Exploration and Value Function Factorisation in Single and Multi-Agent Reinforcement Learning

Tabish Rashid

Keble College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2021

Abstract

The ability to learn from data is crucial in developing satisfactory solutions to many complex problems. In particular, in the design of *intelligent* agents that exist and interact with a complex environment in the pursuit of some goal. In this thesis we investigate some bottlenecks that can prevent such agents from learning and making progress in their respective environments. To do so we adopt the framework of Reinforcement Learning (RL), specifically Deep RL in which deep neural networks are used to learn and represent functions and quantities of interest.

The first bottleneck we investigate, is the difficulty of exploration in Part I. An agent must first explore their environment, by interacting with it to observe the consequences of its actions in a variety of situations, in order to gather enough data to infer good behaviour from. This is of particular importance in *sparse reward* environments, in which significant exploration is often required before any meaningful signal related to its goal is received by the agent. This is the first bottleneck we investigate in Chapter 3. Specifically, how can we incorporate the principle of *optimism in the face of uncertainty* that is prevalent in the tabular literature, into our Deep RL algorithms.

In Part II we then investigate the difficulties of learning as a team of agents. Although exploration is still of paramount importance in this multi-agent setting, we must first be able to confidently solve simpler problems in which we are not limited by our ability to explore. We consider the fully cooperative multi-agent setting, in which we wish to train a team of agents to solve a shared task. In Chapter 4 we explore a line of research into value function factorisation, which can allow for more efficient learning by considering the value function of the team as a whole in a tractable manner. We then investigate the consequences of our factorisation in Chapter 5, focussing on the negative implications they can have on learnability and exploration. To remedy these shortcomings we propose the use of a weighting in our optimisation objective. We then investigate the utility of this weighting beyond its original motivations, and identify new bottlenecks that arise in these improved approaches.

Finally, in Part III we consider problems in which there isn't a well-defined objective. This is the case in games such as Chess or Go, in which the overall goal is to learn to play as well as possible. However, this goal does not explicitly specify who we should benchmark against, nor which opponents we should train against. In these settings, we can make progress by proposing our own appropriate opponent. It is this step, of proposing an *appropriate* opponent that we investigate further. In Chapter 6 we propose a Bayesian algorithm that can incorporate any prior knowledge we might have, and that will take advantage of the structure of the problem in order to efficiently explore the relevant quantities. Thereby drastically reducing the time and computation required to propose such an opponent.

Exploration and Value Function Factorisation in Single and Multi-Agent Reinforcement Learning



Tabish Rashid

Keble College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2021

Acknowledgements

I am incredibly thankful to my supervisor Shimon Whiteson for all of his help, support, and guidance throughout my DPhil. Thank you for taking a chance on me. A big thank you to Michael Osborne and Christopher Amato for examining this thesis, the enjoyable discussion we had, and all of the constructive feedback I received.

I also want to thank all of my colleagues and friends from Oxford, and in particular the WhiRL lab for providing a fantastic environment for research. These include, but are not limited to: Bei Peng, Wendelin Boehmer, Mikayel Samvelyan, Gregory Farquhar, Christian Schroeder de Witt, Jakob Foerster, Anuj Mahajan, Max Igl, Luisa Zintgraf, Matt Fellows, Supratik Paul, Yannis Assael, Shangtong Zhang, and Kristian Hartikainen.

I would also like to thank everyone at Keble College, the JCR during my undergraduate studies and the MCR during my graduate studies, for making my time at Oxford so enjoyable. I am deeply grateful to Stephen Cameron, who is no longer with us, for all of his support and encouragement when I wanted to pursue a PhD.

A big thank you to Jason Nurse and Ioannis Agrafiotis for sparking my passion for research, as well as their support and encouragement in helping me pursue a PhD. Thank you to Varun Kanade for taking the time to help with my understanding of ML and also for his support in my PhD applications.

A big thank you to everyone at Microsoft Research Cambridge for all of their help during my internship: Katja Hoffman, Sam Devlin, Robert Loftin, Jaroslaw Rzepecki, Adrian O'Grady, Cheng Zhang, and Hisham Hussain. In particular, a big thank you to Kamil Ciosek for all of his work, patience, and support during the project.

An enormous thank you to Yuki Uchida for her continuous support and encouragement throughout my DPhil. I wouldn't have managed to keep going and not give up on my research without her.

Finally, I cannot thank my family enough for their constant support and patience with me over the years. To my Sister, Mother and dearly departed Grandmother for their endless love and encouraging me to pursue research.

Abstract

The ability to learn from data is crucial in developing satisfactory solutions to many complex problems. In particular, in the design of *intelligent* agents that exist and interact with a complex environment in the pursuit of some goal. In this thesis we investigate some bottlenecks that can prevent such agents from learning and making progress in their respective environments. To do so we adopt the framework of Reinforcement Learning (RL), specifically Deep RL in which deep neural networks are used to learn and represent functions and quantities of interest.

The first bottleneck we investigate, is the difficulty of exploration in Part I. An agent must first explore their environment, by interacting with it to observe the consequences of its actions in a variety of situations, in order to gather enough data to infer good behaviour from. This is of particular importance in *sparse reward* environments, in which significant exploration is often required before any meaningful signal related to its goal is received by the agent. This is the first bottleneck we investigate in Chapter 3. Specifically, how can we incorporate the principle of *optimism in the face of uncertainty* that is prevalent in the tabular literature, into our Deep RL algorithms.

In Part II we then investigate the difficulties of learning as a team of agents. Although exploration is still of paramount importance in this multi-agent setting, we must first be able to confidently solve simpler problems in which we are not limited by our ability to explore. We consider the fully cooperative multi-agent setting, in which we wish to train a team of agents to solve a shared task. In Chapter 4 we explore a line of research into value function factorisation, which can allow for more efficient learning by considering the value function of the team as a whole in a tractable manner. We then investigate the consequences of our factorisation in Chapter 5, focussing on the negative implications they can have on learnability and exploration. To remedy these shortcomings we propose the use of a weighting in our optimisation objective. We then investigate the utility of this weighting beyond its original motivations, and identify new bottlenecks that arise in these improved approaches.

Finally, in Part III we consider problems in which there isn't a well-defined objective. This is the case in games such as Chess or Go, in which the overall goal is to learn to play as well as possible. However, this goal does not explicitly

specify who we should benchmark against, nor which opponents we should train against. In these settings, we can make progress by proposing our own appropriate opponent. It is this step, of proposing an *appropriate* opponent that we investigate further. In Chapter 6 we propose a Bayesian algorithm that can incorporate any prior knowledge we might have, and that will take advantage of the structure of the problem in order to efficiently explore the relevant quantities. Thereby drastically reducing the time and computation required to propose such an opponent.

Contents

1 Introduction **1**
1.1 Motivations 1
1.2 Challenges 7
1.3 Structure and Contributions 10

2 Background **14**
2.1 Reinforcement Learning 14
2.1.1 Problem Setting 15
2.1.2 Algorithms 18
2.2 Deep Reinforcement Learning 20
2.2.1 DQN 20

I Single-Agent Exploration **23**

3 Exploration **26**
3.1 Introduction 27
3.2 Tabular Reinforcement Learning Background 29
3.3 Optimistic Pessimistically Initialised Q-Learning 31
3.3.1 Motivations 31
3.4 Tabular Reinforcement Learning Results 34
3.4.1 Proof of Main Theorem 35
3.4.2 Necessity for Optimism During Action Selection 46
3.4.3 Necessity for Intrinsic Motivation Bonus 47
3.4.4 Necessity for Optimism During Bootstrapping 49
3.5 Deep Reinforcement Learning Background 49
3.5.1 Counting in Large, Complex State Spaces 50
3.5.2 Granularity of the counting model 51
3.6 Deep Reinforcement Learning OPIQ 51
3.7 Related Work 53
3.7.1 Tabular Reinforcement Learning 53
3.7.2 Deep Reinforcement Learning 58
3.8 Experimental Setup 63

3.8.1 Environments . . . 64
3.8.2 Ablations and Baselines . . . 66
3.9 Results . . . 67
3.9.1 Randomised Chain . . . 67
3.9.2 Maze . . . 69
3.9.3 Montezuma's Revenge . . . 71
3.10 Conclusions . . . 73

II Cooperative Multi-Agent Reinforcement Learning 75

4 Value Function Factorisation 77
4.1 Introduction . . . 78
4.2 Multi-Agent Background . . . 81
4.2.1 Dec-POMDPs . . . 82
4.2.2 Independent Q-Learning . . . 83
4.2.3 Value Decomposition Networks . . . 83
4.3 QMIX . . . 84
4.3.1 Representational Complexity . . . 90
4.4 Two-Step Game . . . 92
4.5 Random Matrix Games . . . 93
4.6 The StarCraft Multi-Agent Challenge . . . 93
4.6.1 Scenarios . . . 97
4.6.2 State and Observations . . . 97
4.6.3 Action Space . . . 99
4.6.4 Rewards . . . 99
4.6.5 Evaluation Methodology . . . 100
4.6.6 Evaluation Metrics . . . 100
4.7 PyMARL . . . 101
4.8 Related Work . . . 102
4.8.1 Independent Learners . . . 102
4.8.2 Centralised Execution . . . 103
4.8.3 Centralised Training and Decentralised Execution . . . 105
4.8.4 Extensions of QMIX . . . 109
4.8.5 PyMARL and SMAC . . . 111
4.8.6 Hypernetworks and Monotonic Networks . . . 112
4.9 SMAC Results . . . 112
4.10 Analysis . . . 115
4.10.1 Role of the Central State . . . 116
4.10.2 Role of Nonlinear Mixing . . . 117

4.10.3 Role of RNN in Agent Network . . . 123
4.10.4 Role of the COMA Critic Architecture . . . 123
4.10.5 Role of Network Size and Depth . . . 124
4.11 Conclusion . . . 125

5 Weighted Value Function Factorisation 127
5.1 Introduction . . . 128
5.2 Multi-Agent MDP Background . . . 129
5.3 QMIX Operator . . . 130
5.3.1 Properties of $\mathcal{T}^*_{\text{Qmix}}$ . . . 132
5.4 Weighted QMIX Operator . . . 133
5.4.1 Weightings . . . 134
5.4.2 Weighted QMIX Operators . . . 136
5.5 Proof of Theoretical Results . . . 137
5.6 Deep Reinforcement Learning Algorithm . . . 147
5.7 Related Algorithms . . . 149
5.7.1 Relationship to Actor-Critic methods . . . 149
5.7.2 Relationship To QTRAN . . . 150
5.7.3 QPLEX . . . 152
5.8 Related Work . . . 153
5.9 Experimental Results . . . 154
5.9.1 Ablations . . . 154
5.9.2 Predator Prey . . . 154
5.9.3 SMAC . . . 156
5.10 Utility of a Weighting . . . 159
5.10.1 Motivations For a Weighting . . . 160
5.10.2 Implicit vs Explicit Weighting . . . 161
5.10.3 Parametrisation . . . 165
5.10.4 Contextual Matrix Game Results . . . 167
5.10.5 Applications of a Weighting Beyond QMIX . . . 173
5.11 Conclusions and Future Work . . . 178

III Empirical Game Theory 181

6 Empirical Game Theory 184
6.1 Introduction . . . 185
6.2 Background . . . 186
6.2.1 Games and α-Rank . . . 186
6.2.2 Single Population α-Rank . . . 187

6.2.3 Worked Example . . . 188
6.2.4 Wasserstein Distance . . . 189
6.3 Related Work . . . 190
6.3.1 ResponseGraphUCB . . . 191
6.4 αIG and αWass . . . 192
6.4.1 Payoffs: Ground Truth and Belief . . . 193
6.4.2 Belief over α-ranks . . . 194
6.4.3 Picking Payoffs to Query . . . 194
6.4.4 Implementation . . . 195
6.4.5 Computational Requirements . . . 196
6.4.6 Query Selection by Maximizing Wasserstein Divergence . . . 197
6.5 Theoretical Results . . . 198
6.6 Proofs . . . 200
6.6.1 Regret Bound For Policy Maximizing Information Gain on Payoffs . . . 202
6.6.2 Regret Bound For Policy Maximizing Information Gain on α-Ranks . . . 203
6.7 Experiments . . . 207
6.7.1 Good-Bad Games . . . 208
6.7.2 Incorporating Prior Knowledge . . . 215
6.7.3 Gaussian Games . . . 216
6.7.4 Conditioning of the belief distribution . . . 217
6.8 Conclusions . . . 218

7 Afterword **219**

Appendices

A Single-agent Exploration **222**
A.1 Experimental Setup . . . 222
A.1.1 Hyperparameters and Network architectures . . . 222
A.1.2 Baselines training details . . . 226

B Value Function Factorisation **228**
B.1 Two Step Game . . . 228
B.1.1 Architecture and Training . . . 228
B.1.2 Learned Value Functions . . . 228
B.1.3 Results . . . 229
B.2 Random Matrix Games . . . 230
B.2.1 Architecture and Training . . . 230

B.3 SMAC 230
B.3.1 Scenarios 230
B.3.2 Environment Setting 231
B.3.3 Architecture and Training 232
B.3.4 Table of Results 234
B.4 Regression Experiment 235

C Weighted Value Function Factorisation 236
C.1 Experimental Setup 236
C.1.1 Predator Prey 237
C.1.2 SMAC Robustness to exploration 237
C.1.3 SMAC Super Hard Maps 238
C.2 Ablations 238

D Importance of Weighted Value Function Factorisation 239
D.1 Simplified Predator Prey 239
D.1.1 Environment 239
D.1.2 Weighting 240
D.2 Contextual Matrix Games 240
D.2.1 Distribution of Errors 240

E Empirical Game Theory 241
E.1 Experimental Setup 241
E.1.1 Implementation Details 241
E.1.2 Baselines 242
E.1.3 Graphs 242
E.1.4 Environments 243
E.1.5 2 Good, 2 Bad 243
E.1.6 3 Good, 5 Bad 243
E.1.7 4x4 Gaussian 244
E.1.8 3 Good, 5 Bad Incorporating Prior Knowledge 244

References 246

1 Introduction

Contents

1.1 Motivations 1
1.2 Challenges 7
1.3 Structure and Contributions 10

1.1 Motivations

Computers are ubiquitous in the modern world. Not just as programmable general purpose computing devices, but increasingly as a part of everyday objects such as fridges, doorbells and washing machines. The pace of advancement in this area has been unprecedented in human history [1]. It is relatively uncontroversial to say that the advancements in raw computational power have largely met the expectations and hopes that early pioneers had. A common quip used to underline this is that a modern smartphone (or even an old smartphone relative to 2021) has more processing power than was used by NASA to land on the moon in 1969 [2]. But more than that, it is relatively common to own such a computing device, with over half of the world's population being estimated as being active internet users [3]. Additionally, for a large chunk of the world's population, particularly in affluent regions, it is not uncommon to own multiple devices that would top

the supercomputing lists in the 1990s [4]. Although continued progress in the raw computation of general purpose CPUs is uncertain [5, 6], we are likely to still enjoy the commoditisation of specialised computing hardware such as GPUs and TPUs [7] in the near future which provides a convenient path for harnessing more computation for specific applications.

Alongside this rapid progress on the hardware side of things, it was only natural to have the same lofty expectations for the software that would run on them. If the raw computational power we have available to us in our phones is equivalent to a supercomputer 20 years ago, surely we must be able to harness that to extraordinary affect. Indeed, many sci-fi stories envisioned a future in which intelligent machines would become commonplace. At the very least, such intelligent machines would be expected to be able to comfortably undertake many tasks such as identifying objects in their environment, playing games, or holding a basic conversation. But if we look around the modern world (circa 2021), such intelligent machines are not commonplace and do not even exist. A natural question to ask is then: *What has prevented us from designing and implementing such intelligent machines?* Therein lies the underlying motivations for the work in this thesis. We wish to make progress towards this goal of intelligent agents, and so we must improve the software to do so.

Part of the difficulty in making progress in this endeavour, is down to the fundamentals of computability. The theory on what is computable, and in particular on how algorithms scale is often a roadblock in making progress on many difficult problems. Problems that belong to *hard* computational classes such as NP, include many interesting problems such as boolean satisfiability, finding cliques in graphs, integer programming, and the travelling salesman problem [8]. They are interesting because they naturally arise in many different situations and other complex problems can be shown to be equivalent to solving one of these. Other computationally challenging classes that are more relevant for the topics investigated in this thesis include PPAD (the computation of a Nash equilibrium in general sum games [9]) and EXPTIME-Complete (calculating the next best move in a game of Chess [10]). Importantly, these theoretical results tell us that, barring a breakthrough result in

Computer Science such as $P = NP$, we cannot design *exact efficient* algorithms for these problems. Furthermore, even if raw computation scales as Moore's Law predicts (by doubling roughly every 18 months), it would not meaningfully affect the size of the problem we can *exactly* solve. Consider for example an algorithm which scales exponentially in the size of its input, an *inefficient* algorithm whose time complexity is $O(2^n)$. If we double the amount of computation we can accomplish within a given time period, which we stress should not be taken lightly, all we can do is increase the size of the problem solvable in the same time period by 1.

Although this paints a bleak picture in terms of what useful computation we can accomplish, in practise we can often resort to smart heuristics or shrinking our domain in order to make useful progress on problems we are interested in. For example, modern SAT solvers are commonly used in industrial applications despite the problem being NP-hard [11]. However, there are still huge difficulties faced in solving or even approximating solutions to other tasks which seem trivial to a human. In 1966 Marvin Minsky famously, possibly apocryphally, estimated that a computer vision project involving foreground-background segmentation would take a team of undergraduates a single summer to solve [12]. Although this task can be easily solved by a human, under a wide range of challenging conditions, decades of computer vision research were required before significant progress was made. The overall point we are aiming to make here, is that there are fundamental limits on the sorts of problems we are able to solve *exactly*. These hard problems, whilst not necessarily offering a full solution to intelligent agents, can (and do) arise as sub-problems in many AI related algorithms. Thus, we must resort to approximations or heuristics in order to increase the scale of the problems we are tractably able to solve. Moreover, even if we resign ourselves to approxiations (and lose any guarantees of an optimal or correct solution) there are still immense difficulties involved. Of the sorts of problems we might be interested in intelligent machines solving, such as those involving images or natural language, it has proved to be incredibly difficult to devise solutions to such tasks.

The field of Artificial Intelligence (AI) is an incredibly broad one, started in its modern incarnation with the advent of programmable computers, with the ambitions of solving complex problems of the sort we have discussed. The bible for AI is often seen as [13], which discusses both the history, philosophy, and techniques under the umbrella term of AI in great detail. The book encompasses a wide variety of approaches that Machine Learning practitioners often now refer to as *traditional* AI techniques. These include algorithms and heuristics for searching in a graph, data structures for representing and manipulating data, logical reasoning systems, and more. Famously DeepBlue [14] used many of these techniques to defeat Garry Kasparov, the World Chess Champion at the time. Whilst it was hailed as a landmark achievement for AI, and represented the first time a machine had successfully beaten a human world champion, it did not herald the introduction of intelligent machines. Instead, many were quick to point out that Deep Blue was not intelligent at all since it did not exhibit *understanding* [15] and was simply a smart variant of a brute-force search.

Learning

In the years after DeepBlue's victory in Chess, Go was touted as the next big challenge for game-playing AI that would require *true* intelligence to solve. This was because the approach taken by DeepBlue for playing Chess would not scale to the game of Go, since Go features an order of magnitude more possible moves at each timestep. And so, the amount of computation that would be required to search a sufficient number of moves ahead in Go in the same manner as DeepBlue, posed a significant bottleneck. Instead, in its victory over Lee Sedol, AlphaGo [16] *learned* appropriate heuristics in order to guide its search procedure much more efficiently than DeepBlue. The ability to learn from data has been incredibly successful in the last decade, and has enabled practitioners to tackle increasingly difficult problems. Both those in which it was incredibly difficult to devise solutions, such as Natural Language Processing [17], and those in which the traditional approaches or hand-crafted heuristics were commonly used such as chip floorplanning [18].

Importantly, these learning-based approaches have the potential to harness the extra computation available to great affect. E.g. Doubling the compute available for training would often result in a significant increase in performance, or a significant difference in the approaches ability to tackle larger problems.

In previous approaches we have discussed earlier, the algorithms and solutions are hard-coded in the sense that they will not change their rules or execution after observing data. In contrast, learning based approaches attempt to effectively learn or update their rules from the data they observe. What we mean by this, is that an approach like DeepBlue will utilise the exact same heuristics in its search until they are reprogrammed by a practitioner. Whereas an approach like AlphaGo can update its heuristics (to utilise the same terminology as DeepBlue) during training based on the data it is observing without a human explicitly reprogramming them (or even being able to interpret them). Machine Learning and Pattern Recognition [19] are areas that fall under the umbrella term of AI that aim to do exactly this. Even if we as the programmers are unable to accurately specify and implement rules and procedures for these problems, it does not mean that there isn't underlying structure in the task that can be learnt and exploited. Natural language processing is one such example. It is extremely difficult to design algorithms that can simulate natural language through rule-based approaches rooted in linguistics. Frederick Jelinek famously remarked (again apocryphally) that "Every time I fire a linguist, the performance of the speech recognizer goes up" [20]. Instead, huge progress was made in attempting to leverage the data available and *learn* from it instead. This is particularly appealing since there is a huge amount of data (and work undertaken to increase that) available for common natural language tasks. For instance there is a large amount of books that have been translated between different languages that could serve as the data for a learning based approach aiming to do translation. There is also almost an unlimited amount of textual data (most of which is perhaps not useful) that can be used to train language models that wish to learn the distribution of natural language, through predicting the next word or character in a sentence for example [17]. These approaches which learn from data,

are now firmly the most successful in the field of natural language processing and are widely used in online translation services. Similarly for computer vision, Machine Learning based approaches are now the leading algorithms and an indispensable component of all modern approaches. AlexNet [21] winning ImageNet in 2012 was a watershed movement for supervised learning in general, because it showed the vast potential for learning from data using deep neural networks and brought significant attention to the paradigm of training on GPUs.

Although these kind of Machine Learning models are now quite commonplace in the modern world, they fall short in being able to capture intelligence in the manner most humans would expect of *intelligent* agents. For instance an agent that can **only** correctly identify all animals in an image would not be described as intelligent. This is because we would expect intelligent agents to exist, and continually act, in an environment of some sort. In addition, we would like the agent to have some notion of a goal or desired behaviour. This is because we cannot think of a stationary agent that simply does nothing as being intelligent. This kind of sequential decision making problem cannot be sensibly modelled in the framework of supervised learning. And so, at a fundamental level, supervised learning alone is not a suitable paradigm or framework in which to model intelligent agents of this kind. Instead, it is for these reasons that we look towards Reinforcement Learning (RL).

RL is an extremely general framework that is commonly used to model agents. In RL, an agent interacts with an environment by taking actions based on what information it observes. Crucially, in this setup an RL agent receives a reward signal which then forms its overall goal: maximise the total return an agent receives. This is an extremely general framework, which can model a huge variety of problems and has been argued is all that is required for modelling an intelligent agent [22]. For the ability to model sequential decision making tasks and its general applicability, we will adopt the framework of Reinforcement Learning in this thesis and aim to investigate the limitations and solutions to the bottlenecks that arise.

1.2 Challenges

As just discussed, Reinforcement Learning is an attractive framework for modelling complex sequential decision making problems. What we have not yet discussed, is how RL algorithms aim to tackle these problems. Without delving into the specifics of RL algorithms, or their terminology, we can safely assume that we will need to represent (and learn) a function of some kind. This function will need to map from the input data the agent receives (states, observations or even the entire history of all actions and observations) to some quantity that will be used to determine how the agent should interact with the environment (such as which action the agent should take). If our input data is discrete and can only be a small number of possible values, we can easily represent our function using a simple table. In this representation each possible value of the input data is mapped to the appropriate values we require for our algorithm. This is commonly referred to as tabular RL, wherein you represent the quantities of interest in a table. This immediately presents a problem for environments in which the input data is not so simple. If our data can take a large number of possible values, or is continuous rather than discrete, then representing our function in this manner quickly becomes impractical or impossible. In order to represent such a function, we need to appeal to methods of function approximation. Fortunately, Machine Learning presents many such methods of which the Deep Learning based neural networks are currently the most flexible and the most popular. This leads us into the area of *Deep* Reinforcement Learning, in which we approximate our quantities of interest using (deep) neural networks. Deep RL has enjoyed many successes, the first arguably being playing Backgammon at close to the level of top human players [23]. More recently advances have occurred at a much more rapid pace, some notable accomplishments include learning to play Atari games [24], beating a world champion at Go [16], and playing Dota 2 [25] and StarCraft 2 [26] at a professional level.

Of fundamental concern in Reinforcement Learning (and by extension Deep RL) is the trade-off between exploration and exploitation [27]. In order for an agent to learn good behaviour, it must be exposed to the appropriate data through the

process of *exploring*. However, if an agent always aims to take new actions, it is not *exploiting* the data it has already seen in order to receive as much reward as possible in the long-term. For example, suppose an agent receives a small positive reward in a particular state. Should the agent aim to continually return and stay in this state to receive this reward and gather as much data as possible close to it? Or should they continue to explore their environment, receiving significantly less reward in doing so and observing significantly less data near that state of interest, in the hopes of finding a better rewarding state elsewhere? To balance this trade-off, our agents must explore their environment *appropriately* in order to observe the necessary data for them to learn good behaviour, where we have pushed all of the hardness into the ill-defined adjective *appropriately*. However, even where then this trade-off does not exist, exploration is still hard. An RL agent cannot hope to learn tabula rasa that reaching the goal square of a maze is good if it has never visited such a square before, and if every other square in the maze provides no reward. How can an agent explore its environment effectively then, without some kind of signal helping it do so? Fortunately, there is a rich vein of literature dedicated to answering this question in the tabular RL setting, which even formally proves that some approaches are *efficient* [28, 29, 30]. But as we discussed earlier, it is often not practical to use a tabular representation, and instead it is highly desired we use something like a neural network. This introduces many complications, since functions represented by neural networks behave very differently to functions represented by a table, especially when we are constantly updating and training them. This means that many of the properties that are crucial to these tabular algorithms, that are essential for proving efficiency, do not hold in the Deep RL setting. In Chapter 3 of this thesis we explore some of the specific difficulties that arise, and devise an approach to recover some of their key benefits even when using neural networks as our function approximators.

Until now, we have only considered situations in which we have a single agent that wishes to act in their environment. However, many situations that arise in the real-world can be more naturally or appropriately modelled as multi-agent systems.

Sports, self-driving cars, Chess, Go, StarCraft, auctions, negotiations and the control of energy grids to name but a few. In these systems, there are multiple agents that can interact with each other within the same environment. In Part II of this thesis we restrict our focus to fully cooperative scenarios, in which a team of agents must learn how to act in order to maximise their shared team return. Here, a critical issue is how can the agents learn to *coordinate* their actions and behaviours in order to maximise the team's long-term rewards. Although all of the previously mentioned problems surrounding exploration remain, we must first learn how to efficiently take advantage of any data that we do observe, before we can begin to think about how to efficiently generate or seek out useful data. In this multi-agent setting there are many additional complications that arise. In particular, simply treating the task as a single-agent problem for each agent separately is often not sufficient to learn coordination. In Chapter 4l we investigate how we can devise an algorithm that can learn to effectively coordinate. Instead of treating each agent separately, our approach instead attempts to operate at the level of the team, whilst remaining both powerful and tractable. Then in Chapter 5 we dig deeper into the properties of our proposed algorithm. In particular, its inability to learn in seemingly ideal scenarios and the implications this can have on both exploration and its ability to learn in a wide variety of scenarios. We appeal to a simplified theoretical analysis in order to help give us intuition about these bottlenecks, and then propose an approach to remedy these issues by introducing a weighting into our learning objective. We then conduct a preliminary investigation into the potential benefits of our weighting in relation to alternate approaches in the literature. Focussing on how it can be extended to other methods, in which its original motivations don't apply, potentially alleviating some practical shortcomings of these alternate approaches we observe.

Of course, there are many other such multi-agent settings beyond the specific fully cooperative setting we investigate in Part II. Most famously 2 player zero-sum games, such as Chess or Go. However, we will not investigate Deep RL methods for such scenarios nor for any non-cooperative scenarios in this thesis. We will instead investigate how RL approaches can be applied to such problems. In many

of these situations, it is difficult to fully specify the environment or task we wish to solve. Take for example the task of playing Go for instance. Without specifying an opponent, we cannot directly frame this as an RL environment. Nonetheless, we still wish to make progress towards this goal. Fortunately, we can specify the opponents ourselves for the purposes of learning. This gives rise to an Open-Ended Learning regime, whereby we propose our own opponents and aim to beat them by utilising techniques from Deep RL. A popular approach which has led to many successes, is to utilise self-play techniques [31, 32] in which you are proposing opponents based on previous versions of the current agent. In this setup we often consider the *meta-game*, which is the matrix game of payoffs for each player playing against another. An important step in these self-play algorithms (more specifically in the framework of PSRO [32]), is to estimate some quantity of this meta-game. However, we do not know the meta-game exactly. Instead, we can only draw samples from it by evaluating the agents against one another. This is the setting of Empirical Game Theory that we consider in Part III.Specifically, we consider this task of estimating the α-Rank [33] of the meta-game and propose an approach that aims to approximate it as efficiently as possible. This is extremely important, because in many of these self-play systems there is a considerable computational cost devoted to evaluating the meta-game accurately. Especially in scenarios involving more than 2 agents, of which α-Rank can be applied. By efficiently picking which samples to draw (i.e. how to explore the meta-game efficiently), we can reduce this computational burden and speed up training a significant amount.

1.3 Structure and Contributions

We now give an overview of the structure of this thesis and my personal contributions towards the work undertaken. For the published papers on which the content of this thesis is based, unless otherwise mentioned I assumed the role of lead author. Concretely, this meant implementing and running all experiments, producing the theoretical proofs, guiding the research direction of the project, and taking the lead on the writing and presentation of the paper. All of this research would not have

come to fruition without the combined efforts of all co-authors. I am extremely grateful to all my co-authors and colleagues whose contribution and support were pivotal in producing and polishing all of the work contained in this thesis.

Background

We first provide a brief background on pertinent concepts that are relevant throughout the thesis in Chapter 2. Each chapter will additionally include more relevant background material and notation specific for the topics discussed in that chapter.

Single-Agent Exploration

Part I focusses on exploration in the single-agent Deep RL setting. Chapter 3 introduces OPIQ, which attempts to incorporate the notion of *optimism in the face of uncertainty* into the Deep RL setting. OPIQ aims to do so by mimicking the benefits of an optimistic initialisation even when our neural network based DQN do not maintain such useful properties. We also draw links towards the tabular literature, and show that our approach can be seen as an extension of the provably efficient UCB-H approach from the tabular setting to the Deep RL setting. It is based on:

- "Optimistic Exploration even with a Pessimistic Initialisation",
  Tabish Rashid, Bei Peng, Wendelin Boehmer, Shimon Whiteson.
  In International Conference on Learning Representations (ICLR) 2020.

Cooperative Multi-Agent Reinforcement Learning

Part II then shifts focus to the cooperative multi-agent setting under the paradigm of centralised training and decentralised execution. Chapter 4 introduces QMIX, and the SMAC benchmark that we use for experimental comparisons. It is based on the following where * indicates equal contribution:

- "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning",
  Tabish Rashid*, Mikayel Samvelyan*, Christian Schroeder de Witt, Gregory

Farquhar, Jakob Foerster, Shimon Whiteson.
In International Conference on Machine Learning (ICML) 2018.

- "The StarCraft Multi-Agent Challenge",
  Mikayel Samvelyan*, Tabish Rashid*, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philip H. S. Torr, Jakob Foerster, Shimon Whiteson.
  In International Conference on Autonomous Agents and Multiagent Systems (AAMAS) 2019 as an Extended Abstract, and at NeurIPS Deep Reinforcement Learning Workshop (2019).

These were later extended into a journal article:

- "Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning",
  Tabish Rashid*, Mikayel Samvelyan*, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, Shimon Whiteson.
  In Journal of Machine Learning Research (JMLR) Volume 21 (2020).

For these 3 papers, I contributed to the implementation and debugging of the codebase, both the original codebase for the ICML 2018 paper and PyMARL which was used for all subsequent work. I ran all experiments, conducted the analysis of QMIX that featured in the JMLR article, and contributed significantly to the writing and presentation of the content.

We then investigate the shortcomings of QMIX in Chapter 5, and propose the use of a weighting in our loss function to alleviate some of them, based on:

- "Weighted QMIX: Expanding Monotonic Value Function Factorisation",
  Tabish Rashid, Gregory Farquhar, Bei Peng, Shimon Whiteson.
  In Advances in Neural Information Processing Systems (NeurIPS) 2020.

Chapter 5 also features unpublished work in which we conduct a preliminary investigation into the additional benefits of the weighting, beyond its original motivations and explore its utility in combination with other approaches.

Empirical Game Theory

Part III then explores the Empirical Game Theory setting, with a focus on α-Rank as the solution concept of choice. Chapter 6 then introduces αIG and αWass, two Bayesian approaches towards estimating the α-Rank in this setting. It is based on:

- "Estimating α-Rank by Maximizing Information Gain",
  Tabish Rashid, Cheng Zhang, Kamil Ciosek.
  In Thirty-Fifth AAAI Conference on Artificial Intelligence (AAAI 2021).

This was work undertaken during an internship at Microsoft Research Cambridge, hosted by Kamil Ciosek. The original idea for this project was conceived by Kamil, who also produced the theoretical proofs. I was responsible for everything else, including implementing and running all experiments, leading the writing and presentation of the paper, the experimental designs, visualisations and analysis.

2
Background

Contents

2.1 Reinforcement Learning 14
2.1.1 Problem Setting 15
2.1.2 Algorithms 18
2.2 Deep Reinforcement Learning 20
2.2.1 DQN 20

In this chapter we provide brief background on Reinforcement Learning and Deep Reinforcement Learning that is important in the rest of this thesis. We will focus on the single-agent setting since many of these concepts straightforwardly carry over into the multi-agent setting. More detailed and relevant background will be provided in the necessary sections of this thesis.

2.1 Reinforcement Learning

In this section we provide a brief overview of the Reinforcement Learning problem setup and popular approaches. For a more detailed description we refer the reader to [27].

2.1.1 Problem Setting

The main task in Reinforcement Learning (RL) is often framed as deciding on which action to take in an environment so as to maximise some notion of cumulative reward. More concretely, the agent is in some state, selects an action to take, which then transitions them to a new state as well as providing them with some reward. The process then repeats until the task finishes (the game the agent is playing has finished for instance). Throughout this thesis we will assume that the actions an agent can take are discrete. We will also assume we are operating in an episodic setup, whereby an agent interacts with the environment in a series of finite-length *episodes* that always terminate. Once an agent enters a terminal state the episode is over and a new one begins.

This sequential decision making problem is formally specified by a Markov Decision Process (MDP). An Episodic MDP is a 6-tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma, S_0 \rangle$, where:
$\mathcal{S}$ is the set of States.
$\mathcal{A}$ is the set of all available Actions.
P is the state transition function, $P(s'|s,a) = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$.
R is the reward function, $R(r|s,a) = \mathbb{P}[R_t = r|S_t = s, A_t = a]$.
$\gamma \in [0,1]$ is the discount factor.
$s_0 \sim S_0$ is the starting state distribution.

To avoid a dependence on the length of an episode in the definitions that follow, we assume that a *terminal* state is such that it only transitions to itself for all actions and produces 0 reward. Conceptually, the episode is over at that point. In a more practical setting, the environment will return a variable indicating that the agent entered a terminal state and that the episode has ended and the environment should be reset in some manner to start a new episode.

Importantly, an MDP enjoys the Markov property (hence the name). That is both the state transition and reward functions only depend on the current state S_t, and the current action A_t. They do not depend on the previous states or actions an agent might have taken.

We will often refer to the random variables as upper case, and their particular realisations as lower case. S_t is the random variable representing the state at timestep t and s_t is one particular realisation.

When acting in the environment the agent will produce a sequence of states, actions, and rewards referred to as a trajectory: $\tau = s_0 a_0 r_0 s_1 a_1 r_1 ....$ We will almost always assume that we do not know the state transition or reward functions, and that we can only receive samples from them by interacting in the environment. In order to get a sample from $P(\cdot|s, a)$ an agent must be at state s and take action a. We do not assume we can query the functions with arbitrary inputs.

We define the policy π as a mapping from states to probabilities over actions: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s]$. At times we will treat it as a deterministic function for simplicity.

Formally, the goal of the agent is then to maximise the expected cumulative discounted sum of rewards, the return, from a starting state:

$$J(\pi) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t].$$

In this quantity the expectation is taken over trajectories τ in which $s_0 \sim S_0, a_t \sim \pi(\cdot|s_t), r_t \sim R(\cdot|s_t, a_t), s_{t+1} \sim P(\cdot|s_t, a_t)$.

We can now define the value and state-action value functions as follows:

$$V^\pi(s) := \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s],$$

$$Q^\pi(s, a) := \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k r_{t+k}|S_t = s, A_t = a].$$

We can also define the advantage function as: $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$.

Here we have abused notation for the sake of convenience by taking the expectation over the policy π. This is shorthand for taking the expectation over the trajectory τ continuing on from s (or sa) assuming we continue to follow the policy π as before.

Both the value and state-action value functions above satisfy a recursive relationship usually known as the Bellman Equations:

$$
\begin{aligned}
V^{\pi}(s) &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k} | S_t = s] \\
&= \mathbb{E}_{\pi}[r_t + \gamma \sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k} | S_t = s] \\
&= \mathbb{E}_{a\sim\pi(\cdot|s), r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a)}[r + \gamma \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{(t+1)+k} | S_{(t+1)} = s'] | S_t = s] \\
&= \mathbb{E}_{a\sim\pi(\cdot|s), r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a)}[r + \gamma V^{\pi}(s')] \\
&= \sum_a \pi(a|s) \sum_r R(r|s,a)(r + \sum_{s'} P(s'|s,a)\gamma V^{\pi}(s')).
\end{aligned}
$$

Similarly for:

$$
\begin{aligned}
Q^{\pi}(s,a) &= \mathbb{E}_{r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a), a'\sim\pi(\cdot|s')}[r + \gamma Q^{\pi}(s',a')] \\
&= \sum_r R(r|s,a)(r + \sum_{s'} P(s'|s,a) \sum_{a'} \pi(a'|s')\gamma Q^{\pi}(s',a')).
\end{aligned}
$$

We can define an ordering over policies using the value functions: $\pi \geq \pi'$ if $V^{\pi}(s) \geq V^{\pi'}(s) \forall s \in S$. It is known that there is always a policy that is better than all others [34]. We refer to this as an optimal policy π^*, but note that it is not necessarily unique.

The optimal value functions are then: $V^*(s) := \max_{\pi} V^{\pi}$ and $Q^* := \max_{\pi} Q^{\pi}$.

These satisfy similar recursive relationships known as the Bellman Optimality Equations:

$$
\begin{aligned}
V^*(s) &= \mathbb{E}_{a\sim\pi^*(\cdot|s), r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a)}[r + \gamma V^*(s')] \\
&= \sum_a \pi^*(a|s) \sum_r R(r|s,a)(r + \sum_{s'} P(s'|s,a)\gamma V^*(s')).
\end{aligned}
$$

$$
\begin{aligned}
Q^*(s,a) &= \mathbb{E}_{r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a), a'\sim\pi^*(\cdot|s')}[r + \gamma Q^*(s',a')] \\
&= \mathbb{E}_{r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a)}[r + \gamma V^*(s')] \\
&= \mathbb{E}_{r\sim R(\cdot|s,a), s'\sim P(\cdot|s,a)}[r + \gamma \max_{a'} Q^*(s',a')] \\
&= \sum_r R(r|s,a)(r + \sum_{s'} P(s'|s,a)\gamma \max_{a'} Q^*(s',a')).
\end{aligned}
$$

Partial Observability

Often we will consider environments in which an agent does not directly observe the state, and instead only receives a partial observation. This setting is formalised by a Partially Observable Markov Decision Process (POMDP) [35]: $\langle \mathcal{S}, \mathcal{A}, Z, P, R, O, \gamma, S_0 \rangle$. $\mathcal{S}, \mathcal{A}, P, R, \gamma,$ and S_0 are defined exactly as for an MDP. A POMDP additionally introduces Z and O, where:
Z is the set of possible observations.
O is the observation function, $O(z|s,a) = \mathbb{P}[O_{t+1} = o | S_t = s, A_t = a]$.

There are many different formulations for the observation function in a POMDP. Here we assume that the observation an agent receives at timestep $t+1$ depends on the previous state and previous action they took.

The important thing to note about a POMDP is that an agent no longer knows the true state of the system, and is only receiving partial observations. Usually a single observation alone is not sufficient to determine the true state of the system, and hence the transition and reward dynamics with respect to the observation are **not** Markov. Thus, an agent may need to condition on their entire history of actions and observations in order to try and act optimally. We refer to reader to [35] for more information on POMDPs and techniques for solving them.

We will cover the formalism we use in Part II, a Decentralised Partially Observable MDP (Dec-POMDP) in much more detail in Chapter 4.

2.1.2 Algorithms

There are many approaches and algorithms for this particular RL setting. We will briefly describe some relevant model-free methods in this section.

Q-learning

Most relevant to the content of this thesis is Q-learning [36]. Essentially, Q-learning approximates the Bellman Optimality equation by using samples drawn from the environment. The update rule for Q-learning is as follows:

$$Q_{n+1}(s_t, a_t) \leftarrow Q_n(s_t, a_t) + \eta(r_t + \gamma \max_{a'} Q_n(s_{t+1}, a') - Q_n(s_t, a_t)),$$

where η is the learning rate. Starting from an initial estimate of the Q^*-values (Q_0) for every state-action pair, after selecting an action, receiving the reward and transitioning to the next state, it updates its estimate of Q^*.

In the tabular setting, in which the Q-values for every state-action pair are stored in a table (i.e., no function approximation), Q-learning converges under some mild assumptions on the learning rate η and provided that every state-action pair is visited infinitely often [37].

In order to act greedily, an agent will act with respect to its Q-value estimates: $\pi(s) = \arg\max_a Q(s, a)$.

Policy Gradient

An alternative approach is to learn a parametrised policy π_θ and update it directly to maximise the objective $J(\pi_\theta)$ by following the gradient. This is made tractable by the celebrated Policy Gradient theorem [38], which we will outline for the episodic setting with a single start state s_0 for simplicity.

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \nabla_\theta V^{\pi_\theta}(s_0) \\ &= \text{... (Sutton et al. [38] provide a full derivation)} \\ &= \mathbb{E}_{s\sim d^{\pi_\theta}, a\sim\pi_\theta(\cdot|s)}[\nabla_\theta \log \pi_\theta(a|s) Q^{\pi_\theta}(s, a)],\end{aligned}$$

where $d^{\pi_\theta}(s) := \sum_{t=0}^{\infty} \gamma^t \mathbb{P}(S_t = s|s_0, \pi_\theta)$ is the discounted state occupancy distribution assuming we start at state s_0 and follow policy π_θ.

The policy gradient theorem lets us approximate the gradient by using samples drawn from the environment and the ability to differentiate our parametrised policy.

One of the simplest methods that uses this is REINFORCE [39], which approximates the expectation and Q^{π_θ} using a single sample Monte-Carlo estimate by sampling a whole trajectory from the environment of length T:

$$\nabla_\theta J(\pi_\theta) \approx \sum_{t=0}^{T} \gamma^t \Big(\sum_{k=0}^{T-t} \gamma^k r_{t+k}\Big) \nabla_\theta \log \pi_\theta(a_t|s_t).$$

A large drawback of REINFORCE is the high variance of the update, due to the use of a single sample to estimate the expectation. One potential method to reduce this variance is to introduce a baseline into our estimate of Q^{π_θ} that only depends on the state:

$$\nabla_\theta J(\pi_\theta) \approx \sum_{t=0}^{T} \gamma^t (\sum_{k=0}^{T-t} \gamma^k r_{t+k} - b(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t).$$

REINFORCE uses an unbiased estimate of V^{π_θ} which has high variance. Alternatively, we can introduce some bias and use a learned estimate of Q^{π_θ}. This is the basis of Actor-Critic algorithms [27, 40, 41]. The *actor* refers to the policy, π_θ, which is trained by an estimate of Q^{π_θ} computed by the critic. Typically, the critic is trained to estimate an expression equivalent to $Q^{\pi_\theta}(s_t, a_t) - b(s_t)$, where the baseline b is typically chosen to be the value function V^{π_θ}. This makes the expression equivalent to the advantage $A^{\pi_\theta}(s_t, a_t)$. We direct the reader to [42] for other alternative expressions that are commonly used.

One important thing to note is the presence of d^{π_θ} in the policy-gradient update (and thus the extra γ^t in our presentation of REINFORCE). Typically, practitioners ignore the discounting present in this distribution and thus forego the extra γ^t term in the update.

2.2 Deep Reinforcement Learning

As we consider more complex environments, with large and possibly non-discrete state spaces, it is necessary to move beyond tabular estimates. **Deep** Reinforcement Learning makes use of deep (typically much smaller than in supervised learning however) neural networks in order to handle large and continuous state spaces. The field is large and encompasses many different research directions and motivations. We direct the reader to [43] for a starting point on Deep RL and its implementations.

2.2.1 DQN

Most relevant to the work in this thesis is a Deep Q Network (DQN) [24]. A DQN represents the Q-values it is estimating using a deep neural network that is typically

parametrised by θ: $Q(\cdot, \cdot; \theta)$. A DQN maintains an experience replay buffer [44], D, which stores the last N experiences it has sampled from the environment. An experience typically refers to a tuple $(s_t, a_t, r_t, s_{t+1}, \text{terminal})$ which contains the current state (s_t), the action the agent took (a_t), and the reward received (r_t) as well as the next state the agent transitioned to (s_{t+1}). We also maintain a boolean to indicate whether the episode terminated after transitioning to s_{t+1}.

The parameters of a DQN are updated by minimising the following loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s') \sim D}[(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2]. \tag{2.1}$$

As is common in deep learning, we perform stochastic gradient descent to approximate the gradient of this loss function by sampling a minibatch of experiences from our experience replay buffer and computing the mean gradient across them. Importantly, to improve stability by providing a stationary target, we periodically freeze the parameters of the *target* network to θ^- that is used to compute the bootstrapped targets: $r + \gamma \max_{a'} Q(s', a'; \theta^-)$. We refer the reader to [45] for a more comprehensive description of Deep Learning.

Concretely, a DQN optimises the following objective with respect to θ after sampling a minibatch of b experiences:

$$\mathcal{L}(\theta) = \frac{1}{b} \sum_{i=1}^{b} (r_i + \gamma \max_{a'} Q(s'_i, a'; \theta^-) - Q(s_i, a_i; \theta))^2. \tag{2.2}$$

ϵ-greedy exploration

The most common exploration strategy for a DQN agent is to utilise ϵ-greedy action selection, in which an agent selects an action uniformly from all actions with probability ϵ and otherwise selects the action with the highest estimated Q-value.

DRQN

A DQN estimates the Q-values for a given state-action pair. However, in partially observable settings the agent does not receive the state to use as input to a DQN. Instead, an agent often wishes to learn Q-values that condition on their entire action and observation history. Hausknecht and Stone [46] propose a Deep Recurrent Q

Network for this purpose. Instead of using a feedforward network that takes the state as input and produces Q-values for each action, they utilise a recurrent neural network (RNN). At each timestep the network takes as input the observation (and usually the previously taken action as well) and then outputs Q-values for each possible action. By using a recurrent neural network, the agent can condition their Q-value estimates on all previous actions and observations.

Double Q-learning

As is extremely common in Deep RL, we will utilise Double Q-learning with our DQN (or DRQN) [47], in which the targets are computed as follows: $r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)$. We utilise our current Q-value estimate, with the most recent set of parameters θ to select the argmax action which is then evaluated by our target network using the frozen parameters θ^-.

Part I

Single-Agent Exploration

Abstract

In this part of the thesis we will focus on how an agent can explore their environment effectively. We will investigate how this can be done in the single-agent setting only, in order to focus our attention on the difficulties associated with exploration without worrying about the additional complexities of a multi-agent setting. We choose to investigate this issue because exploration is of fundamental importance in Reinforcement Learning. An agent must first explore their environment, in order to observe the possible transitions and rewards, so that they have the necessary data to infer a good control policy. This is of particular importance in *sparse reward* environments, in which the highly rewarding states are few and far between. A good example of such an environment, one we will use in this chapter for measuring performance, is a simple 2D maze. The only state the agent will ever receive any positive reward from is the goal state, and hence the agent must explore the maze in order to find that goal. This highlights the difficulties present in such sparse reward environments. Without a reward signal to learn from, it can be extremely difficult for an agent to explore their environment since the data it has observed suggests its Q-value estimates should be 0. A popular principle for achieving efficient exploration is *optimism in the face of uncertainty*. Broadly speaking, it suggests than an agent should be *optimistic* about areas of the state-action space it is uncertain about. Practically, this translates to inflating or deliberately overestimating Q-value estimates for state-action pairs that are deemed novel. The most common form of such novelty, that is pervasive throughout the theoretical literature, is the simple count. The more times you have visited a particular state-action pair, the less novel it becomes. Whilst this works extremely successfully in the tabular domain (many algorithms are provably efficient and are asymptotically close to the theoretical lower bound), we cannot count in the same manner in larger or continuous state spaces. Furthermore, many of these provably efficient algorithms rely on particular choices of learning rates or specific initialisation schemes that we cannot enforce when using neural networks as function approximators. This is a large problem, because efficient exploration becomes increasingly paramount in larger more complex environments, where we cannot even hope to exhaustively explore the environment.

This part of the thesis then focusses on how we can leverage the insights from the tabular setting in order to help us design better exploration schemes for Deep

RL algorithms. We will restrict our attention to sparse reward environments, in order to solely focus on the exploration problem, and are particularly interested in improving exploration over simpler undirected heuristics such as ϵ-greedy.

3
Exploration

Contents

3.1 Introduction 27
3.2 Tabular Reinforcement Learning Background 29
3.3 Optimistic Pessimistically Initialised Q-Learning . . . 31
3.3.1 Motivations 31
3.4 Tabular Reinforcement Learning Results 34
3.4.1 Proof of Main Theorem 35
3.4.2 Necessity for Optimism During Action Selection 46
3.4.3 Necessity for Intrinsic Motivation Bonus 47
3.4.4 Necessity for Optimism During Bootstrapping 49
3.5 Deep Reinforcement Learning Background 49
3.5.1 Counting in Large, Complex State Spaces 50
3.5.2 Granularity of the counting model 51
3.6 Deep Reinforcement Learning OPIQ 51
3.7 Related Work 53
3.7.1 Tabular Reinforcement Learning 53
3.7.2 Deep Reinforcement Learning 58
3.8 Experimental Setup 63
3.8.1 Environments 64
3.8.2 Ablations and Baselines 66
3.9 Results 67
3.9.1 Randomised Chain 67
3.9.2 Maze 69
3.9.3 Montezuma's Revenge 71
3.10 Conclusions 73

3.1 Introduction

In RL, exploration is crucial for gathering sufficient data to infer a good control policy. As environment complexity grows, exploration becomes more challenging and simple randomisation strategies become inefficient.

While most provably efficient methods for tabular RL are model-based [29, 48, 49], in Deep RL, learning models that are useful for planning is notoriously difficult and often more complex [50] than model-free methods. Consequently, model-free approaches have shown the best final performance on large complex tasks [24, 51, 52], especially those requiring hard exploration [53, 54]. Therefore, in this chapter, we focus on how to devise model-free RL algorithms for efficient exploration that scale to large complex state spaces and have strong theoretical underpinnings.

Despite taking inspiration from tabular algorithms, current model-free approaches to exploration in Deep RL do not employ *optimistic initialisation*, which is crucial to provably efficient exploration in all model-free tabular algorithms. This is because Deep RL algorithms do not pay special attention to the initialisation of the neural networks and instead use common initialisation schemes that yield initial Q-values around zero. In the common case of non-negative rewards, this means Q-values are initialised to their lowest possible values, i.e., a *pessimistic initialisation*.

While initialising a neural network optimistically would be trivial, e.g., by setting the bias of the final layer of the network, the uncontrolled generalisation in neural networks changes this initialisation quickly. Instead, to benefit exploration, we require the Q-values for novel state-action pairs must remain high until they are explored.

An empirically successful approach to exploration in Deep RL, especially when reward is sparse, is *intrinsic motivation* [55]. A popular variant is based on *pseudocounts* [53], which derive an intrinsic bonus from approximate visitation counts over states and is inspired by the tabular MBIE-EB algorithm [48]. However, adding a positive intrinsic bonus to the reward yields optimistic Q-values only for state-action pairs that have already been chosen sufficiently often. Incentives to

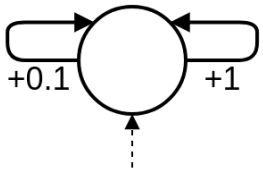


Figure 3.1: A single state MDP with two actions. The left, sub-optimal, action yields +0.1 reward and the right, optimal, action yields +1 reward.

explore unvisited states rely therefore on the generalisation of the neural network. Exactly how the network generalises to those novel state-action pairs is unknown, and thus it is unclear whether those estimates are optimistic when compared to nearby visited state-action pairs.

Consider the simple example with a single state and two actions shown in Figure 3.1. The left action yields +0.1 reward and the right action yields +1 reward. An agent whose Q-value estimates have been zero-initialised must at the first time step select an action randomly. As both actions are underestimated, this will increase the estimate of the chosen action. Greedy agents always pick the action with the largest Q-value estimate and will select the same action forever, failing to explore the alternative. Whether the agent learns the optimal policy or not is thus decided purely at random based on the initial Q-value estimates. This effect will only be amplified by intrinsic reward.

To ensure optimism in unvisited, novel state-action pairs, we introduce Optimistic Pessimistically Initialised Q-Learning (OPIQ). OPIQ does not rely on an optimistic initialisation to ensure efficient exploration, but instead augments the Q-value estimates with count-based bonuses in the following manner:

$$Q^+(s,a) := Q(s,a) + \frac{C}{(N(s,a)+1)^M}, \tag{3.1}$$

where $N(s,a)$ is the number of times a state-action pair has been visited and $M, C > 0$ are hyperparameters. The motivations for this particular form, particularly its links to algorithms for tabular Q-learning such as UCB-H are expanded on in Section 3.3.1. These Q^+-values are then used for both action selection and during bootstrapping, unlike the above methods which only utilise Q-values during these steps. This

allows OPIQ to maintain optimism when selecting actions and bootstrapping, since the Q^+-values can be optimistic even when the Q-values are not.

In the tabular domain, we base OPIQ on UCB-H [28], a simple online Q-learning algorithm that uses count-based intrinsic rewards and optimistic initialisation. Instead of optimistically initialising the Q-values, we pessimistically initialise them and use Q^+-values during action selection and bootstrapping. Pessimistic initialisation is used to enable a worst case analysis where all of our Q-value estimates underestimate Q^* and is **not** a requirement for OPIQ. A Pessimistic initialisation is a worst case scenario because of the greedy action-selection scheme used by UCB-H. In particular, if an action's Q-value estimate that is used for action-selection is pessimistic then there is a possibility it will never be selected. We show that these modifications retain the theoretical guarantees of UCB-H.

Furthermore, our algorithm easily extends to the Deep RL setting. The primary difficulty lies in obtaining appropriate state-action counts in high-dimensional and/or continuous state spaces, which has been tackled by a variety of approaches [53, 54, 56, 57] and is orthogonal to our contributions.

We demonstrate clear performance improvements in sparse reward tasks over:
1) a baseline DQN that just uses intrinsic motivation derived from the approximate counts,
2) simpler schemes that aim for an optimistic initialisation when using neural networks, and
3) strong exploration baselines. We additionally show the importance of optimism during action selection for ensuring efficient exploration in the tabular setting. Visualising the predicted Q^+-values in the Deep RL setting shows that they are indeed optimistic for novel state-action pairs as intended.

3.2 Tabular Reinforcement Learning Background

In this section we will outline the notation we use for our tabular reinforcement learning results.

For the tabular setting, we consider a discrete finite-horizon Markov Decision Process (MDP), which can be defined as a tuple $(\mathcal{S}, \mathcal{A}, \{P_t\}, \{R_t\}, H, \rho)$, where $\mathcal{S}$ is the finite state space, $\mathcal{A}$ is the finite action space, $P_t(\cdot|s,a)$ is the state-transition distribution for timestep $t = 1, ..., H$, $R_t(\cdot|s,a)$ is the distribution over rewards after taking action a in state s, H is the horizon, and ρ is the distribution over starting states. Without loss of generality we assume that $R_t(\cdot|s,a) \in [0,1]$. We use S and A to denote the number of states and the number of actions, respectively, and $N(s,a,t)$ as the number of times a state-action pair (s,a) has been visited at timestep t.

Our goal is to find a set of policies $\pi_t : \mathcal{S} \to \mathcal{A}$, $\pi := \{\pi_t\}$, that chooses the agent's actions at time t such that the expected sum of future rewards is maximised. To this end we define the Q-value at time t of a given policy π as $Q_t^\pi(s,a) := \mathbb{E}\left[r + Q_{t+1}^\pi(s', \pi_{t+1}(s')) \,|\, {}_{r\sim R_t(\cdot|s,a),\, s'\sim P_t(\cdot|s,a)}\right]$, where $Q_t^\pi(s,a) = 0, \forall t > H$. The agent interacts with the environment for K episodes yielding a total regret: $\text{Regret}(K) = \sum_{k=1}^{K} \Big(\max_{\pi^*} Q_1^{\pi^*}(s_1^k, \pi_1^*(s_1^k)) - Q_1^{\pi^k}(s_1^k, \pi_1^k(s_1^k))\Big)$. Here s_1^k refers to the starting state and π^k to the policy at the beginning of episode k. The total number of timesteps the agent interacts with the environment for is then $T := KH$. We are interested in bounding the worst case total regret with probability $1-p$, $0 < p < 1$. We focus on the Regret as opposed to using the PAC-MDP framework to analyse the efficiency of an algorithm in order to leverage the existing results already established for it (particuarly for UCB-H). Note that if an algorithm is considered efficient in one framework it is also efficient in the other [28].

UCB-H [28] is an online Q-learning algorithm for the finite-horizon setting outlined above where the worse case total regret is bounded with a probability of $1-p$ by $\mathcal{O}(\sqrt{H^4 SAT \log(SAT/p)}$. All Q-values for timesteps $t \leq H$ are optimistically initialised at H. The learning rate is defined as $\eta_N = \frac{H+1}{H+N}$, where $N := N(s_t, a_t, t)$ is the number of times state-action pair (s_t, a_t) has been observed at step t and $\eta_1 = 1$ at the first encounter of any state-action pair. The update rule for a transition at step t from state s_t to s_{t+1}, after executing action a_t and receiving reward r_t, is:

$$Q_t(s_t,a_t) \leftarrow (1-\eta_N)\, Q_t(s_t,a_t) + \eta_N \Big(r_t + b_N^T + \min\{H, \max_{a'} Q_{t+1}(s_{t+1}, a')\}\Big), \quad (3.2)$$

where $b_N^T := 2\sqrt{\frac{H^3 \log(SAT/p)}{N}}$, is the count-based intrinsic motivation term. Jin et al. [28] use $\exists c > 0$ s.t. $b_N^T := c\sqrt{\frac{H^3 \log(SAT/p)}{N}}$ in their proof, but we find that $c = 2$ suffices.

3.3 Optimistic Pessimistically Initialised Q-Learning

Our method Optimistic Pessimistically Initialised Q-Learning (OPIQ) ensures optimism in the Q-value estimates of unvisited, novel state-action pairs in order to drive exploration. This is achieved by augmenting the Q-value estimates in the following manner:

$$Q^+(s,a) := Q(s,a) + \frac{C}{(N(s,a)+1)^M},$$

and using these Q^+-values during action selection and bootstrapping. In this section, we first motivate OPIQ, before analysing it in the tabular setting in Section 3.4, and describing a Deep RL implementation in Section 3.6.

3.3.1 Motivations

Optimistic initialisation does not work with neural networks. For an optimistic initialisation to benefit exploration, the Q-values must start sufficiently high. More importantly, the values for unseen state-action pairs must *remain high*, until they are updated. When using a deep neural network to approximate the Q-values, we can initialise the network to output optimistic values, for example, by adjusting the final bias. However, after a small amount of training, the values for novel state-action pairs may not remain high. Furthermore, due to the generalisation of neural networks we cannot know how the values for these unseen state-action pairs compare to the trained state-action pairs. Figure 3.2 (left), which illustrates this effect for a simple regression task, shows that different initialisations can lead to dramatically different generalisations. It is therefore prohibitively difficult to use optimistic initialisation of a deep neural network to drive exploration.

Instead, we augment our Q-value estimates with an optimistic bonus. Our motivation for the form of the bonus in (3.1), $\frac{C}{(N(s,a)+1)^M}$, stems from UCB-H

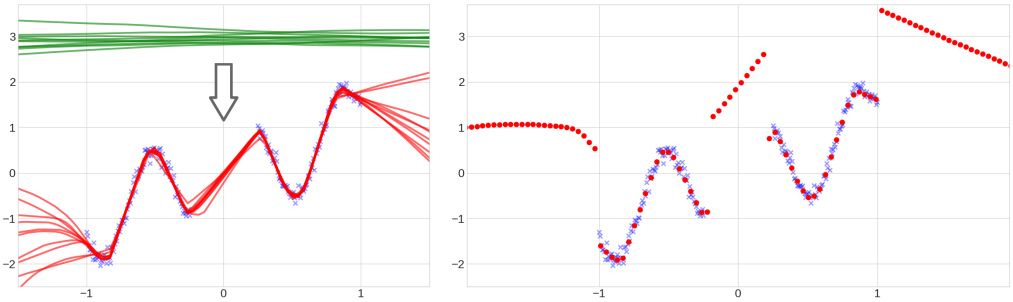


Figure 3.2: A simple regression task to illustrate the effect of an optimistic initialisation in neural networks. **Left:** 10 different networks whose final layer biases are initialised at 3 (shown in green), and the same networks after training on the blue data points (shown in red). **Right:** One of the trained networks whose output has been augmented with an optimistic bonus as in (3.1). The counts were obtained by computing a histogram over the input space $[-2, 2]$ with 50 bins.

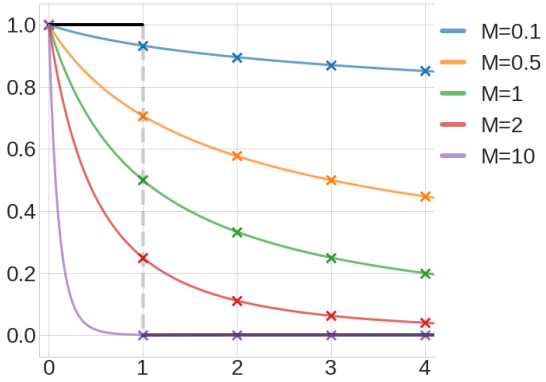


Figure 3.3: $\frac{1}{(x+1)^M}$ for various values of M, and the indicator function $\mathbb{1}\{x < 1\}$ shown in black. Higher values of M provide a better approximation at integer values of x (shown as crosses).

[28], where all tabular Q-values are initialised with H and the first update for a state-action pair completely overwrites that value because the learning rate for the update (η_1) is 1. One can alternatively view these Q-values as zero-initialised with the additional term $Q(s, a) + H \cdot \mathbb{1}\{N(s, a) < 1\}$, where $N(s, a)$ is the visitation count for the state-action pair (s, a). Our approach approximates the discrete indicator function $\mathbb{1}$ as $(N(s, a) + 1)^{-M}$ for sufficiently large M. However, since gradient descent cannot completely overwrite the Q-value estimate for a state-action pair after a single update, it is beneficial to have a smaller hyperparameter M that governs how quickly the optimism decays.

Figure 3.3 compares various values of M with the indicator function $\mathbb{1}\{x < 1\}$.

Although our primary motivations for the bonus are derived from approximating the indicator function, in particular its specific form, there are also links to Bayesian methods. If we start with a prior that is optimistic about all Q-value estimates, then the mean of the Q-value estimates might behave very similarly to our estimates.

For a worst case analysis we assume all Q-value estimates are pessimistic. In the common scenario where all rewards are nonnegative, the lowest possible return for an episode is zero. If we then zero-initialise our Q-value estimates, as is common for neural networks, we are starting with a *pessimistic* initialisation. As shown in Figure 3.2(left), we cannot predict how a neural network will generalise, and thus we cannot predict if the Q-value estimates for unvisited state-action pairs will be optimistic or pessimistic. We thus assume they are pessimistic in order to perform a worst case analysis. However, this is not a requirement: our method works with any initialisation and rewards.

In order to then approximate an optimistic initialisation, the scaling parameter C in (3.1) can be chosen to guarantee unseen Q^+-values are overestimated, for example, $C := H$ in the undiscounted finite-horizon tabular setting and $C := 1/(1-\gamma)$ in the discounted episodic setting (assuming 1 is the maximum reward obtainable at each timestep). However, in some environments it may be beneficial to use a smaller parameter C for faster convergence. These Q^+-values are then used both during action selection and during bootstrapping. Note that in the finite horizon setting the counts N, and thus Q^+, would depend on the timestep t.

Hence, we split the optimistic Q^+-values into two parts: a pessimistic Q-value component and an optimistic component based solely on the counts for a state-action pair. This separates our source of optimism from the neural network function approximator, yielding Q^+-values that remain high for unvisited state-action pairs, assuming a suitable counting scheme. Figure 3.2 (right) shows the effects of adding this optimistic component to a network's outputs. For this particular example, we have used a relatively crude method of discretising the input space $[-2, 2]$ into 50 bins to compute the counts used for the optimism. Since this is a relatively fine-grained discretisation of the space, we are providing large

optimistic bonuses even very close to the data we have observed. In the context of exploration, particularly in a worst case theoretical analysis, this is appropriate because we must ensure (with high probability) that the agent does not ever miss exploring an important action. For example, there might be a very narrow entrance to a cave that contains a large positive reward that the agent could miss exploring if we prematurely conclude we have tried the movement action around the entrance. In a more practical setting, it would be sensible to try and ensure the optimism bonuses are relatively small around the data we have already observed. Section 3.5.1 shows the results for a coarser discretisation.

Optimistic Q^+-values provide an increased incentive to explore. By using optimistic Q^+ estimates, especially during action selection and bootstrapping, the agent is incentivised to try and visit novel state-action pairs. Being optimistic during action selection in particular encourages the agent to try novel actions that have not yet been visited. Without an optimistic estimate for novel state-action pairs the agent would have no incentive to try an action it has never taken before at a given state. Being optimistic during bootstrapping ensures the agent is incentivised to return to states in which it has not yet tried every action. This is because the maximum Q^+-value will be large due to the optimism bonus. Both of these effects lead to a strong incentive to explore novel state-action pairs.

3.4 Tabular Reinforcement Learning Results

Algorithm 1 OPIQ algorithm

Initialise $Q_t(s,a) \leftarrow 0, N(s,a,t) \leftarrow 0, \forall (s,a,t) \in \mathcal{S} \times \mathcal{A} \times \{1,...,H,H+1\}$
for each episode $k = 1,...,K$ **do**
  for each timestep $t = 1,...,H$ **do**
    Take action $a_t \leftarrow \arg\max_a Q_t^+(s_t,a)$.
    Receive $r(s_t,a_t,t)$ and s_{t+1}.
    Increment $N(s_t,a_t,t)$.
    $Q_t(s_t,a_t) \leftarrow (1-\eta_N)Q_t(s_t,a_t) + \eta_N(r(s_t,a_t,t) + b_N^T + \min\{H, \max_{a'} Q_{t+1}^+(s_{t+1},a')\})$.
  end for
end for

In order to ensure that OPIQ has a strong theoretical foundation, we must ensure it is provably efficient in the tabular domain. We restrict our analysis to the finite horizon tabular setting and only consider building upon UCB-H [28] for simplicity. Achieving a better regret bound using UCB-B [28] and extending the analysis to the infinite horizon discounted setting [58] are steps for future work.

Our algorithm removes the optimistic initialisation of UCB-H, instead using a pessimistic initialisation (all Q-values start at 0). We then use our Q^+-values during action selection and bootstrapping. Pseudocode is presented in Algorithm 1.

Theorem 1. *For any $p \in (0,1)$, with probability at least $1-p$ the total regret of Q^+ is at most $\mathcal{O}(\sqrt{H^4SAT\log(SAT/p)})$ for $M \geq 1$ and at most $\mathcal{O}(H^{1+M}SAT^{1-M} + \sqrt{H^4SAT\log(SAT/p)})$ for $0 < M < 1$.*

The proof is based on that of Theorem 1 from [28]. Our Q^+-values are always greater than or equal to the Q-values that UCB-H would estimate, thus ensuring that our estimates are also greater than or equal to Q^*. Our overestimation relative to UCB-H is then governed by the quantity $H/(N(s,a)+1)^M$, which when summed over all timesteps does not depend on T for $M > 1$. As $M \to \infty$ we exactly recover UCB-H, and match the asymptotic performance of UCB-H for $M \geq 1$. Smaller values of M result in our optimism decaying more slowly, which results in more exploration. The full proof is included in the next Section.

We also show that OPIQ without optimistic action selection or the count-based intrinsic motivation term b_N^T is not provably efficient by showing it can incur linear regret with high probability on simple MDPs (see Sections 3.4.2 and 3.4.3).

Our primary motivation for considering a tabular algorithm that pessimistically initialises its Q-values, is to provide a firm theoretical foundation on which to base a Deep RL algorithm, which we describe after our theoretical results.

3.4.1 Proof of Main Theorem

Theorem 1. *For any $p \in (0,1)$, with probability at least $1-p$ the total regret of Q^+ is at most $\mathcal{O}(\sqrt{H^4SAT\log(SAT/p)})$ for $M \geq 1$ and at most $\mathcal{O}(H^{1+M}SAT^{1-M} + \sqrt{H^4SAT\log(SAT/p)})$ for $0 < M < 1$.*

OPIQ is heavily based on UCB-H [28], and as such the proof very closely mirrors its proof except for a few minor differences. For completeness, we reproduce the entirety of the proof with the minor adjustments required for our scheme.

The proof is concerned with bounding the regret of the algorithm after K episodes. The algorithm we follow takes as input the value of K, and changes the magnitudes of b_N^T based on it.

We will make use of a corollary to Azuma's inequality multiple times during the proof.

Theorem 2 (Azuma's Inequality)**.** *Let $Z_0, ..., Z_n$ be a martingale sequence of random variables such that $\forall i \exists c_i : |Z_i - Z_{i-1}| < c_i$ almost surely, then:*

$$P(Z_n - Z_0 \geq t) \leq \exp \frac{-t^2}{2\sum_{i=1}^n c_i^2}$$

Corollary 1. *Let $Z_0, ..., Z_n$ be a martingale sequence of random variables such that $\forall i \exists c_i : |Z_i - Z_{i-1}| < c_i$ almost surely, then with probability at least $1 - \delta$:*

$$|Z_n - Z_0| \leq \sqrt{2(\sum_{i=1}^n c_i^2) \log \frac{2}{\delta}}$$

Lemma 1. *[28]*

Define $\eta_N^0 = \prod_{j=1}^N (1 - \eta_j), \eta_N^i = \eta_i \prod_{j=i+1}^N (1 - \eta_j)$

The following properties hold for η_N^i:

- $\frac{1}{\sqrt{N}} \leq \sum_{i=1}^N \frac{\eta_N^i}{\sqrt{i}} \leq \frac{2}{\sqrt{N}}, \forall N \geq 1$
- $\max_{i=1,...,N} \eta_N^i \leq \frac{2H}{N}, \sum_{i=1}^N (\eta_N^i)^2 \leq \frac{2H}{N}, \forall N \geq 1$
- $\sum_{N=i}^\infty \eta_N^i = 1 + \frac{1}{H}, \forall i \geq 1$

Lemma 2. *Adapted slightly from [28]*

Define $V_t^k(s) := \min\{H, \max_{a'} Q_t^{+,k}(s, a')\}, \forall s \in S$.

For notational convenience, we also define $[P_t^k V_{t+1}](s_t^k, a_t^k) := V_{t+1}(s_{t+1}^k)$, where s_t^k was the state encountered at episode k at timestep t (similarly for a_t^k),

and $[PV_{t+1}](s, a) := \mathbb{E}_{s' \sim P_t(\cdot|s_t=s, a_t=a)} [V_{t+1}(s')]$.

For any $(s,a,t) \in \mathcal{S} \times \mathcal{A} \times [H]$*, episode* $k \leq K$ *and* $N = N(s,a,t)$*. Suppose* (s,a) *was previously taken at step* t *of episodes* $k_1, ..., k_N < k$*. Then:*

$$\begin{aligned}(Q_t^{+,k} - Q_t^*)(s,a) = &-\eta_N^0 Q_t^*(s,a) \\ &+ \sum_{i=1}^{N} \eta_N^i [(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i}) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + b_i^T] \\ &+ \frac{H}{(N+1)^M}\end{aligned}$$

Proof. We have the following recursive formula for Q^+ at episode k and timestep t:

$$Q_t^{+,k}(s,a) = \sum_{i=1}^{N} \eta_N^i [r_t(s,a) + V_{t+1}^{k_i}(s_{t+1}^{k_i}) + b_i^T] + \frac{H}{(N+1)^M} \tag{3.3}$$

We can produce a similar formula for Q^*:

$$Q_t^*(s,a) = (r_t + P_t(s,a)V_{t+1}^*)(s,a)$$

From the Bellman Optimality Equation

$$= \eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i [r_t(s,a) + P_t V_{t+1}^*(s,a)]$$

Since $\sum_{i=1}^{N} \eta_N^i = 1$ and $\eta_N^0 = 0$ for $N \geq 1$ and $\sum_{i=1}^{N} \eta_N^i = 0$ and $\eta_N^0 = 1$ for $N = 0$

$$= \eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i [r_t(s,a) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + P_t^{k_i} V_{t+1}^*(s,a)]$$

$\pm P_t^{k_i} V_{t+1}^*(s,a)$ inside the summation.

$$= \eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i [r_t(s,a) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + V_{t+1}^*(s_{t+1}^{k_i})] \tag{3.4}$$

By definition of P_t^k

Subtracting (3.3) from (3.4) gives the required result.

□

Lemma 3. *Adapted slightly from [28]*

Bounding $Q^+ - Q^*$

There exists an absolute constant $c > 0$ such that, for any $\delta \in (0,1)$, letting $b_N^T = 2\sqrt{\frac{H^3 log(SAT/\epsilon)}{N}}$, we have that $\beta_N^T = 2\sum_{i=1}^N \eta_N^i b_i^T \leq 8\sqrt{\frac{H^3 log(SAT/\epsilon)}{N}}$ and, with probability at least $1-\delta$, the following holds simultaneously for all $(s,a,t,k) \in \mathcal{S} \times \mathcal{A} \times [H] \times [K]$:

$$0 \leq (Q_t^{+,k} - Q_t^*)(s,a) \leq \sum_{i=1}^N \eta_N^i [(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] + \beta_N^T + \frac{H}{(N+1)^M}$$

where $N = N(s,a,t)$ and $k_1, ..., k_N < k$ are the episodes where (s,a) was taken at step t.

Proof. For each fixed $(s,a,t) \in \mathcal{S} \times \mathcal{A} \times [H]$, let $k_0 = 0$ and

$$k_i = \min(\{k \in [K] | k > k_{i-1}, (s_t^k, a_t^k) = (s,a)\} \cup \{K+1\})$$

k_i is then the episode at which (s,a) was taken at step t for the ith time, or $k_i = K+1$ if it has been taken fewer than i times.

Then the random variable k_i is a *stopping time*. Let $\mathcal{F}_i$ be the σ-field generated by all the random variables until episode k_i step t.

Let $\tau \in [K]$.

Let $X_i := \eta_\tau^i \mathbb{1}[k_i \leq K]\ [(P_t^{k_i} - P_t)V_{t+1}^*](s,a))$. Then $Z_i = \sum_{j=1}^i X_i$ is also a martingale sequence with respect to the filtration $(\mathbb{1}[k_i \leq K]\ [(P_t^{k_i} - P_t)V_{t+1}^*](s,a))_{i=1}^\tau$, $Z_0 = 0$, $Z_n - Z_0 = \sum_{i=1}^n X_i$ and $Z_i - Z_{i-1} = X_i$. We also have that $|X_i| \leq \eta_\tau^i H$.

Then by Azuma's Inequality we have that with probability at least $1-2\delta/(SAHK)$

$$\begin{aligned} |\sum_{i=1}^\tau \eta_\tau^i \mathbb{1}[k_i \leq K]\ [(P_t^{k_i} - P_t)V_{t+1}^*](s,a)| &\leq \sqrt{2(\sum_{i=1}^\tau (\eta_\tau^i H)^2) \log \frac{SAT}{\delta}} \\ &= H\sqrt{2(\sum_{i=1}^\tau (\eta_\tau^i)^2) \log \frac{SAT}{\delta}} \end{aligned}$$

Then by a Union bound over all $\tau \in [K]$, we have that with probability at least $1 - 2\delta/(SAH)$:

$$\begin{aligned} \forall \tau \in [K]\ |\sum_{i=1}^\tau \eta_\tau^i \mathbb{1}[k_i \leq K]\ [(P_t^{k_i} - P_t)V_{t+1}^*](s,a)| &\leq H\sqrt{2(\sum_{i=1}^\tau (\eta_\tau^i)^2) \log \frac{SAT}{\delta}} \\ &\leq 2\sqrt{\frac{H^3 \log SAT/\delta}{\tau}} \end{aligned} \tag{3.5}$$

Since From Lemma 1 we have that $\sum_{i=1}^{\tau}(\eta_N^i)^2 \leq \frac{2H}{\tau}$

Since inequality (3.5) holds for all fixed $\tau \in [K]$ uniformly, it also holds for a random variable τ $\tau = N = N^k(s,a,t) \leq K$. Also note that $\mathbb{1}[k_i \leq K] = 1$ for all $i \leq N$.

We can then additionally apply a union bound over all $s \in \mathcal{S}, a \in \mathcal{A}, t \in [H]$ to give:

$$|\sum_{i=1}^{N} \eta_N^i[(P_t^{k_i} - P_t)V_{t+1}^*](s,a)| \leq 2\sqrt{\frac{H^3 \log SAT/\delta}{N}} = b_N^T \tag{3.6}$$

which holds with probability $1-2\delta$ for all $(s,a,t,k) \in \mathcal{S} \times \mathcal{A} \times [H] \times [K]$. We then rescale δ to $\delta/2$.

By Lemma 1 we have that $b_N^T = 2\sqrt{H^3 \log(SAT/\delta)/N} \leq \beta_N^T/2 = \sum_{i=1}^{N} \eta_N^i b_i^T \leq 4\sqrt{H^3 \log(SAT/\delta)/N} = 2b_N^T$.

From Lemma 2 we have that:

$$\begin{aligned}
(Q_t^{+,k} - Q_t^*)(s,a) &= -\eta_N^0 Q_t^*(s,a) \\
&\quad + \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i}) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + b_i^T] \\
&\quad + \frac{H}{(N+1)^M} \\
&= -\eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] \\
&\quad + \sum_{i=1}^{N} \eta_N^i[(P_t - P_t^{k_i})V_{t+1}^*(s,a)] + \sum_{i=1}^{N} \eta_N^i[b_i^T] + \frac{H}{(N+1)^M}
\end{aligned}$$

rearranging terms

$$\leq \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] + b_N^T + \beta_N^T/2 + \frac{H}{(N+1)^M}$$

From Eqn (3.6), defn on β_N^T and non-negativity of Q^*

$$\leq \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] + \beta_N^T + \frac{H}{(N+1)^M}$$

Since $b_N^T \leq \beta_N^T/2$

which gives the R.H.S.

Eqn (3.6) tells us that $\sum_{i=1}^{N} \eta_N^i[(P_t - P_t^{k_i})V_{t+1}^*](s,a) \geq -b_N^T$, along with $b_N^T \leq \beta_N^T/2 = \sum_{i=1}^{N} \eta_N^i b_i^T$ which then gives:

$$\begin{aligned}(Q_t^{+,k} - Q_t^*)(s,a) &= -\eta_N^0 Q_t^*(s,a) \\ &\quad + \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i}) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + b_i^T] \\ &\quad + \frac{H}{(N+1)^M} \\ &\geq -\eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] + \frac{H}{(N+1)^M}\end{aligned}$$

We can then prove the L.H.S. by induction on $t = H, ..., 1$.

For $t = H (Q_t^{+,k} - Q_t^*) \geq 0$. This is because $V_{H+1}^{k_i} = V_{H+1}^* = 0$, for $N = 0$ $\frac{H}{(0+1)^M} = H > Q_H^*$, and for $N > 0$ we have that $\eta_N^0 = 0$. If we assume the statement true for $t+1$, consider t:

$$\begin{aligned}(Q_t^{+,k} - Q_t^*)(s,a) &\geq -\eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i[(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] + \frac{H}{(N+1)^M} \\ &= -\eta_N^0 Q_t^*(s,a) \\ &\quad + \sum_{i=1}^{N} \eta_N^i[\min\{H, \max_{a'} Q_{t+1}^{+,k_i}(s_{t+1}^{k_i}, a')\} - \max_{a''} Q_{t+1}^*(s_{t+1}^{k_i}, a'')] \\ &\quad + \frac{H}{(N+1)^M} \\ &\geq -\eta_N^0 Q_t^*(s,a) + \frac{H}{(N+1)^M}\end{aligned}$$

If $\min\{H, \max_{a'} Q_{t+1}^{+,k_i}(s_{t+1}^{k_i}, a')\} = \max_{a'} Q_{t+1}^{+,k_i}(s_{t+1}^{k_i}, a')$, by our inductive assumption since $(Q_{t+1}^{+,k} - Q_{t+1}^*)(s,a) \geq 0 \implies \max_{a'} Q_{t+1}^{+,k_i}(s,a') \geq \max_{a''} Q_{t+1}^*(s,a'')$

If $\min\{H, \max_{a'} Q_{t+1}^{+,k_i}(s_{t+1}^{k_i}, a')\} = H$, we have that $\max_{a''} Q_{t+1}^*(s_{t+1}^{k_i}, a'') \leq H$.

This then proves the L.H.S.

□

Note on stochastic rewards: We have assumed so far that the reward function is deterministic for a simpler presentation of the proof. If we allowed for a stochastic reward function, the previous lemmas can be easily adapted to allow for it.

Lemma 2 would give us:

$$(Q_t^{+,k} - Q_t^*)(s,a) = -\eta_N^0 Q_t^*(s,a) + \sum_{i=1}^{N} \eta_N^i [(r_t^{k_i} - \mathbb{E}_{r' \sim R_t(\cdot|s,a)}[r']) + (V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i}) + (P_t - P_t^{k_i})V_{t+1}^*(s,a) + b_i^T] + \frac{H}{(N+1)^M}$$

$(r_t^{k_i} - \mathbb{E}_{r' \sim R_t(\cdot|s,a)}[r'])$ can then be bounded in the same way as in Lemma 3. Increasing the constants for b_N^T, β_N^T and rescaling of δ appropriately then completes the necessary changes.

We will now prove that our algorithm is provably efficient. We will do this by showing it has sub-linear regret, following closely the proof of Theorem 1 from [28].

Theorem 1. *For any $p \in (0,1)$, with probability at least $1-p$ the total regret of Q^+ is at most $\mathcal{O}(\sqrt{H^4 SAT \log(SAT/p)})$ for $M \geq 1$ and at most $\mathcal{O}(H^{1+M} SAT^{1-M} + \sqrt{H^4 SAT \log(SAT/p)})$ for $0 < M < 1$.*

Proof. Let

$$\delta_t^k := (V_t^k - V_t^{\pi_k})(s_t^k), \phi_t^k := (V_t^k - V_t^*)(s_t^k)$$

Lemma 3 tells us with probability at least $1-\delta$ that $Q_t^{+,k} \geq Q_t^*$, which also implies that $V_t^k \geq V_t^*$. We can then upperbound the regret as follows:

$$\text{Regret}(K) = \sum_{k=1}^{K} (V_1^* - V_1^{\pi_k})(s_1^k) \leq \sum_{k=1}^{K} (V_1^k - V_1^{\pi_k})(s_1^k) = \sum_{k=1}^{K} \delta_1^k$$

We then aim to bound $\sum_{k=1}^K \delta_t^k$ by the next timestep $\sum_{k=1}^K \delta_{t+1}^k$, which will give us a recursive formula to upperbound the regret. We will accomplish this by relating $\sum_{k=1}^K \delta_t^k$ to $\sum_{k=1}^K \phi_t^k$.

For any fixed $(k,t) \in [K] \times [H]$, let $N_k = N(s_t^k, a_t^k, t)$ where (s_t^k, a_t^k) was

previously taken at step t of episodes $k_1, ..., k_N < k$. We then have:

$$\begin{aligned}\delta_t^k = (V_t^k - V_t^{\pi_k})(s_t^k) &\leq (Q_t^{+,k} - Q_t^{\pi_k})(s_t^k, a_t^k) \\ &= (Q_t^{+,k} - Q_t^*)(s_t^k, a_t^k) + (Q_t^* - Q_t^{\pi_k})(s_t^k, a_t^k) \\ &\leq \sum_{i=1}^{N_k} \eta_{N_k}^i [(V_{t+1}^{k_i} - V_{t+1}^*)(s_{t+1}^{k_i})] \\ &+ \beta_{N_k}^T + \frac{H}{(N_k+1)^M} + [P_t(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k)\end{aligned}$$

By Lemma 3 for the first term. Bounding $(Q_t^* - Q_t^{\pi_k})$ is achieved through the Bellman Equation giving the final term.

$$\begin{aligned}&= \sum_{i=1}^{N_k} \eta_{N_k}^i \phi_{t+1}^{k_i} + \beta_{N_k}^T + \frac{H}{(N_k+1)^M} + [P_t(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k) \\ &+ [P_t^k(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k) - [P_t^k(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k)\end{aligned}$$

$\pm[P_t^k(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k)$ and subbing for ϕ

$$\begin{aligned}&= \sum_{i=1}^{N_k} \eta_{N_k}^i \phi_{t+1}^{k_i} + \beta_{N_k}^T + \frac{H}{(N_k+1)^M} \\ &+ [(P_t - P_t^k)(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k) + (V_{t+1}^* - V_{t+1}^{\pi_k})(s_{t+1}^k)\end{aligned}$$

Since $[P_t^k(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k) = (V_{t+1}^* - V_{t+1}^{\pi_k})(s_{t+1}^k)$

$$= \sum_{i=1}^{N_k} \eta_{N_k}^i \phi_{t+1}^{k_i} + \beta_{N_k}^T + \frac{H}{(N_k+1)^M} + \xi_{t+1}^k + \delta_{t+1}^k - \phi_{t+1}^k \quad (3.7)$$

Letting $\xi_{t+1}^k := [(P_t - P_t^k)(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k).(V_{t+1}^* - V_{t+1}^{\pi_k})(s_{t+1}^k) = \delta_{t+1}^k - \phi_{t+1}^k$.

We must now bound the summation $\sum_{k=1}^K \delta_t^k$, which we will do by considering each term of (3.7) separately.

$\sum_{k=1}^K \frac{H}{(N_k+1)^M}$:

$$\sum_{k=1}^K \frac{H}{(N_k+1)^M} \geq \sum_{k=1}^K \frac{H}{(K+1)^M} = \frac{HK}{(K+1)^M} \implies \sum_{k=1}^K \frac{H}{(N_k+1)^M} \in \Omega(HK^{1-M})$$

The first inequality follows since $N \leq K$. This shows that we require $M > 0$ in order to guarantee a sublinear regret in K.

$$\begin{aligned}\sum_{k=1}^K \frac{H}{(N_k+1)^M} = \sum_{k=1}^K \sum_{n=0}^\infty \mathbb{1}\{N_k = n\} \frac{H}{(n+1)^M} &= \sum_{n=0}^\infty \sum_{k=1}^K \mathbb{1}\{N_k = n\} \frac{H}{(n+1)^M} \\ &\leq SA \sum_{n=0}^\infty \frac{H}{(n+1)^M} = SAH \sum_{n=1}^K \frac{1}{n^M}\end{aligned}$$

The first equality follows by rewriting $\frac{H}{(N_k+1)^M}$ as $\sum_{n=0}^{\infty} \mathbb{1}\{N_k = n\}\frac{H}{(n+1)^M}$. Only one of the indicator functions will be true for episode k, giving us the required value. The inequality is a crude upper bound that suffices for the proof. For a given n, $\mathbb{1}\{N_k = n\}$ can only be true for at most SA times across all of training. They will contribute $\frac{H}{(n+1)^M}$ to the final sum.

For $M > 1$, the sum $\sum_{n=1}^{K} \frac{1}{n^M}$ is bounded, which means that $\sum_{k=1}^{K} \frac{H}{(N_k+1)^M} \in \mathcal{O}(SAH)$

For $M = 1$, $\sum_{n=1}^{K} \frac{1}{n^M} \leq \log(K+1) \implies \sum_{k=1}^{K} \frac{H}{(N_k+1)^M} \in \mathcal{O}(SAH\log(K))$

For $0 < M < 1$, $\sum_{k=1}^{K} \frac{H}{(N_k+1)^M} \in \mathcal{O}(SAHK^{1-M})$, since $\sum_{n=1}^{K} \frac{1}{n^M} \sim \frac{K^{1-M}}{1-M}$ [59]

$\sum_{k=1}^{K}\sum_{i=1}^{N_k} \eta_{N_k}^{i}\phi_{t+1}^{k_i}$:

$$\sum_{k=1}^{K}\sum_{i=1}^{N_k} \eta_{N_k}^{i}\phi_{t+1}^{k_i} \leq \sum_{k'=1}^{K} \phi_{t+1}^{k'} \sum_{i=N_{k'}+1}^{\infty} \eta_i^{N_{k'}} \leq (1+\frac{1}{H})\sum_{k=1}^{K}\phi_{t+1}^{k}$$

The first inequality is achieved by rearranging how we take the sum.

Consider a $k' \in [K]$. The term $\phi_{t+1}^{k'}$ will appear in the summation for $k > k'$ provided that $(s_t^k, a_t^k) = (s_t^{k'}, a_t^{k'})$, e.g. we took the same state-action pair at episodes k and k'.

On the first occasion, the associated learning rate will be $\eta_{N_{k'}+1}^{N_{k'}}$, on the second $\eta_{N_{k'}+2}^{N_{k'}}$.

We can then consider all possible counts up to ∞ to achieve an inequality. By considering all $k' \in [K]$ we will not miss any terms of the original summation.

The second inequality follows by application of Lemma 1 on the sum involving the learning rate.

$\sum_{k=1}^{K} \beta_{N_k}$:

For all $t \in [H]$, we have that:

$$\begin{aligned}
\sum_{k=1}^{K} \beta_{N_k} \leq \sum_{k=1}^{K} 8\sqrt{\frac{H^3 \log(SAT/\delta)}{N_k}} &= \sum_{s,a} \sum_{n=1}^{N_k(s,a,t)} 2\sqrt{\frac{H^3 \log(SAT/\delta)}{n}} \\
&\leq \sum_{s,a} \sum_{n=1}^{N_k(s,a,t)=\frac{K}{SA}} 2\sqrt{\frac{H^3 \log(SAT/\delta)}{n}} \\
&\leq \sum_{s,a} 2\sqrt{H^3 \log(SAT/\delta)} \sum_{n=1}^{\frac{K}{SA}} \frac{1}{\sqrt{n}} \\
&\leq 4SA\sqrt{H^3 \log(SAT/\delta)}\sqrt{\frac{K}{SA}} \\
&\in \mathcal{O}(\sqrt{H^3 SAK \log(SAT/\delta)}) \\
&= \mathcal{O}(\sqrt{H^2 SAT \log(SAT/\delta)})
\end{aligned}$$

We transform the sum over the episode's counts N_k into a sum over all state-action pairs, summing over all of their respective counts.

The first inequality is due to $N_k = K/SA$ maximising the quantity, this is since $1/\sqrt{n}$ grows smaller as n increases. Thus, we want to *spread out* the summation across the state-action space as much as possible.

Then we use the result that $\sum_{i=1}^{n} 1/\sqrt{i} \leq 2\sqrt{n}$, and that we are taking the sum over SA identical terms. We then set $n = K/SA$ and are thus taking the sum over $N_k(s,a,t)$ identical terms.

$\sum_{t=1}^{H} \sum_{k=1}^{K} \xi_{t+1}^{k}$

Following a similar argument as from Lemma 3, we can apply Azuma's inequality and a union bound over all $(s,a,t) \in S \times A \times [H]$. Let $X_i = [(P_t - P_t^k)(V_{t+1}^* - V_{t+1}^{\pi_k})](s_t^k, a_t^k)$. Then $Z_i := \sum_{j=1}^{i} X_j$ is a martingale difference sequence, and we have that $|Z_i - Z_{i-1}| = |X_i| \leq H$, and $Z_0 = 0$. Then with probability at least $1 - 2\delta$ we have that:

$$|\sum_{t=1}^{H}\sum_{k=1}^{K} \xi_{t+1}^{k}| \leq H|\sum_{k=1}^{K} \xi_{t+1}^{k}| \leq H\sqrt{2KH^2 \log(SAT/\delta)} \in \mathcal{O}(H^{3/2}\sqrt{T \log(SAT/\delta)})$$

Finally, we can utilise these intermediate results when bounding the regret via

Equation (3.7):

$$\begin{aligned}\sum_{k=1}^{K}\delta_t^k &\leq \sum_{i=1}^{N_k}\eta_{N_k}^i\phi_{t+1}^{k_i}+\beta_{N_k}^T+\frac{H}{(N_k+1)^M}+\xi_{t+1}^k+\delta_{t+1}^k-\phi_{t+1}^k \\ &\leq (1+\frac{1}{H})\sum_{k=1}^{K}\phi_{t+1}^k-\sum_{k=1}^{K}\phi_{t+1}^k+\sum_{k=1}^{K}\delta_{t+1}^k+\sum_{k=1}^{K}(\beta_{N_k}^T+\xi_{t+1}^k)+\sum_{k=1}^{K}\frac{H}{(N_k+1)^M}\end{aligned}$$

From our intermediate result on ϕ

$$\leq (1+\frac{1}{H})\sum_{k=1}^{K}\delta_{t+1}^k+\sum_{k=1}^{K}(\beta_{N_k}^T+\xi_{t+1}^k+\frac{H}{(N_k+1)^M})$$

Since $V^* \geq V^{\pi_k} \implies \delta_{t+1}^k \geq \phi_{t+1}^k$

Letting $A_t := \sum_{k=1}^{K}(\beta_{N_k}^T+\xi_{t+1}^k+\frac{H}{(N_k+1)^M})$, we have that $\sum_{k=1}^{K}\delta_H^k \leq A_H$ since $\delta_{H+1}=0$. By induction on $t=H,...,1$ we then have that $\sum_{k=1}^{K}\delta_t^k \leq \sum_{j=0}^{H-t}(1+\frac{1}{H})^j A_{H-j} \implies \sum_{k=1}^{K}\delta_1^k \in \mathcal{O}(\sum_{t=1}^{H}\sum_{k=1}^{K}(\beta_{N_k}^T+\xi_{t+1}^k+\frac{H}{(N_k+1)^M}))$ since $(1+1/H)^H \leq e$.

Using our intermediate results for the relevant quantities then gives us with probability at least $1-2\delta$:

$$\begin{aligned}\sum_{k=1}^{K}\delta_1^k &\in \mathcal{O}(\sqrt{(H^4SAT\log(SAT/\delta))}+H^{3/2}\sqrt{T\log(SAT/\delta)})+\mathcal{O}(\sum_{t=1}^{H}\sum_{k=1}^{K}\frac{H}{(N_k+1)^M}) \\ &= \mathcal{O}(\sqrt{(H^4SAT\log(SAT/\delta))})+\mathcal{O}(\sum_{t=1}^{H}\sum_{k=1}^{K}\frac{H}{(N_k+1)^M})\end{aligned}$$

We then consider the 3 cases of M:

For $M>1$ we get $\mathcal{O}(\sqrt{(H^4SAT\log(SAT/\delta))}+H^2SA)$.

For $M=1$ we get $\mathcal{O}(\sqrt{(H^4SAT\log(SAT/\delta))}+H^2SA\log(K))$.

And for $0<M<1$ we have $\mathcal{O}(\sqrt{(H^4SAT\log(SAT/\delta))}+H^{1+M}SAT^{1-M})$.

$T \geq \sqrt{H^4SAT\log(SAT/\delta)} \implies \sqrt{H^4SAT\log(SAT/\delta)} \geq H^2SA\log(K)$, which means we can remove the $H^2SA\log(K)$ or H^2SA from the upperbound. If $T \leq \sqrt{H^4SAT\log(SAT/\delta)}$ then that is also a sufficient upper bound since the regret cannot be more than $HK=T$.

This gives us the required result after rescaling δ to $\delta/2$.

□

This proves OPIQ is provably efficient. We now consider the necessity of the 3 different components:

- Optimism during Action Selection.
- Intrinsic motivation bonus.
- Optimism during Bootstrapping.

3.4.2 Necessity for Optimism During Action Selection

To emphasise the necessity of optimistic Q-value estimates during exploration, we analyse the simple failure case for pessimistically initialised greedy Q-learning provided in the introduction. We use Algorithm 1, but use Q instead of Q^+ for action selection. We will assume the agent will act greedily with respect to its Q-value estimates and break ties uniformly:

$$a_t \leftarrow \text{Uniform}\{\arg\max_a Q_t(s_t, a)\}.$$

Consider the single state MDP in Figure 3.4 with $H = 1$. We use this MDP to show that with 0.5 probability pessimistically initialised greedy Q-learning never finds the optimal policy.

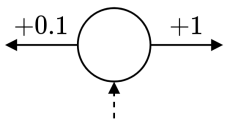


Figure 3.4: A simple failure case for pessimistically initialised greedy Q-learning. There is 1 state with 2 actions and $H = 1$. The agent receives 0.1 reward for the left action and 1 for the right action.

The agent receives a reward of +1 for selecting the right action and 0.1 otherwise. Therefore the optimal policy is to select the right action. Now consider the first episode: $\forall a, Q_1(s, a) = 0$. Thus, the agent selects an action at random with uniform probability. If it selects the left action, it updates:

$$Q_1(s, L) = \eta_1(\underbrace{0.1}_{\text{MDP reward}} + \underbrace{r_{\text{int}}}_{\text{Intrinsic Reward}} + \underbrace{b}_{\text{Bootstrap}}) > 0.$$

Thus, in the second episode it selects the left action again, since $Q_1(s, L) > 0 = Q_1(s, R)$. Our estimate of $Q_1(s, L)$ never drops below 0.1, and so the right action is never taken. Thus, with probability of $\frac{1}{2}$ it never selects the correct action (also a linear regret of $0.9T$).

This counterexample applies for any non-negative intrinsic motivation (including no intrinsic motivation), and is unaffected if we utilise optimistic bootstrapping or not.

3.4.3 Necessity for Intrinsic Motivation Bonus

Despite introducing an extra optimism term with a tunable hyperparameter M, OPIQ also requires the intrinsic motivation term b_i^T to ensure it does not under-explore in stochastic environments.

We will prove that OPIQ without the intrinsic motivation term b_i^T does not satisfy Theorem 1. Specifically we will show that there exists a 1 state, 2 action MDP with stochastic reward function such that for all $M > 0$ the probability of incurring linear regret is greater than the allowed failure probability p. We choose to use stochastic rewards as opposed to stochastic transitions for a simpler proof.

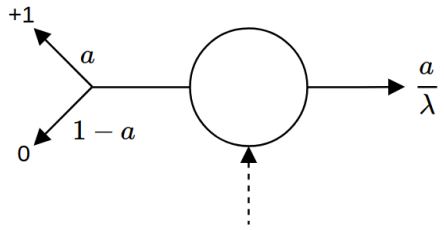


Figure 3.5: The parametrised MDP.

The MDP we will use is shown in Figure 3.5, where $\lambda > 1$ and $a \in (0, 1)$ s.t $p < 1 - a$. $H = 1, S = 1$ and $A = 2$. The reward function for the left action is stochastic, and will return $+1$ reward with probability a and 0 otherwise. The reward for the right action is always a/λ.

Let $p > 0$, the probability with which we are allowed to incur a total regret not bounded by the theorem. OPIQ cannot depend on the value of λ or a as they are unknown.

Pick λ s.t $M > \frac{\log(\frac{\lambda}{a})}{\log(\frac{\log(p)}{\log(1-a)})}$.

OPIQ will recover the sub-optimal policy of taking the right action if every time we take the left action we receive a 0 reward. This will happen since our Q^+-value estimate for the left action will eventually drop below the Q-value estimate for the right action which is $a/\lambda > 0$. The sup-optimal policy will incur a linear regret, which is not bounded by the theorem.

Our probability of failure is at least $(1-a)^R$, where R is the number of times we select the left action, which decreases as R increases. This corresponds to receiving a 0 reward for every one of the R left transitions we take. Note that $(1-a)^R$ is an underestimate of the probability of failure.

For the first 2 episodes we will select both actions, and with probability $(1-a)$ the left action will return 0 reward. Our Q-values will then be: $Q_1(s, L) = 0, Q_1(s, R) = a/\lambda$.

It is possible to take the left action as long as

$$\frac{1}{(R+1)^M} \geq \frac{a}{\lambda},$$

since the optimistic bonus for the right action decays to 0.

This then provides a very loose upper bound for R as $(\frac{\lambda}{a})^{1/M}$, which then leads to a further underestimation of the probability of failure.

Assume for a contradiction that $(1-a)^R < p$:

$$\begin{aligned}
(1-a)^R < p &\iff (1-a)^{(\lambda/a)^{1/M}} < p \\
&\iff (\lambda/a)^{1/M} \log(1-a) < \log(p) \\
&\iff (\lambda/a)^{1/M} > \log(p)/\log(1-a) \\
&\iff (1/M)\log(\lambda/a) > \log(\log(p)/\log(1-a)) \\
&\iff M < \log(\lambda/a)/\log(\log(p)/\log(1-a))
\end{aligned} \tag{3.8}$$

This provides our contradiction as we choose λ s.t $M > \log(\lambda/a)/\log(\log(p)/\log(1-a))$. We can always pick such a λ because $\log(\lambda/a)$ can get arbitrarily close to 0.

So our probability of failure (of which $(1-a)^R$ is a severe underestimate) is greater than the allowed probability of failure p.

3.4.4 Necessity for Optimism During Bootstrapping

We hypothesise that optimism during bootstrapping, that is using Q^+ instead of Q in our bootstrapped targets, is **not** necessary for a provably efficient algorithm. This is because our intrinsic motivation term, b_N^T is greater than H on the first visit to (s, a). Thus, we have that $Q(s, a) \geq H$ as well. Then, assuming we transitioned to the next state s' for the first time, its Q-value estimate will also become greater than H due to its intrinsic motivation bonus as well. Hence, the next time we find ourselves taking this same transition ((s, a) to s'), our bootstrapped estimate will be greater than H and thus optimistic without having to use Q^+.

However, this does not necessarily imply it is a useless component in the Deep RL setting. This is because we do not (and can not) exactly replicate the theoretical setup in our Deep RL experiments. In particular, the scale of the intrinsic motivation bonus is far smaller in the Deep RL setting than is used in our theoretical analysis.

3.5 Deep Reinforcement Learning Background

In this section we briefly outline the setting we consider for our Deep RL experiments and the pseudo count-based intrinsic motivation we make use of.

We consider a Markov Decision Process (MDP) as defined in Chapter 2.

Deep Q-Network (DQN) [24] uses a nonlinear function approximator (a deep neural network) to estimate the action-value function, $Q(s, a; \theta) \approx Q^*(s, a)$, where θ are the parameters of the network. Exploration based on intrinsic rewards [e.g., 53], which uses a DQN agent, additionally augments the observed rewards r_t with a bonus $\beta/\sqrt{N(s_t, a_t)}$ based on pseudo-visitation-counts $N(s_t, a_t)$. The DQN parameters θ are trained by gradient descent on the mean squared regression

loss $\mathcal{L}$ with bootstrapped 'target' y_t:

$$\mathcal{L}(\theta) := \mathbb{E}_{(s_t,a_t,r_t,s_{t+1})\sim D}[(\overbrace{r_t + \frac{\beta}{\sqrt{N(s_t,a_t)}} + \gamma \max_{a'} Q(s_{t+1}, a'; \theta^-)}^{y_t} - Q(s_t, a_t; \theta))^2]. \tag{3.9}$$

The expectation is estimated with uniform samples from a replay buffer D [44]. D stores past transitions (s_t, a_t, r_t, s_{t+1}), where the state s_{t+1} is observed after taking the action a_t in state s_t and receiving reward r_t. To improve stability, DQN uses a target network, parameterised by θ^-, which is periodically copied from the regular network and kept fixed for a number of iterations.

3.5.1 Counting in Large, Complex State Spaces

In Deep RL, the primary difficulty for exploration based on count-based intrinsic rewards is obtaining appropriate state-action counts. In this chapter we utilise approximate counting schemes [53, 54, 56] in order to cope with continuous and/or high-dimensional state spaces. In particular, for the chain and maze environments we use *static hashing* [56] for its simplicity of implementation and minimal use of domain knowledge. It projects a state s to a low-dimensional feature vector $\phi(s) = \text{sign}(Af(s))$, where f flattens the state s into a vector of length D (the flattening doesn't apply if the state s is already a vector, but it is necessary for images); A is a $k \times D$ matrix whose entries are initialised i.i.d. from a unit Gaussian: $\mathcal{N}(0, 1)$; and k is a hyperparameter controlling the *granularity* of counting: higher k leads to more distinguishable states at the expense of generalisation.

Given the vector $\phi(s)$, we use a counting bloom filter [60] to update and retrieve its counts efficiently (leveraging the open-sourced implementation of [56]). To obtain counts $N(s, a)$ for state-action pairs, we maintain a separate data structure of counts for each action (the same vector $\phi(s)$ is used for all actions). This counting scheme is tabular and hence the counts for sufficiently different states do not interfere with one another. This ensures Q^+-values for unseen state-action pairs in (3.1) are large.

For our experiments on Montezuma's Revenge we use the same method of downsampling as in [61], in which the greyscale state representation is resized from

(42x42) to (11x8) and then binned from $\{0, ..., 255\}$ into 8 categories. We then maintain tabular counts over the new representation.

3.5.2 Granularity of the counting model

The granularity of the counting scheme is an important modelling consideration. If it is too granular, then it will assign an optimistic bias in regions of the state space where the network should be trusted to generalise. On the other hand, if it is too coarse then it could fail to provide enough of an optimistic bias in parts of the state space where exploration is still required.

Figures 3.6 shows the differences between 2 levels of granularity (how many equally-sized bins $[-2, 2]$ is split into). Taiga et al. [62] provide a much more detailed analysis on the granularity of the count-based model, and its implications on the learned Q-values.

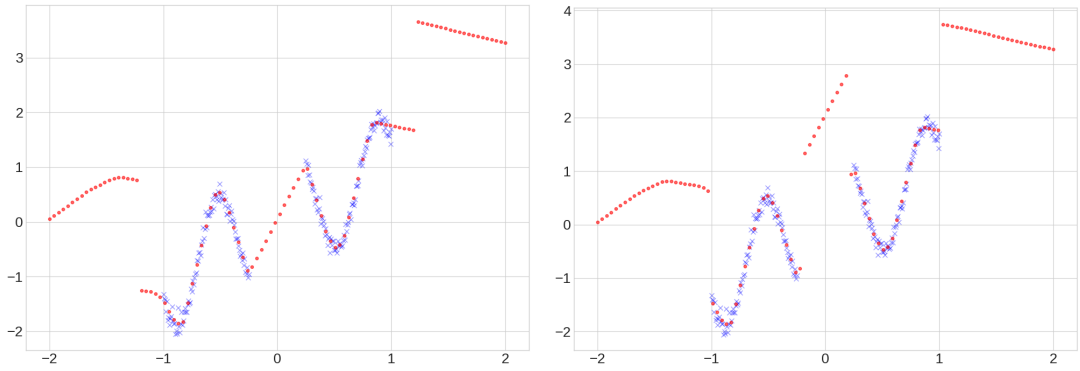


Figure 3.6: We consider the counting scheme from Figure 3.2, but vary the number of bins used. **Left:** 6 bins are used. Only the data points far from the training data are given an optimistic bias. **Right:** 50 bins are used. An optimistic bias is given to all data points that are not very close to the training data.

3.6 Deep Reinforcement Learning OPIQ

For Deep RL, we base OPIQ on DQN [24], which uses a deep neural network with parameters θ as a function approximator Q_θ. During action selection, we use our Q^+-values to determine the greedy action:

$$a_t = \arg\max_a \left\{ Q_\theta(s, a) + \frac{C_{\text{action}}}{(N(s, a) + 1)^M} \right\}, \tag{3.10}$$

where C_{action} is a hyperparameter governing the scale of the optimistic bias during action selection. In practice, we use an ϵ-greedy policy, since a purely greedy action-selection scheme performed terribly in preliminary experiments. After every timestep, we sample a batch of experiences from our experience replay buffer, and use n-step Q-learning [51]. We recompute the counts for each relevant state-action pair, to avoid using stale pseudo-rewards. The network is trained by gradient decent on the loss in (3.9) with the target:

$$y_t := \sum_{i=0}^{n-1} \gamma^i \left(r(s_{t+i}, a_{t+i}) + \frac{\beta}{\sqrt{N(s_{t+i}, a_{t+i})}} \right) + \gamma^n \max_{a'} \left\{ Q_{\theta^-}(s_{t+n}, a') + \frac{C_{\text{bootstrap}}}{(N(s_{t+n}, a') + 1)^M} \right\}. \tag{3.11}$$

where $C_{\text{bootstrap}}$ is a hyperparameter that governs the scale of the optimistic bias during bootstrapping.

For our final experiments on Montezuma's Revenge we additionally use the Mixed Monte Carlo (MMC) target [53, 54], which mixes the target y_t with the environmental monte carlo return for that episode:

$$y_{mmc} := (1 - \beta_{mmc}) y_t + \beta_{mmc} (\sum_{i=0}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}))$$

If the episode hasn't finished yet, we use 0 for the monte carlo return.

Our implementation differs from [53, 54] in that we do not use the intrinsic rewards as part of the monte carlo return. This is because we recompute the intrinsic rewards whenever we are using them as part of the targets for training, and recomputing all the intrinsic rewards for an entire episode (which can be over 1000 timesteps) is computationally prohibitive.

We use the method of static hashing [56] to obtain our pseudocounts on the first 2 of 3 environments we test on. For our experiments on Montezuma's Revenge we count over a downsampled image of the current game frame.

A DQN with pseudocount derived intrinsic reward (DQN + PC) [53] can be seen as a naive extension of UCB-H to the Deep RL setting. However, it does not attempt to ensure optimism in the Q-values used during action selection and bootstrapping,

which is a crucial component of UCB-H. Furthermore, even if the Q-values were initialised optimistically at the start of training they would not remain optimistic long enough to drive exploration, due to the use of neural networks. OPIQ, on the other hand, is designed with these limitations of neural networks in mind. By augmenting the neural network's Q-value estimates with optimistic bonuses of the form $\frac{C}{(N(s,a)+1)^M}$, OPIQ ensures that the Q^+-values used during action selection and bootstrapping are optimistic. We can thus consider OPIQ as a *deep* version of UCB-H. Our results show that optimism during action selection and bootstrapping is extremely important for ensuring efficient exploration.

3.7 Related Work

In this section, we summarise related work in both the tabular and Deep RL setting. For a more comprehensive survey, we direct the reader to [63].

3.7.1 Tabular Reinforcement Learning

There is a wealth of literature related to provably efficient exploration in the tabular domain. We will briefly summarise those approaches that are most related to the work in this chapter.

Broadly speaking, exploration in Tabular Reinforcement Learning can be categorised into two categories:

- Optimism in the Face of Uncertainty,
- Bayesian/Posterior Sampling based approaches.

The former encourages exploration through *optimistic* estimates of unknown quantities, either by augmenting the environmental reward with an intrinsically generated one, or by planning with an optimistic model of the MDP. The latter aims to be Bayesian and maintain a distribution over relevant quantities which represents the uncertainty associated with them.

Importantly, both categories aim to improve upon methods of exploration that are undirected such as ϵ-greedy action selection. Thrun [64] provides some of the

earliest work outlining the deficiencies of such *undirected* methods, and showing that there are relatively simple count-based schemes that do not require an exponential number of interactions (in terms of the size of the state space) to identify an optimal policy. There is also an important distinction between model-based and model-free approaches, particularly if we are interested in scaling the methods to the Deep RL setting. Model-based approaches in the tabular setting attempt to explicitly learn the transition and reward functions, whereas model-free approaches do not.

Model Based - Frequentist Regret

Popular model-based algorithms such as R-MAX [29], MBIE (and MBIE-EB) [48], UCRL2 [65] and UCBVI [49] are all based on the principle of *optimism in the face of uncertainty*. R-MAX [29] plans with a learned model that utilises optimistic value estimates for state-action pairs that have been visited less than a pre-defined number of times. MBIE maintains confidence intervals for both the reward and transition functions, and plans using optimistic estimates derived from both. Similarly, UCRL2 maintains confidence intervals for the reward and transition functions and plans with respect to the possible MDPs and acts with respect to the best estimated policy. MBIE-EB simplifies MBIE by planning with the optimistic Q-value estimates that been been augmented with a count-based bonus: $\beta/\sqrt{n(s,a)}$, instead of having to maintain and consider confidence intervals for the reward and transition functions. UCBVI, which can be viewed as the model-based precursor to UCB-H, extends MBIE-EB to include a different exploration bonus and a tighter regret analysis.

These algorithms have been proven to be efficient in either the PAC-MDP framework, or feature a sub-linear regret. Although the frameworks of Regret and PAC-MDP are quite conceptually different, there are strong links between them [28].

In the literature there are two different regret bounds: 1) Frequentist, or minimax regret bounds, and 2) Bayesian regret bounds. The first, which is often referred to as the frequentist, or minimax, regret is a worst case bound. It bounds the largest possible asymptotic regret across *any* MDP (with high probability). In contrast, the Bayesian regret aims to upper bound the *Expected* regret across a

pre-supplied prior over MDPs. A minimax regret bound implies a Bayesian regret bound, but the converse does not necessarily hold [66].

Wiewiora [67] show that potential-based shaping rewards [68] of the form $\gamma\phi(s') - \phi(s)$ and initialising the Q-values to $\phi(s)$ result in the exact same updates when shown the same data. However, this does not mean that they have the same Q-values, which can thus result in inefficient exploration if they are not suitably optimistic. This is since the result is only true when both approaches are being shown the *same* data, but this rarely holds since the estimated Q-values affect the data gathered.

Model Based - Bayesian Approaches

In the Bayesian literature, there is a focus on maintaining distributions over quantities of interest so that the uncertainty can be appropriately captured.

One approach is to explicitly maintain the uncertainty over the model parameters in the form of *belief state*. This transform the original MDP problem into a Bayes-Adaptive MDP [69]. In general, it is too computationally demanding to maintain and plan over this explicit belief state, and so approximations are often required even in the tabular setting. One notable exception is Gittins indices [70], which computes the Bayes-Optimal policy in a multi-armed bandit setting. Kolter and Ng [71] propose an approximation to the Bayes-Optimal policy for MDPs by using a *Bayesian Exploration Bonus*. Their algorithm is conceptually similar to MBIE-EB, except for the form of the bonus: $\frac{\beta}{1+n(s,a)}$ vs $\frac{\beta}{\sqrt{n(s,a)}}$. Importantly, they show that **no** exploratory bonus can lead to a PAC-MDP algorithm if the the bonus is of the form $\frac{\beta}{(n(s,a))^p}$ for $p > 1/2$, which shows the Bayes-Optimal policy is not PAC-MDP. Intuitively, the Bayes-Optimal policy is only optimal with respect to the data you have seen so far, and behaves more greedily than a PAC-MDP approach.

An alternate approach is to utilise the framework of Posterior Sampling outlined in [72]. In these Posterior Sampling based approaches, an explicit distribution over MDPs is maintained, sampled from, an optimal policy computed and then executed. Osband et al. [66] provide Bayesian regret bounds for this kind of approach, closely

related to Thompson Sampling. Osband and Roy [73] argue that posterior sampling (PSRL) [72] is more practically efficient than approaches that are optimistic in the face of uncertainty, and prove that in terms of the Bayesian expected regret PSRL matches the performance of *any* optimistic algorithm up to constant factors. The main insight offered as to why, is that optimistic algorithms are often *too* optimistic in their estimates to maintain computational tractability. Agrawal and Jia [74] prove that an optimistic variant of PSRL is also provably efficient under a frequentist regret bound. We direct the reader to [75] for a vastly more thorough survey of Bayesian Reinforcement Learning in general.

Model-Free Approaches

In terms of frequentist regret, there are a few provably efficient tabular model-free algorithms based on optimism in the face of uncertainty: Delayed Q-learning [30], UCB-H (and UCB-B) [28] and extensions of UCB-H (or UCB-B) to the infinite horizon setting [58]. Delayed Q-learning optimistically initialises the Q-values that are carefully controlled when they are updated. UCB-H and UCB-B also optimistically initialise the Q-values, but also utilise a count-based intrinsic motivation term and a special learning rate to achieve a regret bound comparable to model-based algorithms. In contrast, OPIQ pessimistically initialises the Q-values. Whilst we base our current analysis on UCB-H, the idea of augmenting pessimistically initialised Q-values can be applied to any model-free algorithm.

Although not provably efficient, Dearden et al. [76] propose to maintain distributions over Q-values instead of the transition/reward functions and act greedily with respect to a myopic value of information criteria. This general approach, of maintaining a distribution over value functions instead of a model of the MDP, is extended by Osband et al. [77] to Randomised Least Squares Value Iteration (RLSVI). Although presented for the linear function approximation case, it is only provably efficient for tabular RL. Importantly, RLSVI does not maintain an explicit distribution over the value functions which can be computationally hard. Instead, they inject Gaussian noise when training in order to approximate the (conjugate)

Bayesian posterior. This provides an approach that is computationally cheap (in comparison to planning over an MDP every episode as in PSRL), which can scale to the Deep RL setting. Osband et al. [77] prove it is efficient in terms of Bayesian regret for the tabular setting, and Russo [78] further shows it is efficient in the frequentist case as well. Although extremely attractive from a theoretical and computational perspective, the Deep RL approaches based on this line of research do not perform as well as others in our sparse reward experiments.

Linear Function Approximation

Zanette et al. [79] prove an optimistic variant of RLSVI is provably efficient with respect to frequentist regret in the linear function approximation regime. Yang and Wang [80] leverage ideas from linear bandit literature to learn a low-rank approximation of the model, and show their approach is efficient in the frequentist setting. Jin et al. [81] extend UCB-H to the linear function approximation setting and prove a sub-linear regret bound. A very similar approach is analysed by Wang et al. [82] and shown to be provably efficient under weaker assumptions and for Generalised Linear Models. Although model-free, these approaches rely on the linear function approximation setting to maintain the optimism required to be provably efficient, which we cannot rely on in the Deep RL setting.

Other Approaches

Cai et al. [83] prove than an optimistic variant of PPO is efficient, demonstrating that policy-based approaches can also be provably efficient. A line of research including [84, 85, 86] prove efficiency in environments with respect to the dimensionality of the underlying, often much simpler, state space.

Pazis and Parr [87] propose C-PACE, which achieves efficient exploration in continuous state space by utilising a non-parametric approach. Each Q-value estimate is a weighted combination of the k-nearest neighbour's Q-value estimates, and if the neighbours are further than the pre-defined distance an optimistic value is used instead. The most similar work in the Deep RL setting to this would be Neural Episodic Control (NEC) [88], which also utilises a similar scheme for its

Q-value estimates. Optimistic extensions of NEC would be an interesting avenue for future research, or in extending the non-parametric scheme to provide a notion of whether a state is *known* or not which could be used in a similar scheme to OPIQ.

Ng et al. [68] prove conditions that guarantee the same optimal policy will be learnt when using intrinsic motivation, that the current intrinsic reward must be the difference between a function of the next state and the current state. Schmidhuber [89] outline how *surprise* or *novelty* based intrinsic motivation can be used to explain the emergence of creativity in agents, in which these quantities as measured as the improvements in an agent's internal model of the environment. The surprise or prediction error can be measured concretely as the difference between the model's predicted next state and the true next state, alternatively Frank et al. [90] propose the use of the KL divergence of the learned model before and after training on this experience. Although surprise and novelty are often used interchangeably in the literature, Barto et al. [91] argue they shouldn't be. The surprise should refer to the mismatch between what is expected or predicted to happen, and what actually happens. Novelty on the other hand is then computed with respect to previous data gathered. Thus, an agent can be surprised in states it deems not novel if the internal prediction model is not yet good at predicting those transitions. In this chapter we focus on novelty through the use of pseudocounts due to their strong theoretical links, although surprisal based approaches such as [92] could conceivably be used instead.

3.7.2 Deep Reinforcement Learning

A (very) popular approach to improving exploration in Deep RL is to utilise intrinsic motivation [55], which computes a quantity derived from the data the agent has acquired to add to the *extrinsic* environmental reward.

Intrinsic Motivation

Houthooft et al. [93] compute the KL divergence between a learned transition model's prior and posterior after training on a new experience. Achiam and Sastry [94] instead use the L_2 error of the learned transition model, and Stadie et al. [95]

use the error of the transition model in a feature space learned by an auto-encoder. Kim et al. [96] learn both embeddings of the states and actions, encouraging the mutual information between them to be as large as possible, construct a transition model from them and then utilise the embedding's prediction error similarly. Pathak et al. [97] argue that an *agent focussed* feature space is a better choice for computing this error, especially when the state is large and complex with many uncontrollable elements. The currently most popular form of intrinsic motivation was introduced in [92], whereby the error (or surprise) between randomly projected features of the current state and a learnt model is used. This approach pairs well with the current deep learning infrastructure, and scales easily to arbitrarily large settings whereas approaches such as pseudocounts do not. O'Donoghue et al. [98] introduce the Uncertainty Bellman Equation, which provides a recursive relationship for the variance of a Q-value estimate. This allows for local uncertainties to propagate, leading to more accurate global uncertainties which can then be used as intrinsic rewards for exploration. Imagawa et al. [99] utilise it to produce an optimistic variant of PPO. Savinov et al. [100] utilise an intrinsic reward derived from a concept of *reachability*, in which an agent is rewarded for visiting states that are considered far from those already visited in the current episode.

Most relevant to our work is that of [53], which takes inspiration from MBIE-EB [48]. Bellemare et al. [53] utilise the number of times a state has been visited to compute the intrinsic reward. They outline a framework for obtaining approximate counts, dubbed *pseudocounts*, through a learned density model over the state space. Ostrovski et al. [54] extend the work to utilise a more expressive PixelCNN [101] as the density model, whereas [102] train a neural network as a discriminator to also recover a density model. Machado et al. [57] instead use the successor representation to obtain generalised counts. Choi et al. [103] learn a feature space to count that focusses on regions of the state space the agent can control, in the same manner as [97]. A simpler and more generic approach to approximate counting is *static hashing* which projects the state into a discrete lower dimensional space before counting [56]. None of these approaches attempt to augment or modify the Q-values

used for action-selection or bootstrapping, and hence do not attempt to ensure optimistic values for novel state-action pairs.

Uncertainty Based Approaches

Lipton et al. [104] use Bayes-By-Backprop [105] in order to maintain a distribution over Q-values by maintaining a mean-field variational approximation to the posterior over the DQN's parameters. More sophisticated variational inference schemes are explored in [106, 107] in order to learn more expressive posteriors. Instead of maintaining a distribution over all of the parameters of the network, [108, 109] only maintain one over the last linear layer. This allows the use of computationally efficient linear Bayesian methods, which can admit exact analytical solutions. Barron et al. [108] utilise their distribution to compute the Bayesian surprise which is provided as an intrinsic reward, whereas Azizzadenesheli et al. [109] utilise Thompson Sampling. Dropout [110] can instead be used to approximate a distribution over Q-values [111] to then conduct Thompson Sampling. Osband et al. [112] argue that using these approaches won't lead to a good approximations of the posterior and demonstrate their shortcomings in both simple examples and in experiments.

Plappert et al. [113] utilise noise in parameter space of the neural network to encourage exploration. A similar approach is explored in [114]. Whilst both of these approaches are attractive *drop-in* replacements for ϵ-greedy, they are not theoretically motivated since they do not converge to any approximation of the Bayesian posterior (despite their similarities). Additionally, they are often unable to make significant progress in sparse reward settings unlike OPIQ.

Chen et al. [115] build upon bootstrapped DQN [116] to obtain uncertainty estimates over the Q-values for a given state in order to act optimistically by choosing the action with the largest UCB. However, they do not utilise optimistic estimates during bootstrapping. Osband et al. [112] also extend bootstrapped DQN to include a prior by extending RLSVI [77] to Deep RL. Bootstrapped DQN with a prior is thus a model-free algorithm that has strong theoretical support in the tabular setting. Empirically, however, its performance on sparse reward tasks is

worse than DQN with pseudocounts. Dwaracherla et al. [117] utilise hyper-networks in order to approximate an ensemble, which is much less computationally expensive compared to training N neural-networks in a conventional ensemble. This allows for a better approximation of the posterior to be learnt, which can lead to more efficient exploration. In particular, their improved posterior approximation allows for Information Directed Sampling [118] to be used instead of Thompson Sampling.

Janz et al. [119] emphasise the fundamental differences between the randomised value function approaches [77, 120, 112] and the other Bayesian style approaches that attempt to represent a distribution over Q-values [108, 109, 104]. To ensure efficient exploration we want to sample whole Q-functions from the posterior distribution, which is the goal of RLSVI. In contrast, the other approaches that maintain local uncertainty estimates of the Q-values through a distribution over the parameters of the network are unlikely to produce proper samples from the posterior Q. This is because the Q-values will not be temporally consistent, and will not satisfy the Bellman Uncertainty Equation, since the estimates are sampled independently. Janz et al. [119] shows that Bootstrapped DQN, whilst aiming for this goal, potentially requires a lot of value function heads in order to obtain a good posterior approximation. This requires a lot of computation that scales linearly with the number of heads, instead they propose to use a formulation based on successor uncertainties which performs just as well without the increased computational cost. Flennerhag et al. [121] additionally show that approaches using function approximation, that are trying to approximate the uncertainty over Q, are likely to have a significant bias. This is because all moments must satisfy a Bellman-type equation (notably the expectation and variance must satisfy the Bellman and Uncertainty Bellman equation respectively), and hence you must be able to model the uncertainty over the reward and transition functions appropriately. Instead they propose to use the uncertainty over the td-error, which is conditioned on the trajectory thereby removing all uncertainty related to the MDP and only keeping the epistemic uncertainty over the model's parameters.

Other Approaches

Machado et al. [122] shift and scale the rewards so that a zero-initialisation is optimistic. When applied to neural networks this approach does not result in optimistic Q-values due to the generalisation of the networks. Bellemare et al. [53] empirically show that using a pseudocount-based intrinsic motivation term performs much better empirically on hard exploration tasks.

Fox et al. [123] attempt to generalise the notion of a count to include information about the counts of future state-actions pairs in a trajectory, which they use to provide bonuses during action selection. Oh and Iyengar [124] extend delayed Q-learning to utilise these generalised counts and prove the scheme is PAC-MDP. The generalised counts are obtained through E-values which are learnt using SARSA with a constant 0 reward and E-value estimates initialised at 1. When scaling to the Deep RL setting, these E-values are estimated using neural networks that cannot maintain their initialisation for unvisited state-action pairs, which is crucial for providing an incentive to explore. By contrast, OPIQ uses a separate source to generate the optimism necessary to explore the environment.

As results in [125] show, often intrinsic bonus-based schemes can harm performance in most environments where such extended exploration is not required. Although, they are crucial to achieving good performance in the sparse reward environments we consider in this chapter. Dabney et al. [126] consider a simple alternative to ϵ-greedy which aims to conduct more temporally consistent exploration instead of the dithering 1-step randomisation of simple ϵ-greedy. They show this approach can help in tasks with a dense reward over ϵ-greedy, but it is not enough as presented to tackle the more challenging sparse reward environments. Badia et al. [127] utilise a multi-armed bandit to adaptively tune hyper-parameters, particularly those related to exploration such as the scale of the intrinsic rewards, as well as per-episode intrinsic rewards in order to outperform a baseline human on **all** 57 Atari games.

Ecoffet et al. [128, 61] propose Go-Explore, which first aims to explore the environment in order to generate highly rewarding trajectories, which are then

treated as expert demonstrations to imitate. In its simplest form, the exploration phase of Go-Explore maintains an archive of states (lower dimensional representations which could include domain knowledge), akin to the frontier in a graph search algorithm. A state is picked from the archive (the node is expanded in the graph search analogy), with less-visited states having a higher probability of being picked, and the simulator is reset to that state directly. The agent then proceeds with exploration from that state, adding states to its archive and storing the trajectories that could lead to them. After a suitable exploration phase, the agent is trained on the best trajectories through imitation learning. This procedure relies heavily on the ability to reset the simulator, or the MDP having no stochastic rewards/transitions, which is generally considered unrealistic. To remedy this, they also propose a new variant of Go-Explore which utilises a goal-conditioned policy in order to return to the chosen states without having to reset the simulator. Their findings, that the ability to be able to return to novel states being a bottleneck for exploration algorithms, matches ours. In our experiments we found that 1-step Q-learning was often insufficient at propagating reward fast enough, which lead to poor exploration since the agent couldn't learn to return to the novel states quickly enough. Additionally, Go-Explore could be seen as leveraging the idea of *optimism in the face of uncertainty* in which an agent will attempt to return to the states it has not visited many times before (through an optimistic estimate of their Q-values). But instead of having this behaviour arise implicitly through planning/learning over optimistic Q-values, they make it explicit.

3.8 Experimental Setup

We compare OPIQ against baselines and ablations on three sparse reward environments.

The first is a randomized version of the Chain environment proposed by Osband et al. [116] and used in [129] with a chain of length 100, which we call Randomised Chain. The second is a two-dimensional maze in which the agent starts in the top left corner (white dot) and is only rewarded upon finding the goal (light grey dot).

We use an image of the maze as input and randomise the actions similarly to the chain. The third is Montezuma's Revenge from Arcade Learning environment [130], a notoriously difficult sparse reward environment commonly used as a benchmark to evaluate the performance and scaling of Deep RL exploration algorithms.

See Appendix A.1 for further details on the training of the baselines and hyperparameters used. Code is available at: https://github.com/oxwhirl/opiq.

We now describe the environments and baselines we use.

3.8.1 Environments

Randomised Chain

A randomized version of the Chain environment proposed by Osband et al. [116] and used in [129]. We use a Chain of length 100. The agent starts the episode in State 2, and interacts with the MDP for 109 steps, after which the agent is reset. The agent has 2 actions that can move it Left or Right. At the beginning of training the action which takes the agent left or right at each state is randomly picked and then fixed. The agent receives a reward of 0.001 for going Left in State 1, a reward of 1 for going Right in State 100 and no reward otherwise. The optimal policy is thus to pick the action that takes it Right at each timestep. Figure 3.7 shows the structure of the 100 Chain. Similarly to Osband et al. [116] we use a thermometer encoding for the state: $\phi(s) := (\mathbb{1}\{x \leq s\}) \in \{0,1\}^1 00$.

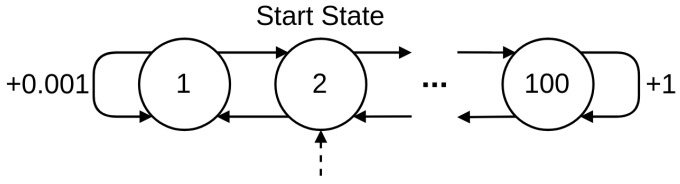


Figure 3.7: 100 Chain environment.

Maze

A 2-dimensional gridworld maze with a sparse reward in which the agent can move Up, Down, Left or Right. The agent starts each episode at a fixed location and must traverse through the maze in order to find the goal which provides +10

reward and terminates the episode, all other rewards are 0. The agent interacts with the maze for 250 timesteps before being reset. Empty space is represented by a 0, walls are 1, the goal is 2 and the player is 3. The state representation is a greyscaled image of the entire grid where each entry is divided by 3 to lie in $[0, 1]$. The shape of the state representation is: $(24, 24, 1)$. Once again the effect of each action is randomised at each state at the beginning of training. Figure 3.8 shows the structure of the maze environment.

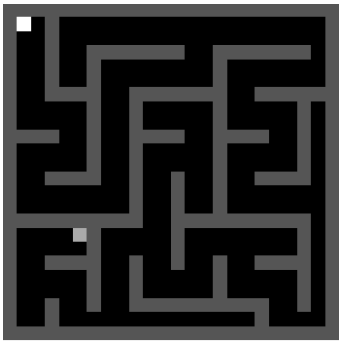


Figure 3.8: Maze environment.

Montezuma's Revenge

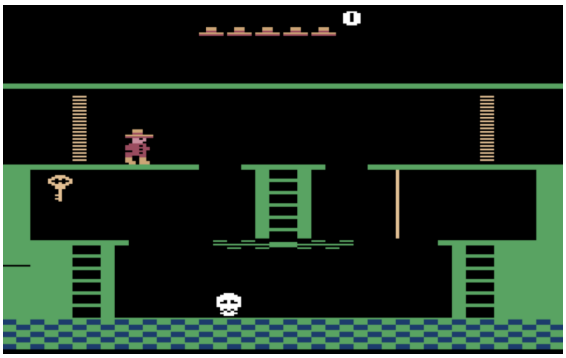


Figure 3.9: The first room of Montezuma's Revenge.

We follow the suggestions in [131] and use the same environmental setup as used in [92]. Specifically, we use stick actions with a probability of $p = 0.25$, a frame skip of 4 and do not show a terminal state on the loss of life.

3.8.2 Ablations and Baselines

We compare OPIQ against a variety of DQN-based approaches that use pseudocount intrinsic rewards, the DORA agent [123] (which generates count-like optimism bonuses using a neural network), and two strong exploration baselines:

ϵ-greedy DQN: a standard DQN that uses an ϵ-greedy policy to encourage exploration. We anneal ϵ linearly over a fixed number of timesteps from 1 to 0.01.
DQN + PC: we add an intrinsic reward of $\beta/\sqrt{N(s,a)}$ to the environmental reward based on [53, 56].
DQN R-Subtract (+PC): we subtract a constant from all environmental rewards received when training, so that a zero-initialisation is optimistic, as described for a DQN in [53] and based on Machado et al. [122].
DQN Bias (+PC): we initialise the bias of the final layer of the DQN to a positive value at the start of training as a simple method for optimistic initialisation with neural networks.
DQN + DORA: we use the *generalised counts* from [123] as an intrinsic reward.
DQN + DORA OA: we additionally use the generalised counts to provide an optimistic bonus during action selection.
DQN + RND: we add the RND bonus from [92] as an intrinsic reward.
BSP: we use Bootstrapped DQN with randomised prior functions [112].

In order to better understand the importance of each component of our method, we also evaluate the following ablations:

Optimistic Action Selection (OPIQ w/o OB): we only use our Q^+-values during action selection, and use Q during bootstrapping (without Optimistic Bootstrapping). The intrinsic motivation term remains.
Optimistic Action Selection and Bootstrapping (OPIQ w/o PC): we use our Q^+-values during action selection and bootstrapping, but do not include an intrinsic motivation term (without Pseudo Counts).

3.9 Results

3.9.1 Randomised Chain

We first consider the visually simple domain of the randomised chain and compare the count-based methods. Figure 3.10 shows the performance of OPIQ compared to the baselines and ablations. OPIQ significantly outperforms the baselines, which do not have any explicit mechanism for optimism during action selection. A DQN with pseudocount derived intrinsic rewards is unable to reliably find the goal state, but setting the final layer's bias to one produces much better performance. For the DQN variant in which a constant is subtracted from all rewards, all of the configurations (including those with pseudocount derived intrinsic bonuses) were unable to find the goal on the right and thus the agents learn quickly to latch on the inferior reward of moving left.

Compared to its ablations, OPIQ is more stable in this task. OPIQ without pseudocounts performs similarly to OPIQ but is more varied across seeds, whereas the lack of optimistic bootstrapping results in worse performance and significantly more variance across seeds.

Looking at the number of states visited, in order to quantify how much of the environment the algorithms explore, we see similar results. OPIQ and ablations explore the environment much more quickly than the count-based baselines. The ablation without optimistic bootstrapping exhibits significantly more variance than the other ablations, showing the importance of optimism during bootstrapping. On

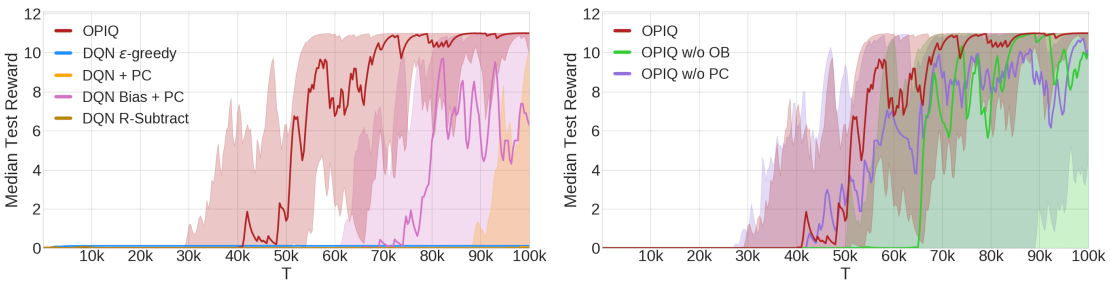


Figure 3.10: Results for the randomised chain environment. Median across 20 seeds is plotted and the 25%-75% quartile is shown shaded. **Left:** OPIQ outperforms the baselines. **Right:** OPIQ is more stable than its ablations.

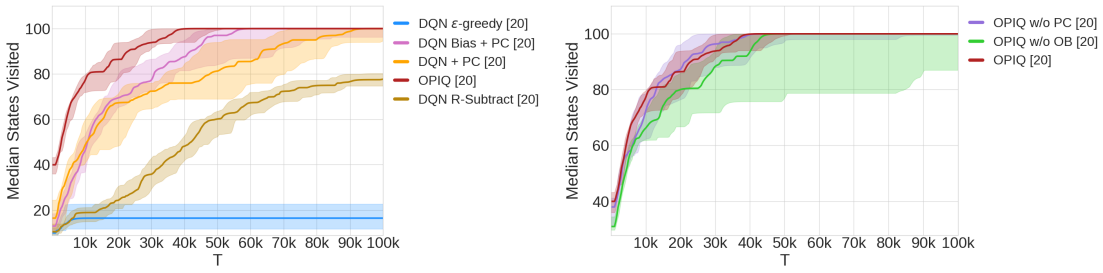


Figure 3.11: The number of distinct states visited over training for the chain environment. The median across 20 seeds is plotted and the 25%-75% quartile is shown shaded.

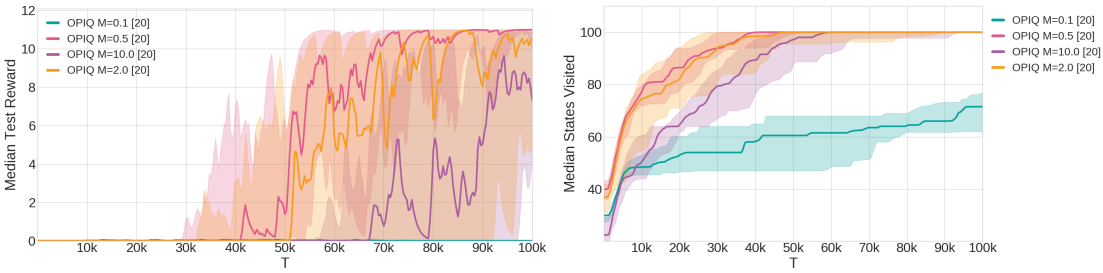


Figure 3.12: Comparing the performance of $M \in \{0.1, 0.5, 2, 10\}$ on the chain environment. The best hyperparameter combination for the differing values of M is shown. The median across 20 seeds is plotted and the 25%-75% quartile is shown shaded.

this simple task the ablation without count-based intrinsic motivation performs on par with the full OPIQ. This is most likely due to the simpler nature of the environment that makes propagating rewards much easier than the Maze. The importance of directed exploration is made abundantly clear by the ϵ-greedy baseline that fails to explore much of the environment.

An important hyperparameter for OPIQ is M, which governs how quickly the count-based optimism decays. Figure 3.12 compares OPIQ with differing values of M, to see the affect of varying this. We can clearly see that a small value of 0.1 results in insufficient exploration, due to the over-exploration of already visited state-action pairs. Additionally if M is too large then the rate of exploration suffers due to the decreased optimism. On this task we found that $M = 0.5$ performed best, but on the harder Maze environment we found that $M = 2$ was better in preliminary experiments.

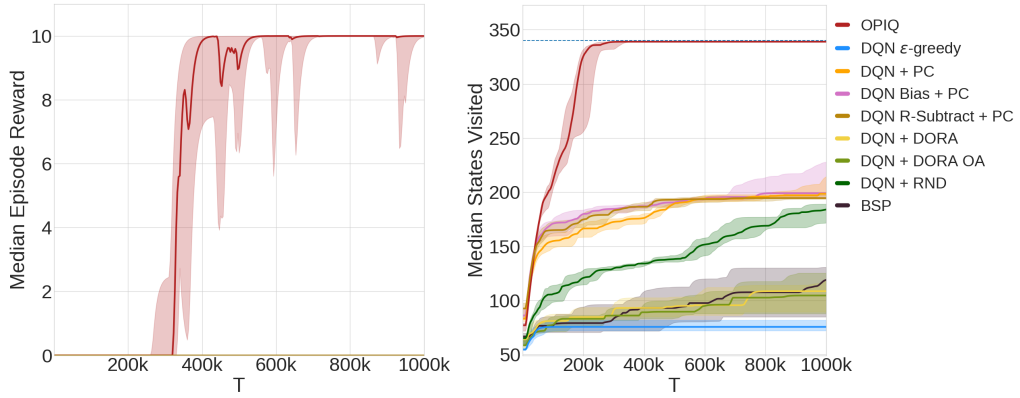


Figure 3.13: Results for the maze environment comparing OPIQ and baselines. Median across 8 seeds is plotted and the 25%-75% quartile is shown shaded. **Left:** The episode reward. **Right:** Number of distinct states visited over training. The total number of states in the environment is shown as a dotted line.

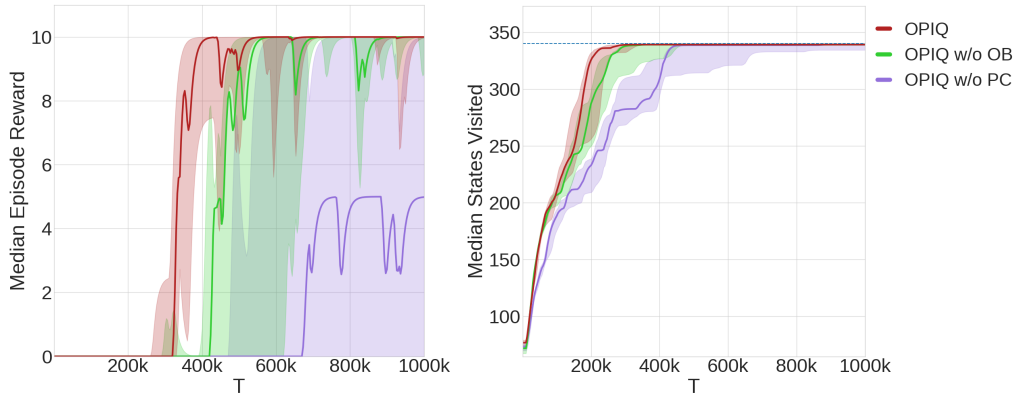


Figure 3.14: Results for the maze environment comparing OPIQ and ablations. Median across 8 seeds is plotted and the 25%-75% quartile is shown shaded. **Left:** The episode reward. **Right:** Number of distinct states visited over training. The total number of states in the environment is shown as a dotted line.

3.9.2 Maze

We next consider the harder and more visually complex task of the Maze and compare against all baselines.

Figure 3.13 shows that only OPIQ is able to find the goal in the sparse reward maze. This indicates that explicitly ensuring optimism during action selection and bootstrapping can have a significant positive impact in sparse reward tasks, and that a naive extension of UCB-H to the Deep RL setting (DQN + PC) results in insufficient exploration.

Figure 3.13 (right) shows that attempting to ensure optimistic Q-values by

adjusting the bias of the final layer (DQN Bias + PC), or by subtracting a constant from the reward (DQN R-Subtract + PC) has very little effect.

As expected DQN + RND performs poorly on this domain compared to the pseudocount based methods. The visual input does not vary much across the state space, resulting in the RND bonus failing to provide enough intrinsic motivation to ensure efficient exploration. Additionally it does not feature any explicit mechanism for optimism during action selection, and thus Figure 3.13 (right) shows it explores the environment relatively slowly.

Both DQN+DORA and DQN+DORA OA also perform poorly in this domain since their source of intrinsic motivation disappears quickly. As noted in Figure 3.2, neural networks do not maintain their starting initialisations after training. Thus, the intrinsic reward DORA produces goes to 0 quickly since the network producing its bonuses learns to generalise quickly.

BSP is the only exploration baseline we test that does not add an intrinsic reward to the environmental reward, and thus it performs poorly compared to the other baselines on this environment.

Figure 3.14 shows that OPIQ and all its ablations manage to find the goal in the maze. OPIQ also explores slightly faster than its ablations (right), which shows the benefits of optimism during both action selection and bootstrapping. In addition, the episodic reward for the the ablation without optimistic bootstrapping is noticeably more unstable (Figure 3.14, left). Interestingly, OPIQ without pseudocounts performs significantly worse than the other ablations. This is surprising since the theory suggests that the count-based intrinsic motivation is only required when the reward or transitions of the MDP are stochastic [28], which is not the case here. We hypothesise that adding PC-derived intrinsic bonuses to the reward provides an easier learning problem, especially when using n-step Q-Learning, which yields the performance gap.

However, our results show that the PC-derived intrinsic bonuses are not enough on their own to ensure sufficient exploration. The large difference in performance between DQN + PC and OPIQ w/o OB is important, since they only differ

in the use of optimistic action selection. The results in Figures 3.13 and 3.14 show that optimism during action selection is extremely important in exploring the environment efficiently. Intuitively, this makes sense, since this provides an incentive for the agent to try actions it has never tried before, which is crucial in exploration.

Figure 3.15 visualises the values used during action selection for a DQN + PC agent and OPIQ, showing the count-based augmentation provides optimism for relatively novel state-action pairs, driving the agent to explore more of the state-action space.

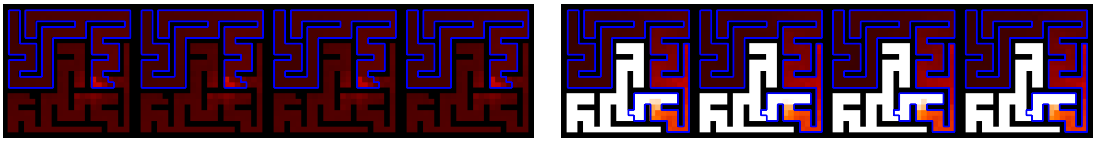


Figure 3.15: Values used during action selection for each of the 4 actions. The region in blue indicates states that have already been visited. Other colours denote Q-values between 0 (black) and 10 (white). **Left:** The Q-values used by DQN with pseudocounts. **Right:** Q^+-values used by OPIQ with $C_{\text{action}} = 100$.

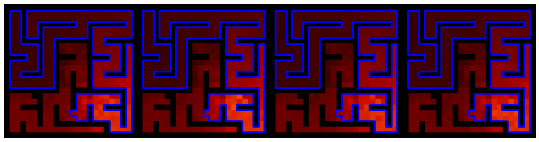


Figure 3.16: The Q^+-values OPIQ used during bootstrapping with $C_{\text{bootstrap}} = 0.01$.

Figure 3.16 shows the values used during bootstrapping for OPIQ. These Q-values show optimism near the novel state-action pairs which provides an extra incentive for the agent to return to this area of the state space.

3.9.3 Montezuma's Revenge

Finally, we consider Montezuma's Revenge, one of the hardest sparse reward games from the ALE [130]. Note that we only train up to 12.5mil timesteps (50mil frames), a 1/4 of the usual training time (50mil timesteps, 200mil frames).

Figure 3.17 shows that OPIQ significantly outperforms the baselines in terms of the episodic reward and the maximum episodic reward achieved during training. The higher episode reward and much higher maximum episode reward of OPIQ

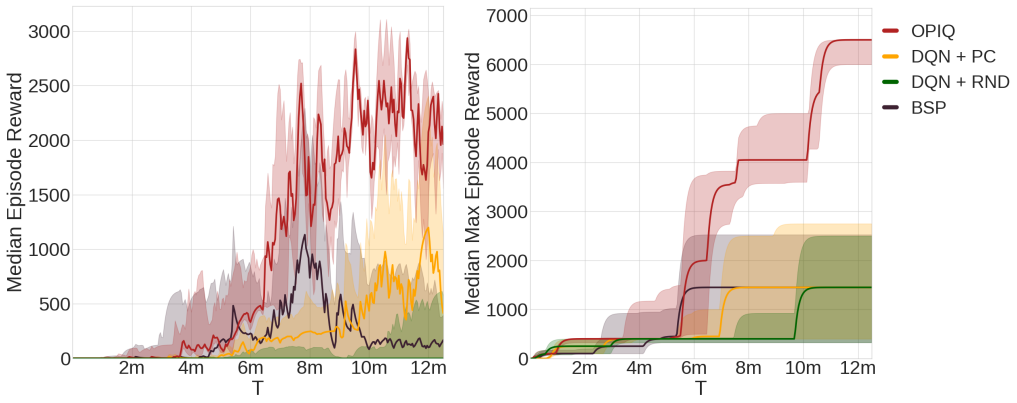


Figure 3.17: Results for Montezuma's Revenge. Median across 4 seeds is plotted and the 25%-75% quartile is shown shaded. **Left:** The episode reward. **Right:** The maximum reward achieved during an episode.

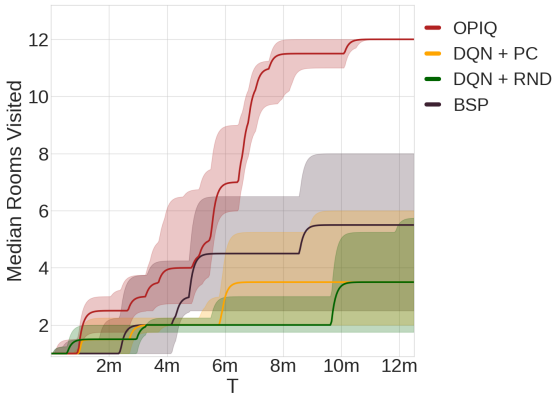


Figure 3.18: Further Results for Montezuma's Revenge showing the number of rooms visited over training comparing OPIQ and baselines. Median across 4 seeds is plotted and the 25%-75% quartile is shown shaded.

compared to DQN + PC once again demonstrates the importance of optimism during action selection and bootstrapping.

In this environment BSP performs much better than in the Maze, but achieves significantly lower episodic rewards than OPIQ.

Figure 3.18 shows the distinct number of rooms visited across the training period. We can see that OPIQ manages to reliably explore 12 rooms during the 12.5mil timesteps, significantly more than the other methods, thus demonstrating its improved exploration in this complex environment.

Our results on this challenging environment show that OPIQ can scale to high dimensional complex environments and continue to provide significant performance

improvements over an agent only using pseudocount based intrinsic rewards.

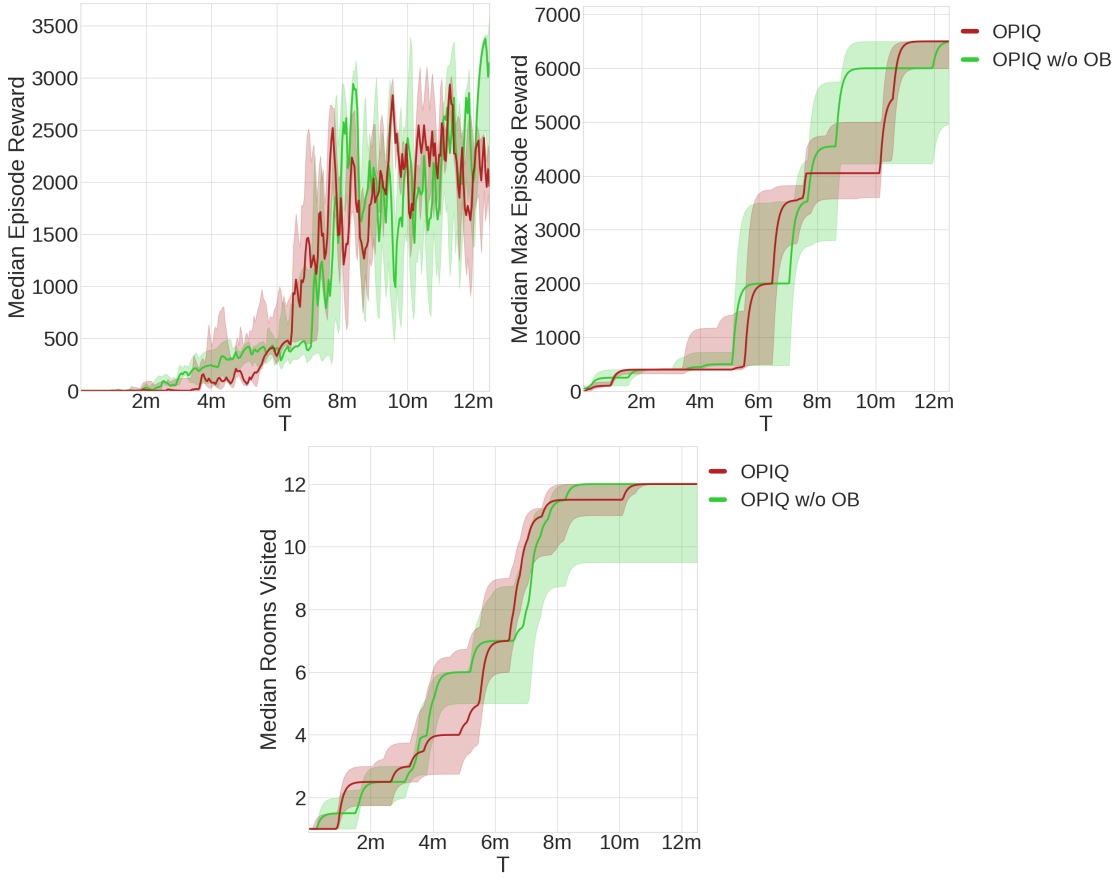


Figure 3.19: Results for Montezuma's Revenge comparing OPIQ and ablation. Median across 4 seeds is plotted and the 25%-75% quartile is shown shaded.

Figure 3.19 shows further results on Montezuma's Revenge comparing OPIQ and its ablation without optimistic bootstrapping (OPIQ w/o OB). Similarly to the Chain and Maze results, we can see that OPIQ w/o OB performs similarly to the full OPIQ but has a higher variance across seeds.

3.10 Conclusions

This chapter presented OPIQ, a model-free algorithm that does not rely on an optimistic initialisation to ensure efficient exploration. Instead, OPIQ augments the Q-values estimates with a count-based optimism bonus. We showed that this is provably efficient in the tabular setting by modifying UCB-H to use a pessimistic initialisation and our augmented Q^+-values for action selection and

bootstrapping. Since our method does not rely on a specific initialisation scheme, it easily scales to Deep RL when paired with an appropriate counting scheme. Our results showed the benefits of maintaining optimism both during action selection and bootstrapping for exploration on a number of hard sparse reward environments including Montezuma's Revenge.

Part II

Cooperative Multi-Agent Reinforcement Learning

Abstract

We now shift our attention to situations in which there are multiple agents. Specifically, we are interested in the purely cooperative multi-agent RL setting in which a team of agents must coordinate in order to maximise the team's shared return. Although exploration was the focus of the previous part of this thesis, before we can even begin to seriously tackle environments that require significant exploration in this multi-agent setting we first need appropriate algorithms. Without the ability to efficiently learn in simpler environments where exploration is not a bottleneck, we cannot reasonably expect to be able to tackle significantly more complex tasks. In Chapter 4 we explore a line of research into value function factorisation, which significantly improves an agent's ability to efficiently learn how to coordinate with their team. In particular, it allows us to employ Q-learning style algorithms without worrying about complications such as the non-stationarity of the environment dynamics from the perspective of one agent which can arise due to the multi-agent setting. This significantly improves both the sample efficiency and the performance of algorithms in this domain, particularly on our new Deep MARL benchmark: SMAC. However, QMIX, our approach to this, restricts the type of Q-values we can represent. In Chapter 5 we investigate the consequences of this and show that these restrictions can have severe negative implications on learnability and exploration. For example, QMIX can fail to recover the optimal policy in a simple 2 agent, 3 action matrix game in which it is training on all joint-actions. This is particularly alarming, since that exact training setup would be ideal in the single-agent setting. Our solution to this is to introduce a weighting into our loss so that we can change the Q-values QMIX learns to have better properties. We then investigate the utility and necessity of this weighting in relation to other algorithms in more detail.

4

Value Function Factorisation

Contents

4.1 Introduction 78
4.2 Multi-Agent Background 81
4.2.1 Dec-POMDPs 82
4.2.2 Independent Q-Learning 83
4.2.3 Value Decomposition Networks 83
4.3 QMIX 84
4.3.1 Representational Complexity 90
4.4 Two-Step Game 92
4.5 Random Matrix Games 93
4.6 The StarCraft Multi-Agent Challenge 93
4.6.1 Scenarios 97
4.6.2 State and Observations 97
4.6.3 Action Space 99
4.6.4 Rewards 99
4.6.5 Evaluation Methodology 100
4.6.6 Evaluation Metrics 100
4.7 PyMARL 101
4.8 Related Work 102
4.8.1 Independent Learners 102
4.8.2 Centralised Execution 103
4.8.3 Centralised Training and Decentralised Execution . . . 105
4.8.4 Extensions of QMIX 109
4.8.5 PyMARL and SMAC 111
4.8.6 Hypernetworks and Monotonic Networks 112
4.9 SMAC Results 112
4.10 Analysis 115
4.10.1 Role of the Central State 116
4.10.2 Role of Nonlinear Mixing 117

4.10.3 Role of RNN in Agent Network 123
4.10.4 Role of the COMA Critic Architecture 123
4.10.5 Role of Network Size and Depth 124
4.11 Conclusion 125

4.1 Introduction

Reinforcement learning (RL) holds considerable promise to help address a variety of cooperative multi-agent problems, such as coordination of robot swarms [132] and autonomous cars [133].

In many such settings, partial observability and/or communication constraints necessitate the learning of *decentralised policies*, which condition only on the local action-observation history of each agent. Fortunately, decentralised policies can often be learned in a centralised fashion in a simulated or laboratory setting. This often grants access to additional state information, otherwise hidden from agents, and removes inter-agent communication constraints. The paradigm of *centralised training with decentralised execution* [134, 135] has recently attracted attention in the RL community [136, 137]. However, many challenges surrounding how to best exploit centralised training remain open.

One of these challenges is how to represent and use the action-value function that many RL methods learn. A centralised action-value function Q_{tot} that conditions on the global state and the joint action can be difficult to learn when there are many agents and, even if it can be learned, offers no obvious way to extract decentralised policies that allow each agent to select only an individual action based on an individual observation.

The simplest option is to forgo a centralised action-value function and let each agent a learn an individual action-value function Q_a independently, as in *independent Q-learning* (IQL) [138]. However, this approach cannot explicitly represent interactions between the agents and may not converge, as each agent's learning is confounded by the learning and exploration of others.

At the other extreme, we can learn a fully centralised action-value function Q_{tot} that conditions on the global state and the joint action, and then use it to guide the optimisation of decentralised policies in an actor-critic framework, an approach taken by *counterfactual multi-agent* (COMA) policy gradients [137], as well as work by Gupta et al. [139]. However, this requires on-policy learning, which can be sample-inefficient, and training the fully centralised critic becomes impractical when there are more than a handful of agents.

In between these two extremes, we can learn a centralised but factored Q_{tot}, an approach taken by *value decomposition networks* (VDN) [140]. By representing Q_{tot} as a sum of individual value functions Q_a that condition only on individual observations and actions, a decentralised policy arises simply from each agent selecting actions greedily with respect to its Q_a. However, VDN severely limits the complexity of centralised action-value functions that can be represented and ignores any extra state information available during training.

In this chapter, we propose a new approach called QMIX which, like VDN, lies between the extremes of IQL and COMA, but can represent a much richer class of action-value functions. Key to our method is the insight that the full factorisation of VDN is not necessary to extract decentralised policies. Instead, we only need to ensure that a global arg max performed on Q_{tot} yields the same result as a set of individual arg max operations performed on each Q_a. To this end, it suffices to enforce a monotonicity constraint on the relationship between Q_{tot} and each Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a. \tag{4.1}$$

QMIX consists of *agent networks* representing each Q_a, and a *mixing network* that combines them into Q_{tot}, not as a simple sum as in VDN, but in a complex nonlinear way that ensures consistency between the centralised and decentralised policies. At the same time, it enforces the constraint of (4.1) by restricting the mixing network to have positive weights. We use *hypernetworks* [141] to condition the weights of the mixing network on the state, which is observed only during training. As a result, QMIX can represent complex centralised action-value functions with a factored



Figure 4.1: *Decentralised unit micromanagement* in StarCraft II, where each learning agent controls an individual unit. The goal is to coordinate behaviour across agents to defeat all enemy units.

representation that scales well in the number of agents and allows decentralised policies to be easily extracted via inexpensive individual argmax operations.

To evaluate QMIX, as well as the growing number of other algorithms recently proposed for multi-agent RL [137, 140], we introduce the StarCraft Multi-Agent Challenge (SMAC)[1]. In single-agent RL, standard environments such as the Arcade Learning Environment [130] and MuJoCo [142] have facilitated rapid progress. While some multi-agent testbeds have emerged, such as Poker [143], Pong [144], Keepaway Soccer [145], or simple gridworld-like environments [146, 147, 148, 149], there are currently no challenging standard testbeds for centralised training with decentralised execution with the exception of the recently introduced Hanabi Challenge [150] which focusses on a setting with less than 5 agents.

SMAC fills this gap. It is built on the popular real-time strategy game StarCraft II and makes use of the SC2LE environment [151]. Instead of tackling the full game of StarCraft with centralised control, it focuses on decentralised micromanagement challenges (Figure 4.1). In these challenges, each of our units is controlled by an independent, learning agent that has to act based only on local observations, while the opponent's units are controlled by the hand-coded built-in StarCraft II AI. SMAC offers a diverse set of scenarios that challenge algorithms to handle high-dimensional inputs and partial observability, and to learn coordinated behaviour even when restricted to fully decentralised execution. In contrast to the diverse set of scenarios included in SMAC, the Hanabi Challenge focusses on a single task

[1]Code is available at `https://github.com/oxwhirl/smac`.

involving between 2-5 agents designed to test agents' ability to reason about the actions of the other agents (an ability in humans referred to as *theory of mind* [152]).

To further facilitate research in this field, we also open-source PyMARL, a learning framework that can serve as a starting point for other researchers and includes implementations of several key multi-agent RL algorithms. PyMARL is modular, extensible, built on PyTorch, and serves as a template for dealing with some of the unique challenges of deep multi-agent RL in practice. We also offer a set of guidelines for best practices in evaluations using our benchmark, including the reporting of standardised performance metrics, sample efficiency, and computational requirements (see Section 4.6).

Our experiments on SMAC show that QMIX outperforms IQL, VDN, and COMA, both in terms of absolute performance and learning speed. In particular, our method shows considerable performance gains on the harder tasks in SMAC, and tasks with heterogeneous agents. Moreover, our analysis and ablations show both the necessity of conditioning on the state information and a flexible multi-layer network for mixing of agent Q-values in order to achieve consistent performance across tasks.

Since its introduction by Rashid et al. [153], QMIX has become an important value-based algorithm in discrete action environments. It has inspired a number of extensions and follow-up work (Section 4.8.4) and is a prominent point of comparison in any work that utilises SMAC (Section 4.8.5). Recently it is becoming increasingly common (and arguably necessary) to evaluate deep multi-agent RL algorithms on non-gridworld environments. This has been driven in part by the introduction of PyMARL and SMAC, which provide an open-source codebase and a standardised testbed for evaluating and comparing deep multi-agent RL algorithms.

4.2 Multi-Agent Background

In this section we outline the formalism we use, Dec-POMDPs [154], for all of the multi-agent work in this chapter. Please note that in this formalism the actions are denoted by u, whereas they were commonly denoted by a in the previous

chapter. Here a is used as the index for identifying a particular agent. e.g. u^a is the action u that agent a took.

4.2.1 Dec-POMDPs

A fully cooperative multi-agent sequential decision-making task can be described as a *decentralised partially observable Markov decision process* (Dec-POMDP) [154] consisting of a tuple $G = \langle S, U, P, R, Z, O, n, \gamma \rangle$.

$s \in S$ is the true state of the environment.

n is the number of agents.

At each time step, each agent $a \in A \equiv \{1, ..., n\}$ chooses an action $u^a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$.

This causes a transition on the environment according to the state transition function $P(s'|s, \mathbf{u})$.

All agents share the same reward function $R(r|s, \mathbf{u})$.

$\gamma \in [0, 1)$ is the discount factor. For episodic tasks we also require an initial state distribution S_0.

We consider a *partially observable* scenario in which each agent draws individual observations $z \in Z$ according to observation function $O(z|s, a)$. Each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a (possibly) stochastic policy $\pi^a(u^a|\tau^a)$. The joint policy π has an associated joint *action-value function*: $Q^\pi(s_t, \mathbf{u}_t) := \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t r_t | S_t = s_t, \mathbf{U}_t = \mathbf{u}_t\right]$.

Unique to our specific multi-agent domain is the setting **C**entralised **T**raining and **D**ecentralised **E**xecution (CTDE). Here we assume that the learning algorithm has access to all local action-observation histories $\boldsymbol{\tau}$, the global state s, and is able to share information freely during the centralised training phase. However, the goal of this setting is to produce agents whose policies can condition only on their own local action-observation histories τ^a. We are then interested in the performance during testing, which we refer to as decentralised execution, in which each agent must act independently in this manner.

4.2.2 Independent Q-Learning

Perhaps the most commonly applied method in multi-agent learning is *independent Q-learning* (IQL) [138], which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. This approach does not address the non-stationarity introduced due to the changing policies of the learning agents, and thus, unlike Q-learning, has no convergence guarantees even in the limit of infinite exploration. In practice, nevertheless, IQL commonly serves as a surprisingly strong baseline even in mixed and competitive games [144, 147].

Concretely, each agent performs Q-learning with a DRQN, using its own local observations and the team's reward as its reward function.

4.2.3 Value Decomposition Networks

By contrast, *value decomposition networks* (VDN) [140] aim to learn a joint action-value function $Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, where $\boldsymbol{\tau} \in \mathbf{T} \equiv \mathcal{T}^n$ is a joint action-observation history and $\mathbf{u}$ is a joint action. It represents Q_{tot} as a sum of individual value functions $Q_a(\tau^a, u^a; \theta^a)$, one for each agent a, that condition only on individual action-observation histories:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=a}^{n} Q_a(\tau^a, u^a; \theta^a). \tag{4.2}$$

Strictly speaking, each Q_a is a *utility function* [155] and not a value function since by itself it does not estimate an expected return. However, for terminological simplicity we refer to both Q_{tot} and Q_a as value functions.

The loss function for VDN is equivalent to a DQN (Equation (2.1)), where Q is replaced by Q_{tot}. An advantage of this representation, which we will discuss in more detail later in this chapter, is that a decentralised policy arises simply from each agent performing greedy action selection with respect to its Q_a.

4.3 QMIX

In this section, we propose a new approach called QMIX which, like VDN, lies between the extremes of IQL and centralised Q-learning. However, QMIX can represent a much richer class of action-value functions than VDN.

Key to our method is the insight that the full factorisation of VDN is not necessary in order to extract decentralised policies that are fully consistent with their centralised counterpart. As long as the environment is not adversarial, there exists a deterministic optimal policy conditioned on the full action-observation history. Hence, we only need to establish consistency between the deterministic greedy decentralised policies and the deterministic greedy centralised policy based on the optimal joint action-value function. When the greedy decentralised policies are determined by an arg max over the Q_a, consistency holds if a global arg max performed on Q_{tot} yields the same result as a set of individual arg max operations performed on each Q_a:

$$\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \tag{4.3}$$

This allows each agent a to participate in a decentralised execution solely by choosing greedy actions with respect to its Q_a. As a side effect, if (4.3) is satisfied, then taking the arg max of Q_{tot}, required by off-policy learning updates, is trivially tractable without an exhaustive evaluation of Q_{tot} for the exponentially many joint actions.

VDN's representation is sufficient to satisfy (4.3). However, QMIX is based on the observation that this representation can be generalised to the larger family of monotonic functions that also satisfy (4.3). Monotonicity in this context is defined as a constraint on the relationship between Q_{tot} and each Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a \in A, \tag{4.4}$$

which is sufficient to satisfy (4.3), as the following theorem shows.

Theorem 3. *If* $\forall a \in A \equiv \{1, 2, ..., n\}$, $\frac{\partial Q_{tot}}{\partial Q_a} \geq 0$ *then,*

$$\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \tag{4.3}$$

Proof. Since $\frac{\partial Q_{tot}}{\partial Q_a} \geq 0$ for $\forall a \in A$, the following holds for any $(u^1, \dots, u^n)$ and the *mixing network* function $Q_{tot}(\cdot)$ with n arguments:

$$\begin{aligned}
&Q_{tot}(Q_1(\tau^1, u^1), \dots, Q_a(\tau^a, u^a), \dots, Q_n(\tau^n, u^n)) \\
&\leq Q_{tot}(\max_{u^1} Q_1(\tau^1, u^1), \dots, Q_a(\tau^a, u^a), \dots, Q_n(\tau^n, u^n)) \\
&\dots \\
&\leq Q_{tot}(\max_{u^1} Q_1(\tau^1, u^1), \dots, \max_{u^a} Q_a(\tau^a, u^a), \dots, Q_n(\tau^n, u^n)) \\
&\dots \\
&\leq Q_{tot}(\max_{u^1} Q_1(\tau^1, u^1), \dots, \max_{u^a} Q_a(\tau^a, u^a), \dots, \max_{u^n} Q_n(\tau^n, u^n)).
\end{aligned}$$

Therefore, the maximiser of the mixing network function is:

$$(\max_{u^1} Q_1(\tau^1, u^1), \dots, \max_{u^a} Q_a(\tau^a, u^a), \dots, \max_{u^n} Q_n(\tau^n, u^n)).$$

Thus,

$$\begin{aligned}
\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) &\coloneqq \max_{\mathbf{u}=(u^1,\dots,u^n)} Q_{tot}(Q_1(\tau^1, u^1), \dots, Q_n(\tau^n, u^n)) \\
&= Q_{tot}(\max_{u^1} Q_1(\tau^1, u^1), \dots, \max_{u^n} Q_n(\tau^n, u^n)).
\end{aligned}$$

Letting $\mathbf{u}_* = (u^1_*, \dots, u^n_*) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}$, we have that:

$$\begin{aligned}
Q_{tot}(Q_1(\tau^1, u^1_*), \dots, Q_n(\tau^n, u^n_*)) &= Q_{tot}(\max_{u^1} Q_1(\tau^1, u^1), \dots, \max_{u^n} Q_n(\tau^n, u^n)) \\
&= \max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u})
\end{aligned}$$

Hence, $\mathbf{u}_* = \arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, which proves (4.3).

□

Figure 4.2 illustrates the relationship between Q_{tot} and individual Q_a functions, and how monotonicity leads to a decentralisable arg max in an example with two agents with three possible actions. Each agent a produces scores $Q_a(u_a)$ for each

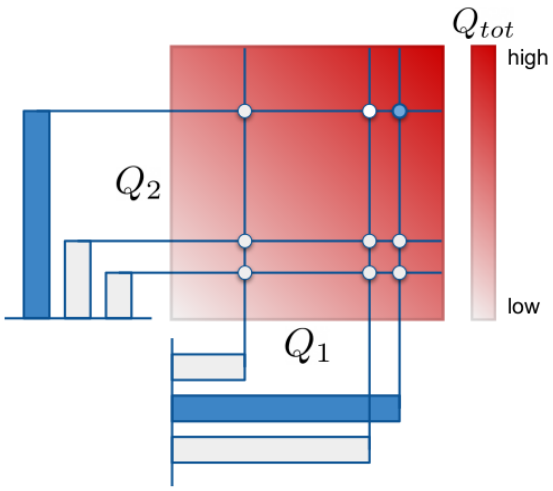


Figure 4.2: The discrete per-agent action-value scores Q_a are fed into the monotonic function $Q_{tot}(Q_1, Q_2)$. The maximum Q_a for each agent is shown in blue, which corresponds to the maximum Q_{tot} also shown in blue. The constraint (4.3) is satisfied due to the monotonicity of Q_{tot}.

discrete action that it can take (columns with the local arg max in blue). These are the inputs to a continuous monotonic mixing function $Q_{tot}(Q_1, ..., Q_N)$, the output of which is represented by the monotonically increasing heatmap. The intersections of the blue lines indicate the estimated Q_{tot} for each of the discrete joint actions. Due to the monotonicity of Q_{tot}, these joint-action value estimates maintain the ordering corresponding to each agent's Q_a, for that agent's actions when the other agents' actions remain fixed. The result is that the global greedy joint action of Q_{tot}, indicated by the blue dot, corresponds to the set of decentralised greedy actions.

We now describe how QMIX enforces (4.4) in practice, by representing Q_{tot} using an architecture consisting of *agent networks*, a *mixing network*, and a set of *hypernetworks* [141]. Figure 4.3 illustrates the overall setup.

For each agent a, there is one agent network that represents its individual value function $Q_a(\tau^a, u^a)$. We represent agent networks as DRQNs that receive the current individual observation o_t^a and the last action u_{t-1}^a as input at each time step, as shown in Figure 4.3c. We include the last actions since they are part of the action-observation history τ^a on which the decentralised policy can condition. Due to the use of stochastic policies during training, it is necessary to provide the actual action that was executed in the environment. If weights are shared across

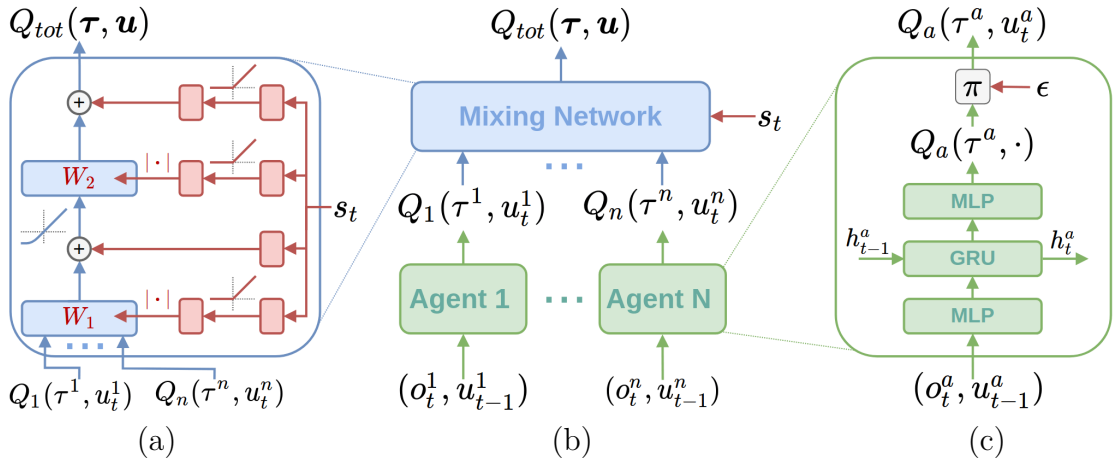


Figure 4.3: (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

the agent networks in order to speed up learning, agent IDs are included as part of the observations to allow for heterogeneous policies.

The mixing network is a feed-forward neural network that takes the agent network outputs as input and mixes them monotonically, producing the values of Q_{tot}, as shown in Figure 4.3a. To enforce the monotonicity constraint of (4.4), the weights (but not the biases) of the mixing network are restricted to be non-negative. This allows the mixing network to approximate any monotonic function arbitrarily closely in the limit of infinite width [156].

The weights of each layer of the mixing network are produced by separate hypernetworks. Each hypernetwork takes the state s as input and generates the weights of one layer of the mixing network. Each hypernetwork consists of two fully-connected layers with a ReLU nonlinearity, followed by an absolute activation function, to ensure that the mixing network weights are non-negative. The output of the hypernetwork is then a vector, which is reshaped into a matrix of appropriate size. The biases are produced in the same manner but are not restricted to being non-negative. The first bias is produced by a hypernetwork with a single linear layer, and the final bias is produced by a two-layer hypernetwork with a ReLU nonlinearity. Figure 4.3a illustrates the mixing network and the hypernetworks.

The state is used by the hypernetworks rather than being passed directly into the mixing network because Q_{tot} is allowed to depend on the extra state information in non-monotonic ways. Thus, it would be overly constraining to pass some function of s through the monotonic network alongside the per-agent values. Instead, the use of hypernetworks makes it possible to condition the weights of the monotonic network on s in an arbitrary way, thus integrating the full state s into the joint action-value estimates as flexibly as possible. The choice of nonlinearity for the mixing network is also an important consideration due to its interaction with the non-negative weights. A ReLU is not a good choice since a negative input to the mixing network is likely to remain negative (depending on the biases), which would then be zeroed out by the ReLU leading to no gradients for all agent networks. It is for this reason that we use an ELU.

QMIX is trained end-to-end to minimise the following loss:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \theta) \right)^2 \right], \tag{4.5}$$

where b is the batch size of transitions sampled from the replay buffer, the DQN target is given by $y^{tot} = r + \gamma \max_{\mathbf{u}'} Q_{tot}(\boldsymbol{\tau}', \mathbf{u}', s'; \theta^-)$, and θ^- are the parameters of a target network as in DQN. (4.5) is analogous to the standard DQN loss of (2.2). Since (4.3) holds, we can perform the maximisation of Q_{tot} in time linear in the number of agents (as opposed to scaling exponentially in the worst case).

The mixing network relies on centralised training, although it may be discarded after training to allow fully decentralised execution of the learned joint policy. In a setting with limited communication, the core algorithmic mechanism of QMIX can still be applied: the mixing network can be viewed as a manager that coordinates learning, but requires only the communication of low-dimensional action-values from, and gradients to, the decentralised agents. In a setting without communication constraints, we can further exploit the centralised training setting by, e.g., sharing parameters between agents for more efficient learning.

In Algorithm 2 we outline the pseudocode for the particular implementation of QMIX we use for all of our experiments. The choice to gather rollouts from an entire

Algorithm 2 QMIX

1: Initialise θ, the parameters of mixing network, agent networks and hypernetwork.
2: Set the learning rate α and replay buffer $\mathcal{D} = \{\}$
3: step $= 0, \theta^- = \theta$
4: **while** step $<$ step$_{max}$ **do**
5: $t = 0, s_0 =$ initial state, $o_0^a =$ initial obs, $u_{-1}^a = 0, \tau_{-1}^a = \emptyset \ \ \forall a$
6: **while** $s_t \neq terminal$ and $t <$ episode limit **do**
7: **for** each agent a **do**
8: $\tau_t^a = \tau_{t-1}^a \cup \{(o_t^a, u_{t-1}^a)\}$
9: $\epsilon =$ epsilon-schedule($step$)
10: $u_t^a = \begin{cases} \arg\max_{u_t^a} Q(\tau_t^a, u_t^a) & \text{with probability } 1-\epsilon \\ randint(1, |U|) & \text{with probability } \epsilon \end{cases}$
11: **end for**
12: Get reward r_t, next state s_{t+1}, and next observations $\mathbf{o}_{t+1}$
13: $\mathcal{D} = \mathcal{D} \cup \{(s_t, \mathbf{o}_t, \mathbf{u}_t, r_t, s_{t+1}, \mathbf{o}_{t+1})\}$
14: $t = t + 1$, step $=$ step $+ 1$
15: **end while**
16: **if** $|\mathcal{D}| >$ batch-size **then**
17: b $\leftarrow$ random batch of episodes from $\mathcal{D}$
18: **for** each timestep t in each episode in batch b **do**
19: $Q_{tot} =$ *Mixing-network*$(Q_1(\tau_t^1, u_t^1), \ldots, Q_n(\tau_t^n, u_t^n);$ *Hypernetwork*$(s_t; \theta))$
20: Calculate target Q_{tot} using *Mixing-network* with *Hypernetwork*$(s_t; \theta^-))$
21: **end for**
22: $\Delta Q_{tot} = y^{tot} - Q_{tot}$ // Eq (4.5)
23: $\Delta\theta = \nabla_\theta (\Delta Q_{tot})^2$
24: $\theta = \theta - \alpha \Delta\theta$
25: **end if**
26: **if** update-interval steps have passed **then**
27: $\theta^- = \theta$
28: **end if**
29: **end while**

episode (line 17) before executing a single gradient descent step (line 24), as well as using per-agent ϵ-greedy action selection (line 10), is not a requirement for QMIX but is an implementation detail. However, the action selection based solely on agent network's Q-values (line 10), as well as calculation of Q_{tot} and its target using the mixing network and hypernetworks (lines 19-20) are essential features of QMIX. The update of per-agent action-value functions and hypernetworks using the DQN-style loss with respect to Q_{tot} and joint reward (lines 22-24) is also central to our method.

4.3.1 Representational Complexity

The value function class representable with QMIX includes any value function that can be factored into a nonlinear monotonic combination of the agents' individual value functions in the fully observable setting.

This follows since the mixing network is a universal function approximator of monotonic functions [156], and hence can represent any value function that factors into a nonlinear monotonic combination of the agent's individual value functions. Additionally, we require that the agent's individual value functions order the values of the actions appropriately. By this we mean that Q_a is such that $Q_a(s_t, u^a) > Q_a(s_t, u'^a) \iff Q_{tot}(s_t, (\mathbf{u}^{-a}, u^a)) > Q_{tot}(s_t, (\mathbf{u}^{-a}, u'^a))$, i.e., they can represent a function that respects the ordering of the agent's actions in the joint-action value function. Since the agents' networks are universal function approximators [157], they can represent such a Q_a. Hence QMIX is able to represent any value function that factors into a nonlinear monotonic combination of the agent's individual value functions.

In a Dec-POMDP, QMIX cannot necessarily represent this same set of value functions due to the partial observability. This is because each agent's observations are no longer the full state, and thus they might not be able to distinguish the true state given their local observations. If the agent's value function ordering is then wrong, i.e., $Q_a(\tau^a, u) > Q_a(\tau^a, u')$ when $Q_{tot}(s_t, (\mathbf{u}^{-a}, u)) < Q_{tot}(s_t, (\mathbf{u}^{-a}, u'))$, then the mixing network would be unable to correctly represent Q_{tot} given the monotonicity constraints.

QMIX thus expands upon the linear monotonic value functions that are representable by VDN. Table 4.1a gives an example of a monotonic value function for the simple case of a two-agent matrix game. Note that VDN is unable to represent this simple monotonic value function. Table 4.2a provides the optimal Q_{tot} values for VDN, minimising a squared loss. Section 4.4 additionally provides results for the values that VDN learns in a function approximation setting.

However, the constraint in (4.4) prevents QMIX from representing value functions that do not factorise in such a manner. A simple example of such a value function for

Agent 1 \ Agent 2	A	B
A	0	1
B	1	8

(a)

Agent 1 \ Agent 2	A	B
A	2	1
B	1	8

(b)

Table 4.1: (a) An example of a monotonic payoff matrix, (b) a nonmonotonic payoff matrix.

Agent 1 \ Agent 2	A	B
A	-1.5	2.5
B	2.5	6.5

(a)

Agent 1 \ Agent 2	A	B
A	4/3	4/3
B	4/3	8

(b)

Table 4.2: (a) Optimal Q_{tot} values for VDN on the example in Table 4.1a. (b) Optimal Q_{tot} values for QMIX on the example in Table 4.1b.

a two-agent matrix game is given in Table 4.1b. Intuitively, any value function for which an agent's best action depends on the actions of the other agents *at the same time step* will not factorise appropriately, and hence cannot be perfectly represented by QMIX. Table 4.2b provides the optimal Q_{tot} values for QMIX when minimising a squared loss. For the example in Table 4.1b, QMIX still recovers the optimal policy and learns the correct maximum over the Q-values. Formally, the class of value functions that cannot be represented by QMIX are called *nonmonotonic* [158].

Although QMIX cannot perfectly represent all value functions, its increased representational capacity allows it to learn Q-value estimates that are more accurate for computing the target values to regress onto during Q-learning targets (bootstrapping). In particular, we show in Section 4.4 that the increased representational capacity allows for QMIX to learn the optimal policy in an environment with no per-timestep coordination, compared to VDN which still fails to learns the optimal policy. In Section 4.5 we show that QMIX learns significantly more accurate maxima over Q_{tot} in randomly generated matrix games, demonstrating that it learns better Q-value estimates for bootstrapping. Finally, we show in Section 4.9 that QMIX significantly outperforms VDN on the challenging SMAC benchmark.

4.4 Two-Step Game

To illustrate the effects of representational complexity of VDN and QMIX, we devise a simple two-step cooperative matrix game for two agents.

At the first step, Agent 1 chooses which of the two matrix games to play in the next timestep. For the first time step, the actions of Agent 2 have no effect. In the second step, both agents choose an action and receive a global reward according to the payoff matrices depicted in Table 4.3.

We train VDN and QMIX on this task for 5000 episodes and examine the final learned value functions in the limit of full exploration ($\epsilon = 1$). Full exploration ensures that each method is guaranteed to eventually explore all available game states, such that the representational capacity of the state-action value function approximation remains the only limitation. The full details of the architecture and hyperparameters used, as well as additional results are provided in Appendix B.1.

		Agent 2	
		A	*B*
Agent 1	*A*	7	7
	B	7	7

State 2A

		Agent 2	
		A	*B*
Agent 1	*A*	0	1
	B	1	8

State 2B

Table 4.3: Payoff matrices of the two-step game after the Agent 1 chose the first action. Action A takes the agents to State 2A and action B takes them to State 2B.

(a)

	State 1		State 2A		State 2B	
	A	*B*	*A*	*B*	*A*	*B*
A	6.94	6.94	6.99	7.02	-1.87	2.31
B	6.35	6.36	6.99	7.02	2.33	6.51

(b)

	A	*B*	*A*	*B*	*A*	*B*
A	6.93	6.93	7.00	7.00	0.00	1.00
B	7.92	7.92	7.00	7.00	1.00	8.00

Table 4.4: Q_{tot} on the two-step game for (a) VDN and (b) QMIX.

Table 4.4, which shows the learned values for Q_{tot}, demonstrates that QMIX's higher representational capacity allows it to accurately represent the joint-action

value function whereas VDN cannot. This directly translates into VDN learning the suboptimal strategy of selecting Action A at the first step and receiving a reward of 7, whereas QMIX recovers the optimal strategy from its learnt joint-action values and receives a reward of 8.

The simple example presented in this section demonstrates the importance of accurate Q-value estimates, especially for the purpose of bootstrapping. In Section 4.5 we provide further evidence that QMIX's increased representational capacity allows it to learn more accurate Q-value, that directly lead to more accurate bootstrapped estimates.

4.5 Random Matrix Games

To demonstrate that QMIX learns more accurate Q-values than VDN for the purpose of bootstrapping, we compare the learnt values of $\max_{\mathbf{u}} Q_{tot}(s, \mathbf{u})$ on randomly generated matrix games. Single-step matrix games allow us to compare the learnt maximum Q-values across both methods in isolation. We focus on $\max_{\mathbf{u}} Q_{tot}(s, \mathbf{u})$ because it is the only learnt quantity involved in the bootstrapping for 1-step Q-learning.

The maximum return for each matrix is 10, and all other values are drawn uniformly from $[0, 10)$. Each individual seed has different values for the payoff matrix, but within each run they remain fixed. We set $\epsilon = 1$ to ensure all actions are sampled and trained on equally.

Figure 4.4 shows the results for a varying number of agents and actions. We can see that QMIX learns significantly more accurate maxima than VDN due to its larger representational capacity.

4.6 The StarCraft Multi-Agent Challenge

In this section, we describe the StarCraft Multi-Agent Challenge (SMAC) to which we apply QMIX and a number of other methods. SMAC is based on the popular real-time strategy (RTS) game StarCraft II. In a regular full game of StarCraft II, one

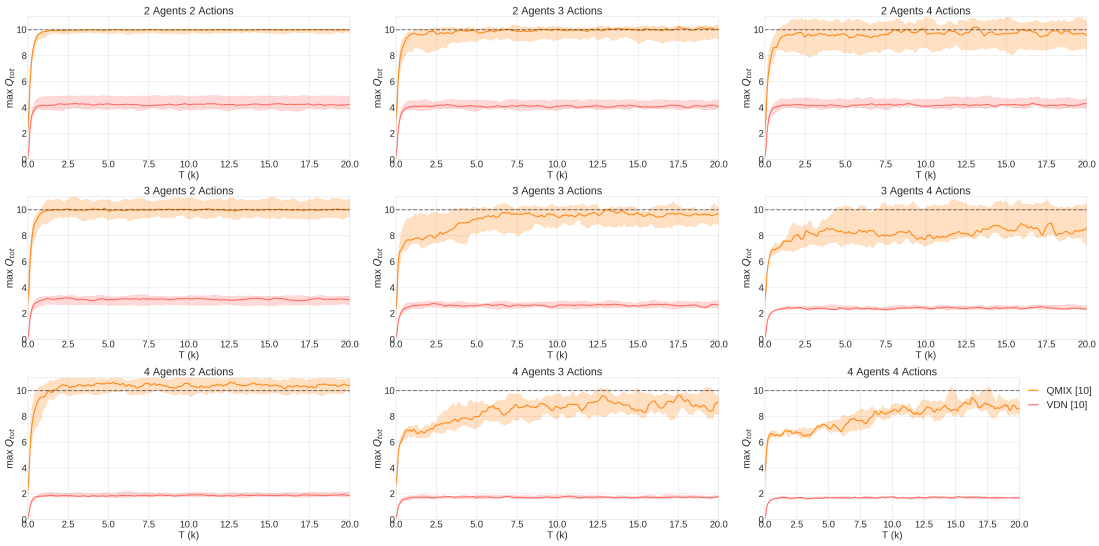


Figure 4.4: The median $\max_{\mathbf{u}} Q_{tot}(s, \mathbf{u})$ for $\{2, 3, 4\}$ agents with $\{2, 3, 4\}$ actions across 10 runs for VDN and QMIX. 25%-75% quartile is shown shaded. The dashed line at 10 indicates the correct value.

or more humans compete against each other or against a built-in game AI to gather resources, construct buildings, and build armies of units to defeat their opponents.

Akin to most RTSs, StarCraft has two main gameplay components: macromanagement and micromanagement. *Macromanagement* refers to high-level strategic considerations, such as economy and resource management. *Micromanagement* (micro), on the other hand, refers to fine-grained control of individual units.

StarCraft has been used as a research platform for AI, and more recently, RL. Typically, the game is framed as a competitive problem: an agent takes the role of a human player, making macromanagement decisions and performing micromanagement as a puppeteer that issues orders to individual units from a centralised controller.

In order to build a rich multi-agent testbed, we instead focus solely on micromanagement. Micro is a vital aspect of StarCraft gameplay with a high skill ceiling, and is practiced in isolation by amateur and professional players. For SMAC, we leverage the natural multi-agent structure of micromanagement by proposing a modified version of the problem designed specifically for decentralised control. In particular, we require that each unit be controlled by an independent agent that conditions only on local observations restricted to a limited field of view centred

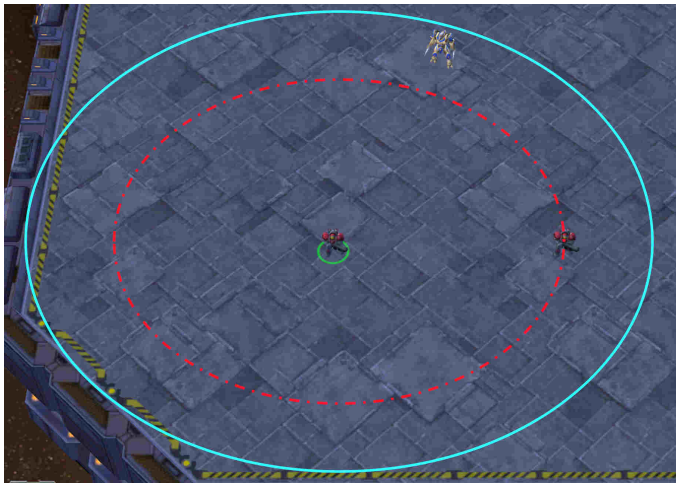


Figure 4.5: The cyan and red circles respectively border the sight and shooting range of the agent.



Figure 4.6: Screenshots of two SMAC scenarios.

on that unit (see Figure 4.5). Groups of these agents must be trained to solve challenging combat scenarios, battling an opposing army under the centralised control of the game's built-in scripted AI.

Proper micro of units during battles maximises the damage dealt to enemy units while minimising damage received, and requires a range of skills. For example, one important technique is *focus fire*, i.e., ordering units to jointly attack and kill enemy units one after another. When focusing fire, it is important to avoid *overkill*: inflicting more damage to units than is necessary to kill them. Other common micro techniques include: assembling units into formations based on their armour types, making enemy units give chase while maintaining enough distance so that little or no damage is incurred (*kiting*, Figure 4.1a), coordinating the positioning of units to attack from different directions or taking advantage of the terrain to defeat the enemy.

SMAC thus provides a convenient environment for evaluating the effectiveness of MARL algorithms. The simulated StarCraft II environment and carefully designed scenarios require learning rich cooperative behaviours under partial observability, which is a challenging task. The simulated environment also provides an additional state information during training, such as information on all the units on the entire map. This is crucial for facilitating algorithms to take full advantage of the centralised training regime and assessing all aspects of MARL methods.

SMAC features the following characteristics that are common in many real-word multi-agent systems:

- Partial observability: Like, e.g., self-driving cars and autonomous drones, agents in SMAC receive only limited information about the environment.
- Large number of agents: the scenarios in SMAC include up to 27 learning agents.
- Diversity: many scenarios feature heterogeneous units with resulting diversity in optimal strategies.
- Long-term planning: defeating the enemy in SMAC often requires the agents to perform a long sequence of actions.
- High-dimensional observation spaces: the inclusion of diverse units and the large size of the map yield an immense state space.
- Large per-agent action space: the agents can perform up to 70 actions at each time step.
- Coordinated teamwork: micromanagement of units requires the individual agents to execute strictly coordinated actions, which is essential to many MARL problems.
- Stochasticity: the behaviour of the enemy differs across individual runs. Also, the amount of time that the agents must wait until being able to shoot again is stochastic.

SMAC is one of the first MARL benchmarks that includes all of these features, making it useful for evaluating the effectiveness of methods for many aspects of

learning decentralised multi-agent control. We hope that it will become a standard benchmark for measuring the progress and a grand challenge for pushing the boundaries of MARL.

4.6.1 Scenarios

SMAC consists of a set of StarCraft II micro scenarios which aim to evaluate how well independent agents are able to learn coordination to solve complex tasks. These scenarios are carefully designed to necessitate the learning of one or more micro techniques to defeat the enemy. Each scenario is a confrontation between two armies of units. The initial position, number, and type of units in each army varies from scenario to scenario, as does the presence or absence of elevated or impassable terrain. Figures 4.1 and 4.6 include screenshots of several SMAC micro scenarios.

The first army is controlled by the learned allied agents. The second army consists of enemy units controlled by the built-in game AI, which uses carefully handcrafted non-learned heuristics. At the beginning of each episode, the game AI instructs its units to attack the allied agents using its scripted strategies. An episode ends when all units of either army have died or when a pre-specified time limit is reached (in which case the game is counted as a defeat for the allied agents). The goal is to maximise the win rate, i.e., the ratio of games won to games played.

The complete list of challenges is presented in Table 4.5. More specifics on the SMAC scenarios and environment settings can be found in Appendices B.3.1 and B.3.2 respectively.

4.6.2 State and Observations

At each timestep, agents receive local observations drawn within their field of view. This encompasses information about the map within a circular area around each unit and with a radius equal to the *sight range* (Figure 4.5). The sight range makes the environment partially observable from the standpoint of each agent. Agents can only observe other agents if they are both alive and located within the sight

Name	Ally Units	Enemy Units
2s3z	2 Stalkers & 3 Zealots	2 Stalkers & 3 Zealots
3s5z	3 Stalkers & 5 Zealots	3 Stalkers & 5 Zealots
1c3s5z	1 Colossus, 3 Stalkers & 5 Zealots	1 Colossus, 3 Stalkers & 5 Zealots
5m_vs_6m	5 Marines	6 Marines
10m_vs_11m	10 Marines	11 Marines
27m_vs_30m	27 Marines	30 Marines
3s5z_vs_3s6z	3 Stalkers & 5 Zealots	3 Stalkers & 6 Zealots
MMM2	1 Medivac, 2 Marauders & 7 Marines	1 Medivac, 3 Marauders & 8 Marines
2s_vs_1sc	2 Stalkers	1 Spine Crawler
3s_vs_5z	3 Stalkers	5 Zealots
6h_vs_8z	6 Hydralisks	8 Zealots
bane_vs_bane	20 Zerglings & 4 Banelings	20 Zerglings & 4 Banelings
2c_vs_64zg	2 Colossi	64 Zerglings
corridor	6 Zealots	24 Zerglings

Table 4.5: SMAC challenges. Note that list of SMAC scenarios has been updated from the earlier version. All scenarios, however, are still available in the repository.

range. Hence, there is no way for agents to distinguish between teammates that are far away from those that are dead.

The feature vector observed by each agent contains the following attributes for both allied and enemy units within the sight range: `distance`, `relative x`, `relative y`, `health`, `shield`, and `unit_type`. Shields serve as an additional source of protection that needs to be removed before any damage can be done to the health of units. All Protoss units have shields, which can regenerate if no new damage is dealt. In addition, agents have access to the last actions of allied units that are in the field of view. Lastly, agents can observe the terrain features surrounding them, in particular, the values of eight points at a fixed radius indicating height and walkability.

The global state, which is only available to agents during centralised training, contains information about all units on the map. Specifically, the state vector includes the coordinates of all agents relative to the centre of the map, together with unit features present in the observations. Additionally, the state stores the `energy` of Medivacs and `cooldown` of the rest of the allied units, which represents the minimum delay between attacks. Finally, the last actions of all agents are attached to the central state.

All features, both in the state as well as in the observations of individual agents, are normalised by their maximum values. The sight range is set to nine for all agents. The feature vectors for both local observations and global state have a fixed size, where information about the ally/enemy is represented as a sequence ordered by the unit ID number.

4.6.3 Action Space

The discrete set of actions that agents are allowed to take consists of `move[direction]`[2], `attack[enemy_id]`, `stop` and `no-op`.[3] As healer units, Medivacs must use `heal[agent_id]` actions instead of `attack[enemy_id]`. The maximum number of actions an agent can take ranges between 7 and 70, depending on the scenario.

To ensure decentralisation of the task, agents can use the `attack[enemy_id]` action only on enemies in their *shooting range* (Figure 4.5). This additionally constrains the ability of the units to use the built-in *attack-move* macro-actions on enemies that are far away. We set the shooting range to six for all agents. Having a larger sight range than a shooting range forces agents to use move commands before starting to fire.

To ensure that agents only execute valid actions, a mask is provided indicating which actions are valid and invalid at each timestep.

4.6.4 Rewards

The overall goal is to maximise the win rate for each battle scenario. The default setting is to use the *shaped reward*, which produces a reward based on the hit-point damage dealt and enemy units killed, together with a special bonus for winning the battle. The exact values and scales for each of these events can be configured using a range of flags. To produce fair comparisons we encourage using this default reward function for all scenarios. We also provide another *sparse reward* option, in

[2]Four directions: north, south, east, or west.
[3]Dead agents can only take `no-op` action while live agents cannot.

which the reward is +1 for winning and -1 for losing an episode which is significantly harder setting to make progress with.

4.6.5 Evaluation Methodology

To ensure the fairness of the challenge and comparability of results, performance should be evaluated under standardised conditions. One should not undertake any changes to the environment used for evaluating the policies. This includes the observation and state spaces, action space, the game mechanics, and settings of the environment (e.g., frame-skipping rate). One should not modify the StarCraft II map files in any way or change the difficulty of the game AI. Episode limits of each scenario should also remain unchanged.

SMAC restricts the execution of the trained models to be decentralised, i.e., during testing each agent must base its policy solely on its own action-observation history and cannot use the global state or the observations of other agents. It is, however, acceptable to train the decentralised policies in centralised fashion. Specifically, agents can exchange individual observations, model parameters and gradients during training as well as make use of the global state.

4.6.6 Evaluation Metrics

Our main evaluation metric is the mean win percentage of evaluation episodes as a function of environment steps observed, over the course of training. Such progress can be estimated by periodically running a fixed number of evaluation episodes (in practice, 32) with any exploratory behaviours disabled. Each experiment is repeated using a number of independent training runs and the resulting plots include the median performance as well as the 25-75% percentiles. We use five independent runs for this purpose in order to strike a balance between statistical significance and the computational requirements. We recommend using the median instead of the mean in order to avoid the effect of any outliers. We report the number of independent runs, as well as environment steps used in training.

We also include the computational resources used, as well as the wall clock time for running each experiment. SMAC provides functionality for saving StarCraft II replays, which can be viewed using a freely available client. The resulting videos can be used to comment on interesting behaviours observed. Each independent run takes between 8 to 16 hours, depending on the exact scenario, using Nvidia Geforce GTX 1080 Ti graphics cards.

4.7 PyMARL

To make it easier to develop algorithms for SMAC, we have also open-sourced our software engineering framework PyMARL[4]. Compared to many other frameworks for deep reinforcement learning [159, 160, 161], PyMARL is designed with a focus on the multi-agent setting. It is also intended for fast, easy development and experimentation for researchers, relying on fewer abstractions than e.g., RLLib [162]. To maintain this lightweight development experience, PyMARL does not support distributed training.

PyMARL's codebase is organized in a modular fashion in order to enable the rapid development of new algorithms, as well as provide implementations of current Deep MARL algorithms to benchmark against. It is built on top of PyTorch to facilitate the fast execution and training of deep neural networks, and take advantage of the rich ecosystem built around it. PyMARL's modularity makes it easy to extend, and components can be readily isolated for testing purposes.

Since the implementation and development of Deep MARL algorithms come with a number of additional challenges beyond those posed by single-agent Deep RL, it is crucial to have simple and understandable code. In order to improve the readability of code and simplify the handling of data between components, PyMARL encapsulates all data stored in the buffer within an easy to use data structure. This encapsulation provides a cleaner interface for the necessary handling of data in Deep MARL algorithms, whilst not obstructing the manipulation of the underlying PyTorch Tensors. In addition, PyMARL aims to maximise the

[4]PyMARL is available at `https://github.com/oxwhirl/pymarl`.

batching of data when performing inference or learning so as to provide significant speed-ups over more naive implementations.

PyMARL features implementations of the following algorithms: QMIX, QTRAN [163], COMA [137], VDN [140], and IQL [138] as baselines.

4.8 Related Work

Recent work in multi-agent RL has started moving from tabular methods [164, 165] to deep learning methods that can tackle high-dimensional state and action spaces [144, 137, 166, 167]. In this part of the thesis, we focus on the fully-cooperative setting in which all agents must maximise a joint reward signal in the paradigm of centralised training and decentralised execution.

4.8.1 Independent Learners

A natural approach to producing decentraliseable agents in a multi-agent system is to directly learn decentralised value functions or policies. *Independent Q-learning* [138] trains independent action-value functions for each agent using Q-learning [168]. Tampuu et al. [144] extend this approach to deep neural networks using DQN [24]. While trivially achieving decentralisation, these approaches are prone to instability arising from the non-stationarity of the environment induced by simultaneously learning and exploring agents. Foerster et al. [169] addresses the issue of non-stationarity when using an experience replay with independent learners to some extent. Lauer and Riedmiller [170] ignore updates which decrease Q-value estimates in order to not prematurely underestimate an action's Q-value due to the exploratory actions of other agents. In a tabular setting, they prove this converges to the optimal policy in deterministic environments, provided there is a unique optimal joint-action. Matignon et al. [171] introduce *Hysteretic Q-learning* which instead uses a smaller learning rate for decreasing Q-value estimates, which is slightly more robust to stochasticity and the presence of multiple optimal joint actions. Omidshafiei et al. [172] utilises Hysteretic Q-learning in a multi-task Deep RL setting. Panait et al. [173] introduce *Leniency* which ignores updates that

decrease Q-value estimates with a probability that is decreasing during training. Wei and Luke [174] show that Leniency outperforms Hysteretic Q-learning in cooperative stochastic games, and Palmer et al. [175] extend Leniency to the Deep RL setting and show benefits over Hysteretic and fully Independent Q-learners on deterministic and stochastic gridworlds with two agents. Palmer et al. [176] maintain a learned interval whose lower bound is approximately the minimum cumulative reward received when all agents are coordinating. They use this interval when decreasing Q-value estimates to distinguish between miscoordination between the agents and the stochasticity of the environment, which Hysteretic and Lenient learners do not distinguish between. Lu and Amato [177] utilise Distributional RL [178] in combination with Hysteretic Q-learning in order to also distinguish between miscoordination and stochasticity, and argue their approach is more stable and robust to hyperparameters than the above methods. All of these approaches do not utilise extra state information available during a centralised training regime, nor do they attempt to learn joint action-value functions.

4.8.2 Centralised Execution

In settings where decentralised execution is not mandatory, the centralised learning of joint action-value function naturally handles the coordination problems and avoids the non-stationarity problem. Nonetheless, a centralised action-value function is impractical to scale since the joint action space grows exponentially in the number of agents. Classical approaches to scalable centralised learning include *coordination graphs* [155], which exploit conditional independencies between agents by decomposing a global reward function into a sum of agent-local terms. This can significantly reduce the computation required in order to find the maximum joint action (depending on the exact structure of the coordination graph supplied) through message passing or variable elimination algorithms. However, specifying an appropriate coordination graph (i.e. not fully-connected) can require significant domain knowledge. *Sparse cooperative Q-learning* [179] is a tabular Q-learning algorithm that learns to coordinate the actions of a group of cooperative agents only

in the states in which such coordination is necessary, encoding those dependencies in a coordination graph. Both methods require the dependencies between agents to be pre-supplied, whereas we do not require such prior knowledge. Castellini et al. [180] investigate the representational capacity of coordination graphs in a Deep RL setting using one-step matrix games. Boehmer et al. [181] also extend coordination graphs to the Deep RL setting, making use of parameter sharing and limiting the graphs to contain only pairwise edges, in order to consider more complex scenarios. Zhou et al. [182] similarly factor the joint Q-function into pairwise interaction terms, performing maximisation using coordinate ascent instead of message-passing. QMIX (and VDN) correspond to the case of a degenerate fully disconnected coordination graph, thus enabling fully decentralised execution.

Jiang et al. [183] utilise a Graph Convolutional Network in order to spread or communicate information between IQL agents. This graph network based approach is quite different to the use of coordination graphs, since the graph network is only used to spread information between agents, thereby changing the data they can condition their output on. Ultimately, each agent is still outputting individual Q-values/utilities, which Jiang et al. [183] train through independent Q-learning. Qu et al. [184] extend this approach to an multi-agent SAC setup, in which the critic utilises a graph convolutional network in the same way, except it represents the joint-action Q-value as the sum of the appropriate utilities as in VDN. The agents also utilise a graph convolutional network, and are trained using the policy-gradient on their individual policies.

DIAL [185] utilises an end-to-end differentiable architecture that allows for a learned inter-agent communication to emerge via backpropagation. CommNet [186] use a centralised network architecture to exchange information between agents. BicNet [166] utilise bidirectional RNNs for inter-agent communication in an actor-critic setting and additionally requires estimating individual agent rewards. MAGNet [187] allow agents to centrally learn a shared graph structure that encodes the relevance of individual environment entities to each agent. Based on this graph, agents can communicate and coordinate their actions during centralised execution.

Zhao and Ma [188] show that the performance of decentralised policies can be improved upon through a centralised communication scheme that uses a central information-filtering neural network in between timesteps. Clustered Deep Q-Networks (CDQN) [189] use a hierarchical approach in order to scale learning to more agents and larger joint action spaces. Low-level agents are clustered into groups, each of which is managed by a higher-level agent within a centralised execution setting. All low-level agents within a cluster execute the same action selected by a vote, which alleviates some of the non-stationarity of the independent learning setting. In contrast to these approaches, QMIX does not require any form of inter-agent communication during decentralised execution.

4.8.3 Centralised Training and Decentralised Execution

A number of methods have developed hybrid approaches that exploit the centralised training opportunity for training fully decentralised policies. Gupta et al. [139] present a centralised actor-critic algorithm with per-agent critics, which scales easily with the number of agents. Similarly, Lowe et al. [146] present MADDPG, which learns a centralised critic for each agent and apply this to competitive games with continuous action spaces. In contrast to Gupta et al. [139], MADDPG takes advantage of the centralised training paradigm by incorporating information from the other agents into the critics, at the expense of scalability due to the increased input size. COMA [137] instead uses a single centralised critic to train decentralised actors in the fully-cooperative setting, estimating a counterfactual advantage function for each agent in order to address multi-agent credit assignment. Iqbal and Sha [190] devise an actor-critic algorithm with per-agent critics that share an attention mechanism in order to improve scalability compared to a single centralised critic. Their approach allows for the agents to have differing action spaces as well allowing for individual rewards. The authors do not compare against QMIX and do not evaluate on environments of similar complexity as StarCraft. Witt et al. [167] construct a multi-agent actor-critic algorithm with a hierarchical actor policy that is able to use common knowledge between agents for coordination.

Learning Individual Intrinsic Reward (LIIR) [191] learns an intrinsic reward function for each agent to supplement the team reward. The intrinsic reward function is trained such that it maximises the team reward function. As a bi-level optimisation process, LIIR is significantly less computationally efficient than QMIX and has not been demonstrated to scale to large numbers of agents. These approaches are all actor-critic variants which use the policy-gradient theorem, and thus are prone to get stuck sub-optimal local minima. Additionally, all but MADDPG are on-policy algorithms which can have poor sample efficiency compared to off-policy algorithms such as QMIX. Lyu et al. [192] demonstrate that merely using a centralised critic does not necessarily lead to better results when updating decentralised agents with a stochastic policy-gradient in an on-policy algorithm. This is because, in expectation, the policy-gradient updates are the same for a centralised critic and for decentralised critics. However, the variance of the centralised critic's policy-gradient can be substantially larger since the decentralised critics have effectively already marginalised over the other agent's actions when learning their Q-values. Additionally, learning a centralised critic which conditions on the state only for training actors which can only condition on partial observations can be problematic both theoretically and empirically [193]. Peng et al. [194] extends MADDPG by taking advantage of the additional structure present in our CTDE cooperative MARL setting and resolve some of these shortcomings related to utilising a centralised critic. This is achieved by learning a single critic, which is factored in a manner similar to QMIX, that enables it to perform well on complex tasks as such those in SMAC. Since the critic can be trained off-policy, there is an immediate benefit to utilising a centralised critic otherwise we would have issues related to non-stationarity when training a decentralised critic on old data. Additionally, a centralised policy-gradient estimator is proposed, which optimises in the space of joint-actions so as to avoid sub-optimal local minima in which no one agent wishes to deviate but there is a path for improvement if 2 or more agents deviate. Importantly, since FACMAC uses a deterministic policy, and a deterministic policy-gradient, there is a clear difference between utilising a per-agent update and a joint-action update. In order to sensibly

update in the space of joint-actions, FACMAC then requires a centralised critic to provide an appropriate gradient direction. Zhou et al. [195] also adapt MADDPG by learning a single critic, whose architecture is the same as QMIX's mixing network without monotonicity, but do not train off-policy and do not optimise the joint-action policy as a whole. DOP [196] note that the joint-action policy gradient is the same as the per-agent policy gradient if you are using a linearly factorised critic (with non-negative weights, so that it is monotonic). Thus, they linearly factorise the critic in this manner and train each agent with the appropriate per-agent policy-gradient. Similarly to FACMAC, this results in the correct joint-action gradient being followed, however it restricts the representational capability of the critics significantly. In contrast, FACMAC has no such restrictions, but finds that empirically a restricted QMIX style critic performs best.

Lin et al. [197] train a centralised action-value function Q which has access to observations of all agent. They then train the decentralised agents via supervised learning to mimic the actions of the centralised policy. Distilling a centralised policy into decentralised agent policies can be problematic since the centralised policy conditions on more information than any of the agents' policies. QMIX instead learns a factored joint Q-function that can be easily decentralised due to its architecture. Sunehag et al. [140] propose *value decomposition networks* (VDN), which allow for centralised value function learning with decentralised execution. Their algorithm decomposes a central action-value function into a sum of individual agent terms. VDN does not make use of additional state information during training and can represent only a limited class of centralised action-value functions. Son et al. [163] introduce QTRAN which learns a centralised, unrestricted, joint Q-function as well as a VDN-factored joint Q-function that is decentralisable. Since the unrestricted Q-value cannot be maximised efficiently, the VDN-factored Q is used to produce the (approximate) maximum joint-action. They show that solving a linear optimisation problem involving all joint actions, in which the VDN-factored Q-value matches the unrestricted Q-value for the approximated maximum joint action and over-estimates for every other action, results in decentralisable policies

with the correct argmax. This allows QTRAN to represent a much larger class of joint Q-function than QMIX, as well as produce decentralisable policies for them. However, exactly solving the linear optimisation problem is prohibitively expensive. Thus, the authors instead optimise a soft approximation using L_2 penalties via stochastic gradient descent. In practice, optimising this loss is difficult which results in poor performance for QTRAN on complex environments such as SMAC.

Yang et al. [198] utilise integrated gradients to decompose the joint-action Q-values of the critic into individual utilities for each agent, a form of multi-agent credit assignment. The agents are then regressed against their respective utilities. Böhmer et al. [199] learn a centralised joint action Q-Value that is approximately maximised by coordinate descent and used to generate trajectories that IQL agents train on. Iqbal and Sha [200] uses a variant of multi-agent SAC that learns multiple policy and critic heads trained on different intrinsic rewards, which are then chosen based on their performance with respect to the extrinsic reward.

Gupta et al. [201] utilise successor features in combination with VDN in order to enable for better exploration and learning in environments in which we can adjust the reward function appropriately. They demonstrate that this can help in overcoming limitations of VDN and QMIX related to their limited representational capacity, by first learning with an *easier* reward function to learn good behaviour that can be prohibitively hard to learn with the *harder* reward function.

Hu and Foerster [202] simplify the Bayesian Action Decoder [203] by augmenting the observations of VDN agents with the greedy action of the other agent (that they may not have executed due to exploration), and demonstrate improved performance in Hanabi [150].

[204, 205, 206] utilise attention based layers in order to enable efficient transfer learning in scenarios such as SMAC where the number of agents/actions vary across tasks.

4.8.4 Extensions of QMIX

Mahajan et al. [158] show that the representational constraints of QMIX can prohibit it from learning an optimal policy. Importantly, they show that this is exacerbated by increased ϵ-greedy exploration. They introduce MAVEN, which conditions QMIX agents on a shared latent space whose value is chosen at the beginning of an episode by a hierarchical policy. Note that this requires access to the initial state and communication at the first timestep. A mutual information loss is added to encourage diversity of trajectories across the shared latent space, which allows for relatively-greedy agents (once conditioned on the latent variable) to still achieve committed exploration during training similar to Bootstrapped DQN [120]. This results in significant performance gains on some scenarios in SMAC [207].

Action Semantics Networks (ASN) [208] extends QMIX with a novel agent network architecture that separates each agent's Q-value estimation of actions that influence other agents from those actions that do not. The Q-values for actions which influence another agent are computed using the relevant parts of the current agent's observation (e.g. the relative position of another agent). Thus, ASN requires intimate a priori knowledge about an environment's action semantics and the structure of the agent's observations, whereas QMIX does not. Zhang et al. [209] propose CollaQ which decomposes the agent's Q-values (which are fed into a QMIX mixing network) into 2 parts. One part conditions only on the agent's observational information pertaining to itself, the other part utilises at attention-based mechanism to condition on the parts of the observation related to the other agents.

Qatten [210] change the architecture of QMIX's mixing network to a 2-layer linear mixing, in which the weights of the first layer are produced through an attention-based mechanism. Wang et al. [211] augment the agent's with roles in order to encourage both helpful behaviour sharing and specialisation of agents. This is achieved by having each agent's final layer generated by a hypernetwork that conditions on their role. Additionally, to encourage the roles to be diverse/distinct a mutual information loss is utilised. Wang et al. [212] extend this idea by additionally learning an action representation and role-based policies that can be selected by

agents in a hierarchical manner. This allows for behaviours and roles to be learnt in a much smaller space (role-based policies as the actions as opposed to all joint-actions), which leads to better performance.

Wang et al. [213] extend QMIX to a setting which allows for communication between agents. Their approach allows agents to condition their Q-values on messages from other agents, which take the form of a real-valued vector. An entropy loss and a mutual information loss incentivise the learning of succinct and expressive messages. Zhang et al. [214] also allow for each agent's Q-values to condition on vector-valued messages from other agents. However, they only utilise the messages from other agents if their gap between the best and second-best action is small enough (interpreted as uncertainty in the best choice of action). Additionally, they minimise the variance of the messages across actions to limit noisy or uninformative messages.

SMIX(λ) [215] replaces the 1-step Q-learning target of QMIX with a SARSA(λ) target. They incorrectly claim that the Q-learning update rule is responsible for the representational limitations of QMIX, instead of the non-negative weights in the mixing network which enforces monotonicity. They also incorrectly claim that their method can represent a larger class of joint action-value functions than QMIX. Since they also use non-negative weights in their mixing network, SMIX(λ) can represent **exactly** the same class of value functions as QMIX.

Pan et al. [216] demonstrate that QMIX can have severe a severe over-estimation bias (as can other similar algorithms such as VDN, QPLEX, and Weighted QMIX) which can result in poor performance, or a catastrophic drop in performance as training progresses. They demonstrate that popular techniques for dealing with overestimation in the single-agent setting, such as Double Q-learning, do not alleviate the issue since it is exacerbated in the multi-agent setting. They propose to regularise the joint-action Q-values towards a monte-carlo estimate of the episodic return, which they demonstrate is successful in dealing with severe overestimation.

Liu et al. [217] investigate transfer learning algorithms in a multi-agent setting using QMIX as a base. Yang et al. [218] use DIAYN [219] to learn lower-level skills

that a higher level QMIX agent uses. Fu et al. [220] extend QMIX to allow for action spaces consisting of both discrete and continuous actions.

The applicability of QMIX in real-world application domains has also been considered such as robot swarm coordination [221] and stratospheric geoengineering [222].

QMIX has been extended from discrete to continuous action spaces [223, COMIX]. COMIX uses the cross-entropy method to approximate the otherwise intractable greedy per-agent action maximisation step. Witt et al. [223] also present experiments with FacMADDPG, i.e., MADDPG with a factored critic, which show that QMIX-like network factorisations also perform well in actor-critic settings. COMIX has been found to outperform previous state-of-the-art MADDPG in continuous robotic control suite Multi-Agent Mujoco, thereby illustrating its versatility beyond SMAC.

Sun et al. [224] extends QMIX to allow for distributional RL, by ensuring that the consistency property is maintained through an appropriate decomposition.

4.8.5 PyMARL and SMAC

A number of papers have established unit micromanagement in StarCraft as a benchmark for deep multi-agent RL. Usunier et al. [225] present an algorithm using a centralised *greedy MDP* and first-order optimisation which they evaluate on Starcraft: BroodWar [226]. Peng et al. [166] also evaluate their methods on StarCraft. However, neither requires decentralised execution. Similar to our setup in SMAC is the work of Foerster et al. [169], who evaluate replay stabilisation methods for IQL on combat scenarios with up to five agents. Foerster et al. [137] also uses this setting.

In this chapter, we construct unit micromanagement tasks using the *StarCraft II Learning Environment* (SC2LE) [151] as opposed to StarCraft, since it is actively supported by the game developers and offers a more stable testing environment. The full game of StarCraft II has already been used as an RL environment [151], and DeepMind's AlphaStar [26] has recently shown an impressive level of play on a StarCraft II matchup using a centralised controller. By contrast, SMAC introduces strict decentralisation and local partial observability to turn the StarCraft II game engine into a new set of decentralised cooperative multi-agent problems.

Since their introduction, SMAC and PyMARL have been used by a number of papers to evaluate and compare multi-agent RL methods [167, 227, 191, 213, 214, 158, 215, 181, 208].

4.8.6 Hypernetworks and Monotonic Networks

QMIX relies on a neural network to transform the centralised state into the weights of another neural network, in a manner reminiscent of *hypernetworks* [141]. This second neural network is constrained to be monotonic with respect to its inputs by keeping its weights positive. Dugas et al. [156] also investigate such functional restrictions for neural networks.

4.9 SMAC Results

In this section, we present the results of our experiments of QMIX and other existing multi-agent RL methods using the new SMAC benchmark.

The evaluation procedure is similar to the one in [153]. The training is paused after every 10000 timesteps during which 32 test episodes are run with agents performing action selection greedily in a decentralised fashion. The percentage of episodes where the agents defeat all enemy units within the permitted time limit is referred to as the *test win rate*.

The architectures and training details are presented in Appendix B.3.3. A table of the results is included in Appendix B.3.4. Data from the individual runs are available at the SMAC repository (`https://github.com/oxwhirl/smac`).

Figure 4.7 plots the median test win percentage averaged across all scenarios in order to compare the algorithms across the entire SMAC suite. We also plot the performance of a simple heuristic AI that selects the closest enemy unit (ignoring partial observability) and attacks it with the entire team until it is dead, upon which the next closest enemy unit is selected. This is a basic form of *focus-firing*, which is a crucial tactic for achieving good performance in micromanagement scenarios. The relatively poor performance of the heuristic AI shows that the suite of SMAC

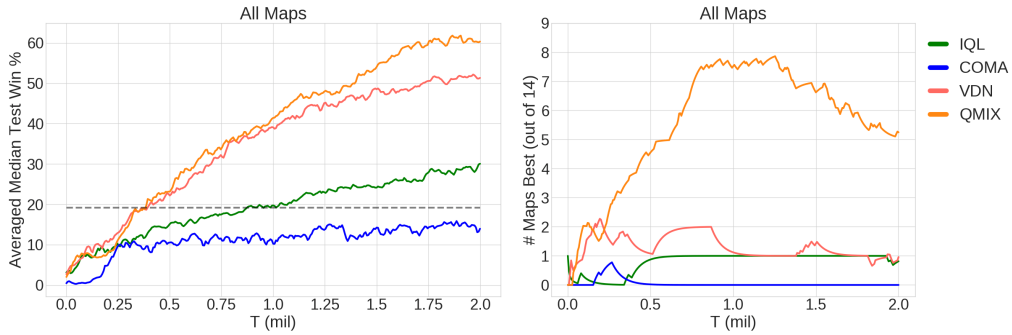


Figure 4.7: Left: The median test win %, averaged across all 14 scenarios. Heuristic's performance is shown as a dotted line. Right: The number of scenarios in which the algorithm's median test win % is the highest by at least 1/32 (smoothed).

scenarios requires more complex behaviour than naively focus-firing the closest enemy, making it an interesting and challenging benchmark.

Overall QMIX achieves the highest test win percentage and is the best performer on up to eight scenarios during training.

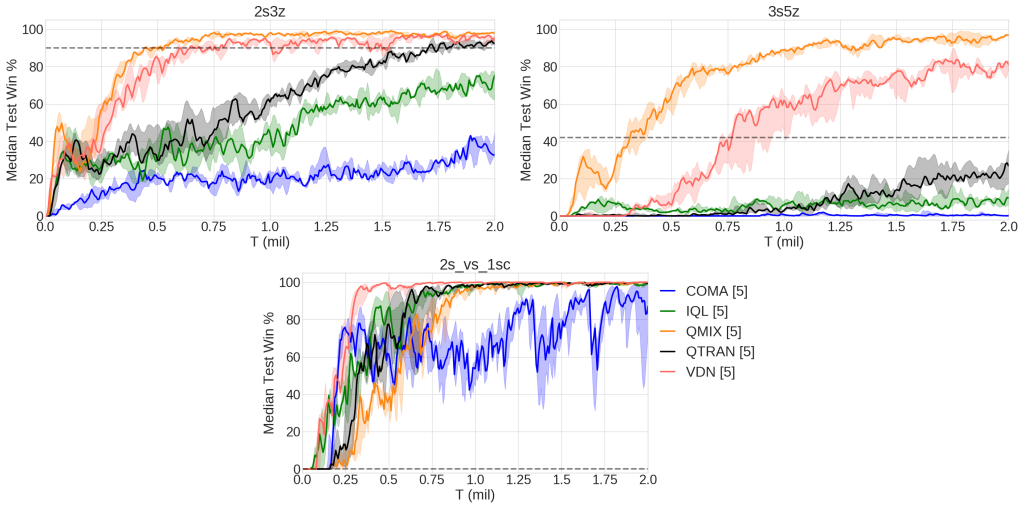


Figure 4.8: Three scenarios including QTRAN.

We also compare to QTRAN on 3 scenarios in Figure 4.8. We can see that QTRAN fails to achieve good performance on *3s5z* and takes far longer to reach the performance of VDN and QMIX on *2s3z*. Since it barely beats IQL on relatively easy scenarios, we do not perform a more comprehensive benchmarking of QTRAN. In preliminary experiments, we found the QTRAN-Base algorithm slightly more performant and more stable than QTRAN-Alt. For more details on the hyperparameters and architectures considered, please see the Appendix B.3.3.

Based on the overall performances of all algorithms, we broadly group the scenarios into three categories: *Easy*, *Hard*, and *Super-Hard*.

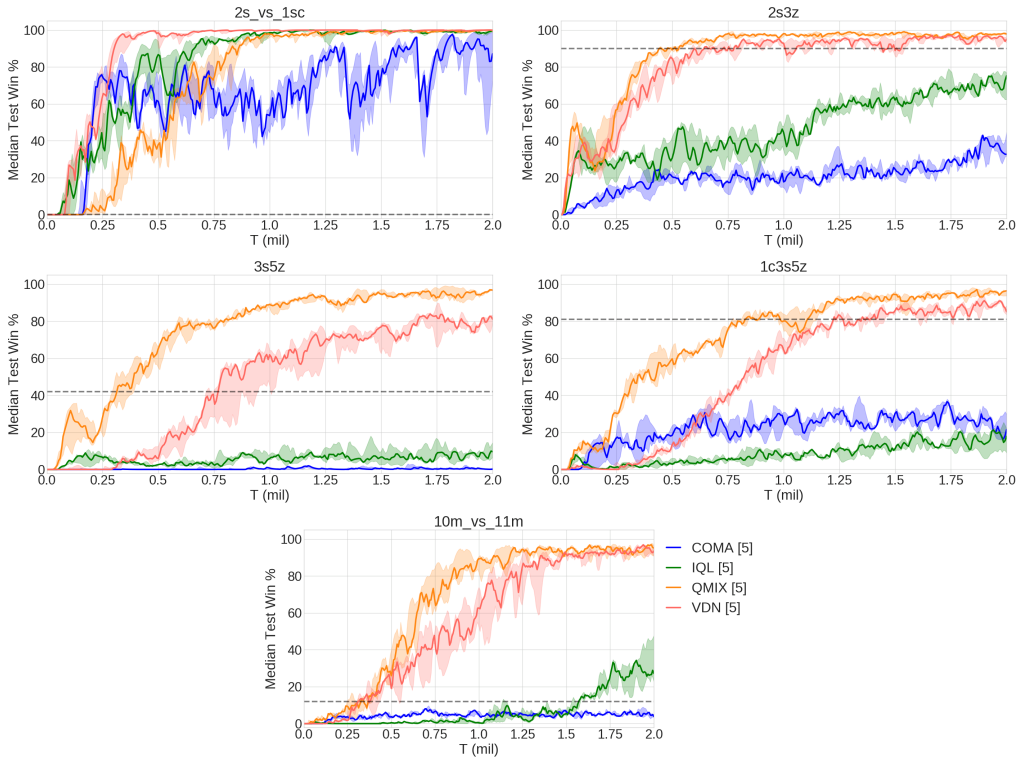


Figure 4.9: Easy scenarios. The heuristic AI's performance shown as a dotted black line.

Figure 4.9 shows that IQL and COMA struggle even on the *Easy* scenarios, performing poorly on four of the five scenarios in this category. This shows the advantage of learning a centralised but factored centralised Q_{tot}. Even though QMIX exceeds 95% test win rate on all of five *Easy* scenarios, they serve an important role in the benchmark as sanity checks when implementing and testing new algorithms.

The *Hard* scenarios in Figure 4.10 each present their own unique problems. *2c_vs_64zg* only contains 2 allied agents, but 64 enemy units (the largest in the SMAC benchmark) making the action space of the agents much larger than the other scenarios. *bane_vs_bane* contains a large number of allied and enemy units, but the results show that IQL easily finds a winning strategy whereas all other methods struggle and exhibit large variance. *5m_vs_6m* is an asymmetric scenario that requires precise control to win consistently, and in which the best performers

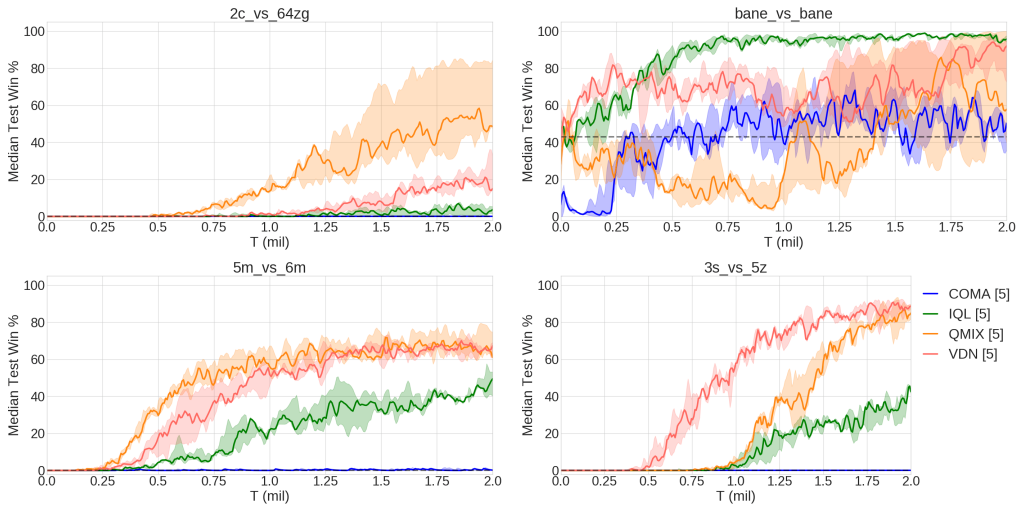


Figure 4.10: Hard scenarios. The heuristic AI's performance shown as a dotted black line.

(QMIX and VDN) have plateaued in performance. Finally, *3s_vs_5z* requires the three allied stalkers to *kite*[5] the enemy zealots for the majority of the episode (at least 100 timesteps), which leads to a delayed reward problem.

The scenarios shown in Figure 4.11 are categorised as *Super Hard* because of the poor performance of all algorithms, with only QMIX making meaningful progress on two of the five. We hypothesise that exploration is a bottleneck in many of these scenarios, providing a nice testbed for future research in this domain.

4.10 Analysis

In this section, we aim to understand the reasons why QMIX outperforms VDN in our experiments. In particular, we consider two possible causes: 1) the inclusion of the central state in the mixing network and 2) the nonlinearity of the mixing network.

We find that inclusion of the central state is extremely important for performance but that it alone is not sufficient to explain QMIX's increased performance. We further find that the mixing network learns nonlinear functions when required, but achieves good performance on the SMAC benchmark even when the learned

[5]A common tactic in StarCraft. The stalker, which can attack from range, will fire on the enemy zealot and then move away to avoid being hit by the zealot's close-range melee attack. This is repeated to drain the enemies health without the stalker incurring any damage.

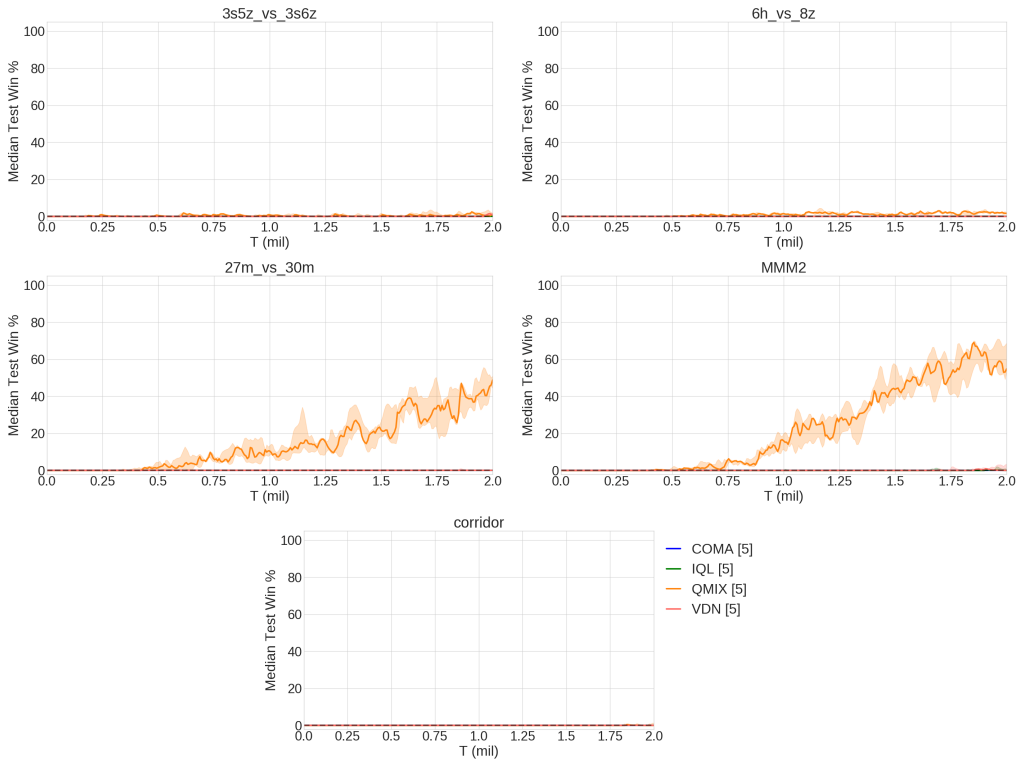


Figure 4.11: Super Hard scenarios. The heuristic AI's performance shown as a dotted black line.

solutions are mostly linear combinations of the agent's utilities. Studying this phenomenon further, we conclude that it is the flexibility of the mixing network's parameterisation, but not its nonlinearity, that accounts for QMIX's superior performance in these tasks.

Our ablation experiments focus on three scenarios, one for each of the three difficulty categories: `3s5z`, `2c_vs_64zg`, and `MMM2`. We chose these three scenarios because they exhibit a large gap in performance between QMIX and VDN, in order to better understand the contribution of the components in QMIX towards that increased performance.

4.10.1 Role of the Central State

In order to disentangle the effects of the various components of the QMIX architecture, we consider the following two ablations:

- **VDN-S:** The output of VDN is augmented by a state-dependent bias.

- **QMIX-NS:** The weights of the mixing network are not a function of the state. The state-dependent bias remains.

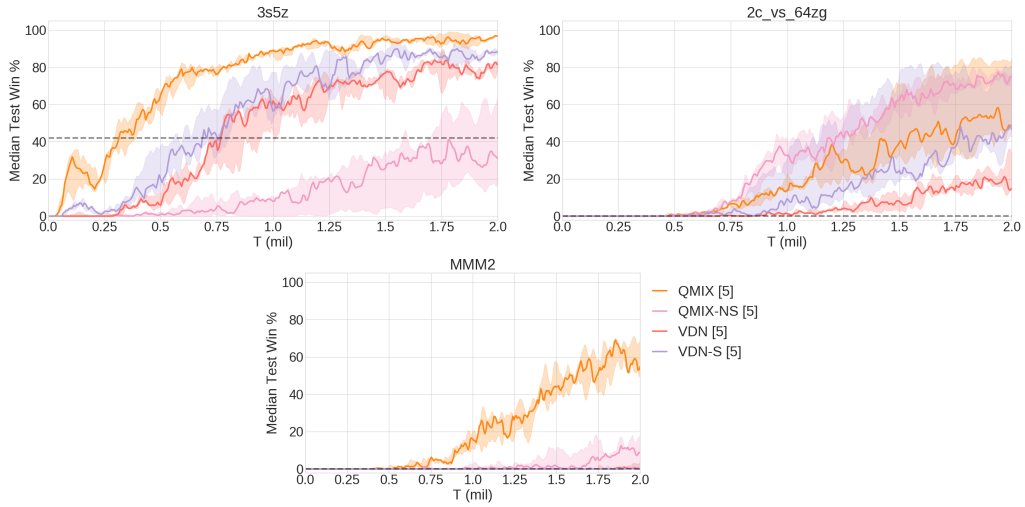


Figure 4.12: Ablations for state experiments.

Figure 4.12 shows that VDN-S performs better than VDN, but still worse than QMIX, indicating that inclusion of the state accounts for some but not all of QMIX's performance. On the easy *3s5z* scenario, QMIX-NS performs very poorly, which is surprising since it is a strict generalisation of VDN-S. However, on the other two scenarios it outperforms VDN-S, indicating that the extra flexibility afforded by a learnt mixing over a fixed summation can be beneficial.

4.10.2 Role of Nonlinear Mixing

The results in Figure 4.12 show that both a state dependent bias on VDN (VDN-S) and a learnt state-independent mixing (QMIX-NS) do not match the performance of QMIX. Thus, we consider another ablation of QMIX:

- **QMIX-Lin:** We remove the final layer and nonlinearity of the mixing network, making it a linear network. The state-dependant bias remains.

Figure 4.13 shows that whilst QMIX-Lin matches QMIX on the easy *3s5z* scenario, it underperforms QMIX on the other two scenarios we considered. This suggests that the ability to perform a nonlinear mixing, through the inclusion

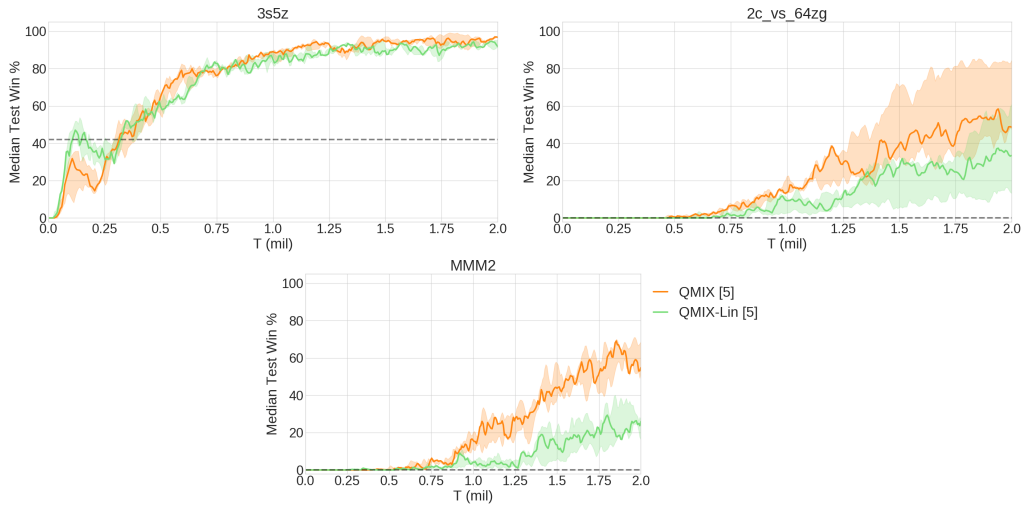


Figure 4.13: Ablations for mixing network linearity experiments.

of a nonlinearity and additional hidden layer in the mixing network, plays an important role QMIX's performance. To confirm this, we examine the function the mixing network learns.

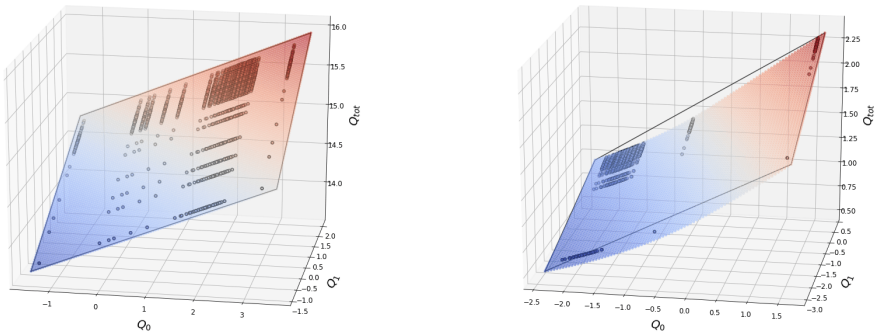


Figure 4.14: The learnt mixing of QMIX on *2c_vs_64zg* at the end of training for timesteps 0 (left) and 50 (right). Circles indicate the Q_{tot}-values for the discrete joint-action space.

Figure 4.14 shows the learnt mixing of the network at the end of training for the *2c_vs_64zg* scenario with two agents trained using QMIX. We generated a sample trajectory by greedily selecting actions and plotted the mixing function at different timesteps. The learnt function is approximately linear for the vast majority of the timesteps in the agent's Q-values, similarly to the first timestep of the episode shown in Figure 4.14 (left). The only timesteps in which we observe a noticeable nonlinearity are at the end of the episode (53 timesteps total). When

trained for longer we still observe that the the mixing function remains linear for the vast majority of timesteps.

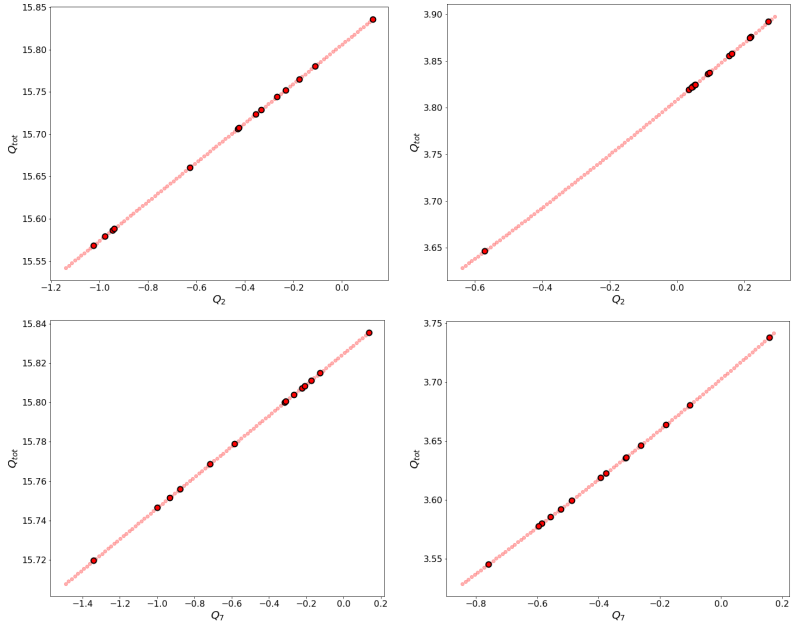


Figure 4.15: The learnt mixing of QMIX on *3s5z* at the end of training for timesteps 0 (left) and 50 (right), for agents 2 (top) and 7 (bottom).

Figure 4.15 shows the learnt mixing for *3s5z* as a function of a single Q_a, keeping the Q-values for the other agent's fixed. The mixing network clearly learns an approximately linear mixing, which is consistent across all timesteps and agents (only timesteps 0 and 50 for agents 2 and 7 are shown since they are representative of all other timesteps and agents).

This is a surprising result. QMIX is certainly capable of learning nonlinear mixing functions in practice, and thereby outperforming the more restrictive VDN decomposition. As an example, Figure 4.16 visualises the learnt mixing function for the didactic two-step game from section 4.4, clearly showing how a nonlinear fit allows QMIX to represent the true optimal value function.

Yet, we observe mostly linear mixing functions in solutions to the SMAC benchmark tasks. If QMIX is learning a mostly linear mixing, then why does it outperform QMIX-Lin? One difference is that QMIX's mixing network is compromised of two fully-connected layers, rather than the single linear layer of QMIX-Lin. If the nonlinearity is not essential, it may be that the extra layer

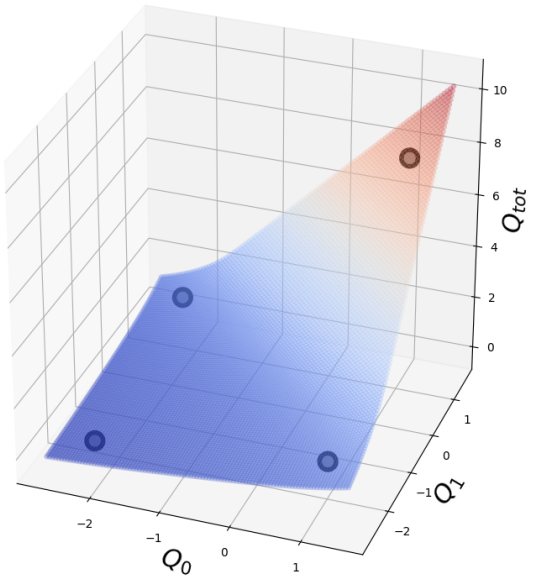


Figure 4.16: The learnt mixing of QMIX on the two-step game from Section 4.4.

in the mixing network is responsible for the performance increase. To test this hypothesis, we consider another ablation:

- **QMIX-2Lin:** QMIX, without a nonlinear activation function in the mixing network.

QMIX-2Lin's mixing network is compromised of two linear layers without any non-linearity between them. Thus, it can only learn a linear transformation of Q_a into Q_{tot}. QMIX-Lin's mixing network is a single linear layer, thus both QMIX-Lin and QMIX-2Lin's mixing networks share the same representational capacity. QMIX's mixing network, on the other hand, has an ELU non-linearity between its two weight layers and thus can represent non-linear transformations of Q_a to Q_{tot}.

We first consider a simple regression experiment in which the states, agent Q-values, and targets are randomly generated and fixed. Further details on the regression task are included in Appendix B.4.

We compare the loss of QMIX (with an ELU non-linearity), QMIX-Lin, and QMIX-2Lin in Figure 4.17, which illustrates that QMIX and QMIX-2Lin perform similarly, and both perform a faster optimisation of the loss than QMIX-Lin. Even

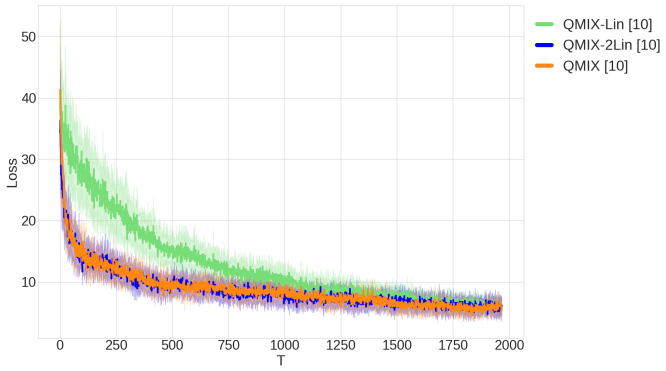


Figure 4.17: The loss on a regression task.

though QMIX-Lin and QMIX-2Lin can both learn only linear mixing functions, their parametrisation of such a linear function are different. Having an extra layer is a better parametrisation in this scenario because it allows for gradient descent to optimise the loss faster [228].

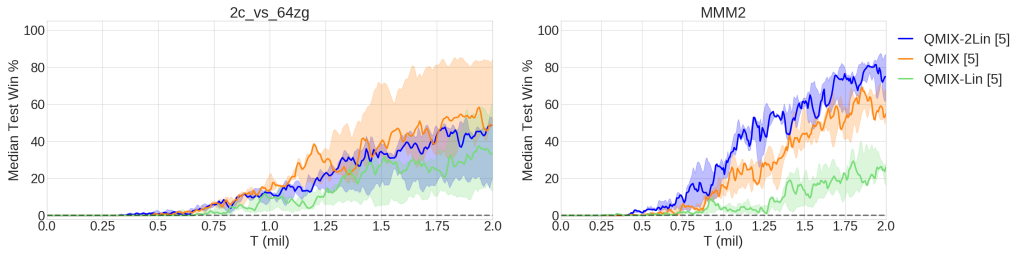


Figure 4.18: Ablations for linear experiments.

Figure 4.18 confirms that the more flexible parameterisation of the mixing network is largely responsible for the increased performance. However, the results on *2c_vs_64zg* show that QMIX performs slightly better than QMIX-2Lin, indicating that in some scenarios nonlinear mixing can still be beneficial. Figure 4.14 (right) shows an example of when such a nonlinearity is learnt on this task.

We also investigate changing the nonlinearity in the mixing network from ELU, which is largely linear, to Tanh.

Figures 4.19 and 4.20 show the learnt mixings when using a tanh non-linearity. The mixing network using tanh learns a more nonlinear mixing, especially for *3s5z*, than with ELU. Figure 4.21 shows how this translates to performance: there are perhaps some small gains in learning speed for some random seeds when using

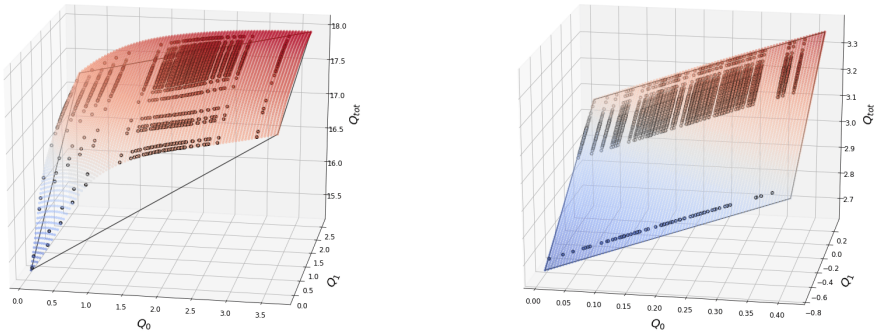


Figure 4.19: The learnt mixing of QMIX on *2c_vs_64zg* using a tanh nonlinearity at the end of training for timesteps 0 (left) and 50 (right).

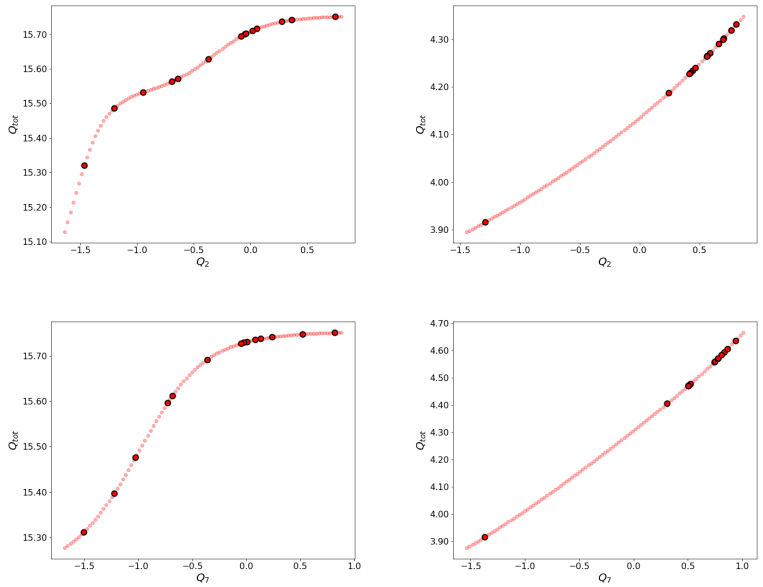


Figure 4.20: The learnt mixing of QMIX on *3s5z*, using a tanh nonlinearity, at the end of training for timesteps 0 (left) and 50 (right), for agents 2 (top) and 7 (bottom).

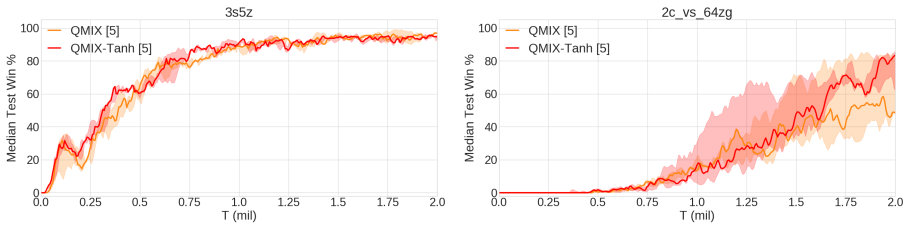


Figure 4.21: The Median test win % comparing QMIX with an ELU nonlinearity (QMIX) and QMIX with a tanh nonlinearity in the mixing network (QMIX-Tanh).

tanh. However, the effect of an additional layer (in improving the optimisation dynamics) is much more significant than encouraging a more non-linear mixing

through the choice of nonlinearity.

4.10.3 Role of RNN in Agent Network

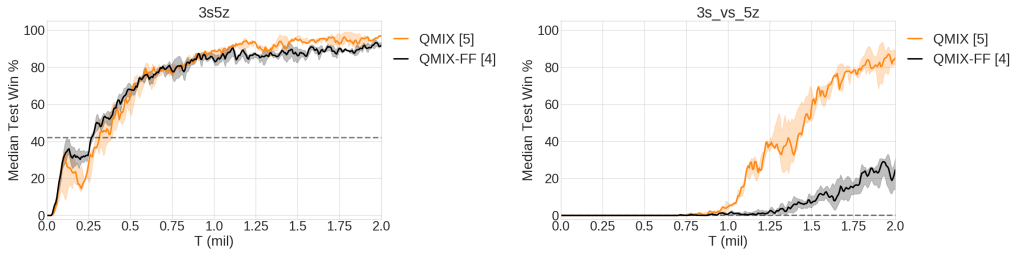


Figure 4.22: Comparing agent networks with and without RNNs (QMIX-FF) on two scenarios.

Finally, we compare the necessity of using an agent's action-observation history by comparing the performance of an agent network with and without an RNN in Figure 4.22. On the easy scenario of *3s5z* we can see that an RNN is not required to use action-observation information from previous timesteps, but on the harder scenario of *3s_vs_5z* it is crucial to learning how to kite effectively.

4.10.4 Role of the COMA Critic Architecture

In addition to the algorithmic differences between QMIX and COMA (QMIX is an off-policy Q-learning algorithm; COMA is an on-policy actor-critic algorithm), the architecture used to estimate the relevant Q-values differs. COMA's critic is a feed-forward network, whereas QMIX's architecture for estimating Q_{tot} contains RNNs (the agent networks). The results of Figure 4.22 raise the question: To what extent do the architectural differences between QMIX and COMA affect final performance? To test this, we change the architecture of the COMA critic to match that of QMIX more closely. Specifically it now consists of per-agent components with the same architecture and inputs as the agent networks used by QMIX, which feed Q_a values for their chosen action into a mixing network. Since there is no requirement for monotonicity in this mixing network, we simplify its architecture significantly to a feed-forward network with ReLU non-linearities and two hidden layers of 128 units. We test two variants of the mixing network, one with access

to the state and one without. The former takes the state as input in addition to the agent's chosen action Q_a-values. The output of the mixing network is $|U|$ units, and we mask out the Q_a-value for the agent whose Q-values are being produced.

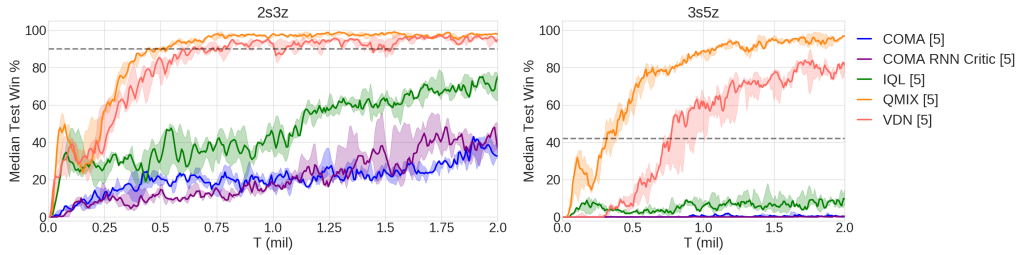


Figure 4.23: Comparing the different architectures for the COMA critic without the state, on two scenarios.

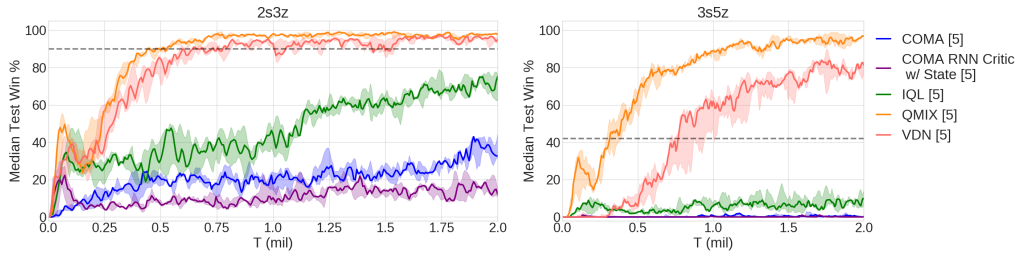


Figure 4.24: Comparing the different architectures for the COMA critic with the state on two scenarios.

Figures 4.23 and 4.24 show that the architectural change has little effect on the performance of COMA, with the addition of the state actually degrading performance in *2s3z*.

4.10.5 Role of Network Size and Depth

Another architectural consideration is the overall size of the network used to estimate Q_{tot} for VDN and VDN-S. Since QMIX has an additional mixing network, the overall architecture is deeper during training. In order to understand the effect of this depth on performance, we consider the following variants of VDN and VDN-S.

- **VDN Wider:** We increase the size of the agent's hidden layer from 64 to 128.
- **VDN Deeper:** We add an additional layer to the agent network after the GRU, with a ReLU non-linearity.

- **VDN-S Bigger:** We increase the size of the state-dependent bias to 2 hidden layers of 64 units.

In our other experiments, we have used the same agent networks for each method, to ensure a fair comparison of the performance of the decentralised policies. By increasing the capacity of the agent networks for VDN Wider and VDN Deeper, we are now using larger agent networks than all other methods. In this sense, the comparisons to these variants are no longer fair. Figure 4.25 shows the results for VDN Wider and VDN Deeper. As expected, increasing the capacity of the agent networks increases performance. In particular, increasing the depth results in a large performance increase on the challenging MMM2. However, despite the larger agent networks, VDN Wider and VDN Deeper still do not match the performance of QMIX with smaller, less expressive agent networks. Figure 4.26 shows the results for VDN-S Bigger, in which we increase the size of the state-dependent bias. The results show that this has a minimal impact on performance over VDN-S.

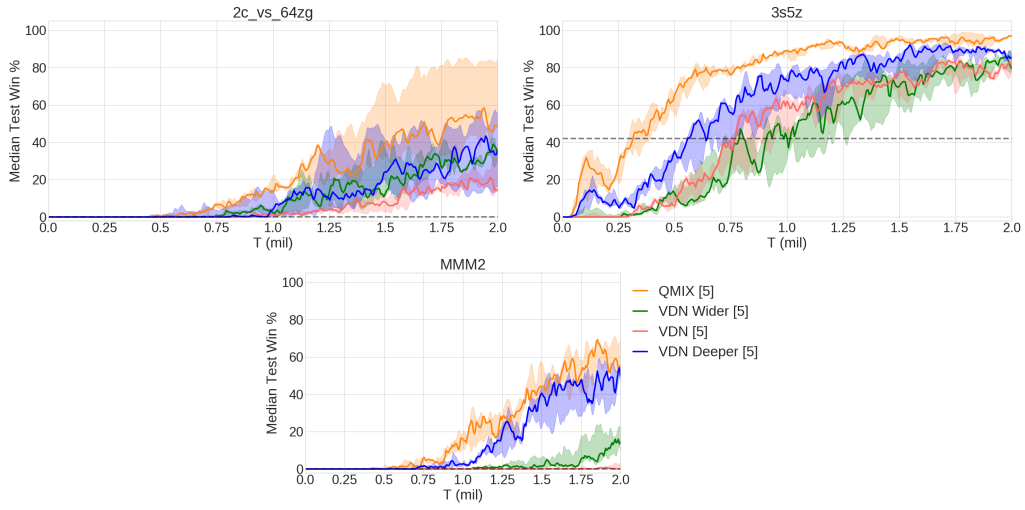


Figure 4.25: Comparing larger agent networks for VDN.

4.11 Conclusion

This chapter presented QMIX, a deep multi-agent RL method that allows end-to-end learning of decentralised policies in a centralised setting and makes efficient use of

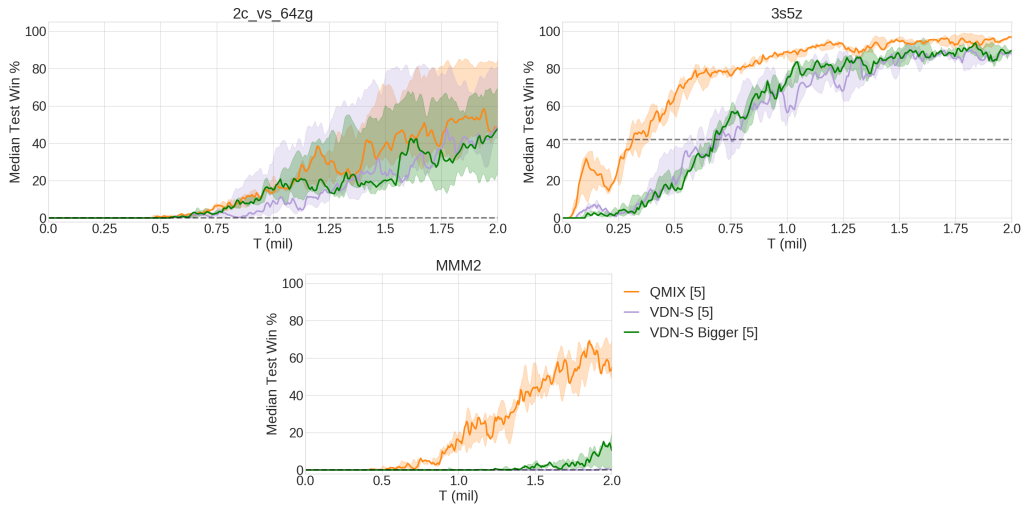


Figure 4.26: Comparing a larger state-dependent bias for VDN-S.

extra state information. QMIX allows the learning of a rich joint action-value function, which admits tractable decompositions into per-agent action-value functions. This is achieved by imposing a monotonicity constraint on the mixing network.

To evaluate QMIX and other multi-agent RL algorithms, we introduced SMAC as a set of benchmark problems for cooperative multi-agent RL. SMAC focuses on decentralised micromanagement tasks and features 14 diverse combat scenarios which challenge MARL methods to handle partial observability and high-dimensional inputs. Additionally, we open-source PyMARL, a framework for cooperative Deep MARL algorithms that features implementations of IQL, VDN, QMIX, QTRAN and COMA.

Our results on SMAC show that QMIX improves the final performance over other relevant deep multi-agent RL algorithms, most notably VDN. We include a detailed analysis through a series of ablation experiments, sensitivity analyses and visualisations for why QMIX outperforms VDN, showing that two components of QMIX are crucial to the performance the algorithm: a state-dependent bias and a learnt mixing of the agent's utilities into Q_{tot} that is conditioned on the state. Through our analysis, we found that the learnt mixing need only be linear to recover the majority of QMIX's performance, but that the parametrisation for this linear function is an important consideration.

5

Weighted Value Function Factorisation

Contents

5.1 Introduction 128
5.2 Multi-Agent MDP Background 129
5.3 QMIX Operator 130
5.3.1 Properties of $\mathcal{T}^*_{\text{Qmix}}$ 132
5.4 Weighted QMIX Operator 133
5.4.1 Weightings 134
5.4.2 Weighted QMIX Operators 136
5.5 Proof of Theoretical Results 137
5.6 Deep Reinforcement Learning Algorithm 147
5.7 Related Algorithms 149
5.7.1 Relationship to Actor-Critic methods 149
5.7.2 Relationship To QTRAN 150
5.7.3 QPLEX 152
5.8 Related Work 153
5.9 Experimental Results 154
5.9.1 Ablations 154
5.9.2 Predator Prey 154
5.9.3 SMAC 156
5.10 Utility of a Weighting 159
5.10.1 Motivations For a Weighting 160
5.10.2 Implicit vs Explicit Weighting 161
5.10.3 Parametrisation 165
5.10.4 Contextual Matrix Game Results 167
5.10.5 Applications of a Weighting Beyond QMIX 173
5.11 Conclusions and Future Work 178

5.1 Introduction

In the previous chapter we introduced QMIX, a Q-learning algorithm with state-of-the-art performance on the StarCraft Multi-Agent Challenge benchmark. QMIX represents the optimal joint action value function using a monotonic *mixing* function of per-agent utilities. This restricted function class $\mathcal{Q}^{mix}$ allows for efficient maximisation during training, and easy decentralisation of the learned policy. However, QMIX is unable to represent joint action value functions that are characterised as *nonmonotonic* [158], i.e., an agent's ordering over its own actions depends on other agents' actions. Consequently, QMIX cannot solve tasks that require significant coordination within a given timestep [229, 181]. In this chapter, we analyse an idealised, tabular version of QMIX to study this representational limitation, and then develop algorithms to resolve these limitations in theory and in practice.

We formalise the objective that QMIX optimises, which allows us to view QMIX as an operator that first computes the Q-learning targets and then projects them into $\mathcal{Q}^{mix}$ by minimising the unweighted squared error across all joint actions. We show that, since in general $Q^* \notin \mathcal{Q}^{mix}$, the projection of Q^*, which we refer to as Q_{tot}, can have incorrect estimates for the optimal joint action, yielding suboptimal policies. These are fundamental limitations of the QMIX algorithm independent from exploration and compute constraints, and occur even with access to the true Q^*.

These limitations primarily arise because QMIX's projection of Q^* yields a Q_{tot} that places equal importance on approximating the Q-values for *all* joint actions. Our key insight is that if we ultimately care only about the greedy optimal policy, it is more important to accurately represent the value of the optimal joint action than the suboptimal ones. Therefore, we can improve the policy recovered from Q_{tot} by appropriately weighting each joint action when projecting Q^* into $\mathcal{Q}^{mix}$.

Based on this intuition, we introduce a weighting function into our projection. In the idealised tabular setting we propose two weighting functions and prove that

the projected Q_{tot} recovers the correct maximal action for any Q, and therefore for Q^* as well. Since this projection always recovers the correct maximal joint action, we benefit from access to Q^* (or a learned approximation of it). To this end, we introduce a learned approximation of Q^*, from an unrestricted function class, which we call $\hat{Q}^*$. By using Q_{tot}, now a weighted projection of $\hat{Q}^*$, to perform maximisation, we show that $\hat{Q}^*$ converges to Q^* and that Q_{tot} thus recovers the optimal policy.

Based on our analysis and results in the tabular setting, we present two scalable versions of our algorithm, Centrally-Weighted (CW) QMIX and Optimistically-Weighted (OW) QMIX. We demonstrate their improved ability to cope with environments with nonmonotonic value functions, by showing superior performance in a predator-prey task in which the trade-offs made by QMIX prevent it from solving the task. Additionally, we demonstrate improved robustness over QMIX to the amount of exploration performed, by showing better empirical performance on a range of SMAC maps. Our ablations and additional analysis experiments demonstrate the importance of both a weighting and an unrestricted $\hat{Q}^*$ in our algorithm. Finally, we investigate the utility of the weighting beyond its original motivations of changing the Q-values learnt by QMIX. We demonstrate that it can improve the performance of other algorithms such as QPLEX, by improving the learning dynamics in the Deep RL setting.

5.2 Multi-Agent MDP Background

As in the previous chapter, the focus of our work is on solving Dec-POMDPs under the CTDE setting.

However, for the purposes of our theoretical analysis, we first consider a *fully observable* setting in which each agent's observations are the full state. This is equivalent to a multi-agent MDP (MMDP) [154] which is itself equivalent to a standard MDP with U^n as the action space. This removes the additional complexities associated with partial observability, and allows us to focus our analysis on the specific algorithmic limitations of QMIX.

The Bellman optimality operator for our fully observable setting is defined by:

$$\mathcal{T}^*Q(s,\mathbf{u}) := \mathbb{E}[r + \gamma \max_{\mathbf{u}'} Q(s',\mathbf{u}')], \tag{5.1}$$

where the expectation is over the next state $s' \sim P(\cdot|s,\mathbf{u})$ and reward $r \sim R(\cdot|s,\mathbf{u})$.

In this setting, QMIX combines the per-agent utilities, Q_a via a continuous monotonic function that is state-dependent: $f_s(Q_1(s,u_1),...,Q_n(s,u_n)) = Q_{tot}(s,\mathbf{u})$, where $\frac{\partial f_s}{\partial Q_a} \geq 0,\ \forall a \in A \equiv \{1,...,n\}$. We do not consider the use of function approximators here, and assume that each $Q_a(\cdot,\cdot)$ is stored in a table as in tabular RL. We also do not restrict the form of f_s beyond being monotonic, and do not concern ourselves with how such a function might be parametrised or represented (since it does not affect our analysis).

5.3 QMIX Operator

In this section we examine an operator that represents an *idealised* version of QMIX in a tabular setting. The purpose of our analysis is primarily to understand the fundamental limitations of QMIX that stem from its training objective and the restricted function class it uses. We write this function class as $\mathcal{Q}^{mix}$:

$$\mathcal{Q}^{mix} := \{Q_{tot} | Q_{tot}(s,\mathbf{u}) = f_s(Q_1(s,u_1),...Q_n(s,u_n)), \frac{\partial f_s}{\partial Q_a} \geq 0, Q_a(s,u) \in \mathbb{R}\}.$$

This is the space of all Q_{tot} that can be represented by monotonic functions of tabular $Q_a(s,u)$. At each iteration of our idealised algorithm, we constrain Q_{tot} to lie in $\mathcal{Q}^{mix}$ by solving the following optimisation problem:

$$\arg\min_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} (\mathcal{T}^*Q_{tot}(s,\mathbf{u}) - q(s,\mathbf{u}))^2. \tag{5.2}$$

To avoid any confounding factors regarding exploration, we assume this optimisation is performed for all states and joint actions at each iteration, as in planning algorithms like value iteration [34]. We also assume the optimisation is performed exactly. However, since it is not guaranteed to have a unique solution, a random q is returned from the set of objective-minimising candidates.

By contrast, the original QMIX algorithm in the Deep RL setting interleaves exploration and approximate optimisation of this objective, using samples from the environment to approximate $\mathcal{T}^*$, and a finite number of gradient descent steps to estimate the argmin. Additionally, sampling uniformly from a replay buffer of the most recent experiences does not strictly lead to a uniform weighting across joint actions. Instead the weighting is proportional to the frequency at which a joint action was taken. Incorporating this into our analysis would introduce significant complexity and detract from our main focus: to analyse the limitations of restricting the representable function class to $\mathcal{Q}^{mix}$.

The optimisation in (5.2) can be separated into two distinct parts: the first computes targets using $\mathcal{T}^*$, and the second projects those targets into $\mathcal{Q}^{mix}$. We define the corresponding projection operator Π_{Qmix} as follows:

$$\Pi_{\text{Qmix}} Q := \operatorname*{arg\,min}_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} (Q(s, \mathbf{u}) - q(s, \mathbf{u}))^2$$

We can then define $\mathcal{T}^*_{\text{Qmix}} := \Pi_{\text{Qmix}} \mathcal{T}^*$ as the QMIX operator, which exactly corresponds to the objective in (5.2).

There is a significant literature studying projections of value functions into linear function spaces for RL (see [27] for a detailed introduction and overview). However, despite some superficial similarities, our focus is considerably different in several ways. First, we consider a nonlinear projection, rendering many methods of analysis inapplicable. Second, as the $Q_a(s, u)$ are tabular, there is no tradeoff in the quality of representation across different states, and consequently no need to weight the different states in the projection. By contrast, unlike linear Q-functions, our restricted space does induce tradeoffs in the quality of representation across different joint actions, and weighting them differently in the projection is central to our method. Third, the targets in our optimisation are fixed to $\mathcal{T}^* Q_{tot}(s, \mathbf{u})$ at each iteration, rather than depending on q as they would in a minimisation of Mean Squared Bellman Error (MSBE) or Mean Squared Projected Bellman Error (MSPBE) in linear RL. This makes our setting closer to that of fitted Q-iteration [230] in which a regression problem is solved at each iteration to fit a function

1	0
0	1

1	1/3
1/3	1/3

1/3	1/3
1/3	1

Table 5.1: Non-monotonic payoff matrix (Left), where both agents can take 2 actions. If both agent's actions match then they receive 1 reward, 0 otherwise. The two possible Q_{tot}'s returned by Π_{Qmix} (Middle and Right) are symmetric, and cannot perfectly represent the payoffs. returned by Π_{Qmix} (Middle and Right)

approximator from a restricted class to Q-learning-style targets. Our focus of study is the unique properties of the particular function space $\mathcal{Q}^{mix}$, and the tradeoffs in representation quality induced by projection into it.

5.3.1 Properties of $\mathcal{T}^*_{\text{Qmix}}$

To highlight the pitfalls of the projection Π_{Qmix} into $\mathcal{Q}^{mix}$, we consider the effect of applying Π_{Qmix} to the true Q^*, which is readily available in deterministic normal form games where Q^* is just the immediate reward.

$\mathcal{T}^*_{\text{Qmix}}$ is not a contraction. The payoff matrix in Table 5.1 (Left) is a simple example of a value function that cannot be perfectly represented in $\mathcal{Q}^{mix}$. Table 5.1 (Middle) and (Right) show two distinct Q_{tot}, both of which are global minima of the optimisation solved by $\Pi_{\text{Qmix}}Q^*$. Hence, $\mathcal{T}^*_{\text{Qmix}}$ is not a contraction, which would have a unique fixed point.

QMIX's argmax is not always correct. There exist Q-functions such that $\arg\max \Pi_{\text{Qmix}}Q \neq \arg\max\ Q$. For example, the payoff matrix in Table 5.2 (Left) (from Son et al. [229]) produces a value function for which QMIX's approximation (Right) does not result in the correct argmax.

8	-12	-12
-12	0	0
-12	0	0

-12	-12	-12
-12	0	0
-12	0	0

Table 5.2: Payoff matrix (Left) in which both agents can take 3 actions. If both agent's take action 0 then they receive the maximum reward of 8, but if only one agent takes action 0 then a penalty of -12 is given. All other actions receive a 0 reward. The Q_{tot} returned from Π_{Qmix} has an incorrect argmax (Right), in particular it severely underestimates the value for the maximum reward of 8 as -12.

QMIX can underestimate the value of the optimal joint action. Furthermore, if it has an incorrect argmax, the value of the true optimal joint action can be underestimated, e.g., -12 instead of 8 in Table 5.2. If QMIX gets the correct argmax then it represents the maximum Q-value perfectly (proved formally in Section 5.5). However, if QMIX's argmax joint action is not the true optimal joint action then QMIX can underestimate the value of that action.

These failure modes are problematic because they show fundamental limitations of QMIX, that are independent from: 1) compute constraints, since we exactly minimise the objectives posed; 2) exploration, since we update every state-action pair; and 3) parametrisation of the mixing function and agent utilities, since we are assume that Q_{tot} can be any member of $\mathcal{Q}^{mix}$, whereas in practice we can only represent a subset of $\mathcal{Q}^{mix}$.

5.4 Weighted QMIX Operator

In this section we introduce an operator for an *idealised* version of our algorithm, *Weighted QMIX* (WQMIX), in order to compare it to the operator we introduced for QMIX.

The negative results in Section 5.3.1 concern the scenario in which we optimise QMIX's loss function across *all* joint actions for every state. We argue that this equal weighting over joint actions when performing the optimisation in (5.2) is responsible for the possibly incorrect argmax of the objective-minimising solution. Consider the example in Table 5.2. A monotonic $Q_{tot} \in \mathcal{Q}^{mix}$ cannot increase its estimate of the value of the single optimal joint action above -12 without either increasing the estimates of the value of the bad joint actions above their true value of -12, or decreasing the estimates of the zero-value joint actions below -12. The error for misestimating several of the suboptimal joint actions would outweigh the improvement from better estimating the single optimal joint action. As a result the optimal action value is underestimated and the resulting policy is suboptimal.

By contrast, consider the extreme case in which we only optimise the loss for the *single* optimal joint action $\mathbf{u}^*$. For a single action, the representational limitation

of QMIX has no effect so we can optimise the objective perfectly, recovering the value of the optimal joint action.

However, we still need to learn that the other action values are lower than $Q_{tot}(\mathbf{u}^*)$ in order to recover the optimal policy. To prioritise estimating $Q_{tot}(\mathbf{u}^*)$ well, while still anchoring down the value estimates for other joint actions, we can add a suitable weighting function w into the projection operator of QMIX:

$$\Pi_w Q := \arg\min_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} w(s, \mathbf{u})(Q(s, \mathbf{u}) - q(s, \mathbf{u}))^2. \tag{5.3}$$

The weighting function $w : S \times \mathbf{U} \to (0, 1]$ weights the importance of each joint action in QMIX's loss function. It can depend on more than just the state and joint action, but we omit this from the notation for simplicity. Setting $w(s, \mathbf{u}) = 1$ recovers the projection operator Π_{Qmix}.

5.4.1 Weightings

The choice of weighting is crucial to ensure that WQMIX can overcome the limitations of QMIX. As shown in Section 5.3, even if we have access to Q^*, if we use a uniform weighting then we can still end up with the wrong argmax after projection into the monotonic function space. We now consider two different weightings and show in Theorems 4 and 5 that these choices of w ensure that the Q_{tot} returned from the projection has the correct argmax. The proofs of these theorems can be found in Section 5.5. For both weightings, let $\alpha \in (0, 1]$ and consider the weighted projection of an arbitrary joint action value function Q.

Idealised Central Weighting The first weighting, which we call Idealised Central Weighting, is quite simple:

$$w(s, \mathbf{u}) = \begin{cases} 1 & \mathbf{u} = \mathbf{u}^* = \arg\max_{\mathbf{u}} Q(s, \mathbf{u}) \\ \alpha & \text{otherwise.} \end{cases} \tag{5.4}$$

To ensure that the weighted projection returns a Q_{tot} with the correct argmax, we simply down-weight every suboptimal action. However, this weighting requires computing the maximum across the joint action space, which is often infeasible. In Section 5.6 we discuss an approximation to this weighting in the Deep RL setting.

Theorem 4. *Let w be the Idealised Central Weighting from* (5.4)*. Then* $\exists \alpha > 0$ *such that* $\arg\max \Pi_w Q = \arg\max Q$ *for any* Q*.*

Theorem 4 provides a sanity check that this choice of weighting guarantees we recover a Q_{tot} with the correct argmax in this idealised setting, with a nonzero weighting for suboptimal actions. The high-level intuition for the proof is as follows: We first consider the loss for a Q_{tot}that has the correct argmax. Next, we show that the loss for a different Q_{tot}' that does not have the correct argmax is strictly larger for an appropriate choice of α.

Optimistic Weighting The second weighting, which we call Optimistic Weighting, affords a practical implementation:

$$w(s, \mathbf{u}) = \begin{cases} 1 & Q_{tot}(s, \mathbf{u}) < Q(s, \mathbf{u}) \\ \alpha & \text{otherwise.} \end{cases} \tag{5.5}$$

This weighting assigns a higher weighting to those joint actions that are underestimated relative to Q, and hence could be the true optimal actions (in an optimistic outlook).

Theorem 5. *Let w be the Optimistic Weighting from* (5.5)*. Then* $\exists \alpha > 0$ *such that,* $\arg\max \Pi_w Q = \arg\max Q$ *for any* Q*.*

Theorem 5 shows that Optimistic Weighting also recovers a Q_{tot} with the correct argmax. The high-level intuition for the proof is similar to that of the idealised central weighting. We first construct a $Q_{tot}$with a specific form: It's estimate of the true maximum action's Q-value is correct, and all other actions' Q-values are slightly lower. Then we show that for any other Q_{tot}' that does not attain the correct argmax, the loss will always be larger. Thus the minimiser of the loss must attain the correct argmax.

5.4.2 Weighted QMIX Operators

We have shown that these two weightings are guaranteed to recover the correct maximum joint action for any Q, and therefore for Q^* as well. This is in contrast to the uniform weighting ($w = 1$) of QMIX, which can fail to recover the correct optimal joint action even for simple matrix games. The weighted projection now allows us to fully take advantage of Q^*.

Since we do not have access to the true optimal value function in general, we learn an approximation to it: $\hat{Q}^*$, which does not need to lie in the restricted monotonic function space $\mathcal{Q}^{mix}$. Performing an exact maximisation of $\hat{Q}^*$ requires a search over the entire joint action space, which is typically intractable and does not admit decentralisation. We instead use our QMIX approximation Q_{tot} to *suggest* the maximum joint action(s), which can then be evaluated by $\hat{Q}^*$.

Learning $\hat{Q}^*$ instead of using Q_{tot} in its place brings some advantages. First, it allows us a richer representational class to approximate Q^* with, since we place no restrictions on the form of $\hat{Q}^*$. In the idealised tabular setting, Q^* is exactly representable by $\hat{Q}^*$. Second, since we are weighting each joint action in Π_w, Q_{tot} (unlike $\hat{Q}^*$) likely has less accurate estimates for those joint actions with a low weighting. Due to these factors, we may bootstrap using more accurate estimates by using $\hat{Q}^*$ instead of Q_{tot}. These properties are necessary to ensure that WQMIX converges to the optimal policy. The operator used to update $\hat{Q}^*$ is:

$$\mathcal{T}_w^* \hat{Q}^*(s, \mathbf{u}) := \mathbb{E}[r + \gamma \hat{Q}^*(s', \arg\max_{\mathbf{u}'} Q_{tot}(s', \mathbf{u}'))]. \tag{5.6}$$

Since Q_{tot} is monotonic, the argmax in (5.6) is tractable. Similarly Q_{tot} is updated in tandem using:

$$\mathcal{T}^*_{\text{WQMIX}} Q_{tot} := \Pi_w \mathcal{T}_w^* \hat{Q}^* \tag{5.7}$$

$\mathcal{T}_w^*$ is similar to the Bellman Optimality Operator in (5.1) but does not directly maximise over $\hat{Q}^*$. Instead it uses $Q_{tot} \in \mathcal{Q}^{mix}$ to suggest a maximum joint action. Setting w to be uniform ($w = 1$) here does not recover QMIX since we are additionally learning $\hat{Q}^*$.

Finally, using our previous results, we show that $\hat{Q}^*$ converges to Q^* and that Q_{tot} recovers an optimal policy. This provides a firm theoretical foundation for Weighted QMIX: in an idealised setting it converges to the optimal policy, whereas QMIX does not.

Corollary 2. *Letting w be the Idealised Central or Optimistic Weighting, then $\exists \alpha > 0$ such that the unique fixed point of $\mathcal{T}_w^*$ is Q^*. Furthermore, $\Pi_w Q^* \subseteq \mathcal{Q}^{mix}$ recovers an optimal policy, and $\max \Pi_w Q^*(s, \cdot) = \max Q^*(s, \cdot)$.*

In this section we have shown that an idealised version of Weighted QMIX can converge to Q^* and recover an optimal policy. Restricting Q_{tot} to lie in $\mathcal{Q}^{mix}$ does not prevent us from representing an optimal policy, since there is *always* an optimal deterministic policy [34] and all deterministic policies can be derived from the argmax of a Q that lies in $\mathcal{Q}^{mix}$. Thus, we do *not* expand the function class that we consider for Q_{tot}. Instead, we change the solution of the projection by introducing a weighting.

5.5 Proof of Theoretical Results

In this section we now prove the theoretical results outlined previously. Where appropriate we will outline where our weighting w depends on more than just the state-action pair.

Proposition 1. *For any $w : S \times \mathbf{U} \to (0, 1]$ and Q. Let $Q_{tot} \in \Pi_w Q$. Then $\forall s \in S$, $\forall \hat{\mathbf{u}} \in \arg\max Q_{tot}(s, \cdot)$. We have that $Q_{tot}(s, \hat{\mathbf{u}}) \geq Q(s, \hat{\mathbf{u}})$.*

Proof. Consider a $s \in S$.
Assume for a contradiction that $\exists \hat{\mathbf{u}} \in \arg\max Q_{tot}(s, \cdot)$ s.t $Q_{tot}(s, \hat{\mathbf{u}}) < Q(s, \hat{\mathbf{u}})$.

Define a new Q'_{tot} as follows:

$$Q'_{tot}(s, \mathbf{u}) = \begin{cases} Q(s, \mathbf{u}) & \mathbf{u} = \hat{\mathbf{u}} \\ Q_{tot}(s, \mathbf{u}) & \text{otherwise,} \end{cases}$$

By construction we have that $Q'_{tot} \in \mathcal{Q}^{mix}$. This is because we are free to increase the Q-value for a single action $\hat{\mathbf{u}}$ if it is an argmax. Then we can consider the loss for this new Q'_{tot}:

$$
\begin{aligned}
&\sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 \\
&= \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 + w(s,\hat{\mathbf{u}})(Q(s,\hat{\mathbf{u}}) - Q'_{tot}(s,\hat{\mathbf{u}}))^2 \\
&= \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 \\
&\quad \text{since } (Q'_{tot}(s,\hat{\mathbf{u}}) = Q(s,\hat{\mathbf{u}})) \\
&= \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 \\
&\quad \text{since } (Q'_{tot}(s,\mathbf{u}) = Q_{tot}(s,\mathbf{u})\ \forall \mathbf{u} \neq \hat{\mathbf{u}}) \\
&< \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 + w(s,\hat{\mathbf{u}})(Q(s,\hat{\mathbf{u}}) - Q_{tot}(s,\hat{\mathbf{u}}))^2 \\
&\quad \text{adding the term for } \hat{\mathbf{u}} \text{ and since } (Q_{tot}(s,\hat{\mathbf{u}}) < Q(s,\hat{\mathbf{u}})) \\
&= \sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2.
\end{aligned}
$$

Thus our original Q_{tot} cannot be the solution of $\Pi_w Q$ since its loss is strictly larger than for Q'_{tot}, a contradiction. And so $\forall \hat{\mathbf{u}} \in \arg\max Q_{tot}(s,\cdot)$ we have that $Q_{tot}(s,\hat{\mathbf{u}}) \geq Q(s,\hat{\mathbf{u}})$. □

Proposition 2. *For any $w : S \times \mathbf{U} \to (0,1]$ such that if w depends on the value of $Q_{tot}(s,\mathbf{u})$ ($q \in \mathcal{Q}^{mix}$) for that specific state-action pair within the projection then $w(s,\mathbf{u},Q_{tot}(s,\mathbf{u}))$ is a monotonic function of $Q_{tot}(s,\mathbf{u})$. Let Q be any target. Let $Q_{tot} \in \Pi_w Q$. Then $\forall s \in S$, $\forall \hat{\mathbf{u}} \in \arg\max Q_{tot}(s,\cdot)$ if $\hat{\mathbf{u}} \in \arg\max_{\mathbf{u}} Q(s,\mathbf{u})$ then $Q_{tot}(s,\hat{\mathbf{u}}) = Q(s,\hat{\mathbf{u}})$.*

Proof. First we would like to note the extra condition we have placed on w that was not present in the previous Proposition. Both weightings we consider satisfy this condition for a fixed target Q. The (Corrected) Idealised Central Weighting does not depend on the value of $Q_{tot}(s,\mathbf{u})$ at all, and the Optimistic Weighting can decrease its value from 1 to $\alpha \leq 1$ if that value increases.

Consider a $s \in S$ as before.

Assume for a contradiction that $\exists \hat{\mathbf{u}} \in \arg\max Q_{tot}(s,\cdot)$ such that if $\hat{\mathbf{u}} \in \arg\max_{\mathbf{u}} Q(s,\mathbf{u})$ then $Q_{tot}(s,\hat{\mathbf{u}}) > Q(s,\hat{\mathbf{u}})$.

Let $\mathbf{u}^* \in \arg\max Q(s, \cdot)$ (and so $Q(s, \mathbf{u}^*) = \max_{\mathbf{u}} Q(s, \mathbf{u})$). We can also assume that $\hat{\mathbf{u}} \in \arg\max Q(s, \cdot)$, otherwise the statements we wish to prove or contradict are vacuous.

Then define a new Q'_{tot} as follows:

$$Q'_{tot}(s, \mathbf{u}) = \begin{cases} Q(s, \mathbf{u}) (= Q(s, \mathbf{u}^*)) & \mathbf{u} = \hat{\mathbf{u}} \\ \min\{Q_{tot}(s, \mathbf{u}), Q(s, \mathbf{u}^*)\} & \text{otherwise,} \end{cases}$$

By construction $Q'_{tot} \in \mathcal{Q}^{mix}$.

This is because our new $Q'_{tot}(s, \mathbf{u}) = \min\{Q_{tot}(s, \mathbf{u}), Q(s, \mathbf{u}^*)\}$ $\forall \mathbf{u}$. Essentially, in the region of the joint-action space in which this change takes affect, the gradient of our function changes from being non-negative to simply 0. An even looser justification is that we by clipping our original Q_{tot} to have a maximum of no more than $Q(s, \mathbf{u}^*)$, we are just flattening the function there which does not make it non-monotonic.

We then consider the loss, using w' to denote the weighting for Q'_{tot} and w for Q_{tot} respectively:

$$
\begin{aligned}
&\sum_{\mathbf{u}\in\mathbf{U}} w'(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 \\
&= \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w'(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 + w'(s,\hat{\mathbf{u}})(Q(s,\hat{\mathbf{u}}) - Q'_{tot}(s,\hat{\mathbf{u}}))^2 \\
&= \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w'(s,\mathbf{u})(Q(s,\mathbf{u}) - Q'_{tot}(s,\mathbf{u}))^2 \\
&\quad \text{since } (Q'_{tot}(s,\hat{\mathbf{u}}) = Q(s,\hat{\mathbf{u}})) \\
&\leq \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 \\
&\quad \text{If } \min\{Q_{tot}(s,\mathbf{u}), Q(s,\mathbf{u}^*)\} = Q_{tot}(s,\mathbf{u}) \text{ then } Q'_{tot}(s,\mathbf{u}) = Q_{tot}(s,\mathbf{u}). \\
&\quad \text{and so } w(s,\mathbf{u}) = w'(s,\mathbf{u}). \\
&\quad \text{Otherwise } Q'_{tot}(s,\mathbf{u}) = Q(s,\mathbf{u}^*) < Q_{tot}(s,\mathbf{u}) \implies \\
&\quad (Q'_{tot}(s,\mathbf{u}^*) - Q(s,\mathbf{u}))^2 = (Q(s,\mathbf{u}^*) - Q(s,\mathbf{u}))^2 < (Q_{tot}(s,\mathbf{u}) - Q(s,\mathbf{u}))^2, \\
&\quad \text{since } Q(s,\mathbf{u}) \leq Q(s,\mathbf{u}^*) < Q_{tot}(s,\mathbf{u}). \\
&\quad \text{And we have that } w'(s,\mathbf{u}) \leq w(s,\mathbf{u}) \text{ since } Q'_{tot}(s,\mathbf{u}) < Q_{tot}(s,\mathbf{u}). \\
&< \sum_{\mathbf{u}\neq\hat{\mathbf{u}}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 + w(s,\hat{\mathbf{u}})(Q(s,\hat{\mathbf{u}}) - Q_{tot}(s,\hat{\mathbf{u}}))^2 \\
&\quad \text{adding the term for } \hat{\mathbf{u}} \text{ back to the sum} \\
&\quad \text{and since } Q_{tot}(s,\hat{\mathbf{u}}) > Q(s,\hat{\mathbf{u}}) \text{ by assumption} \\
&= \sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2.
\end{aligned}
$$

Thus, Q_{tot} cannot be the solution of $\Pi_w Q$, a contradiction. This then proves that $\forall\hat{\mathbf{u}} \in \arg\max Q_{tot}(s,\cdot)$ if $\hat{\mathbf{u}} \in \arg\max_{\mathbf{u}} Q(s,\mathbf{u})$ then $Q_{tot}(s,\hat{\mathbf{u}}) \leq Q(s,\hat{\mathbf{u}})$. Combining with Proposition 1 gives us the desired equality. □

Proposition 3. *Let $Q_{tot} \in \Pi_w Q$, where we consider only the (Corrected) Idealised Central Weighting or the Optimistic Weighting. $\forall s \in S\ \exists\hat{\mathbf{u}} \in \arg\max Q_{tot}(s,\cdot)$ such that $Q_{tot}(s,\hat{\mathbf{u}}) = Q(s,\hat{\mathbf{u}})$.*

Proof. Assume for a contradiction that $\forall\hat{\mathbf{u}} \in \arg\max Q_{tot}$ we have that $Q_{tot}(\hat{\mathbf{u}}) > Q(\hat{\mathbf{u}})$.

Define $\Gamma_s := Q_{tot}(s,\hat{\mathbf{u}}) - \max\{Q_{tot}(s,\mathbf{u}) | \mathbf{u} \in \mathbf{U}, Q_{tot}(s,\mathbf{u}) < Q_{tot}(s,\hat{\mathbf{u}})\}$ to be the difference between the maximum Q-Value and the next biggest Q-Value (the

action gap [231]). Γ_s is well defined as long as there exists a sub-optimal action. If there is not a suboptimal action, then trivially any $\mathbf{u} \in \mathbf{U}$ satisfies the condition.

Let $\epsilon = \min\{\Delta_s/2, (Q_{tot}(s, \hat{\mathbf{u}}) - \max\{Q(s, \mathbf{u}) | \mathbf{u} \in \arg\max Q_{tot}(s, \cdot)\})/2\} > 0$.

Define Q'_{tot} as follows:

$$Q'_{tot}(s, \mathbf{u}) = \begin{cases} Q_{tot}(s, \mathbf{u}) - \epsilon & \forall \mathbf{u} \in \arg\max Q_{tot} \\ Q_{tot}(s, \mathbf{u}) & \text{otherwise.} \end{cases}$$

i.e. we have decreased the Q-Value estimates for the argmax joint actions by a small non-zero amount. Due to the way we have defined ϵ we are safe to do so. This is because $\epsilon < \Gamma_s$, and so we do not need to worry about adjusting other action's estimates to ensure that $Q'_{tot} \in \mathcal{Q}^{mix}$. Additionally, $\epsilon < (Q_{tot}(s, \hat{\mathbf{u}}) - \max\{Q(s, \mathbf{u}) | \mathbf{u} \in \arg\max Q_{tot}(s, \cdot)\})$. i.e. That we have not decreased Q'_{tot} enough to change the ordering between it and the target's Q-value. This ensures that our weighting, specifically for the Optimistic weighting, does not change.

By construction $Q'_{tot} \in \mathcal{Q}^{mix}$.

Then Q'_{tot} has a smaller loss than Q_{tot} since the estimates for the argmax actions are closer to the true values and the values of the weightings haven't changed at all.

This gives our contradiction since $Q_{tot} \in \Pi_w Q$.

Thus $\exists \hat{\mathbf{u}} \in \arg\max Q_{tot}$ such that $Q_{tot}(\hat{\mathbf{u}}) \leq Q(\hat{\mathbf{u}})$. Combined with Proposition 1 gives us the required result. □

Corollary 3. *If Q_{tot} has a unique argmax $\hat{\mathbf{u}}$, then $Q_{tot}(\hat{\mathbf{u}}) = Q(\hat{\mathbf{u}})$.*

Proof. Proposition 3 showed the existence of an argmax action whose Q_{tot}-value matches Q exactly. If there is a unique argmax $\hat{\mathbf{u}}$, then it must match exactly giving our result. □

We will now prove the two main theorems. Note that the statement of the theorems differs from their statement in the previous section. Since $\Pi_w Q$ and Q don't necessarily have unique argmaxes, then $\arg\max \Pi_w Q$ and $\arg\max Q$ should be sets. Hence we change the statement of the theorems, and the definition of the Idealised Central Weighting appropriately.

Since a large purpose of our theoretical results is to provide intuition and inspiration for our Deep RL algorithms, we choose to leave the technically incorrect statements in the previous section. The extra work and overhead required to ensure proper rigour does not provide any actionable or useful insight into the algorithms. Instead, we believe it only obfuscates the intuition we wish to build.

An alternative which we did not pursue, is to place an extra restriction on all of the discussion and theoretical results in Section 5.4. So that we only consider target Q-values that have a unique argmax. This would then result in our projection having a unique minimiser, and that unique minimiser also having a unique argmax. This would significantly simplify some of the presentation, but would rule out simple cases such as the 2 agent 2 action matrix game in Figure 5.1. Although we did not pursue this restriction for our proofs, we are already implicitly assuming it for much for our discussion in Section 5.4.

Theorem 4. *Let w be the Idealised Central Weighting. Then $\exists \alpha > 0$ such that,* $\arg\max \Pi_w Q \subseteq \arg\max Q$ *for any Q.*

Theorem 5. *Let w be the Optimistic Weighting from Equation* (5.5)*. Then $\exists \alpha > 0$ such that,* $\arg\max \Pi_w Q \subseteq \arg\max Q$ *for any Q.*

Proof. Since the proof of both theorems contains a significant overlap, we will merge them both into a single proof. To begin we should clarify the statement of these theorems.

First, we will now redefine the Idealised Central Weighting. Let Q be our target Q-values. Then picking a single $\mathbf{u}^* \in \arg\max Q$ for every state, we can define the (Corrected) Idealised Central Weighting as follows:

$$w(s, \mathbf{u}) = \begin{cases} 1 & \mathbf{u} = \mathbf{u}^* \\ \alpha & \text{otherwise.} \end{cases} \tag{5.8}$$

Next we must clarify what we mean by $\arg\max \Pi_w Q \subseteq \arg\max Q$.

By this we mean:

$\forall Q_{tot} \in \Pi_w Q, \arg\max Q_{tot} \subseteq \arg\max Q$.

i.e. For a valid Q_{tot} that globally minimises the loss (a Q_{tot} that is returned from

the projection), an argmax action for it will always be a valid argmax action in the original target Q. The crucial property that follows from this, that we will rely on later for proving Corollary 4 is :
$\forall s \in S, \forall Q_{tot} \in \Pi_w, \forall \hat{\mathbf{u}} \in \arg\max Q_{tot}(s, \cdot), Q(s, \hat{\mathbf{u}}) = \max Q(s, \mathbf{u})$.

(Corrected) Idealised Central Weighting

We will start by first considering the (Corrected) Idealised Central Weighting: Let $Q_{tot} = \Pi_w Q$ ($Q_{tot} \in \Pi_w Q$ if there are distinct solutions).

Consider a state $s \in S$.

Define $\Delta_s := Q(s, \mathbf{u}^*) - \max\{Q(s, \mathbf{u}) | \mathbf{u} \in \mathbf{U}, Q(s, \mathbf{u}) < Q(s, \mathbf{u}^*)\}$ to be the difference between the maximum Q-Value and the next biggest Q-Value (the action gap [231]). Δ_s is well defined as long as there exists a sub-optimal action. If there is not a suboptimal action, then trivially $\arg\max \Pi_w Q \subseteq \arg\max Q$ for state s.

Let $\hat{\mathbf{u}} \in \arg\max Q_{tot}$, we additionally define $\Gamma_s := Q_{tot}(s, \hat{\mathbf{u}}) - \max\{Q_{tot}(s, \mathbf{u}) | \mathbf{u} \in \mathbf{U}, Q_{tot}(s, \mathbf{u}) < Q_{tot}(s, \hat{\mathbf{u}})\}$ as the action gap for Q_{tot}. We argue now that this is also always well-defined.

Γ_s is well-defined:
Consider a scenario in which it is not well-defined, then every action is an argmax of Q_{tot}. Hence $Q_{tot} = c$ for some constant c. Since we are assuming that Δ_s is well-defined (otherwise we have nothing to prove), then there necessarily exists a sub-optimal action for Q, which we denote as d. Then we can lower our loss by considering a new Q_{tot} which has a lower estimate for d (using similar arguments and constructions as in the previous propositions). Hence, if $Q_{tot} \in \Pi_w Q$ then it must have a sub-optimal action also.

We now begin our proof:
By Proposition 2, $\forall \hat{\mathbf{u}} \in \arg\max Q_{tot}(s, \cdot)$ we have that if $\hat{\mathbf{u}} \in \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$ then $Q_{tot}(s, \hat{\mathbf{u}}) = Q(s, \hat{\mathbf{u}})$.

We will now consider the 2 cases in which $\mathbf{u}^* \in \arg\max Q_{tot}(s, \cdot)$ and not, where $\mathbf{u}^*$ is from the definition of the (Corrected) Idealised Central Weighting in Equation (5.8) and hence $\mathbf{u}^* \in \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$.

Let us first consider the case where $\mathbf{u}^* \in \arg\max Q_{tot}(s, \cdot)$.

Then the loss:

$$\sum_{\mathbf{u} \in \mathbf{U}} w(s, \mathbf{u})(Q(s, \mathbf{u}) - Q_{tot}(s, \mathbf{u}))^2 = \alpha \sum_{\mathbf{u} \neq \mathbf{u}^*} (Q(s, \mathbf{u}) - Q_{tot}(s, \mathbf{u}))^2 < \alpha(\frac{R_{max}}{1-\gamma})^2 |U|^n,$$

where $R_{max} := \max R - \min R$, the difference between the minimum and maximum rewards across all state-actions. The first equality is because we have that $Q_{tot}(s, \mathbf{u}^*) = Q(s, \mathbf{u}^*)$ by Prop 2 and the definition of the weighting from Equation (5.8). The last inequality follows since the maximum difference between Q-values in the discounted setting is then $\frac{R_{max}}{1-\gamma}$, and there are $|U|^n$ joint-actions total.

We now consider the case where $\mathbf{u}^* \notin \arg\max Q_{tot}(s, \cdot)$. The loss is then:

$$\begin{aligned}
&\sum_{\mathbf{u} \in \mathbf{U}} w(s, \mathbf{u})(Q(s, \mathbf{u}) - Q_{tot}(s, \mathbf{u}))^2 \\
&= (Q(s, \mathbf{u}^*) - Q_{tot}(s, \mathbf{u}^*))^2 + \ \alpha \sum_{\mathbf{u} \neq \mathbf{u}^*} (Q(s, \mathbf{u}) - Q_{tot}(s, \mathbf{u}))^2 \\
&\geq \Gamma_s^2,
\end{aligned}$$

since $Q(s, \mathbf{u}^*) - Q_{tot}(s, \mathbf{u}^*) \geq \Gamma_s$, which is proved below.

By Proposition 3 let $\hat{\mathbf{u}}' \in \arg\max Q_{tot}$ such that $Q_{tot}(s, \hat{\mathbf{u}}') = Q(s, \hat{\mathbf{u}}')$.

Then $Q(s, \mathbf{u}^*) \geq Q(s, \hat{\mathbf{u}}') = Q_{tot}(s, \hat{\mathbf{u}}') \geq Q_{tot}(s, \mathbf{u}^*) + \Gamma_s$

$\implies Q(s, \mathbf{u}^*) - Q_{tot}(s, \mathbf{u}^*) \geq \Gamma_s$.

Setting $0 < \alpha_s < \frac{\Gamma_s^2 (1-\gamma)^2}{(R_{max})^2 |U|^n}$ then leads to the loss for the scenario where $\mathbf{u}^* \notin \arg\max Q_{tot}(s, \cdot)$ being strictly higher. Thus, we have that $\mathbf{u}^* \in \arg\max Q_{tot}$. We then let $\alpha = \min_s \alpha_s > 0$ to find the appropriate α that works across all states. For those states in which Γ_s is ill-defined we can simply set α to any appropriate value.

$\nexists \boldsymbol{d} \in \mathbf{arg\,max}\, \boldsymbol{Q_{tot}}$ such that $\boldsymbol{d} \notin \mathbf{arg\,max}\, \boldsymbol{Q}$:

Now consider $d \in \arg\max Q_{tot}$ such that $d \neq \mathbf{u}^*$. If no such d exists then we are done since $\mathbf{u}^*$ is the unique maximiser of Q_{tot} and we know it is a argmax of Q by definition.

We must now prove that all $d \in \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$.

Consider for a contradiction that $\exists d \notin \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$. We then also have that $Q_{tot}(s, d) = Q_{tot}(s, \mathbf{u}^*) = Q(s, \mathbf{u}^*)$.

Then $Q_{tot}(s,d) = Q(s,\mathbf{u}^*) \geq Q(s,d) + \Delta_s$. Thus, we can lower our loss by constructing a new Q_{tot} in which we lower the estimates for all $d \in \arg\max Q_{tot}$ that are not in $\arg\max_{\mathbf{u}} Q(s,\mathbf{u})$. We know we are always able to lower our estimate by a non-negative amount by a similar argument to Proposition 3. Hence if $Q_{tot} \in \Pi_w Q$, then we cannot have that $\exists d \in \arg\max Q_{tot}$ such that $d \notin \arg\max_{\mathbf{u}} Q(s,\mathbf{u})$. This completes the proof for the (Corrected) Idealised Central Weighting.

Optimistic Weighting

For the proof of the Optimistic Weighting we will use many of the same arguments and notation.

We will once again consider a single state $s \in S$ and the action gaps Γ_s and Δ_s. Now as before, pick $\mathbf{u}^* \in \arg\max_{\mathbf{u}} Q(s,\mathbf{u})$.

Then, let us consider a Q_{tot} of a specific form:
For $s \in S$, let $Q_{tot}(s,\mathbf{u}^*) = c_s + \epsilon = Q(s,\mathbf{u}^*)$, where $\epsilon << \min\{\Delta_s, \Gamma_s\}$ and $Q_{tot}(s,\mathbf{u}) = c_s, \forall \mathbf{u} \neq \mathbf{u}^*$. Note that here Q_{tot} has a unique maximum action by definition.

Then the loss:

$$\begin{aligned}\sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 &= \alpha \sum_{\mathbf{u}\neq\mathbf{u}^*} (Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 \\ &\leq \alpha(\frac{R_{max}}{1-\gamma})^2 |U|^n < \Gamma_s^2,\end{aligned}$$

since $Q_{tot}(s,\mathbf{u}) = c_s = Q(s,\mathbf{u}^*) - \epsilon > Q(s,\mathbf{u}), \forall \mathbf{u} \neq \mathbf{u}^*$ which means that $w(s, \mathbf{u} \neq \mathbf{u}^*) = \alpha$. The final inequality follows due to setting $0 < \alpha_s < \frac{\Gamma_s^2(1-\gamma)^2}{(R_{max})^2|U|^n}$ as earlier.

Now consider another Q'_{tot}.

If $\mathbf{u}^* \notin \arg\max Q'_{tot}$, then the loss is:

$$\begin{aligned}&\sum_{\mathbf{u}\in\mathbf{U}} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 \\ &= (Q(s,\mathbf{u}^*) - Q_{tot}(s,\mathbf{u}^*))^2 + \sum_{\mathbf{u}\neq\mathbf{u}^*} w(s,\mathbf{u})(Q(s,\mathbf{u}) - Q_{tot}(s,\mathbf{u}))^2 \\ &\geq \Gamma_s^2.\end{aligned}$$

We now justify this inequality: By Proposition 3 let $\hat{\mathbf{u}}' \in \arg\max Q'_{tot}$ such that $Q'_{tot}(s,\hat{\mathbf{u}}') = Q(s,\hat{\mathbf{u}}')$.

Then $Q(s, \mathbf{u}^*) \geq Q(s, \hat{\mathbf{u}}') = Q_{tot}(s, \hat{\mathbf{u}}') \geq Q_{tot}(s, \mathbf{u}^*) + \Gamma_s$

$\implies Q(s, \mathbf{u}^*) - Q_{tot}(s, \mathbf{u}^*) \geq \Gamma_s > 0$, and hence that $Q_{tot}(s, \mathbf{u}^*) < Q(s, \mathbf{u}^*)$. Thus, $w(s, \mathbf{u}^*) = 1$.

Thus, we have shown that for any Q'_{tot} with $\mathbf{u}^* \notin \arg\max Q'_{tot}$ the loss is greater than the Q_{tot} we first considered with $\mathbf{u}^* \in \arg\max Q'_{tot}$. We set α as before to obtain an appropriate value that works across all states.

Exactly as before we can show that we cannot have that $\exists d \in \arg\max Q_{tot}$ such that $d \notin \arg\max_{\mathbf{u}} Q(s, \mathbf{u})$. The only modification required to that argument, is to note that for the values we are decreasing, we are decreasing them by less than Δ_s from $Q(s, \mathbf{u}^*)$. Hence, our Optimistic Weighting does not change since the next highest Q from $\max Q$ is Δ_s away. This completes our proof for the Optimistic Weighting.

□

Corollary 4. *Letting w be the Central or Optimistic Weighting, then $\exists \alpha > 0$ such that the unique fixed point of $\mathcal{T}_w^*$ is Q^*. Furthermore, $\Pi_w Q^* \subseteq \mathcal{Q}^{mix}$ recovers an optimal policy, and $\max \Pi_w Q^* = \max Q^*$.*

Proof. Using the results of Theorems 4 and 5 we know that $\exists \alpha > 0$ such that $\arg\max \Pi_w Q \subseteq \arg\max Q$. We also know from their proofs that the same α works for both weightings.

Instead of updating Q_{tot} in tandem with $\hat{Q}^*$, we can instead write $\mathcal{T}_w^*$ as:

$$\mathcal{T}_w^* \hat{Q}^*(s, \mathbf{u}) := \mathbb{E}[r + \gamma \hat{Q}^*(s', \arg\max_{\mathbf{u}'} (\Pi_w \hat{Q}^*)(s', \mathbf{u}'))].$$

And so:

$$\hat{Q}^*(s', \arg\max_{\mathbf{u}'} (\Pi_w \hat{Q}^*)(s', \mathbf{u}')) = \max_{\mathbf{u}'} \hat{Q}^*(s', \mathbf{u}'), \ \forall s' \in S.$$

Thus, our operator $\mathcal{T}_w^*$ is equivalent to the usual Bellman Optimality Operator $\mathcal{T}^*$, which is known to have a unique fixed point Q^* [36, 37].

Once again by the results of Theorems 4 and 5, we know that $Q^*_{tot} \in \Pi_w Q^*$ achieves the correct argmax for every state. Thus it is an optimal policy. Finally, Proposition 2 then shows that $\max \Pi_w Q^* = \max Q^*$.

□

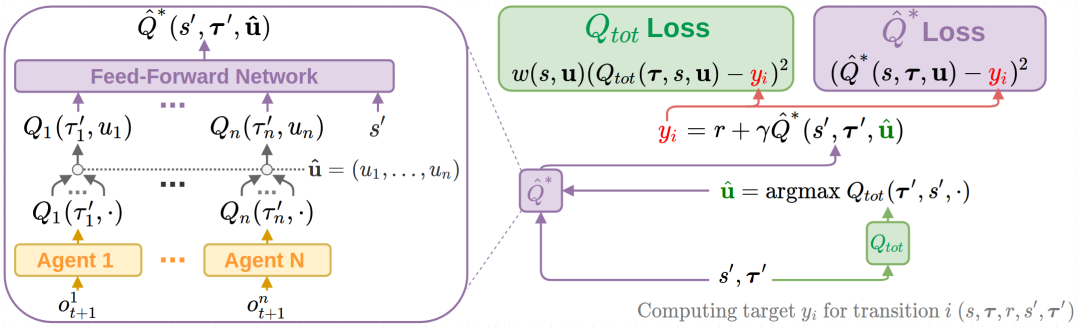


Figure 5.1: Deep RL Weighted QMIX setup. **Left:** The architecture used for $\hat{Q}^*$. **Right:** How the targets y_i for each transition in the minibatch are computed and used.

5.6 Deep Reinforcement Learning Algorithm

So far, we have only considered an idealised setting in order to analyse the fundamental properties of QMIX and Weighted QMIX. However, the ultimate goal of our analysis is to inform the development of new scalable RL algorithms, in combination with, e.g., neural network function approximators. We now describe the realisation of Weighted QMIX for Deep RL, in a Dec-POMDP setting in which each agent does not observe the full state, as described in Section 5.2.

There are three components to Weighted QMIX: 1) Q_{tot}, i.e., the per-agent utilities Q_a (from which the decentralised policies are derived) and the mixing network, 2) an unrestricted joint action $\hat{Q}^*$, and 3) a weighting function w, as in Π_w.

$\mathbf{Q_{tot}}$ The Q_{tot} component is largely the same as that of Rashid et al. [153], using the architecture from Samvelyan et al. [232]. Q_{tot} is trained to minimise the following loss:

$$\sum_{i=1}^{b} w(s, \mathbf{u})(Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) - y_i)^2, \tag{5.9}$$

where $y_i := r + \gamma \hat{Q}^*(s', \boldsymbol{\tau}', \arg\max_{\mathbf{u}'} Q_{tot}(\boldsymbol{\tau}', \mathbf{u}', s'))$ is treated as a fixed target. This differs from the idealised setting considered in Sections 5.3 and 5.4 because we are now only optimising the Q-values for the state-action pairs present in the minibatch sampled from the replay buffer, as opposed to every state-action pair.

Centralised $\hat{Q}^*$ We use a similar architecture to Q_{tot} to represent $\hat{Q}^*$: agent networks whose chosen action's utility is fed into a mixing network. $\hat{Q}^*$ is thus conditioned on the state s and the agents' action-observation histories $\boldsymbol{\tau}$. For the agent networks we use the same architecture as QMIX. They do not share parameters with the agents used in representing Q_{tot}. The mixing network for $\hat{Q}^*$ is a feed-forward network that takes the state and the appropriate actions' utilities as input. This mixing network is not constrained to be monotonic by using non-negative weights. Consequently, we can simplify the architecture by having the state and agent utilities be inputs to $\hat{Q}^*$'s mixing network, as opposed to having hypernetworks take the state as input and generate the weights. The architecture of $\hat{Q}^*$ is shown in Figure 5.1 (Left). $\hat{Q}^*$ is trained to minimise the following loss, using y_i from (5.9):

$$\sum_{i=1}^{b}(\hat{Q}^*(s,\boldsymbol{\tau},\mathbf{u})-y_i)^2. \tag{5.10}$$

Weighting Function Idealised Central Weighting requires knowing the maximal joint action over $\hat{Q}^*$, which is computationally infeasible. In order to derive a practical algorithm, we must make approximations. For each state-action pair in the sampled minibatch, the weighting we use is:

$$w(s,\mathbf{u})=\begin{cases}1 & y_i>\hat{Q}^*(s,\boldsymbol{\tau},\hat{\mathbf{u}}^*) \text{ or } \mathbf{u}=\hat{\mathbf{u}}^*\\ \alpha & \text{otherwise,}\end{cases} \tag{5.11}$$

where $\hat{\mathbf{u}}^*=\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau},\mathbf{u},s)$. Since we do not know the maximal joint action for each state, we make a local approximation: if $y_i>\hat{Q}^*(s,\boldsymbol{\tau},\hat{\mathbf{u}}^*)$, then $\mathbf{u}$ might be the best joint action. We use $\hat{Q}^*(s,\boldsymbol{\tau},\hat{\mathbf{u}}^*)$ instead of $\mathcal{T}_w^*\hat{Q}^*(s,\boldsymbol{\tau},\hat{\mathbf{u}}^*)$ since we do not have direct access to it. We refer to this weighting function as **Centrally-Weighted QMIX (CW)**.

The Optimistic Weighting presented in (5.5) does not require any approximations. The exact weighting we use is:

$$w(s,\mathbf{u})=\begin{cases}1 & Q_{tot}(\boldsymbol{\tau},\mathbf{u},s)<y_i\\ \alpha & \text{otherwise.}\end{cases}$$

We refer to it as **Optimistically-Weighted QMIX (OW)**.

In a Deep RL setting, QMIX implicitly weights the joint-actions proportional to their execution by the behaviour policies used to fill the replay buffer. This forces QMIX to make trade-offs in its Q-value approximation that are directly tied to the exploration strategy chosen. However, as we have shown earlier, this can lead to poor estimates for the optimal joint action and thus yield suboptimal policies. Instead, Weighted QMIX separates the weighting of the joint actions from the behaviour policy. This allows us to focus our monotonic approximation of Q^* on the *important* joint actions, thus encouraging better policies to be recovered irrespective of the exploration performed.

5.7 Related Algorithms

In this section, we will describe related methods and algorithms from the literature and outline the connections between them and Weighted QMIX.

5.7.1 Relationship to Actor-Critic methods

Weighted QMIX bears many similarities to an off-policy actor-critic algorithm, if we view $\hat{Q}^*$ as the critic and the policy implied by Q_{tot} as the actor. Define the deterministic QMIX greedy policy (assuming full observability to simplify the presentation) as:

$$\pi_w(s) = \begin{pmatrix} \arg\max_{u_1} Q_1(s, u_1) \\ \dots \\ \arg\max_{u_n} Q_n(s, u_n) \end{pmatrix}.$$

Weighted QMIX trains $\hat{Q}^*$ to approximate Q^{π_w}, the Q-values of this policy. This is also an approximation to Q-learning since $\pi_w \approx \arg\max \hat{Q}^*$. Viewed in this manner, Weighted QMIX is similar to MADDPG [233] with a single critic, except for how the actors are trained. MADDPG trains each agent's policy π_a via the multi-agent deterministic policy gradient theorem , whereas Weighted QMIX trains the policy indirectly by training Q_{tot} via the weighted loss in (5.9). Peng et al. [194] take advantage of the extra structure present in our CTDE cooperative MARL setting and extend MADDPG to more effectively utilise it. Using ideas from the value

function based literature [234, 153, 235] they factorise the critic, and propose a *centralised* policy gradient estimator that follows the policy gradient of the joint-action as opposed to a per-agent policy gradient. FACMAC is then even more similar to Weighted QMIX than MADDPG, and crucially differs again in how the agents are trained. Multi-agent Soft Actor Critic (MASAC) [236, 237] is another off-policy actor-critic based approach that instead trains the actors by minimising the KL divergence between each agent's policies and the joint-action Q-values. These actor-critic based approaches (as well as COMA [137] and LIIR [191]) do not restrict the class of joint action Q-values they can represent, which theoretically allows them to learn an optimal value function and policy. However, in practice they do not perform as well as QMIX, perhaps due to relative overgeneralisation [238] or the presence of bad local minima. MAPPO [239] is an extremely competitive actor-critic method, which again does not restrict the class of joint-action Q it can represent. Instead of learning Q explicitly, they use GAE(λ) to both take advantage of multi-step returns and to only have to learn and represent $V(s)$. In contrast to QMIX, more tuning and care is required over the actual implementation and setup (e.g. masking out dead agents in SMAC can make a huge difference in performance [239]). However, MAPPO generally performs strongly across a wide range of settings and represents a compelling alternative to the Q-learning based approaches.

5.7.2 Relationship To QTRAN

QTRAN [229] is another Q-learning based algorithm that learns an unrestricted joint action Q function and aims to solve a constrained optimisation problem in order to decentralise it. However, it is empirically hard to scale to more complex tasks (such as SMAC).

QTRAN represents Q_{tot} like VDN:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i(\tau^i, u^i; \theta^i).$$

Additionally, QTRAN learns an unrestricted joint-action $\hat{Q}^*(\boldsymbol{\tau}, \mathbf{u}; \phi)$ and a state-dependent bias $V(s)$.

$\hat{Q}^*$ is trained through the usual Q-learning loss using $\hat{\mathbf{u}} = \arg\max Q_{tot}$ to propose the argmax action which is evaluated by the target $\hat{Q}^*$ during bootstrapping. Q_{tot} is trained to match $\hat{Q}^*$ for the proposed optimal action and to overestimate for every other action.

For a given transition (labelled i) in the minibatch, QTRAN has 3 losses. Letting $\hat{\mathbf{u}} := \arg\max Q_{tot}(\boldsymbol{\tau}, \cdot)$, they are:

$$\begin{aligned} L_{td} &= (\hat{Q}^*(s, \boldsymbol{\tau}, \mathbf{u}; \phi) - y_i)^2, \\ &\quad \text{where } y_i := r + \gamma \hat{Q}^*(s', \boldsymbol{\tau'}, \arg\max Q_{tot}(\boldsymbol{\tau'}, \cdot); \phi^-). \\ L_{opt} &= (Q_{tot}(\boldsymbol{\tau}, \hat{\mathbf{u}}) + V(s) - \hat{Q}^*(s, \boldsymbol{\tau}, \hat{\mathbf{u}}; \phi^-))^2, \\ L_{nopt} &= (\min\{0, Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) + V(s) - \hat{Q}^*(s, \boldsymbol{\tau}, \mathbf{u}; \phi^-)\})^2. \end{aligned}$$

The final loss for QTRAN is then $L_{td} + \lambda_{opt} L_{opt} + \lambda_{nopt} L_{nopt}$.

We can view QTRAN as specific choices of the 3 components of Weighted QMIX, which allows us to better understand its trade-offs and empirical performance in relation to WQMIX. However, the motivations for QTRAN are significantly different. $\mathbf{Q_{tot}}$ is represented using VDN instead of QMIX, and trained using $\hat{Q}^*$ as the target (instead of y_i). This can limit QTRAN's empirical performance because QMIX generally outperforms VDN [232].

$\hat{\mathbf{Q}}^*$ is a network that takes as input an embedding of all agents' chosen actions and observations (and additionally the state if it is available). The agent components share parameters with the agent networks used to approximate Q_{tot}.

Weighting. The weighting function is as follows:

$$w(s, \mathbf{u}) = \begin{cases} \lambda_{opt} & \mathbf{u} = \hat{\mathbf{u}} \\ \lambda_{nopt} & Q_{tot}(s, \mathbf{u}) < \hat{Q}^*(s, \mathbf{u}) \\ 0 & \text{otherwise,} \end{cases} \tag{5.12}$$

where $\hat{\mathbf{u}} = \arg\max Q_{tot}(s, \cdot)$. Using too small a weight in the weighting can have a substantial negative effect on performance. However, using a 0 weight for overestimated Q-values is a fundamental part of the QTRAN algorithm.

Son et al. [240] propose QOPT, an extension of QTRAN, which also learns an unrestricted $\hat{Q}^*$ and utilises an optimistic-style weighting to train Q_{tot} that is

represented by QMIX. In contrast to Weighted QMIX, whose $\hat{Q}^*$ does not share any parameters with Q_{tot}, the unrestricted joint-action Q-values of QOPT are obtained through an unrestricted mixing network (the weights are not constrained to be non-negative) which takes as input the agent utilities. The weighting function used for training Q_{tot} is similar to the weighting for CW-QMIX, in which a smaller weighting is used for the the joint-actions whose Q-values are estimated as lower than the Q-values for the approximate best joint-action ($\hat{\mathbf{u}}^*$). Son et al. [241] further extend QOPT and QTRAN to QTRAN++ with changes to these 3 components to improve empirical performance.

5.7.3 QPLEX

Wang et al. [242] propose QPLEX, which also expands the class of joint-action Q-values that can be represented. They acheive this by decomposing Q_{tot} as a sum of the value function (maximum of Q_{tot}) and a non-positive advantage function. Crucially, this advantage function is 0 for the joint-action in which every agent maximises their own utilities ($\hat{\mathbf{u}}^*$ in our notation).

QPLEX represents the joint-action Q as follows:

$$
\begin{aligned}
A_i(\tau^i, u^i) &:= Q_i(\tau^i, u^i) - \max Q_i(\tau^i, \cdot) \leq 0 \\
Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) &= V_{tot}(\boldsymbol{\tau}) + A_{tot}(\boldsymbol{\tau}, \mathbf{u}) \\
&= \sum_{i=1}^{n} \max Q_i(\tau^i, \cdot) + \sum_{i=1}^{n} \lambda_i(\boldsymbol{\tau}, \mathbf{u}) A_i(\tau^i, u^i), \\
&\text{where } \lambda_i(\boldsymbol{\tau}, \mathbf{u}) > 0.
\end{aligned}
$$

Note that we have dropped the dependence of A_i, λ_i, V_{tot} and A_{tot} on the state s to simplify the presentation.

Since the advantage is non-positive, this ensures consistency between the agent's greedy joint-action ($\hat{\mathbf{u}}^*$), and the true maximum joint-action of Q_{tot}. In contrast, Weighted QMIX does not maintain this consistency. Our experimental results show that despite not restricting the class of joint-action Q-values that can be represented, QPLEX can struggle to learn a good policy in some environments like

QMIX. Furthermore, there are many architectural differences between (Weighted) QMIX and QPLEX, particularly in the use of an attention style architecture in the mixing network.

5.8 Related Work

In this section we briefly outline some closely related work to Weighted QMIX.

Mahajan et al. [158] point out some limitations of QMIX arising from its monotonic factorisation. Specifically, they show that for a specific choice of matrix game, QMIX can fail to learn an optimal policy if each joint action is visited uniformly, which corresponds to our idealised tabular setting in Section 5.3. Additionally, they show that a lower bound on the probability of recovering the optimal policy increases for an ϵ-greedy policy as ϵ increases. This is proved by considering the weighting on each joint-action induced by the exploration policy. Training on a replay buffer of experiences produced by a fixed ϵ-greedy policy corresponds to placing a higher weighting on the *greedier* joint actions, with a more uniform weighting being recovered the higher ϵ is. By contrast, our weighting is independent of the exploration strategy, adding flexibility.

Optimistic Weighting uses a smaller weighting when decreasing Q_{tot} estimates, similarly to Hysteretic Q-learning [171] which uses a smaller learning rate when decreasing value estimates of independent learners.

Wang et al. [243] analyse the properties of VDN in a Fitted Q-Iteration framework similar to the analysis we conduct for QMIX. Although their analysis considers a non-uniform weighting, induced by the behavioural policy, it does not affect the theoretical results since they are always presented in terms of the expectation over state-actions according to the behavioural policy. Instead, the focus of the analysis is on decomposing the update rule for VDN to show it is doing an implicit credit assignment akin to COMA. Under a fully-uniform distribution over the state-action space (as in our analysis), they also show that linear value function factorisation can diverge. Additionally, despite using a Dec-ROMDP for their analysis, the assumptions made and the results derived are the same if they considered a MMDP.

They also show that an increased representational capacity, can be beneficial in an offline RL setting, in which the algorithms cannot affect the implicit weighting over the state-action space.

5.9 Experimental Results

In this section we present our experimental results on the Predator Prey task considered by Boehmer et al. [181] and on a variety of SMAC[1] scenarios. More details about the implementation of each are included in Appendix C.1. For every graph we plot the median and shade the 25%-75% quartile. Code is available at `https://github.com/oxwhirl/wqmix`.

5.9.1 Ablations

In the following experiments we include the results of the following ablations:

- **QMIX + $\hat{Q}^*$** - Weighted QMIX in which the weighting $w = 1$ for all state-actions, and the centralised Q is used during bootstrapping.
- **QMIX + CW** - Weighted QMIX with the centralised weighting without the use of the centralised Q during bootstrapping. Instead QMIX's Q-values are used.
- **QMIX + OW**: Same as above, except the optimistic weighting is used.

5.9.2 Predator Prey

We first consider a partially-observable Predator Prey task involving 8 agents [181], which was designed to test coordination between agents by providing a punishment of -2 reward when only a single agent (as opposed to two agents) attempt to capture a prey. Algorithms which suffer from *relative overgeneralisation* [244, 238], or which make poor trade-offs in their representation (as VDN and QMIX do) can fail to solve this task.

[1] We utilise `SC2.4.6.2.69232` (the same version as [232]) instead of the newer `SC2.4.10`. Performance is **not** comparable across versions.

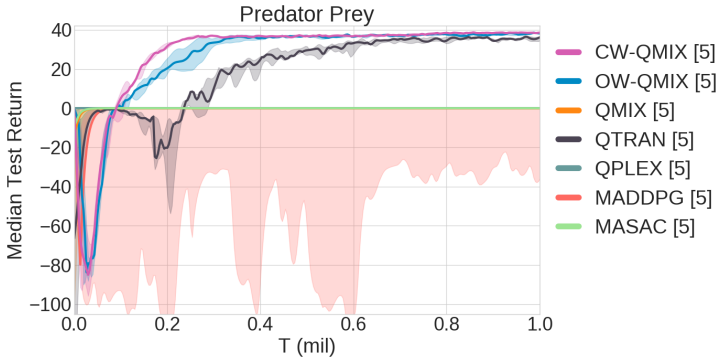


Figure 5.2: Median test return for the predator prey environment, comparing Weighted QMIX and baselines.

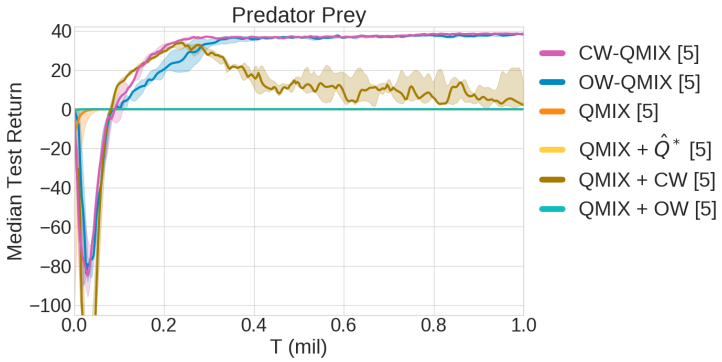


Figure 5.3: Median test return for the predator prey environment, comparing Weighted QMIX and 3 ablations.

As shown in [181], QMIX fails to learn a policy that achieves positive test reward, and our results additionally show the same negative results for MADDPG and MASAC. Interestingly, QPLEX also fails to solve the task despite not having any restrictions on the joint-action Q-values it can represent, suggesting difficulties in learning certain value functions. Figure 5.2 shows that both CW-QMIX and OW-QMIX solve the task faster than QTRAN.

Ablation experiments for Predator Prey.

Our ablation experiments in Figure 5.3 show the necessity for both $\hat{Q}^*$and a weighting in order to solve this task. As expected QMIX $+\hat{Q}^*$ is unable to solve this task due to the challenges of relative overgeneralisation. The use of a uniform weighting in the projection prevents the learning of an optimal policy in which two agents coordinate to capture a prey. Thus, even if $\hat{Q}^*$can theoretically represent the Q-values of this optimal policy, the QMIX agents are unable to recover it.

Figure 5.3 also shows that QMIX with just a weighting in its projection (and no $\hat{Q}^*$) is unable to successfully solve the task.

5.9.3 SMAC

Robustness to increased exploration

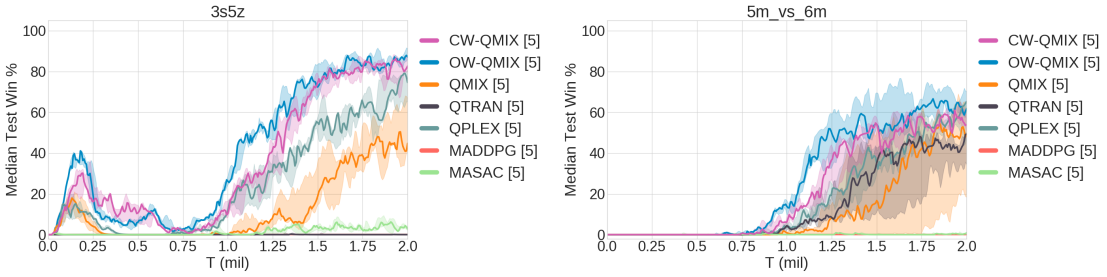


Figure 5.4: Median test win % with an increased rate of exploration.

QMIX is particularly brittle when there is significant exploration being done, since it then tries (and fails) to represent the value of many suboptimal joint actions. We would like our algorithms to efficiently learn from exploratory behaviour. Hence, we use two of the easier SMAC maps, *3s5z* and *5m_vs_6m*, to test the robustness of our algorithms to exploration. We use an ϵ-greedy policy in which ϵ is annealed from 1 to 0.05 over 1 million timesteps, increased from the $50k$ used in [232]. Figure 5.4 shows the results of these experiments, in which both Weighted QMIX variants significantly outperform all baselines. Figure 5.7 (Left) additionally compares the performance of QMIX and Weighted QMIX on `bane_vs_bane`, a task with 24 agents, across two ϵ-schedules. We can see that both variants of Weighted QMIX are able to solve the task irrespective of the level of exploration, whereas QMIX fails to do so.

Ablation experiments testing robustness to increased exploration.

Figure 5.5 shows the results of further ablation experiments, confirming the need for both $\hat{Q}^*$and a weighting to ensure consistent performance. Note in particular that combining QMIX with a weighting results in significantly worse performance in *5m_vs_6m*, and no better performance in *3s5z*.

Effect of α on performance.

Figure 5.6 shows the effect of varying α in the weighting function for CW-QMIX and OW-QMIX. We can see that if α is too low, performance degrades considerably.

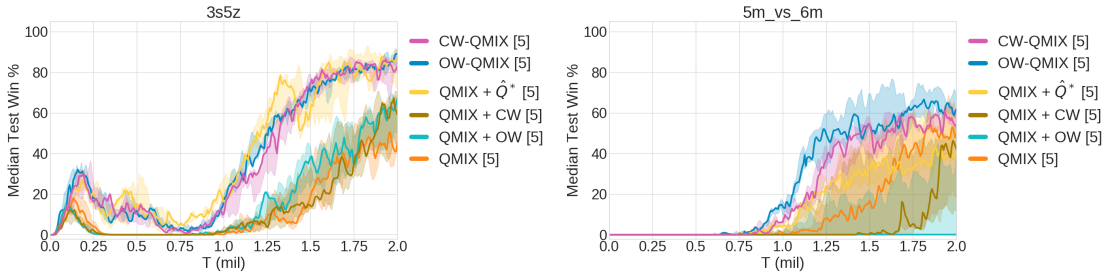


Figure 5.5: Median test win % with an increased rate of exploration, comparing Weighted QMIX and 3 ablations.

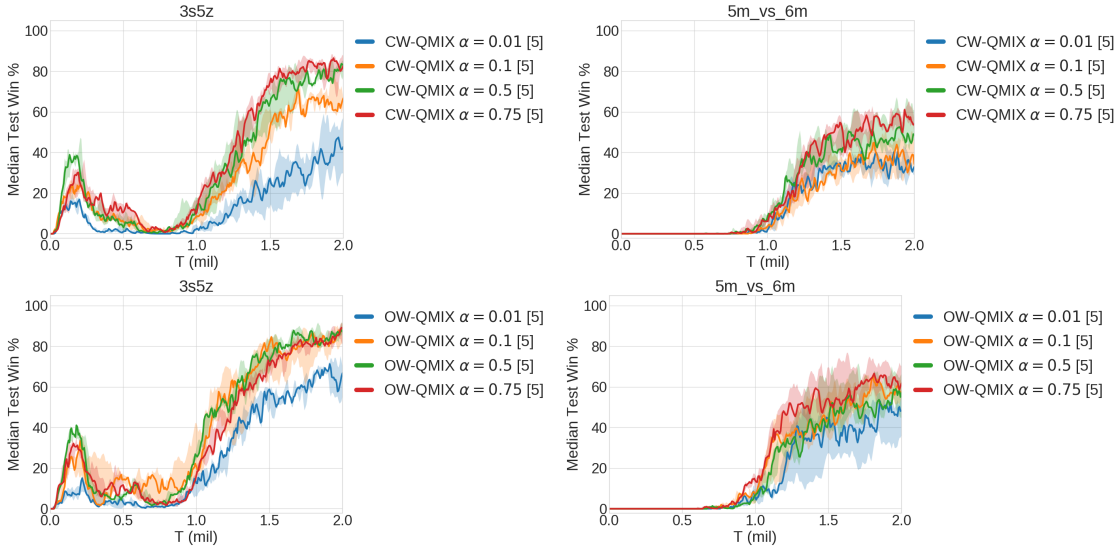


Figure 5.6: Median test win % with an increased rate of exploration. **Above:** The effect of varying α for CW-QMIX. **Below:** The effect for OW-QMIX.

Necessity of increased exploration

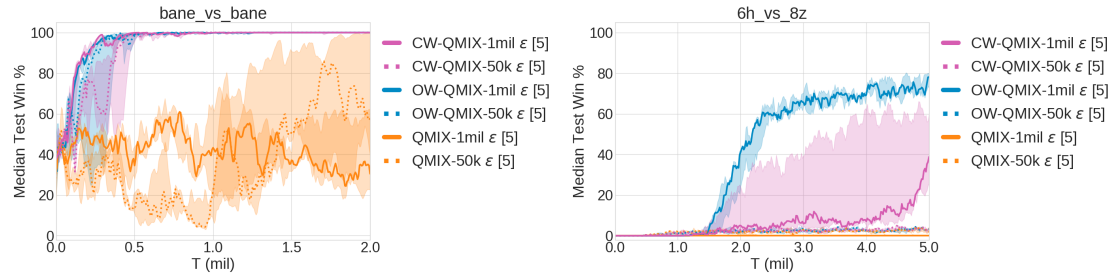


Figure 5.7: Median test win % with 2 exploration schedules on `bane_vs_bane` and *6h_vs_8z*.

Next, we compare our method on the challenging *6h_vs_8z*, which is classified as a *super hard* SMAC map due to current method's poor performance [232]. Figure 5.7 (Right) compares QMIX and Weighted QMIX with two differing exploration

schedules (annealing ϵ over 50k or 1 million timesteps, denoted -50k ϵ and -1mil ϵ respectively). We can see that a larger rate of exploration is required, and only Weighted QMIX can successfully recover a good policy, demonstrating the benefits of our method for improving performance in a challenging coordination problem.

Limitations

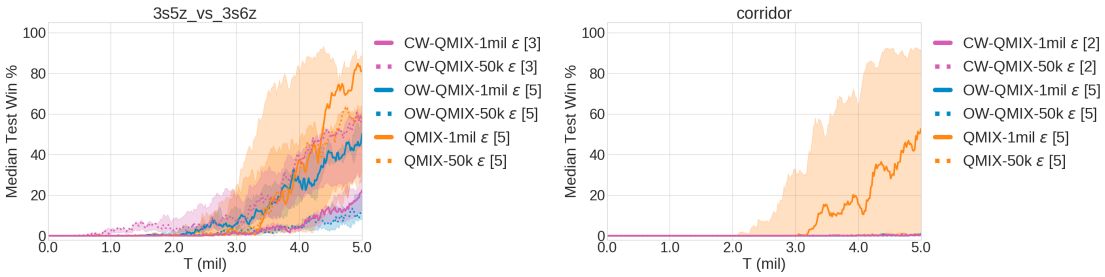


Figure 5.8: Median test win % with 2 exploration schedules on 2 *super-hard* SMAC maps.

Finally, we present results on 2 more *super hard* SMAC maps in order to show the limitations of our method in Figure 5.8. In *3s5z_vs_3s6z* we observe that the extra exploration is not helpful for any method. Since QMIX is almost able to solve the task, this indicates that both exploration and the restricted function class of QMIX are not limiting factors in this scenario. On *corridor* we see that only QMIX with an extended exploration schedule manages non-zero performance, showing the importance of sufficient exploration on this map. The poor performance of Weighted QMIX shows that the extra complexity of our method (notably learning $\hat{Q}^*$) can sometimes harm performance, indicating that closer attention must be paid to the architecture and weighting functions.

Figure 5.9 shows the results on *corridor* for Weighted QMIX with a slightly modified architecture for $\hat{Q}^*$. The significantly improved performance for Weighted QMIX indicates that the architecture of $\hat{Q}^*$ is partly responsible for the regression in performance over QMIX.

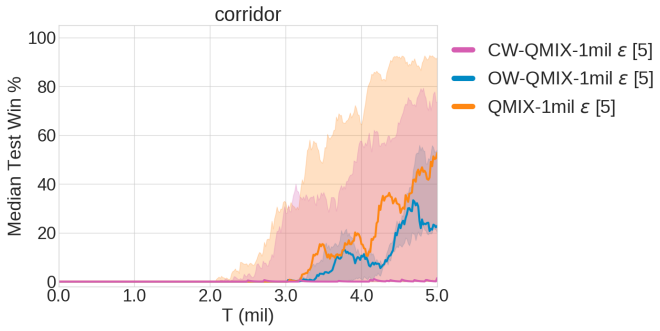


Figure 5.9: Median test win % on *corridor* with a modified architecture for $\hat{Q}^*$.

5.10 Utility of a Weighting

In this section, we aim to better understand the role a weighting plays, and its potential applications as part of other algorithms. Specifically, we first revisit our original motivations for introducing a weighting in the context of algorithmic advances such as QPLEX. We then expand our analysis to better capture the training setup we adopt in the Deep RL regime, and demonstrate that it behaves as expected in a simple task. Our results highlight the need for an *explicit* weighting that we have full control over, and show that without such an appropriate weighting algorithms such as QPLEX can fail on our simple task. In order to investigate why the weighting is helping, we introduce another approach which we will call Corrected QMIX (CorQMIX) that utilises the same type of factorisation as QPLEX but behaves very differently. Our results suggest that the exact parametrisation of the joint-action Q is extremely important to a method's performance, beyond the class of joint-action Q-values that are representable. Finally, we investigate weighted versions of QPLEX and CorQMIX and show that the weighting can improve performance over Weighted QMIX by *focussing* the optimisation on the more important Q-values. Our results show that a weighting can be beneficial in changing the joint-action Q-values that are being learned beyond those found at the global minima of the loss. However, our results also strongly suggest that there is still much room for improvement in our current weighting functions and in our understanding of its interaction with the parametrisation of the joint-action Q-values.

5.10.1 Motivations For a Weighting

In the previous section's results and throughout the literature, we have seen that methods such as VDN and QMIX can fail catastrophically on some tasks due to their restricted representational capacities. In order to tackle this issue there are two possible approaches: increase the representational capacity of the method, or introduce a weighting into the loss function. The first is the approach taken by QPLEX, which expands the class of joint-action Q-values that are representable. QPLEX's smart factorisation enables it to theoretically represent *any* joint-action Q-value, whilst maintaining consistency between the greedy policies of the centralised joint-action Q and the individual agent's Q-values. From a purely theoretical perspective, QPLEX is then an ideal extension over the previous joint-action Q-learning based approaches such as QMIX. The second approach, which we proposed and investigated the theoretical foundations of earlier in this chapter, is to introduce a weighting function into our loss in order to change the solutions learnt by QMIX. A downside of our approach however, is that Weighted QMIX introduces additional components and complexity over QMIX. We now have a weighting function which we must appropriately specify and its hyperparameters we must tune. We also have an entirely new model, the unrestricted $\hat{Q}^*$, whose choice of architecture we have already demonstrated can have a significant affect on performance. The other major downside, is that we lose the consistency between our centralised and decentralised policies.

A natural question to ask, is why then would we favour Weighted QMIX over an algorithm like QPLEX? The short answer is in its empirical performance. Although QPLEX exhibits strong performance on many complex SMAC scenarios, it can still fail in relatively simple ones such as the Predator Prey task in Figure 5.2. This is despite having an unrestricted representational capacity. In contrast, Weighted QMIX performs incredibly well in these simpler scenarios where the representational capacity of QMIX can prevent it from solving the task efficiently.

In this section we begin an investigation into the question:
How can a weighting be beneficial in our cooperative MARL setting?

To begin our investigation, we first expand our analysis to include the *implicit* weighting that is present when training in an RL setup.

5.10.2 Implicit vs Explicit Weighting

In Section 5.3 we analysed an operator which represented QMIX:

$$\arg\min_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} (\mathcal{T}^* Q_{tot}(s, \mathbf{u}) - q(s, \mathbf{u}))^2. \tag{5.13}$$

Importantly, this viewpoint makes the simplifying assumption of a uniform weighting across **all** joint-actions. In the vast majority of practical scenarios this is an unrealistic assumption, since we cannot generate valid transitions for any state-action pair of our choosing (we do not have access to a generative model of the environment). Instead, we can only train on the data stored in our replay buffer, that has been gathered by our exploration/behavioural policies. Thus, there is an *implicit* weighting defined by the combination of behavioural policies and replay buffer sampling strategies. To focus our analysis we can assume a uniform sampling strategy from the replay buffer, as is common in many Deep MARL algorithms including all our previous experiments. Then, only the exploration/behavioural policies affect our weighting. We can refer to this implicit weighting as $b(s, \mathbf{u})$, and introduce it to our analysis:

$$\arg\min_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} b(s, \mathbf{u})(\mathcal{T}^* Q_{tot}(s, \mathbf{u}) - q(s, \mathbf{u}))^2. \tag{5.14}$$

This weighting is similar to that used in [158] to prove that the probability QMIX can fail to recover an optimal policy increases with its rate of exploration in certain matrix games. Although Equation (5.14) differs from the Weighted QMIX operator in only the use of $b(s, \mathbf{u})$ as opposed to $w(s, \mathbf{u})$, this is an important difference [2].

The implicit weighting, b, is determined by the exploration/behavioural policies used. Therefore it is difficult to directly control or predict. In contrast, w is a user-definable part of the Weighted QMIX operator and is entirely independent

[2] we will ignore the dependence of w on other factors.

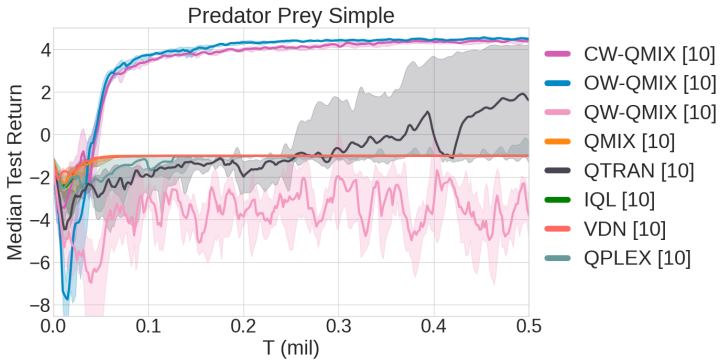


Figure 5.10: Performance of algorithms without any modifications.

of the exploration performed. Nonetheless, the affect b can have on the $q \in \mathcal{Q}^{mix}$ that is learnt by QMIX is the same as w.

We can also introduce this implicit weighting into our analysis of a Weighted QMIX operator:

$$\arg\min_{q \in \mathcal{Q}^{mix}} \sum_{\mathbf{u} \in \mathbf{U}} b(s, \mathbf{u}) w(s, \mathbf{u}) (\mathcal{T}^* Q_{tot}(s, \mathbf{u}) - q(s, \mathbf{u}))^2. \tag{5.15}$$

Which makes it clear that the combination of both b **and** w is responsible for what is returned by the operator or learnt by QMIX.

To empirically validate this, we utilise a simple 2-agent predator prey environment in which both agents must coordinate to capture the prey at the same time to obtain the positive $+5$ reward. However, if only a single agent is next to the prey a negative -0.5 reward is given. The decision to capture or not-capture results in the ordering over an agent's actions depending on the action of the other agent. This is a significantly simplified version of the task we used in Figure 5.2, which allows us to focus on whether an algorithm will solve the task or not. In this task, due to the coordination required, VDN and QMIX cannot perfectly represent the target Q-values.

Figure 5.10 shows the results of the Q-learning based approaches without modification on this task. Here QW-QMIX is our Weighted QMIX approach which uses a weighting function very close to that of QTRAN as described in Equation 5.12. We can see that only Weighted QMIX is able to solve this task reliably and efficiently. The next best performer is QTRAN, which can also be viewed as

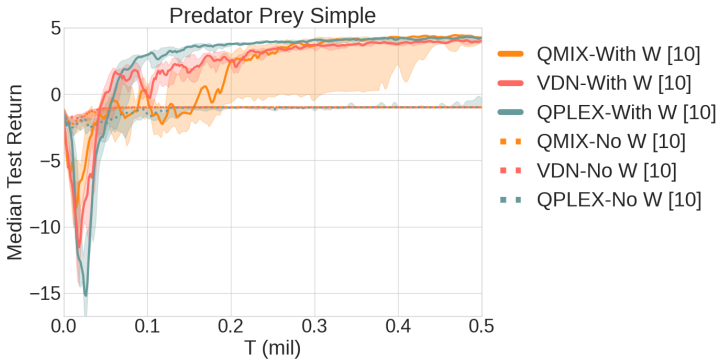


Figure 5.11: Comparing the performance of VDN, QMIX and QPLEX with and without the environment specific simple weighting.

utilising a weighting. Interestingly, applying that weighting to our Weighted QMIX setup results in significantly worse performance. We hypothesise that the use of QMIX as opposed to VDN as the learner for the decentralised agents makes the use of a weighting with a 0 weight perform poorly. This is perhaps because the joint-action Q-values learnt by a VDN learner are more coupled with each other compared to those learnt by a QMIX learner.

In this simplified scenario, we can consider a simple weighting for this task that only applies at the transitions where a reward is given. We set $w(s, \mathbf{u}) = 0.1$ whenever the reward is negative which occurs when only one agent is next to the prey (indicating miscoordination since both agents are required to capture the prey).

Figure 5.11 shows that VDN, QMIX and QPLEX are unable to solve this relatively simple task without the use of this explicit weighting. For this simple task we can also graph the proportion of positive and negatively rewarding transitions that are being trained on. Note, that this doesn't exactly correspond with b at those pivotal states in which the agents are in a position to capture the prey, since they could both move away.

Figure 5.12 shows that without the use of an explicit weighting, less than 3% of the rewarding experiences are positive. This leads to a very small implicit weighting for the optimal joint-actions. In contrast, the same proportion grows much larger over training when utilising an explicit weighting. This is because the explicit weighting allows for the algorithms to learn to capture the prey together,

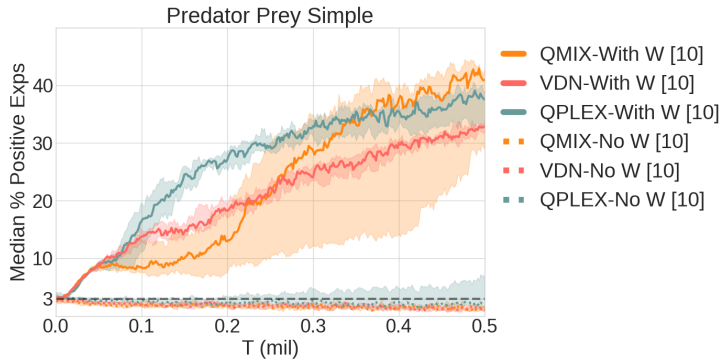


Figure 5.12: The percentage of positively rewarding experiences being trained compared to all rewarding transitions being trained. This graph compares this for VDN, QMIX and QPLEX with and without the weighting.

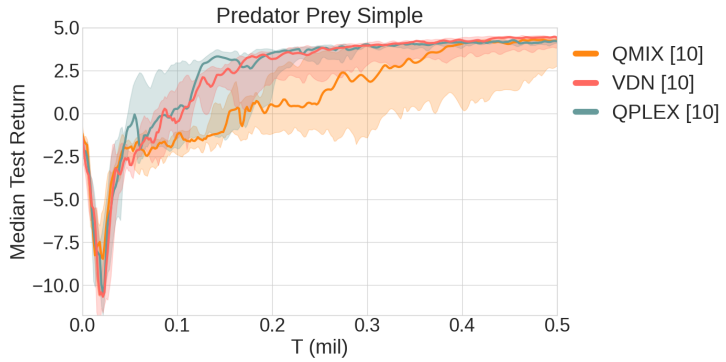


Figure 5.13: Performance of VDN, QMIX and QPLEX when the explicit weighting is only active if the percentage of positively rewarding experiences is under 10%.

resulting in an increasing proportion of positive experiences as the policies improve. Note however that the explicit weighting is not directly affecting the exploration performed. Instead, it is allowing the algorithms to learn effectively from the data they are already gathering, which is the same for all methods at the start of training.

Figure 5.13 shows that the same algorithms can learn when the explicit weighting w is removed if the proportion of positive to negative experiences is more than 10%. We can see that the algorithms exhibit the same performance as when trained with an explicit weighting throughout training. This highlights that a favourable b **can** lead to good solutions being learned. However, without the use of an explicit weighting, the algorithms are unable to learn from the initial implicit weighting b as shown in Figure 5.11. Due to the difficulties in influencing it directly, it is often useful to utilise an explicit weighting to ensure good performance irrespective of the implicit b.

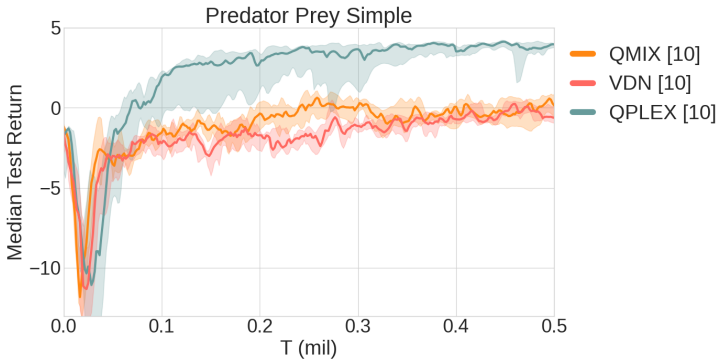


Figure 5.14: Performance of VDN, QMIX and QPLEX with the explicit weighting and a modified adversarial implicit weighting.

Figure 5.14 shows than an explicit weighting can make up for a bad implicit weighting. In this experiment, all algorithms are using the explicit weighting that enabled them to learn in Figure 5.11. However, we now modify the behavioural weighting to be more unfavourable. Specifically, we randomly drop positively rewarding transitions from our minibatch to ensure the proportion of positive to negative transitions is never above 3%. Note that this is an overestimate of the proportion of positively rewarding experiences for states in which it is possible for the agents to capture the prey and solve the task. We can see that despite this, QPLEX is able to recover a good policy. VDN and QMIX on the other hand, are unable to learn to solve the task under the same setup, but they do perform marginally better than without a weighting at all.

5.10.3 Parametrisation

The previous results demonstrate that QPLEX can fail to learn in many simple situations without the use of an explicit weighting. This is surprising, since our original motivations for introducing a weighting function were focussed on QMIX. Specifically, we introduced a weighting into our loss to change the $q \in \mathcal{Q}^{mix}$ that QMIX learns at the minimum of its loss. This was done because QMIX cannot represent *all* joint-action Q-values, and hence to ensure its learnt solution recovers the correct argmax we must modify the loss we use to train it. However, QPLEX does not have the same representational constraints. In theory, QPLEX's factorisation

allows it to represent *any* joint-action Q-value whilst maintaining the consistency between the greedy decentralised agent policies and the centralised joint-action policy. Hence, our original motivations for the weighting do not apply. At the minimum of its loss, QPLEX will always recover the target Q-value and thus the correct argmax. Nonetheless, we have demonstrated on a simple task that the weighting can be extremely beneficial. In the remainder of this section we aim to investigate why.

In order to do so, we must first introduce two additional methods whose factorisation is heavily based on QPLEX. These methods factorise the joint-action Q in the same manner: taking away a non-negative quantity from the Q-value of the proposed argmax action. This enables them to theoretically represent *any* joint-action Q whilst maintaining the consistency between the centralised and decentralised policies, the same as QPLEX. The reason we introduce and consider them, is to consider the affect the exact parametrisation of the joint-action Q has on performance.

CorQMIX-V:

The first method we consider is Corrected QMIX with V (CorQMIX-V). Letting $\hat{\mathbf{u}}$ be the argmax joint-action of the decentralised agent policies, we represent Q_{tot} as:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) = \begin{cases} V(s) & \mathbf{u} = \hat{\mathbf{u}}, \\ V(s) - |Q(\boldsymbol{\tau}, \mathbf{u}, s)| & \text{otherwise.} \end{cases} \tag{5.16}$$

We could also write it more compactly as:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) = V(s) - \mathbb{1}\{\mathbf{u} = \hat{\mathbf{u}}\}|Q(\boldsymbol{\tau}, \mathbf{u}, s)|.$$

In order to stay as close to QMIX as possible, we represent Q in the same manner as QMIX except we do not enforce monotonicity in the mixing network. This is achieved by not taking the absolute value of the hypernetworks output. $V(s)$ is represented as a simple MLP taking the state as input.

CorQMIX-Q:

Similarly we consider Corrected QMIX with Q (CorQMIX-Q), which bears even more resemblance to QPLEX.

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s) = \begin{cases} Q(\boldsymbol{\tau}, \hat{\mathbf{u}}, s) & \mathbf{u} = \hat{\mathbf{u}}, \\ Q(\boldsymbol{\tau}, \hat{\mathbf{u}}, s) - |Q(\boldsymbol{\tau}, \mathbf{u}, s)| & \text{otherwise.} \end{cases} \tag{5.17}$$

The architecture for Q is the same as for CorQMIX-V.

To maintain as much similarity as possible between the methods we will consider we use the same architecture for the unrestricted $\hat{Q}^*$ of Weighted QMIX instead of the feedforward MLP.

We introduce CorQMIX-V and CorQMIX-Q in order to compare and test the importance of a function's parametrisation. Previously, we have primarily focussed on a factorisations theoretical representational capacity. However, we have already seen in Figures 4.17 and 5.9, that the exact parametrisation for a method can have a large impact on performance. From the perspective of supervised learning, this is not at all surprising (it is perhaps even considered foundational knowledge at this point). An RNN can perform much worse than an LSTM, even though both have no limits on their representational capacity. Similarly, very deep networks are extremely difficult to train successfully without the use of skip connections or the use of batchnorm such as in ResNet. And so, in our specific cooperative MARL setting, the exact parametrisation we use for our joint-action Q-value could also affect how successfully it minimises its loss and its overall performance.

5.10.4 Contextual Matrix Game Results

In order to compare the methods, we utilise a simple contextual matrix game setup. In our experiments there are 10 different states that are sampled from uniformly for each episode. The rewards are randomly generated integers between -10 and 10. For each state, there is a single unique maximum action with a value of 10, all other entries provide 9 reward or less. In the following experiments we use 3 or 4 agents (indicated by the title of the graph), each with 3 actions each.

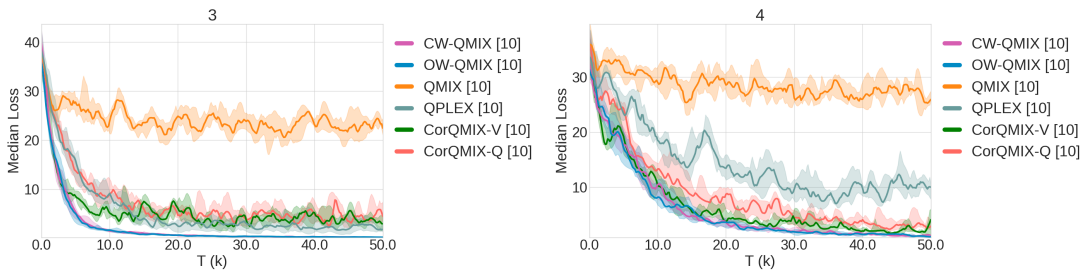


Figure 5.15: Loss of the methods during training.

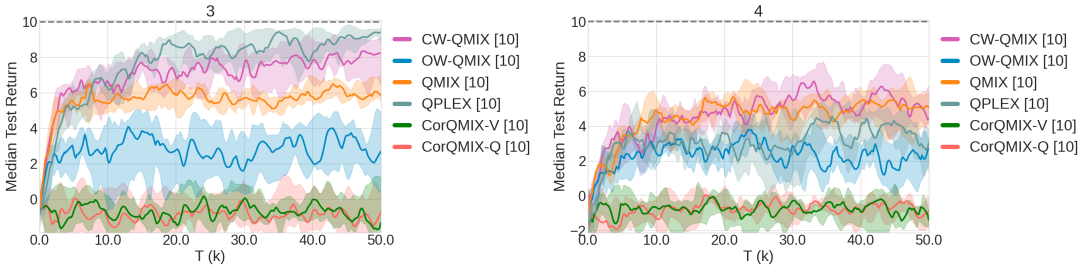


Figure 5.16: Testing performance of all algorithms.

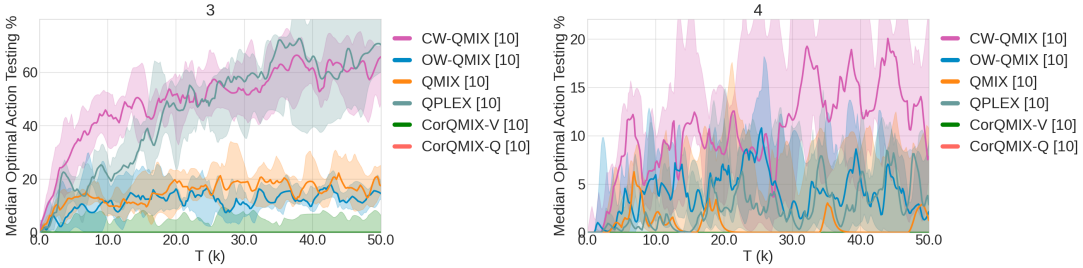


Figure 5.17: % of testing episodes the methods take the optimal action.

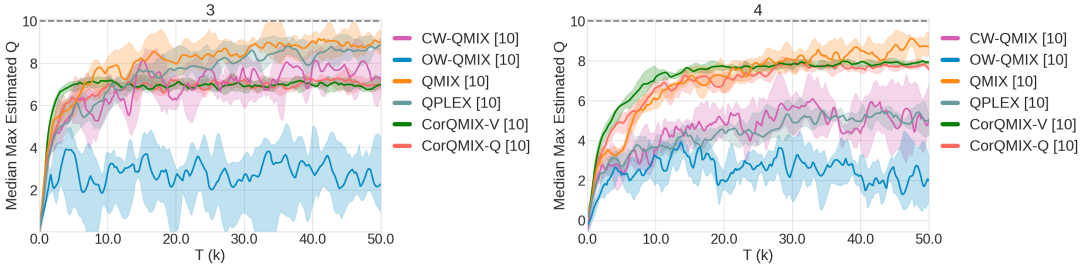


Figure 5.18: Maximum estimated Q for the methods.

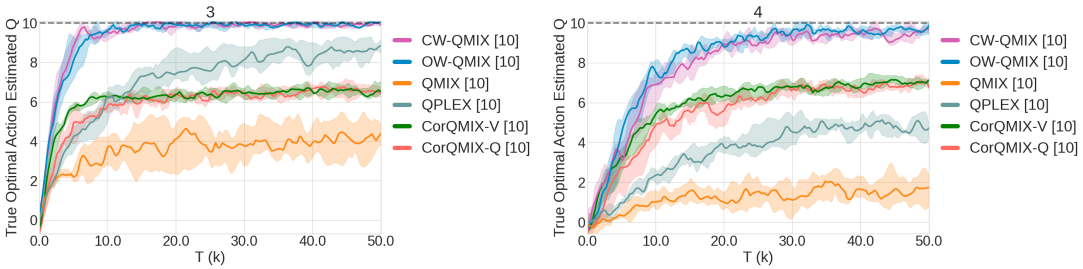


Figure 5.19: Q-value estimate for the true optimal action.

We can see from the results in Figure 5.15 that the different methods minimise their respective losses quite differently to each other. In particular, Weighted QMIX has the lowest loss, followed by CorQMIX and then QPLEX despite all 3 methods being able to represent the same class of joint-action Q-values. Figures 5.16 and 5.17 show that not only does CorQMIX fail to recover the correct argmax in the vast majority of runs, the joint-action it does suggest as the argmax is also quite far from optimal. We can also see in Figure 5.18 that CorQMIX achieves a reasonable estimate for the maximum estimated Q-value (even though it is a completely wrong estimate for that argmax action), but that it underestimates the true optimal joint-actions Q-value in Figure 5.19.

The stark difference between the performance of QPLEX and CorQMIX demonstrates that the parametrisation of the joint-action Q is extremely important. Furthermore, our results show that the ability to minimise the loss well does not correspond to achieving the desirable properties we would want in a Q-learning algorithm, namely recovering the correct argmax and correctly estimating the maximum Q-value. To investigate this further, we can look at the differences between the target Q-values the methods are trained against and their estimated values. We will refer to this as the distribution of errors, and will graph these values for a single seed (representative of the method's characteristics in general unless otherwise mentioned).

Looking at the distribution of errors for CorQMIX in Figures 5.21 and 5.20 we can see the presence of some severe overestimates for bad joint-actions. We can also see that its errors are (roughly) uniformly distributed across all joint-actions, but that the estimates for the best joint-actions are often underestimated relative to others. All of this suggests that a weighting function could be desirable in *focussing* our training budget on the more important joint-actions. This can be especially important in the Deep RL setting, in which we are training with small minibatches that almost certainly do not contain all possible joint-actions, or even our best estimated joint-action for a given state. This is so that optimising the loss for the

bad joint-actions (those with a Q-value far from the optimal Q-value) does not adversely affect our estimates for the more important ones.

Looking at the performance of Weighted QMIX, we can see it estimates $\hat{Q}^*$ significantly better than the other approaches. Its loss is the lowest across all methods, and it correctly estimates the true optimal Q-value. Although its parametrisation is similar to CorQMIX, it is different enough to lead to these large changes in its ability to learn the true Q-values. Despite $\hat{Q}^*$ learning the true Q-values quite well, the approach as a whole performs quite poorly. In particular, the argmax action which is suggested by the QMIX agents trained with the weighted loss, is usually not correct. This then also causes an underestimation in the maximum estimated Q-value, which is ultimately what would be used for bootstrapping. This suggests that the QMIX actors are not performing their role well, in identifying the correct argmax action. In particular, the Optimistic Weighting performs much worse than the Central Weighting. We hypothesise that this is due to the differences between the tabular setting, in which these weightings were originally conceived, and our Deep RL training setup. The use of small minibatches in our training can adversely affect both weightings. If we are only training on a small subset of the possible joint-actions, the loss we are optimising is quite different to what is being optimised in the tabular setting in which we were training over all joint-actions. e.g. We could place a small weighting of α on all the joint-actions for a given state, which would then essentially result in having no weighting at all. The Optimistic Weighting could suffer significantly more than the Central Weighting because of this, since the magnitude of the underestimation is an important part of the objective in the tabular setting. This is because it results in the underestimated joint-actions with a higher target being favoured over those joint-actions that are underestimated but have a low target Q-value. However, in the Deep RL setting the exact loss for a given minibatch can be quite different.

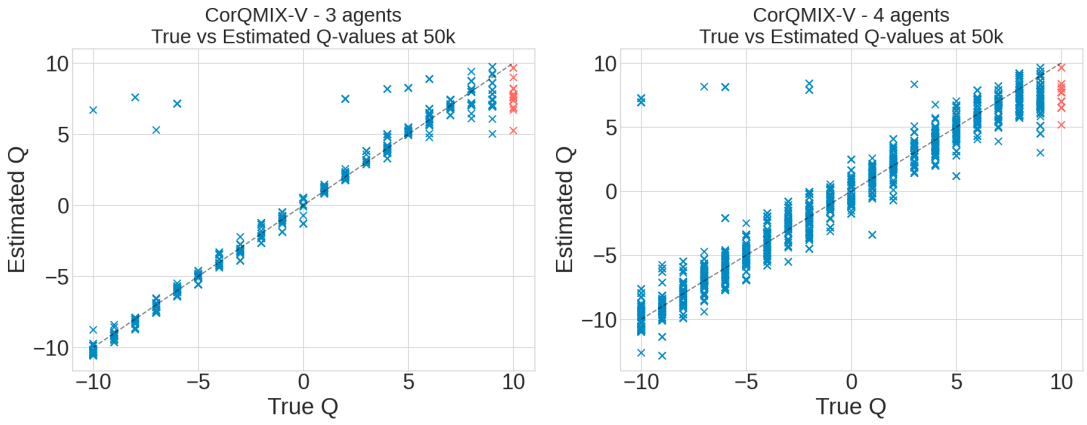


Figure 5.20: True Q-values compared to the estimated Q-values for CorQMIX-V at the end of training.

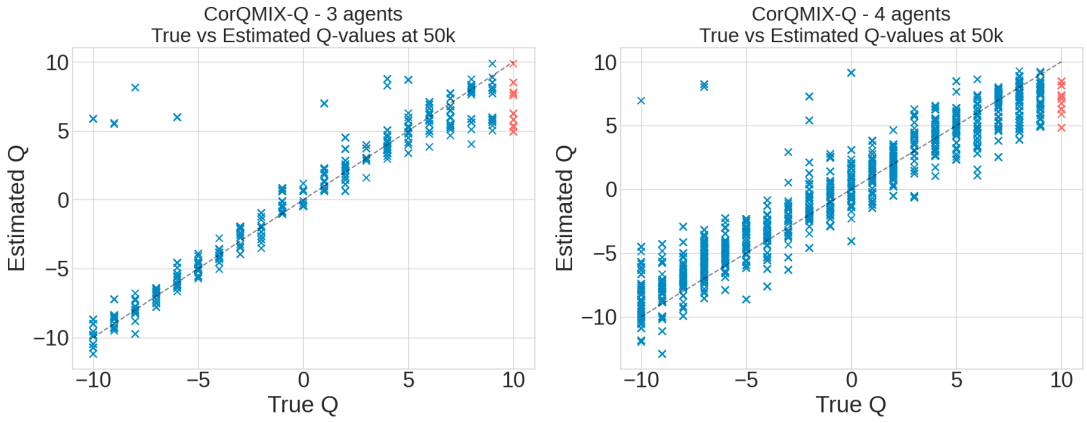


Figure 5.21: True Q-values compared to the estimated Q-values for CorQMIX-Q at the end of training.



Figure 5.22: True Q-values compared to the estimated Q-values for QMIX at the end of training.

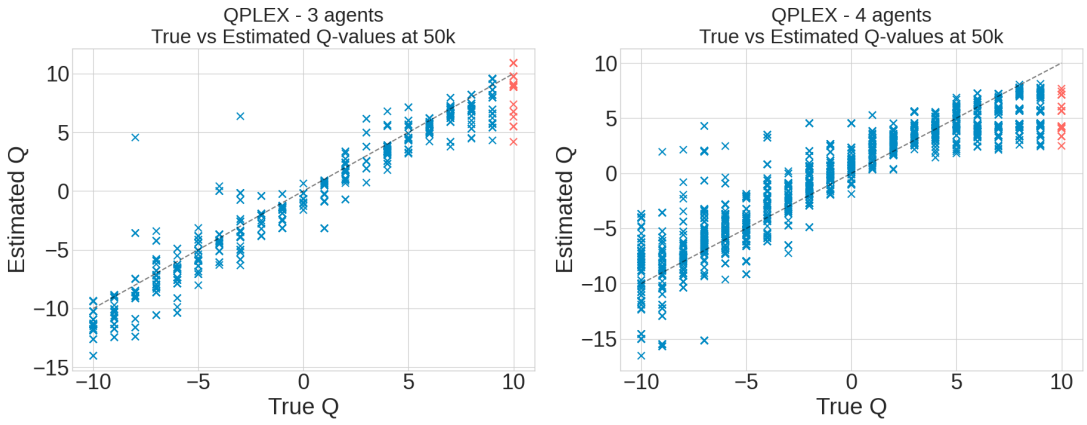


Figure 5.23: True Q-values compared to the estimated Q-values for QPLEX at the end of training.

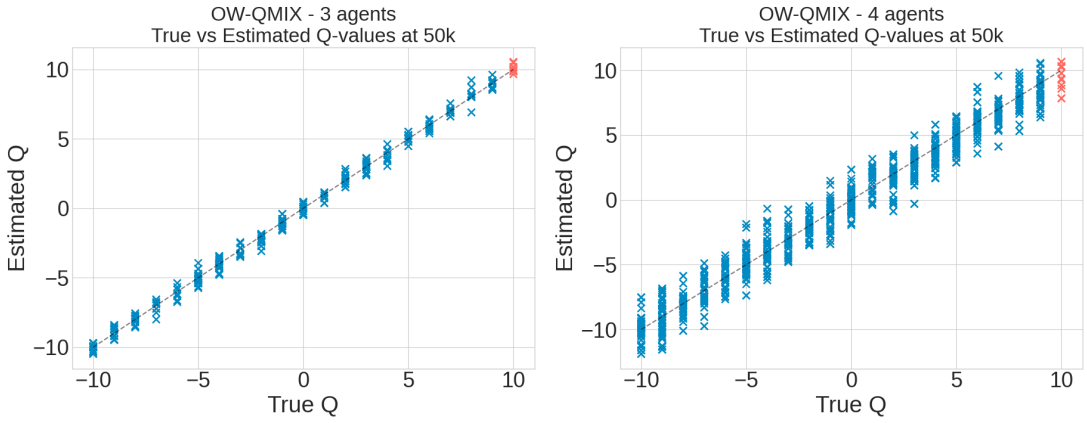


Figure 5.24: True Q-values compared to the estimated Q-values for OW-QMIX at the end of training.

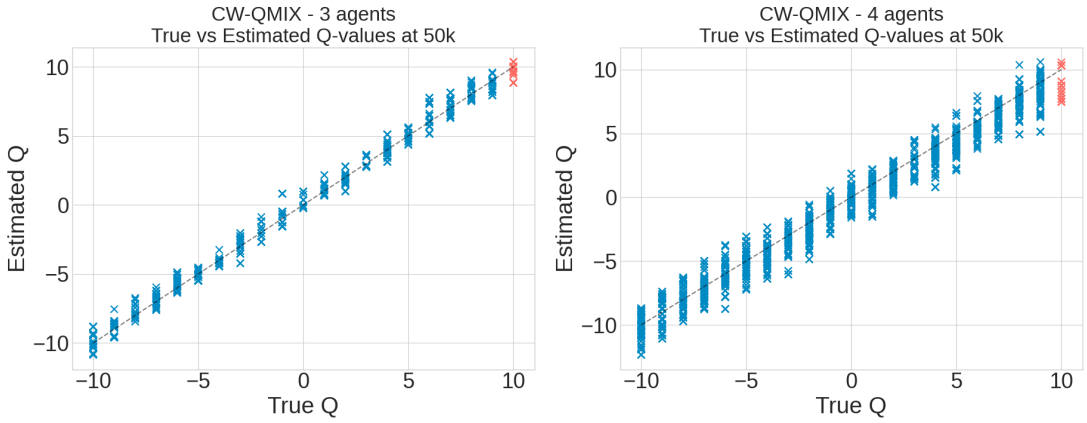


Figure 5.25: True Q-values compared to the estimated Q-values for CW-QMIX at the end of training.

5.10.5 Applications of a Weighting Beyond QMIX

The previous results strongly suggest that a weighting could be beneficial for QPLEX and CorQMIX, by modifying their losses to *focus* more on the important Q-value estimates. Note that this is different from the original motivation for introducing the weighting. In Section 5.4, we introduced the weighting in order to change the Q-values that QMIX learns *at the global minimum of its loss*. This was because the Q-values learnt at a global minimum of the loss could have incorrect argmaxes. In contrast, for QPLEX and CorQMIX, at the[3] global minimum of the loss the learnt Q-values can perfectly match the desired target Q-values. However, in the Deep RL setting where we are training on relatively small minibatches with networks of limited capacity, we cannot reasonably expect to reach the global minimum of the true expected loss. And so, we are now interested in changing the Q-values that are learnt even when the loss is not exactly minimised, in order to make up for deficiencies in our optimisation/learning dynamics. Similarly to the motivations for introducing $\hat{Q}^*$ in Weighted QMIX, that it can learn more accurate Q-value estimates, we also with to learn a separate unweighted joint-action Q-value. Our results in this section justify this motivation since they have shown that the $\hat{Q}^*$ of Weighted QMIX learns more accurate Q-values than both QPLEX and CorQMIX. The primary failing of the Weighted QMIX approach in our experiments is that the QMIX agents trained with the weighted loss do not recover the correct argmax. Thus, we consider replacing them with either QPLEX or CorQMIX agents that are also trained with the same weighted loss.

We first consider the Central Weighting. Note that in these graphs, "QMIX with CW" and "QMIX With OW" is equivalent to "CW-QMIX" and "OW-QMIX" respectively.

From the results in Figures 5.26, 5.27, and 5.28 we can see that Weighted QPLEX strongly outperforms Weighted QMIX on the smaller 3 agent, 3 action matrix games. For the larger 4 agent, 3 action matrix games it performs similarly, indicating its

[3]QMIX can have multiple global minimisers, whereas QPLEX and CorQMIX have unique global minimums.

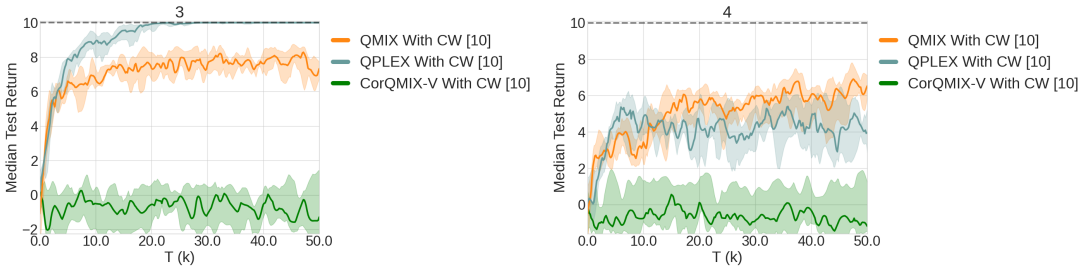


Figure 5.26: Testing performance of all algorithms.

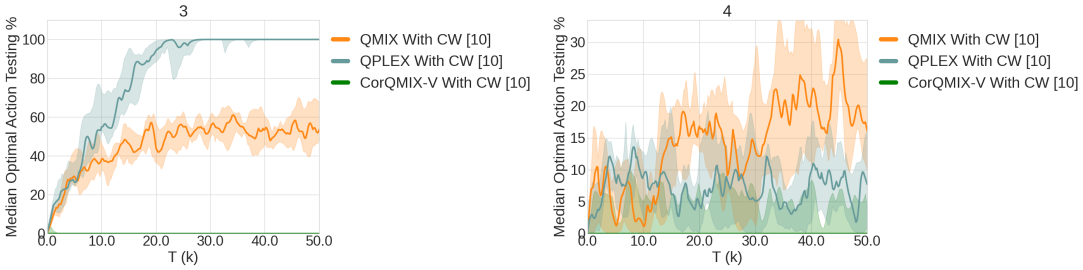


Figure 5.27: % of testing episodes the methods take the optimal action.

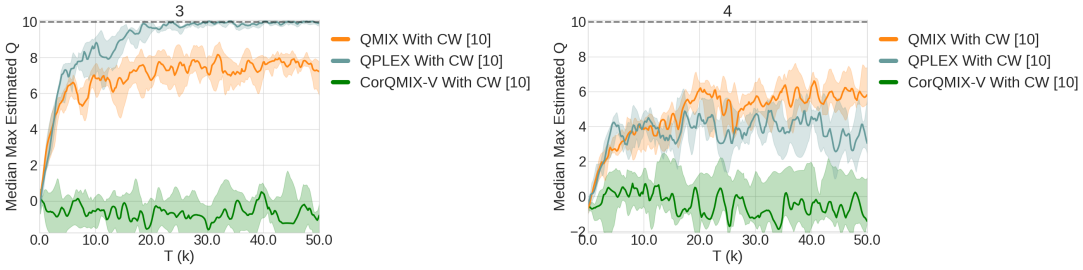


Figure 5.28: Maximum estimated Q for the methods.

benefits do not appear to scale in our specific training setup. Interestingly, the Weighted CorQMIX-V variant performs very poorly. We also do not include the Weighted CorQMIX-Q variant since it did not perform stably, and exhibited large variance between seeds.

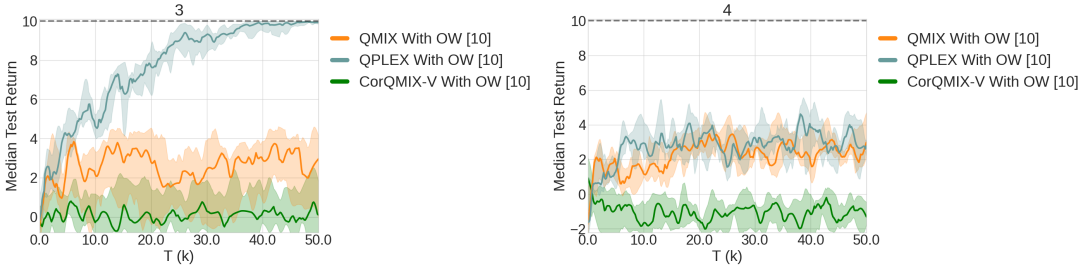


Figure 5.29: Testing performance of all algorithms.

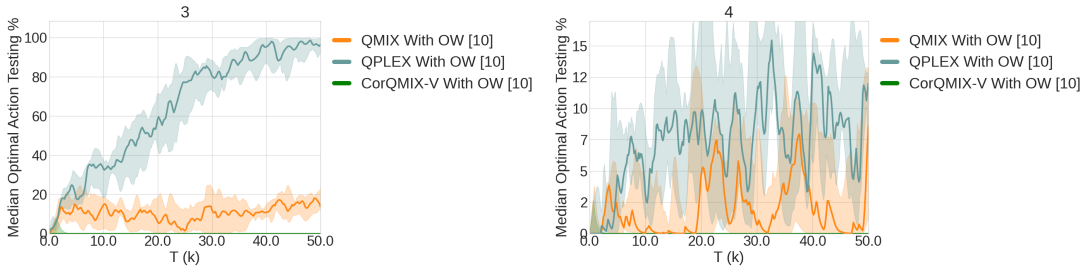


Figure 5.30: % of testing episodes the methods take the optimal action.

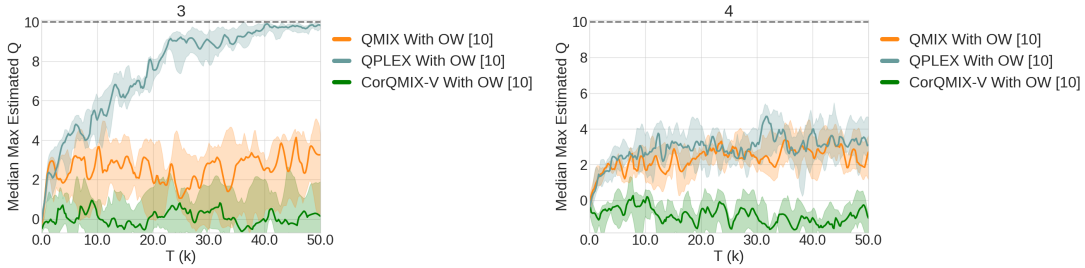


Figure 5.31: Maximum estimated Q for the methods.

Figures 5.29, 5.30, and 5.31 show the results with the OW weighting, which does not depend on the $\hat{Q}^*$ in this environment since we are not bootstrapping. As with the CW weighting, we observe the same trend in results. Weighting QPLEX's loss leads to much better performance over both the Weighted QMIX and unweighted QPLEX approaches in the smaller matrix games, and little to no benefit in the larger games. Once again, CorQMIX-V performs poorly and CorQMIX-Q is not shown due to instability.

These results tentatively suggest that the weighting might be beneficial beyond its original motivations, since it improves QPLEX's performance greatly on the smaller matrix games. However, it does not help in the larger tasks which are closer to the actual environments we are interested in improving performance on in terms of the number of agents considered. This suggests that our choice of weightings (and their hyper-parameters), can be significantly improved in a Deep RL training setup. Additionally, the use of weighting does not improve the performance of CorQMIX-V and introduces significant instability into CorQMIX-Q, which suggests that the interaction between the weighting and the parametrisation is non-trivial. To investigate a little further, we again consider the distribution

of errors, this time for the weighted agents.

First we consider the distribution of errors for the OW Weighting. Figure 5.32 shows that in this task, the OW Weighting does not change the distribution of errors for QMIX significantly compared to Figure 5.24. Similarly for CorQMIX-V in Figure 5.34, we can see that the OW Weighting is not substantially changing the distribution of errors compared to Figure 5.20. For QPLEX on the other hand, in Figure 5.33, we can see that there is a significant amount of underestimation of some joint-actions, but that the weighting significantly improves the ability to recover and represent the correct optimal Q-values. For QPLEX especially, we can see that the introduction of a weighting negatively affects many of the Q-value estimates across the joint-action space, and so it is necessary to introduce a separate unweighted $\hat{Q}^*$ for the purpose of evaluating the Q-values.

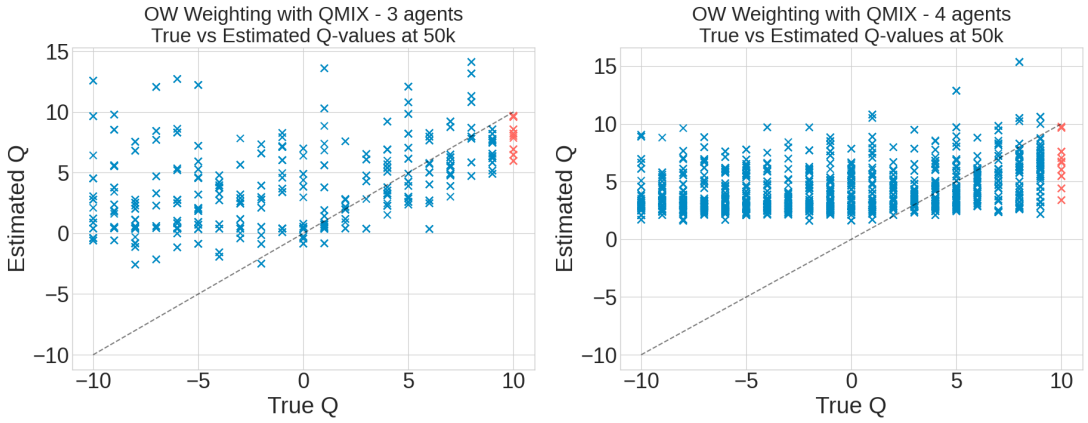


Figure 5.32: True Q-values compared to the estimated Q-values for QMIX with OW weighting at the end of training.

We next look at the distribution of errors for the CW weighting, which generally performs better than the OW weighting in this task. For QMIX, we can see a slight improvement in the estimation of the optimal action's Q-values compared to no weighting or the OW weighting. Similarly to the OW weighting, the CW weighting causes severe underestimation for some actions in QPLEX. For CorQMIX-V, we can see that there is once again no noticeable benefit to using this weighting.

We have mentioned that CorQMIX-Q was unstable in our experiments and exhibited a lot of variance between seeds. Figure 5.38 shows the distribution of

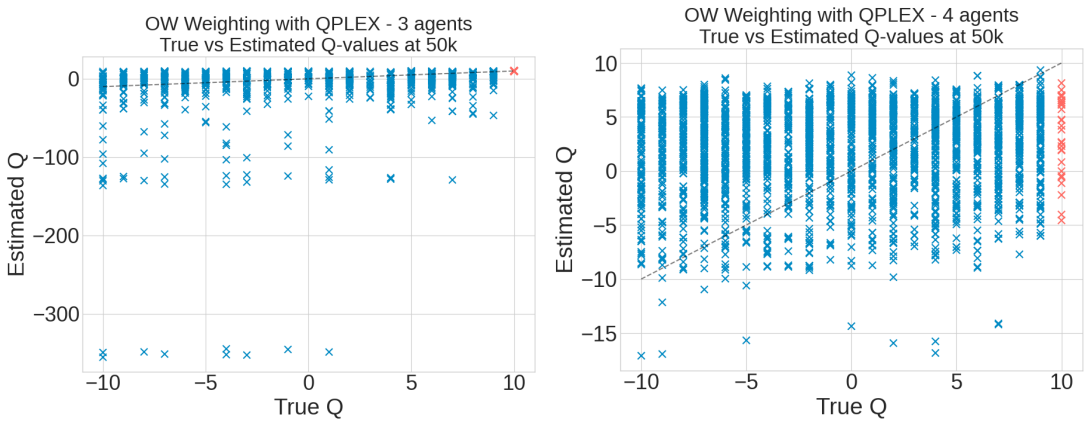


Figure 5.33: True Q-values compared to the estimated Q-values for QPLEX with OW weighting at the end of training.

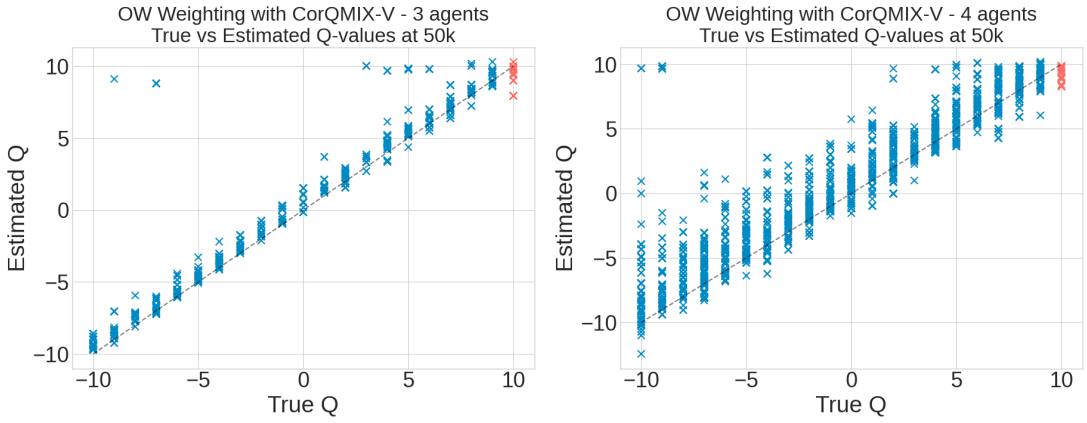


Figure 5.34: True Q-values compared to the estimated Q-values for CorQMIX-V with OW weighting at the end of training.

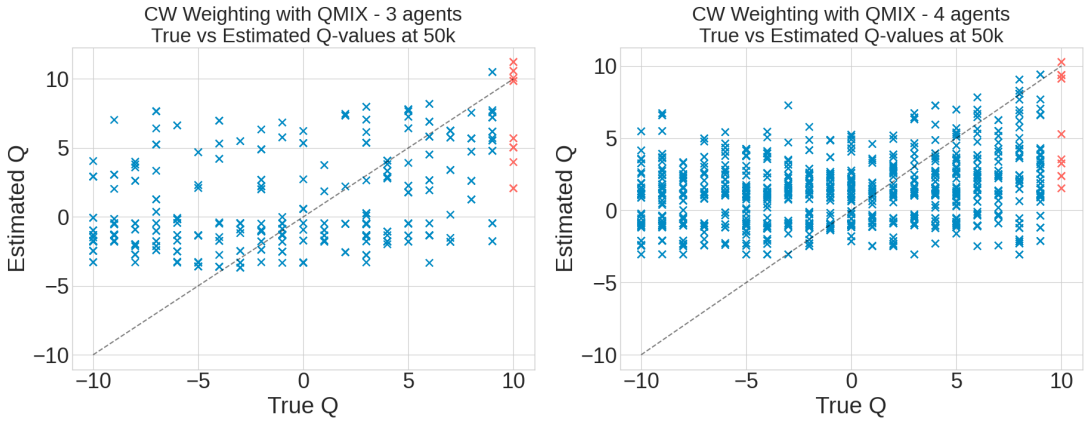


Figure 5.35: True Q-values compared to the estimated Q-values for QMIX with CW weighting at the end of training.

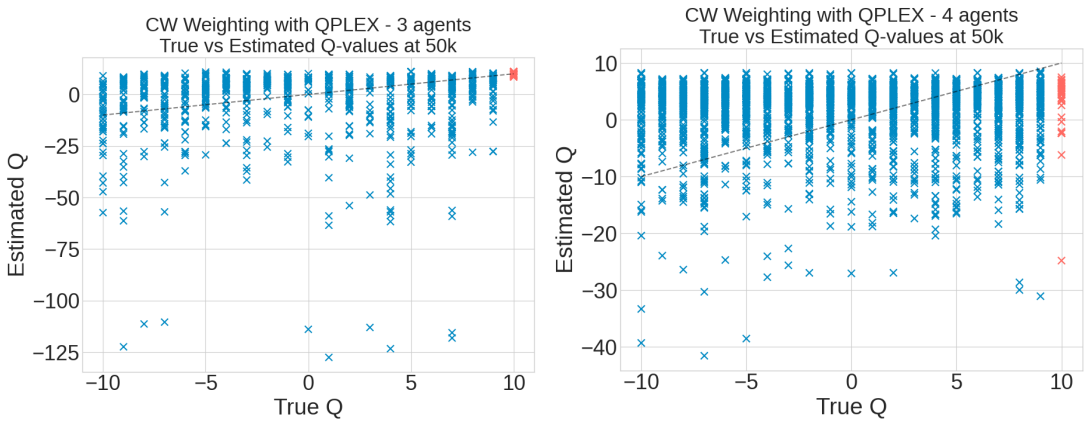


Figure 5.36: True Q-values compared to the estimated Q-values for QPLEX with CW weighting at the end of training.

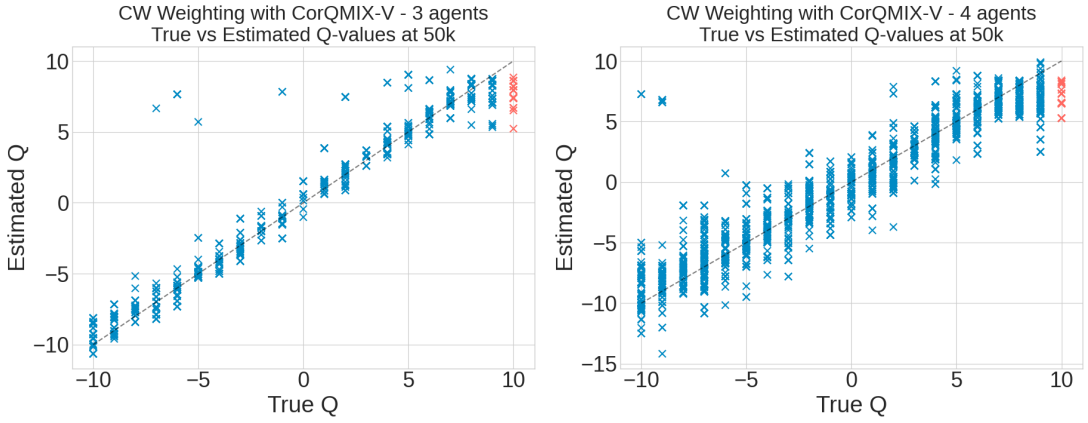


Figure 5.37: True Q-values compared to the estimated Q-values for CorQMIX-V with CW weighting at the end of training.

errors for 4 different seeds, for both the OW and CW weightings and with 3 and 4 agents. Some seeds appear similar in their errors to CorQMIX-V, but others can have extremely large magnitudes for the Q-value estimates.

5.11 Conclusions and Future Work

This chapter presented Weighted QMIX, which was inspired by analysing an idealised version of QMIX that first computes the Q-learning targets and then projects them into $\mathcal{Q}^{mix}$. QMIX uses an unweighted projection that places the same emphasis on every joint action, which can lead to suboptimal policies. Weighted QMIX rectifies this by using a weighted projection that allows more emphasis to be placed on *better*

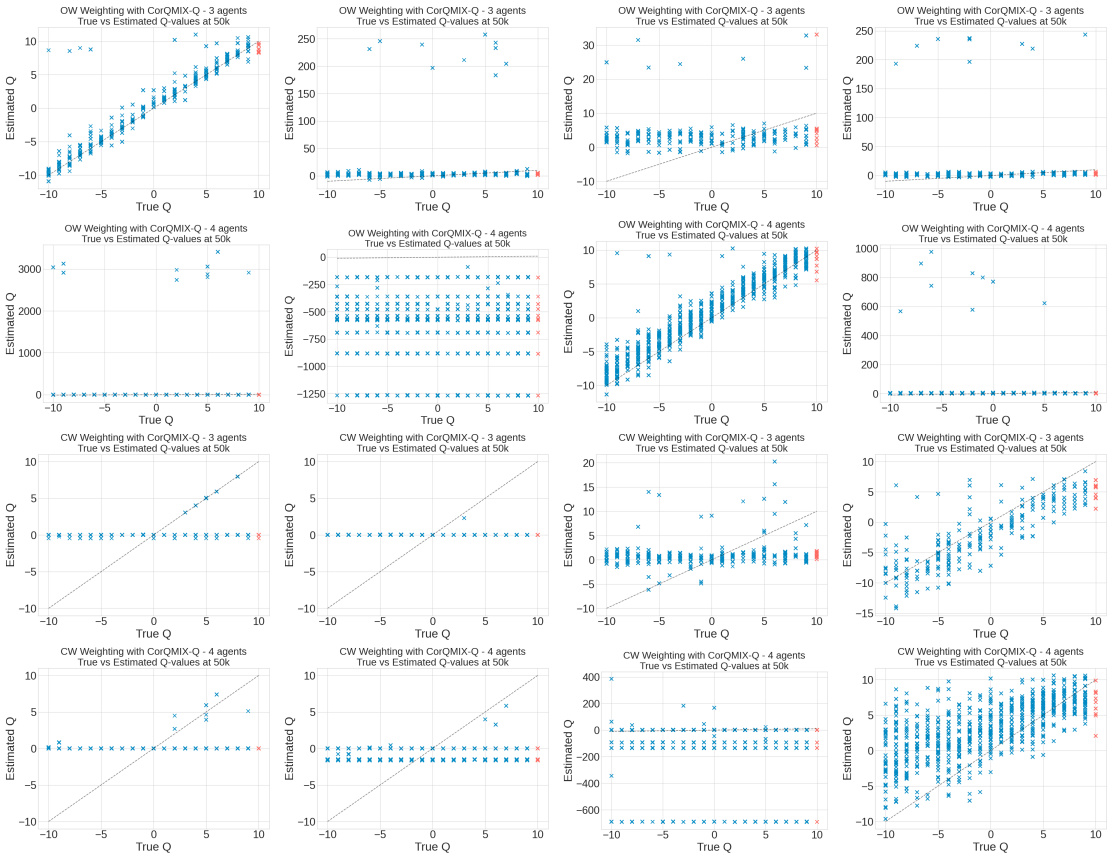


Figure 5.38: True Q-values compared to the estimated Q-values for CorQMIX-Q across 4 different seeds with OW and CW weightings at the end of training.

joint actions. We formally proved that for two specific weightings, the weighted projection is guaranteed to recover the correct maximal joint action for any Q. To fully take advantage of this, we additionally learn an unrestricted joint action $\hat{Q}^*$, and prove that it converges to Q^*. We extended Weighted QMIX to Deep RL and showed its improved ability to coordinate and its robustness to an increased rate of exploration. We then further investigated the utility of the weighting beyond its original motivations in modifying the Q-values learnt by QMIX at the global minimum of its loss. Our analysis suggested that the use of an explicit weighting improved the learning dynamics, by decoupling the loss function from the implicit weighting determined by the data-gathering processes. This could prove even more useful in an Offline RL setting in which we are not able to control the data we are training on at all. We then introduced CorQMIX, in order to investigate the affect the parametrisation of the network has on performance beyond the class of

joint-action Q-values that can be represented. Based on our results in contextual matrix games, we showed that the exact parametrisation can have a huge affect on performance. Our results also shed some light on the potential benefits of the weighting beyond QMIX. Namely that it has the potential to improve the properties of the joint-action Q-values that are being learnt throughout training, particularly in situations where the loss is not exactly minimised and we are training on relatively small minibatches as is common in Deep RL. However, our specific choices of the weighting are not sufficient to realise this potential in full. For CorQMIX-V we observed little to no benefit in using our two specific weightings, and for CorQMIX-Q we observed severe instability indicating that the interaction between the weighting and the parametrisation is non-trivial. Although the weightings improved QPLEX's ability to recover the correct argmax, we observed a potentially adverse side-effect of extremely inaccurate Q-value estimates being learnt. We hypothesise that the use of a weighting which enforces a large action-gap [231] might substantially improve the performance of these methods, in particular CorQMIX-V. Additionally, our results suggested that a weighting which is less sensitive to the exact transitions sampled in the minibatch performed better in a Deep RL training setup. Although our results in this chapter were not entirely positive, they have identified some immediate bottlenecks for these approaches. We are optimistic that future work could build on these results to substantially improve the performance of some of these algorithms through simple but well-motivated changes. In particular, we believe that CorQMIX-V could become a very strong performer with appropriate changes, based on its distribution of errors. Future work could also include the investigation of more sophisticated weightings to replace the relatively simplistic weightings we used where w is either 1 or α. Additionally, improved architectures and the training setup for $\hat{Q}^*$could be investigated in order to ensure it is not a bottleneck in more complex tasks.

Part III

Empirical Game Theory

Abstract

In the previous chapters of this thesis, we have considered environments and tasks that are fully specified. By this, we mean that the transition and reward functions are only a function of our single agent's action (or the team's joint-action). This is the natural setting for RL, and encompasses many situations such as mazes, single-player video games, matrix games, etc. However, there are many situations in which the task and environment are not so neatly setup and well-defined. The most prevalent example being a two-player game in which the opponent is not explicitly specified or known. This differs from the setup we adopted for SMAC, in which we defined the built-in AI as the fixed opponent. In many cases it is not possible or desired to fix the opponent in this manner. For example, say we wish to train an agent to play the full game of StarCraft 2 at a Grandmaster level (the goal of AlphaStar [26]). In this scenario there is no bot or computerised agent which can play at this level, nor can we reasonably expect to be able to play millions of games against real humans at this skill-level (or any level for that matter). Furthermore, even if we could do so, it might be too hard to make progress against such a strong opponent which would make learning inefficient or stall altogether. A solution to addressing these issues, is to propose our own opponents with the goal of continually learning and improving. A popular and successful instantiation of this idea is Self-Play. In the most basic of self-play setups, an agent is playing against itself, but more sophisticated extensions are available through the Policy-Space Response Oracles (PSRO) framework. Broadly speaking, PSRO maintains a population of agents over which it calculates some *solution concept* (commonly the Nash equilibrium). It then computes a *Best Response* to this solution concept, commonly though training an RL agent, and then adds this new agent to the population and repeats. The key step we will focus on in this chapter, is the calculation of the solution concept of the meta-game. The meta-game is the matrix game of expected payoffs where the population of agents are evaluated against each other (e.g. expected win rate between players in a Chess tournament). The problem is that we often do not know the expected payoffs of the meta-game, and must estimate them by sampling. Practically this means playing the game to completion between the chosen agents. It can thus be very computationally expensive to obtain accurate approximations in large domains where samples exhibit

significant noise, e.g. 2 AlphaStar agents playing a game of StarCraft results in a payoff drawn from a Bernoulli distribution at significant computational cost. Our goal in this chapter is to then construct an algorithm for smartly picking which payoffs of the meta-game to gather more samples from. Our ultimate aim being to drastically reduce the time and computation required for this key-step, or to significantly improve our approximation for a given compute budget. This can then significantly reduce the overall training time required in these setups, which is often a severe bottleneck, and thus enable larger and more complex scenarios to be tackled. Additionally, in many of these scenarios (especially in self-play settings) it would be desirable to encode domain-specific or training-specific knowledge into our algorithm. Hence, we also desire an algorithm that can allow for such prior knowledge to be encoded sensibly and flexibly.

6

Empirical Game Theory

Contents

6.1 Introduction 185
6.2 Background 186
6.2.1 Games and α-Rank 186
6.2.2 Single Population α-Rank 187
6.2.3 Worked Example 188
6.2.4 Wasserstein Distance 189
6.3 Related Work 190
6.3.1 ResponseGraphUCB 191
6.4 αIG and αWass 192
6.4.1 Payoffs: Ground Truth and Belief 193
6.4.2 Belief over α-ranks 194
6.4.3 Picking Payoffs to Query 194
6.4.4 Implementation 195
6.4.5 Computational Requirements 196
6.4.6 Query Selection by Maximizing Wasserstein Divergence 197
6.5 Theoretical Results 198
6.6 Proofs 200
6.6.1 Regret Bound For Policy Maximizing Information Gain on Payoffs 202
6.6.2 Regret Bound For Policy Maximizing Information Gain on α-Ranks 203
6.7 Experiments 207
6.7.1 Good-Bad Games 208
6.7.2 Incorporating Prior Knowledge 215
6.7.3 Gaussian Games 216
6.7.4 Conditioning of the belief distribution 217
6.8 Conclusions 218

6.1 Introduction

Traditionally, game theory is applied in situations where the game is fully known. More recently, empirical game theory addresses the setting where this is not the case, instead, the game is initially unknown and has to be interacted with by sampling [245]. One area in which this is becoming increasingly common is the ranking of trained agents relative to one another. Specifically, in the field of Reinforcement Learning game-theoretic rankings are used not just as a metric for measuring algorithmic progress [246], but also as an integral component of many population-based training methods [247, 32, 26]. In particular, for ranking, two popular solution concepts have recently emerged: Nash averaging [246, 248] and α-rank [33].

In this chapter, we aim to estimate the α-rank of a game using as few samples as possible. We use the α-rank solution concept for two reasons. First, it admits a unique solution whose computation easily scales to K-player games. Second, unlike older schemes such as Elo [249], α-rank is designed with intransitive interactions in mind. Because measuring payoffs can be very expensive, it is important to do it by using as few samples as possible. For example, playing a match of chess [250] can take roughly 40 minutes (assuming a typical game-length of 40 and up to 1 minute per move as used during evaluation), and playing a full game of Dota 2 can take up to 2 hours [25]. *Our objective is thus to accurately estimate the α-rank using a small number of payoff queries.*

Rowland et al. [251] proposed ResponseGraphUCB (RG-UCB) for this purpose, inspired by pure exploration bandit literature. RG-UCB maintains confidence intervals over payoffs. When they don't overlap, it draws a conclusion about their ordering, until all comparisons relevant for the computation of α-rank have been made. While this is provably sufficient to determine the true α-rank with a high probability in the infinite-α regime, their approach has two important limitations. First, since the frequentist criterion is indirect, relying on payoff

ordering rather than the α-rank, the obtained payoffs aren't always used optimally. Second, it is nontrivial to include useful domain knowledge about the entries or structure of the payoff matrix.

To remedy these problems, we propose a Bayesian approach. Specifically, we utilize a Gaussian Process to maintain an epistemic belief over the entries of the payoff matrix, providing a powerful framework in which to supply domain knowledge. This payoff distribution induces an epistemic belief over α-ranks. We determine which payoff to sample by maximizing information gain between the α-rank belief and the obtained payoff. This allows us to focus our sampling on the entries that are expected to have the largest effect on our belief over possible α-ranks.

Contributions: Theoretically, we justify the use of information gain by showing a regret bound for a version of our criterion in the infinite-α regime. Empirically, our contribution is threefold. First, we compare to RG-UCB on stylized games, showing that maximizing information gain provides competitive performance by focusing on sampling the more relevant payoffs. Second, we evaluate another objective based on minimizing the Wasserstein divergence, which offers competitive performance while being computationally much cheaper. Finally, we demonstrate the benefit of building in prior assumptions.

6.2 Background

6.2.1 Games and α-Rank

A game with K players, where the kth player can play S_k strategies is characterized by its expected payoffs $M \in \mathcal{R}^{K \times S_1 \times ... \times S_K}$ [252] for each player. Letting $\mathcal{S} = S_1 \times ... \times S_K$ be the space of pure strategy profiles, the game also specifies a distribution over the payoffs associated with each player when $s \in \mathcal{S}$ is played. The α-rank of a game is computed by first defining an irreducible Markov Chain whose nodes are pure strategy profiles in $\mathcal{S}$. We denote the stochastic matrix defining this chain as C. The transition probabilities of the chain C are calculated as follows: Let $\sigma, \tau \in \mathcal{S}$ be such that τ only differs from σ in a single player's strategy and let $\eta = (\sum_{k=1}^{K}(|S_k| - 1))^{-1}$ be the reciprocal of the total number of those distinct τ.

Let $M_k(\sigma)$ denote the expected payoff for player k when σ is played. Then, the probability of transitioning from σ to τ which varies only in player k's strategy is

$$C_{\sigma,\tau} = \begin{cases} \eta \frac{1-\exp(-\alpha(M_k(\tau)-M_k(\sigma)))}{1-\exp(-\alpha m(M_k(\tau)-M_k(\sigma)))} & \text{if } M_k(\tau)\neq M_k(\sigma), \\ \frac{\eta}{m} & \text{otherwise.} \end{cases}$$

$C_{\sigma,\upsilon} = 0$ for all υ that differ from σ in more than a single player's strategy, $C_{\sigma,\sigma} = 1 - \sum_{\tau\neq\sigma} C_{\sigma,\tau}$ to ensure a valid transition distribution, and $\alpha \geq 0, m \in \mathbb{N}^{>0}$ are parameters of the algorithm. We define the α-rank $r \in \mathcal{R}^{|\mathcal{S}|}$ as the unique stationary distribution of the chain C [33, 251] as $\alpha \to \infty$. In practice, a large finite value of α is used, or a perturbed version of the transition matrix C is used with an infinite α to ensure the resulting Markov Chain C is irreducible.

6.2.2 Single Population α-Rank

In this chapter we focus on the infinite-α regime and restrict our attention to the 2-player single population case of α-rank which differs slightly from above. Importantly, our method can be easily applied to multiple populations as described above in a straightforward way, but we focus on the single population case for simplicity. Let $S = S_1$ and $M(\sigma, \tau)$ denote the payoff when the first player plays σ and the second player plays τ. Note that $S_1 = S_2$ since the single population case considers a player playing a game against an identical player. In this particular setting, the α-rank $r \in \mathcal{R}^{|S|}$ and the perturbed transition matrix $C \in \mathcal{R}^{S\times S}$ is calculated as follows:

$$C_{\sigma,\tau} = \begin{cases} (|S|-1)^{-1}(1-\epsilon) & \text{if } M(\tau,\sigma) > M(\sigma,\tau), \\ (|S|-1)^{-1}\epsilon & \text{if } M(\tau,\sigma) < M(\sigma,\tau), \\ 0.5(|S|-1)^{-1} & \text{if } M(\tau,\sigma) = M(\sigma,\tau), \end{cases}$$

for $\sigma \neq \tau$. $C_{\sigma,\sigma} = 1 - \sum_{\tau\neq\sigma} C_{\sigma,\tau}$ again to ensure a valid transition distribution and ϵ is a small perturbation to ensure irreducibility of the resulting chain. We abstract the above computation into the α-rank function $f : \mathcal{M} \to \mathcal{R}^{|S|}$, where $\mathcal{M}$ is the space of 2-player payoff matrices with S strategies for each player.

6.2.3 Worked Example

In order to make our setting clearer, we now include a worked example on how to calculate the α-Rank in the single-population setting in the infinite-α regime. The purpose of this example is to help build intuition about α-Rank in our particular setting. We will use 2 Good, 2 Bad matrix game, shown in Figure 6.1 as an example, where the payoffs represent the expected win rate between agents.

	G1	G2	B1	B2
G1	0.5	0.45	1	1
G2	0.55	0.5	1	1
B1	0	0	0.5	0.5
B2	0	0	0.5	0.5

Figure 6.1: Payoff matrices for 2 Good, 2 Bad with the strategies for each player labeled.

Note that in this example, we are considering a player playing the game against an identical opponent. The strategy space is $S = \{G1, G2, B1, B2\}$, with $G1$ and $G2$ representing the *good* agents, and $B1$ and $B2$ the *bad* agents. The good agents **always** win against the bad agents (hence $M(G1, B1) = 1$ and $M(B1, G1) = 0$ to use $G1$ and $B1$ as examples). The α-Rank is then a probability vector $r \in \Delta^{4-1} \subset \mathcal{R}^4$.

In order to compute the α-Rank, we must first construct the Markov Chain whose nodes are elements of S.

Figure 6.2 is a diagram representing this Markov Chain.

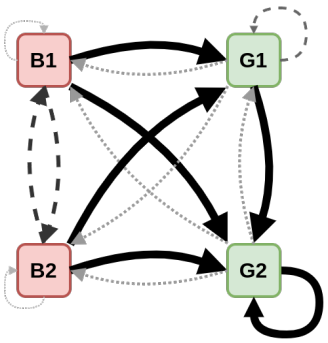


Figure 6.2: Markov Chain constructed to compute the α-Rank. The transition probabilities between the nodes are represented using the width of the arrow.

The 4 nodes are the the elements of $S = \{G1, G2, B1, B2\}$, and the transition probabilities are as defined in Section 6.2. The width of the edges between the nodes in Figure 6.2 represent the magnitude of their transition probabilities. Since the *good* agents consistently beat the *bad* agents there is a large transition probability from nodes $B1$ and $B2$ to nodes $G1$ and $G2$. Likewise there is a very small transition probability from nodes $G1, G2$ to $B1, B2$. Since strategies $B1$ and $B2$ are equally matched, there is an equal probability of transition from $B1$ to $B2$ as there is from transitioning from $B2$ to $B1$. Crucially though, that transition probability is significantly smaller than the transition probabilities from $B1$ and $B2$ to nodes $G1$ and $G2$. All nodes' self-transition probabilities ensure that there is a valid transition matrix for the Markov Chain. Importantly, $B1$ and $B2$ have small self-transition probabilities whereas $G2$ has a very large self-transition probability.

We know $(|S| - 1)^{-1} = \frac{1}{3}$, and letting $\epsilon > 0$ be small, we can compute the exact transition matrix:

$$\begin{array}{c} \\ G1 \\ G2 \\ B1 \\ B2 \end{array} \begin{array}{c} \begin{array}{cccc} G1 & G2 & B1 & B2 \end{array} \\ \begin{pmatrix} (2-\epsilon)/3 & (1-\epsilon)/3 & \epsilon/3 & \epsilon/3 \\ (1-\epsilon)/3 & 1-\epsilon & \epsilon/3 & \epsilon/3 \\ (1-\epsilon)/3 & (1-\epsilon)/3 & (1+4\epsilon)/6 & 1/6 \\ (1-\epsilon)/3 & (1-\epsilon)/3 & 1/6 & (1+4\epsilon)/6 \end{pmatrix} \end{array}$$

Intuitively, we can see that the transition probabilities are all *leading* to node $G2$. So we would expect to spend a large proportion of time at node $G2$, if we were *traversing* the graph according to the transition probabilities. Hence, $G2$ is a *stronger* strategy than the others and so we would expect the α-Rank to reflect that.

This is formalised in the computation of α-Rank by considering the unique stationary distribution of the Markov Chain. The α-Rank is then $(0, 1, 0, 0)$ $(G1 = 0, G2 = 1, B1 = 0, B2 = 0)$ as $\epsilon \to 0$.

6.2.4 Wasserstein Distance

Let p and q be probability distributions supported on $\mathcal{X}$, and $c : \mathcal{X} \times \mathcal{X} \to [0, \infty)$ be a distance. Define Π as the space of all joint probability distributions with

marginals p and q. Wasserstein distance [253] with cost function c, is defined as:

$$\mathcal{W}_c(p, q) := \min_{\pi \in \Pi} \int_{\mathcal{X} \times \mathcal{X}} c(x, y) d\pi(x, y).$$

In this chapter, we will utilize the Wasserstein distance between our belief distributions over α-rank, and so we set $\mathcal{X} = \Delta^{S-1}$, the $(S-1)$ probability simplex, and use $c(x, y) = \frac{1}{2}\|x - y\|_1$, i.e. the total variation distance. We will drop the suffix and denote this simply as $\mathcal{W}$.

6.3 Related Work

There are many methods related to the ranking and evaluation of agents in games. Elo [249] and TrueSkill [254, 255] both quantify the performance of an agent using a single number, which means they are unable to model *intransitive* interactions. Chen and Joachims [256] extend TrueSkill to better model such interactions, while Balduzzi et al. [246] do the same for Elo, improving its predictive power by introducing additional parameters. Balduzzi et al. [246] also re-examines the use of Nash equilibrium, proposing to disambiguate across possible equilibria by picking the one with maximum entropy. However, it is well known that computing the Nash equilibrium is computationally difficult [9] and only computationally tractable for restricted classes of games. In this chapter, we focus on α-rank [33] since it has been designed with *intransitive* interactions in mind, it is computationally tractable for N-player games and shows considerable promise as a component of self-play frameworks [247].

Empirical Game Theory [245] is concerned with situations in which a game can only be interacted with through sampling. The most related work to ours investigates sampling strategies and concentration inequalities for the Nash equilibrium as opposed to the α-rank. Walsh et al. [257] introduce Heuristic Payoff Tables (HPTs) in order to choose the samples that provide the most information about the currently chosen Nash equilibrium, where information is quantified as the reduction in estimated error. This differs from our approach both in the use of α-rank as opposed to the Nash equilibrium as our solution concept, and in the criterion used

to select the observed payoff. Tuyls et al. [258] provide concentration bounds for estimated Nash equilibria. Jordan et al. [259] find Nash equilibria from limited data by using information gain on distributions over strategies, a concept different from our information gain on distributions over ranks. We also utilize α-rank as the solution concept, rather than Nash equilibria.

Muller et al. [247] utilise α-rank as part of a PSRO [32] framework. They do not use an adaptive sampling strategy for deciding which entries to sample, but are a natural application for applying our algorithm (and RG-UCB). Yang et al. [260] introduce an approximate gradient-based algorithm which does not require access to the entire payoff matrix at once in order to compute α-rank. Although their method does not require the entire payoff matrix at every iteration, it is not designed for operating in the same incomplete information setting that we explore in this chapter since they assume every entry can be cheaply queried with no noise. Srinivas et al. [261] prove regret bounds for Bayesian optimization with GPs. We use their concentration result to derive our bounds as well as as inspiration for our information gain criterion.

6.3.1 ResponseGraphUCB

Closest to our work is ResponseGraphUCB (RG-UCB) introduced by Rowland et al. [251], which can be viewed as a frequentist analogue to our method which also operates in the infinite-α regime. RG-UCB first specifies an error threshold $\delta > 0$ and then samples payoffs until a stopping criteria determines the estimated α-rank is correct with probability at least $1 - \delta$. A key observation that RG-UCB relies on, is that in the infinite-α regime only the ordering between relevant payoffs is important. e.g. For pure strategy profiles σ and τ (with payoffs $M_k(\sigma)$ and $M_k(\tau)$ respectively) that are used in the computation of the Markov Chain transition probabilities, determining whether $M_k(\sigma) > M_k(\tau)$ or $M_k(\sigma) < M_k(\tau)$ is enough to know the transition probability accurately (their magnitude difference $|M_k(\sigma) - M_k(\tau)|$ is unimportant). RG-UCB maintains $(1-\delta)$ confidence intervals for all values of $M_k(\sigma)$, and determines the ordering between σ and τ is correct when they do not overlap.

A strategy profile is chosen to be sampled until all of its ordering are correctly determined. When all orderings are correctly determined the algorithm terminates.

Since the confidence intervals are constructed using frequentist concentration inequalities, we refer to RG-UCB as being a frequentist algorithm. In contrast, our Bayesian perspective provides a principled method for incorporating prior knowledge into our algorithm whereas it is much more difficult to encode modelling assumptions and prior knowledge with RG-UCB. The second important difference between our work and RG-UCB is that our information gain criterion is a direct objective, which selects the payoffs to sample based on how likely the received sample is to affect the α-rank. On the other hand, RG-UCB works indirectly, reducing uncertainty about the orderings between individual payoffs without considering their impact on the final α-rank, which makes it less efficient. Rowland et al. [251] also theoretically justify the use of RG-UCB in the infinite-α regime by proving sample complexity results, whereas we provide asymptotic regret bounds for our approach which are commonly used to justify the sample efficiency of a Bayesian algorithm [261]. Rowland et al. [251] additionally provide a method for obtaining uncertainty estimates in the infinite-α regime, which is, however, not used as part of an adaptive sampling strategy.

Recently, Du et al. [262] extend RG-UCB to include a low-rank matrix completion component. It is based on the insights from [263], in which the meta-game payoff matrix of *real-world* games are shown to have a low rank. To take advantage of this, Du et al. [262] propose to only apply RG-UCB on a limited subset of the possible entries, and then to utilise a low-rank matrix completion algorithm to then reconstruct the entire meta-game for use in calculating α-Rank.

6.4 αIG and αWass

On a high level, our method works by maintaining an epistemic belief over α-ranks and selecting payoffs that lead to the maximum reduction in the entropy of that belief. Figure 6.3 provides a pictorial overview. In the middle of the figure, we maintain an explicit distribution over the entries of the payoff matrix. This payoff

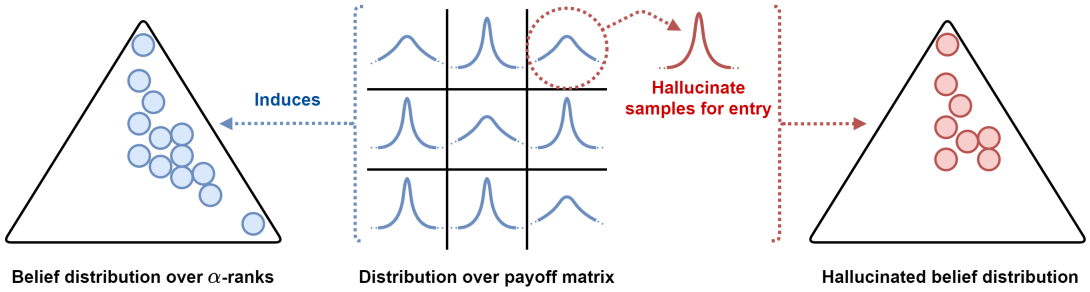


Figure 6.3: Overview: On the left, a belief over α-ranks is induced by a belief over the payoff matrix shown in the middle. A hallucinated belief distribution is shown on the right.

distribution induces a belief over α-ranks, shown on the left. When deciding which payoff to sample, we examine hypothetical belief states after sampling, striving to end up with a belief with the lowest entropy. One such hypothetical, or 'hallucinated' belief is shown on the right. We now describe our method formally, first describing the probabilistic model and then the implementation.

6.4.1 Payoffs: Ground Truth and Belief

We denote the unknown true payoff matrix as $M_\star$, a function mapping from a pure strategy profile $s \in \mathcal{S}$ to the expected payoffs for each agent. To quantify our uncertainty about what this true payoff is, we employ a Gaussian Process whose mean we will denote as M. This will allow us to encode prior knowledge about payoff dependencies that a practitioner might have. For example, the payoffs for Agent's 1 and 2 against Agent 7 are very close, or the payoffs for Agent 1 and Agent 2 add up to 1 because they represent the expected win rate. Our framework is sufficiently general to allow for other approaches such as Bayesian Matrix Factorization [264] or probabilistic methods for Collaborative Filtering [265] to be used as an alternative to the Gaussian Processes. We choose to use Gaussian Processes due to their flexibility in encoding prior knowledge and modelling assumptions, and their ubiquity throughout literature.

The GP models noise in the payoffs as $\tilde{M} = M + \epsilon$, where $\epsilon \sim \mathcal{N}(0, I\sigma_A^2)$. When interacting with the game sequentially, the received payoffs are assumed

to be generated as $m_t = M_\star(a_t) + \epsilon'_t$. Here, ϵ'_t are i.i.d. random variables with support on the interval $[-\sigma_A, \sigma_A]$. While it may at first seem surprising that we use Gaussian observation noise in the GP model ($\epsilon \sim \mathcal{N}(0, I\sigma_A^2)$), while assuming a truncated observation noise for the actual observation ($\epsilon'_t \in [-\sigma_A, \sigma_A]$), this does not in fact affect our theoretical guarantees. We provide more details in Section 6.5. We denote the history of interactions at time t by H_t. Because of randomness in the observations, H_t is a random variable. The sequence of random variables $H_1, H_2, \ldots$ forms a filtration. We use the symbol h_t to denote particular realization of history so that $h_t = a_1, m_1, \ldots, a_{t-1}, m_{t-1}$.

6.4.2 Belief over α-ranks

Our explicit distribution over the entries of the payoff matrix M induces an implicit belief distribution over the α-ranks. For all valid α-ranks r, $P(r) = P(M \in f^{-1}(r))$ where f^{-1} denotes the pre-image of r under f. In other words, the probability assigned to an α-rank r is the probability assigned to its pre-image by our belief over the payoffs. Since r is represented implicitly (we only maintain an explicit distribution via the GP over the expected payoffs), we cannot query its mass function directly since it would be intractable to compute. Instead, we access r via sampling. This is done by first drawing a payoff from $m \sim M$ and then computing the resulting α-rank $f(m)$.

6.4.3 Picking Payoffs to Query

At time t, we query the payoff that provides us with the largest information gain about the α-rank. Formally,

$$
\begin{aligned}
a_t &= \arg\max_a \mathbb{I}(r\,;\,(\tilde{M}_t(a), a) \mid H_t = h_t) \\
&= \arg\max_a \mathbb{H}\left(r \mid H_t = h_t\right) \\
&\quad - \mathop{\mathbb{E}}_{\tilde{m}_t \sim \tilde{M}_t(a)} \left[\mathbb{H}\left(r \mid H_t = h_t, A_t = a, \tilde{M}_t(a) = \tilde{m}_t\right)\right] \qquad (6.1) \\
&= \arg\min_a \mathop{\mathbb{E}}_{\tilde{m}_t \sim \tilde{M}_t(a)} \left[\mathbb{H}\left(r \mid H_t = h_t, A_t = a, \tilde{M}_t(a) = \tilde{m}_t\right)\right]. \qquad (6.2)
\end{aligned}
$$

In Equation (6.1), $\mathbb{H}(r \mid H_t = h_t)$ is the entropy of our current belief distribution over α-ranks, which does not depend on a and can be dropped from the maximization, producing Equation (6.2). The expectation in (6.2) has an intuitive interpretation as the expected negative entropy of our *hallucinated* belief, i.e. belief obtained by conditioning on a sample $\tilde{m}_t$ from the current model. In essence, we are pretending to receive a sample for entry a, and then computing what our resulting belief over α-ranks will be. By picking the entry as in (6.2), we are picking the entry whose sample will lead to the largest reduction in the entropy of our belief over α-ranks in expectation.

6.4.4 Implementation

Algorithm 3 αIG algorithm. αIG(NSB) and αIG(Bin) variants differ in entropy estimator (Line 7).

1: **for** $t = 1, 2, \ldots T$ **do**
2: **for** $a = 1, 2, \ldots |\mathcal{S}|$ **do**
3: **for** $i = 1, 2, \ldots N_e$ **do**
4: $\tilde{m}_t \sim \tilde{M}_t(a)$ 'Hallucinate' a payoff.
5: Obtain hallucinated posterior payoff: $P(\hat{M}_t | H_t = h_t, A_t = a, \tilde{M}_t(a) = \tilde{m}_t)$
6: $D = \{r_1, \ldots, r_{N_b}\}$, where $r_i \sim f(\hat{M}_t)$ i.i.d.
7: $\hat{h}_a^i =$ ESTIMATE-ENTROPY(D)
8: **end for**
9: $\hat{h}_a = \frac{1}{N_e} \sum_{i=1}^{N_e} \hat{h}_a^i$
10: **end for**
11: Query payoff $a_t = \arg\min_a \hat{h}_a$ Implements Eq. (6.2).
12: **end for**

Our algorithm, which we refer to as αIG, is summarized in Algorithm 3. At a high-level, αIG selects an action/payoff to query at each timestep (Line 1). In order to select a payoff to query as in Equation (6.2), we must approximate the expectation for each payoff (Line 2). In Line 4, we use our epistemic model to obtain a 'hallucinated' outcome $\tilde{m}_t$, as if we received a sample from selecting payoff a at timestep t. In Line 5, we condition our epistemic model on this 'hallucinated' sample $\tilde{m}_t$ in order to obtain our 'hallucinated' posterior over payoffs $\hat{M}_t$. In Line 7, we empirically estimate the entropy of the resulting induced belief distribution over

α-ranks. To approximate the expectation in (6.2), we average out entropy estimates obtained from N_e different possible hallucinated payoffs in Line 9. Since our approximation of this expectation is a crucial part of the query selection procedure, one could utilise better methods of approximation such as Bayesian Quadrature [266, 267] which we do not explore here. Finally, in Line 11, we use these estimates to perform query selection as in (6.2) to select a payoff to query at timestep t.

Our algorithm depends on an entropy estimator ESTIMATE-ENTROPY, used in Line 7. We present results for 2 different entropy estimators: simple binning and NSB. The simple binning estimator estimates the entropy using a histogram. For comparison, we also used NSB [268], an entropy estimator designed to produce better estimates in the small-data regime.

6.4.5 Computational Requirements

The main computational bottleneck of our algorithm is the calculations of α-rank in Line 6 of Algorithm 3. In order to perform query selection as in (6.2), we must compute the α-rank $|\mathcal{S}| \times N_e \times N_b$ times. For our experiments on the 4x4 Gaussian game this results in $16 \times 10 \times 500 = 80,000$ computations of α-rank (setting $N_e = 10, N_b = 500$), to select a payoff to query. Relative to ResponseGraphUCB, our method thus requires significantly more computation in order to select a payoff to query. However, in Empirical Game Theory, it is commonly assumed that obtaining samples from the game is very computationally expensive (which is true in many potential practical applications [25, 250, 26]). The increased computation required by our method to select a payoff to sample should then have a negligible impact to the overall computation time required, but the increased sample efficiency could potentially lead to large speed-ups.

We perform two simple optimizations when deploying the algorithm in practice. To save computational cost, we observe the same payoff N_r times in Line 11 rather than once, similar to rarely-switching bandits [269]. Moreover, the number of samples N_b we can use to estimate the entropy is limited due to the computational cost of computing α-rank. In order to obtain better differentiation between the entropy of

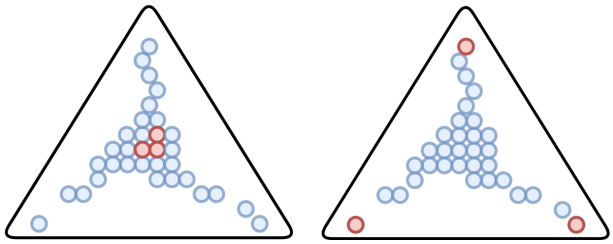


Figure 6.4: Diagram depicting the current belief (Blue) and 2 different hallucinated beliefs (Red). The current belief is the distribution over alpha-ranks, $P(r)$, according to our current posterior over the true expected payoff entries. The 2 hallucinated beliefs, $P(r_1)$ and $P(r_2)$, are the distributions over alpha-ranks according to the two posteriors obtained after conditioning on hallucinated payoffs from the GP. We are assuming a discrete distribution over α-ranks (which is the case since there are a finite number of alpha-ranks, albeit large in number), where the belief is concentrated at the center of each circle. We have made the circles large to make the visualisation simpler.

beliefs arising from sampling different payoffs, we heuristically perform conditioning in Line 5 N_c times. Section 6.7.4 contains a more detailed discussion on this.

6.4.6 Query Selection by Maximizing Wasserstein Divergence

While the query objective proposed in (6.2) is backed both by an appealing intuition and a theoretical argument (see Section 6.5), it can be expensive to evaluate due to the cost of accurate entropy estimation. To address this difficulty, we also investigate an alternative involving the Wasserstein distance. The objective we consider is

$$\arg\max_a \mathbb{E}_{\tilde{m}_t \sim \tilde{M}_t}[\mathcal{W}(P(r|H_t = h_t), P(r|H_t = h_t, A_t = a, \tilde{M}_t(a) = \tilde{m}_t))]. \tag{6.3}$$

Since the computation of Wasserstein distance from empirical distributions can be achieved by solving a linear program [270], Equation (6.3) naturally lends itself to being approximated via samples. In our implementation, we use POT [271] to approximate this distance.

The Wasserstein distance is built on the notion of cost, which allows a practitioner the opportunity to supply additional prior knowledge. In our case, since α-ranks are probability distributions, a natural way to measure accuracy is to use the total variation distance, which corresponds to setting the cost to $c(x, y) = \frac{1}{2}\|x - y\|_1$. On the other hand, in cases where we are interested in finding the relative ordering of

agents under the α-rank, an alternative cost such as the Kendall Tau metric [272] could be used. While we emphasize the ability of the Wasserstein divergence to work with any cost, we leave the empirical study of non-standard costs for future work.

It is important to note that the objective in (6.3) is qualitatively different to the information gain objective proposed in (6.2). Figure 6.4 provides a diagram illustrating a major difference between the two objectives. The entropy for both belief distributions shown in red is the same. In contrast, the Wasserstein distance in (6.3) between the current belief in blue and the hallucinated belief in red is much smaller for the distribution on the left compared to the distribution on the right.

6.5 Theoretical Results

Notions of Regret

We quantify the performance of our method by measuring regret. Our main analysis relies on Bayesian regret [118], defined as

$$J_t^B = 1 - \mathbb{E}_{h_t}\left[P(r = r_\star | H_t = h_t)\right], \tag{6.4}$$

where we used $r_\star$ to denote the α-rank with the highest probability under r at time t. In (6.4), the expectation is over realizations of the observation model. Since J_t^B, like all purely Bayesian notions, does not involve the ground truth payoff, we need to justify its practical relevance. We do this by benchmarking it against two notions of frequentist regret. The first measures how accurate the probability we assign to the ground truth $r_{\text{GT}} = f(M_\star)$ is

$$J_t^F = 1 - \mathbb{E}_{h_t}\left[P(r = r_{\text{GT}} | H_t = h_t)\right]. \tag{6.5}$$

The second measures if the mean of our payoff belief, which we denote M_μ, evaluates to the correct α-rank

$$J_t^M = 1 - \mathbb{E}_{h_t}\left[\delta\Big[f(M_\mu) = r_{\text{GT}}\Big]\right], \tag{6.6}$$

where the symbol $\delta\Big[\text{predicate}\Big]$ evaluates to 1 or 0 depending on whether the predicate is true or false. In Section 6.7, we empirically conclude that these three notions of regret are closely coupled in practice, changing at a comparable rate.

Regret Bounds

As an intermediate step before discussing information gain on the α-ranks, we first analyze the behavior of a query selection rule which maximizes information gain over the payoffs.

$$\pi_{\text{IGM}}(a|H_t = h_t) = \arg\max_a \mathbb{I}(\tilde{M}_t\,;\,(\tilde{M}_t(a), a) \mid H_t = h_t). \tag{6.7}$$

The following result shows that using sampling strategy π_{IGM} for T timesteps leads to a decay in regret of at least $Te^{\mathcal{O}(-\sqrt[3]{\Delta^2 T})}$, proving it will incur no regret as $T \to \infty$.

Proposition 4 (Regret Bound For Information Gain on Payoffs)**.** *If we select actions using strategy π_{IGM}, the regret at timestep T is bounded as*

$$J_T^B \leq J_T^F = 1 - \mathbb{E}_{h_T}\left[P(r = r_{GT}|H_T = h_T)\right] \leq Te^{g(T)} \tag{6.8}$$
$$\textit{where } \; g(T) = \mathcal{O}(-\sqrt[3]{\Delta^2 T}).$$

The proof, and an explicit form of g are found in Section 6.6. We now proceed to our second result, where we maximize information gain on the α-ranks directly. Consider a querying strategy that is an extension of (6.1) to T-step look-ahead, defined as

$$\pi_{\text{IGR}} = \arg\max_{a_1,\ldots,a_T} \mathbb{I}(r\,;\,(\tilde{M}_1(a_1), a_1), \ldots, (\tilde{M}_T(a_T), a_T)). \tag{6.9}$$

We quantify regret achieved by π_{IGR} in the proposition below.

Proposition 5 (Regret Bound For Information Gain on Belief over α-Ranks)**.** *If we select actions using strategy π_{IGR}, regret is bounded as*

$$J_T^B = 1 - P(r = r_\star|H_T = h_T) \to 0 \quad as \quad T \to \infty.$$

Proposition 5 provides a theoretical justification for querying the strategies that maximize information gain on the α-ranks. A more explicit regret bound (similar to Proposition 4) and the proof are provided in Section 6.6. In practice, to avoid

the combinatorial expense of selecting action sequences using π_{IGR}, we use the greedy query selection strategy in equation (6.1). While the regret result above does not carry over, this idealized setting at least provides some justification for information gain as a query selection criterion.

6.6 Proofs

Notation

As a reminder, we reintroduce notation that is relevant to this section.

$M_\star$ is the true payoff vector, which is unknown to us.

M is our prior distribution over the entries of the payoff vector, represented by a GP.

The GP models noise in the observations of the payoff as $\tilde{M} = M + \epsilon$, where $\epsilon \sim \mathcal{N}(0, I\sigma_A^2)$. We model the payoffs we receive from the real game at timestep t when taking action a_t, as $m_t \sim M_\star(a_t) + \epsilon'_t$. Where, ϵ'_t is i.i.d. and has support on the interval $[-\sigma_A, \sigma_A]$.

Clipped Noise

Note that it is important the observation noise ϵ'_t is clipped, since it allows us to apply Lemma 4 which is an existing result from Srinivas et al. [261, Theorem 6]. In that paper, Srinivas et al. [261] assume the noise terms ϵ'_t are uniformly bounded by σ_A which is equivalent to all ϵ'_t having support on the interval $[-\sigma_A, \sigma_A]$. Since this assumption is shared by the seminal paper [261], we do not believe it to be overly restrictive in our theoretical analysis.

Regret

We quantify the performance of our method by measuring regret. Our main analysis relies on Bayesian regret [118], defined as

$$J_t^B = 1 - \mathbb{E}\left[P(r = r_\star | H_t = h_t)\right], \tag{6.10}$$

where the expectation is taken over the following:

- Our prior distribution, M_t, representing our uncertainty over the true unknown payoff vector $M_\star$ at timestep t.
- The randomness in the actions we have taken and the corresponding observations we have received up to timestep t. These are encoded by our history H_t, in which a particular realization is $h_t = a_1, m_1, ..., a_t, m_t$. $a_t \sim A_t$, our distribution over actions to take at timestep t, and $m_t \sim M_\star(a_t) + \epsilon'_t$ our clipped noise model when interacting with the game.

In this formulation $r_\star$ is used to denote the α-rank with the highest probability under r at time t, where r is the distribution over α-ranks according to the prior, $P(r) = P(M_t \in f^{-1}(r))$. Since J_t^B, like all purely Bayesian notions, does not involve the ground truth payoff, we need to justify its practical relevance. We do this by benchmarking it against two notions of frequentist regret. The first measures how accurate the probability we assign to the ground truth $r_{\text{GT}} = f(M_\star)$ is

$$J_t^F = 1 - \mathbb{E}_{h_t}\left[P(r = r_{\text{GT}}|H_t = h_t)\right]. \tag{6.11}$$

The second measures if the mean of our payoff belief, which we denote M_μ, evaluates to the correct α-rank

$$J_t^M = 1 - \mathbb{E}_{h_t}\left[\delta\Big[f(M_\mu) = r_{\text{GT}}\Big]\right], \tag{6.12}$$

where the symbol $\delta\Big[\text{predicate}\Big]$ evaluates to 1 or 0 depending on whether the predicate is true or false. For both these notions of regret the expectation is taken only over the history $h_t \sim H_t$.

Permutation Property

We begin by explicitly stating a property of the infinite-α version of α-rank. The function f computing the α-rank satisfies the permutation property, defined as

$$\pi(M_1) = \pi(M_2) \implies f(M_1) = f(M_2). \tag{6.13}$$

Here, $\pi(M)$ denotes the ordering of the elements of the vector M using the standard $\geq$ operation on real numbers. This is the same property exploited by frequentist

analysis by Rowland et al. [251]. Letting $R := \mathcal{R}^{|\mathcal{S}|}$, be the space of all valid α-ranks, Property (6.13) implies that R is a finite set and

$$|R| \leq N!, \tag{6.14}$$

where $N := |\mathcal{S}|$ the number of pure strategies/actions. Note that our proofs consider the general multi-population case of α-rank, and our not restricted to just the single population scenario.

Separability Assumption

Similarly to the work of Rowland et al. [251], we limit ourselves to payoffs that are distinguishable in order to make α-rank robust to small changes in the payoffs. We assume that there exists a constant $\Delta > 0$ such that for all payoff indices i, j

$$|M_\star(i) - M_\star(j)| \geq \Delta. \tag{6.15}$$

Information Gain and Entropy

We recall a formula for the information gain in terms of the entropy:

$$\begin{aligned}
&\mathbb{I}(r\,;\,(\tilde{M}_t(a), a) \mid H_t = h_t) \\
&= \mathbb{H}\left(r \mid H_t = h_t\right) - \mathbb{H}\left(r \mid H_t = h_t, A_t = a, \tilde{M}_t(a)\right) &(6.16)\\
&= \mathbb{H}\left(r \mid H_t = h_t\right) - \mathop{\mathbb{E}}_{\tilde{m}_t \sim \tilde{M}_t(a)}\left[\mathbb{H}\left(r \mid H_t = h_t, A_t = a, \tilde{M}_t(a) = \tilde{m}_t\right)\right]. &(6.17)
\end{aligned}$$

6.6.1 Regret Bound For Policy Maximizing Information Gain on Payoffs

We now show a regret bound for a policy that maximizes information gain on the payoffs. Define:

$$\pi_{\text{IGM}}(a|H_t = h_t) = \arg\max_a \mathbb{I}(\tilde{M}_t\,;\,(\tilde{M}_t(a), a) \mid H_t = h_t), \tag{6.18}$$

as the policy which selects the action that maximises the information gain on the payoffs (given any history H_t and prior M_t). Let H_T^{IGM} denote the history when following π_{IGM} for T timesteps.

Proposition 1 [Regret Bound For Information Gain on Payoffs] If we select actions using strategy π_{IGM}, the regret at timestep T is bounded as

$$J_T^B \leq J_T^F \leq 1 - \mathbb{E}_{h_T \sim H_T^{\text{IGM}}}\left[P(r = r_{\text{GT}} | H_T = h_T)\right] \leq T e^{g(T)} \tag{6.19}$$
$$\text{where } \; g(T) = \mathcal{O}(-\sqrt[3]{\Delta^2 T}).$$

Proof. We know that $P(r = r_\star | H_T = h_T) \geq P(r = r_{\text{GT}} | H_T = h_T)$ since $r_\star$ is defined as the α-rank with the highest probability under r and time t.
Thus, $J_T^B \leq J_T^F$ for any history.

Fix a history h_T. By assumption of separability, we have

$$P(r = r_{\text{GT}} | H_T = h_t) \geq P\left(|M_t - M_\star|_\infty \leq \frac{\Delta}{2}\right). \tag{6.20}$$

We now use concentration results for Gaussian Processes. Specifically, we invoke Corollary 5, stated later, together with an explicit formula for $g(T)$.

This proves $J_T^F \leq T e^{g(T)}$, ending our proof. □

6.6.2 Regret Bound For Policy Maximizing Information Gain on α-Ranks

We move on to show a bound for a policy that maximizes information gain on the α-ranks. Define:

$$\pi_{\text{IGR}} = \underset{a_1,\ldots,a_T}{\arg\max}\, \mathbb{I}(r\,;\, (\tilde{M}_1(a_1), a_1), \ldots, (\tilde{M}_T(a_T), a_T)), \tag{6.21}$$

as the policy which maximizes information gain on the α-ranks directly. Note that this is an extension of (6.1) to T-step look-ahead.

Let H_T^{IGR} denote the history when following π_{IGR} up to timestep T. Denote by $h_b(p) = -(p \log p + (1-p)\log(1-p))$ the entropy of a Bernoulli random variable with parameter p, and denote h_b^{-1} as the inverse of the restriction of h_b to the interval $[1/2, 1]$.

Proposition 2 Expanded [Regret Bound For Information Gain on Belief over α-Ranks] If we select actions using strategy π_{IGR}, the bayesian regret is bounded as

$$J_T^B = 1 - \mathbb{E}_{h_T \sim H_T^{\text{IGR}}} \left[P(r = r_\star | H_T = h_T) \right] \leq$$
$$1 - \delta\Big[Te^{g(T)} \leq (N \log N)^{-1}\Big] \left[h_b^{-1}(\mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right]) \right], \tag{6.22}$$

and $g(T)$ is as in Proposition 4.

Proof. We start by bounding the entropy of the α-rank distribution. Let the abbreviation $p^\star = P(r = r_\star | H_T = H_T^{\text{IGM}})$.

We have

$$\begin{aligned}
&\mathbb{H}\left(r \mid H_T^{\text{IGR}}\right) \\
&\quad \overset{(a)}{\leq} \mathbb{H}\left(r \mid H_T^{\text{IGM}}\right) \\
&\quad \overset{(b)}{\leq} \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(p^\star) + (1 - p^\star) \log(|R|) \right] \\
&\quad \overset{(c)}{\leq} \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(p^\star) + (1 - p^\star) N \log N \right].
\end{aligned}$$

Here, (a) follows from the definition of π_{IGR} and equation (6.17), (b) follows by Lemma 6 and (c) holds because $|R| \leq N!$ by Equation (6.14). Combining the above with the bound $1 - p^\star \leq Te^{g(T)}$ from Proposition 4, we have

$$\mathbb{H}\left(r \mid H_T^{\text{IGR}}\right) \leq \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(p^\star) + (1 - p^\star) N \log N \right] \tag{6.23}$$
$$\leq \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(p^\star) + Te^{g(T)} N \log N \right]. \tag{6.24}$$

Let us now assume that $Te^{g(T)} \leq 1/2$, since we are interested in the behaviour of our regret bound as $T \to \infty$, and we know that as $T \to \infty, Te^{g(T)} \to 0$. If $Te^{g(T)} > 1/2$ then we can trivially bound our expression above by 1. Then $p^\star \geq 1 - Te^{g(T)} \geq 1/2 \implies h_b(p^\star) \leq h_b(1 - Te^{g(T)})$.

We now proceed to bound the probability of $r_\star$ in terms of the entropy of the α-ranks. We have

$$h_b(P(r = r_\star | H_T^{\text{IGR}} = h_t)) \leq \mathbb{H}\left(r \mid H_T^{\text{IGR}}\right).$$

This, together with (6.24) and $h_b(p^\star) \leq h_b(1 - Te^{g(T)})$ implies

$$h_b(P(r = r_\star | H_T^{\text{IGR}} = h_t)) \leq \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right]. \quad (6.25)$$

Since the codomain of h_b is $[0, 1]$, we must introduce additional restrictions in order to be able to invert the function.

To ensure $h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \leq 1$ we restrict our analysis to when $Te^{g(T)} \leq (N \log N)^{-1}$. Note that this subsumes our earlier restriction of $Te^{g(T)} \leq 1/2$. Again, we can trivially bound our final expression above by 1, should this condition not be met.

We denote by h_b^{-1} the inverse of the restriction of h_b to the interval $[1/2, 1]$. Note that $h_b(x) \leq y \implies x \geq h_b^{-1}(y)$ for $x \in [1/2, 1], y \in [0, 1]$.

$$h_b(P(r = r_\star | H_T^{\text{IGR}} = h_t)) \leq \mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right] \quad (6.26)$$

$$\implies$$

$$P(r = r_\star | H_T^{\text{IGR}} = h_t) \geq h_b^{-1}(\mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right]) \quad (6.27)$$

$$\implies$$

$$1 - P(r = r_\star | H_T^{\text{IGR}} = h_t) \leq 1 - h_b^{-1}(\mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right]). \quad (6.28)$$

Finally, we state our final regret bound incorporating our restrictions/assumptions we have made.

$$1 - P(r = r_\star | H_T^{\text{IGR}} = h_t) \leq$$
$$1 - \delta\left[Te^{g(T)} \leq (N \log N)^{-1} \right] \left[h_b^{-1}(\mathbb{E}_{h_T \sim H_T^{\text{IGM}}} \left[h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N \right]) \right]. \quad (6.29)$$

This then proves the expanded form of the proposition.

The simpler form of the Proposition in Section 6.5 then follows. This is because as $T \to \infty, Te^{g(T)} \to 0$ and both $h_b(1 - Te^{g(T)}), Te^{g(T)} N \log N \to 0$ thus ensuring that $h_b^{-1}(h_b(1 - Te^{g(T)}) + Te^{g(T)} N \log N) \to 1$.

□

We use the following result by Srinivas et al. [261, their Theorem 6], which we specialize in our notation. We use the term Gaussian Process despite the fact that the index set is finite, since the model includes observation noise.

Lemma 4 (Srinivas et al., Concentration for a Gaussian Process)**.** *Consider a Gaussian Process M, with N indices. Assume M uses a zero-mean prior with constant variance σ_0^2 and observation noise σ_A. The posterior process M_t is obtained by conditioning on t observations. The observations are obtained as $m_t = M_\star(a_t) + \epsilon'_t$, where ϵ'_t are i.i.d random variables with support bounded by $[-\sigma_0, \sigma_0]$. Denote the RKHS norm of $M_\star$ under the GP prior with $\|M_\star\|_k$. Denote by $\gamma_t^\star$ the maximum information gain about M obtainable in t timesteps. Then, for any $\Delta > 0$, and for any timestep t, we have*

$$P\left[|M - M_\star|_\infty \leq \frac{\Delta}{2}\right] \geq 1 - te^{-\sqrt[3]{\frac{\left(\frac{\Delta}{2\sigma_t^{max}}\right)^2 - 2\|M_\star\|_k}{300\gamma_t^\star}}}. \tag{6.30}$$

The above lemma requires knowledge of the RKHS norm and the maximum obtainable information gain.

Lemma 5 (Worst-Case Constants)**.** *For any kernel, we have*

$$\|M_\star\|_k \leq \frac{1}{\sigma_0^{-2}} \|M_\star\|_2^2 \;\; \textit{and} \;\; \gamma_t^\star \leq \frac{1}{2} \log\det(I + \sigma_A^{-2} K).$$

Moreover, for a strategy that maximizes information gain on payoffs, we have

$$\sigma_t^{max} \leq \frac{\sigma_A \sigma_0}{\sqrt{\sigma_A^2 + (\frac{T}{N} - 1)\sigma_0^2}}.$$

Proof. The inequalities for posterior variance and the RKHS norm are obtained by using the independent kernel, which represents the worst-case. The inequality for information gain follows by writing

$$\gamma_t^\star = \frac{1}{2} \log \frac{\det(I + \sigma_A^{-2} K)}{\det(I + \sigma_A^{-2} \Sigma)} \leq \frac{1}{2} \log\det(I + \sigma_A^{-2} K). \tag{6.31}$$

The inequality follows since the denominator is greater than one. Here, we denoted the prior covariance with K and the posterior covariance with Σ. □

Corollary 5. *For a strategy that maximizes the payoff information gain and for any time-step T, we have:*

$$P\left[|M_t - M_\star|_\infty \leq \frac{\Delta}{2}\right] \geq 1 - Te^{g(T)}, \quad \textit{where } g(T) = \mathcal{O}(-\sqrt[3]{\Delta^2 T})$$

Specifically,

$$g(T) = -\sqrt[3]{\frac{\left(\frac{\Delta}{2}\right)^2 \frac{\sigma_A^2 + (\frac{T}{N} - 1)\sigma_0^2}{\sigma_A^2 \sigma_0^2} - 2\frac{1}{\sigma_0^{-2}}\|M_\star\|_2^2}{300\frac{1}{2}\log\det(I + \sigma_A^{-2}K)}}.$$

Lemma 6 (Upper Bound on Entropy)**.** *For any discrete random variable x with n outcomes, we have, for each outcome i*

$$\mathbb{H}(x) \leq h_b(p_i) + (1 - p_i)\log(n-1).$$

Proof. Without loss of generality, assume $i = 1$.

$$\begin{aligned}
\mathbb{H}(x) &= -p_1 \log p_1 - \sum_{j>1} p_j \log(p_j) \\
&= -p_1 \log p_1 - (n-1)\sum_{j>1} \frac{1}{n-1} p_1 \log(p_j) \\
&\overset{(a)}{\leq} -p_1 \log p_1 - (n-1)\left(\sum_{j>1} \frac{p_j}{n-1}\right)\log\left(\sum_{j>1} \frac{p_j}{n-1}\right) \\
&= -p_1 \log p_1 - (1-p_1)\log\left(\frac{1-p_1}{n-1}\right) \\
&= -p_1 \log p_1 - (1-p_1)\log\left(\frac{1-p_1}{n-1}\right) \\
&= h_b(p_j) + (1-p_j)\log(n-1)
\end{aligned}$$

There, (a) follows from Jensen's inequality applied to the function $x \log x$. □

6.7 Experiments

In this section, we describe our results on synthetic games, graphing the Bayesian regret J_t^B described in Section 6.5. We also justify the use of Bayesian regret,

showing that it is highly coupled with the ground truth payoff. We benchmark two versions of our algorithms, αIG (Bins) and αIG (NSB), which differ in the employed entropy estimator. We compare to three baselines: **RG-UCB**, a frequentist bandit algorithm [251], **Payoff**, which maximizes the information gain about the payoff distribution, and **Uniform**, which selects payoffs uniformly at random. **RG-UCB** represents the current SOTA in this domain, **Payoff** represents the performance of a Bayesian method that does not take into account the structure of the mapping between payoffs and α-ranks, and **Uniform** provides a point of reference as the simplest/most naive method[1]. A detailed explanation of the experimental setup[2] and details on the used hyperparameters are included in Appendix E.1.

It is important to reiterate that all of experimental results are intended as a proof of concept for the use of Gaussian Processes in conjunction with our methods. All but one of our experiments utilises independent Gaussians for each entry of the payoff matrix, in order to test our approach without access to useful domain knowledge.

6.7.1 Good-Bad Games

To investigate our algorithm, we study two environments whose payoffs are shown in Figure 6.5. We start with the relatively simple environment with 4 agents. Figure 6.5 (Left) shows the expected payoffs, which we can interpret as the win-rate. Samples are drawn from a Bernoulli distribution with the appropriate mean. We refer to the environment as '2 Good, 2 Bad' since agents 1 and 2 are much stronger than the other 2 agents, winning 100% of the games against them. Since the ordering between agents 3 and 4 has no effect on the α-rank, gathering samples to determine this ordering (highlighted in **Purple**) does not affect the belief distribution over α-ranks. Furthermore, since we treat this as a 1-population game, the entries highlighted in **Green** where each agent plays against themselves do not affect the α-rank. Entries that are necessary to determine the ordering between agents 1 and 2 are the most relevant for the α-rank and are highlighted in **Red**. Since agent 2

[1] We do not include Uniform on the regret graphs, since there is no reasonable value we could compute for it.

[2] Code is available at `github.com/microsoft/InfoGainalpharank`.

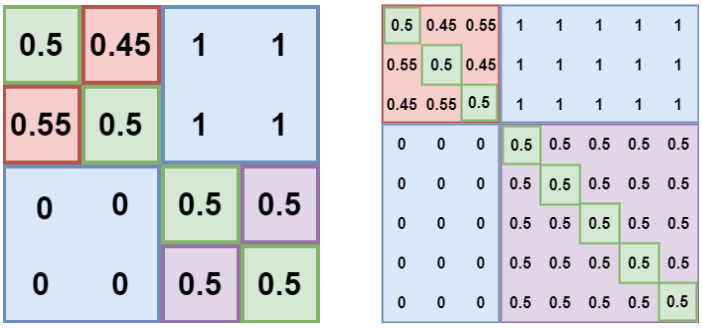


Figure 6.5: Payoff matrices for 2 Good, 2 Bad (Left) and 3 Good, 5 Bad (Right). Best viewed in color.

is slightly better than agent 1, the true α-rank is $(0, 1, 0, 0)$. However, it can be difficult to determine the correct ordering between agents 1 and 2 without drawing many samples from these entries. The game thus provides a model for the common scenario of agents with clustered performance ratings.

Focusing on Relevant Payoffs

Figure 6.6 presents the behavior of our method and RG-UCB on this task. As expected, RG-UCB splits its sampling between the **Red** entries and the **Purple** entries, whereas our method concentrates its sampling much more significantly on the relevant entries, determining the ordering between agents 1 and 2. This is because, in contrast to our method, RG-UCB aims to correctly determine the ordering between **all** entries used in the calculating of α-rank, irrespective of whether they matter for the final outcome.

Figure 6.7 additionally shows the behaviour of the methods on the more challenging 3 Good, 5 Bad.

Wasserstein Payoff Selection Does Well

Comparing the Wasserstein Criterion with Information Gain payoff section, we can see that it enjoys better concentration of the sampling on the **Red** entries, and improved performance towards the end of training.

Figure 6.9 shows the values used by the different objectives during training. The top row shows the values after sampling 5 values for each entry, showing a clear separation between the **Red** entries and the rest. The bottom row shows the

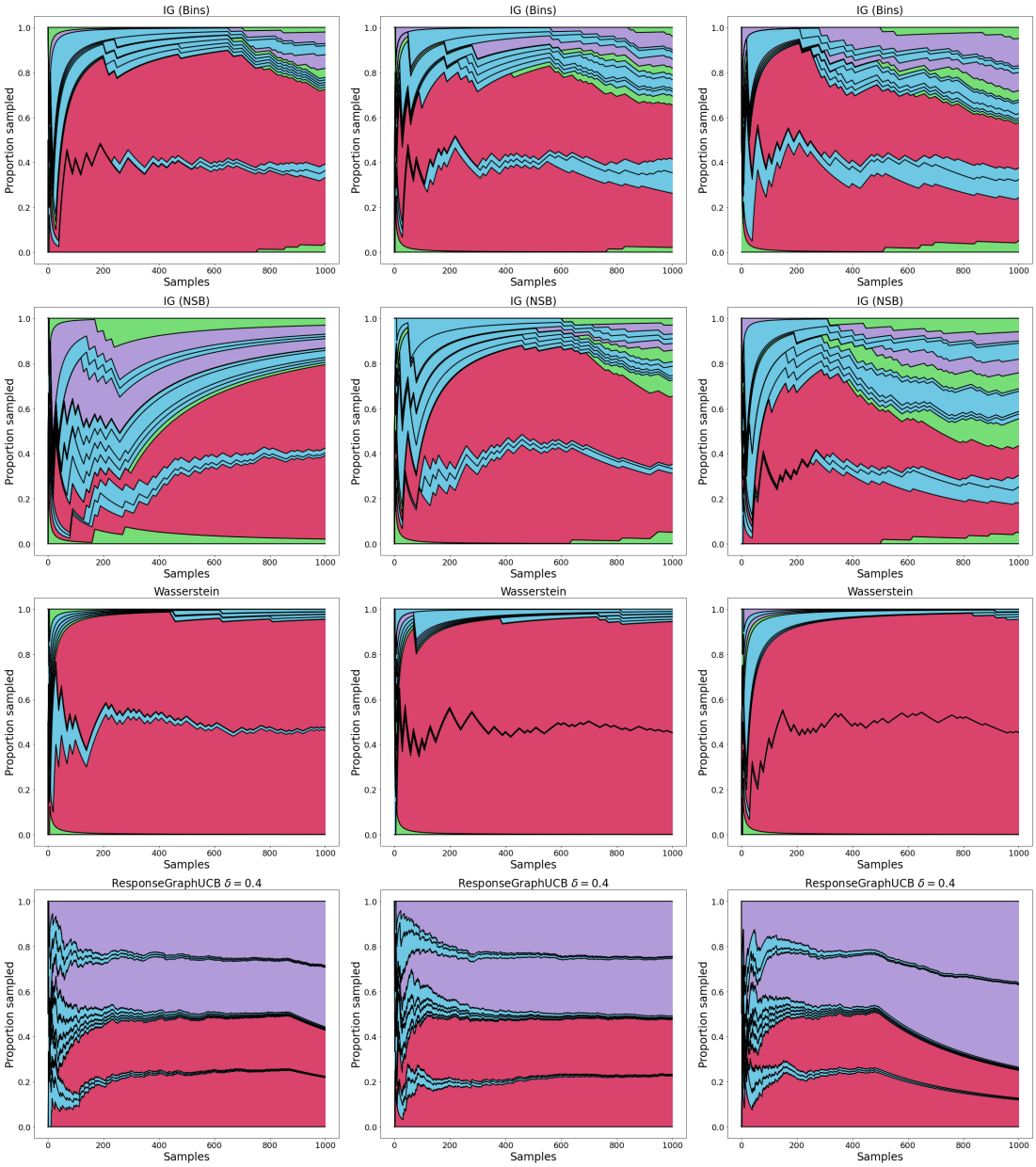


Figure 6.6: Proportion of entries sampled on 2 Good, 2 Bad for different methods and objectives across 3 seeds.

values after additionally sampling 250 values for the Red entries. We can then see a large difference between the Wasserstein and Entropy-based objectives. As desired the Wasserstein-based objective shows a large separation between the Red entries and the others, additionally assigning the smallest values to the irrelevant Green, and Purple entries.

Figures 6.10 and 6.11 additionally show the results on 3 Good, 5 Bad.

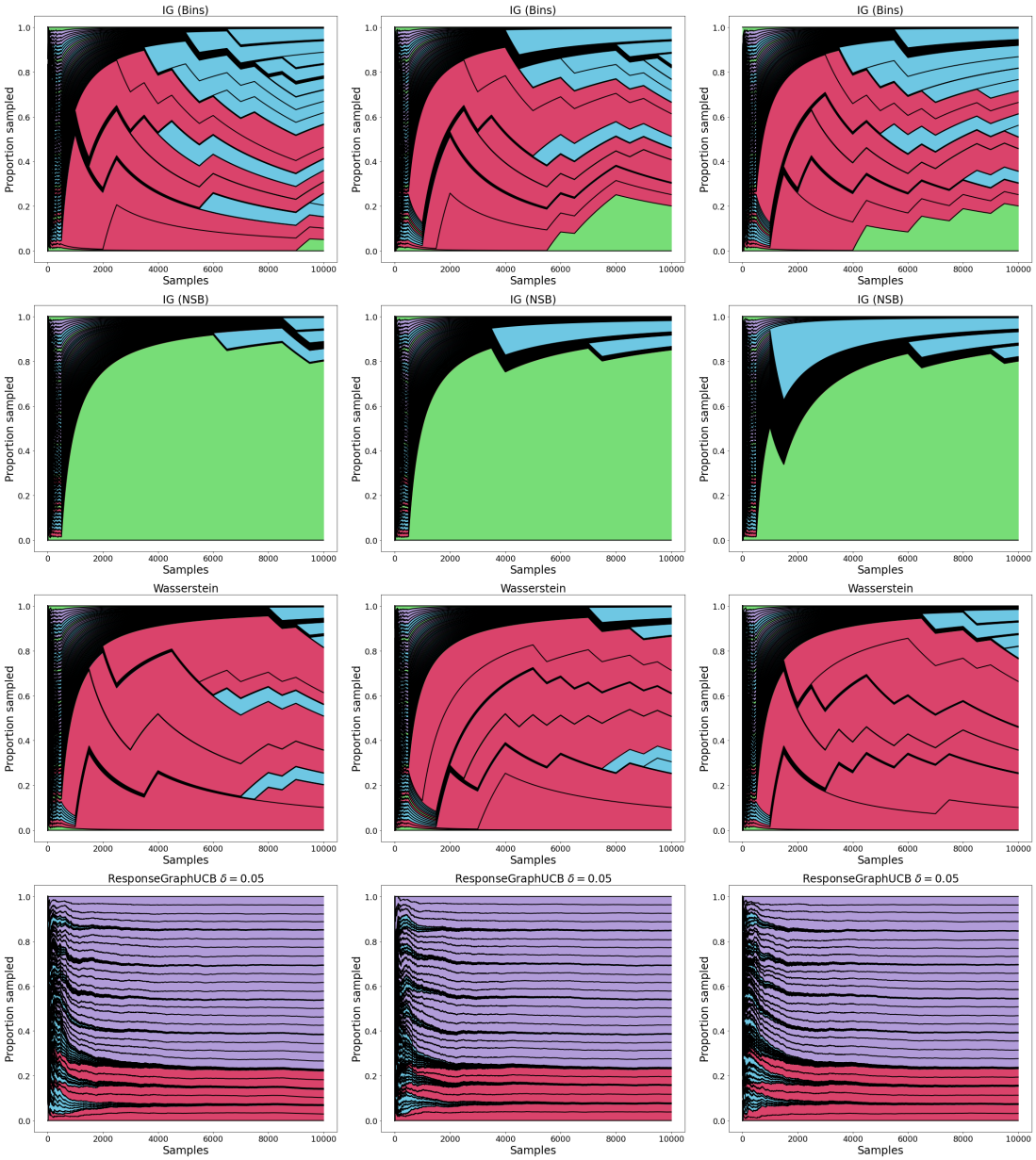


Figure 6.7: Proportion of entries sampled on 3 Good, 5 Bad for more seeds.

Bayesian and Frequentist Regret Go Down

Figure 6.8 shows the resulting performance of the methods on this task, measured by the regret. Due to the relative simplicity of the game, there is limited benefit to our method over RG-UCB, but there is a clear benefit over more naive methods that systematically or uniformly sample the entries. We can see that the Bayesian regret J_t^B and Frequentist regrets J_t^F and J_t^M are highly correlated, providing empirical justification for minimizing J_t^B and validating that our method is concentrating

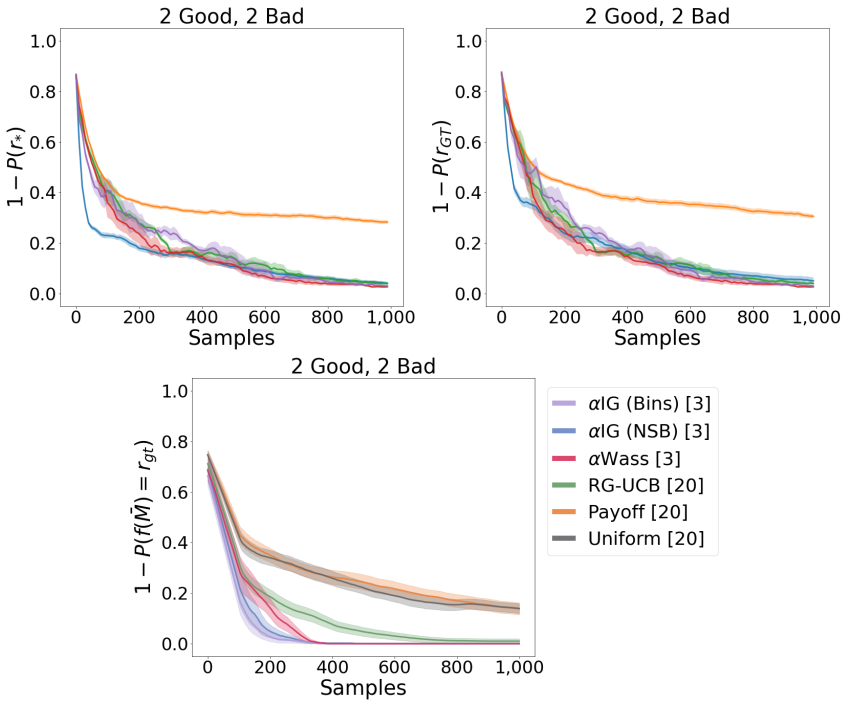


Figure 6.8: Results for 2 Good, 2 Bad. Graphs show the mean and standard error of the mean over multiple runs (shown in brackets) of 10 repeats each.

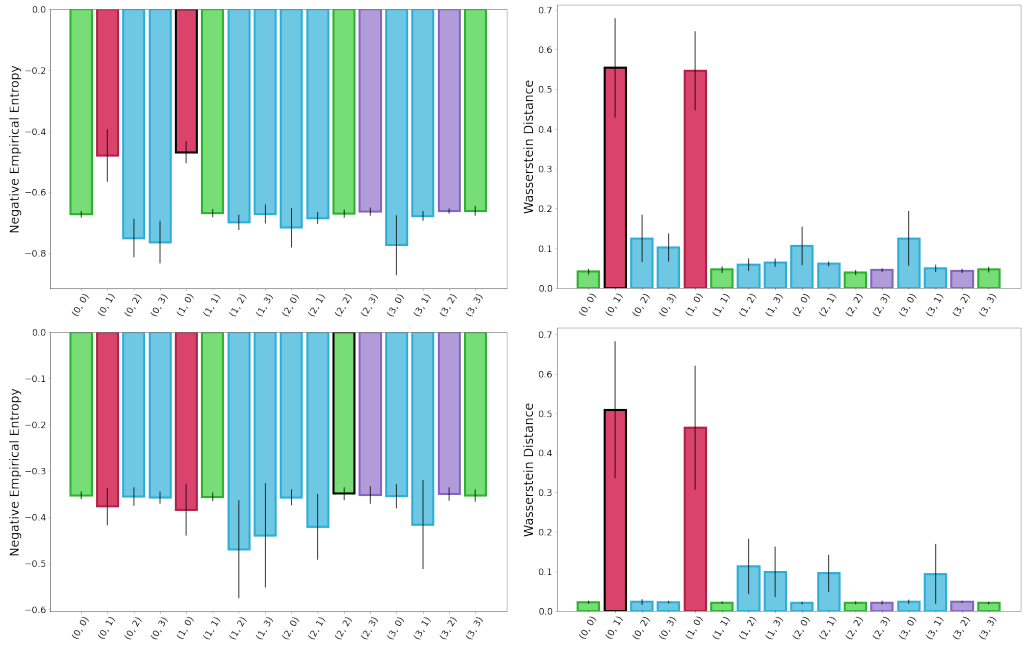


Figure 6.9: Value of the objectives for each entry after sampling 5 values for every entry (top) and additionally sampling 250 values for the red entries (below). Mean and standard deviation are plotted across 10 seeds, maximum entry is highlighted in black.

on the ground truth.

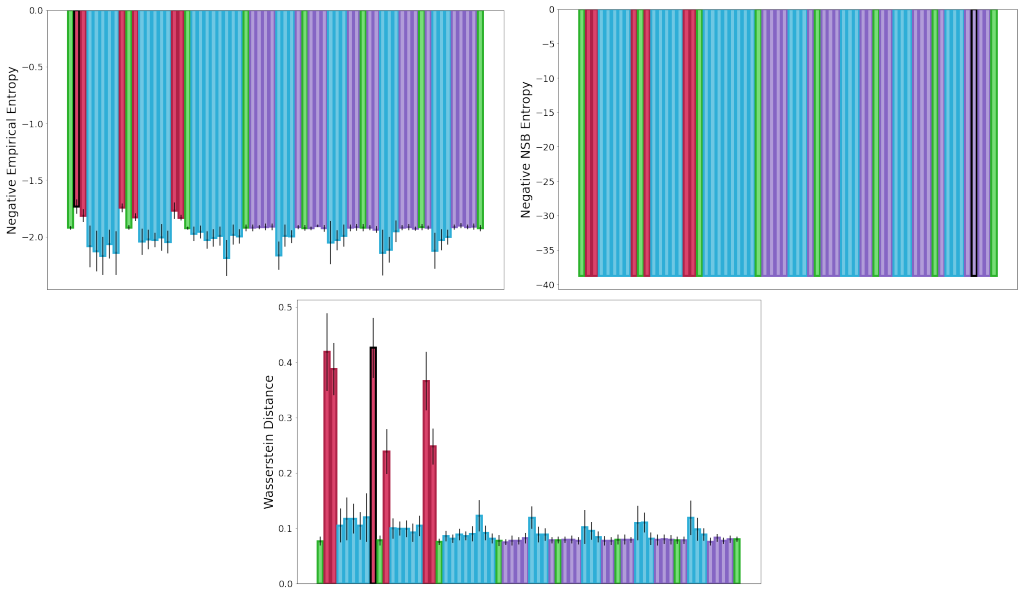


Figure 6.10: Values of the objectives for each entry on '3 Good, 5 Bad' after sampling 5 values for every entry. Mean and standard deviation across 10 seeds is shown, maximum highlighted in black.

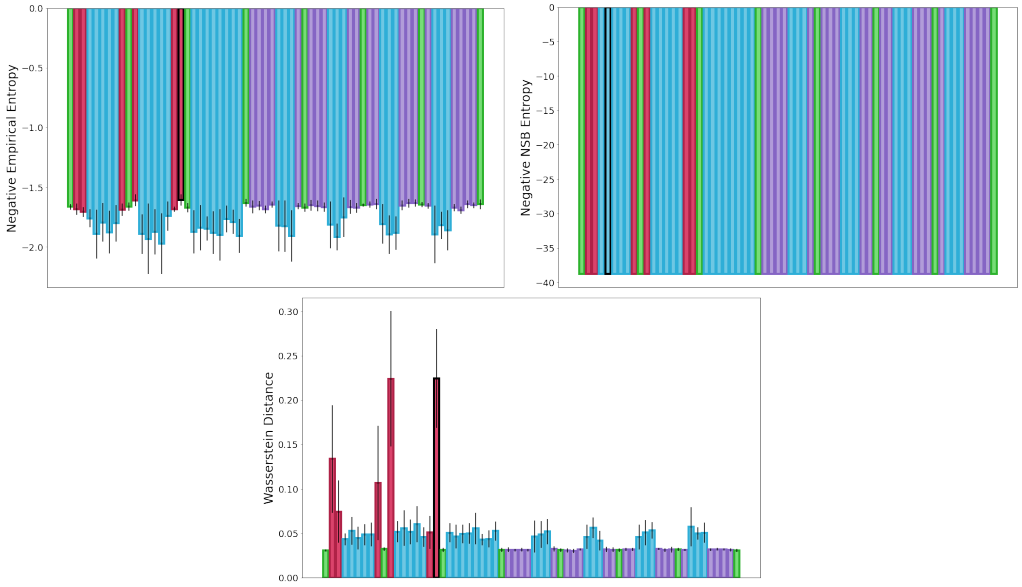


Figure 6.11: Values of the objectives for each entry on '3 Good, 5 Bad' after sampling 5 values for every entry, and additionally sampling 1000 values for the red entries. Mean and standard deviation across 10 seeds is shown, maximum highlighted in black.

Comparing Entropy Estimators

We also investigate a larger scale version of 2 Good, 2 Bad with 3 good and 5 bad agents. Figure 6.12 shows the results, demonstrating a clear benefit for our method using the Binning estimator for the Information Gain or the Wasserstein

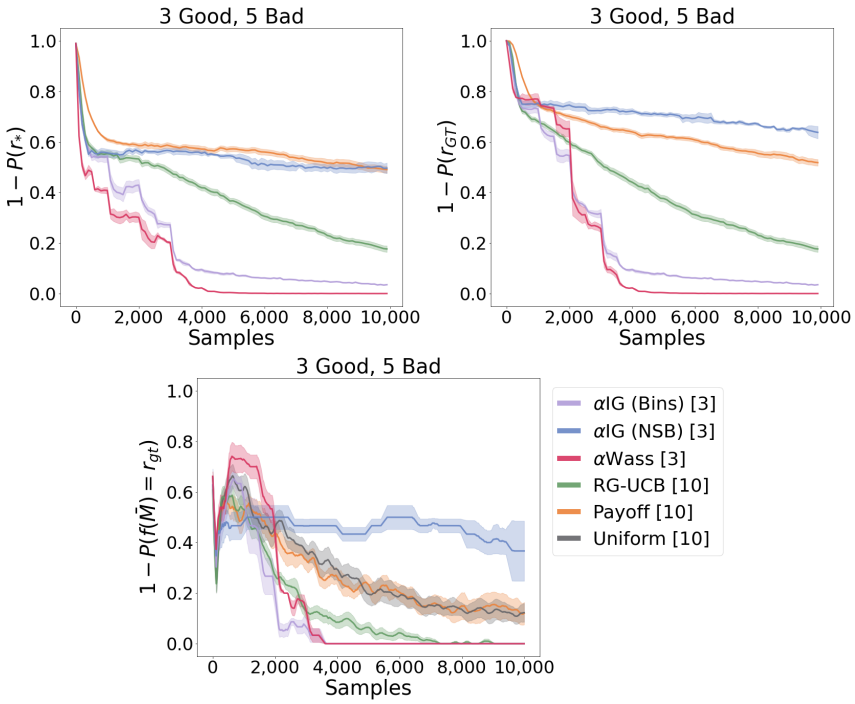


Figure 6.12: Results for 3 Good, 3 Bad. Graphs show the mean and standard error of the mean over multiple runs (shown in brackets) of 10 repeats each.

objective. The performance of the NSB entropy estimator is not surprising given the significantly larger nature of this task compared to '2 Good, 2 Bad'. A necessary part of the NSB estimator is an upper-bound on the total number of atoms in the distributions, for which we only have a crude approximation that grows exponentially with the size of the payoff matrix. Figure 6.7 shows the proportion of entries sampled for αIG (Bins), the Wasserstein objective, and RG-UCB. Once again, RG-UCB spends a significant part of its sampling budget determining the ordering between agents that do not have an effect on the α-rank of the game (in this task agents 3 to 8). In contrast, our methods concentrate their sampling on the **Red** entries that determine the payoffs between the top 3 agents, and hence the true α-rank. In general, our algorithm does not depend as much on accurate estimates on entropy but on identifying the distribution with a lowest entropy, for which the NSB estimator isn't tuned.

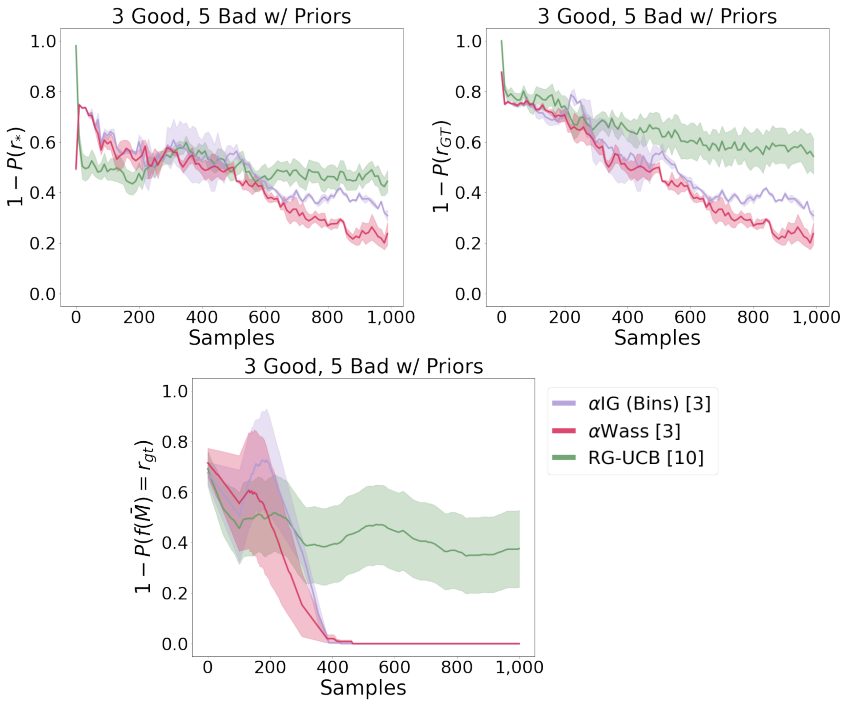


Figure 6.13: Results on 3 Good, 5 Bad when incorporating prior knowledge into the models.

6.7.2 Incorporating Prior Knowledge

A large benefit of our Bayesian-based approach is the ability to incorporate prior knowledge and modelling assumptions into our model in a principled manner. To demonstrate the benefits, we incorporate the following prior knowledge into both our algorithm and RG-UCB: 1) $M(\sigma, \tau) + M(\tau, \sigma) = 1$. 2) Entries in their respective blocks are equal to each other (except for the top left block). The first constraint is extremely common in constant-sum games such as Go, Chess and StarCraft where there is a winner and a loser. The second constraint can arise in many games with a cyclic component such as Rock-Paper-Scissors, in which all Rock-like strategies will behave very similarly against all Paper-like strategies. Many high-level competitive games such as StarCraft II explicitly try and feature such cyclic components, so that no one strategy can be dominant over all others. A detailed description of the setup is included in Appendix E.1.8. Figure 6.13 compares the performance of αIG, αWass, and RG-UCB on 3 Good, 5 Bad when utilizing this prior knowledge. We can see that our approach significantly outperforms RG-UCB on this task,

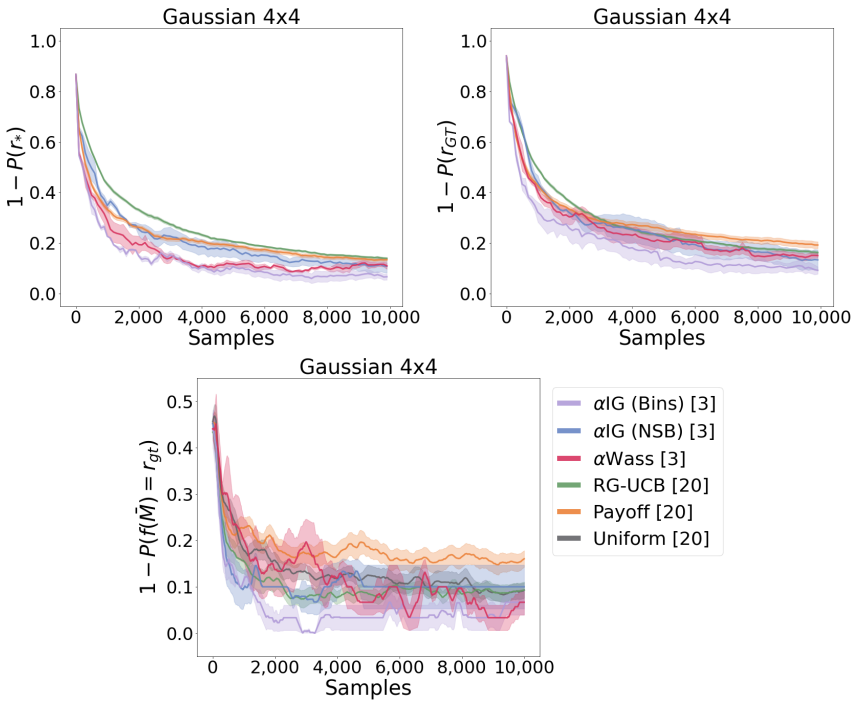


Figure 6.14: Results for the 4x4 Gaussian Game. Graphs show the the mean and standard error of the mean over multiple runs (shown in brackets) of 10 randomly sampled games.

further demonstrating the importance of our direct information gain objective. The results also show significantly improved sample efficiency over the results in Figure 6.12, demonstrating that our αIG and αWass are able to efficiently take advantage of the prior knowledge supplied.

Whilst this particular experiment does utilise some of the modelling power of a Gaussian Process, it is still only encoding quite basic prior assumptions. A significantly more complicated setting would be required in order to actually have the ability to take advantage of the modelling power of a Gaussian Process. For example a self-play setting in which each member of the population can be described or summarised via a contextual vector (encoding information such as the algorithm used, the length of training, the entropy of its policy, etc).

6.7.3 Gaussian Games

Figure 6.14 shows the results on 4x4 games with Gaussian noise, demonstrating improved performance across all 3 regret metrics for the αIG (Bins). This is empirical

confirmation of our theoretical results, and shows that our method achieves better performance compared to RG-UCB on general games.

6.7.4 Conditioning of the belief distribution

In our experiments we found that setting $N_c = 1$ as suggested by theory is not always sufficient and use $N_c = 100$ for all experiments.

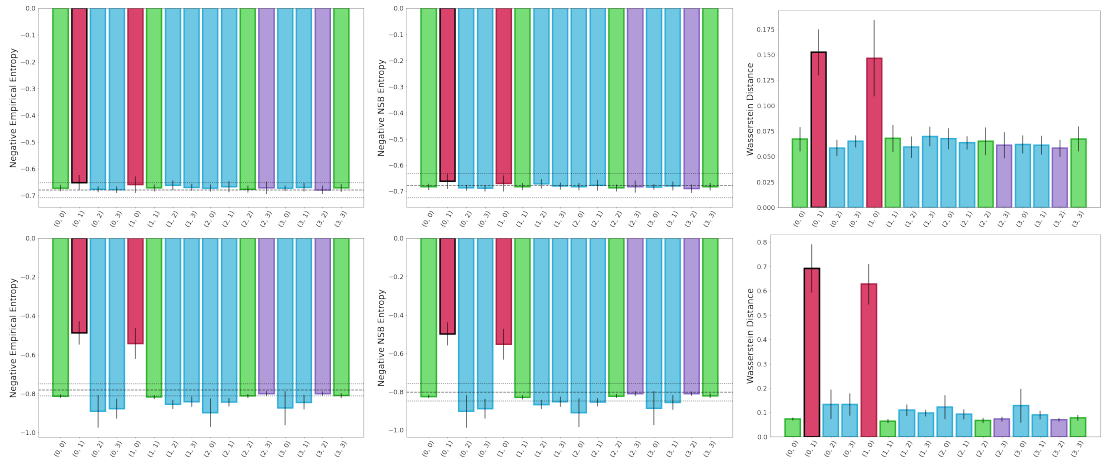


Figure 6.15: Comparing the values of the objectives for each entry after sampling 5 values for every entry. **Top** shows the results for $N_c = 1$. **Bottom** shows the results for $N_c = 100$. Mean and standard deviation are plotted across 10 seeds, maximum entry is highlighted in black. The mean and standard deviation of the (estimated) entropy of the current belief distribution is also plotted as a dashed horizontal line.

After drawing a sample $m'_t \sim M_t + \epsilon$, we then condition our belief distribution over α-rank on this sample N_c times and then approximate the Entropy of the resulting hallucinated belief distribution (or the Wasserstein distance between the current belief and the hallucinated belief). Theory suggests that setting $N_c = 1$ is sufficient, however empirically we found that this did not produce satisfactory results. Figure 6.15 shows that only conditioning once produces very little separation between the values for the different entries. Additionally, we can see that there is very little separation between the current belief's entropy and the hallucinated belief's entropy. In contrast, we can see that conditioning 100 times produces significantly more separation. Figure 6.16 shows the same trend, after additionally sampling 250 values for the red entries. The Wasserstein objective shows the same

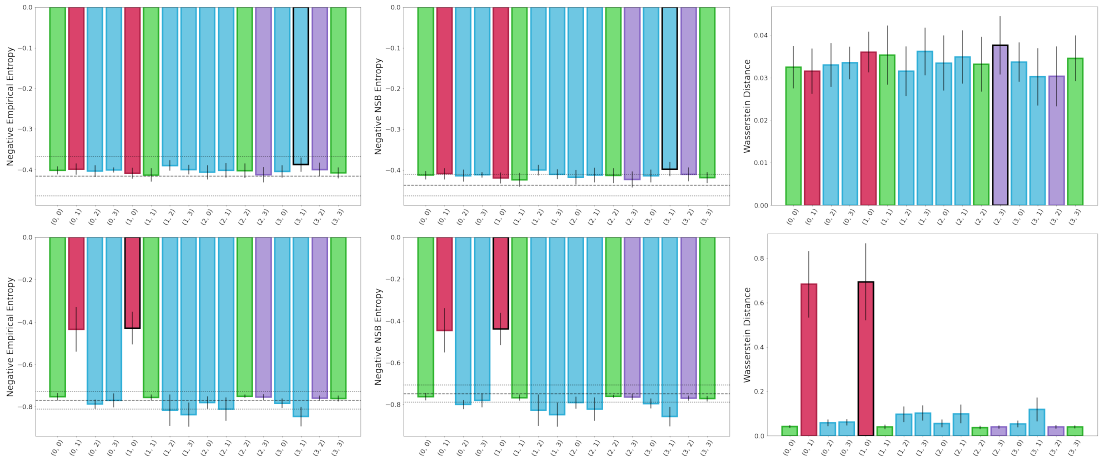


Figure 6.16: Comparing the values of the objectives for each entry after sampling 5 values for every entry, and then additionally sampling 250 values for the Red entries. **Top** shows the results for $N_c = 1$. **Bottom** shows the results for $N_c = 100$. Mean and standard deviation are plotted across 10 seeds, maximum entry is highlighted in black. The mean and standard deviation of the (estimated) entropy of the current belief distribution is also plotted as a dashed horizontal line.

trend, that conditioning more than once produces significantly more separation. A Wasserstein Distance of 0 indicates that the two distributions are identical.

6.8 Conclusions

We described αIG, an algorithm for estimating the α-rank of a game using a small number of payoff evaluations. αIG works by maximizing information gain. It achieves competitive sample efficiency and allows a way of building in prior knowledge about the payoffs.

7
Afterword

In this thesis we have proposed a number of methods to tackle major bottlenecks that RL agents and associated algorithms encounter in the process of training.

In Part I we explored the bottleneck of exploration, where we focussed on sparse-reward environments. We proposed OPIQ, a model-free Q-learning based algorithm that scales the use of an optimistic initialisation from the tabular setting to the Deep RL one. We proved that this approach is provably efficient, thereby providing a firm theoretical foundation to build on, and then demonstrated it can scale to complex environments such as Montezuma's Revenge.

In Part II we investigated the difficulties of learning as a team of agents in the fully cooperative multi-agent setting. In Chapter 4, we first explored a line of research into value function factorisation and proposed QMIX, another Q-learning based algorithm that expanded upon the class of value functions representable through the use of a learnt monotonic function. We introduced a benchmark for Deep MARL in the Centralised Training and Decentralised Execution paradigm, and showed that QMIX can make efficient use of the state information to significantly outperform other approaches. We next explored the consequences of the factorisation that QMIX employs in Chapter 5, and proposed the use of a weighting in our objective function. We provided theoretical justification that our weighting can remedy the shortcomings of QMIX's factorisation, and demonstrated it can enable the learning

of optimal behaviour in situations that QMIX would previously fail in. Finally, we investigated the utility of a weighting beyond its original motivations of improving QMIX and identified some new bottlenecks that arose in our proposed approaches.

Part III examined a setting in which we must produce our own opponents in order to be able to train in an environment. Crucial in this setting is the ability to be able to calculate a solution concept of the meta-game. We proposed αIG and αWass, two Bayesian algorithms that aim to minimise the number of samples from the meta-game that we require in order to estimate the α-Rank as accurately as possible. We demonstrated that they can exploit the structure of the meta-game, and additionally utilise any domain knowledge a practitioner might have, in order to accurately estimate the true α-Rank using fewer samples than the frequentist-based baseline of ResponseGraphUCB.

The algorithms and analysis we have produced in this thesis have contributed to improving the capabilities of RL agents that can be trained, but there is still significant room for improvement. First, our ability to explore in sparse reward settings is still very inefficient. Montezuma's Revenge for example is a very simple and small game compared to those that are played by humans nowadays. Yet, our algorithms are still unable to complete the first level without the use of billions of frames [127] or significant domain and modelling knowledge [61]. Second, many situations cannot be appropriately modelled as fully-cooperative ones in the manner that we did so in Part II. Finally, even if we could quickly and efficiently compute the α-Rank of the meta-game in a self-play training paradigm, we might still not be able to make adequate progress on our task without a prohibitively large population of agents. As the environments and agents grow more complex, this quickly becomes the major computational bottleneck in making progress.

The pace of Deep RL research, and in particular the grand challenges that have been bested by it, has been staggering to witness as a researcher in this domain. We look forward to seeing RL and related approaches be pushed even further, and the continual improvement and refining of the ideas and approaches in this field.

Appendices

A

Single-agent Exploration

A.1 Experimental Setup

A.1.1 Hyperparameters and Network architectures

In all experiments we set $\gamma = 0.99$, use RMSProp with a learning rate of 0.0005 and scale the gradient norms during training to be at most 5.

Randomised Chain

The network used is a MLP with 2 hidden layers of 256 units and ReLU non-linearities. We use 1 step Q-Learning.

Training lasts for 100k timesteps. ϵ is fixed at 0.01 for all methods except for ϵ-greedy DQN in which it is linearly decayed from 1 to 0.01 over $\{100, 50\text{k}, 100\text{k}\}$ timesteps. We train on a batch size of 64 after every timestep with a replay buffer of size 10k. The target network is updated every 200 timesteps. The embedding size used for the counts is 32. We set $\beta = 0.1$ for the scale of the count-based intrinsic motivation.

For reward subtraction we consider subtracting $\{0.1, 1, 10\}$ from the reward. For an optimistic initialisation bias, we consider setting the final layer's bias to $\{0.1, 1, 10\}$. We consider both of the methods with and without count-based intrinsic motivation.

For OPIQ and its ablations we consider:

$M \in \{0.1, 0.5, 2, 10\}, C_{\text{action}} \in \{0.1, 1, 10\}, C_{\text{bootstrap}} \in \{0.01, 0.1, 1, 10\}$.

For all methods we run 20 independent runs across the cross-product of all relevant parameters considered. We then sort them by the median test reward (largest area underneath the line) and report the median, lower and upper quartiles.

The best hyperparameters we found were:

DQN ϵ-greedy: Decay rate: 100 timesteps.

Optimistic Initialisation Bias: Bias: 1, Pseudocount intrinsic motivation: True.

Reward Subtraction: Constant to subtract: 1, Pseudocount intrinsic motivation: False.

OPIQ: M: 0.5, C_{action}: 1, $C_{\text{bootstrap}}$: 1.

OPIQ without Optimistic Bootstrapping: M: 2, C_{action}: 10.

OPIQ without Pseudocounts: M: 2, C_{action}: 10, $C_{\text{bootstrap}}$: 10.

For Figure 3.12 the best hyperparameters for OPIQ with differing values of M are:

M: 0.1**:** C_{action}: 10, $C_{\text{bootstrap}}$: 1.

M: 0.5**:** C_{action}: 1, $C_{\text{bootstrap}}$: 1.

M: 2**:** C_{action}: 10, $C_{\text{bootstrap}}$: 1.

M: 10**:** C_{action}: 10, $C_{\text{bootstrap}}$: 10.

Maze

The network used is the following feedforward network:

(State input: (24,24,1))

→ (Conv Layer, 3x3 Filter, 16 Channels, Stride 2) → ReLU

→ (Conv Layer, 3x3 Filter, 16 Channels, Stride 2) → ReLU

→ Flatten

→ (FC Layer, 400 Units) → ReLU

→ (FC Layer, 200 Units)

→ $\mathcal{A} = 4$ outputs.

We use 3 step Q-Learning.

Training lasts for 1mil timesteps. ϵ is decayed linearly from 1 to 0.01 over 50k timesteps for all methods except for ϵ-greedy DQN in which it is linearly decayed from 1 to 0.01 over $\{100, 50\text{k}, 100\text{k}\}$ timesteps. We train on a batch of 64 after every timestep with a replay buffer of size 250k. The target network is updated every 1000 timesteps. The embedding dimension for the counts is 128.

For DQN + PC we consider $\beta \in \{0.01, 0.1, 1, 10, 100\}$. For all other methods we set $\beta = 0.1$ as it performed best.

For reward subtraction we consider subtracting $\{0.1, 1, 10\}$ from the reward. For an optimistic initialisation bias, we consider setting the final layer's bias to $\{0.1, 1, 10\}$. Both methods utilise a count-based intrinsic motivation.

For OPIQ and its ablations we set $M = 2$ since it worked best in preliminary experiments. We consider: $C_{\text{action}} \in \{0.1, 1, 10, 100\}, C_{\text{bootstrap}} \in \{0.01, 0.1, 1, 10\}$.

For the RND bonus we use the same architecture as the DQN for both the target and predictor networks, except the output is of size 128 instead of $|\mathcal{A}|$. We scale the squared error by $\beta_{rnd} \in \{0.001, 0.01, 0.1, 1, 10, 100\}$:

For DQN + DORA we use the same architecture for the E-network as the DQN. We add a sigmoid non-linearity to the output and initialise the final layer's weights and bias to 0 as described in [123]. We sweep across the scale of the intrinsic reward $\beta_{dora} \in \{0.01, 0.1, 1, 10\}$. For DQN + DORA OA we use $\beta_{dora} = 0.01$ and sweep across $\beta_{dora_action} \in \{0.01, 0.1, 1, 10, 100\}$.

For BSP we use the following architecture:

Shared conv layers:

(State input: (24,24,1))

→ (Conv Layer, 3x3 Filter, 16 Channels, Stride 2) → ReLU

→ (Conv Layer, 3x3 Filter, 16 Channels, Stride 2) → ReLU

→ Flatten

Q-value Heads:

→ (FC Layer, 400 Units) → ReLU

→ (FC Layer, 200 Units)

→ $\mathcal{A}$ = 4 outputs.

We use $K = 10$ different bootstrapped DQN heads, and sweep over $\beta_{bsp} \in \{0.1, 1, 3, 10, 30, 100\}$.

For all methods we run 8 independent runs across the cross-product of all relevant parameters considered. We then sort them by the median episodic reward (largest area underneath the line) and report the median, lower and upper quartiles.

The best hyperparameters we found were:

DQN ϵ-greedy: Decay rate: 100k timesteps.

DQN + PC: $\beta = 0.1$.

Optimistic Initialisation Bias: Bias: 1.

Reward Subtraction: Constant to subtract: 0.1.

DQN + RND: $\beta_{rnd} = 10$.

DQN + DORA: $\beta_{dora} = 0.01$.

DQN + DORA OA: $\beta_{dora} = 0.01$ and $\beta_{dora_action} = 0.1$.

BSP: $\beta_{bsp} = 100$.

OPIQ: M: 2, C_{action}: 100, $C_{\text{bootstrap}}$: 0.01.

OPIQ without Optimistic Bootstrapping: M: 2, C_{action}: 100.

OPIQ without Pseudocounts: M: 2, C_{action}: 100, $C_{\text{bootstrap}}$: 0.1.

Montezuma's Revenge

The network used is the standard DQN used for Atari [24, 53].

We use 3 step Q-Learning.

Training lasts for 12.5mil timesteps (50mil frames in Atari). ϵ is decayed linearly from 1 to 0.01 over 1mil timesteps. We train on a batch of 32 after every 4th timestep with a replay buffer of size 1mil. The target network is updated every 8000 timesteps.

For all methods we consider $\beta_{mmc} \in \{0.005, 0.01, 0.025\}$.

For DQN + PC we consider $\beta \in \{0.01, 0.1, 1\}$.

For OPIQ and its ablations we set $M = 2$. We consider: $C_{\text{action}} \in \{0.1, 1\}, C_{\text{bootstrap}} \in \{0.01, 0.1\}$, $\beta \in \{0.01, 0.1\}$.

For the RND bonus we use the same architectures as in [92] (target network is smaller than the learned predictor network) except we use ReLU non-linearity. The output is the same of size 512. We scale the squared error by $\beta_{rnd} \in \{0.001, 0.01, 0.1, 1\}$:

For BSP we use the same architecture as in [112].

We use $K = 10$ different bootstrapped DQN heads, and sweep over $\beta_{bsp} \in \{0.1, 1, 10, 100\}$.

For all methods we run 4 independent runs across the cross-product of all relevant parameters considered. We then sort them by the median maximum episodic reward (largest area underneath the line) and report the median, lower and upper quartiles.

The best hyperparameters we found were:

DQN + PC: $\beta = 0.01, \beta_{mmc} = 0.01$.

DQN + RND: $\beta_{rnd} = 0.1, \beta_{mmc} = 0.01$.

BSP: $\beta_{bsp} = 0.1, \beta_{mmc} = 0.025$.

OPIQ: M= 2, $C_{\text{action}} = 0.1$, $C_{\text{bootstrap}} = 0.01$, $\beta_{mmc} = 0.01$.

OPIQ without Optimistic Bootstrapping: M= 2, $C_{\text{action}} = 0.1, \beta_{mmc} = 0.005$.

A.1.2 Baselines training details

DQN + RND: We do a single gradient descent step on a minibatch of the 32 most recently visited states. We also recompute the intrinsic rewards when sampling minibatches to train the DQN. The intrinsic reward used for a state s, is the squared error between the predictor network and the target network $\beta_{rnd}||\text{predictor}(s) - \text{target}(s)||_2^2$.

DQN + DORA: We train the E-values network using n-step SARSA (same n as the DQN) with $\gamma_E = 0.99$. We maintain a replay buffer of size (batch size $* 4$) and sample batch size elements to train every timestep. The intrinsic reward we use is $\frac{\beta_{dora}}{\sqrt{-\log E(s,a)}}$.

DQN + DORA OA: We train the DQN + DORA agent described above and additionally augment the Q-values used for action selection with $\frac{\beta_{dora_action}}{\sqrt{-\log E(s,a)}}$.

BSP: We train each Bootstrapped DQN head on all of the data from the replay buffer (as is done in [116, 112]. We normalise the gradients of the shared part of the network by $1/K$, where K is the number of heads. The output of each head is $Q_k + \beta_{bsp} p_k$, where p_k is a randomly initialised network (of the same architecture as Q_k) which is kept fixed throughout training. β_{bsp} is a hyperparameter governing the scale of the prior regularisation.

B

Value Function Factorisation

B.1 Two Step Game

B.1.1 Architecture and Training

The architecture of all agent networks is a DQN with a single hidden layer comprised of 64 units with a ReLU nonlinearity. Each agent performs independent ϵ greedy action selection, with $\epsilon = 1$. We set $\gamma = 0.99$. The replay buffer consists of the last 500 episodes, from which we uniformly sample a batch of size 32 for training. The target network is updated every 100 episodes. The learning rate for RMSprop is set to 5×10^{-4}. We train for $10k$ timesteps. The size of the mixing network is 8 units. All agent networks share parameters, thus the agent id is concatenated onto each agent's observations. We do not pass the last action taken to the agent as input. Each agent receives the full state as input.

Each state is one-hot encoded. The starting state for the first timestep is State 1. If Agent 1 takes Action A, it transitions to State 2 (whose payoff matrix is all 7s). If agent 1 takes Action B in the first timestep, it transitions to State 3.

B.1.2 Learned Value Functions

The learned value functions for the different methods on the Two Step Game are shown in Tables B.1 and B.2.

(a)

	State 1		State 2A		State 2B	
	A	*B*	*A*	*B*	*A*	*B*
A	6.94	6.94	6.99	7.02	-1.87	2.31
B	6.35	6.36	6.99	7.02	2.33	6.51

(b)

	A	*B*	*A*	*B*	*A*	*B*
A	6.93	6.93	7.00	7.00	0.00	1.00
B	7.92	7.92	7.00	7.00	1.00	8.00

(c)

	A	*B*	*A*	*B*	*A*	*B*
A	6.94	6.93	7.03	7.02	0.00	1.01
B	7.93	7.92	7.02	7.01	1.01	8.02

(d)

	A	*B*	*A*	*B*	*A*	*B*
A	6.98	6.97	7.01	7.02	-1.39	2.57
B	6.37	6.36	7.02	7.04	2.67	6.58

(e)

	A	*B*	*A*	*B*	*A*	*B*
A	6.95	6.99	6.99	7.06	-1.21	2.73
B	6.18	6.22	7.01	7.09	2.46	6.40

Table B.1: Q_{tot} on the 2 step game for (a) VDN, (b) QMIX, (c) QMIX-NS, (d) QMIX-Lin and (e) VDN-S

Agent 1

A	*B*	*A*	*B*	*A*	*B*
6.96	4.47	6.98	7.00	0.50	4.50

Agent 2

A	*B*	*A*	*B*	*A*	*B*
5.70	5.78	7.00	7.02	0.50	4.47

Table B.2: Q_a for IQL on the 2 step game

B.1.3 Results

Figure B.1 shows the loss for the different methods. Table B.3 shows the final testing reward for each method.

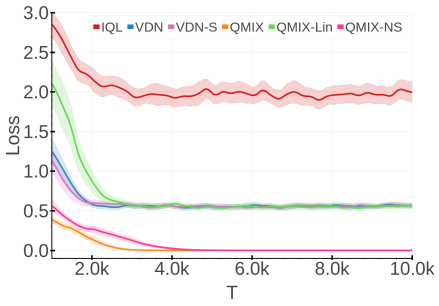


Figure B.1: Loss for all six methods on the Two Step Game. The mean and 95% confidence interval is shown across 30 independent runs.

IQL	VDN	VDN-S	QMIX	QMIX-Lin	QMIX-NS
7	7	7	8	7	8

Table B.3: The final Test Reward achieved.

B.2 Random Matrix Games

B.2.1 Architecture and Training

The architecture of all agent networks is a DQN with 2 hidden layers comprised of 64 units with a ReLU nonlinearity. Each agent performs independent ϵ greedy action selection, with $\epsilon = 1$. The replay buffer consists of the last 500 episodes, from which we uniformly sample a batch of size 32 for training. The target network is updated every 100 episodes. The learning rate for RMSprop is set to 5×10^{-4}. We train for $20k$ timesteps. The mixing network is identical to the SMAC experiments. All agent networks share parameters, thus the agent id is concatenated onto each agent's observations. We do not pass the last action taken to the agent as input. Each agent receives the full state (a vector of 1s with dimension 5) as input.

B.3 SMAC

B.3.1 Scenarios

Perhaps the simplest scenarios are **symmetric** battle scenarios, where the two armies are composed of the same units. Such challenges are particularly interesting when some of the units are extremely effective against others (this is known as

countering), for example, by dealing bonus damage to a particular armour type. In such a setting, allied agents must deduce this property of the game and design an intelligent strategy to protect teammates vulnerable to certain enemy attacks.

SMAC also includes more challenging scenarios, for example, in which the enemy army outnumbers the allied army by one or more units. In such **asymmetric** scenarios it is essential to consider the health of enemy units in order to effectively target the desired opponent.

Lastly, SMAC offers a set of interesting **micro-trick** challenges that require a higher-level of cooperation and a specific micro trick to defeat the enemy. An example of such scenario is the `corridor` scenario (Figure 4.6d). Here, six friendly Zealots face 24 enemy Zerglings, which requires agents to make effective use of the terrain features. Specifically, agents should collectively wall off the choke point (the narrow region of the map) to block enemy attacks from different directions. The `3s_vs_5z` scenario features three allied Stalkers against five enemy Zealots (Figure 4.1a). Since Zealots counter Stalkers, the only winning strategy for the allied units is to kite the enemy around the map and kill them one after another. Some of the micro-trick challenges are inspired by *StarCraft Master* challenge missions released by Blizzard [273].

B.3.2 Environment Setting

SMAC makes use of the StarCraft II Learning Environment (*SC2LE*) [151] to communicate with the StarCraft II engine. SC2LE provides full control of the game by making it possible to send commands and receive observations from the game. However, SMAC is conceptually different from the RL environment of SC2LE. The goal of SC2LE is to learn to play the full game of StarCraft II. This is a competitive task where a centralised RL agent receives RGB pixels as input and performs both macro and micro with the player-level control similar to human players. SMAC, on the other hand, represents a set of cooperative multi-agent micro challenges where each learning agent controls a single military unit.

SMAC uses the *raw API* of SC2LE. Raw API observations do not have any graphical component and include information about the units on the map such as health, location coordinates, etc. The raw API also allows sending action commands to individual units using their unit IDs. This setting differs from how humans play the actual game, but is convenient for designing decentralised multi-agent learning tasks.

Furthermore, to encourage agents to explore interesting micro strategies themselves, we limit the influence of the StarCraft AI on our agents. In the game of StarCraft II, whenever an idle unit is under attack, it automatically starts a reply attack towards the attacking enemy units without being explicitly ordered. We disable such automatic replies towards the enemy attacks or enemy units that are located closely by creating new units that are the exact copies of existing ones with two attributes modified: *Combat: Default Acquire Level* is set to *Passive* (default *Offensive*) and *Behaviour: Response* is set to *No Response* (default *Acquire*). These fields are only modified for allied units; enemy units are unchanged.

The sight and shooting range values might differ from the built-in *sight* or *range* attribute of some StarCraft II units. Our goal is not to master the original full StarCraft II game, but rather to benchmark MARL methods for decentralised control.

The game AI is set to level 7, *very difficult*. Our experiments, however, suggest that this setting does significantly impact the unit micromanagement of the built-in heuristics.

B.3.3 Architecture and Training

The architecture of all agent networks is a DRQN with a recurrent layer comprised of a GRU with a 64-dimensional hidden state, with a fully-connected layer before and after. Exploration is performed during training using independent ϵ-greedy action selection, where each agent a performs ϵ-greedy action selection over its own Q_a. Throughout the training, we anneal ϵ linearly from 1.0 to 0.05 over $50k$ time steps and keep it constant for the rest of the learning. We set $\gamma = 0.99$ for all experiments. The replay buffer contains the most recent 5000 episodes. We sample batches of

32 episodes uniformly from the replay buffer, and train on fully unrolled episodes, performing a single gradient descent step after every episode. The target networks are updated after every 200 training episodes. The Double Q-Learning update rule from [47] is used for all Q-Learning variants (IQL, VDN, QMIX and QTRAN).

To speed up the learning, we share the parameters of the agent networks across all agents. Because of this, a one-hot encoding of the `agent_id` is concatenated onto each agent's observations. All neural networks are trained using RMSprop[1] with learning rate 5×10^{-4}.

The mixing network consists of a single hidden layer of 32 units, utilising an ELU non-linearity. The hypernetworks consist of a feedforward network with a single hidden layer of 64 units with a ReLU non-linearity. The output of the hypernetwork is passed through an absolute function (to acheive non-negativity) and then resized into a matrix of appropriate size.

The architecture of the COMA critic is a feedforward fully-connected neural network with the first 2 layers having 128 units, followed by a final layer of $|U|$ units. We set $\lambda = 0.8$. We utilise the same ϵ-floor scheme as in [137] for the agents' policies, linearly annealing ϵ from 0.5 to 0.01 over 100k timesteps. For COMA we roll-out 8 episodes and train on those episodes. The critic is first updated, performing a gradient descent step for each timestep in the episode, starting with the final timestep. Then the agent policies are updated by a single gradient descent step on the data from all 8 episodes.

The architecture of the centralised Q for QTRAN is similar to the one used in [163]. The agent's hidden states (64 units) are concatenated with their chosen action ($|U|$ units) and passed through a feedforward network with a single hidden layer and a ReLU non-linearity to produce an agent-action embedding ($64 + |U|$ units). The network is shared across all agents. The embeddings are summed across all agents. The concatenation of the state and the sum of the embeddings is then passed into the Q network. The Q network consists of 2 hidden layers with ReLU non-linearities with 64 units each. The V network takes the state

[1] We set $\alpha = 0.99$ and do not use weight decay or momentum.

as input and consists of 2 hidden layers with ReLU non-linearities with 64 units each. We set $\lambda_{opt} = 1$ and $\lambda_{nopt_min} = 0.1$.

We also compared a COMA style architecture in which the input to Q is the state and the joint-actions encoded as one-hot vectors. For both architectural variants we also tested having 3 hidden layers. For both network sizes and architectural variants we performed a hyperparameter search over $\lambda_{opt} = 1$ and $\lambda_{nopt_min} \in \{0.1, 1, 10\}$ on all three of the scenarios we tested QTRAN on and picked the best performer out of all configurations.

VDN-S uses a state-dependent bias that has the same architecture as the final bias in QMIX's mixing network, a single hidden layer of 64 units with a ReLU non-linearity. VDN-S Bigger uses 2 hidden layers of 64 units instead.

B.3.4 Table of Results

	IQL	**COMA**	**VDN**	**QMIX**	**Heuristic**
2s_vs_1sc	100	98	100	100	0
2s3z	75	43	97	99	90
3s5z	10	1	84	97	42
1c3s5z	21	31	91	97	81
10m_vs_11m	34	7	97	97	12
2c_vs_64zg	7	0	21	58	0
bane_vs_bane	99	64	94	85	43
5m_vs_6m	49	1	70	70	0
3s_vs_5z	45	0	91	87	0
3s5z_vs_3s6z	0	0	2	2	0
6h_vs_8z	0	0	0	3	0
27m_vs_30m	0	0	0	49	0
MMM2	0	0	1	69	0
corridor	0	0	0	1	0

Table B.4: The Test Win Rate % of the various algorithms.

The table shows the final median performance (maximum median across the testing intervals within the last $250k$ of training) of the algorithms tested. The

mean test win %, across 1000 episodes, for the heuristic-based ai is also shown.

B.4 Regression Experiment

For the regression experiment we simulate a task with 4 agents. There are 10 states, of dimensionality 10, in which each dimension is uniformly sampled from $[-1, 1]$. For each of the 10 states, the agent Q-values are uniformly sampled from $[-1, 1]$. The Q_{tot} targets for each state are uniformly sampled from $[0, 10]$. The agent Q-values, targets and states are kept fixed, only the parameters of the mixing network may change.

We use a mixing embed dimension of 32, and train for a total of 2000 timesteps. At each timestep we uniformly sample a state, add it to the replay buffer of size 200, and execute a training step. Each training step consists of sampling a minibatch of 32 from the replay buffer and minimising the L_2 loss between the Q_{tot} targets and the network's outputs.

C

Weighted Value Function Factorisation

C.1 Experimental Setup

We adopt the same training setup as [232], using PyMARL to run all experiments. The architecture for QMIX is also the same as in [232].

The architecture of the mixing network for $\hat{Q}^*$ is a feed forward network with 3 hidden layers of 256 dim and ReLU non-linearities. Shown in Figure 5.1.

For the experiment in Figure 5.9 the architecture for $\hat{Q}^*$ is modified slightly. We replace the first hidden layer with a hypernetwork layer. A hypernetwork with a single hidden layer of dim 64 (with ReLU) takes the state as input and generates the weight matrix. Inspired by [210] we then take the column-wise softmax of this weight matrix, which can be viewed as an approximation of multi-head attention.

The architecture and setup for QTRAN is also the same, except we use a 3 layer feedforward network of dim $\{64, 256\}$ to match the depth of $\hat{Q}^*$.

MADDPG and MASAC's critic shares the same architecture as $\hat{Q}^*$.

MADDPG is trained using the deterministic multi-agent policy-gradient theorem, via the Gumbel-Softmax trick, as in [233, 237]. Specifically for agent a we produce $Q(s, u_a, u^{-a})$ (with target network Q) for each possible action, where u^{-a} are the actions of the other agents produced by their most recent policies. We multiply these by the agent's policy (one-hot vector since it is deterministic) and use the

Straight Through Gumbel-Softmax estimator [274] to differentiate through this with a temperature of 1.

MASAC is trained by minimising the KL divergence between each agent's policy π_a and $\exp(Q(s,\cdot,u^{-a}) - \alpha_{ent}\log\pi_a)$., Since the KL divergence is an expectation: $\mathbb{E}_{\pi_a}[\log(\frac{\pi_a}{\exp(Q(s,\cdot,u^{-a})-\alpha_{ent}\log\pi_a)}]$, we approximate it by sampling an action from π_a for each agent. These sampled actions are used for u^{-a}. For the actor's policies we use the same ϵ-greedy floor technique as in [137].

QPLEX uses the formulation of the mixing network that does not require the state to be pre-processed into individual agent parts (as is does in [210]), in order to provide a fairer comparison against the other methods. The hypernetworks feature 1 hidden layer, of 64 dimensions, similarly to QMIX. We use 10 attention heads.

C.1.1 Predator Prey

For Weighted QMIX variants (and ablations with just a weighting), we consider $\alpha \in \{0.1, 0.5\}$ and set $\alpha = 0.1$ for all variants.

For QTRAN we set $\lambda_{opt} = 1$ and consider $\lambda_{nopt} \in \{0.1, 1, 10\}$ (since only their relative weighting makes a difference), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{nopt} = 10$ and the dim of the mixing network to 64.

For MASAC we consider $\alpha_{ent} \in \{0, 0.001, 0.01, 0.1\}$ and set it to 0.001.

C.1.2 SMAC Robustness to exploration

For Weighted QMIX we consider $\alpha \in \{0.01, 0.1, 0.5, 0.75\}$ and set $\alpha = 0.75$ for CW and $\alpha = 0.5$ for OW.

For the Weighted QMIX ablations we considered $\alpha \in \{0.5, 0.75\}$ and set $\alpha = 0.75$.

For QTRAN we set $\lambda_{opt} = 1$ and consider $\lambda_{nopt} \in \{1, 10\}$ (since these 2 performed best in preliminary experiments), and the dim of the mixing network in $\{64, 256\}$. We set $\lambda_{nopt} = 10$ for *5m_vs_6m* and $\lambda nopt = 1$ for *3s5z*. The dim of the mixing network is set to 64.

For MASAC we consider $\alpha_{ent} \in \{0, 0.001, 0.01\}$ and set it to 0 for *3s5z* and 0.01 for *5m_vs_6m*.

C.1.3 SMAC Super Hard Maps

We consider $\alpha \in \{0.01, 0.1, 0.5, 0.75\}$ and set $\alpha = 0.5$ for OW-QMIX and $\alpha = 0.75$ for CW-QMIX.

For the experiment in Figure 5.9 we only considered $\alpha \in \{0.5, 0.75\}$ and set $\alpha = 0.75$ for both methods.

C.2 Ablations

In order to better understand our method, we examine 3 additional baselines:

QMIX + $\hat{Q}^*$. This ablation removes the weighting in our loss ($w = 1$), but still uses $\hat{Q}^*$to bootstrap. This ablation allows us to see if a better bootstrap estimate alone can explain the performance of WQMIX.

QMIX + CW/OW. This ablation introduces the CW/OW weightings into QMIX's loss function for Q_{tot}. We do not additionally learn $\hat{Q}^*$.

D

Importance of Weighted Value Function Factorisation

D.1 Simplified Predator Prey

We utilise PyMARL to run all experiments. The architecture for all methods is the same as in Chapter 5. ϵ is annealed from 1 to 0.05 over 50k timesteps.

D.1.1 Environment

We use the `PredatorPrey5x5-v0` environment from `ma-gym` [275]. This environment features 2 agents and 1 prey in a 5x5 toroidal gridworld. The predator is considered to be *catching* the prey, if the prey is on one of the 4 squares directly adjacent to the predator (the squares on the diagonal don't count). If only one of the predators is attempting to catch the prey, a reward of -0.5 is given. If both predators are attempting to catch the prey, then a positive reward of $+5$ is given and the environment ends. A small punishment of -0.01 is added at every timestep, and the maximum length of an episode is 100 steps. The state is the flattened representation of the 5x5 gridworld (25 dimensional) in which predator 1 is represented as 1/3, predator 2 as 2/3 and the prey as 1.

D.1.2 Weighting

The explicit weighting we use only applies to those transitions in which a punishment of -0.5 or a reward of $+5$ is given. If the punishment is given, and the overall reward for that timestep is -0.51 (due to the extra -0.01 at every timestep) then the weighting $w = 1$ at this transition.

The % of positive experiences is calculated for a given minibatch as:

$$\frac{\text{Number of transitions with} > 0 \text{ reward}}{\text{Number of transitions with reward} \neq 0.01}.$$

% of negative experiences is 1 minus this.

D.2 Contextual Matrix Games

In the contextual matrix games, there are 10 different states which are uniformly sampled for each episode. Each state has a dimensionality of 32, and each entry is uniformly sampled from $[0, 1)$ and fixed throughout training. For all of our experiments we utilise the same 10 random seeds for all methods. Each agent's observations are the state. All rewards are integers uniformly sampled from $\{-10, ..., 9\}$. A single joint-action is then uniformly sampled and selected as the optimal action, whose reward is then set to $+10$.

For all experiments we set $\alpha = 0.1$ for both the CW and OW weightings. Although better results could potentially be acheived with a smaller value, our experiments in Chapter 5 indicate that too low a value can negatively impact performance in larger sequential decision making tasks.

D.2.1 Distribution of Errors

In order to produce the values for the distribution of errors visualisations, we calculate the true (just the single step reward) and estimated Q-values for every state in the minibatch of size 32 and log them.

E

Empirical Game Theory

E.1 Experimental Setup

E.1.1 Implementation Details

α-Rank

In the computation of α-rank we set $\epsilon = 10^{-6}$ in all of our experiments.

αIG (Bins).

For this binning entropy estimator we split $[0, 1]$ into 101 equal bins of width 0.005 (implemented by rounding to the nearest second decimal place). We then estimated the entropy using a histogram.

αIG (NSB).

The NSB estimator requires an upper bound on the total number of atoms, but since we do not know the true upper bound we utilize an estimate on the total number of possible α-ranks, which we describe below. We use the open-source implementation provided in [276].

Upper bound on number of α-ranks

In the infinite-α regime there are a finite number of possible α-ranks. This is because only the ordering between relevant entries in the payoff matrix changes the

transition matrix of the Markov Chain produced in the computation of α-rank [251].

Let there be k populations each with S strategies. Then there are S^k strategies considered and so the transition matrix of the Markov Chain has S^k rows, one for each of the possible joint-strategies.

Each possible joint-strategy σ can transition to at most $k(S-1)$ other strategies $\tau \neq \sigma$. The probability of a self-transition is uniquely determined based on these probabilities.

This gives at most $2^{(k(S-1))}$ unique values for that row.

There are then $[2^{(k(S-1))}]^{S^k} = 2^{S^k(k(S-1))}$ unique transition matrices. Thus, the possible number of unique α-ranks is upper-bounded by $2^{S^k(k(S-1))}$. This bound is not tight, since there are many transition matrices with equal stationary distributions.

In our experiments with $K = 1$ this gives $2^{S(S-1)}$.

E.1.2 Baselines

ResponseGraphUCB, uses a Hoeffding Bound to construct the confidence interval: $(\mu - \sqrt{\frac{\log(2/\delta)(b-a)}{2N}}, \mu + \sqrt{\frac{\log(2/\delta)(b-a)}{2N}})$. Where δ is the confidence hyperparameter of the algorithm, b is the maximum value an entry can take, a is the minimum value, and N is the number of times a value has been seen for an entry. For all experiments we swept over $\delta \in \{0.4, 0.3, 0.2, 0.1, 0.05, 0.01, 0.001\}$, and the final value is selected by considering the area under the curve for $1 - P(f(\bar{M}) = r_{GT})$.

Uniform. The entry to sample if picked uniformly from all possible entries.

Payoff. The entry which maximises the information gain between its sample and the payoff distribution is chosen. For an isotropic Gaussian this is equivalent to picking the entry which the lowest count, which results in systematic sampling of each entry. For a non isotropic Gaussian the same procedure as [261] is used.

E.1.3 Graphs

$1 - P(f(\bar{M}) = r_{GT})$. At each timestep we compute the α-rank of the mean payoff matrix. Equality is determined if $|f(\bar{M}) = r_{GT})|_1 < 0.01$. The choice of 0.01 is largely arbitrary, we did not find the results to be sensitive to this.

$1 - P(r_{GT})$. 100 times during training (evenly spaced), we sample 2000 samples from the current belief distribution over α-ranks. $P(r_{GT})$ is determined from these 2000 samples (which are aggregated by rounding each value to the nearest $3d.p.$) by counting the number of sampled α-ranks r such that $|r - r_{GT}| < 0.01$.

$1 - P(r_*)$, is determined similarly to $1 - P(r_{GT})$, except we use the 2000 samples to calculate the mode.

For ResponseGraphUCB, we construct a distribution over the payoff entries as being uniform over the confidence intervals.

E.1.4 Environments

E.1.5 2 Good, 2 Bad

Observations are sampled from $Ber(x)$, where x is the value in the payoff matrix.

αIG(Bins), αIG(NSB), αWass. Prior used is $\mathcal{N}(\mu_0, \sigma_0^2)$, with aleatoric noise σ_A^2. $\mu_0 = 0.5$. Swept over $\sigma_0^2 \in \{0.5, 1\}, \sigma_A^2 \in \{0.25, 0.5, 1\}$. 20 samples are used to approximate the expectation, $N_e = 20$. 1000 samples are drawn from the belief distribution(s) to approximate the quantities inside the expectation, $N_b = 1000$. We set $N_r = 10$.

For all 3 methods we set $\sigma_0^2 = 1$. For αIG (Bins) and αIG (NSB) we set $\sigma_A^2 = 0.5$, and for αWass we set $\sigma_A^2 = 0.25$.

ResponseGraphUCB. We set $\delta = 0.4$. Maximum value is 1, minimum value is 0.

E.1.6 3 Good, 5 Bad

Observations are sampled from $Ber(x)$, where x is the value in the payoff matrix.

αIG(Bins), αIG(NSB), αWass. Prior used is $\mathcal{N}(\mu_0, \sigma_0^2)$, with aleatoric noise σ_A^2. $\mu_0 = 0.5$. Swept over $\sigma_0^2 \in \{0.5, 1\}, \sigma_A^2 \in \{0.25, 0.5, 1\}$. $N_e = 10$. $N_b = 500$. $N_r = 500$.

For all 3 methods we set $\sigma_0^2 = 1$. For αIG (Bins) and αIG (NSB) we set $\sigma_A^2 = 0.5$, and for αWass we set $\sigma_A^2 = 0.25$.

ResponseGraphUCB. We set $\delta = 0.05$.

E.1.7 4x4 Gaussian

To match the games considered in our theoretical analysis, Observations are sampled from $\mathcal{N}(x, 1)$ and then clipped to be within 1 of x, where x is the value of the entry in the payoff matrix. The values of x are uniformly drawn from $[0, 1)$.

αIG(Bins), αIG(NSB), αWass. Prior used is $\mathcal{N}(\mu_0, \sigma_0^2)$, with aleatoric noise σ_A^2. $\mu_0 = 0.5$. Swept over $\sigma_0^2 \in \{0.5, 1\}, \sigma_A^2 \in \{0.5, 1\}$. $N_e = 10$. $N_b = 500$. $N_r = 100$. For αIG(Bins) we set $\sigma_0^2 = 1$ and $\sigma_A^2 = 1$. For αIG(NSB) we set $\sigma_0^2 = 0.5$ and $\sigma_A^2 = 0.5$. For αWass we set $\sigma_0^2 = 1$ and $\sigma_A^2 = 0.5$.

ResponseGraphUCB. We set $\delta = 0.3$. Maximum value is 2, minimum value is -1.

E.1.8 3 Good, 5 Bad Incorporating Prior Knowledge

αIG(Bins), αWass. Aleatoric noise σ_A^2 is set to 0.5 and the mean $\mu = 0.5$. $\mu_0 = 0.5$. Swept over $\sigma_A^2 \in \{0.25, 0.5, 1\}$. $N_e = 10$. $N_b = 500$. $N_r = 100$.

The kernel K we use for the GP is specified as follows:
In order to encode the prior knowledge that elements within a block (except the top left block) are equal, we partition the payoff matrix. A strategy $\sigma = (x, y)$ is a member of block b_1 if $1 \leq x \leq 3$ and $1 \leq y \leq 3$, if it is a payoff between a Good agent and another Good agent. Block b_2 if $1 \leq x \leq 3$ and $y > 3$, a payoff between a Good agent and a Bad agent. Block b_3 if $1 \leq y \leq 3$ and $x > 3$, a payoff between a Bad agent and a Good agent. Block b_4 otherwise, if it is a payoff between a Bad agent and a Bad agent.

The kernel k encoding block-wise equality is then defined as:
$k(b_i, b_j) = 1$ for all $i \neq j$.
$k(b_i, b_i) = 0$ for $i \in \{2, 3, 4\}$.
$k(b_1, b_1) = 1$.

To additionally encode anti-symmetry (about the mean $\mu = 0.5$) we then produce a new kernel k' from k as follows:
For a strategy $\sigma = (x, y)$, define $\sigma^t := (y, x)$ the transpose of σ. We wish to encode

that $M(\sigma) = 1 - M(\sigma^t)$.

$k'(\sigma, \tau) = k(\sigma, \tau) + k(\sigma^t, \tau^t) - k(\sigma, \tau^t) - k(\sigma^t, \tau)$.

The finished kernel K is then defined as $K := (k')^T(k')$ to ensure it is positive definite. The entries are then divided by 500 to ensure a suitable magnitude for the variance.

ResponseGraphUCB. We set $\delta = 0.05$.

In order to incorporate the same modelling assumptions into RG-UCB, for every real sample we receive from the payoff matrix, we pretend to receive an appropriate sample for the relevant entries.

After receiving a payoff $p \in \{0, 1\}$ for strategy σ, we then pretend to receive:

$1 - p$ for σ^t to encode anti-symmetry.

p for all τ in the same block as σ (except the top-left block), where blocks are defined the same as for the kernel k specified above.

References

[1] Declan Butler. "Tomorrow's world: technological change is accelerating today at an unprecedented speed and could create a world we can barely begin to imagine". In: *Nature* 530.7591 (2016), pp. 398–402.

[2] Michio Kaku. *Physics of the future: The inventions that will transform our lives*. Penguin UK, 2012.

[3] Joseph Johnson. *Global digital population as of January 2021*. 2021.

[4] *Top500 SuperComputing List*. `https://top500.org/lists/top500/`. Accessed: 2021-08-27.

[5] Robert R Schaller. "Moore's law: past, present and future". In: *IEEE spectrum* 34.6 (1997), pp. 52–59.

[6] Chris Edwards. "Moore's Law: what comes next?" In: *Commun. ACM* (2021).

[7] Norman P. Jouppi et al. "In-Datacenter Performance Analysis of a Tensor Processing Unit". In: *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*. 2017.

[8] Michael Sipser. "Introduction to the Theory of Computation". In: *ACM Sigact News* 27.1 (1996), pp. 27–29.

[9] Constantinos Daskalakis, Paul W Goldberg, and Christos H Papadimitriou. "The complexity of computing a Nash equilibrium". In: *SIAM Journal on Computing* 39.1 (2009), pp. 195–259.

[10] Aviezri S Fraenkel and David Lichtenstein. "Computing a perfect strategy for n× n chess requires time exponential in n". In: *International Colloquium on Automata, Languages, and Programming*. Springer. 1981, pp. 278–293.

[11] Gilles Audemard and Laurent Simon. "Predicting Learnt Clauses Quality in Modern SAT Solvers". In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. 2009.

[12] Seymour A Papert. "The summer vision project". In: (1966).

[13] Stuart Russell and Peter Norvig. "Artificial intelligence: a modern approach". In: (2002).

[14] Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. "Deep blue". In: *Artificial intelligence* 134.1-2 (2002), pp. 57–83.

[15] Drew McDermott. "How intelligent is deep blue?" In: *New York Times (May)* 14 (1997).

[16] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), pp. 484–489.

[17] Tom B. Brown et al. “Language Models are Few-Shot Learners”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[18] Azalia Mirhoseini et al. “A graph placement methodology for fast chip design”. In: *Nature* 594.7862 (2021), pp. 207–212.

[19] Christopher M Bishop. “Pattern recognition”. In: *Machine learning* 128.9 (2006).

[20] J Hirschberg. “Every time I fire a linguist, my performance goes up, and other myths of the statistical natural language processing revolution. Invited talk”. In: *Fifteenth National Conference on Artificial Intelligence (AAAI-98)*. 1998.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*. 2012.

[22] David Silver et al. “Reward is enough”. In: *Artificial Intelligence* (2021), p. 103535.

[23] Gerald Tesauro et al. “Temporal difference learning and TD-Gammon”. In: *Communications of the ACM* 38.3 (1995), pp. 58–68.

[24] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[25] Christopher Berner et al. “Dota 2 with Large Scale Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1912.06680* (2019).

[26] Oriol Vinyals et al. “Grandmaster level in StarCraft II using multi-agent reinforcement learning”. In: *Nature* 575.7782 (2019), pp. 350–354.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] Chi Jin et al. “Is $\boldsymbol{Q}$-learning Provably Efficient?” In: *Advances in Neural Information Processing Systems*. 2018, pp. 4863–4873.

[29] Ronen I Brafman and Moshe Tennenholtz. “R-max-a general polynomial time algorithm for near-optimal reinforcement learning”. In: *Journal of Machine Learning Research* 3.Oct (2002), pp. 213–231.

[30] Alexander L. Strehl et al. “PAC model-free reinforcement learning”. In: *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006*. 2006.

[31] Johannes Heinrich and David Silver. “Deep Reinforcement Learning from Self-Play in Imperfect-Information Games”. In: *arXiv preprint arXiv:1603.01121* (2016).

[32] Marc Lanctot et al. “A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017.

[33] Shayegan Omidshafiei et al. "α-rank: Multi-agent evaluation by evolution". In: *Scientific reports* 9.1 (2019), pp. 1–29.

[34] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. 1994.

[35] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artificial intelligence* 101.1-2 (1998), pp. 99–134.

[36] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.

[37] Francisco S Melo. "Convergence of Q-learning: A simple proof". In: *Institute Of Systems and Robotics, Tech. Rep* (2001), pp. 1–4.

[38] Richard S. Sutton et al. "Policy Gradient Methods for Reinforcement Learning with Function Approximation". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999.

[39] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[40] Hajime Kimura, Shigenobu Kobayashi, et al. "An analysis of actor-critic algorithms using eligibility traces: reinforcement learning with imperfect value functions". In: *Journal of Japanese Society for Artificial Intelligence* 15.2 (2000), pp. 267–275.

[41] Vijay R. Konda and John N. Tsitsiklis. "Actor-Critic Algorithms". In: *Advances in Neural Information Processing Systems 12, [NIPS Conference, Denver, Colorado, USA, November 29 - December 4, 1999]*. 1999.

[42] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. 2016.

[43] Joshua Achiam. "Spinning Up in Deep Reinforcement Learning". In: (2018).

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

[45] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[46] Matthew J. Hausknecht and Peter Stone. "Deep Recurrent Q-Learning for Partially Observable MDPs". In: *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*. 2015.

[47] Hado van Hasselt, Arthur Guez, and David Silver. "Deep Reinforcement Learning with Double Q-Learning". In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. 2016.

[48] Alexander L Strehl and Michael L Littman. "An analysis of model-based interval estimation for Markov decision processes". In: *Journal of Computer and System Sciences* 74.8 (2008), pp. 1309–1331.

[49] Mohammad Gheshlaghi Azar, Ian Osband, and Rémi Munos. "Minimax Regret Bounds for Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[50] Danijar Hafner et al. "Learning Latent Dynamics for Planning from Pixels". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[51] Volodymyr Mnih et al. "Asynchronous Methods for Deep Reinforcement Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016.

[52] Matteo Hessel et al. "Rainbow: Combining Improvements in Deep Reinforcement Learning". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 2018.

[53] Marc G. Bellemare et al. "Unifying Count-Based Exploration and Intrinsic Motivation". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[54] Georg Ostrovski et al. "Count-Based Exploration with Neural Density Models". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[55] Pierre-Yves Oudeyer and Frédéric Kaplan. "What is intrinsic motivation? A typology of computational approaches". In: *Frontiers Neurorobotics* (2007).

[56] Haoran Tang et al. "# exploration: A study of count-based exploration for deep reinforcement learning". In: *31st Conference on Neural Information Processing Systems (NIPS)*. Vol. 30. 2017, pp. 1–18.

[57] Marlos C. Machado, Marc G. Bellemare, and Michael Bowling. "Count-Based Exploration with the Successor Representation". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. 2020.

[58] Yuanhao Wang et al. "Q-learning with UCB Exploration is Sample Efficient for Infinite-Horizon MDP". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[59] Sudhir K Goel and Dennis M Rodriguez. "A note on evaluating limits using Riemann sums". In: *Mathematics Magazine* 60.4 (1987), pp. 225–228.

[60] Li Fan et al. "Summary cache: a scalable wide-area web cache sharing protocol". In: *IEEE/ACM Trans. Netw.* (2000).

[61] Adrien Ecoffet et al. "Go-Explore: a New Approach for Hard-Exploration Problems". In: *arXiv preprint arXiv:1901.10995* (2019).

[62] Adrien Ali Taiga, Aaron C. Courville, and Marc G. Bellemare. "Approximate Exploration through State Abstraction". In: *arXiv preprint arXiv:1808.09819* (2018).

[63] Susan Amin et al. "A Survey of Exploration Methods in Reinforcement Learning". In: *arXiv preprint arXiv:2109.00157* (2021).

[64] Sebastian B Thrun. "Efficient Exploration In Reinforcement Learning". In: (1992).

[65] Thomas Jaksch, Ronald Ortner, and Peter Auer. "Near-optimal Regret Bounds for Reinforcement Learning." In: *Journal of Machine Learning Research* 11.4 (2010).

[66] Ian Osband, Daniel Russo, and Benjamin Van Roy. "(More) Efficient Reinforcement Learning via Posterior Sampling". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*. 2013.

[67] Eric Wiewiora. "Potential-based shaping and Q-value initialization are equivalent". In: *Journal of Artificial Intelligence Research* 19 (2003), pp. 205–208.

[68] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. "Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping". In: *Proceedings of the Sixteenth International Conference on Machine Learning (ICML 1999), Bled, Slovenia, June 27 - 30, 1999*. 1999.

[69] Michael O'Gordon Duff. *Optimal Learning: Computational procedures for Bayes-adaptive Markov decision processes*. University of Massachusetts Amherst, 2002.

[70] John C Gittins. "Bandit processes and dynamic allocation indices". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 41.2 (1979), pp. 148–164.

[71] J. Zico Kolter and Andrew Y. Ng. "Near-Bayesian exploration in polynomial time". In: *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*. 2009.

[72] Malcolm J. A. Strens. "A Bayesian Framework for Reinforcement Learning". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. 2000.

[73] Ian Osband and Benjamin Van Roy. "Why is Posterior Sampling Better than Optimism for Reinforcement Learning?" In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[74] Shipra Agrawal and Randy Jia. "Optimistic posterior sampling for reinforcement learning: worst-case regret bounds". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017.

[75] Mohammad Ghavamzadeh et al. "Bayesian Reinforcement Learning: A Survey". In: *Foundations and Trends in Machine Learning* 8.5-6 (2015), pp. 359–483.

[76] Richard Dearden, Nir Friedman, and Stuart J. Russell. "Bayesian Q-Learning". In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98, July 26-30, 1998, Madison, Wisconsin, USA*. 1998.

[77] Ian Osband et al. "Deep Exploration via Randomized Value Functions". In: *Journal of Machine Learning Research* 20 (2019), pp. 1–62.

[78] Daniel Russo. "Worst-Case Regret Bounds for Exploration via Randomized Value Functions". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[79] Andrea Zanette et al. "Frequentist Regret Bounds for Randomized Least-Squares Value Iteration". In: *The 23rd International Conference on Artificial Intelligence and Statistics, AISTATS 2020, 26-28 August 2020, Online [Palermo, Sicily, Italy]*. 2020.

[80] Lin Yang and Mengdi Wang. "Reinforcement Learning in Feature Space: Matrix Bandit, Kernels, and Regret Bound". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 2020.

[81] Chi Jin et al. "Provably efficient reinforcement learning with linear function approximation". In: *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*. 2020.

[82] Yining Wang et al. "Optimism in Reinforcement Learning with Generalized Linear Function Approximation". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[83] Qi Cai et al. "Provably Efficient Exploration in Policy Optimization". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 2020.

[84] Nan Jiang et al. "Contextual Decision Processes with low Bellman rank are PAC-Learnable". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[85] Kefan Dong et al. "Root-n-Regret for Learning in Markov Decision Processes with Function Approximation and Low Bellman Rank". In: *Conference on Learning Theory, COLT 2020, 9-12 July 2020, Virtual Event [Graz, Austria]*. 2020.

[86] Dipendra Misra et al. "Provable Rich Observation Reinforcement Learning with Combinatorial Latent States". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[87] Jason Pazis and Ronald Parr. "PAC Optimal Exploration in Continuous Space Markov Decision Processes". In: *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA*. 2013.

[88] Alexander Pritzel et al. "Neural Episodic Control". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[89] Jürgen Schmidhuber. "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)". In: *IEEE Transactions on Autonomous Mental Development* 2.3 (2010), pp. 230–247.

[90] Mikhail Frank et al. "Curiosity driven reinforcement learning for motion planning on humanoids". In: *Frontiers Neurorobotics* (2013).

[91] Andrew Barto, Marco Mirolli, and Gianluca Baldassarre. "Novelty or surprise?" In: *Frontiers in psychology* 4 (2013).

[92] Yuri Burda et al. "Exploration by random network distillation". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[93] Rein Houthooft et al. "VIME: Variational Information Maximizing Exploration". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[94] Joshua Achiam and Shankar Sastry. "Surprise-Based Intrinsic Motivation for Deep Reinforcement Learning". In: *arXiv preprint arXiv:1703.01732* (2017).

[95] Bradly C. Stadie, Sergey Levine, and Pieter Abbeel. "Incentivizing Exploration In Reinforcement Learning With Deep Predictive Models". In: *arXiv preprint arXiv:1507.00814* (2015).

[96] Hyoungseok Kim et al. "EMI: Exploration with Mutual Information Maximizing State and Action Embeddings". In: *arXiv preprint arXiv:1810.01176* (2018).

[97] Deepak Pathak et al. "Curiosity-driven Exploration by Self-supervised Prediction". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[98] Brendan O'Donoghue et al. "The Uncertainty Bellman Equation and Exploration". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018.

[99] Takahisa Imagawa, Takuya Hiraoka, and Yoshimasa Tsuruoka. "Optimistic Proximal Policy Optimization". In: *arXiv preprint arXiv:1906.11075* (2019).

[100] Nikolay Savinov et al. "Episodic Curiosity through Reachability". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[101] Aäron van den Oord et al. "Conditional Image Generation with PixelCNN Decoders". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[102] Justin Fu, John D. Co-Reyes, and Sergey Levine. "EX2: Exploration with Exemplar Models for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017.

[103] Jongwook Choi et al. "Contingency-Aware Exploration in Reinforcement Learning". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[104] Zachary C. Lipton et al. "BBQ-Networks: Efficient Exploration in Deep Reinforcement Learning for Task-Oriented Dialogue Systems". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 2018.

[105] Charles Blundell et al. "Weight uncertainty in neural network". In: *International Conference on Machine Learning*. PMLR. 2015, pp. 1613–1622.

[106] Yunhao Tang and Alp Kucukelbir. "Variational Deep Q Network". In: *arXiv preprint arXiv:1711.11225* (2017).

[107] Ahmed Touati et al. "Randomized Value Functions via Multiplicative Normalizing Flows". In: *Proceedings of the Thirty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI 2019, Tel Aviv, Israel, July 22-25, 2019*. 2019.

[108] Trevor Barron, Oliver Obst, and Heni Ben Amor. "Information Maximizing Exploration with a Latent Dynamics Model". In: *arXiv preprint arXiv:1804.01238* (2018).

[109] Kamyar Azizzadenesheli, Emma Brunskill, and Animashree Anandkumar. "Efficient Exploration Through Bayesian Deep Q-Networks". In: *2018 Information Theory and Applications Workshop, ITA 2018, San Diego, CA, USA, February 11-16, 2018*. 2018.

[110] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

[111] Yarin Gal and Zoubin Ghahramani. "Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning". In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. 2016.

[112] Ian Osband, John Aslanides, and Albin Cassirer. "Randomized Prior Functions for Deep Reinforcement Learning". In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 2018.

[113] Matthias Plappert et al. "Parameter Space Noise for Exploration". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018.

[114] Meire Fortunato et al. "Noisy Networks For Exploration". In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018.

[115] Richard Y. Chen et al. "UCB and InfoGain Exploration via *boldsymbol{Q}*-Ensembles". In: *arXiv preprint arXiv:1706.01502* (2017).

[116] Ian Osband et al. "Deep Exploration via Bootstrapped DQN". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[117] Vikranth Dwaracherla et al. “Hypermodels for Exploration”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[118] Daniel Russo and Benjamin Van Roy. “Learning to Optimize via Information-Directed Sampling”. In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. 2014.

[119] David Janz et al. “Successor Uncertainties: Exploration and Uncertainty in Temporal Difference Learning”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[120] Ian Osband et al. “Deep Exploration via Bootstrapped DQN”. In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[121] Sebastian Flennerhag et al. “Temporal Difference Uncertainties as a Signal for Exploration”. In: *arXiv preprint arXiv:2010.02255* (2020).

[122] Marlos C. Machado, Sriram Srinivasan, and Michael H. Bowling. “Domain-Independent Optimistic Initialization for Reinforcement Learning”. In: *Learning for General Competency in Video Games, Papers from the 2015 AAAI Workshop, Austin, Texas, USA, January 26, 2015*. 2015.

[123] Lior Fox, Leshem Choshen, and Yonatan Loewenstein. “DORA The Explorer: Directed Outreaching Reinforcement Action-Selection”. In: *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. 2018.

[124] Min-hwan Oh and Garud Iyengar. “Directed Exploration in PAC Model-Free Reinforcement Learning”. In: *arXiv preprint arXiv:1808.10552* (2018).

[125] Adrien Ali Taiga et al. “On bonus based exploration methods in the arcade learning environment”. In: *International Conference on Learning Representations*. 2019.

[126] Will Dabney, Georg Ostrovski, and André Barreto. “Temporally-Extended {\epsilon}-Greedy Exploration”. In: *arXiv preprint arXiv:2006.01782* (2020).

[127] Adrià Puigdomènech Badia et al. “Agent57: Outperforming the atari human benchmark”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 507–517.

[128] Adrien Ecoffet et al. “First return, then explore”. In: *Nature* 590.7847 (2021), pp. 580–586.

[129] Pranav Shyam, Wojciech Jaskowski, and Faustino Gomez. “Model-Based Active Exploration”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[130] Marc G Bellemare et al. “The arcade learning environment: An evaluation platform for general agents”. In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 253–279.

[131] Marlos C Machado et al. “Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents”. In: *Journal of Artificial Intelligence Research* 61 (2018), pp. 523–562.

[132] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. “Guided Deep Reinforcement Learning for Swarm Systems”. In: *AAMAS 2017 Autonomous Robots and Multirobot Systems (ARMS) Workshop*. 2017.

[133] Yongcan Cao et al. “An Overview of Recent Progress in the Study of Distributed Multi-Agent Coordination”. In: *IEEE Trans. Ind. Informatics* (2013).

[134] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. “Optimal and approximate Q-value functions for decentralized POMDPs”. In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 289–353.

[135] Landon Kraemer and Bikramjit Banerjee. “Multi-agent reinforcement learning as a rehearsal for decentralized planning”. In: *Neurocomputing* (2016).

[136] Emilio Jorge, Mikael Kågebäck, and Emil Gustavsson. “Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence”. In: *arXiv preprint arXiv:1611.03218* (2016).

[137] Jakob N. Foerster et al. “Counterfactual Multi-Agent Policy Gradients”. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 2018.

[138] Ming Tan. “Multi-agent reinforcement learning: Independent vs. cooperative agents”. In: *Proceedings of the Tenth International Conference on Machine Learning*. 1993, pp. 330–337.

[139] Jayesh K. Gupta, Maxim Egorov, and Mykel J. Kochenderfer. “Cooperative Multi-agent Control Using Deep Reinforcement Learning”. In: *Autonomous Agents and Multiagent Systems - AAMAS 2017 Workshops, Best Papers, S textasciitilde ao Paulo, Brazil, May 8-12, 2017, Revised Selected Papers*. 2017.

[140] Peter Sunehag et al. “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 2018, Stockholm, Sweden, July 10-15, 2018*. 2018.

[141] David Ha, Andrew M. Dai, and Quoc V. Le. “HyperNetworks”. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[142] Matthias Plappert et al. “Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research”. In: *arXiv preprint arXiv:1802.09464* (2018).

[143] Johannes Heinrich and David Silver. “Deep Reinforcement Learning from Self-Play in Imperfect-Information Games”. In: *arXiv preprint arXiv:1603.01121* (2016).

[144] Ardi Tampuu et al. “Multiagent Cooperation and Competition with Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1511.08779* (2015).

[145] Peter Stone et al. “Keepaway Soccer: From Machine Learning Testbed to Benchmark”. In: *RoboCup 2005: Robot Soccer World Cup IX*. 2005.

[146] Ryan Lowe et al. "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017.

[147] Joel Z. Leibo et al. "Multi-agent Reinforcement Learning in Sequential Social Dilemmas". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, S
textasciitilde ao Paulo, Brazil, May 8-12, 2017*. 2017.

[148] Minne Li et al. "Efficient Ridesharing Order Dispatching with Mean Field Multi-Agent Reinforcement Learning". In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*. 2019.

[149] Lianmin Zheng et al. "MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. 2018.

[150] Nolan Bard et al. "The hanabi challenge: A new frontier for ai research". In: *Artificial Intelligence* 280 (2020), p. 103216.

[151] Oriol Vinyals et al. "StarCraft II: A New Challenge for Reinforcement Learning". In: *arXiv preprint arXiv:1708.04782* (2017).

[152] Neil C. Rabinowitz et al. "Machine Theory of Mind". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018.

[153] Tabish Rashid et al. "QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018.

[154] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. 2016.

[155] Carlos Guestrin, Daphne Koller, and Ronald Parr. "Multiagent Planning with Factored MDPs". In: *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*. 2001.

[156] Charles Dugas et al. "Incorporating Functional Knowledge in Neural Networks." In: *Journal of Machine Learning Research* 10.6 (2009).

[157] Allan Pinkus. "Approximation theory of the MLP model in neural networks". In: *Acta numerica* 8 (1999), pp. 143–195.

[158] Anuj Mahajan et al. "MAVEN: Multi-Agent Variational Exploration". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[159] Adam Stooke and Pieter Abbeel. "rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch". In: *arXiv preprint arXiv:1909.01500* (2019).

[160] Ashley Hill et al. *Stable Baselines*. `https://github.com/hill-a/stable-baselines`. 2018.

[161] Matt Hoffman et al. "Acme: A Research Framework for Distributed Reinforcement Learning". In: *arXiv preprint arXiv:2006.00979* (2020).

[162] Eric Liang et al. "RLlib: Abstractions for Distributed Reinforcement Learning". In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018.

[163] Kyunghwan Son et al. "QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[164] Erfu Yang and Dongbing Gu. *Multiagent reinforcement learning for multi-robot systems: A survey*. Tech. rep. 2004.

[165] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A Comprehensive Survey of Multiagent Reinforcement Learning". In: *IEEE Trans. Syst. Man Cybern. Part C* (2008).

[166] Peng Peng et al. "Multiagent Bidirectionally-Coordinated Nets: Emergence of Human-level Coordination in Learning to Play StarCraft Combat Games". In: *arXiv preprint arXiv:1703.10069* (2017).

[167] Christian Schröder de Witt et al. "Multi-Agent Common Knowledge Reinforcement Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[168] Christopher Watkins. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

[169] Jakob N. Foerster et al. "Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

[170] Martin Lauer and Martin A. Riedmiller. "An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems". In: *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*. 2000.

[171] Laëtitia Matignon, Guillaume Laurent, and Nadine Le Fort-Piat. "Hysteretic Q-Learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS'07*. 2007, pp. 64–69.

[172] Shayegan Omidshafiei et al. "Deep Decentralized Multi-task Multi-Agent RL under Partial Observability". In: *Proceedings of the 34th International Conference on Machine Learning*. 2017, pp. 2681–2690.

[173] Liviu Panait, Karl Tuyls, and Sean Luke. "Theoretical advantages of lenient learners: An evolutionary game theoretic perspective". In: *The Journal of Machine Learning Research* 9 (2008), pp. 423–457.

[174] Ermo Wei and Sean Luke. "Lenient learning in independent-learner stochastic cooperative games". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 2914–2955.

[175] Gregory Palmer et al. "Lenient Multi-Agent Deep Reinforcement Learning". In: *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*. 2018.

[176] Gregory Palmer, Rahul Savani, and Karl Tuyls. "Negative Update Intervals in Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[177] Xueguang Lu and Christopher Amato. "Decentralized Likelihood Quantile Networks for Improving Performance in Deep Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1812.06319v4* (2019).

[178] Marc G. Bellemare, Will Dabney, and Rémi Munos. "A Distributional Perspective on Reinforcement Learning". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017.

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

[180] Jacopo Castellini et al. "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, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[181] Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. "Deep Coordination Graphs". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 2020.

[182] Ming Zhou et al. "Factorized Q-learning for large-scale multi-agent systems". In: *Proceedings of the First International Conference on Distributed Artificial Intelligence, DAI 2019, Beijing, China, October 13-15, 2019*. 2019.

[183] Jiechuan Jiang et al. "Graph Convolutional Reinforcement Learning". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[184] Chao Qu et al. "Intention Propagation for Multi-agent Reinforcement Learning". In: *arXiv preprint arXiv:2004.08883* (2020).

[185] Jakob N. Foerster et al. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[186] Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. "Learning Multiagent Communication with Backpropagation". In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. 2016.

[187] Aleksandra Malysheva et al. "Deep Multi-Agent Reinforcement Learning with Relevance Graphs". In: *arXiv preprint arXiv:1811.12557* (2018).

[188] Yuhang Zhao and Xiujun Ma. "Learning Efficient Communication in Cooperative Multi-Agent Environment". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[189] Simon Pageaud et al. "Multiagent Learning and Coordination with Clustered Deep Q-Network". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[190] Shariq Iqbal and Fei Sha. "Actor-Attention-Critic for Multi-Agent Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[191] Yali Du et al. "LIIR: Learning Individual Intrinsic Reward in Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[192] Xueguang Lyu et al. "Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning". In: *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*. 2021.

[193] Andrea Baisero and Christopher Amato. "Unbiased Asymmetric Actor-Critic for Partially Observable Reinforcement Learning". In: *arXiv preprint arXiv:2105.11674* (2021).

[194] Bei Peng et al. "FACMAC: Factored Multi-Agent Centralised Policy Gradients". In: *arXiv preprint arXiv:2003.06709* (2020).

[195] Meng Zhou et al. "Learning Implicit Credit Assignment for Multi-Agent Actor-Critic". In: *arXiv preprint arXiv:2007.02529* (2020).

[196] Yihan Wang et al. "DOP: Off-Policy Multi-Agent Decomposed Policy Gradients". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[197] Alex Tong Lin et al. "CESMA: Centralized Expert Supervises Multi-Agents". In: *arXiv preprint arXiv:1902.02311* (2019).

[198] Yaodong Yang et al. "Q-value Path Decomposition for Deep Multiagent Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 2020.

[199] Wendelin Böhmer, Tabish Rashid, and Shimon Whiteson. "Exploration with Unreliable Intrinsic Reward in Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1906.02138* (2019).

[200] Shariq Iqbal and Fei Sha. "Coordinated Exploration via Intrinsic Rewards for Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:1905.12127* (2019).

[201] Tarun Gupta et al. "UneVEn: Universal Value Exploration for Multi-Agent Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. 2021.

[202] Hengyuan Hu and Jakob N. Foerster. "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[203] Jakob N. Foerster et al. "Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[204] Siyi Hu et al. "UPDeT: Universal Multi-agent Reinforcement Learning via Policy Decoupling with Transformers". In: *arXiv preprint arXiv:2101.08001* (2021).

[205] Tianze Zhou et al. "Cooperative Multi-Agent Transfer Learning with Level-Adaptive Credit Assignment". In: *arXiv preprint arXiv:2106.00517* (2021).

[206] Shariq Iqbal et al. "Randomized Entity-wise Factorization for Multi-Agent Reinforcement Learning". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. 2021.

[207] Mikayel Samvelyan et al. "The StarCraft Multi-Agent Challenge". In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[208] Weixun Wang et al. "Action Semantics Network: Considering the Effects of Actions in Multiagent Systems". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[209] Tianjun Zhang et al. "Multi-Agent Collaboration via Reward Attribution Decomposition". In: *arXiv preprint arXiv:2010.08531* (2020).

[210] Yaodong Yang et al. "Qatten: A General Framework for Cooperative Multiagent Reinforcement Learning". In: *arXiv preprint arXiv:2002.03939* (2020).

[211] Tonghan Wang et al. "ROMA: Multi-Agent Reinforcement Learning with Emergent Roles". In: *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*. 2020.

[212] Tonghan Wang et al. "RODE: Learning Roles to Decompose Multi-Agent Tasks". In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[213] Tonghan Wang et al. "Learning Nearly Decomposable Value Functions Via Communication Minimization". In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[214] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. "Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control". In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[215] Xinghu Yao et al. "SMIX(λ): Enhancing Centralized Value Functions for Cooperative Multi-Agent Reinforcement Learning". In: *arXiv:1911.04094 [cs]* (2019).

[216] Ling Pan et al. "Softmax with Regularization: Better Value Estimation in Multi-Agent Reinforcement Learning". In: *arXiv preprint arXiv:2103.11883* (2021).

[217] Yong Liu et al. "Value Function Transfer for Deep Multi-Agent Reinforcement Learning Based on N-Step Returns". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019.

[218] Jiachen Yang, Igor Borovikov, and Hongyuan Zha. "Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery". In: *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*. 2020.

[219] Benjamin Eysenbach et al. "Diversity is All You Need: Learning Skills without a Reward Function". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[220] Haotian Fu et al. "Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. 2019.

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

[222] Christian Schröder de Witt and Thomas Hornigold. "Stratospheric Aerosol Injection as a Deep Reinforcement Learning Problem". In: *arXiv preprint arXiv:1905.07366* (2019).

[223] Christian Schröder de Witt et al. "Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control". In: *arXiv preprint arXiv:2003.06709* (2020).

[224] Wei-Fang Sun, Cheng-Kuang Lee, and Chun-Yi Lee. "DFAC Framework: Factorizing the Value Function via Quantile Mixture for Multi-Agent Distributional Q-Learning". In: *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*. 2021.

[225] Nicolas Usunier et al. "Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks". In: *arXiv preprint arXiv:1609.02993* (2016).

[226] Gabriel Synnaeve et al. "TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games". In: *arXiv preprint arXiv:1611.00625* (2016).

[227] Weixun Wang et al. "From Few to More: Large-Scale Dynamic Multiagent Curriculum Learning". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. 2020.

[228] Sanjeev Arora et al. “A Convergence Analysis of Gradient Descent for Deep Linear Neural Networks”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[229] Kyunghwan Son et al. “QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[230] Damien Ernst, Pierre Geurts, and Louis Wehenkel. “Tree-Based Batch Mode Reinforcement Learning”. In: *J. Mach. Learn. Res.* (2005).

[231] Marc G. Bellemare et al. “Increasing the Action Gap: New Operators for Reinforcement Learning”. In: *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. 2016.

[232] Mikayel Samvelyan et al. “The StarCraft Multi-Agent Challenge”. In: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS ’19, Montreal, QC, Canada, May 13-17, 2019*. 2019.

[233] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*. 2017.

[234] Peter Sunehag et al. “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 2018, Stockholm, Sweden, July 10-15, 2018*. 2018.

[235] Tabish Rashid et al. “Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning”. In: *Journal of Machine Learning Research* 21 (2020), pp. 1–51.

[236] Tuomas Haarnoja et al. “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor”. In: *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*. 2018.

[237] Shariq Iqbal and Fei Sha. “Actor-Attention-Critic for Multi-Agent Reinforcement Learning”. In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. 2019.

[238] Ermo Wei et al. “Multiagent Soft Q-Learning”. In: *2018 AAAI Spring Symposia, Stanford University, Palo Alto, California, USA, March 26-28, 2018*. 2018.

[239] Chao Yu et al. “The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games”. In: *arXiv preprint arXiv:2103.01955* (2021).

[240] Kyunghwan Son et al. “QOPT: Optimistic Value Function Decentralization for Cooperative Multi-Agent Reinforcement Learning”. In: *arXiv preprint arXiv:2006.12010* (2020).

[241] Kyunghwan Son et al. “QTRAN++: Improved Value Transformation for Cooperative Multi-Agent Reinforcement Learning”. In: *arXiv preprint arXiv:2006.12010* (2020).

[242] Jianhao Wang et al. “QPLEX: Duplex Dueling Multi-Agent Q-Learning”. In: *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. 2021.

[243] Jianhao Wang et al. “Towards Understanding Linear Value Decomposition in Cooperative Multi-Agent Q-Learning”. In: *arXiv preprint arXiv:2006.00587* (2020).

[244] Liviu Panait, Sean Luke, and R. Paul Wiegand. “Biasing Coevolutionary Search for Optimal Multiagent Behaviors”. In: *IEEE Trans. Evol. Comput.* (2006).

[245] Michael P. Wellman. “Methods for Empirical Game-Theoretic Analysis”. In: *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. 2006.

[246] David Balduzzi et al. “Re-evaluating evaluation”. In: *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*. 2018.

[247] Paul Muller et al. “A Generalized Training Approach for Multiagent Learning”. In: *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. 2020.

[248] John Nash. “Non-cooperative games”. In: *Annals of mathematics* (1951), pp. 286–295.

[249] Arpad E Elo. *The rating of chessplayers, past and present*. Arco Pub., 1978.

[250] David Silver et al. “Mastering the game of go without human knowledge”. In: *nature* 550.7676 (2017), pp. 354–359.

[251] Mark Rowland et al. “Multiagent Evaluation under Incomplete Information”. In: *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*. 2019.

[252] Drew Fudenberg and Jean Tirole. *Game theory (3. pr.)* 1991.

[253] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer Science & Business Media, 2008.

[254] Ralf Herbrich, Tom Minka, and Thore Graepel. “TrueSkill™: a Bayesian skill rating system”. In: *Advances in neural information processing systems*. 2007, pp. 569–576.

[255] Tom Minka, Ryan Cleven, and Yordan Zaykov. “Trueskill 2: An improved bayesian skill rating system”. In: *Tech. Rep.* (2018).

[256] Shuo Chen and Thorsten Joachims. “Modeling Intransitivity in Matchup and Comparison Data”. In: *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining, San Francisco, CA, USA, February 22-25, 2016*. 2016.

[257] William E. Walsh, David C. Parkes, and Rajarshi Das. “Choosing Samples to Compute Heuristic-Strategy Nash Equilibrium”. In: *Agent-Mediated Electronic Commerce V, Designing Mechanisms and Systems, AAMAS 2003 Workshop, AMEC 2003, Melbourne, Australia, July 15, 2003, Revised Selected Papers*. 2003.

[258] Karl Tuyls et al. “Bounds and dynamics for empirical game theoretic analysis”. In: *Auton. Agents Multi Agent Syst.* (2020).

[259] Patrick R. Jordan, Yevgeniy Vorobeychik, and Michael P. Wellman. “Searching for approximate equilibria in empirical games”. In: *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 2*. 2008.

[260] Yaodong Yang et al. “$\alpha\alpha$-Rank: Practically Scaling α-Rank through Stochastic Optimisation”. In: *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*. 2020, pp. 1575–1583.

[261] Niranjan Srinivas et al. “Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design”. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*. 2010.

[262] Yali Du et al. “Estimating α-Rank from A Few Entries with Low Rank Matrix Completion”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 2870–2879.

[263] Wojciech M. Czarnecki et al. “Real World Games Look Like Spinning Tops”. In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*. 2020.

[264] Ruslan Salakhutdinov and Andriy Mnih. “Bayesian probabilistic matrix factorization using Markov chain Monte Carlo”. In: *Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland, June 5-9, 2008*. 2008.

[265] Xiaoyuan Su and Taghi M. Khoshgoftaar. “A Survey of Collaborative Filtering Techniques”. In: *Adv. Artif. Intell.* (2009).

[266] Anthony O’Hagan. “Bayes–hermite quadrature”. In: *Journal of statistical planning and inference* 29.3 (1991), pp. 245–260.

[267] Carl Edward Rasmussen and Zoubin Ghahramani. “Bayesian monte carlo”. In: *Advances in neural information processing systems* (2003), pp. 505–512.

[268] Ilya Nemenman, F. Shafee, and William Bialek. “Entropy and Inference, Revisited”. In: *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*. 2001.

[269] Yasin Abbasi-Yadkori, Dávid Pál, and Csaba Szepesvári. “Improved Algorithms for Linear Stochastic Bandits”. In: *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain*. 2011.

[270] Nicolas Bonneel et al. “Displacement interpolation using Lagrangian mass transport”. In: *ACM Trans. Graph.* (2011).

[271] R’emi Flamary and Nicolas Courty. *POT Python Optimal Transport library*. 2017. URL: `https://pythonot.github.io/`.

[272] Ronald Fagin et al. “Comparing Partial Rankings”. In: *SIAM J. Discret. Math.* (2006).

[273] Blizzard Entertainment. *New Blizzard Custom Game: StarCraft Master*. http://us.battle.net/sc2/en/blog/4544189/new-blizzard-custom-game-starcraft-master-3-1-2012. Accessed: 2018-11-16. 2012.

[274] Eric Jang, Shixiang Gu, and Ben Poole. "Categorical Reparameterization with Gumbel-Softmax". In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. 2017.

[275] Anurag Koul. *ma-gym: Collection of multi-agent environments based on OpenAI gym*. https://github.com/koulanurag/ma-gym. 2019.

[276] Marsili Simone. *ndd - Bayesian entropy estimation from discrete data*. URL: https://github.com/simomarsili/ndd.