 0.1	$8{,}177 \pm 2$	1.00 ± 0.0
Aloha $n = 6$ (64 joint actions, 2 optimal)							
Joint	1.13 ± 0.0	0.00 ± 0.0	2 ± 0	0.00 ± 0.0	0.08 ± 0.0	51 ± 1	0.88 ± 0.0
F1	4.78 ± 0.0	50.93 ± 0.1	0 ± 0	6.00 ± 0.0	4.04 ± 0.0	27 ± 1	0.67 ± 0.0
F2R	4.05 ± 0.4	35.00 ± 7.0	0 ± 0	5.00 ± 1.3	3.69 ± 0.4	22 ± 4	0.70 ± 0.0
F3R	3.16 ± 0.5	20.64 ± 4.6	0 ± 0	4.20 ± 1.4	3.23 ± 0.9	26 ± 4	0.74 ± 0.0
F2C	0.91 ± 0.0	0.14 ± 0.0	2 ± 0	0.00 ± 0.0	-0.04 ± 0.0	42 ± 0	0.89 ± 0.0
F3C	0.07 ± 0.0	0.14 ± 0.0	2 ± 0	0.00 ± 0.0	0.22 ± 0.0	64 ± 0	1.00 ± 0.0
F2O	3.27 ± 0.3	20.63 ± 3.0	0 ± 0	4.40 ± 1.2	3.24 ± 0.5	23 ± 4	0.74 ± 0.0
F3O	1.46 ± 0.3	3.55 ± 1.3	1 ± 1	0.80 ± 1.3	1.19 ± 0.4	29 ± 5	0.83 ± 0.0
FTF	0.00 ± 0.0	0.00 ± 0.0	2 ± 0	0.00 ± 0.0	0.00 ± 0.0	64 ± 0	1.00 ± 0.0
M1	8.26 ± 0.0	50.84 ± 0.1	0 ± 0	6.00 ± 0.0	5.47 ± 0.0	27 ± 1	0.67 ± 0.0
M2R	6.52 ± 0.2	44.17 ± 1.5	0 ± 0	5.00 ± 1.3	4.57 ± 0.1	25 ± 4	0.70 ± 0.0
M3R	4.53 ± 0.5	31.35 ± 3.8	0 ± 0	4.00 ± 2.2	3.41 ± 0.6	28 ± 5	0.77 ± 0.1
M2C	6.51 ± 0.0	44.65 ± 0.1	0 ± 0	6.00 ± 0.0	4.59 ± 0.0	28 ± 0	0.76 ± 0.0
M3C	4.56 ± 0.0	33.45 ± 0.1	0 ± 0	6.00 ± 0.0	3.62 ± 0.0	36 ± 1	0.86 ± 0.0
M2O	6.63 ± 0.4	44.51 ± 1.1	0 ± 0	5.20 ± 1.0	4.62 ± 0.2	22 ± 5	0.66 ± 0.0
M3O	4.71 ± 0.3	33.36 ± 1.2	0 ± 0	5.20 ± 1.0	3.65 ± 0.2	25 ± 4	0.74 ± 0.0
MTF	3.20 ± 0.0	23.88 ± 0.1	0 ± 0	2.00 ± 0.0	2.78 ± 0.0	36 ± 1	0.88 ± 0.0
Dispersion Game $n = 9$ (512 joint actions, 252 optimal)							
Joint	0.93 ± 0.5	1.74 ± 0.9	162 ± 30	0.00 ± 0.0	0.25 ± 0.1	307 ± 77	0.49 ± 0.2
F1	0.74 ± 0.0	0.53 ± 0.0	124 ± 1	1.10 ± 0.9	0.47 ± 0.0	191 ± 3	0.02 ± 0.0
F2R	0.66 ± 0.0	0.53 ± 0.0	143 ± 2	0.00 ± 0.0	0.40 ± 0.0	206 ± 3	0.19 ± 0.0
F3R	0.57 ± 0.0	0.51 ± 0.0	171 ± 2	0.40 ± 0.5	0.32 ± 0.0	279 ± 3	0.37 ± 0.0
F2C	0.06 ± 0.0	0.03 ± 0.0	252 ± 0	0.00 ± 0.0	0.09 ± 0.0	512 ± 0	1.00 ± 0.0
F3C	0.06 ± 0.0	0.03 ± 0.0	252 ± 0	0.00 ± 0.0	0.09 ± 0.0	512 ± 0	1.00 ± 0.0
F2O	0.57 ± 0.0	0.49 ± 0.0	159 ± 4	0.20 ± 0.4	0.32 ± 0.0	254 ± 9	0.35 ± 0.0
F3O	0.36 ± 0.0	0.33 ± 0.0	178 ± 4	0.10 ± 0.3	0.21 ± 0.0	305 ± 14	0.54 ± 0.0
M1	0.74 ± 0.0	0.53 ± 0.0	124 ± 1	0.90 ± 0.8	0.47 ± 0.0	192 ± 4	0.01 ± 0.0
M2R	0.70 ± 0.0	0.52 ± 0.0	142 ± 2	0.00 ± 0.0	0.45 ± 0.0	204 ± 5	0.19 ± 0.0
M3R	0.64 ± 0.0	0.48 ± 0.0	170 ± 1	0.30 ± 0.5	0.42 ± 0.0	276 ± 4	0.37 ± 0.0
M2C	0.70 ± 0.0	0.51 ± 0.0	252 ± 0	0.00 ± 0.0	0.45 ± 0.0	512 ± 0	1.00 ± 0.0
M3C	0.63 ± 0.0	0.46 ± 0.0	252 ± 0	0.00 ± 0.0	0.42 ± 0.0	512 ± 0	1.00 ± 0.0
M2O	0.70 ± 0.0	0.51 ± 0.0	159 ± 3	0.10 ± 0.3	0.45 ± 0.0	254 ± 10	0.35 ± 0.0
M3O	0.63 ± 0.0	0.47 ± 0.0	172 ± 4	0.10 ± 0.3	0.42 ± 0.0	289 ± 12	0.48 ± 0.0
Dispersion Game $n = 12$ (4096 joint actions, 924 optimal)							
Joint	19.48 ± 0.7	31.72 ± 0.9	210 ± 8	2.30 ± 1.0	0.80 ± 0.0	$1{,}108 \pm 34$	0.00 ± 0.0
F1	1.17 ± 0.0	1.82 ± 0.0	186 ± 39	2.40 ± 1.3	0.80 ± 0.0	$1{,}137 \pm 37$	0.01 ± 0.0
F2R	1.08 ± 0.0	1.75 ± 0.0	303 ± 14	0.00 ± 0.0	0.70 ± 0.0	$1{,}635 \pm 17$	0.18 ± 0.0
F3R	0.99 ± 0.0	1.67 ± 0.0	351 ± 7	0.90 ± 0.7	0.63 ± 0.0	$1{,}364 \pm 20$	0.28 ± 0.0
F2C	0.17 ± 0.0	0.37 ± 0.0	924 ± 0	0.00 ± 0.0	0.27 ± 0.0	$4{,}096 \pm 0$	1.00 ± 0.0
F3C	0.20 ± 0.0	0.43 ± 0.1	774 ± 194	0.00 ± 0.0	0.27 ± 0.0	$3{,}711 \pm 510$	0.92 ± 0.1
F2O	0.99 ± 0.0	1.64 ± 0.0	297 ± 8	0.60 ± 0.5	0.64 ± 0.0	$1{,}403 \pm 56$	0.28 ± 0.0
F3O	0.74 ± 0.0	1.31 ± 0.0	344 ± 13	0.70 ± 0.6	0.49 ± 0.0	$1{,}673 \pm 40$	0.43 ± 0.0
M1	1.17 ± 0.0	1.83 ± 0.0	187 ± 30	2.20 ± 1.5	0.80 ± 0.0	$1{,}134 \pm 36$	0.00 ± 0.0
M2R	1.14 ± 0.0	1.80 ± 0.0	303 ± 4	0.00 ± 0.0	0.78 ± 0.0	$1{,}633 \pm 6$	0.18 ± 0.0
M3R	1.09 ± 0.0	1.75 ± 0.0	350 ± 9	1.00 ± 0.6	0.75 ± 0.0	$1{,}363 \pm 24$	0.28 ± 0.0

Model	Mean square error	MSE on optimal actions	Optimal actions found	Value loss	Boltzmann value loss	Correctly ranked	Kendall τ
M2C	1.14 ± 0.0	1.80 ± 0.0	813 ± 69	0.00 ± 0.0	0.78 ± 0.0	$3{,}831 \pm 246$	0.95 ± 0.0
M3C	1.08 ± 0.0	1.74 ± 0.0	920 ± 5	0.00 ± 0.0	0.75 ± 0.0	$4{,}089 \pm 10$	1.00 ± 0.0
M2O	1.14 ± 0.0	1.80 ± 0.0	291 ± 8	0.40 ± 0.5	0.78 ± 0.0	$1{,}348 \pm 37$	0.27 ± 0.0
M3O	1.08 ± 0.0	1.74 ± 0.0	329 ± 12	0.60 ± 0.5	0.75 ± 0.0	$1{,}560 \pm 57$	0.39 ± 0.0
Generalized FF $n = 9$ (524.288 joint actions, 17.682 optimal)							
Joint	69.98 ± 0.9	142.27 ± 1.3	$2{,}556 \pm 50$	$3{,}808.40 \pm 60.8$	$2{,}339.16 \pm 1.3$	$94{,}559 \pm 522$	0.02 ± 0.0
F1	0.25 ± 0.0	0.31 ± 0.0	$14{,}887 \pm 90$	319.00 ± 103.2	64.62 ± 1.0	$333{,}466 \pm 1{,}069$	0.86 ± 0.0
F2R	0.68 ± 0.2	0.86 ± 0.3	$11{,}986 \pm 546$	621.40 ± 64.6	325.70 ± 51.0	$257{,}663 \pm 12{,}168$	0.78 ± 0.0
F3R	0.17 ± 0.0	0.20 ± 0.1	$15{,}934 \pm 479$	157.80 ± 74.6	38.63 ± 12.7	$393{,}625 \pm 30{,}195$	0.91 ± 0.0
F2C	0.00 ± 0.0	0.00 ± 0.0	$17{,}682 \pm 0$	0.00 ± 0.0	0.00 ± 0.0	$524{,}288 \pm 0$	1.00 ± 0.0
F3C	0.00 ± 0.0	0.00 ± 0.0	$17{,}682 \pm 0$	0.00 ± 0.0	0.00 ± 0.0	$524{,}288 \pm 0$	1.00 ± 0.0
F2O	0.18 ± 0.0	0.21 ± 0.0	$15{,}761 \pm 286$	185.70 ± 40.5	40.58 ± 10.1	$385{,}288 \pm 20{,}999$	0.90 ± 0.0
F3O	0.09 ± 0.0	0.10 ± 0.0	$16{,}939 \pm 415$	51.40 ± 38.7	22.72 ± 8.6	$464{,}893 \pm 26{,}119$	0.96 ± 0.0
FTF	0.00 ± 0.0	0.00 ± 0.0	$17{,}682 \pm 0$	0.00 ± 0.0	0.00 ± 0.0	$524{,}288 \pm 0$	1.00 ± 0.0
M1	6.55 ± 0.0	17.68 ± 0.0	$14{,}848 \pm 73$	331.80 ± 62.7	$2{,}030.74 \pm 0.9$	$333{,}711 \pm 767$	0.86 ± 0.0
M2R	4.54 ± 0.2	13.09 ± 0.2	$12{,}444 \pm 531$	596.60 ± 121.2	$1{,}718.26 \pm 11.8$	$265{,}765 \pm 10{,}895$	0.80 ± 0.0
M3R	2.77 ± 0.2	9.05 ± 0.2	$15{,}596 \pm 429$	209.40 ± 59.9	$1{,}425.89 \pm 9.9$	$371{,}664 \pm 25{,}374$	0.89 ± 0.0
M2C	4.30 ± 0.0	12.83 ± 0.0	$17{,}680 \pm 0$	0.00 ± 0.0	$1{,}721.18 \pm 0.9$	$465{,}019 \pm 2$	0.95 ± 0.0
M3C	2.71 ± 0.0	8.99 ± 0.0	$17{,}680 \pm 0$	0.00 ± 0.0	$1{,}422.88 \pm 0.7$	$465{,}090 \pm 2$	0.95 ± 0.0
M2O	4.37 ± 0.1	12.87 ± 0.2	$15{,}439 \pm 433$	231.40 ± 72.6	$1{,}723.51 \pm 16.1$	$267{,}534 \pm 15{,}642$	0.78 ± 0.0
M3O	2.96 ± 0.1	9.24 ± 0.1	$16{,}783 \pm 303$	54.00 ± 23.9	$1{,}431.61 \pm 23.0$	$298{,}182 \pm 26{,}364$	0.82 ± 0.0
MTF	5.30 ± 0.0	14.34 ± 0.0	$17{,}682 \pm 0$	0.00 ± 0.0	$1{,}765.64 \pm 1.0$	$512{,}022 \pm 810$	0.99 ± 0.0
Aloha $n = 9$ (512 joint actions, 1 optimal)							
Joint	4.86 ± 1.6	47.77 ± 9.2	0 ± 0	4.40 ± 1.7	4.47 ± 0.6	215 ± 17	0.71 ± 0.0
F1	6.14 ± 0.0	130.68 ± 0.6	0 ± 0	10.00 ± 0.0	6.62 ± 0.0	127 ± 1	0.64 ± 0.0
F2R	6.97 ± 0.4	123.35 ± 26.5	0 ± 0	9.00 ± 1.8	6.73 ± 0.3	100 ± 11	0.58 ± 0.0
F3R	5.05 ± 0.5	87.35 ± 15.5	0 ± 0	8.00 ± 2.2	5.97 ± 0.6	133 ± 7	0.68 ± 0.0
F2C	2.25 ± 0.0	9.03 ± 0.2	1 ± 0	0.00 ± 0.0	2.06 ± 0.0	187 ± 1	0.82 ± 0.0
F3C	1.32 ± 0.0	18.60 ± 0.3	0 ± 0	2.00 ± 0.0	1.47 ± 0.0	365 ± 1	0.88 ± 0.0
F2O	4.98 ± 0.2	82.30 ± 7.7	0 ± 0	6.60 ± 1.8	5.71 ± 0.4	135 ± 9	0.69 ± 0.0
F3O	3.95 ± 0.4	49.17 ± 9.6	0 ± 0	7.00 ± 2.2	5.01 ± 0.7	138 ± 8	0.73 ± 0.0
FTF	0.00 ± 0.0	0.00 ± 0.0	1 ± 0	0.00 ± 0.0	0.00 ± 0.0	512 ± 0	1.00 ± 0.0
M1	11.93 ± 0.0	145.97 ± 0.1	0 ± 0	10.00 ± 0.0	9.68 ± 0.0	126 ± 1	0.64 ± 0.0
M2R	10.45 ± 0.2	140.15 ± 5.7	0 ± 0	9.00 ± 1.6	8.85 ± 0.1	105 ± 11	0.60 ± 0.0
M3R	8.71 ± 0.3	125.24 ± 6.2	0 ± 0	7.80 ± 0.6	7.85 ± 0.2	131 ± 7	0.69 ± 0.0
M2C	10.31 ± 0.0	138.52 ± 0.1	0 ± 0	10.00 ± 0.0	8.79 ± 0.0	140 ± 1	0.68 ± 0.0
M3C	8.68 ± 0.0	126.24 ± 0.2	0 ± 0	10.00 ± 0.0	7.86 ± 0.0	152 ± 1	0.72 ± 0.0
M2O	10.48 ± 0.2	138.60 ± 5.8	0 ± 0	9.60 ± 0.8	8.86 ± 0.1	114 ± 10	0.62 ± 0.0
M3O	8.86 ± 0.4	124.02 ± 12.3	0 ± 0	9.40 ± 1.8	7.87 ± 0.3	125 ± 12	0.63 ± 0.0
MTF	6.25 ± 0.0	85.95 ± 1.1	0 ± 0	2.00 ± 0.0	6.10 ± 0.0	191 ± 5	0.84 ± 0.0
Aloha $n = 12$ (4096 joint actions, 2 optimal)							
Joint	23.98 ± 0.6	135.36 ± 2.5	0 ± 0	7.40 ± 3.1	12.12 ± 0.3	700 ± 13	0.42 ± 0.0
F1	7.30 ± 0.0	231.09 ± 0.3	0 ± 0	12.00 ± 0.0	7.96 ± 0.0	861 ± 4	0.64 ± 0.0
F2R	6.84 ± 0.2	199.45 ± 11.1	0 ± 0	10.40 ± 1.2	7.67 ± 0.2	851 ± 33	0.65 ± 0.0
F3R	6.56 ± 0.3	182.17 ± 15.5	0 ± 0	9.80 ± 2.1	7.59 ± 0.3	860 ± 47	0.66 ± 0.0
F2C	2.18 ± 0.0	10.36 ± 0.1	2 ± 0	0.00 ± 0.0	2.06 ± 0.0	$1{,}380 \pm 5$	0.83 ± 0.0
F3C	0.81 ± 0.0	12.45 ± 0.2	1 ± 0	2.00 ± 0.0	1.10 ± 0.0	$2{,}488 \pm 10$	0.91 ± 0.0
F2O	6.32 ± 0.3	169.46 ± 13.1	0 ± 0	10.20 ± 1.7	7.29 ± 0.3	864 ± 34	0.67 ± 0.0
F3O	4.71 ± 0.5	92.22 ± 18.6	0 ± 0	7.60 ± 2.3	5.94 ± 0.7	954 ± 67	0.73 ± 0.0
FTF	0.00 ± 0.0	0.00 ± 0.0	2 ± 0	0.00 ± 0.0	0.00 ± 0.0	$4{,}096 \pm 0$	1.00 ± 0.0
M1	15.37 ± 0.0	230.65 ± 0.4	0 ± 0	12.00 ± 0.0	12.45 ± 0.0	845 ± 26	0.63 ± 0.0
M2R	13.85 ± 0.1	225.18 ± 2.3	0 ± 0	10.80 ± 1.3	11.61 ± 0.1	838 ± 22	0.65 ± 0.0

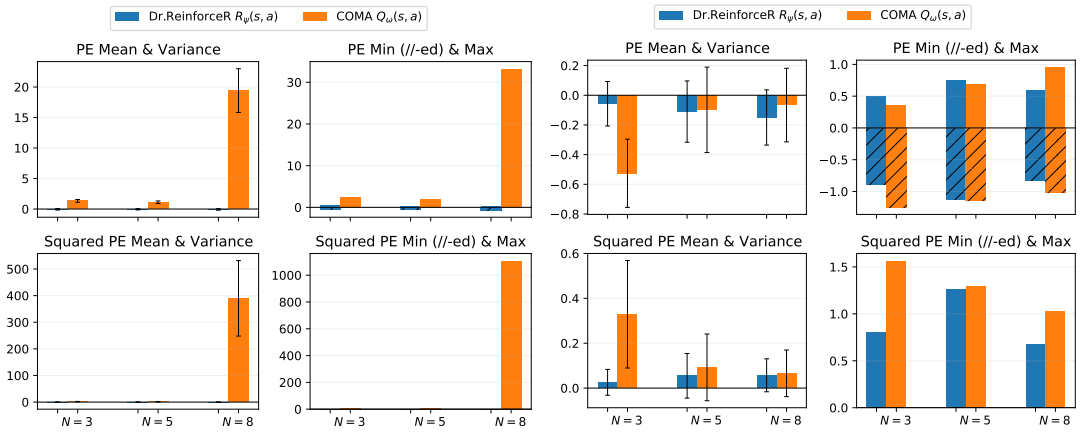
Model	Mean square error	MSE on optimal actions	Optimal actions found	Value loss	Boltzmann value loss	Correctly ranked	Kendall τ
M3R	12.38 ± 0.2	217.04 ± 5.0	0 ± 0	10.00 ± 1.8	10.76 ± 0.1	853 ± 34	0.66 ± 0.0
M2C	13.83 ± 0.0	225.33 ± 0.2	0 ± 0	12.00 ± 0.0	11.60 ± 0.0	908 ± 7	0.67 ± 0.0
M3C	12.27 ± 0.0	214.92 ± 0.2	0 ± 0	12.00 ± 0.0	10.69 ± 0.0	969 ± 6	0.71 ± 0.0
M2O	13.94 ± 0.2	225.44 ± 1.5	0 ± 0	11.60 ± 0.8	11.61 ± 0.1	671 ± 58	0.56 ± 0.0
M3O	12.56 ± 0.3	214.01 ± 3.2	0 ± 0	11.40 ± 0.9	10.85 ± 0.2	752 ± 87	0.61 ± 0.0
MTF	9.64 ± 0.0	167.68 ± 0.6	0 ± 0	4.00 ± 0.0	8.82 ± 0.0	$1{,}254 \pm 14$	0.83 ± 0.0

Table B.1: Accuracy results with respect to both action ranking and reconstruction error for the different games. Best (green) and worst (red) performances for each game are highlighted.



(a) Multi-Rover (b) Predator-Prey

Figure B.1: Distribution statistics for Dr.ReinforceR reward network R_ψ and COMA critic Q_ω on the on-policy dataset, normalized by the value of $r_{max} - r_{min}$ (respectively $q_{max} - q_{min}$ for COMA critic), for the two environments.



(a) Multi-Rover (b) Predator-Prey

Figure B.2: Distribution statistics for Dr.ReinforceR reward network R_ψ and COMA critic Q_ω on the off-policy dataset, normalized by the value of $r_{max} - r_{min}$ (respectively $q_{max} - q_{min}$ for COMA critic), for the two environments.

B.2 ADDITIONAL GRIDWORLD ANALYSIS PLOTS

Figures B.1 and B.2 show a comparison of Dr.ReinforceR centralized reward network $R_\psi(s,a)$ and COMA Q-function critic $Q_\omega(s,a)$ in terms of prediction error on the on-policy and off-policy dataset respectively. Both the mean error with variance and the minimum and maximum errors are reported, together with the squared values for both. Again, all the prediction errors have been normalized by the value of $r_{max} - r_{min}$ (respectively $q_{max} - q_{min}$ for COMA critic) for each environment and number of agents individually, so that the resulting values are comparable across the two different methodologies and across different instances.

These figures clearly show how learning an accurate centralized representation

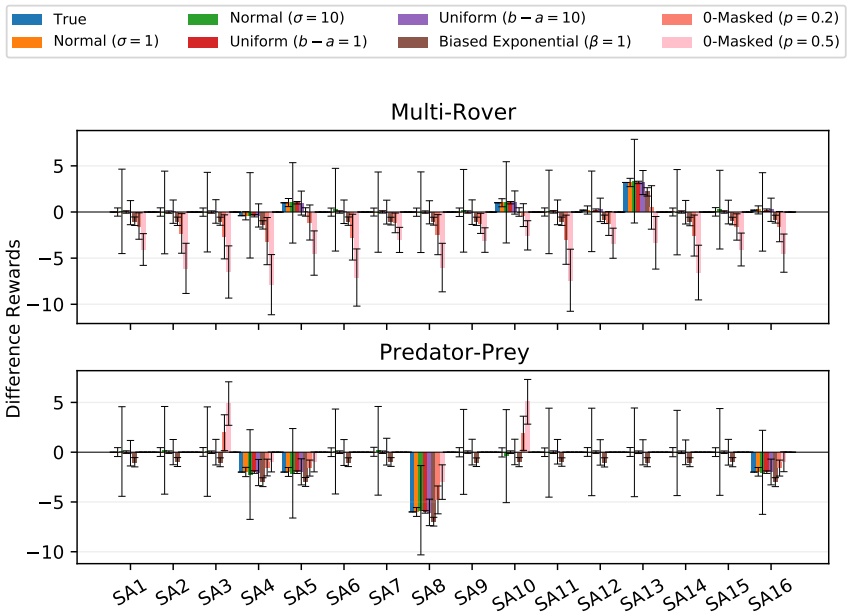


Figure B.3: Mean and variance of difference rewards for a set of samples under different noise profiles.

of the action-value function in a multi-agent system may be a challenging task [53]: COMA squared mean error is high on both problems and datasets when the number of agents increases. This is due to the exponential number of joint actions that the critic needs to approximate, together with bootstrapping that exacerbates these difficulties by making wrong estimates influence other estimates in turn. On the other hand, the reward network seems to learn a good approximation that is capable of generalizing also to possibly unseen state-action pairs, as shown by the low errors on the off-policy dataset. Although it still depends on the exponentially many joint actions (and thus faces an harder learning problem when the size of the system increases), the simpler regression task seems to help in achieving an accurate approximation.

Finally, Figure B.3 shows the effect of more types of noise on the difference rewards computation.

0-masking seems to be extremely detrimental: if some of the reward values uses in the computation are cancelled, the resulting difference rewards value may be entirely different from the true one. However, other types of noise, like uniform or biased noise, do not seem to alter the final value significantly, probably cancelling out the relative errors of each reward term one with another. This means that, although the introduced centralized reward network is only capable of learning an approximation of the true reward function, such an approximation does not need to be perfect in order to provide the agents with a perfect or nearly-perfect learning signal.

B.3 ADDITIONAL SMAC PLOTS

Chapter 5 showed the improved performances of Dr.ReinforceR over other policy gradient methods on SMAC in terms of median return, together with the win rate for some of the most successfully solved scenarios. In the following Figure B.4, the median win rate and $25-75\%$ percentiles across 10 independent runs for the complete set of available scenarios is reported. These plots highlight how Dr.ReinforceR, although capable of generally improved performances on the simpler settings, is still in general performing poorly (as generally policy gradient methods are known to do) on the most challenging ones. Indeed, on some of these scenarios, like those presenting unbalanced teams for the two factions, no significant win rate is achieved, meaning that the learned team of agents is never capable of outperform the opposing AI.

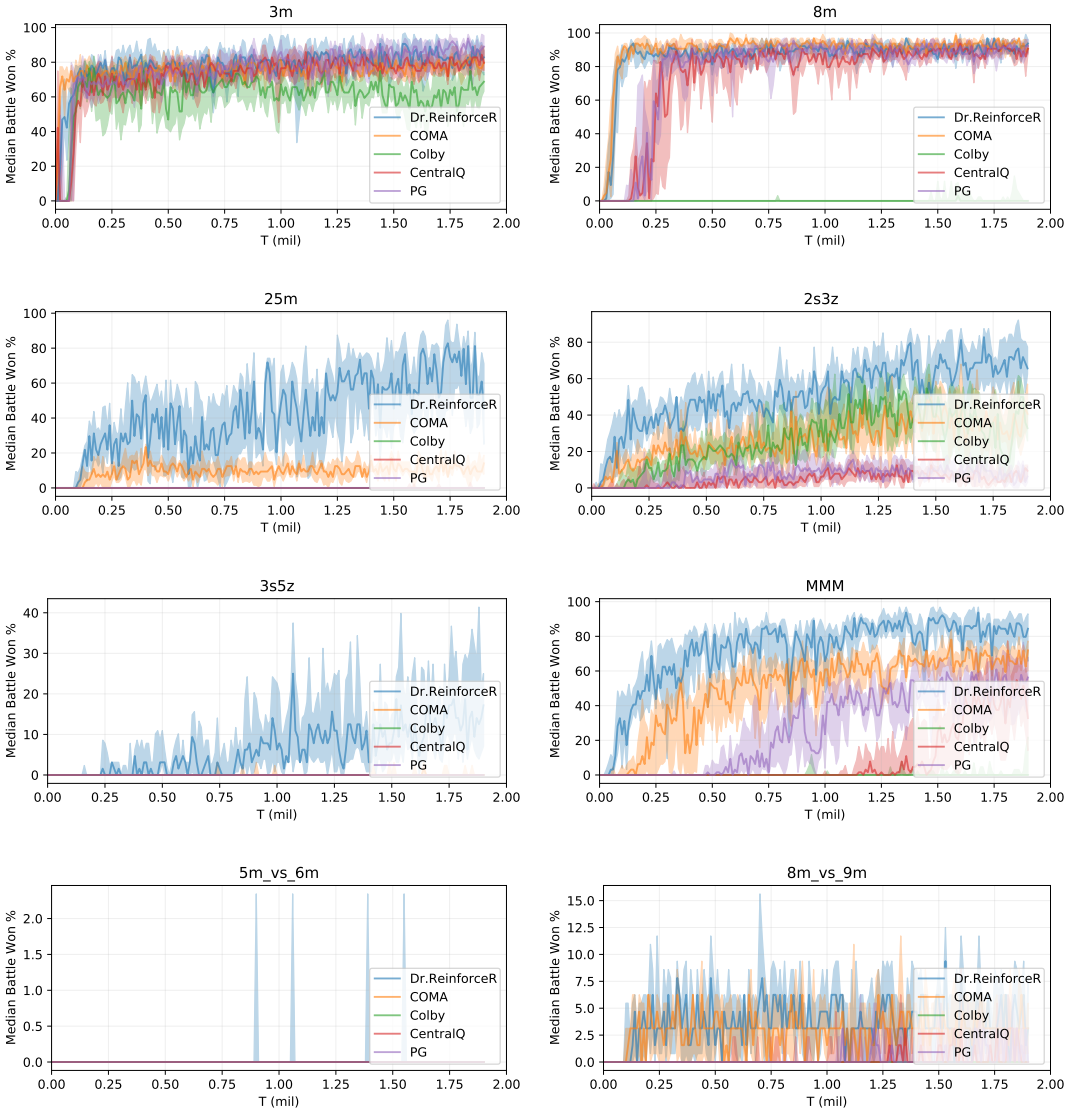


Figure B.4: Training curves on the entire set of SMAC scenarios, showing the median victory rate and $25-75\%$ percentiles across seeds.

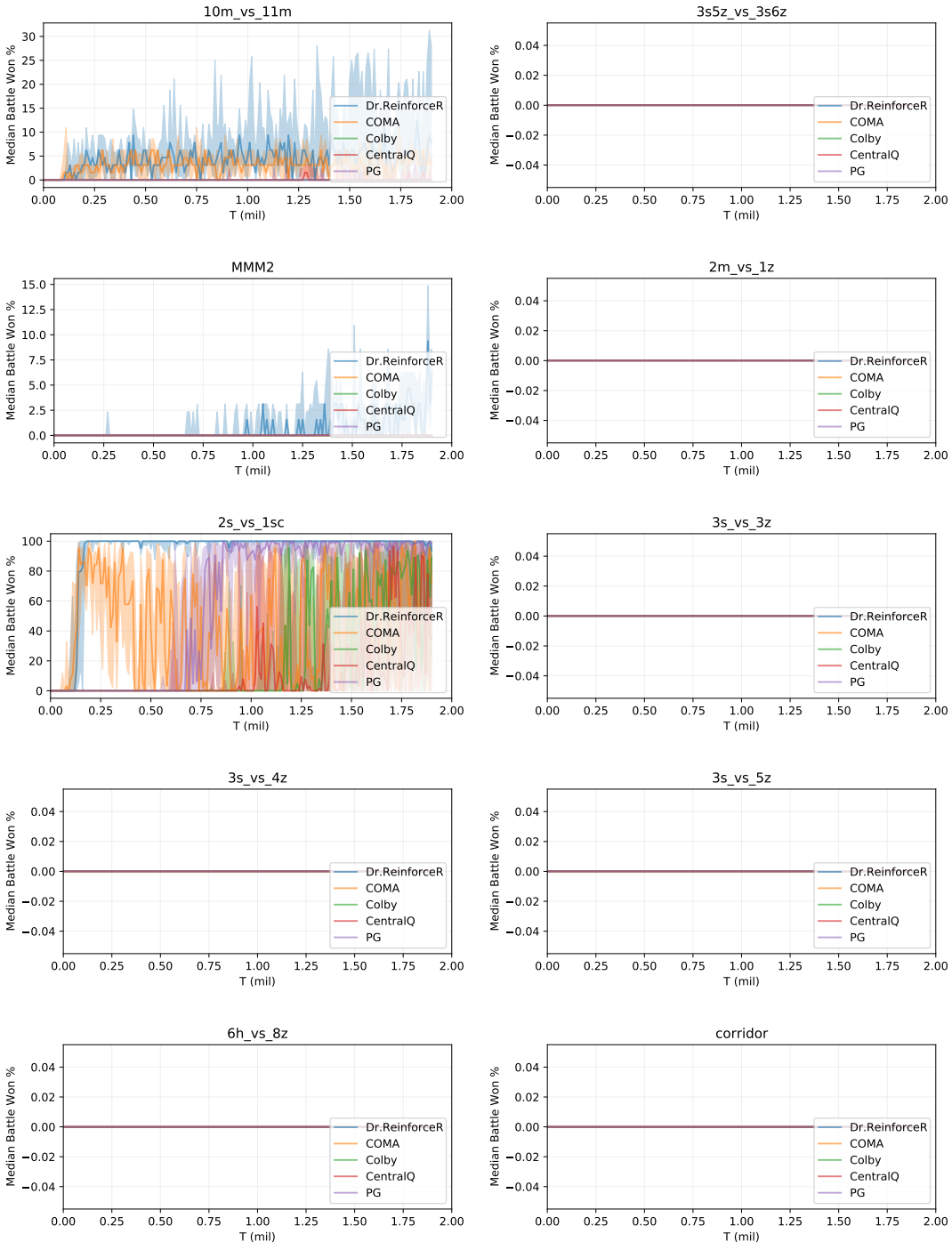


Figure B.4: Training curves on the entire set of SMAC scenarios, showing the median victory rate and $25-75\%$ percentiles across seeds.

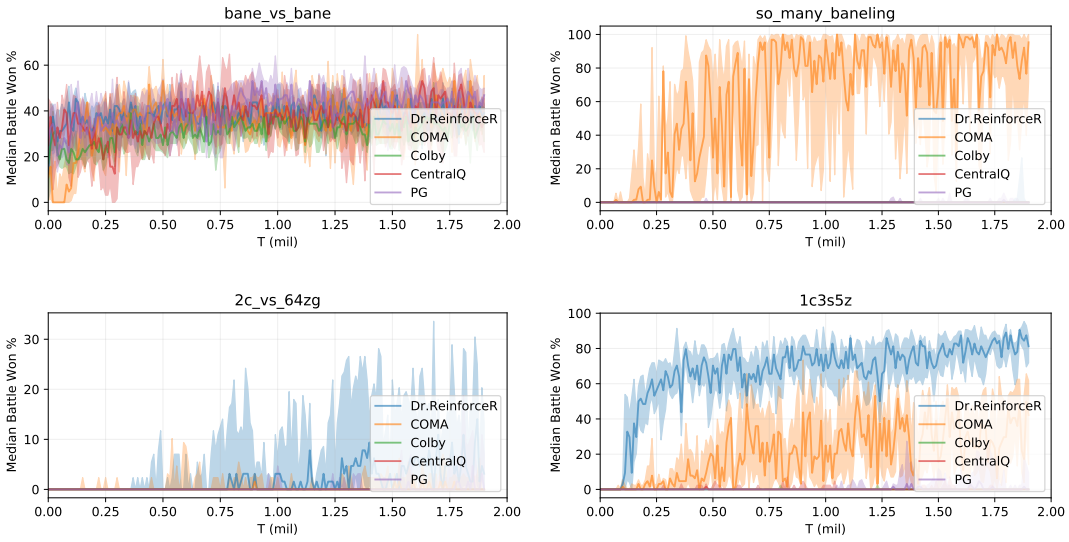


Figure B.4: Training curves on the entire set of SMAC scenarios, showing the median victory rate and $25-75\%$ percentiles across seeds.

ACRONYMS

AI Artificial Intelligence.

AU Aristocrat Utility.

CG Coordination Graph.

COMA Counterfactual Multi-Agent Policy Gradients.

CTDE Centralized Training-Decentralized Execution.

DCG Deep Coordination Graph.

Dec-POMDP Decentralized Partially Observable Markov Decision Process.

DL Deep Learning.

DQN Deep Q-Network.

DR Difference Rewards.

DRL Deep Reinforcement Learning.

GPU Graphical Processing Unit.

GRU Gated Recurrent Unit.

GT Game Theory.

IGM Individual-Global Maximum.

IL Independent Learners.

IQL Independent Q-Learners.

LSTM Long Short-Term Memory.

MACA Multi-Agent Credit Assignment.

MADDPG Multi-Agent Deep Deterministic Policy Gradients.

MAPG Multi-Agent Policy Gradients.

MARL Multi-Agent Reinforcement Learning.

MAS Multi-Agent System.

MDP Markov Decision Process.

MMDP Multi-Agent Markov Decision Process.

MSE Mean-Squared Error.

NE Nash Equilibrium.

NN Neural Network.

PE Prediction Error.

PG Policy Gradients.

POMDP Partially Observable Markov Decision Process.

POSG Partially Observable Stochastic Game.

RL Reinforcement Learning.

RNN Recurrent Neural Network.

SG Stochastic Game.

SGD Stochastic Gradient Descent.

SMAC StarCraft II Multi-Agent Challenge.

TD Temporal Difference.

TPU Tensor Processing Unit.

VDN Value-Decomposition Networks.

WLU Wonderful Life Utility.

BIBLIOGRAPHY

[1] Adrian K. Agogino and Kagan Tumer. Analyzing and visualizing multiagent rewards in dynamic and stochastic domains. *Autonomous Agents and Multi-Agent Systems*, 17(2):320–338, 2008.

[2] Christopher Amato and Frans A. Oliehoek. Scalable planning and learning for multiagent POMDPs. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, AAAI'15, pages 1995–2002. AAAI Press, 2015.

[3] 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.

[4] Robert Axelrod and William D. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

[5] Andrea Baisero and Christopher Amato. Unbiased asymmetric actor-critic for partially observable reinforcement learning. *arXiv*, abs/2105.11674:1–23, 2021.

[6] Marc G. Bellemare, Salvatore Candido, Pablo S. Castro, Jun Gong, Marlos C. Machado, Subhodeep Moitra, Sameera S. Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588 (7836):77–82, 2020.

[7] Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[8] Dimitri P. Bertsekas. Multiagent value iteration algorithms in dynamic programming and reinforcement learning. *arXiv*, abs/2005.01627:1–18, 2020.

[9] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal speed and accuracy of object detection. *arXiv*, abs/2004.10934:1–17, 2020.

[10] Wendelin Böehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, pages 980–991. PMLR, 2020.

[11] Léon Bottou. *On-Line Learning and Stochastic Approximations*, pages 9–42. Cambridge University Press, 1999.

[12] Matthew Botvinick, Jane X. Wang, Will Dabney, Kevin J. Miller, and Zeb Kurth-Nelson. Deep reinforcement learning and its neuroscientific implications. *Neuron*, 107(4):603–616, 2020.

[13] Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *Proceedings of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK'96, pages 195–210. Morgan Kaufmann Publishers Inc., 1996.

[14] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(2):156–172, 2008.

[15] Yongcan Cao, Wenwu Yu, Wei Ren, and Guanrong Chen. An overview of recent progress in the study of distributed multi-agent coordination. *IEEE Transactions on Industrial Informatics*, 9(1):427–438, 2013.

[16] Jacopo Castellini, Frans A. Oliehoek, Rahul Savani, and Shimon Whiteson. The representational capacity of action-value networks for multi-agent reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'19, pages 1862–1864. International Foundation for Autonomous Agents and MultiAgent Systems, 2019.

[17] Jacopo Castellini, Frans A. Oliehoek, Rahul Savani, and Shimon Whiteson. Analysing factorizations of action-value networks for cooperative multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 35(25):53 pages, 2021.

[18] Michael Castronovo, Francis Maes, Raphael Fonteneau, and Damien Ernst. Learning exploration/exploitation strategies for single trajectory reinforcement learning. In *Proceedings of the 10th European Workshop on Reinforcement Learning*, EWRL'13, pages 1–10. PMLR, 2013.

[19] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. *Computational Aspects of Cooperative Game Theory*. Morgan & Claypool Publishers, 1st edition, 2011.

[20] Yu-Han Chang, Tracey Ho, and Leslie P. Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Advances in Neural Information Processing Systems 17*, NIPS'03, pages 807–814. MIT Press, 2003.

[21] Shu-Yu Chen, Wanchao Su, Lin Gao, Shihong Xia, and Hongbo Fu. Deepfacedrawing: Deep generation of face images from sketches. *ACM Transactions on Graphics*, 39(4):16 pages, 2020.

[22] Junyoung Chung, Caglar Gulcehre, Kyung Hyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS'14 Workshop on Deep Learning and Representation Learning*, NIPS'14, page 9, 2014.

[23] Daniel Claes, Philipp Robbel, Frans A. Oliehoek, Karl Tuyls, Daniel Hennes, and Wiebe van der Hoek. Effective approximations for multi-robot coordination in spatially distributed tasks. In *Proceedings of the 14th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'15, pages 881–890. International Foundation for Autonomous Agents and MultiAgent Systems, 2015.

[24] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In *Proceedings of the 15th AAAI Conference on Artificial Intelligence*, AAAI'98, pages 746–752. AAAI Press, 1998.

[25] Mitchell K. Colby, William Curran, Carrie Rebhuhn, and Kagan Tumer. Approximating difference evaluations with local knowledge. In *Proceedings of the 13th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'14, pages 1577–1578. International Foundation for Autonomous Agents and MultiAgent Systems, 2014.

[26] Mitchell K. Colby, William Curran, and Kagan Tumer. Approximating difference evaluations with local information. In *Proceedings of the 14th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'15, pages 1659–1660. International Foundation for Autonomous Agents and MultiAgent Systems, 2015.

[27] George B. Dantzig. Linear programming. *Operations Research*, 50(1):42–47, 2002.

[28] Li Deng. A tutorial survey of architectures, algorithms, and applications for deep learning. *APSIPA Transactions on Signal and Information Processing*, 3(1):29 pages, 2014.

[29] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv*, abs/1810.04805:1–16, 2018.

[30] Sam Devlin and Daniel Kudenko. Theoretical considerations of potential-based reward shaping for multi-agent systems. In *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems*, AAMAS'11, pages 225–232. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

[31] Sam Devlin and Daniel Kudenko. Dynamic potential-based reward shaping. In *Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'12, pages 433–440. International Foundation for Autonomous Agents and MultiAgent Systems, 2012.

[32] Sam Devlin, Logan Yliniemi, Daniel Kudenko, and Kagan Tumer. Potential-based difference rewards for multiagent reinforcement learning. In *Proceedings*

of the 13th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS'14, pages 165–172. International Foundation for Autonomous Agents and MultiAgent Systems, 2014.

[33] Ali Dorri, Salil S. Kanhere, and Raja Jurdak. Multi-agent systems: A survey. *IEEE Access*, 6:28573–28593, 2018.

[34] Prashant Doshi and Piotr J. Gmytrasiewicz. Monte Carlo sampling methods for approximating interactive POMDPs. *Journal of Artificial Intelligence Research*, 34(1):297–337, 2009.

[35] Gabriel Dulac-Arnold, Daniel J. Mankowitz, and Todd Hester. Challenges of real-world reinforcement learning. *arXiv*, abs/1904.12901:1–13, 2019.

[36] Adam Eck, Leen-Kiat Soh, Sam Devlin, and Daniel Kudenko. Potential-based reward shaping for finite horizon online POMDP planning. *Autonomous Agents and Multi-Agent Systems*, 30(3):403–445, 2015.

[37] Nima Fazeli, Miquel Oller, Jiajun Wu, Z. Wu, Joshua B. Tenenbaum, and A. Rodriguez. See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. *Science Robotics*, 4(26):21 pages, 2019.

[38] Jakob Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 30*, NIPS'16, pages 2137–2145. Curran Associates, Inc., 2016.

[39] 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*, AAMAS'18, pages 122–130. International Foundation for Autonomous Agents and MultiAgent Systems, 2018.

[40] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the 32th AAAI Conference on Artificial Intelligence*, AAAI'18, pages 2974–2982. AAAI Press, 2018.

[41] Vincent Francois-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends in Machine Learning*, 11(3-4):219–354, 2018.

[42] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'18, pages 1587–1596. PMLR, 2018.

[43] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory and Practice*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers Inc., 1st edition, 2004.

[44] Ben Goertzel and Cassio Pennachin. *Artificial General Intelligence*. Cognitive Technologies. Springer, 1st edition, 2007.

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 1st edition, 2016.

[46] Alex Graves. Generating sequences with recurrent neural networks. *arXiv*, abs/1308.0850:1–43, 2013.

[47] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5:1471–1530, 2004.

[48] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions of Neural Network and Learning Systems*, 28(10):2222–2232, 2017.

[49] Trond Grenager, Rob A. Powers, and Yoav Shoham. Dispersion games: General definitions and some specific learning results. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence*, AAAI'02, pages 398–403. AAAI Press, 2002.

[50] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In *Advances in Neural Information Processing Systems 15*, NIPS'02, pages 1523–1530. Morgan Kaufmann Publishers Inc., 2002.

[51] Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, ICML'02, pages 227–234. Morgan Kaufmann Publishers Inc., 2002.

[52] Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Proceedings of the 18th AAAI Conference on Artificial Intelligence*, AAAI'02, pages 253–259. AAAI Press, 2002.

[53] Carlos Guestrin, Daphne Koller, Ronald Parr, and Shobha Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research*, 19(1):399–468, 2003.

[54] Jayesh K. Gupta, Maxim Egorov, and Mykel J. Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *AAMAS'17 Workshop on Learning and Adaptation in Multi-Agent Systems*, AAMAS'17, pages 66–83. Springer, 2017.

[55] David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR'17, page 4. OpenReview.net, 2017.

[56] Eric A. Hansen, Daniel S. Bernstein, and Shlomo Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th AAAI Conference on Artifical Intelligence*, AAAI'04, pages 709–715. AAAI Press, 2004.

[57] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *AAAI'15 Fall Symposium on Sequential Decision Making for Intelligent Agents*, AAAI'15, 2015.

[58] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall Press, 2nd edition, 1998.

[59] 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.

[60] Douglas R. Hofstadter. *Metamagical Themas: Questing for the Essence of Mind and Pattern*. Basic Books, Inc., 1st edition, 1985.

[61] Wenlong Huang, Igor Mordatch, and Deepak Pathak. One policy to control them all: Shared modular policies for agent-agnostic control. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, pages 4455–4464. PMLR, 2020.

[62] Chris Huntingford, Elizabeth S Jeffers, Michael B Bonsall, Hannah M Christensen, Thomas Lees, and Hui Yang. Machine learning and artificial intelligence to aid climate change research and preparedness. *Environmental Research Letters*, 14(12):124007, 2019.

[63] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, pages 2961–2970. PMLR, 2019.

[64] Fei Jiang, Yong Jiang, Hui Zhi, Yi Dong, Hao Li, Sufeng Ma, Yilong Wang, Qiang Dong, Haipeng Shen, and Yongjun Wang. Artificial intelligence in healthcare: past, present and future. *Stroke and Vascular Neurology*, 2(4):230–243, 2017.

[65] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4(1):237–285, 1996.

[66] Maurice Kendall and Jean D. Gibbons. *Rank Correlation Methods*. Oxford University Press, 5th edition, 1990.

[67] Jelle R. Kok and Nikos Vlassis. Collaborative multiagent reinforcement learning by payoff propagation. *Journal of Machine Learning Research*, 7:1789–1828, 2006.

[68] Jelle R. Kok and Nikos A. Vlassis. Sparse cooperative Q-learning. In *Proceedings of the 21th International Conference on Machine Learning*, ICML'04, pages 481–488. Association for Computing Machinery, 2004.

[69] Vijay R. Konda and John N. Tsitsiklis. On actor-critic algorithms. *SIAM Journal of Control and Optimization*, 42(4):1143–1166, 2003.

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

[71] Lior Kuyer, Shimon Whiteson, Bram Bakker, and Nikos Vlassis. Multiagent reinforcement learning for urban traffic control using coordination graphs. In *Proceedings of the 6th Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML-PKDD'08, pages 656–671. Springer, 2008.

[72] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521 (7553):436–444, 2015.

[73] Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. In *Proceedings of the 16th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'17, pages 464–473. International Foundation for Autonomous Agents and MultiAgent Systems, 2017.

[74] Victor R. Lesser. Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):133—142, 1999.

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

[76] Sheng Li, Jayesh K. Gupta, Peter Morales, Ross Allen, and Mykel J. Kochenderfer. Deep implicit coordination graphs for multi-agent reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'21, pages 764–772. International Foundation for Autonomous Agents and MultiAgent Systems, 2021.

[77] 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. *arXiv*, abs/1509.02971:1–14, 2015.

[78] Xiao Liu, Yuanwei Liu, and Yue Chen. Reinforcement learning in multiple-UAV networks: Deployment and movement design. *IEEE Transactions on Vehicular Technology*, 68(8):8036–8049, 2019.

[79] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems 31*, NIPS'17, pages 6379–6390. Curran Associates, Inc., 2017.

[80] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'21, pages 844–852. International Foundation for Autonomous Agents and MultiAgent Systems, 2021.

[81] Xueguang Lyu, Andrea Baisero, Yuchen Xiao, and Christopher Amato. A deeper understanding of state-based critics in multi-agent reinforcement learning. *arXiv*, abs/2201.01221:1–20, 2022.

[82] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. MAVEN: Multi-agent variational exploration. In *Advances in Neural Information Processing Systems 33*, NeurIPS'19. Curran Associates, Inc., 2019.

[83] Laetitia Matignon, Guillaume J. Laurent, and Nadine Le Fort-Piat. Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems. *Knowledge Engineering Review*, 27(1):1–31, 2012.

[84] Francisco S. Melo and Manuela M. Veloso. Learning of coordination: exploiting sparse interactions in multiagent systems. In *Proceedings of the 8th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'09, pages 773–780. International Foundation for Autonomous Agents and MultiAgent Systems, 2009.

[85] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. In *NIPS'13 Workshop on Deep Learning*, NIPS'13, page 9, 2013.

[86] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[87] Volodymyr Mnih, Adrià P. Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings 33rd International Conference on Machine Learning*, ICML'16, pages 1928–1937. PMLR, 2016.

[88] Guillem Muñoz, Cristina Barrado, Ender Çetin, and Esther Salami. Deep reinforcement learning for drone delivery. *Drones*, 3(3):72 pages, 2019.

[89] Navid Naderializadeh, Fan H Hung, Sean Soleyman, and Deepak Khosla. Graph convolutional value decomposition in multi-agent reinforcement learning. *arXiv*, abs/2010.04740:1–19, 2020.

[90] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 807–814. Omnipress, 2010.

[91] John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49, 1950.

[92] Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems 31*, NIPS'17, pages 5947–5956. Curran Associates, Inc., 2017.

[93] Andrew Y. Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the 16th International Conference on Machine Learning*, ICML'99, pages 278–287. Morgan Kaufmann Publishers Inc., 1999.

[94] Duc T. Nguyen, Akshat Kumar, and Hoong C. Lau. Credit assignment for collective multiagent RL with global rewards. In *Advances in Neural Information Processing Systems 32*, NeurIPS'18, pages 8113–8124. Curran Associates, Inc., 2018.

[95] Raz Nissim and Ronen I. Brafman. Multi-agent Af* for parallel and distributed systems. In *Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'12, pages 1265–1266. International Foundation for Autonomous Agents and MultiAgent Systems, 2012.

[96] Ann Nowé, Katja Verbeeck, and Maarten Peeters. Learning automata as a basis for multi agent reinforcement learning. In *Proceedings of the 1st International Conference on Learning and Adaption in Multi-Agent Systems*, LAMAS'05, pages 71–85. Springer Berlin Heidelberg, 2006.

[97] Frans A. Oliehoek. *Value-Based Planning for Teams of Agents in Stochastic Partially Observable Environments*. PhD thesis, Informatics Institute, University of Amsterdam, 2010.

[98] Frans A. Oliehoek and Christoper Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer, 1st edition, 2016.

[99] Frans A. Oliehoek, Matthijs T. J. Spaan, Shimon Whiteson, and Nikos A. Vlassis. Exploiting locality of interaction in factored Dec-POMDPs. In *Proceedings of the 7th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'08, pages 517–524. International Foundation for Autonomous Agents and MultiAgent Systems, 2008.

[100] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Exploiting agent and type independence in collaborative graphical Bayesian games. *arXiv*, abs/1108.0404:1–46, 2011.

[101] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Exploiting structure in cooperative Bayesian games. In *Proceedings of the 28th Conference on Uncertainty in Artificial Intelligence*, UAI'12, pages 654–665. AUAI Press, 2012.

[102] Frans A. Oliehoek, Stefan J. Witwicki, and Leslie P. Kaelbling. Influence-based abstraction for multiagent systems. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1422–1428. AAAI Press, 2012.

[103] Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Approximate solutions for factored Dec-POMDPs with many agents. In *Proceedings of the 12th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'13, pages 563–570. International Foundation for Autonomous Agents and MultiAgent Systems, 2013.

[104] Frans A. Oliehoek, Matthijs T. J. Spaan, and Stefan J. Witwicki. Factored upper bounds for multiagent planning problems under uncertainty with non-factored value functions. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, IJCAI'15, pages 1645–1651. AAAI Press, 2015.

[105] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, 1st edition, 1994.

[106] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient multi-agent deep reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'18, pages 443–451. International Foundation for Autonomous Agents and MultiAgent Systems, 2018.

[107] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11(3):387–434, 2005.

[108] Georgios Papoudakis, Filippos Christianos, Arrasy Rahman, and Stefano V. Albrecht. Dealing with non-stationarity in multi-agent deep reinforcement learning. *arXiv*, abs/1906.04737:1–8, 2019.

[109] James Parker, Ernesto Nunes, Julio Godoy, and Maria Gini. Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork. *Journal of Field Robotics*, 33(7):877–900, 2016.

[110] Maarten Peeters, Ville Könönen, Katja Verbeeck, and Ann Nowé. A learning automata approach to multi-agent policy gradient learning. In *Proceedings of the 12th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, Part II*, KES'08, pages 379–390. Springer-Verlag, 2008.

[111] Bei Peng, Tabish Rashid, Christian Schröder de Witt, Pierre-Alexandre Kamienny, Philip H. S. Torr, Wendelin Böhmer, and Shimon Whiteson. FACMAC: Factored multi-agent centralised policy gradients. *arXiv*, abs/2003.06709:1–22, 2020.

[112] Leonid Peshkin, Kee-Eung Kim, Nicolas Meuleau, and Leslie Pack Kaelbling. Learning to cooperate via policy search. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI'00, pages 489–496. Morgan Kaufmann Publishers Inc., 2000.

[113] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. A survey on deep learning: Algorithms, techniques, and applications. *ACM Computing Surveys*, 51(5):1–36, 2018.

[114] Scott Proper and Kagan Tumer. Modeling difference rewards for multiagent learning. In *Proceedings of the 11th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'12, pages 1397–1398. International Foundation for Autonomous Agents and MultiAgent Systems, 2012.

[115] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*, ICML'18, pages 4292–4301. PMLR, 2018.

[116] D. Sai Koti Reddy, Amrita Saha, Srikanth G. Tamilselvam, Priyanka Agrawal, and Pankaj Dayama. Risk averse reinforcement learning for mixed multi-agent environments. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'19, pages 2171–2173. International Foundation for Autonomous Agents and MultiAgent Systems, 2019.

[117] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[118] Alex Rogers, Alessandro Farinelli, Ruben Stranders, and Nick R. Jennings. Bounded approximate decentralised coordination via the max-sum algorithm. *Artificial Intelligence*, 175(2):730–759, 2011.

[119] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[120] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. MIT Press, 1986.

[121] Gavin A. Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, 1994.

[122] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009.

[123] Mikayel Samvelyan, Tabish Rashid, Christian Schröeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft multi-agent challenge. *arXiv*, abs/1902.04043:1–14, 2019.

[124] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv*, abs/1511.05952:1–21, 2015.

[125] Jeff Schneider, Weng-Keen Wong, Andrew Moore, and Martin Riedmiller. Distributed value functions. In *Proceedings of the 16th International Conference on Machine Learning*, ICML'99, pages 371–378. Morgan Kaufmann Publishers Inc., 1999.

[126] Christian Schröder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the StarCraft multi-agent challenge? *arXiv*, abs/2011.09533:1–11, 2020.

[127] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, abs/1707.06347:1–12, 2017.

[128] Roberto Serrano. Cooperative games: Core and Shapley value. Technical report, CEMFI, 2007.

[129] Shahin Shahrampour, Alexander Rakhlin, and Ali Jadbabaie. Multi-armed bandits in multi-agent networks. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP'17, pages 2786–2790. Curran Associates, Inc., 2017.

[130] Jianzhun Shao, Hongchang Zhang, Yuhang Jiang, Shuncheng He, and Xiangyang Ji. Credit assignment with meta-policy gradient for multi-agent reinforcement learning. *arXiv*, abs/2102.12957:1–14, 2021.

[131] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953.

[132] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings 31st International Conference on Machine Learning*, ICML'14, pages 387–395. PMLR, 2014.

[133] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[134] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[135] Kyunghwan Son, Daewoo Kim, Wan J. Kang, David Hostallero, and Yung Yi. QTRAN: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, ICML'19, pages 5887–5896. PMLR, 2019.

[136] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[137] Jianyu Su, Stephen C. Adams, and Peter A. Beling. Value-decomposition multi-agent actor-critics. *arXiv*, abs/2007.12306:1–13, 2020.

[138] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems 30*, NIPS'16, pages 2252–2260. Curran Associates, Inc., 2016.

[139] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*,

AAMAS'18, pages 2085–2087. International Foundation for Autonomous Agents and MultiAgent Systems, 2018.

[140] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, 1988.

[141] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018.

[142] 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 14*, NIPS'00, pages 1057–1063. MIT Press, 2000.

[143] Pieter J. 't Hoen and Johannes A. La Poutré. A decommitment strategy in a competitive multi-agent transportation setting. In *Proceedings of the 2nd International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'03, pages 1010–1011. Association for Computing Machinery, 2003.

[144] Pieter J. 't Hoen, Karl Tuyls, Liviu Panait, Sean Luke, and Johannes A. La Poutré. An overview of cooperative and competitive multiagent learning. In *AAMAS'05 Workshop on Learning and Adaptation in Multi-Agent Systems*, AAMAS'05, page 46, 2005.

[145] 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):1–15, 2017.

[146] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the 10th International Conference on Machine Learning*, ICML'93, pages 330–337. Morgan Kaufmann Publishers Inc., 1993.

[147] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the DARPA grand challenge. *Journal of Field Robotics*, 23(9):661–692, 2006.

[148] Michael Tomasello, Alicia P. Melis, Claudio Tennie, Emily Wyman, and Esther Herrmann. Two key steps in the evolution of human cooperation: The interdependence hypothesis. *Current Anthropology*, 53(6):673–692, 2012.

[149] John Tsitsiklis and Benjamin Van Roy. An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690, 1997.

[150] Kagan Tumer and Adrian Agogino. Distributed agent-based air traffic flow management. In *Proceedings of the 6th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'07, pages 1–8. Association for Computing Machinery, 2007.

[151] Elise Van der Pol and Frans A. Oliehoek. Coordinated deep reinforcement learners for traffic light control. In *NIPS'16 Workshop on Learning, Inference and Control of Multi-Agent Systems*, NIPS'16, 2016.

[152] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. *arXiv*, abs/1509.06461:1–13, 2015.

[153] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 31*, NIPS'17, pages 5998–6008. Curran Associates, Inc., 2017.

[154] Matej Vecerík, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin A. Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv*, abs/1707.08817:1–10, 2017.

[155] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander (Sasha) Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John P. Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy P. Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. StarCraft II: A new challenge for reinforcement learning. *arXiv*, abs/1708.04782:1–20, 2017.

[156] Oriol Vinyals, Igor Babuschkin, Wojciech Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John Agapiou, Max Jaderberg, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[157] Jianhao Wang, Zhizhou Ren, Beining Han, and Chongjie Zhang. Towards understanding linear value decomposition in cooperative multi-agent Q-learning. *arXiv*, abs/2006.00587:1–39, 2020.

[158] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex dueling multi-agent Q-learning. *arXiv*, abs/2008.01062:1–27, 2020.

[159] Jianhong Wang, Yuan Zhang, Tae-Kyun Kim, and Yunjie Gu. Shapley Q-value: A local reward approach to solve global reward games. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, AAAI'20, pages 7285–7292. AAAI Press, 2020.

[160] Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Off-policy multi-agent decomposed policy gradients. *arXiv*, abs/2007.12322: 1–20, 2020.

[161] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. *arXiv*, abs/1511.06581:1–15, 2015.

[162] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8 (3):279–292, 1992.

[163] Ermo Wei and Sean Luke. Lenient learning in independent-learner stochastic cooperative games. *Journal of Machine Learning Research*, 17(84):1–42, 2016.

[164] Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[165] Shimon Whiteson, Matthew E. Taylor, and Peter Stone. Adaptive tile coding for value function approximation. Technical Report AI-TR-07-339, University of Texas at Austin, 2007.

[166] Daan Wierstra, E. Foerster, Jan Peters, and Juergen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *Proceedings of the 16th International Conference on Artificial Neural Networks*, ICANN'07, pages 697–706. Springer, 2007.

[167] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[168] David H. Wolpert and Kagan Tumer. An introduction to collective intelligence. Technical report, NASA-ARC-IC-99-63, Nasa Ames Research Center, 1999.

[169] David H. Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. *Advances in Complex Systems*, 4(2-3):265–279, 2001.

[170] Zifan Wu, Chao Yu, Deheng Ye, Junge Zhang, Haiyin Piao, and Hankz Hankui Zhuo. Coordinated proximal policy optimization. *arXiv*, abs/2111.04051:1–20, 2021.

[171] Michael Wunder, Michael L. Littman, and Monica Babes. Classes of multiagent Q-learning dynamics with epsilon-greedy exploration. In *Proceedings of the 27th International Conference on Machine Learning*, ICML'10, pages 1167–1174. Omnipress, 2010.

[172] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv*, abs/1505.00853:1–5, 2015.

[173] Chuanyu Yang, Kai Yuan, Qiuguo Zhu, Wanming Yu, and Zhibin Li. Multi-expert learning of adaptive legged locomotion. *Science Robotics*, 5(49):14 pages, 2020.

[174] Yaodong Yang, Jianye Hao, Guangyong Chen, Hongyao Tang, Yingfeng Chen, Yujing Hu, Changjie Fan, and Zhongyu Wei. Q-value path decomposition for deep multiagent reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20, pages 10706–10715. PMLR, 2020.

[175] Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A general framework for cooperative multiagent reinforcement learning. *arXiv*, abs/2002.03939:1–14, 2020.

[176] Yuxiang Yang, Ken Caluwaerts, Atil Iscen, Tingnan Zhang, Jie Tan, and Vikas Sindhwani. Data efficient reinforcement learning for legged robots. In *Proceedings of the 3rd Annual Conference on Robot Learning*, CoRL'19, pages 1–10. PMLR, 2019.

[177] Dayon Ye, Minji Zhang, and Yu Yang. A multi-agent framework for packet routing in wireless sensor networks. *Sensors*, 15(5):10026–10047, 2015.

[178] Logan Yliniemi and Kagan Tumer. Multi-objective multiagent credit assignment through difference rewards in reinforcement learning. In *Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning*, SEAL'14, pages 407–418. Springer, 2014.

[179] Chao Yu, Xin Wang, Jianye Hao, and Zhanbo Feng. Reinforcement learning for cooperative overtaking. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS'19, pages 341–349. International Foundation for Autonomous Agents and MultiAgent Systems, 2019.

[180] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre M. Bayen, and Yi Wu. The surprising effectiveness of MAPPO in cooperative, multi-agent games. *arXiv*, abs/2103.01955:1–29, 2021.

[181] Yan Zhang and Michael M. Zavlanos. Distributed off-policy actor-critic reinforcement learning with policy consensus. *arXiv*, abs/1903.09255:1–8, 2019.

[182] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems 34*, NeurIPS'20, pages 11853–11864. Curran Associates, Inc., 2020.