Coordination and Communication in Deep Multi-Agent Reinforcement Learning

Christian A. Schroeder de Witt

St Catherine's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity 2021

Abstract

A growing number of real-world control problems require teams of software agents to solve a joint task through cooperation. Such tasks naturally arise whenever human workers are replaced by machines, such as robot arms in manufacturing or autonomous cars in transportation. At the same time, new technologies have given rise to novel cooperative control problems that are beyond human reach, such as in package routing.

Be it for physical constraints such as partial observability, robustness requirements, or to manage large joint action spaces, cooperative agents are often required to function in a fully decentralised fashion. This means that each agent merely has access to its own local sensory input during task execution, and does not have explicit communication channels to other agents. Deep multi-agent reinforcement learning (DMARL) is a natural framework for learning control policies in such settings. When trained in simulation or in a laboratory, learning algorithms often have access to additional information that will not be available at execution. Such centralised training with decentralised execution (CTDE) poses a number of technical challenges to DMARL algorithms that try to exploit the centralised setting in order to facilitate the training of decentralised policies. These difficulties arise primarily from the apparent incongruency between joint policy learning, which can learn arbitrary policies but is not naively decentralisable and scales poorly with the number of agents, and independent learning, which is readily decentralisable and scalable but provably less expressive and prone to environment non-stationarity due to the presence other of learning agents.

The first part of this thesis develops algorithms that use the technique of value decomposition in order to exploit the centralised training of decentralised policies. In *Monotonic Value Factorisation for Deep Multi-Agent Reinforcement Learning*, we introduce the novel Q-learning algorithm QMIX. QMIX uses a centralised monotonic mixing network in order to model joint team action-value functions that are nevertheless decomposable into decentralised agent policies over discrete action spaces. To evaluate the performance of QMIX, we develop a novel benchmark suite, the StarCraft Multi-Agent Challenge (SMAC), which features a variety of discrete-action cooperative control tasks in StarCraft II unit micromanagement. Unlike pre-existing toy environments, SMAC scenarios feature diverse dynamics owing to a large number of different unit types and sophisticated in-built enemy heuristics. Many robotic control tasks feature continuous action spaces. To extend value decomposition to those settings, in *FACMAC: Factored Multi-Agent Centralised Policy Gradients*, we focus on actor-critic approaches to multi-agent learning in CTDE settings. The resulting learning algorithm, *FACMAC*, achieves state-of-the-art performance on SMAC and opens the door toward using nonmonotonic critic factorisations. Just as for QMIX, we introduce a novel benchmark suite for cooperative continuous control tasks, Multi-Agent Mujoco (MAMujoco). MAMujoco decomposes robots from the popular Mujoco benchmark suite into multiple agents with configurable partial observability constraints.

The second part of this thesis explores the value of *common knowledge* as a resource for both coordinating and communicating through actions. Common knowledge between groups of agents arises in a large class of tasks of practical interest, for example, if agents can recognize each other in overlapping fields of view. In *Multi-Agent Common Knowledge Reinforcement Learning*, we introduce a novel actor-critic method, MACKRL, which constructs a hierarchy of controllers over common knowledge across agent groups of varying sizes. This hierarchy gives rise to a decentralized policy structure that effectuates a joint-independent hybrid policy which executes decentralized joint policies or falls back to independent policies depending on whether the common knowledge between agent groups is sufficiently informative for action coordination. In this way, MACKRL enjoys the coordinative advantage of joint policy training while being fully decentralised.

The third part of thesis investigates how to learn efficient implicit communi-

cation protocols for collaborative tasks. In *Communicating via Markov Decision Processes*, we explore how a sender agent can execute a task optimally while at the same time communicating information to a receiver agent solely through its actions. In this novel *implicit* referential game, both sender and receiver agents commonly know both the sender policy, as well as the sender's trajectory. By splitting the sender task into a single-agent maximum entropy reinforcement learning task and a separate message encoding step based on minimum-entropy coupling, we show that our method *GME* allows establishing communication channels of significantly higher bandwidth than those trained end-to-end.

In summary, this thesis presents a number of significant contributions deep multi-agent reinforcement for cooperative control within the framework of centralised training with decentralised execution and two associated novel benchmark suites. Within this setting, we make contributions to *value decomposition*, the use of *common knowledge* in multi-agent learning, and how to learn *implicit communication* protocols efficiently.

Coordination and Communication in Deep Multi-Agent Reinforcement Learning

Christian A. Schroeder de Witt

St Catherine's College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Trinity 2021

Acknowledgements

I owe thanks to a large number of people who have been supporting my work in myriads of ways. First, my supervisors: Prof. Philip Torr has given me a home in the Torr Vision Group and has been a source of endless encouragement and support. Prof. Shimon Whiteson accepted to co-supervise me as part of WhiRL and was likewise a never-ending source of advice. I could not have wished for a better team of supervisors, as I have had the best of many worlds.

Secondly, the generous support from industry that both funded my research generously, as well as provided me with an invaluable window into real-world applications. I would like to thank Microsoft UK and the project Free the Drones (FreeD) under the Innovation Fund Denmark for generously funding the first three years of my DPhil, and Armasuisse Science and Technology for providing funding for my fourth year. In particular, I would like to thank Brad Beach (UAS Test Centre Denmark) and Dr. Martin Strohmeier (Armasuisse) for all the advice and support they have provided me with during the duration of my thesis.

During the duration of my DPhil, I had the opportunity to accept a variety of internship positions. Dr. Anthony Ledford was not only an excellent supervisor throughout my internship at Man AHL ahead of my DPhil in summer 2016, but also warmly encouraged me to return to Oxford for my DPhil: With Dr. Anthony Ledford and Prof. Stephen Roberts, my career may have taken a very different path. I would like to thank Dominik Roblek and Prof. Marco Tagliasacchi for letting me dive into AutoML and self-supervised learning during my internship at Google AI in Zurich in the summer of 2019. My participation in the Frontier Development Lab in the summer of 2020 allowed me to pick up valuable expertise in the climate sciences. I would like to thank James Parr, Dr Duncan Watson-Parris and Dr Matthew Chantry for making this possible.

I owe Prof. Jakob Foerster not only for inciting my interest in deep multi-agent reinforcement learning, but also for providing me with focus, encouragement and opportunities throughout my postgraduate degree. In alphabetic order, I thank my collaborators: Khaulat Abdulhakeem, Eltayeb Ahmed, Myles Allen,

Gunes Baydin, Harkirat Behl, Marilena Bescuca, Piotr Bilinski, Wendelin Böhmer, Julien Brajard, Noam Brown, Nicholas Burden, Matthew Chantry, Daniele De Martini, Peter Dueben, Gregory Farquhar, Yarin Gal, Andrew Gambardella, Stuart Golodetz, Prateek Gupta, Thomas Hornigold, Maximilian Igl, Freddie Kalaitzis, Lucas Kruitwagen, Pierre-Alexandre Kamienny, Aidas Kilda, Lucas Kruitwagen, Bertrand Le Saux, Adam Lerer, Yat "Richie" Lo, Jelena Luketina, Ondra Miksik, Darius Muglich, Nantas Nardelli, Bei Peng, Dylan Radovic, Tom Rainforth, Tabish Rashid, Stephan Rasp, Stefan Saftescu, Mikayel Samvelyan, Martin Scholl, Mattie Smith, Sam Sokota, Edward Fleri Soler, Carole Stoker, Catherine Tong, Oliver Warwick, Duncan Watson-Parris, Laura Weihl, Yee Whye Teh, Zhao Yang, Valentina Zantedeschi, Robert Zinkov and Luisa Zintgraf.

I owe huge thanks to the Oxford School of Climate Change, run by the Oxford Climate Society. Learning from some of the world's foremost experts on climate policy and immersing myself in the extraordinary community of students and faculty. Thomas Hornigold introduced me to solar geoengineering, leading to a prize-winning joint proposal paper at the international Climate Change and AI community that captured the attention of Andrew Ng and John Platt. I would like to thank Dr. Carol Stoker and Prof. Myles Allen for furthering my horizon on geoengineering research, as well as Prof. Yoshua Bengio for accepting me to his working group on retroactive carbon pricing. I am proud to consider the inspiring Kaya Axelsson and Tristram Walsh my friends, it has been such a pleasure to be part of Oxford Net Zero working on SME decarbonisation and gaining insights into climate policy.

Countless friends and peers have accompanied me on my way, for better or worse. I am particularly grateful to Julia, Judith, Jeremy, Jet, Ruby, Mona, Puffin, Butterscotch, Sunny, and all the other furry inhabitants of Sandiacre Farm, Ev, Emily, Claudia, Jenya, Sofi, Isla, Irofili, gFamily, Mark, Katrin, Kai, Olmo, Matt, Martin, Stefan, Emma, Peter, and many more.

Naturally, I would like to thank my parents Christina and Holger, and my sister Anna, for providing never-ending support and encouragement throughout my career.

Abstract

A growing number of real-world control problems require teams of software agents to solve a joint task through cooperation. Such tasks naturally arise whenever human workers are replaced by machines, such as robot arms in manufacturing or autonomous cars in transportation. At the same time, new technologies have given rise to novel cooperative control problems that are beyond human reach, such as in package routing.

Be it for physical constraints such as partial observability, robustness requirements, or to manage large joint action spaces, cooperative agents are often required to function in a fully decentralised fashion. This means that each agent merely has access to its own local sensory input during task execution, and does not have explicit communication channels to other agents. Deep multi-agent reinforcement learning (DMARL) is a natural framework for learning control policies in such settings. When trained in simulation or in a laboratory, learning algorithms often have access to additional information that will not be available at execution. Such centralised training with decentralised execution (CTDE) poses a number of technical challenges to DMARL algorithms that try to exploit the centralised setting in order to facilitate the training of decentralised policies. These difficulties arise primarily from the apparent incongruency between joint policy learning, which can learn arbitrary policies but is not naively decentralisable and scales poorly with the number of agents, and independent learning, which is readily decentralisable and scalable but provably less expressive and prone to environment non-stationarity due to the presence other of learning agents.

The first part of this thesis develops algorithms that use the technique of value decomposition in order to exploit the centralised training of decentralised policies. In *Monotonic Value Factorisation for Deep Multi-Agent Reinforcement Learning*, we introduce the novel Q-learning algorithm QMIX. QMIX uses a centralised monotonic mixing network in order to model joint team action-value functions that are nevertheless decomposable into decentralised agent policies over discrete action spaces. To evaluate the performance of QMIX, we develop a novel benchmark suite, the StarCraft Multi-Agent Challenge (SMAC), which features a variety of discrete-action cooperative control tasks in StarCraft II unit micromanagement. Unlike pre-existing toy environments, SMAC scenarios feature diverse dynamics owing to

a large number of different unit types and sophisticated in-built enemy heuristics. Many robotic control tasks feature continuous action spaces. To extend value decomposition to those settings, in *FACMAC: Factored Multi-Agent Centralised Policy Gradients*, we focus on actor-critic approaches to multi-agent learning in CTDE settings. The resulting learning algorithm, *FACMAC*, achieves state-of-the-art performance on SMAC and opens the door toward using nonmonotonic critic factorisations. Just as for QMIX, we introduce a novel benchmark suite for cooperative continuous control tasks, Multi-Agent Mujoco (MAMujoco). MAMujoco decomposes robots from the popular Mujoco benchmark suite into multiple agents with configurable partial observability constraints.

The second part of this thesis explores the value of *common knowledge* as a resource for both coordinating and communicating through actions. Common knowledge between groups of agents arises in a large class of tasks of practical interest, for example, if agents can recognize each other in overlapping fields of view. In *Multi-Agent Common Knowledge Reinforcement Learning*, we introduce a novel actor-critic method, MACKRL, which constructs a hierarchy of controllers over common knowledge across agent groups of varying sizes. This hierarchy gives rise to a decentralized policy structure that effectuates a joint-independent hybrid policy which executes decentralized joint policies or falls back to independent policies depending on whether the common knowledge between agent groups is sufficiently informative for action coordination. In this way, MACKRL enjoys the coordinative advantage of joint policy training while being fully decentralised.

The third part of thesis investigates how to learn efficient implicit communication protocols for collaborative tasks. In *Communicating via Markov Decision Processes*, we explore how a sender agent can execute a task optimally while at the same time communicating information to a receiver agent solely through its actions. In this novel *implicit* referential game, both sender and receiver agents commonly know both the sender policy, as well as the sender's trajectory. By splitting the sender task into a single-agent maximum entropy reinforcement learning task and a separate message encoding step based on minimum-entropy coupling, we show that our method *GME* allows establishing communication channels of significantly higher bandwidth than those trained end-to-end.

In summary, this thesis presents a number of significant contributions deep multi-agent reinforcement for cooperative control within the framework of centralised training with decentralised execution and two associated novel benchmark suites. Within this setting, we make contributions to *value decomposition*, the use of *common knowledge* in multi-agent learning, and how to learn *implicit communication* protocols efficiently.

Contents

List of Figures

List of Abbreviations and Notations

AC . . . . . Actor-Critic

AI . . . . . Artificial Intelligence

Dec-POMDP Decentralised Partially Observable Markov Decision Process

DMARL . . . Deep Multi-Agent Reinforcement Learning

DQN . . . . . Deep Q-Network

DRQN . . . . . Deep Recurrent Q-Network

ERA . . . . . E

FF . . . . . Feedforward Neural Network

IL . . . . . Independent Learning

IQL . . . . . Independent Q-Learning

GME . . . . . Greedy Minimum Entropy Coupling

MACKRL . . Multi-Agent Common Knowledge Reinforcement Learning

MARL . . . . Multi-Agent Reinforcement Learning

MCG . . . . . Markov Coding Game

MDProc . . . Markov Decision Process

MDP . . . . . Markov Decision Problem

ML . . . . . Machine Learning

MLP . . . . . Multi-Layer Perceptron

NN . . . . . Deep Neural Network

PG . . . . . Policy Gradient

POSG . . . . . Partially Observable Stochastic Game

RL . . . . . Reinforcement Learning

SAI . . . . . Stratospheric Aerosol Injection

SF . . . . . Score Function

STD . . . . . Standard Deviation

Notation

$s \in \mathcal{S}$	State and state space.
n	Number of agents/players.
t	Timestep in episode.
T	Duration of episode.
$a \in \{1, \ldots, n\}$	Agent index.
$z_t^a = \mathcal{Z}(s_t, a)$	Observation for agent a and observation function.
$u^a \in \mathcal{U}$	Action of agent a and action space.
r_t^a	Reward of agent a at time step t.
γ	Discount factor.
$R_t^a = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^a$	Forward-looking return of agent a.
$\tau_t^a = \{z_0^a, \ldots, z_t^a\}$	Action-observation history of agent a.
$\pi^a(u^a \vert \tau^a)$	(Stochastic or deterministic) policy of agent a.
$\mu^a(u^a \vert \tau^a)$	Deterministic policy of agent a.
m_t^a	Message from agent a sent at time t.
$\mathbf{u}, \boldsymbol{\pi}, \boldsymbol{\tau}$	Joint action, joint policy, joint observation history across all agents.
$P(s' \vert s, \mathbf{u})$	State transition function.
$V^{\boldsymbol{\pi}}(s)$	State value function for policy $\boldsymbol{\pi}$.
$Q^{\boldsymbol{\pi}}(s)$	Q-function: State-action value function, conditioned on the central state.
$Q^{\pi}(\tau^a, u^a)$	Local Q-function for agent a.
$J^a(\boldsymbol{\pi})$	Expected return of agent a induced by the joint policy $\boldsymbol{\pi}$.
h_t^a	Hidden state of a recurrent network for agent a.

1 Introduction

Contents

Artificial intelligence (AI) has become a major agent of change to the world's economy: Spurred by breakthroughs in computational and robotic hardware design and manufacture, the availability and accessibility of training data, and machine learning algorithms, AI has already played a substantial role in establishing tech companies as seven out of the top ten most valuable companies in the world at the time of writing (Statista, 2020). This trend is generally thought to continue, with AI alone projected to contribute roughly China's 2020 GDP to the world economy by 2030 (PriceWaterhouseCoopers, 2017), with about three fifths arising from consumption, and the rest from productivity increases.

A common narrative of and motivation for AI research is the replacement of cognition tasks performed by human workers through intelligent machines - a process that is analogeous to the replacement of manual labor during earlier industrial revolutions (Thomas, 2020). Recent works have investigated the effects on AI on various aspects of society, such as future politics (Susskind, 2018), justice (Susskind, 2019) and professions (Susskind and Susskind, 2016).

Among technical developments, the advent of *supervised deep learning* stands out as particularly impactful (LeCun et al., 2015). However, control settings which involve sequential interaction with an environment violate the i.i.d. sampling assumption behind supervised deep learning. The nascent field of deep reinforcement learning (DRL) combines neural network function approximators with sequential decision-making within the reinforcement learning (RL) framework, where an agent sequentially interacts with an environment and receives scalar reward signals and high-dimensional observations in return. RL has historically drawn much research interest due to its analogies with human and animal learniMORE OPTIONS ng (Watkins, 1989). Single-agent DRL has celebrated notable success, starting with super-human gameplay performance on Atari (Mnih et al., 2015) and even promising results in real-world robotic applications (Andrychowicz et al., 2020). Besides its unique ability to learn control policies by directly conditioning on high-dimensional observations, DRL has also been considered as a possible building block of artificial general intelligence (AGI) (Joe Booth, 2019).

In many cases, control tasks feature multiple agents each of which having access to their own local sets of observations. Deep Multi-Agent Reinforcement Learning (DMARL) is a natural extension of single-agent DRL settings. Many important real-world problems such as autonomous driving (Cao et al., 2013), swarm control (Barnes et al., 2009), robotics (Seuken and Zilberstein, 2007), communication networks (Peshkin, 2000), crisis management (Paquet et al., 2005), recreational games (Bard et al., 2020) and load balancing (Cogill et al., 2006) require multiple agents to work together as a team while solving a single goal. This *decentralized control* (Mahajan and Mannan, 2016) setting is commonly formalised by Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs) in the planning literature (Oliehoek and Amato, 2016), known as common-payoff or identical interest games in game theoretic literature (Leyton-Brown and Shoham, 2008) and referred to as (cooperative) multi-agent reinforcement learning in reinforcement learning (RL) literature (Zhang et al., 2019a). In this thesis, we focus on *cooperative* settings,

a particularly rich space of control tasks that is now starting to find increasing attention by the DMARL research community (Dafoe et al., 2020).

Cooperative multi-agent control centers on two important concepts: *collaboration*, and *communication*. Collaboration refers to the challenge of learning agent policies concurrently with other learning agents that introduce *nonstationarity* into the shared learning environment and potentially confound reward attribution at individual agent level (also known as the *multi-agent credit assignment* problem). Chapters 3, 4 and 5 introduce novel DMARL methods *QMIX*, *FACMAC* and *MACKRL* that can stabilise multi-agent learning under various conditions. Communication refers to the challenge of learning communication protocols that allow agents to exchange information during policy execution. Communication between agents can be either explicit using *cheap-talk* channels, or *implicit*, by observing other agents' actions or their effect on the environment (see Section 2.6). Chapter 6 introduces *GME*, a method that allows agents to construct implicit communication protocols that can scale to very large messages while allowing to complete the agent's tasks perfectly. We note that both collaboration and communication also appear in settings that are not purely cooperative, i.e. in which agents both compete and cooperate. The methods developed in this thesis may inform method development in general-sum settings (see e.g. (Anthony et al., 2020; Gray et al., 2020)).

DMARL, as other deep-learning based methods, requires access to a large set of training samples. Fortunately, in most situations, DMARL policies can be trained in simulation or in a laboratory. In such settings learning can naturally proceed in a *centralised* fashion, meaning that constraints on agent communication can be lifted and extra state information can be used throughout the training process. Part I introduces a variety of new algorithms that can make use of the *centralized training with decentralized execution (CTDE)* (Oliehoek and Amato, 2016) setting. We note that transferring control policies learnt in simulation to the real world is a very active field of DRL research (Tobin et al., 2017; Sadeghi and Levine, 2017).

Unfortunately, computing even an approximately optimal joint policy to Dec-POMDPs is NEXP-complete (Rabinovich et al., 2003). This clearly separates

Dec-POMDPs from the problem of decentrally controlling a system with free communication, which is known to be in P-SPACE(Blondel and Tsitsiklis, 2000). At first sight, this problem hardness has an important consequence: We cannot expect to be able to ever find an algorithm that can solve all Dec-POMDPs problems efficiently - no matter how much we scale our algorithm (Sutton, 2019), or how many inductive biases we use (Goyal and Bengio, 2021). However, there is a rather subtle caveat: While it is known that the class of Dec-POMDP problems contains at least some problems that are hard, it does not necessarily mean that every problem of every practical interest is indeed as hard. In fact, the problem classes constructed to prove the aforementioned complexity results, such as *TILING* (Bernstein et al., 2000), are arguably far removed from real-world problems.

This reflection on problem complexity underlines a recurrent theme throughout this thesis: Meaningful method development in multi-agent systems control crucially depends on the design of meaningful benchmark tasks. Meaningful benchmark tasks should ultimately be grounded in meaningful applications. In this thesis, we therefore introduce a variety of novel benchmark environments for DMARL research: Chapter 3 introduces the StarCraft *Multi-Agent Challenge (SMAC)*, a suite of unit micromanagement tasks, Chapter 4 introduces *Multi-Agent Mujoco (MAMujoco)*, a collection of multi-agent robotic control tasks and Chapter *ch-gme* introduces a variety of *implicit referential game (IRG)* benchmark tasks.

Different application domains may require the use of fundamentally different inductive biases and algorithmic approaches: For example, as the work presented in this thesis shows, algorithms that work efficiently on micromanagement tasks in StarCraft and multi-agent robotic control (see Part I) may be entirely inefficient on tasks involving the learning of communication protocols, and vice versa (see Part II). While it has been argued that all that matters is how well a learning algorithm scales with hardware (Sutton, 2019), we suggest that algorithmic scalability is a necessary, but not sufficient, research goal. We establish common knowledge, also referred to as public information, as an effective inductive bias for deep multi-agent reinforcement learning in a variety of decentralised control settings in Part II.

1.1 Thesis Structure

This thesis starts with a background section followed by three main parts, a brief overview of which are given below. Each part tackles one or several of the challenges presented above.

Background

Before presenting original research, we familiarise the reader with the necessary terminology and concepts that are commonly required by all of the research chapters (see Chapter 2). Concepts and terminology, as well as specialised literature reviews, specific to individual research chapters are introduced *in situ* to facilitate the reading process.

Part 1: Value decomposition

The first part of this thesis develops algorithms that use the technique of value decomposition in order to exploit the centralised training of decentralised policies. In *Monotonic Value Factorisation for Deep Multi-Agent Reinforcement Learning*, we introduce the novel Q-learning algorithm QMIX. QMIX uses a centralised monotonic mixing network in order to model joint team action-value functions that are nevertheless decomposable into decentralised agent policies over discrete action spaces. To evaluate the performance of QMIX, we develop a novel benchmark suite, the StarCraft Multi-Agent Challenge, which features a variety of discrete-action cooperative control tasks in StarCraftII unit micromanagement. Unlike pre-existing toy environments, SMAC scenarios feature diverse dynamics owing to a large number of different unit types and sophisticated in-built enemy heuristics.

Many robotic control tasks feature continuous action spaces. To extend value decomposition to those settings, in *FACMAC: Factored Multi-Agent Centralised Policy Gradients* we propose two novel DMARL algorithms: FACMAC and COMIX. COMIX is a variant of QMIX that scales to continuous action spaces and FacMADDPG is a deterministic policy gradient method with a factored critic. Unlike COMIX, FACMAC allows us to compare the performance of arbitrary value decompositions

without infringing on policy decentralisability. Just as for QMIX, we introduce a novel benchmark suite for cooperative continuous control tasks, Multi-Agent Mujoco (MAMujoco). MAMujoco decomposes robots from the popular Mujoco benchmark suite into multiple agents with configurable partial observability constraints.

The above chapters are based on the following papers and pre-prints. Note that throughout this thesis, "*" indicates equal contribution.

- T. Rashid*, M. Samvelyan*, C. Schroeder de Witt, G Farquhar, J Foerster, and S Whiteson. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. Journal of Machine Learning Research, 21 (178):1–51, 2020. ISSN 1533-7928.

- B. Peng*, T. Rashid*, C. Schroeder de Witt*, P. Kamienny, P. Torr, W. Böhmer, and S. Whiteson. FACMAC: Factored Multi-Agent Centralised Policy Gradients. Accepted at NeurIPS 2021 and to be published in Advances in Neural Information Processing Systems. arXiv:2003.06709 [cs, stat].

Part 2: Common Knowledge

The second part of this thesis explores the value of *common knowledge* as a resource for both coordinating and communicating through actions. Common knowledge between groups of agents arises in a large class of tasks of practical interest, for example, if agents can recognize each other in overlapping field-of-views. In *Multi-Agent Common Knowledge Reinforcement Learning*, we introduce a novel actor-critic method, MACKRL, which constructs a hierarchy of controllers over common knowledge across agent groups of varying sizes. This hierarchy gives rise to a decentralized policy structure that effectuates a joint-independent hybrid policy that executes decentralized joint policies or falls back to independent policies depending on whether the common knowledge between agent groups is sufficiently informative for action coordination. In this way, MACKRL enjoys the coordinative

advantage of joint policy training while being fully decentralised.

This chapter is based on the following papers and pre-prints:

- C. Schroeder de Witt*, J. Foerster*, G. Farquhar, P. Torr, W. Böhmer, and S. Whiteson. Multi-agent Common Knowledge Reinforcement Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 32, pages 9927–9939. Curran Associates, Inc., 2019.

Part 3: Implicit Communication

The third, and final part of this thesis introduces *implicit* referential games, in which a sender agent needs to both execute a single-agent task, while at the same time choosing actions such that a receiver agent observing their trajectory will be able to decode a message from within a possibly huge message space. In *Communicating via Markov Decision Processes*, we explore how common knowledge can enable a sender agent to execute a task optimally while communicating information to a receiver agent solely through its actions. In this form of *implicit* referential game, both sender and receiver agents commonly know both the sender policy, as well as the sender's trajectory. By splitting the sender task into a single-agent maximum entropy reinforcement learning task and a separate message encoding step based on minimum-entropy coupling, we show that our method *GME* allows establishing communication channels of significantly higher bandwidth than those trained end-to-end.

- S. Sokota* and C. Schroeder de Witt*, L. Zintgraf, M. Igl, Z. Kolter, P.H.S. Torr, S. Whiteson, J. Foerster. Communicating via Markov Decision Processes. arXiv:2107.08295 [cs.AI]. 2021. (*: equal contribution, order determined by coin flip)

Quotation style

All direct quotations in this thesis are indicated visually by single line spacing, whereas original thesis writing is indicated by double spacing.

Comments within quoted blocks are indicated by a change of color.

To make for comfortable reading, supplementary material and other appendices of the original publications included with this thesis have been inserted at appropriate places within the main text. The sections thus inserted are marked by [A].

2
Background

Contents

2.1 Reinforcement Learning

Reinforcement Learning (RL) treats a sequential decision-making process between an agent and an environment $\mathcal{E}$. The dynamics of this decision-making process are characterised by a Markov decision process (MDProc) (Bellman, 1957), which is an extension of a Markov chain (Markov, 1906) to *control* settings in which, at any time t prior to termination at time T, a decision-making agent can take

actions u_t from an action space $\mathcal{U}$ and receives a scalar *reward* $r_t \in \mathbb{R}$ in return. Specifically, given the environment is in state $s_t \in \mathcal{S}$, with a finite state space $\mathcal{S}$, at time t and the agent chooses an action u_t according to some stochastic agent policy $\pi(u_t|s_t) : \mathcal{S}\times\mathcal{U} \to [0, 1]$, the environment transitions to a new state s_{t+1} according to a state transition probability distribution characterised by $P(s_{t+1}|s_t, u_t)$ and emits a reward r_t according to a, potentially stochastic, reward function $R(s_t, u_t) : \mathcal{S}\times\mathcal{U}\to\mathbb{R}$. Both the agent's action space $\mathcal{U}$ and the state space $\mathcal{S}$ can either be continuous or discrete, and we will encounter both cases in this thesis.

We restrict ourselves to the *model-free setting*, in which the agent does not have knowledge of either P or R, nor has access to a model from which it can sample freely. Instead, the agent has to reconstruct these environment statistics through sequential interaction with the environment. We defer a detailed discussion of this choice to Section 2.2.2.

A Markov decision problem (MDP) is a MDProc together with an optimisation goal J. By default, J is given by the total expected cumulative discounted return $J = \mathbb{E}_{\tau\sim P(\tau)} R_0(\tau)$ with $R_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, $\tau = \langle s_0, u_0, r_0, \ldots, s_T\rangle$ is the trajectory and $\gamma \in [0, 1]$ is a temporal discount parameter. For an episode with a finite (or infinite) horizon of T steps, we can write out explicitly:

$$\mathbb{E}_\pi\left[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, u_t)\right] = \int \left(\sum_{t=0}^{T} \gamma^{t-1} r(s_t, u_t)\right) p_1(s_1) \prod_{t=0}^{T} p(s_{t+1}|s_t, u_t)\pi(u_t|s_t)\,\mathrm{d}\tau. \quad (2.1)$$

The distinction between a MDProc and a MDP is not commonly made (for a rare example see Littman et al., 1995), but useful: In maximum-entropy reinforcement learning (MaxEnt RL) (Ziebart et al., 2008; Haarnoja et al., 2017), the objective J is modified to regularize the entropy of the policy $H(\pi)$:

$$J_{MaxEnt} = \mathbb{E}_{\tau\sim P(\tau)} R_0(\tau) + \alpha H(\pi(u_t|s_t)), \quad (2.2)$$

with $\alpha \in \mathbb{R}_+$ being a hyperparameter. Maximum-entropy reinforcement learning is widely used based on providing effective exploration, however, the theoretical

validity of such a use has been questioned (Eysenbach and Levine, 2019). Chapter 6 supplies a novel justification for using a maximum-entropy objective based on maximising policy information capacity in implicit communication (see Section 2.6).

2.1.1 Partial Observability

Real-world robotic agents almost always have to operate under partial observability, i.e. they cannot observe the full state of the world at every time step. This important limitation arises naturally in a number of ways, including through sensor limitations such as limited camera aperture, sensor range, signal noise or resolution. Even if sensors are omni-observant, limited on-board processing capacity may not enable sensory information to be processed or stored appropriately. In addition, occlusion through terrain, buildings or other robotic agents can render the world partially observable. A *Partially Observable Markov Decision Problem (POMDP)* is a generalisation of a MDP in which, instead of observing the environment Markov state s_t directly, the agent only receives an observation $z_t \in \mathcal{Z}$ according to a probability distribution characterised by $O(s_t, u_t)$, where $\mathcal{Z}$ is the observation space. In POMDPs, agent policies condition on the entire action-observation history $\tau \in \mathcal{T} \equiv (\mathcal{Z} \times \mathcal{U})^*$, for which, normally, a sufficient statistic of τ is to be learned using a function approximator (see Section 2.1.3).

2.1.2 Q-learning, policy gradients and actor-critic methods

In RL, gradient-based policy optimisation θ-parametrised $J = \mathbb{E}_{\tau_\theta \sim P(\tau)} R_0(\tau)$ requires estimating $\nabla J(\theta)$ from sampled policy rollouts. Policy gradient methods (Williams, 1992, REINFORCE) use the equivalence

$$\nabla_\theta \mathbb{E}_\pi \left[R(\tau) \right] = \mathbb{E}_\pi \left[R(\tau) \nabla \log \pi(\tau) \right] \tag{2.3}$$

in order to estimate $\nabla J(\theta)$ through MCMC sampling (Metropolis et al., 1953) at high variance. This variance can be reduced through the use of a suitable baseline $b(\tau) : (Z \times U)* \to \mathbb{R}$ that is subtracted from $r(\tau)$. The gradient estimate

remains unbiased as

$$\nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{\tau\ P,\pi} \left[b(\tau) \right] = 0 \tag{2.4}$$

Other solution approaches require the introduction of the concept of a *value function*

$$V_\pi(s_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t:\infty}} \left[R_t | s_t \right] \tag{2.5}$$

induced by an agent's policy π. Similarly, the *action-value function* (or $Q-function$) is defined as

$$Q_\pi(s_t, u_t) = \mathbb{E}_{s_{t+1:\infty}, u_{t+1:\infty}} \left[R_t | s_t, u_t \right]. \tag{2.6}$$

Q-learning (Watkins, 1989) is an off-policy model-free learning algorithm that is based on iteratively solving the Bellman optimality equation (Bellman, 1957)

$$Q^*(s_t, u_t) = \mathbb{E}_{s'} \left[r_t + \gamma \max_{u'} Q^*(s', u') \middle| s_t, u_t \right] \tag{2.7}$$

which can be shown to hold for any MDP as a direct consequence of the Markov property (* indicates optimality). This induces an optimal policy as $u^* = \arg\max_u Q^*(u, s_t)$. Because this method does not require the evaluation of expectations over π, transition tuples can be sampled off-policy. This enables the use of experience replay (Lin, 1992a), which tends to stabilise the learning process by making transition sampling less dependent. Another important characteristic of Q-learning is its usually employed off-policy ϵ-greedy exploration strategy, in which an action u is chosen by an *action selector* that chooses the optimal action with probability $1-\epsilon$ and chosses a random explorative action at probability ϵ, thus striking a balance between exploration and exploitation (Sutton and Barto, 2018). While not central to this thesis, we remark that *Bayesian* reinforcement learning approaches do not obviate the need to handle the exploration/exploitation tradeoff explicitly (Ghavamzadeh et al., 2015). Scaling Bayesian RL is an interesting area for further research (also see Section 2.5) ·

Actor-critic methods (Samuel, 1959; Sutton, 1988) feature a *critic* represented by a value function estimator $\tilde{V}$, and an *actor* represented by a stochastic policy

π. After each action selection, the critic critiques the action taken by the actor by evaluation the *TD-error*

$$\delta_t = r_t + \gamma \tilde{V}(s_{t+1}) - \tilde{V}(s_t). \tag{2.8}$$

The policy is then updated in the direction of δ_t, i.e.

$$\nabla_\theta J(\theta) = \mathbb{E}_\tau \left[\delta_t \nabla_\theta \log \pi_\theta(u_t|s_t) \right] \tag{2.9}$$

.

2.1.3 Deep Learning in RL

In real-world settings, both observations and agent states are generally high-dimensional and thus the value function does not admit a tabular representation. In this case, the true value function can be estimated using a function approximator. Traditional approaches to value approximation, including affine (Ormoneit and Sen, 2002) or kernel-based functions (Sutton, 1996), partially addressed this difficulty. The advent of deep learning (LeCun et al., 2015) enabled the use of deep neural networks as value function approximators for high-dimensional state and observation spaces, however, it took a few years until the learning process within reinforcement learning could be stabilized (Mnih et al., 2015).

Deep Q-Learning

In the following, we outline how Q-learning (see Section 2.1.2) can be extended to deep function approximators. Deep Q-learning (Mnih et al., 2015, DQN) represents the action-value function with a deep neural network parameterised by θ. A *replay memory* (Lin, 1992b) stores transition tuples $\langle s, u, r, s' \rangle$ from policy rollout. The next state s' is observed after taking the action u in state s and receiving reward r from the environment. θ is learnt by sampling batches of b transitions from the replay memory and minimising the squared *TD error* using gradient descent:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{\mathrm{DQN}} - Q(s_i, u_i; \theta) \right)^2 \right], \tag{2.10}$$

where

$$y_i^{\text{DQN}} = r_i + \gamma \max_{u'} Q(s'_i, u'_i; \theta^-) \tag{2.11}$$

is the so-called *value target*. θ^- are the parameters of a *target network* that are periodically copied from θ and kept constant for a number of iterations. The slowly updated target network, together with the use of experience replay, plays an important role in stabilizing the learning process.

Deep Recurrent Q-Learning

In partially observable settings, agents cannot condition their value functions on the true state of the environment. The best agents can do is to maintain a belief state over the true environment state that conditions on their entire action-observation history. However, such belief states can be impractically large. Hausknecht and Stone (2015) propose *deep recurrent Q-networks* (DRQN) that make use of recurrent neural networks that can efficiently extract summary features from action-observation sequences. Typically, gated architectures such as LSTM (Hochreiter and Schmidhuber, 1997) or GRU (Chung et al., 2014) are used to facilitate learning over longer timescales.

2.2 Multi-Agent Reinforcement Learning

So far in this chapter, we have considered reinforcement learning settings in which a single agent interacts with an environment. We now extend this setting to n agents, identified by $a \in \mathcal{N} \equiv \{1, \dots, n\}$ that choose actions u_t^a, together forming a *joint action* $\mathbf{u} \in \mathcal{U} \equiv \mathcal{U}^n$. As a consequence, the single shared environment undergoes state transitions and emits rewards according to a MDProc (see Section 2.1) with action $\mathcal{U}$. Multi-agent settings can be either *zero-sum*, *general-sum* or *cooperative*. In this thesis, we focus on *fully cooperative* settings, i.e. settings in which all agents receive the same *team* reward, i.e. $r(s, \mathbf{u}, a) = r(s, \mathbf{u}, a'),\ \forall a, a'$.

In settings that are both fully observable and fully cooperative, agents can, in principle, learn a joint stochastic policy

$$\boldsymbol{\pi} : \mathcal{S} \times \mathcal{U} \to [0, 1]. \tag{2.12}$$

Apart from semantics, this reduces the multi-agent problem to a single-agent problem. However, two problems arise in practice: for one, the joint action space $\mathcal{U}$ grows exponentially with the number of agents, which may render greedy action selection, exploration and learning architectures intractable. This suggests to instead learn an exact but factorised joint policy

$$\boldsymbol{\pi}(\mathbf{u}|s_t) = \prod_a \pi^a(u^a|s_t). \tag{2.13}$$

Factorised policies also naturally address a second practical issue: partial observability (see Section 2.1.1) and a lack of reliable communication bandwidth may require agents to learn policies that can be executed based on their own local observations alone. As this thesis is focused on such *decentralised* control settings, we now investigate these in detail.

2.2.1 Decentralised control

Partial observability, together with communication constraints, require that trained agents should be able to operate safely and efficiently in a *decentralised* fashion. *Decentralised execution* signifies that each agent is able to execute its control policy based on its local sensory observations alone. While many real-world robotic agents are equipped with at least short-range, low-bandwidth communication links, solving the fully decentralised setting first and subsequently subjecting it to relaxations may pose a productive starting point in multi-agent systems research.

Decentralised partially observable Markov decision processes (Dec-POMDPs) (Oliehoek and Amato, 2016) formalise cooperative partially-observable multi-agent systems as follows:

A Dec-POMDP is defined by a tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{U}, P, r, \mathcal{Z}, O, \rho, \gamma \rangle$, where $\mathcal{N} := \{1, \dots, n\}$ denotes the set of n agents and $s \in \mathcal{S}$ describes the discrete or continuous state of the environment. The initial state $s_0 \sim \rho$ is drawn from distribution ρ, and at each time step t, all agents $a \in \mathcal{N}$ choose simultaneously discrete or continuous actions $u_t^a \in \mathcal{U}$, yielding the joint action $\boldsymbol{u}_t := \{u_t^a\}_{a=1}^N \in \mathcal{U}^N$. After executing the

joint action $\boldsymbol{u}_t$ in state s_t, the next state $s_{t+1} \sim P(s_t, \boldsymbol{u}_t)$ is drawn from transition kernel P and the collaborative reward $r_t = r(s_t, \boldsymbol{u}_t)$ is returned to the team.

Under partial observability, the true state of the environment cannot be directly observed by the agents. Each agent $a \in \mathcal{N}$ draws an individual observation $z_t^a \in \mathcal{Z}, \boldsymbol{z}_t := \{z_t^a\}_{a=1}^N$, from the observation kernel $O(s_t, a)$.

The history of an agent's observations and actions is denoted by $\tau_t^a \in \mathcal{T}_t := (\mathcal{Z} \times \mathcal{U})^t \times \mathcal{Z}$, and the set of all agents' histories is $\boldsymbol{\tau}_t := \{\tau_t^a\}_{a=1}^N$. Agent a chooses its actions with a decentralised policy $u_t^a \sim \pi(\cdot|\tau_t^a)$ based only on its individual history.

The team of agents aims to learn a *joint policy*

$$\pi(\mathbf{u}|\boldsymbol{\tau}) := \prod_{a=1}^{N} \pi^a(u^a|\tau_t^a) \tag{2.14}$$

that maximises their expected discounted return, $\mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in [0, 1)$ is a discount factor. This joint policy induces a joint action-value function Q^π that estimates the expected discounted return when the agents take joint action $\mathbf{u}_t$ with histories $\boldsymbol{\tau}_t$ in state s_t and then follow some joint policy π through

$$Q^\pi(s_t, \boldsymbol{\tau}_t, \mathbf{u}_t) := \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i r_{t+i}\right]. \tag{2.15}$$

A particular technicality arises around the optimality of joint value functions for Dec-POMDPs. Assuming that optimal value should reflect the value of an optimal joint policy, it turns out that the maximal expected return from a time t onwards is undefined if we do not assume that an optimal joint policy $\boldsymbol{\pi}^*$ has indeed been followed up to t (Oliehoek et al., 2008a, Proposition 4.1). While, in principle, this thesis does make the assumption that all team agents follow an optimal at all times at convergence, it should be noted that an alternative notion of optimality, namely *Sequentially rational* Q^* (Oliehoek et al., 2008a, Theorem 4.4), may be more useful in real-world settings, where agents may deviate from optimal policy execution by mistake.

The decentralisability constraints inherent to Dec-POMDP settings have a striking effect on the complexity of finding optimal policies: Unlike centralised settings, even those where otherwise decentralised agents have limited-bandwidth

communication with a centralised controller, Dec-POMDPs are generally NEXP-complete (Bernstein et al., 2000). This suggests that standard solution methods from centralised control may not be directly transferrable to this setting, opening up a rich space for further research that we explore in this thesis.

2.2.2 Centralised Training with Decentralised Execution (CTDE)

When trained in simulation or in a laboratory, learning algorithms often have access to additional information that is not available at execution. Such centralised training with decentralised execution (CTDE) poses a number of technical challenges to DMARL algorithms that try to exploit the centralised setting in order to facilitate the training of decentralised policies. These difficulties arise primarily from the perceived tension between joint policy learning, which can learn arbitrary policies but is not naively decentralisable and scales poorly with the number of agents, and independent learning, which is readily decentralisable and scalable but provably less expressive and prone to *environment non-stationarity* due to the presence of other learning agents (Busoniu et al., 2008). Another difficulty that independent learning approaches are exposed to in cooperative settings is the problem of *multi-agent credit assignment*, which refers to the challenge of relating the joint team reward signal to individual agent's actions (Tumer and Agogino, 2007; Foerster et al., 2018).

We note in Section 2.1 that, in this thesis, we restrict ourselves to ourselves to the *model-free setting*, in which the agent does not have knowledge of either P or R, nor has access to a model from which it can sample freely. Instead, the agent has to reconstruct these environment statistics through sequential interaction with the environment.

This choice may seem overly restrictive in centralised training settings where unrestricted access to the environment simulator can be assumed, hence allowing to clone (or even freely prepare) simulator states, or arbitrarily control pseudo-random number generators. Making efficient use of these additional capabilities, however, is often not trivial. We note that the emergent field of *model-based multi-agent*

reinforcement learning is a very active area of research that has grown beyond the scope of this thesis. An illustration for how access to the model can be leveraged on top of end-to-end learnt baseline policies is recent work in deep multi-agent reinforcement learning has scaled test-time Monte-Carlo tree search to cooperative multi-agent systems (Lerer et al., 2020, SPARTA).

A more exotic avenue for further research that has perhaps not yet received due attention is based on the insight that having access to the simulator code may inform efficient inductive biases for control as the simulator code itself may be seen to induce a *implicit generative model* that is amenable to likelihood-free inference and related techniques (Rainforth, 2017). A closely related, higly interesting yet under-explored setting is open-source game theory, where software agents may read their own and other software agent's program code (Tennenholtz, 2004).

2.2.3 Independent Q-Learning

Perhaps the most commonly applied method in multi-agent learning is *Independent Q- Learning* (IQL) (Tan, 1993), which decomposes a multi-agent problem into a collection of simultaneous single-agent problems that share the same environment. *Deep Independent Q-learning* (Tampuu et al., 2015, DIQL) is an independent learning algorithm that is used particularly often in CTDE settings. In (Tampuu et al., 2015), each agent a simultaneously learns its own Q-network $Q^a(s, u^a; \theta_i^a)$, from which it independently samples its own actions u_t^a. While DIQL is decentralised by construction, it is not immediately clear how it can benefit from extra state information and communication channels available during centralised training. This approach does not address the non-stationarity introduced due to the changing policies of the learning agents, and thus, unlike joint Q-learning, has no convergence guarantees even in the limit of infinite exploration. In practice, nevertheless, DIQL commonly serves as a surprisingly strong baseline even in mixed and competitive games (Tampuu et al., 2015; Leibo et al., 2017). Chapters 3, 5 and 4 develop novel MARL algorithms that learn decentralised policies, while being able to efficiently exploit CTDE training.

2.3 Dec-POMDPs are intractable, now what?

The fact that Dec-POMDPs are generally NEXP-complete (see section 2.2.1) directly implies that there exist at least some instances of Dec-POMDPs for which no tractable exact learning algorithms exist. In fact, it may even be impossible to closely approximate such solutions in some cases (Rabinovich et al., 2003). This conversely implies that, for such cases, any efficient learning algorithm cannot guarantee an absolute bound on the error. So what can we at all hope to achieve when studying Dec-POMDP learning algorithms? Are we eternally confined to solving all but the tiniest Dec-POMDPs instances?

A perhaps trivial response to this question is to realise that while some Dec-POMDPs are intractable, this does not mean that *all* Dec-POMDP instances of practical interest are NEXP-complete. In fact, the DEC-POMDP instances used to establish general complexity results are arguably *artificial* and far removed from practical application: *TILING* (Papadimitriou, 2003; Bernstein et al., 2000) tries to find a consistent rectangular tiling given a set of compatibility relations, and Multi-Prover Interactive Protocols (MIPs) (Rabinovich et al., 2003) describe a setting in which decentralized set of probabilistic provers jointly try to establish whether a given input language falls within a given family of languages. It does not seem trivial that either of these problem settings is of immediate practical interest to domains such as robotics or self-driving cars.

One might thus hope that hard Dec-POMDPs are confined to special cases of little to no practial interest. A promising research avenue is then to try partitioning the family of all Dec-POMDPs into those few cases that are generally intractable, and those that aren't. Unfortunately, the space of NEXP-complete Dec-POMDPs seems rather large (Allen and Zilberstein, 2009).

In the light of these results, much recent work on Dec-POMDP planning has focused on giving up theoretically guarantees in favour of empirically investigating learning algorithms that can exploit *factorized* Dec-POMDPs, where the joint reward function can be represented by an interaction graph over agent-local reward functions (Oliehoek et al., 2008b). In domains where such factorized substructure

is present, planning algorithms may scale to large numbers of agents while still performing well in practice (Oliehoek et al., 2013). A more detailed overview of related successes is found in section 2.4.

When it comes to deep multi-agent reinforcement learning, forfeiting theoretical guarantees in favour of empirical investigation is currently unavoidable: As the use of deep neural network function approximators upends known reinforcement learning convergence guarantees, any additional error guarantees related to specific Dec-POMDP structure would not result in absolute error bounds for the overall learning algorithm. Instead, one may investigate how learning algorithms with different kinds of inductive biases perform on benchmark environments that reflect properties of real-world tasks of interest. Hereby the ability to learn end-to-end from high-dimensional state representations allows deep multi-agent reinforcement learning benchmarks to more closely resemble control settings with high-dimensional sensory input. Pre-deep learning, such settings were virtually inaccessible.

This thesis introduces a variety of novel multi-agent benchmark suites, including the *StarCraft Multi-Agent Challenge (SMAC)* (Samvelyan et al., 2019; Rashid et al., 2020b) and *Multi-Agent Mujoco (MAMujoco)* (Schroeder de Witt et al., 2020; Peng et al., 2021). Both these benchmark environments exploit that, unlike traditional planning approaches (Oliehoek and Amato, 2016), deep multi-agent reinforcement learning is able to directly learn end-to-end from high-dimensional observations. I terms of closeness to real-world tasks, both SMAC and MAMujoco therefore present a step-change over traditional planning benchmarks, such as *factored firefighting* (Oliehoek et al., 2008b).

We find that both SMAC and MAMujoco benefit from learning algorithms with *value decomposition* (Samvelyan et al., 2019; Rashid et al., 2020b; Schroeder de Witt et al., 2020; Peng et al., 2021), perhaps indicating that at least some real-world tasks benefit from this inductive bias. On the other hand, vanilla value decomposition methods are found to perform suboptimally in the presence of communication games, such as Markov Coding Games introduced in (Sokota et al., 2021). We suggest that developing performant benchmark environments that adequately reflect the relevant

characteristics of real-world tasks of interest, followed by subsequently developing efficient deep multi-agent reinforcement learning algorithms for these tasks, will be crucial to progress in the field. The end goal is to build up a 'registry' of both interesting subclasses of Dec-POMDPs, and their corresponding suitable inductive learning biases and algorithms. This registry could then be used to retrieve building blocks and inspiration for solving novel problems of interest.

It should be noted that scalable Dec-POMDP planning approaches could serve as independent testbeds for deep multi-agent reinforcement learning methods and yield valuable insights on an algorithm's workings. The multi-agent learning community should therefore welcome progress in this field.

2.4 A short history of Value Factorisation

Both Chapters 3 and 4 introduce methods based on *value factorisation*. Value factorisation has a rich history within the multi-agent planning community. We here provide a brief overview of relevant developments.

Factored value functions were initially explored in order to make coordination in planning settings with centralised execution more tractable. The first notion of *Factored Value Functions (FVFs)* was introduced in (Guestrin et al., 2002c). FVFs simplify coordination over joint action spaces by introducing *coordination graphs*. This idea was subsequently extended to (centralised) policy-gradient methods in the framework of *coordinated reinforcement learning* (Guestrin et al., 2002b).

Kok and Vlassis (2004) *sparse cooperative Q-learning*, which scales to coordination graphs of high induced width. Their resulting *max-sum* method was echoed by Farinelli et al. (2008) and subsequently further developed in (Farinelli et al., 2009).

Kok et al. (2005) introduce *coordination dependencies*, which allow coordination graphs to admit state-dependency. A number of other works explore concepts that may be regarded as similar to context-specific factorisation, including (Melo and Veloso, 2009; De Hauwere et al., 2010, 2011). An empirical demonstration of the utility of state-dependent coordination graphs was supplied by Bakker et al. (2010) in the context of traffic light control.

While *Independent Q-learning* (Tan, 1993, IQL) (see section 2.2.3) may be regarded as the first special case of value factorisation under decentralised execution, Nair et al. (2005) were the first to introduce factored value functions to Dec-POMDPs, albeit only a restricted subspace called *ND-POMDPS*. The first analysis of factored value functions for general factored Dec-POMDPs was conducted by (Oliehoek et al., 2008b), exploiting exact last-stage factorisation.

While allowing for a compact problem representation, Dec-POMDPs nevertheless share worst-case NEXP-completeness with general Dec-POMDPs. Pajarinen and Peltonen (2011) introduced an expectation-maxmization based approach that allows planning in factored Dec-POMPDs to be scaled to larger numbers of agents.Kumar and Zilberstein (2010); Kumar et al. (2011) provide a rigorous axiomatisation of Dec-POMDPs for which the value function factorization holds and, again, propose a solution algorithm based on expectation-maximisation. Techniques that enable value factorisation coordination graphs for factored Dec-POMDPs to be learnt were developed by (Yeoh et al., 2013).

Following these mostly theoretical developments, a number of works sought to better understand under what conditions factorisation arises in multi-agent tasks. Oliehoek et al. (2012b) used *influence-based abstraction (IBA)* to show that, for sufficiently *local* models, near-exact factorisation can be expected as long as history-dependence is allowed. The analysis was subsequently expanded in (Oliehoek et al., 2021). Oliehoek et al. (2015) showed that, using *optimistic influences*, factored upper bounds for value functions can be derived even if Dec-POMDPs do not admit exact factored value functions. *Type independence*, a different type of factorisation, was identified in certain Bayesian games and Dec-POMDPs by (Oliehoek et al., 2012a). *Factored-value POMCP* (Amato and Oliehoek, 2015) exploits value factorisation in online planning (Silver and Veness, 2010). Choudhury et al. (2021) further develop this idea.

The first demonstration of factored value-functions (FVFs) in deep multi-agent reinforcement learning was made in (Pol and Oliehoek, 2016).Böhmer et al. (2020) scaled centralised coordination graphs to end-to-end deep multi-agent reinforcement

learning. An empirical investigation of the ability of factored value functions to learn non-factored payoff function was conducted by (Castellini et al., 2021). In particular, it was found that *high-order factorisations* with multiple agents can improve the accuracy of action-value function representation substantially , and that random overlapping factors can perform surprisingly well.

In the light of this rich history, *VDN*(Sunehag et al., 2018) may be seen as 'individually-factorised FVFs in one computational graph', whereas *QMIX* (Rashid et al., 2020b) employs monotonic factorisation with history-based local Q-components. Acccording to the theory of *influence-based abstraction (IBA)* (Oliehoek et al., 2012b, 2021), such history-based local Q-components may in fact lead to accurate representations for Dec-POMDP planning and learning, hence the empirical success of VDN and QMIX on certain domains may not be surprising.

The type of policy factorisation employed in MACKRL (Schroeder de Witt et al., 2019) does not seem to squarely fit into historical frameworks. A distantly related precedent can perhaps be found in *coordinated reinforcement learning*Guestrin et al. (2002b), although the policy factorisation presented here is not decentralisable. Another observation is that the factorisations employed in MACKRL with field-of-view common knowledge (see section 2.7) exhibit a natural kind of *locality*, perhaps suggesting that a policy-side extension of IBA theory might shed light on its empirical success.

Further work related to QMIX, FACMAC and MACKRL, including additional contemporary developments, are discussed in the respective chapters 3, 4, and 5.

2.5 Bayesian multi-agent learning

Although not central to this thesis, there also exist Bayesian approaches to multi-agent reinforcement learning. While often posing scalability challenges, Bayesian approaches admit a more heuristic frameworks for trading off exploration and exploitation (Ghavamzadeh et al., 2015).

Among tabular approaches, Amato and Oliehoek (2013) investigate Bayesian RL for best-response models. Building up on *Bayes-Adaptive POMDPs (BA-POMDPs)*

(Guez et al., 2012), Amato and Oliehoek (2015) tackle Bayesian reinforcement learning for multi-agent POMDPs. Subsequent work has focused on scaling up single-agent BA-POMDPs (Katt et al., 2017, 2019), although such work has not been extended to function approximation with deep neural networks. A notable attempt solving single-agent BA-MDPs with deep reinforcement learning has been made by (Zintgraf et al., 2020, VariBAD).

2.6 Implicit communication

A major focus point of this thesis is to investigate the relationship between two different forms of communication between agents, namely *explicit* and *implicit* communication. For a concise definition, we reproduce (Oliehoek and Amato, 2016, Definition 31):

When a multiagent decision framework has a separate set of communication actions, we say that it supports explicit communication. Frameworks without explicit communication can and typically do still allow for implicit communication: the act of influencing the observations of one agent through the actions of another

Real-world constraints can render explicit communication unreliable (or at all unavailable) for to a number of reasons: first of all safety constraints over sensor failure, security constraints as adversaries may intercept or spoof communication signals, resource constraints related to battery or equipment budget and lastly, physical constraints due to terrain, weather or signal interference. In such situations, or in addition to communication through reliable channels, agents may be able to rely on *implicit communication*, i.e. communication achieved through observing each other's actions directly, or their effects on the environment.

As a real-world illustration of implicit communication, bees indicating the location and distance to food sources through waggle dances to others communicate through their actions being observed by other bees, while a lost mountaineer writing "SOS" in the snow in the hope that rescuers will read it from the air is effecting the environment in a way that can be perceived by other agents (who do not observe the mountaineer necessarily while she writes the message).

There are many benefits to being able to learn how to communicate implicitly. First of all, one may avoid the installation cost or energy requirements of extra communication equipment. Secondly, implicit communication may be robust to signal interference and jamming in the (non-visual) electromagnetic spectrum. Implicit communication may increase communication bandwidth on top of available cheap talk channels, or provide additional redundancy under signal noise or interference. Choosing to communicate implicitly may also allow masking the very act of communication, which may give rise to plausible deniability in settings in which cooperation between agents needs to be concealed.

Despite its advantages, implicit communication also has some important limitations: As actions are used simultaneously for communication and to manipulate the environment, implicit communication may interfere with task execution. Reversely, task execution may greatly constrain the amount of information to be communicated. Environment modifications for the purpose of implicit communication might not be detected in time by the other agents.

In Chapter 6, we introduce GME, a novel algorithm that allows agents to efficiently encode information in their trained policies.

2.7 Common Knowledge

Throughout this thesis, but in particular in Chapter II, a particular form of *group knowledge* plays an important part:

Common knowledge for a group of agents consists of facts that all agents know and "each individual knows that all other individuals know it, each individual knows that all other individuals know that all the individuals know it, and so on" (Osborne and Rubinstein, 1994). This may arise in a wide range of multi-agent problems, e.g., whenever a reliable communication channel is present. But common knowledge can also arise without communication, if agents can infer some part of each other's observations.

The process in which common knowledge ξ arises among a group of agents can be modeled as an infinite recursion: Let ξ_0 be a set of information known to all agents in a group and define ξ_{i+1} as the subset of ξ_i that is known by all agents to be in ξ_i. ξ is then *commonly known* between agents in $\mathcal{G}$ if

$$\xi := \bigcap_{i=1}^{\infty} \xi_i \tag{2.16}$$

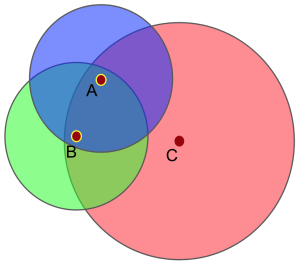


Figure 2.1: Three agents and their fields of view. A and B's locations are common knowledge to A and B as they are within each other's fields of view. Although C can see A and B, it shares no common knowledge with them.

Any data ξ that are known to all agents before execution/training, like a shared random seed, are obviously common knowledge. For example, practically any cooperative learning setting assumes that each agent's policy is commonly known among all agents during execution.

Crucially, every agent $a \in \mathcal{G}$ can deduce the same *history of common knowledge* $\tau_t^{\mathcal{G}}$ from its own history τ_t^a and the commonly known data ξ, that is, $\tau_t^{\mathcal{G}} := \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi) = \mathcal{I}^{\mathcal{G}}(\tau_t^{\bar{a}}, \xi), \forall a, \bar{a} \in \mathcal{G}$. Furthermore, any actions taken by a policy $\pi^{\mathcal{G}}(\mathbf{u}_{\text{env}}^{\mathcal{G}} | \tau_t^{\mathcal{G}})$ over the group's joint action space $\mathcal{U}_{\text{env}}^{\mathcal{G}}$ are themselves common knowledge, if the policy is deterministic or pseudo-random with a shared random seed and conditions only on the common history $\tau_t^{\mathcal{G}}$, i.e. the set formed by restricting each transition tuple within the joint history of agents in $\mathcal{G}$ to what is commonly known in $\mathcal{G}$ at time t. Common knowledge of subgroups $\mathcal{G}' \subset \mathcal{G}$ cannot decrease, that is, $\mathcal{I}^{\mathcal{G}'}(\tau_t^a, \xi) \supseteq \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi)$.

Common knowledge can arise through a great variety of physical or conceptual mechanisms. For example, if each agent can reliably observe objects within its field of view and the agents know each other's fields of view, then they share common knowledge whenever they see each other. This setting is illustrated in Figure 2.1 and applies to a range of real-world scenarios, for example, to robo-soccer (Genter et al., 2017), fleets of self-driving cars and multi-agent StarCraft micromanagement (Synnaeve et al., 2016). This type of common knowledge is formalised as *common knowledge with entities* in Chapter 5.3.

Another example of common knowledge arising between agents is the popular card game Hanabi (Bard et al., 2020). In Hanabi, agents can only observe other agents' hands, but not their own. One of the most successful hand-coded Hanabi bots, (Cox et al., 2015; Wu, 2018, WTFWThat), uses a *hat coding* mechanism that derives *public* knowledge, i.e. knowledge that is commonly known between all agents without requiring further deduction, from common knowledge between pairs of agents. So far, MARL-based approaches to learning optimal policies for Hanabi agents explicitly exploit common knowledge that arises simultaneously for all agents, but not common knowledge that arises for smaller agents subgroups (Foerster et al., 2019; Hu and Foerster, 2020; Hu et al., 2020), with the exception of (Lerer et al., 2020, SPARTA), which combines search with arbitrary blueprint policies.

Common knowledge has been studied extensively in a variety of contexts, including philosophy and game theory. Examples of applications to multi-agent systems include Thomas et al. (2014), who explore the psychology of common knowledge and coordination. Importantly, multi-agent coordination sometimes cannot rely on lower-order group knowledge: Rubinstein (1989) shows that any finite number of reasoning steps, short of the infinite number required for common knowledge, can be insufficient for achieving coordination (see Appendix 5.3).

Nayyar et al. (2013) show that common knowledge can be used to reformulate decentralised planning problems as POMDPs (see section 2.1.1) to be solved by a central coordinator using dynamic programming. Foerster et al. (2019, BAD) subsequently propose a method for scaling this to learning beliefs over high-dimensional state spaces. However, BAD does not trivially scale to continuous state spaces and dynamic groupwise common knowledge hierarchies. In Chapter 5, we introduce MACKRL (Schroeder de Witt et al., 2019), an entirely model-free multi-agent algorithm that exploits common knowledge at different levels of the agent group hierarchy and learns trivially decentralisable control policies end-to-end without the computational expense of explicit belief construction.

In the absence of common knowledge, complex decentralised coordination has to rely on implicit communication (see section 2.6), i.e., observing each other's

actions or their effects (Heider and Simmel, 1944; Rasouli et al., 2017). However, implicit communication protocols for complex coordination problems are difficult to learn and, as they typically require multiple timesteps to execute, can limit the agility of control during execution (Tian et al., 2020) (see section 2.6). By contrast, coordination based on common knowledge is *simultaneous*, that is, does not require learning communication protocols (Halpern and Moses, 2000).

Part I

Value Decomposition

3

Monotonic Value Factorisation for Deep Multi-Agent Reinforcement Learning

Contents

3.1 Introduction

In Chapter 1, we discussed that many important control problems, including autonomous driving (Cao et al., 2013), swarm control (Barnes et al., 2009), robotics (Seuken and Zilberstein, 2007), communication networks (Peshkin, 2000), crisis management (Paquet et al., 2005), recreational games(Bard et al., 2020) and load balancing (Cogill et al., 2006), fall within the paradigm of *centralised training with decentralised execution* introduced in Section 2.2.2. In this Chapter, we tackle various open challenges surrounding how to best exploit centralised training in CTDE settings.

Central to this Chapter is the concept of *value function factorisation*, i.e. the assumption that the *joint action-value function for the system can be additively decomposed into value functions across agents*(Sunehag et al., 2018). Since publication of (Rashid et al., 2020b), significant progress has been made in understanding under what conditions such an assumption may be reasonable. In fact, Oliehoek et al. (2021) show that for any *factored Dec-POMDP* for which a set of *local form models* (Oliehoek et al., 2021, Definition 10) can be created, the joint *value* function can indeed be decomposed into per-agent factors as long as each factor conditions on a sufficient statistic of respective agent's belief state. While this notion does not directly carry over to *state-action* value functions, the empirical

success of methods based on factored state-action value functions on environments like SMAC (Rashid et al., 2020b) or Multi-Agent Mujoco (Peng et al., 2021) show it to approximately hold across a number of real-world setups. For a more detailed discussion of why this may be and why this assumption may break down for more tightly-coupled environments and off-policy reinforcement learning methods, we refer to (Oliehoek et al., 2021, Section 8.1).

We now quote from (Rashid et al., 2020b), with the quoted text being visually indicated by reduced linespacing. Please note that the quote is literal except for where original supplementary material and appendices have been inserted in-place and internal document references have been updated accordingly.

One of these challenges is how to represent and use the action-value function that many RL methods learn. On the one hand, properly capturing the effects of the agents' actions requires a centralised action-value function Q_{tot} that conditions on the global state and the joint action. On the other hand, such a function is 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) (Tan, 1993), see Section 2.2.3. 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} 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 (Foerster et al., 2018), as well as work by Gupta et al. (2017). 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) (Sunehag et al., 2018). 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:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a. \tag{3.1}$$

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.

For a concise definition of the consistency condition please refer to Equation 3.3.

At the same time, it enforces the constraint of (3.1) by restricting the mixing network to have positive weights. We use *hypernetworks* (Ha et al., 2017) 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 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 (Foerster et al., 2018; Sunehag et al., 2018), we introduce the StarCraft Multi-Agent Challenge (SMAC)[1]. In single-agent RL, standard environments such as the Arcade Learning Environment (Bellemare et al., 2013) and MuJoCo (Plappert et al., 2018) have facilitated rapid progress. While some multi-agent testbeds have emerged, such as Poker (Heinrich and Silver, 2016), Pong (Tampuu et al., 2015), Keepaway Soccer (Stone et al., 2005), or simple gridworld-like environments (Lowe et al., 2017; Leibo et al., 2017; Yang et al., 2018; Zheng et al., 2017), there are currently no challenging standard testbeds for centralised training with decentralised execution with the exception of the recently introduced Hanabi Challenge (Bard et al., 2020) 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 (Vinyals et al., 2017). Instead of tackling the full game of StarCraft with centralised control, it focuses on decentralised micromanagement challenges (Figure 3.1). 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

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

(a) 3 Stalkers vs 5 Zealots (b) 2 Collosi vs 64 Zerglings

Figure 3.1: *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.

single task 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* (Rabinowitz et al., 2018)).

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 3.6).

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. (2018), QMIX has become an important value-based algorithm in discrete action environments. It has inspired a number of extensions and follow-up work (Section 3.2.4) and is a prominent point of comparison in any work that utilises SMAC (Section 3.2.5). 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.[2]

[2]This work was originally published as a conference paper (Rashid et al., 2018). The main additions in this article are:

- Introducing SMAC, a benchmark for the cooperative multi-agent RL setting of centralised training and decentralised execution (Section 3.6).
- Releasing PyMARL, a framework for running and developing multi-agent RL algorithms

3.2 Related Work

Recent work in multi-agent RL has started moving from tabular methods (Yang and Gu, 2004; Busoniu et al., 2008) to deep learning methods that can tackle high-dimensional state and action spaces (Tampuu et al., 2015; Foerster et al., 2018; Peng et al., 2017; Schroeder de Witt et al., 2019). In this paper, 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.

3.2.1 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* (Tan, 1993) trains independent action-value functions for each agent using Q-learning (Watkins, 1989). Tampuu et al. (2015) extend this approach to deep neural networks using DQN (Mnih et al., 2015). 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. (2017) addresses the issue of non-stationarity when using an experience replay with independent learners to some extent. Lauer and Riedmiller (2000) 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. (2007) 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. (2017) utilises Hysteretic Q-learning in a multi-task deep RL setting. Panait et al. (2008) introduce *Leniency* which ignores updates that decrease Q-value estimates with a probability that is decreasing during training. Wei and Luke (2016) show that Leniency outperforms Hysteretic Q-learning in cooperative stochastic games, and Palmer et al. (2017) 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. (2019) 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

(Section 3.7).

- Experimental results comparing IQL, VDN, COMA, and QTRAN on the SMAC benchmark (Section 3.8).
- Further analysis and ablation experiments to investigate why QMIX outperforms VDN (Sections 3.5 and 3.9).
- Comprehensive literature review of classical as well as novel cooperative multi-agent RL approaches (Section 3.2).

to distinguish between miscoordination between the agents and the stochasticity of the environment, which Hysteretic and Lenient learners do not distinguish between. Lyu and Amato (2019) utilise Distributional RL (Bellemare et al., 2017) 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.

3.2.2 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* (Guestrin et al., 2002a), 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* (Kok and Vlassis, 2006) 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. (2019) investigate the representational capacity of coordination graphs in a deep RL setting using one-step matrix games. Böhmer et al. (2020) 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. Chen et al. (2018) 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.

DIAL (Foerster et al., 2016) utilises an end-to-end differentiable architecture that allows for a learned inter-agent communication to emerge via backpropagation. CommNet (Sukhbaatar et al., 2016) use a centralised network architecture to exchange information between agents. BicNet (Peng et al., 2017) utilise bidirectional RNNs for inter-agent communication in an actor-critic setting and additionally requires estimating individual agent rewards. MAGNet (Malysheva et al., 2018) 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 (2019) 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) (Pageaud et al., 2019) 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.

3.2.3 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. (2017) present a centralised actor-critic algorithm with per-agent critics, which scales easily with the number of agents. Similarly, Lowe et al. (2017) 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. (2017), 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 (Foerster et al., 2018) 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 (2019) 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. Schroeder de Witt et al. (2019) 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) (Du et al., 2019) 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.

Lin et al. (2019) 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. (2018) 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. (2019) 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.

3.2.4 Extensions of QMIX

Mahajan et al. (2019) 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 (Osband et al., 2016). This results in significant performance gains on some scenarios in SMAC (Samvelyan et al., 2019).

Action Semantics Networks (ASN) (Wang et al., 2019a) 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.

Wang et al. (2020c) 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. (2019b) 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) (Yao et al., 2019) 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.

Liu et al. (2019b) investigate transfer learning algorithms in a multi-agent setting using QMIX as a base. Yang et al. (2019) use DIAYN (Eysenbach et al., 2018) to learn lower-level skills that a higher level QMIX agent uses. Fu et al. (2019) 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 (Hüttenrauch et al., 2018) and stratospheric geoengineering (de Witt and Hornigold, 2019).

QMIX has been extended from discrete to continuous action spaces (Peng et al., 2021, COMIX). COMIX uses the cross-entropy method to approximate the otherwise intractable greedy per-agent action maximisation step. Peng et al. (2021) 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.

3.2.5 PyMARL and SMAC

A number of papers have established unit micromanagement in StarCraft as a benchmark for deep multi-agent RL. Usunier et al. (2016) present an algorithm using a centralised *greedy MDP* and first-order optimisation which they evaluate on Starcraft: BroodWar (Synnaeve et al., 2016). Peng et al. (2017) also evaluate their methods on StarCraft. However, neither requires decentralised execution. Similar to our setup in SMAC is the work of Foerster et al. (2017), who evaluate replay stabilisation methods for IQL on combat scenarios with up to five agents. Foerster et al. (2018) also uses this setting.

In this paper, we construct unit micromanagement tasks using the *StarCraft II Learning Environment* (SC2LE) (Vinyals et al., 2017) 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 (Vinyals et al., 2017), and DeepMind's AlphaStar (Vinyals et al., 2019) 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 (Schroeder de Witt et al., 2019; Wang et al., 2019b; Du et al., 2019; Wang et al., 2020c; Zhang et al., 2019b; Mahajan et al., 2019; Yao et al., 2019; Böhmer et al., 2020; Wang et al., 2019a).

3.2.6 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* (Ha et al., 2017). This second neural network is constrained to be monotonic with respect to its inputs by keeping its weights positive. Dugas et al. (2009) also investigate such functional restrictions for neural networks.

3.2.7 Value Decomposition Networks

By contrast, *value decomposition networks* (VDNs) (Sunehag et al., 2018) 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:

$$Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \sum_{i=1}^{n} Q_i(\tau^i, u^i; \theta^i). \tag{3.2}$$

Strictly speaking, each Q_a is a *utility function* (Guestrin et al., 2002a) 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 (2.10), where Q is replaced by Q_{tot}. An advantage of this representation is that a decentralised policy arises simply from each agent performing greedy action selection with respect to its Q_a.

3.3 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:

$$\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \tag{3.3}$$

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 (3.3) 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 (3.3). However, QMIX is based on the observation that this representation can be generalised to the larger family of monotonic functions that also satisfy (3.3). Monotonicity in this context is defined as a constraint on the relationship between Q_{tot} and each Q_a:

$$\frac{\partial Q_{tot}}{\partial Q_a} \geq 0, \ \forall a \in A, \tag{3.4}$$

which is sufficient to satisfy (3.3), as the following theorem shows.

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

$$\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}) = \begin{pmatrix} \arg\max_{u^1} Q_1(\tau^1, u^1) \\ \vdots \\ \arg\max_{u^n} Q_n(\tau^n, u^n) \end{pmatrix}. \tag{3.3}$$

[A] The following section is taken from the paper's appendix.

Proof. Since $\frac{\partial Q_{tot}}{\partial Q_a} \geq 0$ for $\forall a \in A$, the following holds for any $(u^1, \ldots, u^n)$ and the

mixing network function $Q_{tot}(\cdot)$ with n arguments:

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

Therefore, the maximiser of the mixing network function is:

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

Thus,

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

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

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

Hence, $\mathbf{u}_* = \arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u})$, which proves (3.3). □

Figure 3.2 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

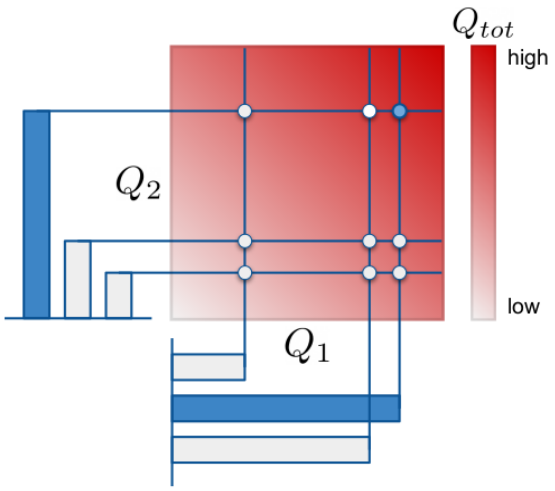


Figure 3.2: The discrete per-agent action-value scores Q_a are fed into the monotonic function $Q_{tot}(Q_1, Q_2)$. The maximum Q_a for each agent is shown in blue, which corresponds to the maximum Q_{tot} also shown in blue. The constraint (3.3) is satisfied due to the monotonicity of Q_{tot}.

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 (3.4) in practice, by representing Q_{tot} using an architecture consisting of *agent networks*, a *mixing network*, and a set of *hypernetworks* (Ha et al., 2017). Figure 3.3 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 3.3c. 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 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 3.3a. To enforce the monotonicity constraint of (3.4), 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 (Dugas et al., 2009).

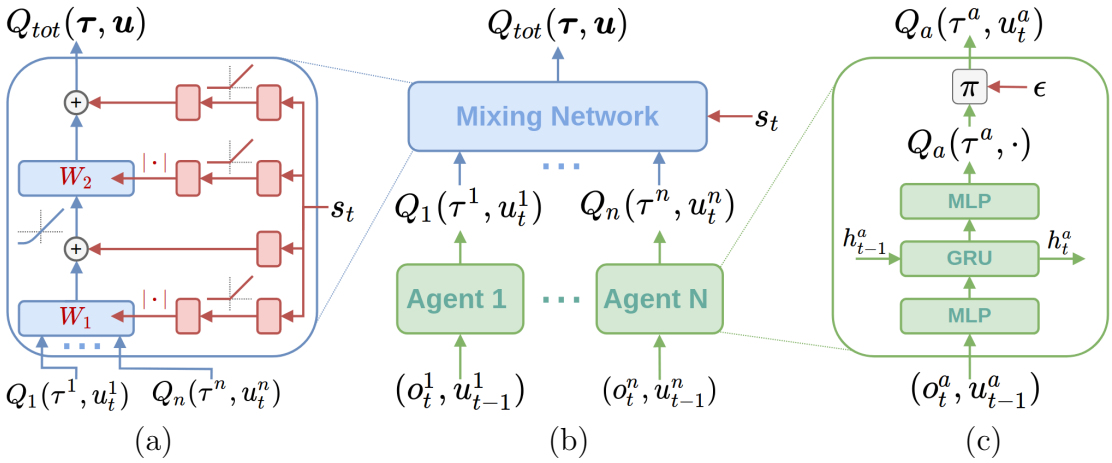


Figure 3.3: (a) Mixing network structure. In red are the hypernetworks that produce the weights and biases for mixing network layers shown in blue. (b) The overall QMIX architecture. (c) Agent network structure. Best viewed in colour.

The 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 3.3a 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:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \left[\left(y_i^{tot} - Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \theta) \right)^2 \right], \tag{3.5}$$

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. (3.5) is analogous to the standard DQN loss of (2.10). Since (3.3) 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.

Algorithm 1 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
6: **while** $s_t \neq terminal$ and $t <$ episode limit **do**
7: **for** each agent a **do**
8: $\tau_t^a = \tau_{t-1}^a \cup \{(o_t, u_{t-1})\}$
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: Get reward r_t and next state s_{t+1}
12: $\mathcal{D} = \mathcal{D} \cup \{(s_t, \mathbf{u}_t, r_t, s_{t+1})\}$
13: $t = t + 1$, step $=$ step $+ 1$
14: **if** $|\mathcal{D}| >$ batch-size **then**
15: b $\leftarrow$ random batch of episodes from $\mathcal{D}$
16: **for** each timestep t in each episode in batch b **do**
17: $Q_{tot} =$ *Mixing-network* $(Q_1(\tau_t^1, u_t^1), \ldots, Q_n(\tau_t^n, u_t^n);$ *Hypernetwork*$(s_t; \theta))$
18: Calculate target Q_{tot} using *Mixing-network* with *Hypernetwork*$(s_t; \theta^-))$
19: $\Delta Q_{tot} = y^{tot} - Q_{tot}$ // Eq (3.5)
20: $\Delta\theta = \nabla_\theta (\Delta Q_{tot})^2$
21: $\theta = \theta - \alpha \Delta\theta$
22: **if** update-interval steps have passed **then**
23: $\theta^- = \theta$

In Algorithm 1 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 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.

3.3.1 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 (Dugas et al., 2009), 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 (Pinkus, 1999), 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 the true optimal Q-function. 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 3.1a 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 3.2a provides the optimal Q_{tot} values for VDN, minimising a squared loss. Section 3.4 additionally provides results for the values that VDN learns in a function approximation setting.

However, the constraint in (3.4) prevents QMIX from representing value functions that do not factorise in such a manner. A simple example of such a value function for a two-agent matrix game is given in Table 3.1b. 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 3.2b provides the optimal Q_{tot} values for QMIX when minimising a squared loss. For the example in Table 3.1b, QMIX still recovers the optimal policy and learns the correct maximum over the Q-values. Formally, the class of value

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 3.1: (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 3.2: (a) Optimal Q_{tot} values for VDN on the example in Table 3.1a. (b) Optimal Q_{tot} values for QMIX on the example in Table 3.1b.

functions that cannot be represented by QMIX are called *nonmonotonic* (Mahajan et al., 2019).

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 3.4 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 3.5 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 3.8 that QMIX significantly outperforms VDN on the challenging SMAC benchmark.

3.4 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 3.4.

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.

3.4.1 Architecture and Training [A]

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.

3.4.2 Results

Figure 3.4 shows the loss for the different methods. Table 3.3 shows the final testing reward for each method.

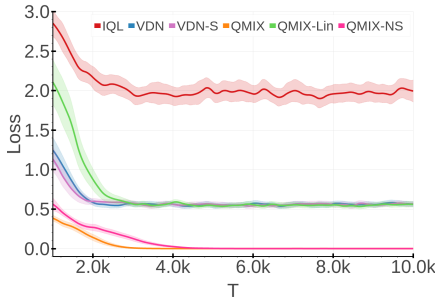


Figure 3.4: Loss for all six methods on the Two Step Game. The mean and 95% confidence interval is shown across 30 independent runs.

IQL	VDN	VDN-S	QMIX	QMIX-Lin	QMIX-NS
7	7	7	8	7	8

Table 3.3: The final Test Reward acheived.

Table 3.5, 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

Agent 1 \ Agent 2	A	B
A	7	7
B	7	7

State 2A

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

State 2B

Table 3.4: 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 3.5: Q_{tot} on the two-step game for (a) VDN and (b) QMIX.

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 3.5 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.

[A] The following section is taken from the paper's appendix.

3.4.3 Learned Value Functions [A]

The learned value functions for the different methods on the Two Step Game are shown in Tables 3.6 and 3.7.

[A] The following section is taken from the paper's appendix.

(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 3.6: Q_{tot} on the 2 step game for (a) VDN, (b) QMIX, (c) QMIX-NS, (d) QMIX-Lin and (e) VDN-S

	A	*B*	*A*	*B*	*A*	*B*
Agent 1	6.96	4.47	6.98	7.00	0.50	4.50
Agent 2	5.70	5.78	7.00	7.02	0.50	4.47

Table 3.7: Q_a for IQL on the 2 step game

3.5 Random Matrix Games [A]

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.

Note that all matrix games studied in this paper are single-step, rather than repeated, games.

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.

3.5.1 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.

Figure 3.5 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.

3.6 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

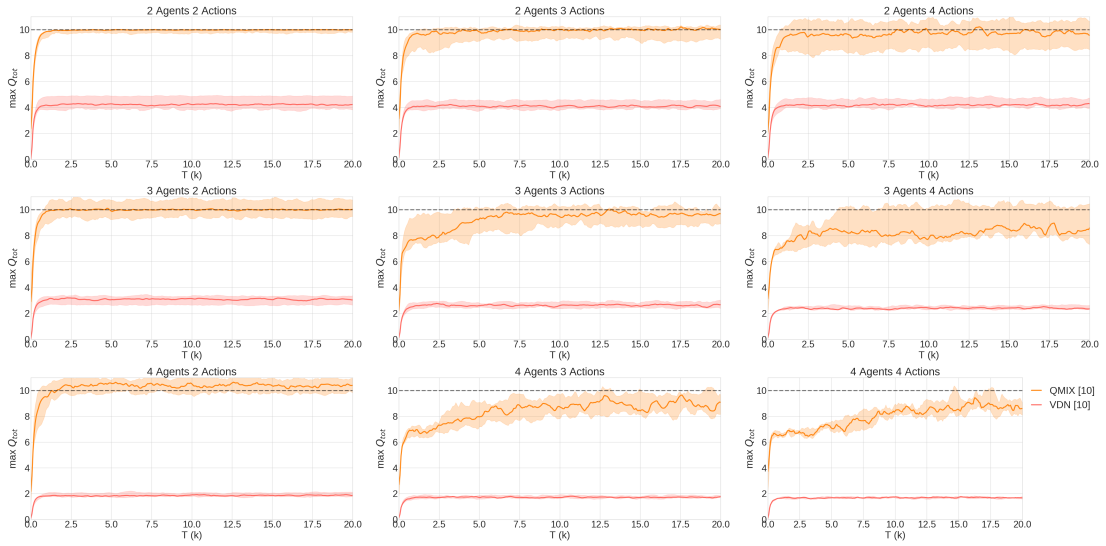


Figure 3.5: The median $\max_{\mathbf{u}} Q_{tot}(s, \mathbf{u})$ for $\{2, 3, 4\}$ agents with $\{2, 3, 4\}$ actions across 10 runs for VDN and QMIX. 25%-75% quartile is shown shaded. The dashed line at 10 indicates the correct value.

II, one 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 on that unit (see Figure 3.6). 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

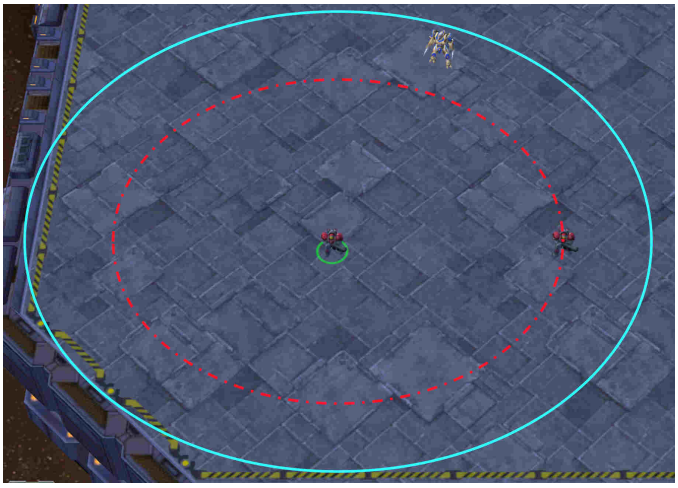


Figure 3.6: The cyan and red circles respectively border the sight and shooting range of the agent.

(a) `MMM2` (b) `corridor`

Figure 3.7: Screenshots of two SMAC scenarios.

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 3.1a), 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.

3.6.1 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 3.1 and 3.7 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 3.8.

[A] The following section is taken from the paper's appendix.

3.6.2 Scenarios [A]

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 3.7d). 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 3.1a). 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 (Blizzard Entertainment, 2012).

3.6.3 Environment Setting [A]

SMAC makes use of the StarCraft II Learning Environment (*SC2LE*) (Vinyals et al., 2017) 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

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

heuristics.

3.6.4 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 3.6). 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 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.

3.6.5 Action Space

The discrete set of actions that agents are allowed to take consists of `move[direction]`[3], `attack[enemy_id]`, `stop` and `no-op`.[4] 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 3.6). 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.

3.6.6 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 comparisions we encourage using this default reward function for all scenarios. We also provide another *sparse reward* option, in which the reward is +1 for winning and -1 for losing an episode.

3.6.7 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.

[3] Four directions: north, south, east, or west.

[4] Dead agents can only take `no-op` action while live agents cannot.

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.

3.6.8 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.

3.7 PyMARL

To make it easier to develop algorithms for SMAC, we have also open-sourced our software engineering framework PyMARL[5]. Compared to many other frameworks for deep reinforcement learning (Stooke and Abbeel, 2019; Hill et al., 2018; Hoffman et al., 2020), 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 (Liang et al., 2018). 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.

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

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 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 (Son et al., 2019), COMA (Foerster et al., 2018), VDN (Sunehag et al., 2018), and IQL (Tan, 1993) as baselines.

3.8 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 (Rashid et al., 2018). 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*.

A table of the results is included in Table 3.8.2. Data from the individual runs are available at the SMAC repository (`https://github.com/oxwhirl/smac`).

[A] The following section is taken from the paper's appendix.

3.8.1 Architecture and Training [A]

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 (Van Hasselt et al., 2016) 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[6] 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 (Foerster et al., 2018) 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 (Son et al., 2019). 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

[6] We set $\alpha = 0.99$ and do not use weight decay or momentum.

$(64 + |U|$ units). The network is shared across all agents. The embeddings are summed across all agents. The concatentation 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 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.

3.8.2 Table of Results [A]

Table 3.9 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.

Figure 3.8 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 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. Additionally, IQL, VDN, and QMIX all

	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 3.9: The Test Win Rate % of the various algorithms.

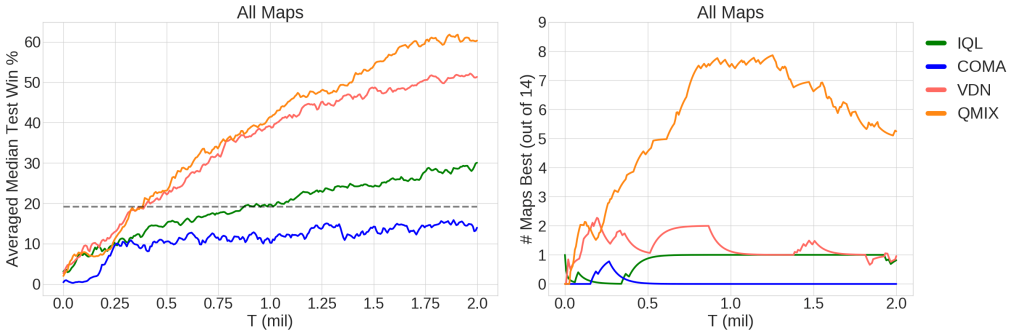


Figure 3.8: Left: The median test win %, averaged across all 14 scenarios. Heuristic's performance is shown as a dotted line. Right: The number of scenarios in which the algorithm's median test win % is the highest by at least 1/32 (smoothed).

significantly outperform COMA, demonstrating the sample efficiency of off-policy value-based methods over on-policy policy gradient methods.

We also compare to QTRAN on 3 scenarios in Figure 3.9. 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 3.8.1.

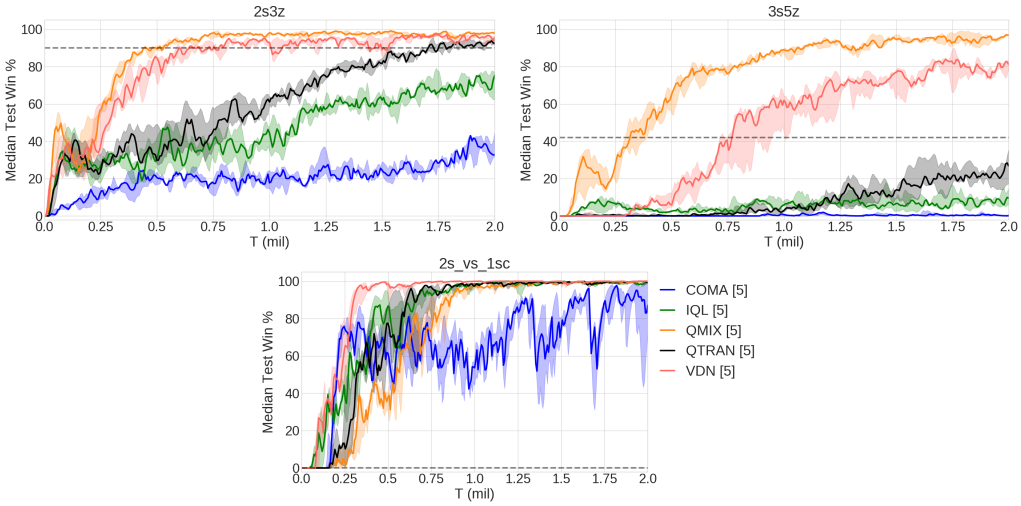


Figure 3.9: Three scenarios including QTRAN.

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

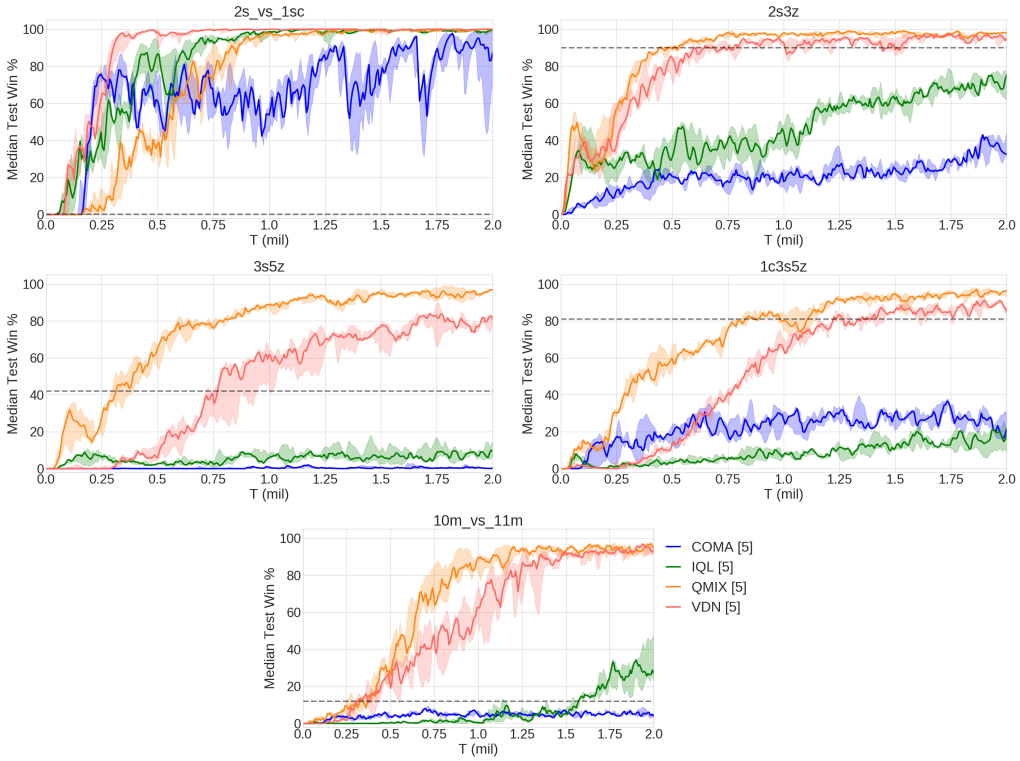


Figure 3.10: Easy scenarios. The heuristic AI's performance shown as a dotted black line.

Figure 3.10 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.

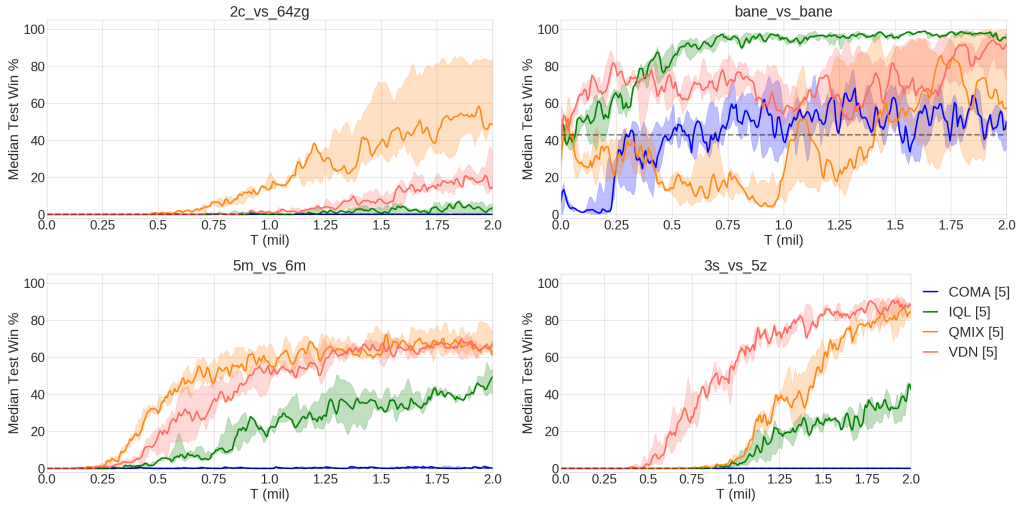


Figure 3.11: Hard scenarios. The heuristic AI's performance shown as a dotted black line.

The *Hard* scenarios in Figure 3.11 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 (QMIX and VDN) have plateaued in performance. Finally, *3s_vs_5z* requires the three allied stalkers to *kite* 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 3.12 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.

Note that the difficulty of the scenarios is not measured relative to the performance of a single learning algorithm, but across the whole ensemble of algorithms tested. For example, IQL can solve some of the Hard scenarios, while it may fail on easier ones.

As stated in 3.6.8, the shaded areas in the results plots indicate the 25-75% percentiles.

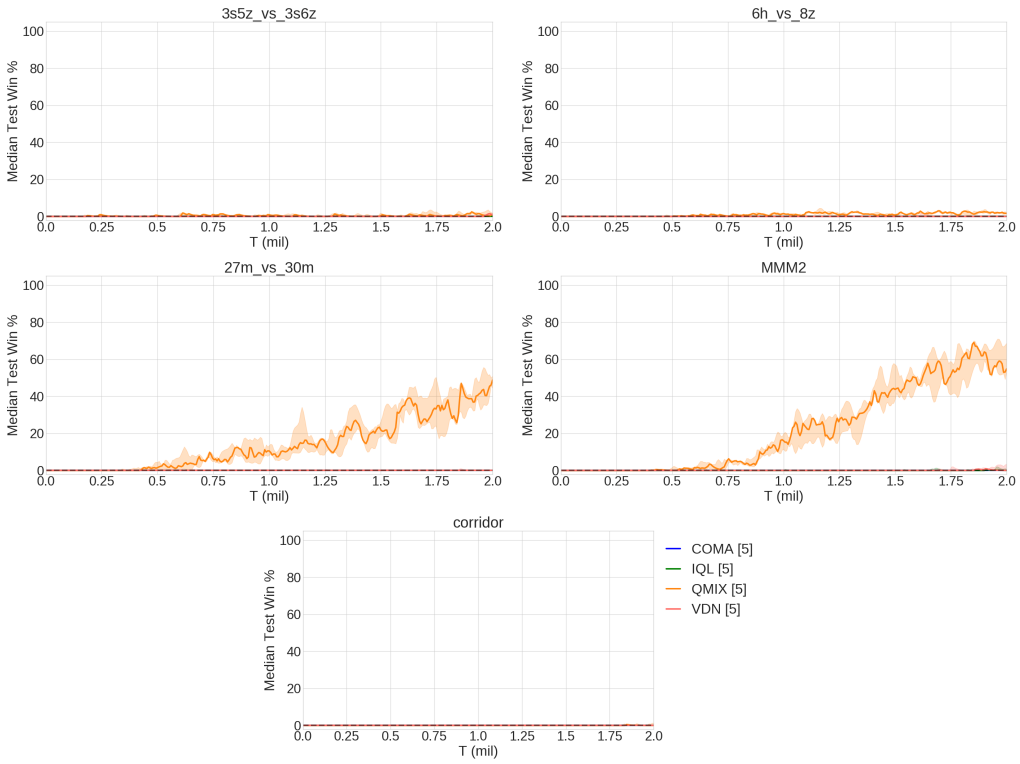


Figure 3.12: Super Hard scenarios. The heuristic AI's performance shown as a dotted black line.

3.9 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 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.

3.9.1 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.

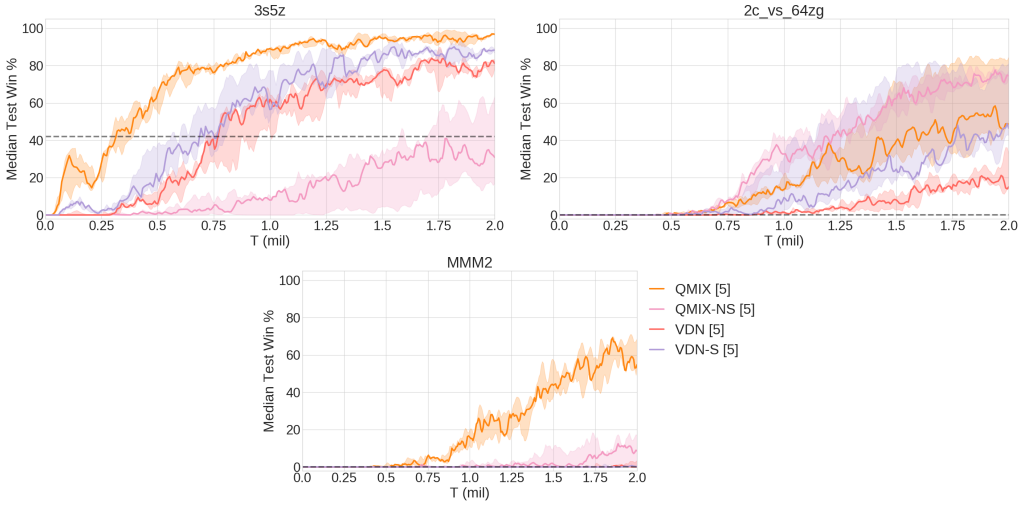


Figure 3.13: Ablations for state experiments.

Figure 3.13 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.

3.9.2 Role of Nonlinear Mixing

The results in Figure 3.13 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 3.14 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 of a nonlinearity

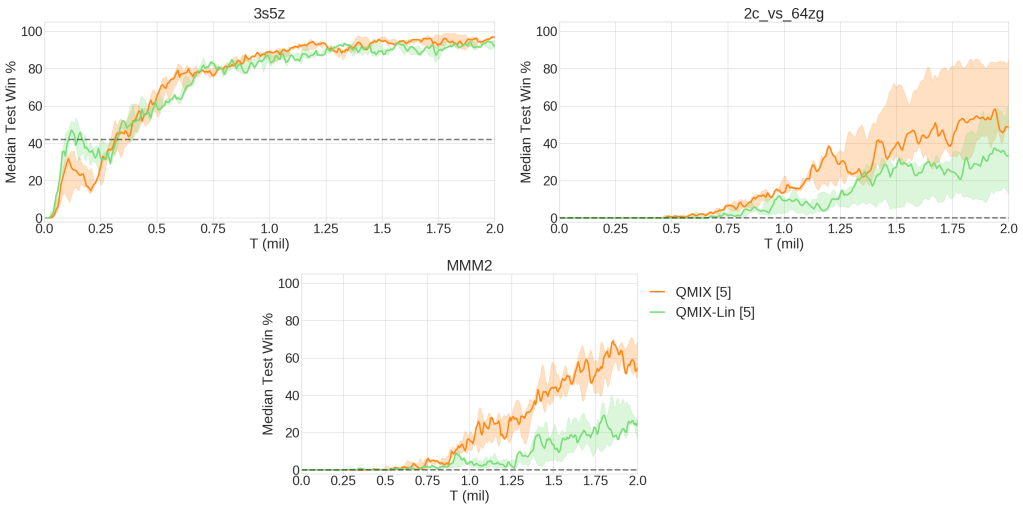


Figure 3.14: Ablations for mixing network linearity experiments.

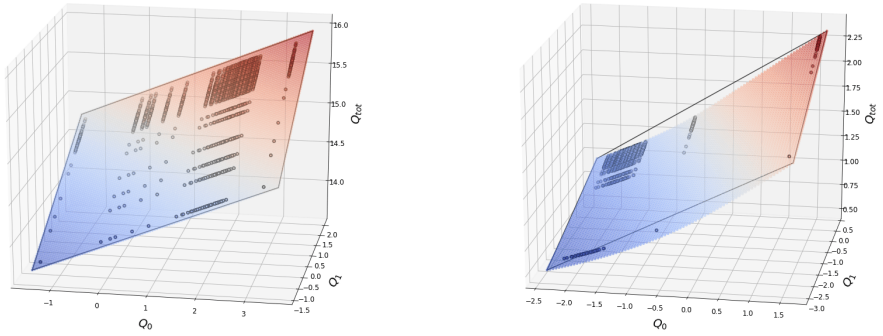


Figure 3.15: The learnt mixing of QMIX on *2c_vs_64zg* at the end of training for timesteps 0 (left) and 50 (right). Circles indicate the Q_{tot}-values for the discrete joint-action space.

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 3.15 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 3.15 (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 3.16 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

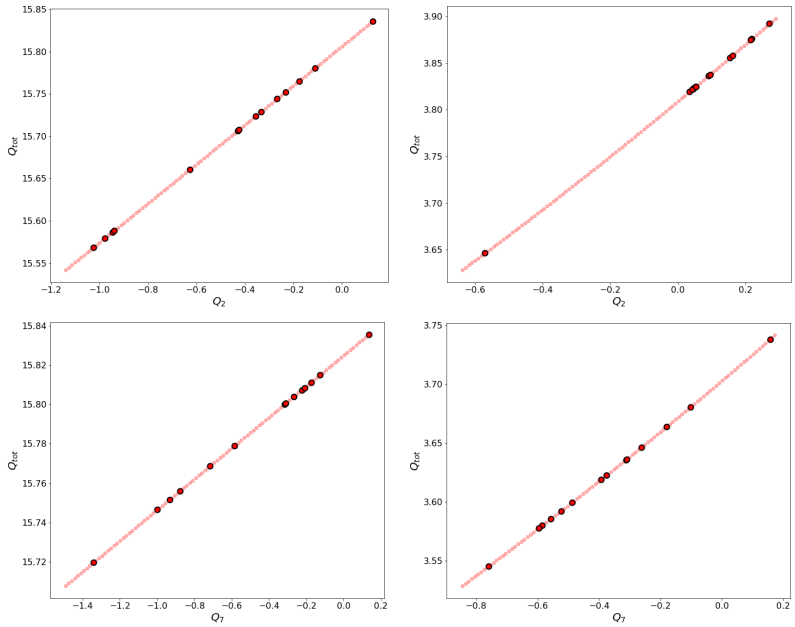


Figure 3.16: The learnt mixing of QMIX on *3s5z* at the end of training for timesteps 0 (left) and 50 (right), for agents 2 (top) and 7 (bottom).

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 3.17 visualises the learnt mixing function for the didactic two-step game from section 3.4, 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 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

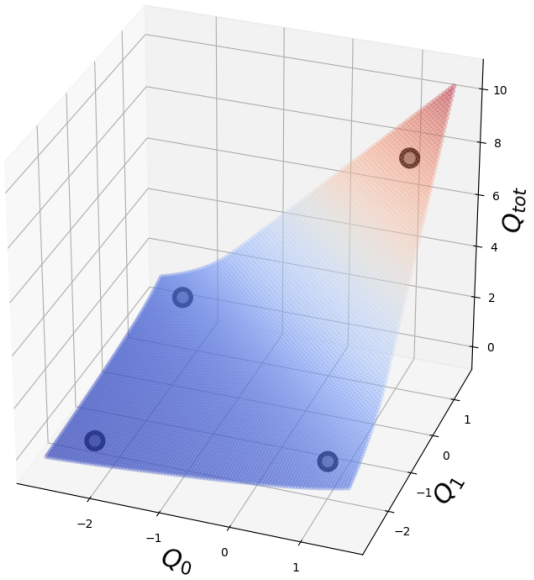


Figure 3.17: The learnt mixing of QMIX on the two-step game from Section 3.4.

task are presented below in Section 3.9.3.

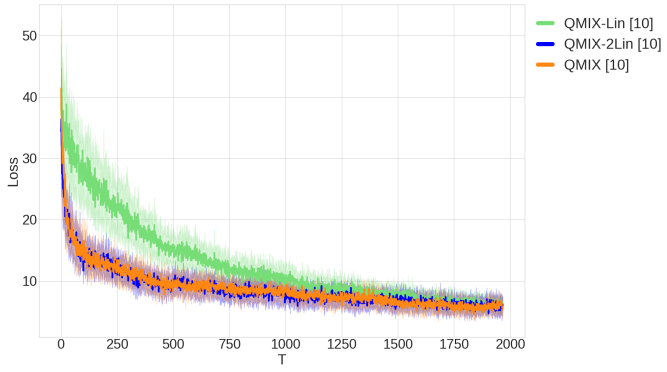


Figure 3.18: The loss on a regression task.

We compare the loss of QMIX (with an ELU non-linearity), QMIX-Lin, and QMIX-2Lin in Figure 3.18, which illustrates that QMIX and QMIX-2Lin perform similarly, and both perform a faster optimisation of the loss than QMIX-Lin. Even 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 (Arora et al., 2018).

Figure 3.19 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 3.15 (right)

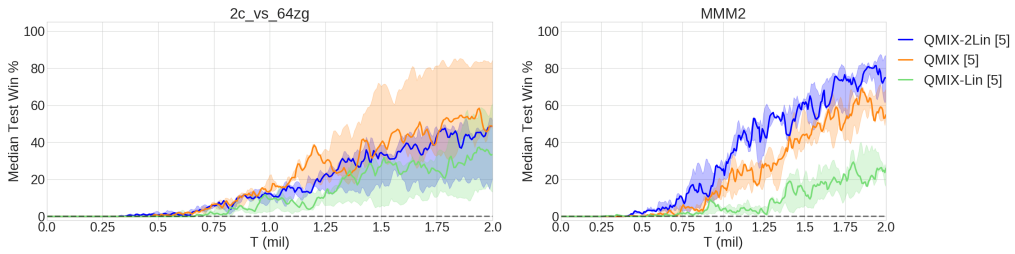


Figure 3.19: Ablations for linear experiments.

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.

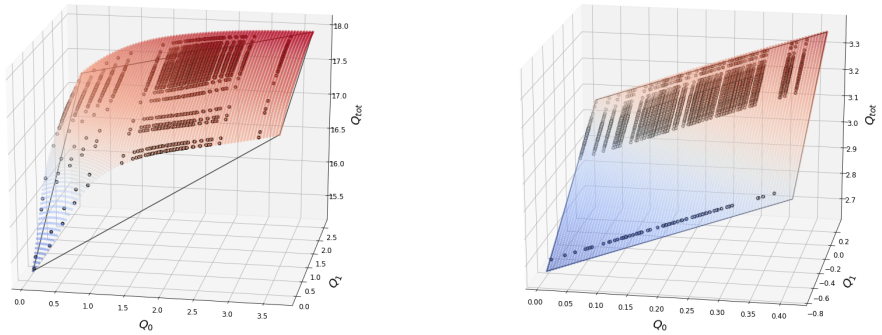


Figure 3.20: The learnt mixing of QMIX on *2c_vs_64zg* using a tanh nonlinearity at the end of training for timesteps 0 (left) and 50 (right).

Figures 3.20 and 3.21 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 3.22 shows how this translates to performance: there are perhaps some small gains in learning speed for some random seeds when using 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.

[A] The following section is taken from the paper's appendix.

3.9.3 Regression Experiment [A]

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

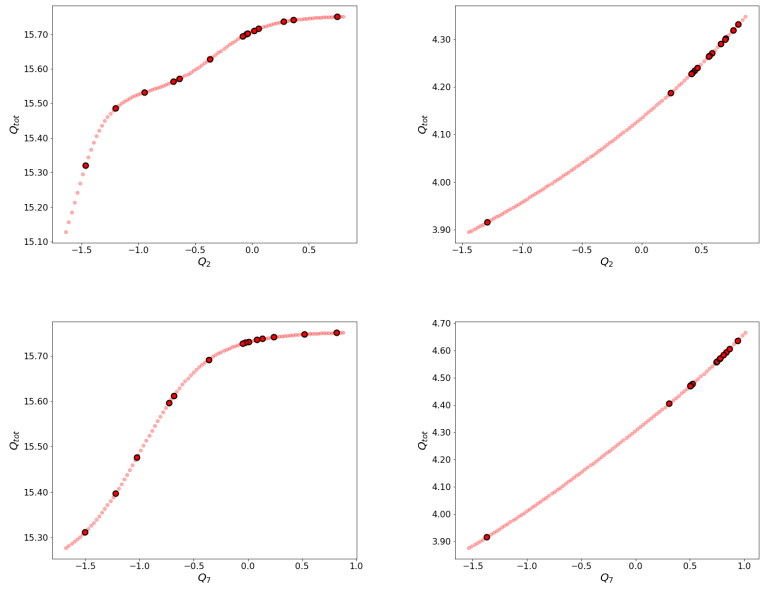


Figure 3.21: The learnt mixing of QMIX on *3s5z*, using a tanh nonlinearity, at the end of training for timesteps 0 (left) and 50 (right), for agents 2 (top) and 7 (bottom).

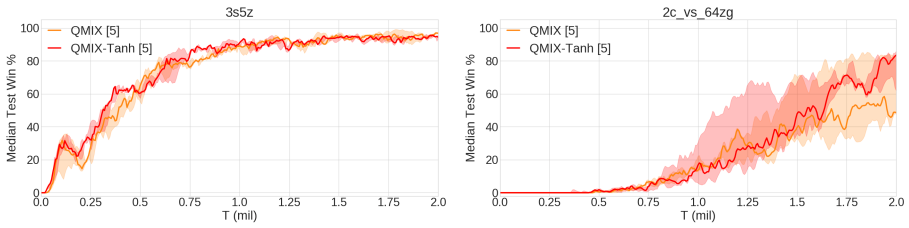


Figure 3.22: The Median test win % comparing QMIX with an ELU nonlinearity (QMIX) and QMIX with a tanh nonlinearity in the mixing network (QMIX-Tanh).

$[-1, 1]$. 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.

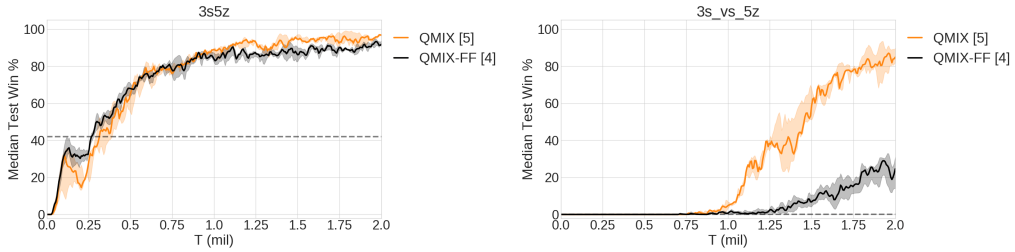


Figure 3.23: Comparing agent networks with and without RNNs (QMIX-FF) on two scenarios.

3.9.4 Role of RNN in Agent Network

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 3.23. 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.

3.9.5 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 3.23 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.

Figures 3.24 and 3.25 show that the architectural change has little effect on the performance of COMA, with the addition of the state actually degrading performance in *2s3z*.

3.9.6 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

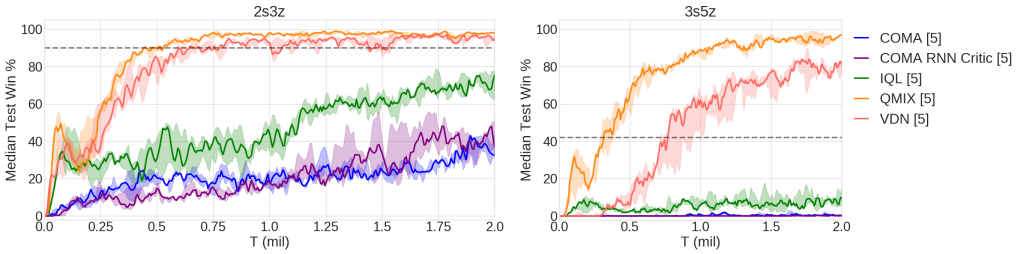


Figure 3.24: Comparing the different architectures for the COMA critic without the state, on two scenarios.

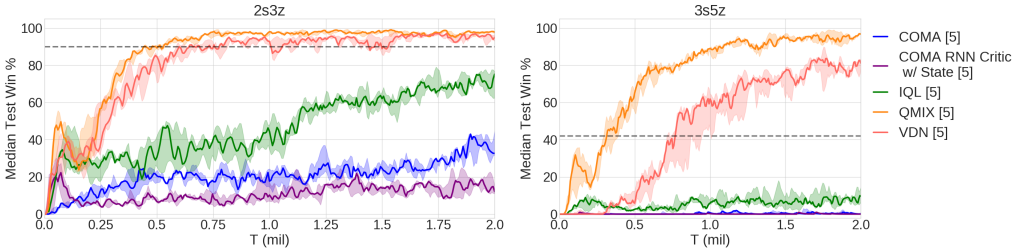


Figure 3.25: Comparing the different architectures for the COMA critic with the state on two scenarios.

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 3.26 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 3.27 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.

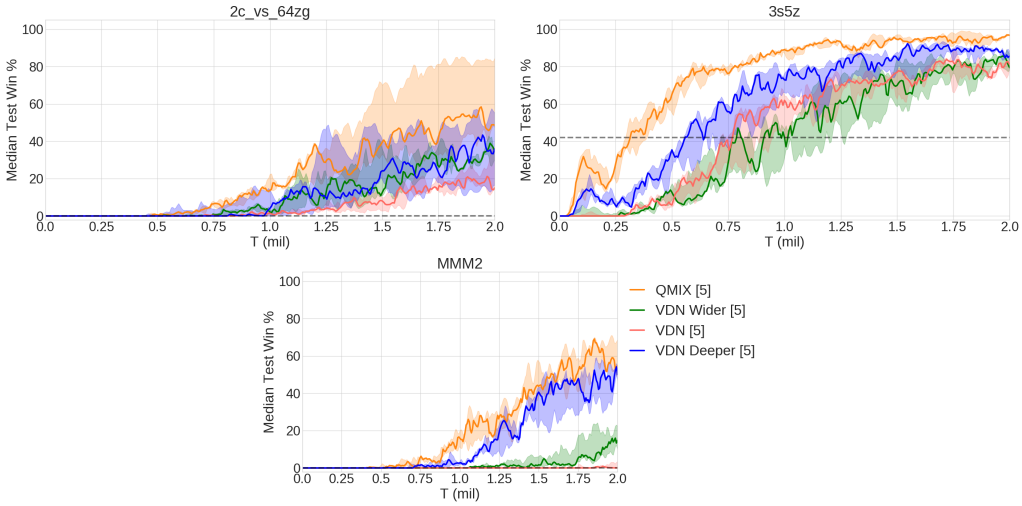


Figure 3.26: Comparing larger agent networks for VDN.

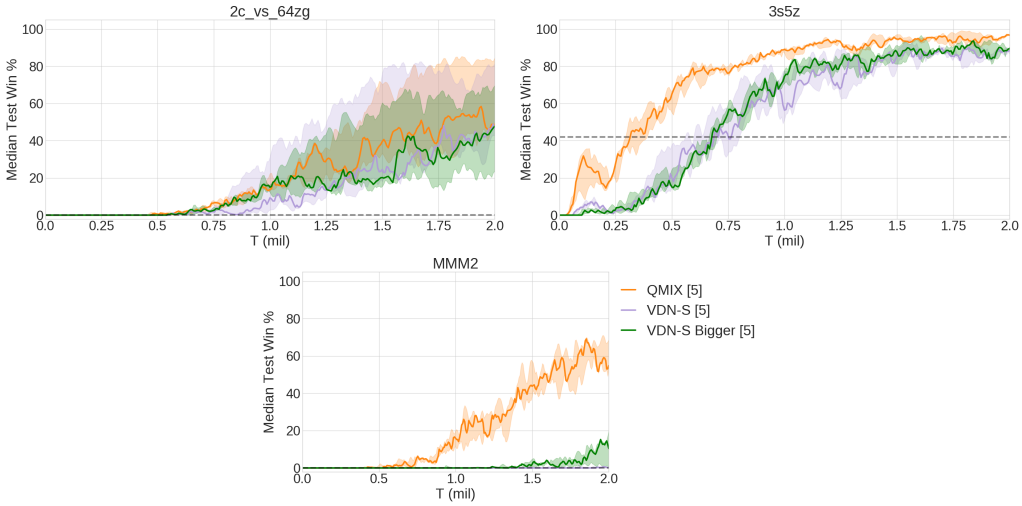


Figure 3.27: Comparing a larger state-dependent bias for VDN-S.

3.10 Conclusion

This article 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 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 ablations 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.

In the future, we aim to extend SMAC with new challenging scenarios that feature a more diverse set of units and require a higher level of coordination amongst agents. Particularly, we plan to make use of the rich skill set of StarCraft II units, and host scenarios that require the agents to utilise the features of the terrain. With harder multi-agent coordination problems, we aim to explore the gaps in existing multi-agent RL approaches and motivate further research in this domain.

There are several interesting avenues for future research:

- Investigating the performance gap between value-based methods and policy-based methods. In our experiments, QMIX (and VDN) significantly outperform COMA in all scenarios, both in terms of sample efficiency and final performance. The primary differences between the methods are the use of Q-learning vs the Policy Gradient theorem, use of off-policy data from the replay buffer and the architecture used to estimate the joint-action Q-Values.
- Expanding upon the joint-action Q-values representable by QMIX, to allow for the representation of coordinated behaviour. Importantly, this must scale gracefully to the deep RL setting.
- Better algorithms for multi-agent exploration that go beyond ϵ-greedy. There have been many advances with regards to exploration in single-agent RL, dealing with both sparse and dense rewards environments. Taking inspiration from these, or developing entirely new approaches that improve exploration in a cooperative multi-agent RL setting is an important problem. SMAC's *super-hard* scenarios provide a nice benchmark for evaluating these advances.

This concludes our quote.

Coming up next...

In this chapter, we have tackled various open challenges surrounding how to best exploit centralised training in CTDE settings. In particular, we 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 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.

QMIX does not naively scale to continuous action spaces as the evaluation of $\arg\max_u q(u, \tau_i)$ over continuous action spaces is not analytically tractable. In addition, SMAC also cannot naturally extended to continuous action spaces owing to the intrinsically discrete unit control actions in StarCraft2 micromanagement tasks. In Chapter 4, we introduce COMIX, a variant of QMIX that scales to continuous action spaces. However, we find that a novel actor-critic algorithm, FACMAC, To stimulate research into continuous control multi-agent algorithms, we also release Multi-Agent Mujoco (MAMujoco), a novel benchmark environment for continuous multi-agent robotic control tasks.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	*Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*
Publication Status	x Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	T. Rashid*, M. Samvelyan*, **C. Schroeder de Witt**, G. Farquhar, J. Foerster, and S. Whiteson (2020). *Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning*. Journal of Machine Learning Research, 21(178):1–51, 2020. ISSN 1533-7928. (*: equal contribution)

Student Confirmation

Student Name:	Christian Schroeder de Witt		
Contribution to the Paper	• Generation of Ideas with co-authors • Software Development with co-authors • Preliminary Experiments • Wrote and edited the paper with co-authors		
Signature	C. Schroeder de Witt	Date	30.04.2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Professor Philip H.S. Torr			
Supervisor comments			
Signature	Philip Torr	Date	04.05.2021

This completed form should be included in the thesis, at the end of the relevant chapter.

4

FACMAC: Factored Multi-Agent Centralised Policy Gradients

Contents

4.1 Introduction

In Chapter 3, we have introduced QMIX, a novel DMARL algorithm that can efficiently exploit CTDE settings amenable to monotonic critic factorisations (see Section 2.2.2). In particular, QMIX exhibits strongs performance on SMAC, a novel benchmark suite for discrete action CTDE settings. However, many real-world problems of interest feature continuous action spaces, most notably cooperative robotics. However, work on decentralised continuous control has received comparatively little attention and has been largely limited to deep deterministic policy gradient approaches (Lowe et al., 2017; Iqbal and Sha, 2019; Li et al., 2019). We believe the lack of diverse continuous benchmarks is one factor limiting progress in continuous MARL.

In addition, state-of-the-art cooperative multi-agent actor-critic approaches for discrete control, such as QMIX (Rashid et al., 2020b), have long tended to be based on Q-learning rather than actor-critic approaches, such as Central-V or COMA (Foerster et al., 2018). This begs the question of whether, and how, actor-critic approaches could be extended to discrete multi-agent control.

We now quote from (Peng et al., 2021), with the quoted text being visually indicated by reduced linespacing. Please note that the quote is literal except for where original supplementary material and appendices have been inserted in-place and internal document references have been updated accordingly.

Significant progress has been made in cooperative multi-agent reinforcement learning (MARL) under the paradigm of *centralised training with decentralised execution* (CTDE) (Oliehoek et al., 2008a; Kraemer and Banerjee, 2016) in recent years, both in value-based (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2020b; Pan et al., 2021) and actor-critic (Lowe et al., 2017; Foerster et al., 2018; Schroeder de Witt et al., 2020; Iqbal and Sha, 2019; Du et al., 2019) approaches. Most popular multi-agent actor-critic methods such as COMA (Foerster et al., 2018) and MADDPG (Lowe et al., 2017) learn a *centralised critic* with decentralised actors. The critic is centralised to make use of all available information (i.e., it can condition on the global state and the joint action) to estimate the joint action-value function Q_{tot}, unlike a *decentralised*

critic that estimates the local action-value function Q_a based only on individual observations and actions for each agent a.[1] Even though the joint action-value function these actor-critic methods can represent is not restricted, in practice they significantly underperform value-based methods like QMIX (Rashid et al., 2018) on the challenging StarCraft Multi-Agent Challenge (SMAC) (Samvelyan et al., 2019) benchmark (Rashid et al., 2020b,a).

In this paper, we propose a novel approach called ***FAC****tored* ***M****ulti-****A****gent* ***C****entralised policy gradients* (FACMAC), which works for both discrete and continuous cooperative multi-agent tasks. Like MADDPG, our approach uses deep deterministic policy gradients (Lillicrap et al., 2016) to learn decentralised policies. However, FACMAC learns a single *centralised but factored critic*, which factors the joint action-value function Q_{tot} into per-agent utilities Q_a that are combined via a non-linear monotonic function, as in the popular Q-learning algorithm QMIX (Rashid et al., 2018). While the critic used in COMA and MADDPG is also centralised, it is monolithic rather than factored.[2] Compared to learning a monolithic critic, our factored critic can potentially scale better to tasks with a larger number of agents and/or actions. In addition, in contrast to other value-based approaches such as QMIX, there are no inherent constraints on factoring the critic. This allows us to employ rich value factorisations, including *nonmonotonic* ones, that value-based methods cannot directly use without forfeiting decentralisability or introducing other significant algorithmic changes. We thus also employ a nonmonotonic factorisation and empirically demonstrate that its increased representational capacity allows it to solve some tasks that cannot be solved with monolithic, or monotonically factored critics.

In MADDPG, a separate policy gradient is derived for each agent individually, which optimises its policy assuming all other agents' actions are fixed. This could cause the agents to converge to sub-optimal policies in which no single agent wishes to change its action unilaterally. In FACMAC, we use a new *centralised* gradient estimator that optimises over the entire joint action space, rather than optimising over each agent's action space separately as in MADDPG. The agents' policies are thus trained as a single joint-action policy, which can enable learning of more coordinated behaviour, as well as the ability to escape sub-optimal solutions. The centralised gradient estimator fully reaps the benefits of learning a centralised critic, by not implicitly marginalising over the actions of the other agents in the policy-gradient update. The gradient estimator used in MADDPG is also known to be vulnerable to relative overgeneralisation (Wei and Luke, 2016). To overcome this issue, in our centralised gradient estimator, we sample all actions from all agents' current policies when evaluating the joint action-value function.

[1] COMA learns a single centralised critic for all cooperative agents due to parameter sharing. For each agent the critic has different inputs and can thus output different values for the same state and joint action. In MADDPG, each agents learns its own centralised critic, as it is designed for general multi-agent learning problems, including cooperative, competitive, and mixed settings.

[2] We use "centralised and monolithic critic" and "monolithic critic" interchangeably to refer to the centralised critic used in COMA and MADDPG, and "centralised but factored critic" and "factored critic" interchangeably to refer to the critic used in our approach.

In the following, we try to sharpen the meaning of the term *relative overgeneralisation*. The earliest use of the term dates back to Wiegand (2004), who define the term as:

> *The coevolutionary behavior that occurs when populations in the system are attracted towards areas of the space in which there are many strategies that perform well relative to the interacting partner.*

A perhaps more workable definition was subsequently supplied in (Wei and Luke, 2016, Section 3):

> *Relative overgeneralization occurs when a suboptimal Nash Equilibrium in the joint space of actions is preferred over an optimal Nash Equilibrium because each agents' action in the suboptimal equilibrium is a better choice when matched with arbitrary actions from the collaborating agents.*

Wei and Luke (2016) illustrate relative overgeneralisation very well using a simple 2-player matrix game in which the agents would not end up learning the jointly best policy if their learning signal was based on the average payoff, as is the case in independent learning settings.

To overcome *relative overgeneralisation*, please note that both centralised gradients, as well as agent actions derived from current policy evaluation, are required. Each of these two measures individually does not suffice.

We empirically show that MADDPG can quickly get stuck in local optima in a simple continuous matrix game, whereas our centralised gradient estimator finds the optimal policy. While Lyu et al. (2021) recently show that merely using a centralised critic (with per-agent gradients that optimise over each agent's actions separately) does not necessarily lead to better coordination between agents, our centralised gradient estimator re-establishes the value of using centralised critics.

Most recent works on continuous MARL focus on evaluating their algorithms on the multi-agent particle environments (Lowe et al., 2017), which feature a simple two-dimensional world with some basic simulated physics. To demonstrate FACMAC's scalability to more complex continuous domains and to stimulate more progress in continuous MARL, we introduce *Multi-Agent MuJoCo* (MAMuJoCo), a new,

comprehensive benchmark suite that allows the study of decentralised continuous control. Based on the popular single-agent MuJoCo benchmark (Brockman et al., 2016), MAMuJoCo features a wide variety of novel robotic control tasks in which multiple agents within a single robot have to solve a task cooperatively.

We evaluate FACMAC on variants of the multi-agent particle environments (Lowe et al., 2017) and our novel MAMuJoCo benchmark, which both feature continuous action spaces, and the challenging SMAC benchmark (Samvelyan et al., 2019), which features discrete action spaces. Empirical results demonstrate FACMAC's superior performance over MADDPG and other baselines on all three domains. In particular, FACMAC scales better when the number of agents (and/or actions) and the complexity of the task increases. Results on SMAC show that FACMAC significantly outperforms stochastic DOP (Wang et al., 2021), which recently claimed to be the first multi-agent actor-critic method to outperform state-of-the-art valued-based methods on SMAC, in all scenarios we tested. Moreover, our ablations and additional experiments demonstrate the advantages of both factoring the critic and using our centralised gradient estimator. We show that, compared to learning a monolithic critic, learning a factored critic can: 1) better take advantage of the centralised gradient estimator to optimise the agent policies when the number of agents and/or actions is large, and 2) leverage a nonmonotonic factorisation to solve tasks that cannot be solved with monolithic or monotonically factored critics.

4.2 Background

We consider a *fully cooperative multi-agent task* in which a team of agents interacts with the same environment to achieve some common goal. It can be modeled as a *decentralised partially observable Markov decision process* (Dec-POMDP) (Oliehoek and Amato, 2016) consisting of a tuple $G = \langle \mathcal{N}, S, U, P, r, \Omega, O, \gamma \rangle$. Here $\mathcal{N} \equiv \{1, \dots, n\}$ denotes the finite set of agents and $s \in S$ describes the true state of the environment. At each time step, each agent $a \in \mathcal{N}$ selects a discrete or continuous action $u_a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$. This results in a transition to the next state s' according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to [0, 1]$ and a team reward $r(s, \mathbf{u})$. $\gamma \in [0, 1)$ is a discount factor. Due to the *partial observability*, each agent $a \in \mathcal{N}$ draws an individual partial observation $o_a \in \Omega$ from the observation kernel $O(s, a)$. Each agent learns a stochastic policy $\pi_a(u_a|\tau_a)$ or a deterministic policy $\mu_a(\tau_a)$, conditioned only on its local action-observation history $\tau_a \in T \equiv (\Omega \times U)^*$. The joint stochastic policy $\boldsymbol{\pi}$ induces a joint *action-value function*: $Q^{\boldsymbol{\pi}}(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}} [R_t|s_t, \mathbf{u}_t]$, where $R_t = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$ is the discounted return. Similarly, the joint deterministic policy $\boldsymbol{\mu}$ induces a joint action-value function denoted $Q^{\boldsymbol{\mu}}(s_t, \mathbf{u}_t)$. We adopt the *centralised training with decentralised execution* (CTDE) paradigm (Oliehoek et al., 2008a; Kraemer and Banerjee, 2016), where policy training can exploit extra global information that might be available and has the freedom to share information between agents during training. However, during execution, each agent must act with only access to its own action-observation history.

VDN and QMIX. VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) are Q-learning algorithms for cooperative MARL tasks with discrete actions. They both aim to efficiently learn a centralised but factored action-value function $Q^{\boldsymbol{\pi}}_{tot}$, using CTDE. To ensure consistency between the centralised and decentralised policies, VDN and QMIX factor $Q^{\boldsymbol{\pi}}_{tot}$ assuming additivity and monotonicity, respectively. More specifically, VDN factors $Q^{\boldsymbol{\pi}}_{tot}$ into a sum of the per-agent utilities: $Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau}, \mathbf{u}; \boldsymbol{\phi}) = \sum_{a=1}^{n} Q_a^{\pi_a}(\tau_a, u_a; \phi_a)$. QMIX, however, represents $Q^{\boldsymbol{\pi}}_{tot}$ as a continuous monotonic mixing function of each agent's utilities: $Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \boldsymbol{\phi}, \psi) = f_\psi\Big(s, Q_1^{\pi_1}(\tau_1, u_1; \phi_1), \ldots, Q_n^{\pi_n}(\tau_n, u_n; \phi_n)\Big)$, where $\frac{\partial f_\psi}{\partial Q_a} \geq 0, \forall a \in \mathcal{N}$. This is sufficient to ensure that the global arg max performed on $Q^{\boldsymbol{\pi}}_{tot}$ yields the same result as a set of individual arg max performed on each $Q_a^{\pi_a}$. Here f_ψ is approximated by a monotonic mixing network, parameterised by ψ. Monotonicity can be guaranteed by non-negative mixing weights. These weights are generated by separate *hypernetworks* (Ha et al., 2017), which condition on the full state s. QMIX is trained end-to-end to minimise the following loss:

$$\mathcal{L}(\boldsymbol{\phi}, \psi) = \mathbb{E}_{\mathcal{D}}\Big[\Big(y^{tot} - Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \boldsymbol{\phi}, \psi)\Big)^2\Big], \tag{4.1}$$

where the bootstrapping target $y^{tot} = r + \gamma \max_{\mathbf{u}'} Q^{\boldsymbol{\pi}}_{tot}(\boldsymbol{\tau}', \mathbf{u}', s'; \boldsymbol{\phi}^-, \psi^-)$. Here r is the global reward, and $\boldsymbol{\phi}^-$ and ψ^- are parameters of the target Q and mixing network, respectively, as in DQN (Mnih et al., 2015). The expectation is estimated with a minibatch of transitions sampled from an experience replay buffer $\mathcal{D}$ (Lin, 1992b). During execution, each agent selects actions greedily with respect to its own $Q_a^{\pi_a}$.

MADDPG. MADDPG (Lowe et al., 2017) is an extension of DDPG (Lillicrap et al., 2016) to multi-agent settings. It is an actor-critic, off-policy method that uses the paradigm of CTDE to learn deterministic policies in continuous action spaces. In MADDPG, a separate actor and critic is learned for each agent, such that each agent can have its own arbitrary reward function. It is therefore applicable to either cooperative, competitive, or mixed settings. We assume each agent a has a deterministic policy $\mu_a(\tau_a; \theta_a)$, parameterised by θ_a (abbreviated as μ_a), and let $\boldsymbol{\mu} = \{\mu_a(\tau_a; \theta_a)\}_{a=1}^n$ be the set of all agent policies. In MADDPG, a *centralised and monolithic critic* that estimates the joint action-value function $Q_a^{\boldsymbol{\mu}}(s, u_1, \ldots, u_n; \phi_a)$ is learned for each agent a separately. The critic is said to be *centralised* as it utilises information only available to it during the *centralised* training phase, the global state s[3] and the actions of all agents, $u_1, \ldots, u_n$, to estimate the joint action-value function $Q_a^{\boldsymbol{\mu}}$, which is parameterised by ϕ_a. This joint action-value function is trained by minimising the following loss:

$$\mathcal{L}(\phi_a) = \mathbb{E}_{\mathcal{D}}\Big[\Big(y^a - Q_a^{\boldsymbol{\mu}}(s, u_1, \ldots, u_n; \phi_a)\Big)^2\Big], \tag{4.2}$$

[3]If the global state s is not available, the centralised and monolithic critic can condition on the joint observations or action-observation histories.

where $y^a = r_a + \gamma Q_a^{\boldsymbol{\mu}}(s', u'_1, \dots, u'_n|_{u'_a=\mu_a(\tau'_a;\theta_a^-)}; \phi_a^-)$. Here r_a is the reward received by each agent a, $u'_1, \dots, u'_n$ is the set of target policies with delayed parameters θ_a^-, and ϕ_a^- are the parameters of the target critic. The replay buffer $\mathcal{D}$ contains the transition tuples $(s, s', u_1, \dots, u_n, r_1, \dots, r_n)$.

The following policy gradient can be calculated individually to update the policy of each agent a:

$$\nabla_{\theta_a} J(\mu_a) = \mathbb{E}_{\mathcal{D}}\Big[\nabla_{\theta_a}\mu_a(\tau_a)\nabla_{u_a} Q_a^{\boldsymbol{\mu}}(s, u_1, \dots, u_n)\Big|_{u_a=\mu_a(\tau_a)}\Big], \tag{4.3}$$

where the current agent a's action u_a is sampled from its current policy μ_a when evaluating the joint action-value function $Q_a^{\boldsymbol{\mu}}$, while all other agents' actions are sampled from the replay buffer $\mathcal{D}$.

4.3 FACMAC

In this section, we propose a new approach called ***FAC****tored* ***M****ulti-****A****gent* ***C****entralised policy gradients* (FACMAC) that uses a centralised but factored critic and a centralised gradient estimator to learn continuous cooperative tasks. We start by describing the idea of learning a centralised but factored critic. We then discuss our new centralised gradient estimator and demonstrate its benefit in a simple continuous matrix game. Finally, we discuss how we adapt our method to discrete cooperative tasks.

4.3.1 Learning a Centralised but Factored Critic

Learning a centralised and monolithic critic conditioning on the global state and the joint action can be difficult and/or impractical when the number of agents and/or actions is large (Iqbal and Sha, 2019). We thus employ value function factorisation in the multi-agent actor-critic framework to enable scalable learning of a centralised critic in Dec-POMDPs. Another key advantage of adopting value factorisation in an actor-critic framework is that, compared to value-based methods, it allows for a more flexible factorisation as the critic's design is not constrained. One can employ any type of factorisation, including nonmonotonic factorisations that value-based methods cannot directly use without forfeiting decentralisability or introducing other significant algorithmic changes.

Specifically, in FACMAC, all agents share a centralised critic $Q_{tot}^{\boldsymbol{\mu}}$ that is factored as:

$$Q_{tot}^{\boldsymbol{\mu}}(\boldsymbol{\tau}, \mathbf{u}, s; \boldsymbol{\phi}, \psi) = g_\psi\Big(s, \{Q_a^{\mu_a}(\tau_a, u_a; \phi_a)\}_{a=1}^n\Big), \tag{4.4}$$

where $\boldsymbol{\phi}$ and ϕ_a are parameters of the joint action-value function $Q_{tot}^{\boldsymbol{\mu}}$ and agent-wise utilities $Q_a^{\mu_a}$, respectively. In our canonical implementation which we refer to as FACMAC, g_ψ is a non-linear monotonic function parametrised as a mixing network

with parameters ψ, as in QMIX (Rashid et al., 2018). To evaluate the policy, the centralised but factored critic is trained by minimising the following loss:

$$\mathcal{L}(\boldsymbol{\phi}, \psi) = \mathbb{E}_{\mathcal{D}}\Big[\Big(y^{tot} - Q^{\boldsymbol{\mu}}_{tot}(\boldsymbol{\tau}, \mathbf{u}, s; \boldsymbol{\phi}, \psi)\Big)^2\Big], \tag{4.5}$$

where $y^{tot} = r + \gamma Q^{\boldsymbol{\mu}}_{tot}(\boldsymbol{\tau'}, \boldsymbol{\mu}(\boldsymbol{\tau'}; \boldsymbol{\theta}^-), s'; \boldsymbol{\phi}^-, \psi^-)$. Here $\mathcal{D}$ is the replay buffer, and $\boldsymbol{\theta}^-$, $\boldsymbol{\phi}^-$, and ψ^- are parameters of the target actors, critic, and mixing network, respectively.

Leveraging the flexibility of our approach, namely the lack of restrictions on the form of the critic, we also explore a new nonmonotonic factorisation with full representational capacity. The joint action-value function $Q^{\boldsymbol{\mu}}_{tot}$ is represented as a non-linear non-monotonic mixing function of per-agent utilities $Q^{\mu_a}_a$. This nonmonotonic mixing function is parameterised as a mixing network, with a similar architecture to g_ψ in FACMAC, but without the constraint of monotonicity enforced by using non-negative weights. We refer to this method as FACMAC-nonmonotonic. Additionally, to better understand the advantages of factoring a centralised critic, we also explore two additional simpler factorisation schemes. These include factoring the centralised critic $Q^{\boldsymbol{\mu}}_{tot}$ into a sum of per-agent utilities $Q^{\mu_a}_a$ as in VDN (FACMAC-vdn), and as a sum of $Q^{\mu_a}_a$ and a state-dependent bias (FACMAC-vdn-s). Our value factorisation technique is general and can be readily applied to any multi-agent actor-critic algorithms that learn centralised and monolithic critics (Lowe et al., 2017; Foerster et al., 2018; Du et al., 2019).

4.3.2 Centralised Policy Gradients

To update the decentralised policy of each agent, a naive adaptation of the deterministic policy gradient used by MADDPG (shown in (4.3)) is

$$\nabla_{\theta_a} J(\mu_a) = \mathbb{E}_{\mathcal{D}}\Big[\nabla_{\theta_a}\mu_a(\tau_a)\nabla_{u_a} Q^{\boldsymbol{\mu}}_{tot}(\boldsymbol{\tau}, u_1, \ldots, u_n, s)\Big|_{u_a=\mu_a(\tau_a)}\Big]. \tag{4.6}$$

Compared to the policy gradient used in MADDPG, the updates of all agents' individual deterministic policies now depend on the single shared factored critic $Q^{\boldsymbol{\mu}}_{tot}$, as opposed to learning and utilising a monolithic critic $Q^{\boldsymbol{\mu}}_a$ for each agent. However, there are two main problems in both policy gradients. First, each agent optimises its own policy assuming all other agents' actions are fixed, which could cause the agents to converge to sub-optimal policies in which no single agent wishes to change its action unilaterally. Second, both policy gradients make the corresponding methods vulnerable to relative overgeneralisation (Wei and Luke, 2016) as, when agent a ascends the policy gradient based on $Q^{\boldsymbol{\mu}}_a$ or $Q^{\boldsymbol{\mu}}_{tot}$, only its own action u_a is sampled from its current policy μ_a, while all other agents' actions are sampled from the replay buffer $\mathcal{D}$. The other agents' actions thus might be drastically different from the actions their current policies would choose. This could cause the agents to converge to sub-optimal actions that appear to be a better choice when considering the effect of potentially arbitrary actions from the other collaborating agents.

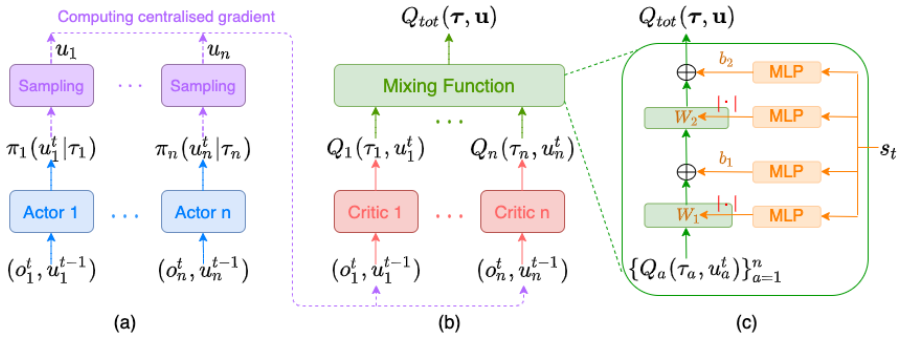


Figure 4.1: The overall FACMAC architecture. (a) The decentralised policy networks. There is a sampling step since we sample from the categorical distribution when using discrete actions. (b) The centralised but factored critic. (c) The non-linear monotonic mixing function.

In FACMAC, we use a new *centralised* gradient estimator that optimises over the entire joint action space, rather than optimising over each agent's actions separately as in both (4.3) and (4.6), to achieve better coordination among agents. In addition, to overcome relative overgeneralisation, when calculating the policy gradient we sample all actions from all agents' current policies when evaluating $Q_{tot}^{\boldsymbol{\mu}}$. Our centralised policy gradient can thus be estimated as

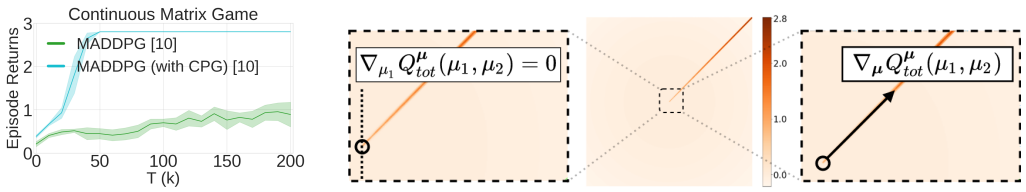
$$\nabla_\theta J(\boldsymbol{\mu}) = \mathbb{E}_{\mathcal{D}}\Big[\nabla_\theta \boldsymbol{\mu} \nabla_{\boldsymbol{\mu}} Q_{tot}^{\boldsymbol{\mu}}(\boldsymbol{\tau}, \mu_1(\tau_1), \ldots, \mu_n(\tau_n), s)\Big], \tag{4.7}$$

where $\boldsymbol{\mu} = \{\mu_1(\tau_1; \theta_1), \ldots, \mu_n(\tau_n; \theta_n)\}$ is the set of all agents' current policies and all agents share the same actor network parameterised by θ. However, it is not a requirement of our method for all agents to share parameters in this manner.

Note that not sharing parameters among agents reduces sample-efficiency of the learning algorithm as the ratio of gradients to parameters reduces.

If the critic factorisation is linear, as in FACMAC-vdn, then the centralised gradient is equivalent to the per-agent gradients that optimise over each agent's actions separately. This is explored in more detail by DOP (Wang et al., 2021), which restricts the factored critic to be linear to exploit this equivalence. A major benefit of our method then, is that it does not place any such restrictions on the critic. As remarked by Lyu et al. (2021), merely using a centralised critic with per-agent gradients does not necessarily lead to better coordination between agents due to the two problems outlined above. Even with a factored critic, methods that use stochastic policy gradients can still suffer from the problems caused by the per-agent gradients if a fully factored stochastic policy is used. Our centralised gradient estimator, which uses deterministic policies and optimises over the entire joint action space, is required in order to fully take advantage of a centralised critic.

Figure 4.1 illustrates the overall FACMAC architecture. For each agent a, there is one policy network that decides which individual action (discrete or continuous) to



(a) (b)

Figure 4.2: (a) Mean test return on Continuous Matrix Game. The shaded area in denotes the $25-75\%$ of the error of the mean. (b) **Left:** Per-agent policy gradient at the origin. For agent 1 (similarly for agent 2) it is 0 since the gradient term assumes the other agent's action to be fixed and thus it only considers the relative improvements along the dotted line (agent 1's own action space). **Right:** Our centralised policy gradient correctly determines the gradient for improving the joint action.

take. There is also one critic network for each agent a that estimates the individual agent utilities Q_a, which are then combined into the joint action-value function Q_{tot} via a non-linear monotonic mixing function as in QMIX. Q_{tot} is then used by our centralised gradient estimator to help the actor update its policy parameters.

To show the benefits of our new centralised gradient estimator, we compare MADDPG with the centralised policy gradient (CPG) against the original MADDPG on a simple continuous cooperative matrix game with two agents. Figure 4.5 in Appendix 4.6.1 illustrates this matrix game. There is a narrow path (shown in red) starting from the origin $(0, 0)$ to $(1, 1)$, in which the reward gradually increases. Everywhere else there is a small punishment moving away from the origin, increasing in magnitude further from the origin. Experimental results are shown in Figure 4.2(a). MADDPG quickly gets stuck in the local optimum within $200k$ timesteps, while MADDPG (with CPG) robustly converges to the optimal policy. Figure 4.2(b) visualises the differences between the per-agent and centralised policy gradients, demonstrating that the per-agent policy gradient can be wrong in our continuous matrix game and our centralised policy gradient is necessary to take advantage of the centralised critic. In Section 4.7, we further demonstrate the benefits of our centralised gradient estimator in more complex tasks.

4.3.3 Discrete Policy Learning

As FACMAC requires differentiable policies and the sampling process of discrete actions from a categorical distribution is not differentiable, we use the Gumbel-Softmax estimator (Jang et al., 2016) to enable efficient learning of FACMAC on cooperative tasks with discrete actions. The Gumbel-Softmax estimator is a continuous distribution that approximates discrete samples from a categorical distribution to produce differentiable samples. Moreover, we use the Straight-Through Gumbel-Softmax Estimator (Jang et al., 2016) to ensure the action dynamics during training and evaluation are the same. Specifically, during training, we sample discrete actions u_a from the original categorical distribution in the forward pass, but use the continuous Gumbel-Softmax sample x_a in the backward pass to

approximate the gradients: $\nabla_{\theta_a} u_a \approx \nabla_{\theta_a} x_a$. We can then update the agent's policy using our centralised policy gradient: $\nabla_\theta J(\boldsymbol{x}) \approx \mathbb{E}_{\mathcal{D}}\Big[\nabla_\theta \boldsymbol{x} \nabla_{\boldsymbol{x}} Q^{\boldsymbol{x}}_{tot}(\boldsymbol{\tau}, x_1, \ldots, x_n, s)\Big]$, where $\boldsymbol{x} = \{x_1, \ldots, x_n\}$ is the set of continuous samples that approximates the discrete agent actions. The softmax temperature hyperparameter τ is set to be 1 in our experiments.

[A] The following section was taken from the paper's appendix.

4.4 COVDN and COMIX

Q-learning has shown considerable success in multi-agent settings with discrete action spaces (Sunehag et al., 2018; Rashid et al., 2018; Son et al., 2019; Rashid et al., 2020a; Wang et al., 2020b). However, performing greedy action selection in Q-learning requires evaluating $\arg\max_{\mathbf{u}} Q_{tot}(\boldsymbol{\tau}, \mathbf{u}, s)$, where Q_{tot} is the joint action-value function. In discrete action spaces, this operation can be performed efficiently through enumeration (unless the action space is extremely large). In continuous action spaces, however, enumeration is impossible. Hence, existing continuous Q-learning approaches in single-agent settings either impose constraints on the form of Q-value to make maximisation easy (Gu et al., 2016; Amos et al., 2016), at the expense of estimation bias, or perform only approximate greedy action selection (Kalashnikov et al., 2018). Neither approach scales easily to the large joint action spaces inherent to multi-agent settings, as 1) the joint action space grows exponentially in the number of agents, and 2) training Q_{tot} required for greedy action selection becomes impractical when there are many agents.

This highlights the importance of learning a centralised but factored Q_{tot}. To factor large joint action spaces efficiently in a decentralisable fashion, COVDN represents the joint action-value function Q_{tot} as a sum of the per-agent utilities Q_a as in VDN (Sunehag et al., 2018), while COMIX represents Q_{tot} as a non-linear monotonic combination of Q_a as in QMIX (Rashid et al., 2018). COVDN and COMIX are thus simple variants of VDN and QMIX, respectively, that scale to continuous action

spaces. They both perform approximate greedy selection of actions u_a with respect to utility functions Q_a for each agent a using the cross-entropy method (CEM) (de Boer et al., 2005). CEM is a sampling-based derivative-free heuristic search method that has been successfully used to find approximate maxima of nonconvex Q-networks in a number of single-agent robotic control tasks (Kalashnikov et al., 2018). The centralised but factored Q_{tot} allows us to use CEM to sample actions for each agent independently and to use the individual utility function Q_a to guide the selection of maximal actions.

In both COVDN and COMIX, CEM is used by each agent a to find an action that approximately optimises its local utility function Q_a. Specifically, CEM iteratively draws a batch of N random samples from a candidate distribution $\mathcal{D}_k$, e.g., a Gaussian, at each iteration k. The best $M < N$ samples (with the highest utility values) are then used to fit a new Gaussian distribution $\mathcal{D}_{k+1}$, and this process repeats K times. We use a CEM hyperparameter configuration similar to Qt-Opt (Kalashnikov et al., 2018), where $N = 64$, $M = 6$, and $K = 2$.[4] Gaussian distributions are initialised with mean $\mu = 0$ and standard deviation $\sigma = 1$. Algorithm 2 outlines the full CEM process used in both COVDN and COMIX. Algorithm 3 outlines the full process for COMIX. Note we do not consider COVDN and COMIX significant algorithmic contributions but instead merely baseline algorithms.

4.5 Multi-Agent MuJoCo

The evaluation of continuous MARL algorithms has recently been largely limited to the simple multi-agent particle environments (Lowe et al., 2017). We believe the lack of diverse continuous benchmarks is one factor limiting progress in continuous MARL. To demonstrate FACMAC's scalability to more complex continuous domains and to stimulate more progress in continuous MARL, we develop *Multi-Agent MuJoCo* (MAMuJoCo), a novel benchmark for continuous cooperative multi-agent robotic control. Starting from the popular fully observable single-agent robotic MuJoCo

[4] We empirically find 2 iterations to suffice.

Algorithm 2 For each agent a, we perform n_c CEM iterations. Hyper-parameters $d_i \in \mathbb{N}$ control how many actions are sampled at the ith iteration.

for $a := 1,\ a \leq N$ **do**
  $\boldsymbol{\mu}_a \leftarrow \mathbf{0} \in \mathbb{R}^{|\mathcal{A}_a|}$
  $\boldsymbol{\sigma}_a \leftarrow \mathbf{1} \in \mathbb{R}^{|\mathcal{A}_a|}$
  for $i := 1,\ i \leq n_c$ **do**
    for $j := 1,\ j \leq d_i$ **do**
      $\mathbf{v'}_{aj} \sim \mathcal{N}(\boldsymbol{\mu}_a, \boldsymbol{\sigma}_a)$
      $\mathbf{v}_{aj} \leftarrow \tanh(\mathbf{v'}_{aj})$
      $q_{aj} \leftarrow Q_a(\tau_a, \mathbf{v}_{aj})$
      $j \leftarrow j + 1$
    if $i < n_c$ **then**
      $U \leftarrow \left\{\mathbf{v'}_{al} \mid q_{al} \in \text{top}k_i(q_{a1}, \ldots, q_{ad_i}), \forall l \in \{1 \ldots N\}\right\}$
      $\boldsymbol{\mu}_a \leftarrow \text{sample\_mean}(U)$
      $\boldsymbol{\sigma}_a \leftarrow \text{sample\_std}(U)$
    else
      $m \leftarrow \arg\max_j q_{aj}$
      $\mathbf{u}_a \leftarrow \mathbf{v}_{am}$
    $i \leftarrow i + 1$
  $a \leftarrow a + 1$
return $\langle \mathbf{u}_1, \ldots, \mathbf{u}_n \rangle$

Algorithm 3 Algorithmic description of COMIX. The function CEM is defined in Algorithm 2.

Initialise ReplayBuffer, $\theta, \theta^-, \phi, \phi^-$
for each training episode e **do**
  $s_0, \mathbf{z}_0 \leftarrow \text{EnvInit}()$
  for $t := 0$ until $t = T$ step 1 **do**
    $\mathbf{u}_t \leftarrow \text{CEM}(Q_1, \ldots, Q_N, \tau_t^1, \ldots, \tau_t^N)$
    $\langle s_{t+1}, \mathbf{z}_{t+1}, r_t \rangle \leftarrow \text{EnvStep}(\mathbf{u}_t)$
    ReplayBuffer $\leftarrow \langle s_t, \mathbf{u}_t, \mathbf{z}_t, r_t, s_{t+1}, \boldsymbol{z}_{t+1} \rangle$
  $\{\langle s_i, \boldsymbol{u}_i, \boldsymbol{z}_i, r_i, s'_i, \boldsymbol{z}'_i \rangle\}_{i=1}^b \sim$ ReplayBuffer
  $y_i \leftarrow r_i + \gamma \max_{\mathbf{u}'_i} Q_{\text{tot}}(s'_i, \boldsymbol{z}'_i, \boldsymbol{u}'_i; \boldsymbol{\theta}^-),\ \forall i$
  $\mathcal{L} \leftarrow \sum_{i=1}^{b} \Big(y_i - Q_{\text{tot}}(s_i, \boldsymbol{z}_i, \boldsymbol{u}_i; \boldsymbol{\theta}) \Big)^2$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\boldsymbol{\theta}} \mathcal{L}$

(Todorov et al., 2012) control suite included with OpenAI Gym (Brockman et al., 2016), we create a wide variety of novel scenarios in which multiple agents within a single robot have to solve a task cooperatively.

This design offers important benefits. It facilitates comparisons to existing literature on both the fully observable single-agent domain (OpenAI, 2020), as well as settings with low-bandwidth communication (Wang et al., 2018). More importantly, it

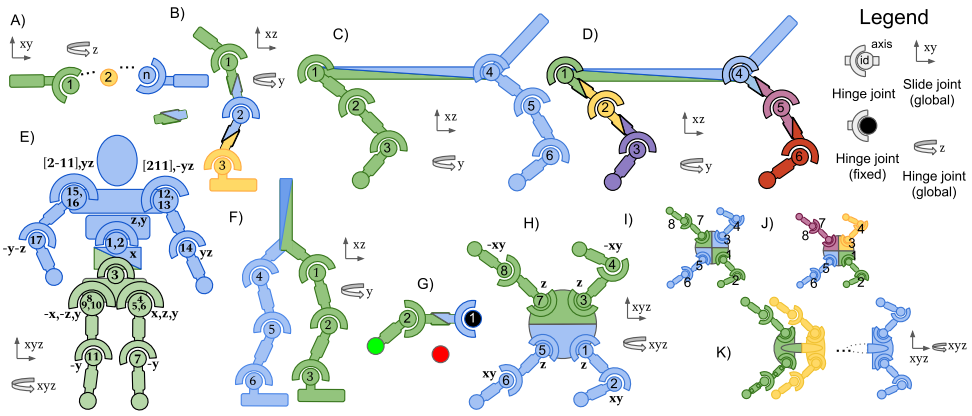


Figure 4.3: Agent partitionings for MAMuJoCo environments: A) Manyagent Swimmer, B) 3-Agent Hopper [3x1], C) 2-Agent HalfCheetah [2x3], D) 6-Agent HalfCheetah [6x1], E) 2-Agent Humanoid and 2-Agent HumanoidStandup (each [1x9,1x8]), F) 2-Agent Walker [2X3], G) 2-Agent Reacher [2x1], H) 2-Agent Ant [2x4], I) 2-Agent Ant Diag [2x4], J) 4-Agent Ant [4x2], and K) Manyagent Ant. Colours indicate agent partitionings. Each joint corresponds to a single controllable motor. Split partitions indicate shared body segments. Square brackets indicate [(number of agents) x (joints per agent)]. Joint IDs are in order of definition in the corresponding OpenAI Gym XML asset files (Brockman et al., 2016). Global joints indicate degrees of freedom of the center of mass of the composite robotic agent.

allows for the study of novel MARL algorithms for decentralised coordination in isolation (scenarios with multiple robots may add confounding factors such as spatial exploration), which is currently a gap in the research literature. MAMuJoCo also includes scenarios with a larger and more flexible number of agents, which takes inspiration from modular robotics (Yim et al., 2002; Kurokawa et al., 2008). Compared to traditional robots, modular robots are more versatile, configurable, and scalable. We thus develop two scenarios named ManyAgent Swimmer and ManyAgent Ant, in which one can configure an arbitrarily large number of agents (within the memory limits), each controlling a consecutive segment of arbitrary length.

Single-robot multi-agent tasks in MAMuJoCo arise by first representing a given single robotic agent as a *body graph*, where vertices (joints) are connected by adjacent edges (body segments), as shown in Figure 4.3. We then partition the body graph into disjoint sub-graphs, one for each agent, each of which contains one or more joints that can be controlled. Note that in ManyAgent Swimmer (see Figure 4.3A) and ManyAgent Ant (see Figure 4.3K), the number of agents are not limited by the given single robotic agent. See Appendix 4.5 for more details about MAMuJoCo.

[A] The following section was taken from the paper's appendix. While several MARL benchmarks with continuous action spaces have been released, few are simul-

taneously diverse, fully cooperative, decentralisable, and admit partial observability. The multi-agent particle suite (Lowe et al., 2017) features a few decentralisable tasks in a fully observable planar point mass toy environment. Presumably due to its focus on real-world robotic control, RoboCup soccer simulation (Kitano et al., 1997; Stone and Sutton, 2001; Riedmiller et al., 2009) does not currently feature an easily configurable software interface for MARL, nor suitable AI-controlled benchmark opponents. Liu et al. (2019a) introduce MuJoCo soccer environment, a multi-agent soccer environment with continuous simulated physics that cannot be used in a purely cooperative setting and does not admit partial observability.

To demonstrate FACMAC's scalability to more complex continuous domains and to stimulate more progress in continuous MARL, we develop *Multi-Agent MuJoCo* (MAMuJoCo), a novel benchmark for continuous cooperative multi-agent robotic control. Starting from the popular fully observable single-agent robotic MuJoCo (Todorov et al., 2012) control suite included with OpenAI Gym (Brockman et al., 2016), we create a wide variety of novel scenarios in which multiple agents within a single robot have to solve a task cooperatively. Single-robot multi-agent tasks in MAMuJoCo arise by first representing a given single robotic agent as a *body graph*, where vertices (joints) are connected by adjacent edges (body segments), as shown in Figure 4.3. We then partition the body graph into disjoint sub-graphs, one for each agent, each of which contains one or more joints that can be controlled. Note that in ManyAgent Swimmer (see Figure 4.3A) and ManyAgent Ant (see Figure 4.3K), the number of agents are not limited by the given single robotic agent.

Multiple agents are introduced within a single robot as partial observability arises through latency, bandwidth, and noisy sensors in a single robot. Even if communication is free and instant when it works, we want policies that keep working even when communication channels within the robot malfunction. Without access to the exact full state, local decision rules become more important and

introducing autonomous agents at individual decision points (e.g., each physical component of the robot) is reasonable and beneficial. This also makes it more robust to single-point failures (e.g., broken sensors) and more adaptive and flexible as new independent decision points (thus agents) may be added easily. Similar physical decompositions can be found in early behavior-based robotic research (Brooks, 1986; Arkin, 1989).

Each agent's action space in MAMuJoCo is given by the joint action space over all motors controllable by that agent. For example, the agent corresponding to the green partition in 2-Agent HalfCheetah (Figure 4.3C) consists of three joints (joint ids 1, 2, and 3) and four adjacent body segments. Each joint has an action space $[-1, 1]$, so the action space for each agent is a 3-dimensional vector with each entry in $[-1, 1]$.

For each agent a, observations are constructed in a two-stage process. First, we infer which body segments and joints are observable by agent a. Each agent can always observe all joints within its own sub-graph. A configurable parameter $k \geq 0$ determines the maximum graph distance to the agent's subgraph at which joints are observable (see Figure 4.4 for an example). Body segments directly attached to observable joints are themselves observable. The agent observation is then given by a fixed order concatenation of the representation vector of each observable graph element. Depending on the environment and configuration, representation vectors may include attributes such as position, velocity, and external body forces. In addition to joint and body-segment specific observation categories, agents can also be configured to observe the position and/or velocity attributes of the robot's central torso.

Restricting both the observation distance k, as well as limiting the set of observable element categories imposes partial observability. However, task goals remain unchanged from the single-agent variants, except that the goals must be reached

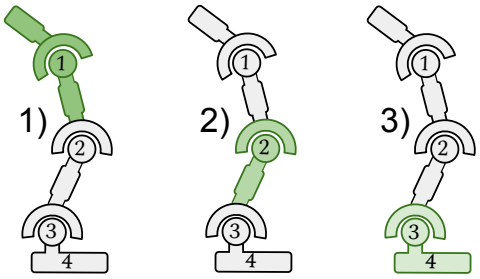


Figure 4.4: Observations by distance for 3-Agent Hopper (as seen from agent 1). Each corresponds to joints and body parts observable at 1) zero graph distance from agent 1, 2) one unit graph distance from agent 1, and 3) two unit graph distances from agent 1.

collaboratively by multiple agents: we simply repurpose the original single-agent reward signal as a team reward signal. Default team reward is summarised in Table 4.1.

Task	Goal	Special observations	Reward function
2-Agent Swimmer	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 0.0001\alpha$
2-Agent Reacher	Fingertip (green) needs to reach target at random location (red).	Target is only visible to green agent.	$-\lVert\text{distance from fingertip to target}\rVert_2^2 + \alpha$
2-Agent Ant	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 5 \cdot 10^{-4} \lVert\text{external contact forces}\rVert_2^2 + 0.5\alpha + 1$
2-Agent Ant Diag	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 5 \cdot 10^{-4} \lVert\text{external contact forces}\rVert_2^2 + 0.5\alpha + 1$
2-Agent HalfCheetah	Maximise +ve x-speed.	-	$\frac{\Delta x}{\Delta t} + 0.1\alpha$
2-Agent Humanoid	Maximise +ve x-speed.	-	$0.25\frac{\Delta x}{\Delta t} + \min(10, 5 \cdot 10^{-6} \lVert\text{external contact forces}\rVert_2^2)$
2-Agent HumanoidStandup	Maximise +ve x-speed.	-	$\frac{y}{\Delta t} + \min(10, 5 \cdot 10^{-6} \lVert\text{external contact forces}\rVert_2^2)$
3-Agent Hopper	Maximise +ve x-speed.	-	$\frac{\Delta x}{\Delta t} + 0.001\alpha + 1.0$
4-Agent Ant	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 5 \cdot 10^{-4} \lVert\text{external contact forces}\rVert_2^2 + 0.5\alpha + 1$
6-Agent HalfCheetah	Maximise +ve x-speed.	-	$0.25\frac{\Delta x}{\Delta t} + \min(10, 5 \cdot 10^{-6} \lVert\text{external contact forces}\rVert_2^2)$
ManyAgent Swimmer	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 0.0001\alpha$
ManyAgent Ant	Maximise +ve x-speed.	All agents can observe velocities of the central torso.	$\frac{\Delta x}{\Delta t} + 5 \cdot 10^{-4} \lVert\text{external contact forces}\rVert_2^2 + 0.5\alpha + 1$

Table 4.1: Overview of tasks contained in MAMuJoCo. We define α as an action regularisation term $-\lVert\mathbf{u}\rVert_2^2$.

The most similar existing environments, though not as diverse as MAMuJoCo, are the decomposed MuJoCo environments Centipede and Snakes (Wang et al., 2018). The latter is similar to MAMuJoCo's 2-Agent Swimmer. Ackermann et al. (2019) evaluate on one environment similar to a configuration of 2-Agent Ant, but,

similarly to Gupta et al. (2017), do not consider tasks across different numbers of agents and MuJoCo scenarios.

[A] The following section was taken from the paper's appendix.

4.6 Environment Details

4.6.1 Continuous Matrix Game

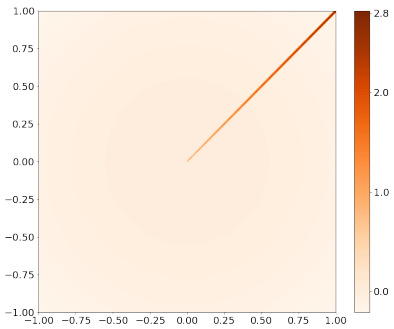


Figure 4.5: The continuous matrix game.

Figure 4.5 illustrates the continuous matrix game with two agents. There is a narrow path (shown in red) starting from the origin $(0, 0)$ to $(1, 1)$, in which the reward gradually increases. Everywhere else there is a small punishment moving away from the origin, increasing in magnitude further from the origin.

4.6.2 Continuous Predator-Prey

We consider the mixed *simple tag* environment (Figure 4.6) introduced by Lowe et al. (2017), which is a variant of the classic predator-prey game. Three slower cooperating circular agents (red), each with continuous movement action spaces $u^a \in \mathbb{R}^2$, must catch a faster circular prey (green) on a randomly generated two-dimensional toroidal plane with two large landmarks blocking the way.

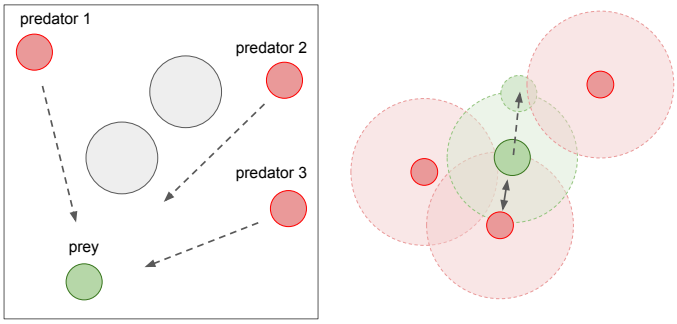


Figure 4.6: Continuous Predator-Prey. **Left:** Top-down view of toroidal plane, with predators (red), prey (green) and obstacles (grey). **Right:** Illustration of the prey's avoidance heuristic. Observation radii of both agents and prey are indicated.

To obtain a purely cooperative environment, we replace the prey's policy by a hard-coded heuristic, that, at any time step, moves the prey to the sampled position with the largest distance to the closest predator. If one of the cooperative agents collides with the prey, a team reward of $+10$ is emitted; otherwise, no reward is given. In the original simple tag environment, each agent can observe the relative positions of the other two agents, the relative position and velocity of the prey, and the relative positions of the landmarks. This means each agent's private observation provides an almost complete representation of the true state of the environment.

To introduce partial observability to the environment, we add an agent *view radius*, which restricts the agents from receiving information about other entities (including all landmarks, the other two agents, and the prey) that are out of range. Specifically, we set the view radius such that the agents can only observe other agents roughly 60% of the time. We refer to this environment as Continuous Predator-Prey.

In addition, we implement a variant of our Continuous Predator-Prey task (with 3 agents and 1 prey), where the reward function is modified to make the task *nonmonotonic*. Specifically, if one agent collides with the prey while at least another one being close enough, a team reward of $+10$ is given. However, if only one agent collides with the prey without any other agent being close enough, a negative team reward of -1 is given. Otherwise, no reward is provided.

4.6.3 Multi-Agent MuJoCo

All MAMuJoCo environments we tested are configured according to its default configuration, where each agent can observe only positions (not velocities) of its own body parts and at graph distances greater than zero. In ManyAgent Swimmer, we configure the number of agents to be 10, each controlling a consecutive segment of length 2. We thus refer to this environment as ManyAgent Swimmer [10x2]. We set maximum observation distances to $k = 0$ (which means each agent can observe only positions of its own body parts) for all three environments tested, including 2-Agent Humanoid, 2-Agent HumanoidStandup, and ManyAgent Swimer [10x2]. Default team reward is used (see Table 4.1).

4.6.4 SMAC

SMAC consists of a set of complex StarCraft II micromanagement tasks that are carefully designed to study decentralised multi-agent control. The tasks in SMAC involve combat between two armies of units. The first army is controlled by a group of learned allied agents. The second army consists of enemy units controlled by the built-in heuristic AI. The goal of the allied agents is to defeat the enemy units in battle, to maximise the win rate. The action space consists of a set of discrete actions: `move` in four cardinal directions, `attack` any selected enemy (available if the enemy is within the agent's shooting range), `stop`, and `noop`. Hence the number of actions increases as the number of enemies increases. All experiments on SMAC use the default reward and observation settings of the SMAC benchmark (Samvelyan et al., 2019).

4.7 Experimental Results

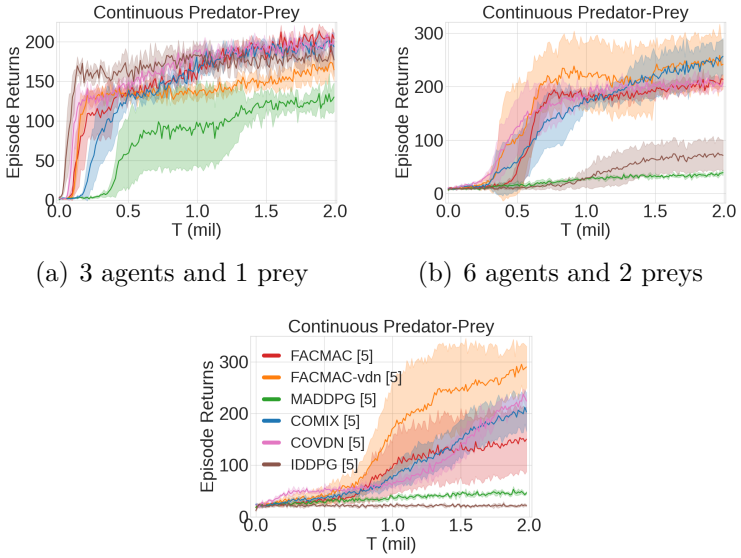
In this section we present our experimental results on our cooperative variants of the continuous *simple tag* environment introduced by Lowe et al. (2017) (we refer

to this environment as Continuous Predator-Prey), our novel continuous benchmark MAMuJoCo, and the challenging SMAC[5] (Samvelyan et al., 2019) benchmark with discrete action spaces. In discrete cooperative tasks, we compare with state-of-the-art multi-agent actor-critic algorithms MADDPG (Lowe et al., 2017), COMA (Foerster et al., 2018), CentralV (Foerster et al., 2018), DOP (Wang et al., 2021), VDAC-mix (Samvelyan et al., 2019), and value-based methods QMIX (Rashid et al., 2018) and QPLEX (Wang et al., 2020b). In continuous cooperative tasks, we compare with MADDPG (Lowe et al., 2017) and independent DDPG (IDDPG), as well as COVDN and COMIX, two novel baselines described below. We also explore different forms of critic factorisation to better understand the advantages of factoring a centralised critic. More details about the environments, experimental setup, and training details are included in Appendix 4.6 and 4.8.

COVDN and COMIX. We find that not many multi-agent value-based methods work off the shelf with continuous actions. To compare FACMAC against value-based approaches in continuous cooperative tasks, we use existing continuous Q-learning approaches in single-agent settings to extend VDN and QMIX to continuous action spaces. Specifically, we introduce COVDN and COMIX, which use VDN-style and QMIX-style factorisation respectively and both perform approximate greedy action selection using the cross-entropy method (CEM) (de Boer et al., 2005). CEM is a sampling-based derivative-free heuristic search method that has been successfully used to find approximate maxima of nonconvex Q-networks in single-agent robotic control tasks (Kalashnikov et al., 2018). The centralised but factored Q_{tot} allows us to use CEM to sample actions for each agent independently and to use the per-agent utility Q_a to guide the selection of maximal actions. We do not consider COVDN and COMIX significant algorithmic contributions but instead merely baseline algorithms. See Appendix 4.4 for more details about them.

FACMAC outperforms MADDPG and other baselines in both discrete and continuous action tasks. Figure 4.7 and 4.8 illustrate the mean episode return attained by different methods on Continuous Predator-Prey with varying number of agents and different MAMuJoCo tasks, respectively. We can see that FACMAC significantly outperforms MADDPG on all these continuous cooperative tasks, both in terms of absolute performance and learning speed. On discrete SMAC tasks, Figure 4.9 shows that FACMAC performs significantly better than MADDPG on 4 out of 6 maps we tested, and achieves similar performance to MADDPG on the other 2 maps. Additionally, on all 6 SMAC maps, FACMAC significantly outperforms all multi-agent actor-critic baselines (COMA, CentralV, DOP, and VDAC-mix), while DOP is recently claimed to be the first multi-agent actor-critic method that outperforms state-of-the-art valued-based methods on SMAC. FACMAC is also competitive with state-of-the-art value-based methods (QMIX and QPLEX), with significantly better performance on *MMM*, *bane_vs_bane*, *MMM2*, and *27m_vs_30m*. These results demonstrate the benefits of our method for improving performance in challenging cooperative tasks with discrete and continuous

[5]We utilise SC2.4.10., which is used by the latest PyMARL (Samvelyan et al., 2019) framework. The original results reported in Samvelyan et al. (2019) and Rashid et al. (2020b) use SC2.4.6. Performance is **not** always comparable across versions.



(a) 3 agents and 1 prey

(b) 6 agents and 2 preys

(c) 9 agents and 3 preys

Figure 4.7: Mean episode return on Continuous Predator-Prey with different number of agents and preys. The mean across 5 seeds is plotted and the 95% confidence interval is shown shaded. The numbers in square brackets in the figure legend represent the number of random seeds used to run each method (similarly for all other figures).

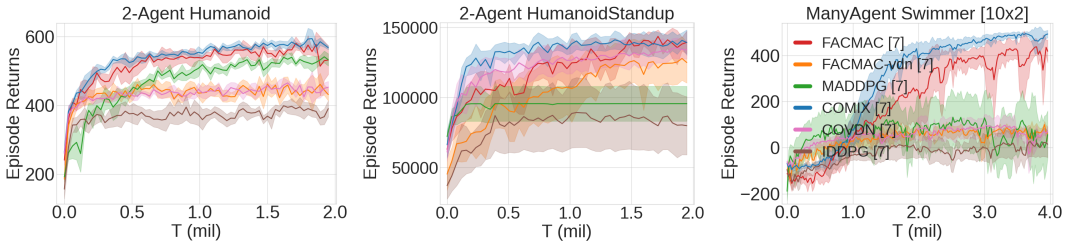


Figure 4.8: Mean episode return on different MAMuJoCo tasks. In ManyAgent Swimmer, we configure the number of agents to be 10, each controlling a consecutive segment of length 2. The mean across 7 seeds is plotted and the 95% confidence interval is shown shaded.

action spaces.

In Continuous Predator-Prey, FACMAC-vdn scales better than FACMAC when the number of agents increases. However, on MAMuJoCo, FACMAC-vdn performs drastically worse than FACMAC in 2-Agent Humanoid and ManyAgent Swimmer (with 10 agents), demonstrating the necessity of the non-linear mixing of agent utilities and conditioning on the central state information in order to achieve competitive performance in such tasks. Furthermore, on SMAC, Figure 4.13 in Appendix 4.8.4 shows that FACMAC is noticeably more stable than FACMAC-vdn and FACMAC-vdn-s across different maps, and achieves significantly better performance on the *super hard* map *MMM2*.

Interestingly, we find that FACMAC performs similarly to COMIX on both Con-

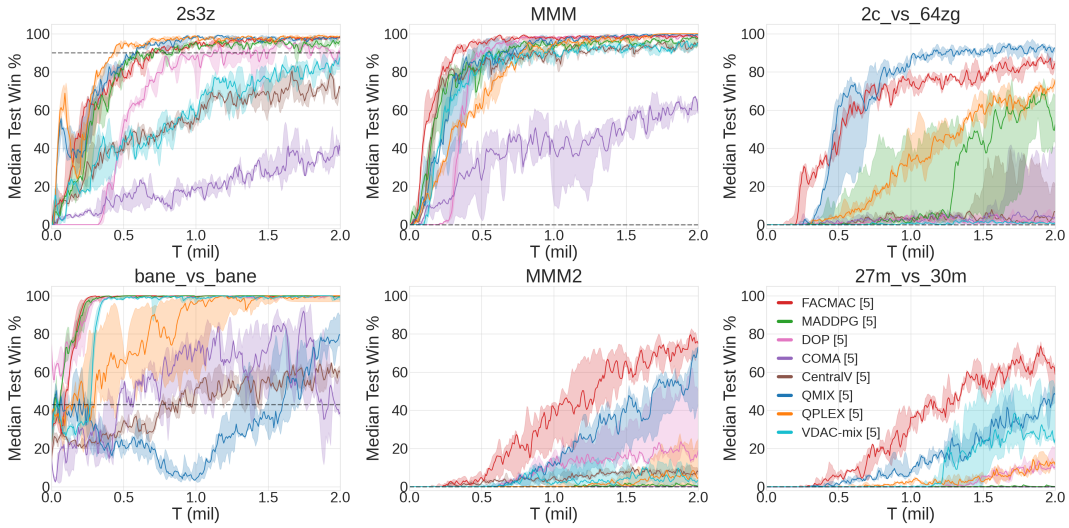


Figure 4.9: Median test win % on six different SMAC maps, including *2s3z* (easy), *MMM* (easy), *2c_vs_64zg* (hard), *bane_vs_bane* (hard), *MMM2* (super hard), and *27m_vs_30m* (super hard). The median across 5 seeds is plotted and the $25-75\%$ percentiles is shown shaded. The performance of the heuristic-based algorithm is shown as a dashed line. We report the median instead of the mean as recommended by Samvelyan et al. (2019) in order to avoid the effect of any outliers.

tinuous Predator-Prey and MAMuJoCo tasks. As FACMAC and COMIX use the same value factorisation as in QMIX and are both off-policy, this suggests that, in these continuous cooperative tasks, factorisation of the joint Q-value function plays a greater role in performance than the underlying algorithmic choices. On SMAC, however, FACMAC performs significantly better than QMIX on *MMM*, *bane_vs_bane*, *MMM2*, and *27m_vs_30m*. For instance, on *bane_vs_bane*, a task with 24 agents, while QMIX struggles to find the optimal policy with 2 million timesteps, FACMAC, with exactly the same value factorisation, can quickly recover the optimal policy and achieve 100% test win rate. This shows the convergence advantages of policy gradient methods in this type of multi-agent settings (Wang et al., 2020a).

FACMAC scales better as the number of agents (and/or actions) and the complexity of the task increases. As shown in Figure 4.7(b) and 4.7(c), MADDPG performs poorly if we increase the number of agents in Continuous Predator-Prey, while both FACMAC and FACMAC-vdn achieve significantly better performance. The monolithic critic in MADDPG simply concatenates all agents' observations into a single input vector, which can be quite large when there are many agents and/or entities and make it more difficult to learn a good critic. Factoring the critic enables scalable critic learning, by combining individual agent utilities that condition on much smaller observations into a joint action-value function. On MAMuJoCo (shown in Figure 4.8), similarly, the largest performance gap between FACMAC and MADDPG can be seen on ManyAgent Swimmer (with 10 agents), a task with the largest number of agents among three MAMuJoCo tasks tested.

On SMAC (shown in Figure 4.9), the largest performance gap between FACMAC

and MADDPG can be seen on the challenging *MMM2* and *27m_vs_30m* with a large number of agents, which are classified as 2 *super hard* SMAC maps due to current methods' poor performance (Samvelyan et al., 2019). We can see that FACMAC is able to learn to consistently defeat the enemy, whereas MADDPG fails to learn anything useful in both tasks. The second largest performance gap between FACMAC and MADDPG can be seen on the hard map *2c_vs_64zg*, where MADDPG not only performs significantly worse but also exhibits significantly more variance than FACMAC across seeds. While there are only 2 agents in this scenario, the number of actions each agent can choose is the largest among all 6 maps tested as there are 64 enemies. These results further demonstrate that FACMAC scales better when the number of agents (and/or actions) and the complexity of the tasks increases.

Factoring the critic can better take advantage of our centralised gradient estimator to optimise the agent policies when the number of agents and/or actions is large. We conduct ablation experiments to investigate the influence of factoring the critic and using the centralised gradient estimator in our method. FACMAC (without CPG) is our method without the centralised policy gradient. It uses a naive adaptation of the deterministic policy gradient used in MADDPG (shown in (4.6)). Thus, the only difference between FACMAC (without CPG) and MADDPG is that the previous one learns a non-linearly factored critic while the latter one learns a monolithic critic. We also evaluate MADDPG with our centralised policy gradient and refer to it as MADDPG (with CPG).

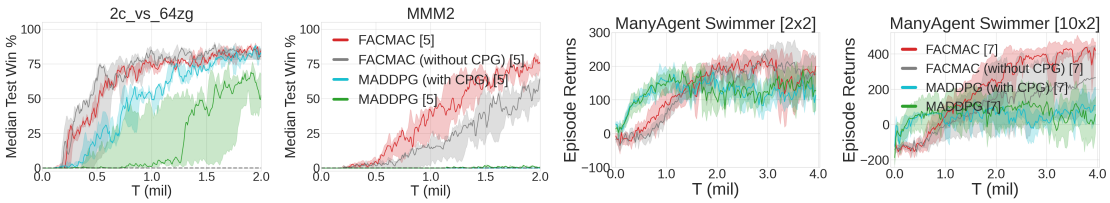


Figure 4.10: Ablations for different FACMAC components on SMAC and MAMuJoCo tasks.

Figure 4.10 shows the ablation results on SMAC and MAMuJoCo. We can see that FACMAC (without CPG) performs significantly better than MADDPG on both SMAC maps tested, demonstrating the advantages of factoring the critic in challenging coordination problems. With the centralised policy gradient, MADDPG (with CPG) performs significantly better than MADDPG on *2c_vs_64zg*. However, on the harder map *MMM2*, MADDPG with both policy gradients fail to learn anything useful. By contrast, FACMAC significantly outperforms FACMAC (without CPG) on *MMM2*, and has lower variance across seeds on *2c_vs_64zg*. Furthermore, on ManyAgent Swimmer with 2 agents, our centralised gradient estimator does not affect the performance of both MADDPG and FACMAC. However, on the same task with 10 agents, using the centralised gradient estimator significantly improves the learning performance when learning a factored critic. These results demonstrate that factoring the critic can better take advantage of our centralised gradient estimator to optimise the agent policies when the number of

agents and/or actions is large.

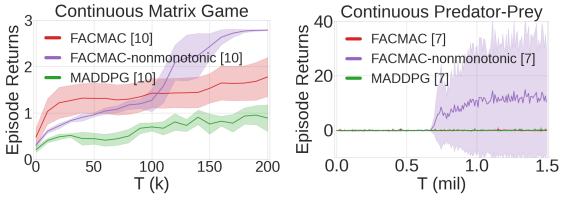


Figure 4.11: Mean episode return on (**Left**) Continuous Matrix Game and (**Right**) a variant of our Continuous Predator-Prey task (with 3 agents and 1 prey) with *nonmonotonic* value functions.

Nonmonotonically factored critics can solve tasks that cannot be solved with monolithic or monotonically factored critics. In our multi-agent actor-critic framework, there are no inherent constraints on factoring the critic, we thus also employ a nonmonotonic factorisation and refer to it as FACMAC-nonmonotonic (as discussed in Section 4.3.1). As shown in Figure 4.11 (left), on our continuous matrix game (as discussed in Section 4.3.2), FACMAC-nonmonotonic can robustly learn the optimal policy, while both FACMAC and MADDPG converge to some sub-optimal policy within $200k$ timesteps. On a variant of Continuous Predator-Prey task (see Appendix 4.6.2 for more details about this task) with *nonmonotonic* value functions (i.e., an agent's ordering over its own actions depends on other agents' actions (Mahajan et al., 2019)), Figure 4.11 (right) shows that both FACMAC and MADDPG fail to learn anything useful, while FACMAC-nonmonotonic successfully learns to capture the prey. These results demonstrate that nonmonotonically factored critics can solve tasks that cannot be solved with monolithic or monotonically factored critics.

It is important to note that the relative performance of FACMAC and FACMAC-nonmonotonic is task dependent. On the original Continuous Predator-Prey task (with 3 agents and 1 prey), FACMAC-nonmonotonic yields similar performance to FACMAC (see Figure 4.12 in Appendix 4.8.4). On SMAC (see Figure 4.13 in Appendix 4.8.4), FACMAC-nonmonotonic performs similarly to FACMAC on easy maps, but exhibits significantly worse performance on harder maps. This shows that, in this type of tasks, using an unconstrained factored critic could lead to an increase in learning difficulty. We thus expect FACMAC-nonmonotonic to be more useful in tasks with nonmonotonic value functions.

Future ablation studies could compare MADDPG performance to that of DDPG within MMDP and MPOMDP settings. Doing so would help entangle the relative benefits of value factorisation, history dependent policies, and joint action learning. For Multi-Agent Mujoco, we would expect MPOMDP performance to be similar to MMDP performance, given the joint agent observations will amount to the central state on most scenarios. These results, however, are not directly relevant to the main

purpose of this paper's experiments, which is the relative performance comparison between both novel and existing deep multi-agent reinforcement learning algorithms.

[A] The following section was taken from the paper's appendix.

4.8 Experimental Details

We evaluate the performance of each method using the following procedure: for each run of a method, we pause training every fixed number of timesteps (2000 timesteps for Continuous Predator-Prey, 4000 timesteps for MAMuJoCo, and 10000 timesteps for SMAC) and run a fixed number of independent episodes (10 episodes for Continuous Predator-Prey and MAMuJoCo, and 32 episodes for SMAC) with each agent performing action selection greedily in a decentralised fashion. On both Continuous Predator-Prey and MAMuJoCo, the mean value of these episode returns are used to evaluate the performance of the learned policies. On SMAC, we use the median test win rate (i.e., the percentage of the 32 episodes where the agents defeat all enemy units within the permitted time limit) to evaluate the learned policies, as in (Samvelyan et al., 2019). All experiments are carried out on NVIDIA GeForce GTX 1080 GPU.

4.8.1 Continuous Predator-Prey

In value-based methods COVDN and COMIX, the architecture of the shared agent network 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. In actor-critic methods FACMAC, FACMAC-vdn, MADDPG, and IDDPG, the architecture of the shared agent network is also a DRQN with a recurrent layer comprised of a GRU with a 64-dimensional hidden state, with a fully-connected layer before and after, while the final output layer is a tanh layer, to bound actions. The shared critic network

is a MLP with 2 hidden layers of 64 units and ReLU non-linearities. All agent networks receive the current local observation and last individual action as input. In MADDPG, the centralised critic receives the global state and the joint action of all agents as input. The global state consists of the joint observations of all agents in Continuous Predator-Prey. In other actor-critic methods, there is a shared critic network that approximates per-agent utilities, which receives each agent's local observation and individual action as input.

During training and testing, we restrict each episode to have a length of 25 time steps. Training lasts for 2 million timesteps. To encourage exploration, we use uncorrelated, mean-zero Gaussian noise with $\sigma = 0.1$ during training (for all 2 million timesteps). We set $\gamma = 0.85$ for all experiments. The replay buffer contains the most recent 10^6 transitions. We train on a batch size of 1024 after every timestep. For the soft target network updates we use $\tau = 0.001$. All neural networks (actor and critic) are trained using Adam optimiser with a learning rate of 0.01. To evaluate the learning performance, the training is paused after every 2000 timesteps during which 10 independent test episodes are run with agents performing action selection greedily in a decentralised fashion.

4.8.2 Multi-Agent MuJoCo

In all value-based methods, the architecture of all agent networks is a MLP with 2 hidden layers with 400 and 300 units respectively, similar to the setting used in OpenAI Spinning Up.[6] All agent networks use ReLU non-linearities for all hidden layers. In all actor-critic methods, the architecture of the shared agent network and critic network is also a MLP with 2 hidden layers with 400 and 300 units respectively, while the final output layer of the actor network is a tanh layer, to bound the actions. In all value-based methods, the agent receives its current local

[6] `https://spinningup.openai.com/en/latest/`.

observation as input. In MADDPG, the centralised critic receives the global state and the joint action of all agents as input. The global state consists of the full state information returned by the original OpenAI Gym (Brockman et al., 2016). In other actor-critic methods, there is a shared critic network that approximates per-agent utilities, which receives each agent's local observation and individual action as input.

During training and testing, we restrict each episode to have a length of 1000 time steps. Training lasts for 2 million or 4 million timesteps. To encourage exploration, we use uncorrelated, mean-zero Gaussian noise with $\sigma = 0.1$ during training. We also use the same trick as in OpenAI Spinning Up to improve exploration at the start of training. For a fixed number of steps at the beginning (we set it to be 10000), the agent takes actions which are sampled from a uniform random distribution over valid actions. After that, it returns to normal Gaussian exploration. We set $\gamma = 0.99$ for all experiments. The replay buffer contains the most recent 10^6 transitions. We train on a batch size of 100 after every timestep. For the soft target network updates we use $\tau = 0.001$. All neural networks (actor and critic) are trained using Adam optimiser with a learning rate of 0.001. To evaluate the learning performance, the training is paused after every 4000 timesteps during which 10 independent test episodes are run with agents performing action selection greedily in a decentralised fashion.

4.8.3 SMAC

For baseline algorithms DOP (Wang et al., 2021), COMA (Foerster et al., 2018), CentralV (Foerster et al., 2018), VDAC-mix (Su et al., 2021), QMIX (Rashid et al., 2020b), and QPLEX (Wang et al., 2020b), we use the the same training setup as provided by their authors where the hyperparameters have been fine-tuned on the SMAC benchmark.

Most of our training hyperparameters for FACMAC and MADDPG (Lowe et al., 2017) follow (Rashid et al., 2020b). In both methods, the architecture of the shared actor network 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. The shared critic network is a MLP with 2 hidden layers of 64 units and ReLU non-linearities. Exploration is performed during training using a scheme similar to COMA (Foerster et al., 2018). Action probabilities are produced from the final layer of the actor network, z, via a bounded softmax distribution that lower-bounds the probability of any given action by $\epsilon/|U|$: $P(u) = (1-\epsilon)\text{softmax}_u + \epsilon/|U|$, where $|U|$ is the size of the joint action space. Throughout the training, we anneal ϵ linearly from 0.5 to 0.05 over $50k$ timesteps and keep it constant for the rest of the training. 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. In MADDPG, we use a target network for the actor and critic, respectively. In FACMAC, we use a target network for the actor, critic, and mixing network, respectively. All target networks are periodically updated every 200 training steps. All neural networks are trained using Adam optimiser with learning rate 0.0025 for the actor network and 0.0005 for the critic network. We set $\gamma = 0.99$ for all experiments.

The architecture of the mixing network in FACMAC follows (Rashid et al., 2020b). It consists of a single hidden layer of 32 units with an ELU non-linearity. The weights of the mixing network are produced by separate hypernetworks. 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 activation function (to acheive non-negativity) and then resized into a matrix of appropriate size.

[A] The following section was taken from the paper's appendix.

4.8.4 Additional Results on Different Critic Factorisations

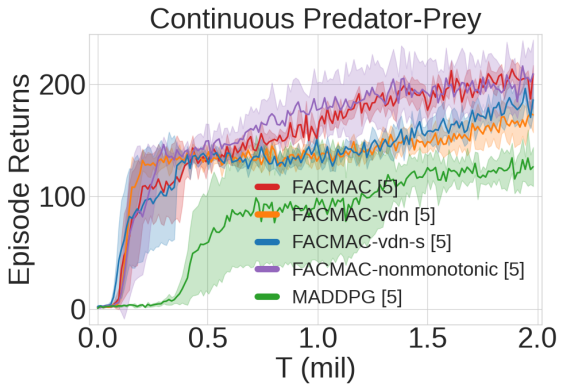
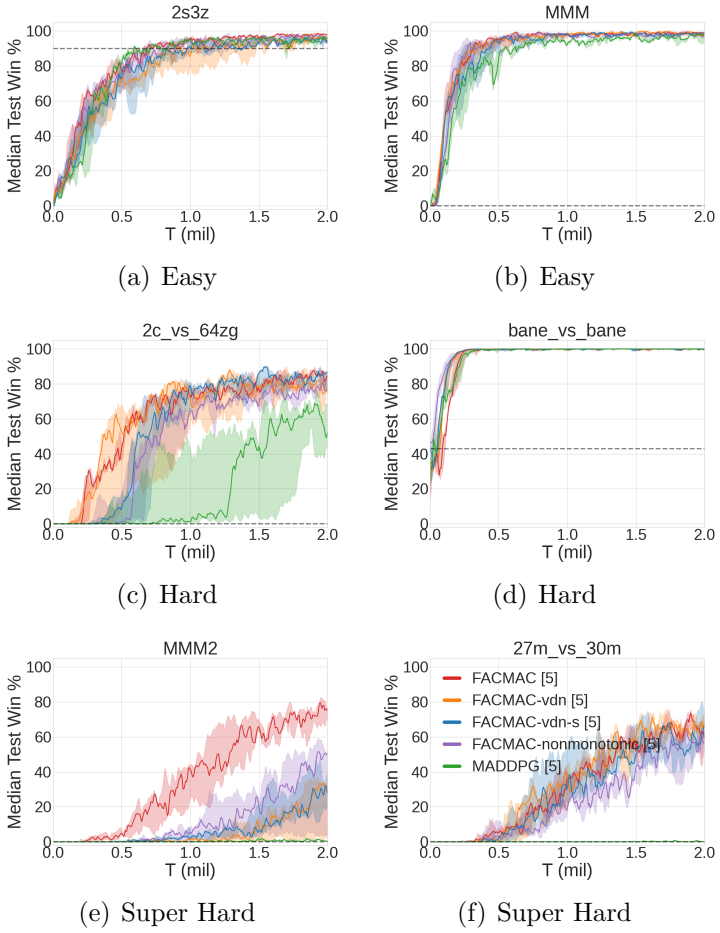


Figure 4.12: Mean episode return on our Continuous Predator-Prey task (with 3 agents and 1 prey).



(a) Easy (b) Easy

(c) Hard (d) Hard

(e) Super Hard (f) Super Hard

Figure 4.13: Median test win % on six different SMAC maps: (a) *2s3z* (easy), (b) *MMM* (easy), (c) *2c_vs_64zg* (hard), (d) *bane_vs_bane* (hard), (e) *MMM2* (super hard), and (f) *27m_vs_30m* (super hard), comparing FACMAC with different forms of critic factorisations.

4.9 Related Work

Value function factorisation (Koller and Parr, 1999) has been widely employed in value-based MARL algorithms. VDN (Sunehag et al., 2018) and QMIX (Rashid et al., 2018) factor the joint action-value function into per-agent utilities that are combined via a simple summation or a monotonic mixing function respectively, to ensure consistency between the arg max of the centralised joint-action value function and the arg max of the decentralised polices. This monotonicity constraint, however, prevents them from representing joint action-value functions that are characterised as *nonmonotonic* (Mahajan et al., 2019), i.e., an agent's ordering over its own actions depends on other agents' actions. A large number of recent works (Son et al., 2019; Yang et al., 2020; Rashid et al., 2020a; Wang et al., 2020b; Son et al., 2020) thus focus on developing new value-based MARL algorithms that address this representational limitation, in order to learn a richer class of action-value functions.

QTRAN (Son et al., 2019) learns an unrestricted joint action-value function and aims to solve a constrained optimisation problem in order to decentralise it, but has been shown to scale poorly to more complex tasks such as SMAC (Mahajan et al., 2019). QPLEX (Wang et al., 2020b) takes advantage of the dueling network architecture to factor the joint action-value function in a manner that does not restrict the representational capacity, whilst also remaining easily decentralisable, but can still fail to solve simple tasks with nonmonotonic value functions (Rashid et al., 2020a). Weighted QMIX (Rashid et al., 2020a) introduces a weighting scheme to place more importance on the better joint actions to learn a richer class of joint action-value functions. QTRAN++ (Son et al., 2020) addresses the gap between the empirical performance and theoretical guarantees of QTRAN. Our multi-agent actor-critic framework with decentralised actors and a centralised but factored critic, by contrast, provides a more direct and simpler way of coping with nonmonotonic tasks as one can simply factor the centralised critic in any manner without constraints. Additionally, our framework can be readily applied to tasks with continuous action spaces, whereas these value-based algorithms require additional algorithmic changes.

Most state-of-the-art multi-agent actor-critic methods (Lowe et al., 2017; Foerster et al., 2018; Iqbal and Sha, 2019; Du et al., 2019) learn a centralised but unfactored critic conditioning on the global state and the joint action to stabilise learning. Even though the joint action-value function they can represent is not restricted, in practice they significantly underperform value-based methods like QMIX on the challenging SMAC benchmark (Rashid et al., 2020a,b). In contrast, FACMAC utilises a centralised but factored critic to allow it to scale to the more complex tasks in SMAC, and follows the centralised policy gradient instead of per-agent policy gradients.

Lyu et al. (2021) recently provide some interesting insights about the pros and cons of centralised and decentralised critics for on-policy actor-critic algorithms. One important issue that they highlight is that merely utilising a centralised critic does not necessarily lead to the learning of more coordinated behaviours. This is

because the use of a per-agent policy gradient can lead to the agents getting stuck in sub-optimal solutions in which no one agents wishes to change their policy, as discussed in 4.3.2. Our centralised policy gradient resolves this issue by taking full advantage of the centralised training paradigm to optimise over the joint-action policy, which allows us to reap the benefits of a centralised critic. Since FACMAC is off-policy, we also benefit immensely from utilising a centralised critic over a decentralised one since we avoid the issues of non-stationarity when training on older data.

Zhou et al. (2020) propose to use a single centralised critic for MADDPG, whose weights are generated by hypernetworks that condition on the state, similarly to QMIX's mixing network without the monotonicity constraints. FACMAC also uses a single centralised critic, but factorises it similarly to QMIX (not just using the mixing network) which allows for more efficient learning on more complex tasks. Of existing work, the deterministic decomposed policy gradients (DOP) algorithm proposed by Wang et al. (2021) is perhaps most similar to our own approach. Deterministic DOP is off-policy and factors the centralised critic as a weighted linear sum of individual agent utilities and a state bias. It is limited to only considering linearly factored critics, which have limited representational capacity, whilst we are free to choose any method of factorisation in FACMAC to allow for the learning of a richer class of action-value functions. While they claim to be the first to introduce the idea of value function factorisation into the multi-agent actor-critic framework, it is actually first explored by Bescuca (2019), where a monotonically factored critic is learned for COMA (Foerster et al., 2018). However, their performance improvement on SMAC is limited since COMA requires on-policy learning and it is not straightforward to extend COMA to continuous action spaces. Furthermore, both works only consider monotonically factored critics, whilst we employ a nonmonotonic factorisation and demonstrate its benefits. We also investigate the benefits of learning a centralised but factored critic more thoroughly, providing a better understanding about the type of tasks that can benefit more from a factored critic. Additionally, both deterministic DOP and LICA (Zhou et al., 2020) use a naive adaptation of the deterministic policy gradient used by MADDPG and suffer from the same problems as discussed in Section 4.3.2, while our centralised policy gradients allow for better coordination across agents in certain tasks.

4.10 Conclusion

This paper presented FACMAC, a multi-agent actor-critic method that learns decentralised policies with a centralised but factored critic, working for both discrete and continuous cooperative tasks. We showed the advantages of both factoring the critic and using the new centralised gradient estimator in our approach. We also introduced a novel benchmark suite MAMuJoCo to demonstrate FACMAC's scalability to more complex continuous tasks. Our results on three different domains demonstrated FACMAC's superior performance over existing MARL algorithms. Future work will explore more forms of nonmonotonic factorisation to tackle tasks with nonmonotonic value functions.

4.10.1 Social Impact

MAMuJoCo is a simulated environment that mimics certain fundamental aspects of robotic actuator chains. Such actuator chains are commonly found in industrial robotics, perhaps most prominently in robot arms on assembly lines, but also a variety of other civil use cases, including remote space exploration or bomb disposal. However, in order to be applicable to such use cases, policies trained in MAMuJoCo first need to undergo a transfer process to the real world, for example by using Sim2Real techniques. MAMuJoCo can not only help assess algorithmic advances, but also contribute toward understanding and mitigating risks inherent to civil autonomous robotics applications. For example, MAMuJoCo could facilitate the development of safe reinforcement learning algorithms that keep actuator parameters of real-world robotic arms within safe ranges, thus avoiding manufacturing errors or injury.

As most developments in robotic control, algorithmic progress based on our benchmark environment, or the novel algorithms introduced in this paper, might ultimately find application in autonomous warfare. MAMuJoCo has been developed with fundamental algorithmic development for civil purposes in mind and does not provide any features that would make it particularly suitable to non-civil use. Any robotic agents trained with reinforcement learning algorithms on our environment, including our novel methods, should be certified with respect to fairness and security before real-world deployment. Like any artificial intelligence system, our proposed framework has the potential to greatly improve human productivity. However, it may also reduce the need for human workers, resulting in job losses.

This concludes our quote.

Coming up next...

In this chapter, we introduced FACMAC, a deterministic policy gradients algorithm for cooperative multi-agent learning that uses centralised policy gradient updates to optimise over the joint action space during centralised training. FACMAC achieves state-of-the-art performance on SMAC. In addition, we introduce a novel benchmark suite for continuous control, Multi-Agent Mujoco and use it to evaluate both FACMAC, as well as COMIX, a novel variant of QMIX 3 that scales to continuous action spaces.

Next, in Chapter 5, we will tackle another challenge of centralised training: While we have, in both Chapters 3 and 4, succeeded in training joint, centralised value functions in CTDE settings and thus address non-stationarity issues related to independent learning, each agent still executes its own, independent policy. In the next Chapter, we show how common knowledge between agents can be used to allow agents to execute joint policies in a fully decentralised fashion.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

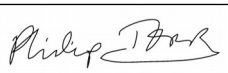
Title of Paper	*FACMAC: Factored Multi-Agent Centralised Policy Gradients*
Publication Status	☐ Published ☐ Accepted for Publication x Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**B. Peng*, T. Rashid*, C. Schroeder de Witt***, P. Kamienny, P. Torr, W. Böhmer, and S. Whiteson. *FACMAC: Factored Multi-Agent Centralised Policy Gradients (2021).* ArXiv: 2003.06709. (*: equal contribution). *Accepted at NeurIPS 2021.*

Student Confirmation

Student Name:	Christian Schroeder de Witt		
Contribution to the Paper	• Generation of Ideas • Software Development • Designed and performed experiments • Wrote preliminary paper with co-authors, edited final paper		
Signature		Date	31.05.2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Professor Philip H.S. Torr			
Supervisor comments			
Signature		Date	04.05.2021

This completed form should be included in the thesis, at the end of the relevant chapter.

Part II

Common Knowledge

5

Multi-Agent Common Knowledge Reinforcement Learning

Contents

5.1 Introduction

In Chapters 3 and 4, we have introduced a variety of novel deep multi-agent reinforcement learning algorithms (QMIX, FACMAC, COMIX, ...) for settings that

allow for centralised learning, but require decentralised execution (see Section 2.2.2 in Chapter 2). These algorithms all admit constructing a centralised joint value function that factors into individual local agent utility functions through monotonic mixing networks, thus putting a constraint on what types of return distributions can be arbitrarily closely approximated.

But is it possible to construct a joint policy that is nevertheless fully decentralisable? We have learnt in 2.2.3 that executing joint policies either requires full observability, or immediate communication channels through which agents can exchange their local observations. A perhaps surprising insight that we now introduce is, however, that even in fully decentralised settings, agents can make advantage of joint learning under certain conditions: Namely, if agents can, at each point in time, dynamically infer what information is common knowledge between themselves and the group of agents they wish to coordinate with, then agents can coordinate based on a commonly-known joint policy that conditions on the group common knowledge. As training is assumed to be centralised, such a commonly known joint policy can be easily shared during before execution.

While not all conceivable Dec-POMDPs may give rise to useful common knowledge between agents, a large class of application-relevant environments does. This class includes environments where agent's observations consist of a field-of-view representation that allows each agent to detect the presence of other agents in their own observation and infer whether the detection is mutual by observing the orientation of the other's agents field-of-view sensors. This so-called field-of-view common knowledge (FoV-CK) in fact arises in almost any cooperative robotic simulation, be it about autonomous drones or self-driving vehicles. The key to FoV-CK to be commonly reconstructable during decentralised execution is that agents are free to exchange information about their observation sensors during

centralised training [1].

In this paper, we develop MACKRL, a novel deep multi-agent reinforcement learning algorithm that is able to efficiently exploit common knowledge that dynamically arises among agent groups of different sizes and compositions. MACKRL is an actor-critic algorithm that features a centralised value function and a tree-structured end-to-end differentiable joint policy that allows action coordination based on group-wise common knowledge when beneficial, and delegation to independent policies otherwise. We show that a pairwise variant of MACKRL achieves competitive performance on the StarCraft Multi-Agent Challenge (SMAC, see chapter (Samvelyan et al., 2019)). We also propose a way in which MACKRL can be scaled to tasks involving a large number of agents, and how robustness to moderate levels of sensor noise can be achieved.

As MACKRL's common knowledge exploiting, group-wise agent controllers constitute a context-specific decomposition into (local) sub-problems, one might intuitively assume this to implies some form of value factorisation (refer to Chapter 3). However, as of time of publication of this thesis, the exact relationship thereof remains unclear and an interesting question for further research.

The term 'sensor noise' is kept purposefully vague. What we refer to are any physical properties or processes that lead two or more agents to reconstruct differing common knowledge representations. Visual sensors, such as e.g. cameras, could experience oversaturation, glare or thermal noise. Even if sensory inputs were to agree between different agents, differing processing software version or temporal hardware contexts could result in incompatible common knowledge reconstructions.

[1] Please note that the unusual case where another agent's sensory equipment fails during execution can, in practice, either be addressed by sending an observable broadcast signal (e.g. a red LED indicates sensor failure) or by concluding sensor failure from repeated failed coordination attempts

In the context of *common knowledge with entities*, agents might in the worst case disagree over the presence of certain entities in their overlapping fields of view. Such 'hard' disagreements might impact coordination far more than 'soft' ones, such as e.g. slight disagreements over relative positionings of mutually perceived entities.

In this paper, we limit empirical investigation of the effect of sensor noise to a very abstract setting, i.e. *bit flip noise* in a matrix game (see Section 5.5.1). While this type of noise may not directly equate to real-world sensor noise in the above sense, we nevertheless manage to illustrate the robustness of learnt function approximators to small changes in sensory input. Further investigation of robustness to more realistic types of sensor noise are left for future work.

We now quote from (Schroeder de Witt et al., 2019), with the quoted text being visually indicated by reduced linespacing. Please note that the quote is literal except for where original supplementary material and appendices have been inserted in-place and internal document references have been updated accordingly.

5.2 Problem Setting

Cooperative multi-agent tasks with n agents $a \in \mathcal{A}$ can be modelled as *decentralised partially observable Markov decision processes* (Dec-POMDPs, Oliehoek et al., 2008a). The state of the system is $s \in \mathcal{S}$. At each time-step, each agent a receives an observation $z^a \in \mathcal{Z}$ and can select an action $u^a_{\text{env}} \in \mathcal{U}^a_{\text{env}}$. We use the env-subscript to denote actions executed by the agents in the environment, as opposed to latent 'actions' that may be taken by higher-level controllers of the hierarchical method introduced in Section 5.4. Given a joint action $\mathbf{u}_{\text{env}} := (u^1_{\text{env}}, \dots, u^n_{\text{env}}) \in \mathcal{U}_{\text{env}}$, the discrete-time system dynamics draw the successive state $s' \in \mathcal{S}$ from the conditional distribution $P(s'|s, \mathbf{u}_{\text{env}})$ and yield a cooperative reward according to the function $r(s, \mathbf{u}_{\text{env}})$.

The agents aim to maximise the discounted return $R_t = \sum_{l=0}^{H} \gamma^l \, r(s_{t+l}, \mathbf{u}_{t+l,\text{env}})$ from episodes of length H. The joint policy $\pi(\mathbf{u}_{\text{env}}|s)$ is restricted to a set of decentralised policies $\pi^a(u^a_{\text{env}}|\tau^a_t)$ that can be executed independently, i.e., each agent's policy conditions only on its own action-observation history $\tau^a_t := [z^a_0, u^a_0, z^a_1, \dots, z^a_t]$. Following previous work (Rashid et al., 2020b; Foerster et al., 2016, 2017, 2018; Kraemer and Banerjee, 2016; Jorge et al., 2016), we allow decentralised policies to be learnt in a centralised fashion.

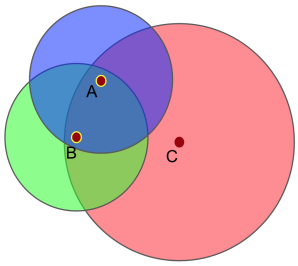


Figure 5.1: Three agents and their fields of view. A and B's locations are common knowledge to A and B as they are within each other's fields of view. Although C can see A and B, it shares no common knowledge with them.

Common knowledge of a group of agents $\mathcal{G}$ refers to facts that all members know, and that "each individual knows that all other individuals know it, each individual knows that all other individuals know that all the individuals know it, and so on" (Osborne and Rubinstein, 1994). Any data ξ that are known to all agents before execution/training, like a shared random seed, are obviously common knowledge. Crucially, every agent $a \in \mathcal{G}$ can deduce the same *history of common knowledge* $\tau_t^{\mathcal{G}}$ from its own history τ_t^a and the commonly known data ξ, that is, $\tau_t^{\mathcal{G}} := \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi) = \mathcal{I}^{\mathcal{G}}(\tau_t^{\bar{a}}, \xi), \forall a, \bar{a} \in \mathcal{G}$. Furthermore, any actions taken by a policy $\pi^{\mathcal{G}}(\mathbf{u}_{\text{env}}^{\mathcal{G}} | \tau_t^{\mathcal{G}})$ over the group's joint action space $\mathcal{U}_{\text{env}}^{\mathcal{G}}$ are themselves common knowledge, if the policy is deterministic or pseudo-random with a shared random seed and conditions only on the common history $\tau_t^{\mathcal{G}}$, i.e. the set formed by restricting each transition tuple within the joint history of agents in $\mathcal{G}$ to what is commonly known in $\mathcal{G}$ at time t. Common knowledge of subgroups $\mathcal{G}' \subset \mathcal{G}$ cannot decrease, that is, $\mathcal{I}^{\mathcal{G}'}(\tau_t^a, \xi) \supseteq \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi)$.

Given a Dec-POMDP with noisy observations, agents in a group $\mathcal{G}$ might not be able to establish true common knowledge even if sensor noise properties are commonly known (Halpern and Moses, 2000). Instead, each agent a can only deduce its own *beliefs* $\tilde{\mathcal{I}}_a^{\mathcal{G}}(\tilde{\tau}_t^a)$ over what is commonly known within $\mathcal{G}$, where $\tilde{\tau}_t^a$ is the agent's belief over what constitutes the groups' common history. Each agent a can then evaluate its own *belief over the group policy* $\tilde{\pi}_a^{\mathcal{G}}(u_{\text{env}}^{\mathcal{G}} | \tilde{\tau}_t^{\mathcal{G}})$. In order to minimize the probability of disagreement during decentralized group action selection, agents in $\mathcal{G}$ can perform *optimal correlated sampling* based on a shared random seed (Holenstein, 2007; Bavarian et al., 2020).

Probabilistic common knowledge measures how much group common knowledge group members can infer at high probability given they share information but possibly no true common knowledge (Krasucki et al., 1991). Formally, a group of agents $\mathcal{G}$ admits *probabilistic common knowledge* iff $\forall x \in \bigcup_{n=0}^{\infty} \{1, \ldots, |\mathcal{G}|\}^n$, $inf(x) > 0$ where $inf(x)$ measures the amount of information that all agents in x have about the partitions underlying each other's individual knowledge accessibility relations (Krasucki et al., 1991, Definitions 8,13). *Probabilistic common knowledge* therefore denotes a weaker form of group knowledge that can be exploited even if no true common knowledge exists, such as e.g. if observations are noisy.

Learning under common knowledge (LuCK) is a novel cooperative multi-agent reinforcement learning setting, where a Dec-POMDP is augmented by a common knowledge function $\mathcal{I}^{\mathcal{G}}$ (or *probabilistic* common knowledge function $\tilde{\mathcal{I}}_a^{\mathcal{G}}$). Groups of agents $\mathcal{G}$ can coordinate by learning policies that condition on their common knowledge. In this paper $\mathcal{I}^{\mathcal{G}}$ (or $\tilde{\mathcal{I}}_a^{\mathcal{G}}$) is fixed apriori, but it could also be learnt during training. The setting accommodates a wide range of real-world and simulated multi-agent tasks. Whenever a task is cooperative and learning is centralised, then agents can naturally learn suitable $\mathcal{I}^{\mathcal{G}}$ or $\tilde{\mathcal{I}}_a^{\mathcal{G}}$. Policy parameters can be exchanged during training as well and thus become part of the commonly known data ξ. Joint policies where coordinated decisions of a group $\mathcal{G}$ only condition on the common knowledge of $\mathcal{G}$ can be executed in a fully decentralised fashion. In Section 5.4 we introduce MACKRL, which uses centralised training to learn fully decentralised policies under common knowledge.

Field-of-view common knowledge is a form of *complete-history common knowledge* (Halpern and Moses, 2000), that arises within a Dec-POMDP if agents can deduce parts of other agents' observations from their own. In this case, an agent group's common knowledge is the intersection of observations that all members can reconstruct from each other. In Appendix 5.3 we formalise this concept and show that, under some assumptions, common knowledge is the intersection of all agents' sets of visible objects, if and only if all agents can see each other. Figure 5.1 shows an example for three agents with circular fields of view. If observations are noisy, each agent bases its belief on its own noisy observations thus inducing an equivalent form of probabilistic common knowledge $\tilde{\mathcal{I}}_a^{\mathcal{G}}$.

Field-of-view common knowledge naturally occurs in many interesting real-world tasks, such as autonomous driving (Cao et al., 2013) and robo-soccer (Genter et al., 2017), as well as in simulated benchmarks such as StarCraft II (Vinyals et al., 2017). A large number of cooperative multi-agent tasks can therefore benefit from common knowledge-based coordination introduced in this paper.

[A] The following section is partially taken from the paper's appendix.

5.3 Common Knowledge with Entities

To exploit a particular form of *field-of-view common knowledge* with MACKRL, we formalise an instance of a Dec-POMDP, in which such common knowledge naturally arises. In this Dec-POMDP, the state s is composed of a number of entities $e \in \mathcal{E}$, with state features s^e, i.e., $s = \{s^e \,|\, e \in \mathcal{E}\}$. Some entities are agents $a \in \mathcal{A} \subseteq \mathcal{E}$. Other entities could be enemies, obstacles, or goals.

The agents have a particular form of partial observability: the observation z^a contains the subset of state features s^e from all the entities e that a can see. Whether a can observe e is determined by the binary mask $\mu^a\left(s^a, s^e\right) \in \{\top, \bot\}$ over the agent's and entity's observable features. An agent can always observe itself, i.e., $\mu^a(s^a, s^a) = \top, \forall a \in \mathcal{A}$. The set of all entities the agent can see is therefore $\mathcal{M}^a := \{e \,|\, \mu^a(s^a, s^e)\} \subseteq \mathcal{E}$, and the agent's observation is specified by the deterministic observation function $o(s, a)$ such that $z^a = o(s, a) = \{s^e \,|\, e \in \mathcal{M}^a\} \in \mathcal{Z}$. In the example of Figure 5.1, $\mathcal{M}^A = \mathcal{M}^B = \{A, B\}$ and $\mathcal{M}^C = \{A, B, C\}$.

This special Dec-POMDP yields perceptual aliasing in which the state features of each entity are either accurately observed or completely occluded. The Dec-POMDP could be augmented with additional state features that do not correspond to entities, as well as additional possibly noisy observation features, without disrupting the common knowledge we establish about entities. For simplicity, we omit such additions.

A key property of the binary mask μ^a is that it depends only on the features s^a and s^e to determine whether agent a can see entity e. If we assume that an agent a's mask μ^a is common knowledge, then this means that another agent b, that can see a and e, i.e., $a, e \in \mathcal{M}^b$, can deduce whether a can also see e. This assumption can give rise to common knowledge about entities. Figure 5.1 demonstrates this for 3 agents with commonly known observation radii.

The *mutual knowledge* $\mathcal{M}^{\mathcal{G}}$ of a group of agents $\mathcal{G} \subseteq \mathcal{A}$ in state s is the set of entities that all agents in the group can see in that state: $\mathcal{M}^{\mathcal{G}} := \cap_{a \in \mathcal{G}} \mathcal{M}^a$. However, mutual knowledge does not imply common knowledge. Instead, the *common knowledge* $\mathcal{I}^{\mathcal{G}}$ of group $\mathcal{G}$ in state $s \in \mathcal{S}$ is the set of entities such that all agents in $\mathcal{G}$ see $\mathcal{I}^{\mathcal{G}}$, know that all other agents in $\mathcal{G}$ see $\mathcal{I}^{\mathcal{G}}$, know that they know that all other agents see $\mathcal{I}^{\mathcal{G}}$, and so forth (Osborne and Rubinstein, 1994).

To know that another agent b also sees $e \in \mathcal{E}$, agent a must see b and b must see e,

i.e., $\mu^a(s^a, s^b) \wedge \mu^b(s^b, s^e)$. Common knowledge $\mathcal{I}^{\mathcal{G}}$ can then be formalised recursively for every agent $a \in \mathcal{G}$ as:

$$\mathcal{I}_0^a := \mathcal{M}^a, \qquad \mathcal{I}_m^a := \bigcap_{b \in \mathcal{G}} \left\{ e \in \mathcal{I}_{m-1}^b \,|\, \mu^a(s^a, s^b) \right\}, \qquad \mathcal{I}^{\mathcal{G}} := \lim_{m \to \infty} \mathcal{I}_m^a . \tag{5.1}$$

This definition formalises the above description that common knowledge is the set of entities that a group member sees ($m = 0$), that it knows all other group members see as well ($m = 1$), and so forth ad infinitum. In the example of Figure 5.1, $\mathcal{I}^{AB} = \{A, B\}$ and $\mathcal{I}^{AC} = \mathcal{I}^{BC} = \mathcal{I}^{ABC} = \varnothing$. To see this, let us go through Equation 5.1 step-by-step. Clearly, $\mathcal{M}^A = \left\{ e | \mu^A(s^A, s^e) \right\} = \{A, B\}$, $\mathcal{M}^B = \left\{ e | \mu^B(s^B, s^e) \right\} = \{B, A\} = \{A, B\}$ and $\mathcal{M}^C = \left\{ e | \mu^C(s^C, s^e) \right\} = \{A, B, C\}$. Hence, $\mathcal{I}_0^A = \{A, B\}$, $\mathcal{I}_0^B = \{B, A\}$ and $\mathcal{I}_0^C = \{A, B, C\}$. Applying Equation 5.1 again, we find

$$\mathcal{I}_1^A = \left\{ e \in \mathcal{I}_0^A \,\middle|\, \mu^A(s^A, s^A) \right\} \wedge \left\{ e \in \mathcal{I}_0^B \,\middle|\, \mu^B(s^B, s^A) \right\} \wedge \left\{ e \in \mathcal{I}_0^C \,\middle|\, \mu^C(s^C, s^A) \right\}$$

which simplifies to $\left\{ e \in \mathcal{I}_B^A \right\} \wedge \left\{ e \in \mathcal{I}_0^B \right\} = \mathcal{M}^A \wedge \mathcal{M}^B = \{A, B\}$. Clearly, as we take the limit $m \to \infty$, $\mathcal{I}_m^A$ will not change further. Hence we have $\mathcal{I}^{AB} = \mathcal{I}_1^A$. Rinse and repeat for $\mathcal{I}^{ABC}$.

The following lemma establishes that, in our setting, if a group of agents can all see each other, their common knowledge is their mutual knowledge.

Lemma 1. In the setting described in this Section, and when all masks are known to all agents, the common knowledge of a group of agents $\mathcal{G}$ in state $s \in \mathcal{S}$ is

$$\mathcal{I}^{\mathcal{G}} = \begin{cases} \mathcal{M}^{\mathcal{G}}, & \text{if } \bigwedge_{a,b \in \mathcal{G}} \mu^a(s^a, s^b) \\ \varnothing, & \text{otherwise} \end{cases} . \tag{5.2}$$

Proof. The lemma follows by induction on m. The recursive definition of common

knowledge (5.1) holds trivially if $\mathcal{I}^{\mathcal{G}} = \varnothing$. Starting from the knowledge of any agent a in state s, $\mathcal{I}_0^a = \mathcal{M}^a$, definition (5.1) yields:

$$\mathcal{I}_1^a = \begin{cases} \mathcal{M}^{\mathcal{G}}, & \text{if } \bigwedge\limits_{b\in\mathcal{G}} \mu^a(s^a, s^b) \\ \varnothing, & \text{otherwise} \end{cases} .$$

Next we show inductively that if all agents in group $\mathcal{G}$ know the mutual knowledge $\mathcal{M}^{\mathcal{G}}$ of state s at some iteration m, that is, $\mathcal{I}_m^c \overset{\text{ind.}}{=} \mathcal{M}^{\mathcal{G}}$, then this mutual knowledge becomes common knowledge two iterations later. Applying the definition (5.1) for any agent $a \in \mathcal{G}$ twice yields:

$$\begin{aligned} \mathcal{I}_{m+2}^a &= \bigcap_{b\in\mathcal{G}} \left\{ e \in \left[\bigcap_{c\in\mathcal{G}} \left\{ e' \in \mathcal{I}_m^c \middle| \mu^b(s^b, s^c) \right\} \right] \middle| \mu^a(s^a, s^b) \right\} \\ &= \bigcap_{b\in\mathcal{G}} \bigcap_{c\in\mathcal{G}} \left\{ e \in \mathcal{I}_m^c \middle| \mu^a(s^a, s^b) \wedge \mu^b(s^b, s^c) \right\} \\ &= \left\{ e\in\mathcal{E} \middle| \bigwedge_{b\in\mathcal{G}} \left(\mu^a(s^a,s^b) \wedge \bigwedge_{c\in\mathcal{G}} \left(\mu^b(s^b,s^c) \wedge e\in\mathcal{I}_m^c \right) \right) \right\} \\ &\overset{\text{ind.}}{=} \left\{ e \in \mathcal{M}^{\mathcal{G}} \middle| \bigwedge_{b,c\in\mathcal{G}} \mu^b(s^b, s^c) \right\}, \end{aligned}$$

which is the right side of (5.2), and where we used

$$\bigwedge_{b\in\mathcal{G}} \Big(\mu^a(s^a, s^b) \wedge \bigwedge_{c\in\mathcal{G}} \mu^b(s^b, s^c) \Big) = \bigwedge_{b,c\in\mathcal{G}} \mu^b(s^b, s^c) , \ \forall a \in \mathcal{G} .$$

Finally, applying (5.1) one more time to this result, yields:

$$\mathcal{I}_{m+3}^a \quad = \quad \bigcap_{b\in\mathcal{G}} \left\{ e \in \mathcal{I}_{m+2}^b \middle| \mu^a(s^a, s^b) \right\} \quad = \quad \mathcal{I}_{m+2}^a .$$

For all $m \geq 3$, $\mathcal{I}_m^a$ remains thus the right hand side of (5.2). As $\mathcal{I}^{\mathcal{G}} = \lim_{m\to\infty} \mathcal{I}_m^a$, we can thus conclude that, starting at the knowledge of any agent of group $\mathcal{G}$, in which all agents can see each other, the mutual knowledge is the common knowledge. □

The common knowledge can be computed using only the visible set $\mathcal{M}^a$ of every agent $a \in \mathcal{G}$. Moreover, actions that have been chosen by a policy, which itself is

common knowledge, and that further depends only on common knowledge and a shared random seed, are also common knowledge. The common knowledge of group $\mathcal{G}$ up to time t is thus some common prior knowledge ξ and the commonly known trajectory $\tau_t^{\mathcal{G}} = (\xi, z_1^{\mathcal{G}}, \mathbf{u}_1^{\mathcal{G}}, \ldots, z_t^{\mathcal{G}}, \mathbf{u}_t^{\mathcal{G}})$, with $z_k^{\mathcal{G}} = \{s_k^e \,|\, e \in \mathcal{I}^{\mathcal{G}}\}$. Knowing all binary masks μ^a makes it possible to derive $\tau_t^{\mathcal{G}} = \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi)$ from the observation trajectory $\tau_t^a = (z_1^a, \ldots, z_t^a)$ of any agent $a \in \mathcal{G}$ and the shared prior knowledge ξ. A function that conditions on $\tau^{\mathcal{G}}$ can therefore be computed independently by every member of $\mathcal{G}$.

Note that (by definition) common knowledge can only arise from entities that are observed *identically* by all agents. If only one agent receives non-deterministic observations, for example induced by sensor noise, the other agents cannot deduce the group's mutual (and thus common) knowledge. Our method therefore only guarantees perfect decentralisation of the learned policy in settings with deterministic observations, like simulations and computer games. However, in Section 5.5.1 we show empirically that, using a naive correlated sampling protocol similar to the theoretically optimal Holenstein protocol (Holenstein, 2007; Bavarian et al., 2020), MACKRL can still succeed in the presence of moderate sensor noise.

5.4 Multi-Agent Common Knowledge Reinforcement Learning

The key idea behind MACKRL is to learn decentralised policies that are nonetheless coordinated by common knowledge. As the common knowledge history $\tau_t^{\mathcal{G}}$ of a group of agents $\mathcal{G}$ can be deduced by every member, i.e., $\tau_t^{\mathcal{G}} = \mathcal{I}^{\mathcal{G}}(\tau_t^a, \xi), \forall a \in \mathcal{G}$, any deterministic function based only on $\tau_t^{\mathcal{G}}$ can thus be independently computed by every member as well. The same holds for pseudo-random functions like stochastic policies, if they condition on a commonly known random seed in ξ.

MACKRL uses a hierarchical policy $\pi(\mathbf{u}_{\text{env}} | \{\tau_t^a\}_{a \in \mathcal{A}}, \xi)$ over the joint environmental action space of all agents $\mathcal{U}_{\text{env}}$. The hierarchy forms a tree of sub-policies $\pi^{\mathcal{G}}$ over groups $\mathcal{G}$, where the root $\pi^{\mathcal{A}}$ covers all agents. Each sub-policy $\pi^{\mathcal{G}}(u^{\mathcal{G}} | \mathcal{I}^{\mathcal{G}}(\tau_t^{\mathcal{G}}, \xi))$ conditions on the common knowledge of $\mathcal{G}$, including a shared random seed in ξ, and can thus be executed by every member of $\mathcal{G}$ independently. The corresponding

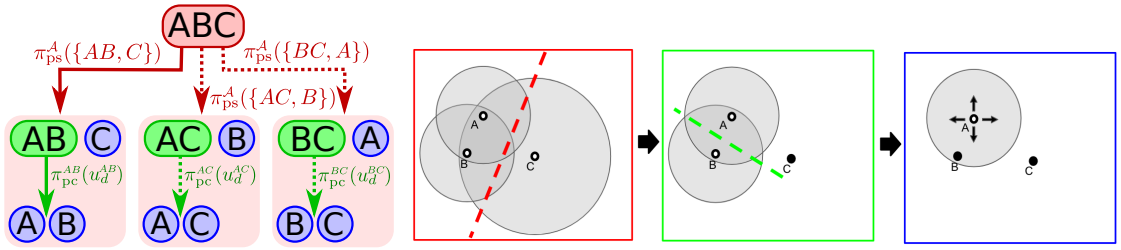


Figure 5.2: An illustration of Pairwise MACKRL. [Left]: the full hierarchy for 3 agents (dependencies on common knowledge are omitted for clarity). Only solid arrows are computed during decentralised sampling with Algorithm 4, while all arrows must be computed recursively during centralised training (see Algorithm 5). [Right]: the (maximally) 3 steps of decentralised sampling from the perspective of agent A. (i) Pair selector $\pi^{\mathcal{A}}_{\text{ps}}$ chooses the partition $\{AB, C\}$ based on the common knowledge of all agents $\mathcal{I}^{ABC}(\tau^A, \xi) = \varnothing$. (ii) Based on the common knowledge of pair A and B, $\mathcal{I}^{AB}(\tau^A, \xi)$, the pair controller π^{AB}_{pc} can either choose a joint action $(u^A_{\text{env}}, u^B_{\text{env}})$, or delegate to individual controllers by selecting u^{AB}_d. (iii) If delegating, the individual controller π^A must select the action u^A_{env} for the single agent A. All steps can be computed based on A's history τ^A.

action space $\mathcal{U}^{\mathcal{G}}$ contains the environmental actions of the group $\mathbf{u}^{\mathcal{G}}_{\text{env}} \in \mathcal{U}^{\mathcal{G}}_{\text{env}}$ *and/or* a set of group partitions, that is, $\mathbf{u}^{\mathcal{G}} = \{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$ with $\mathcal{G}_i \cap \mathcal{G}_j = \varnothing$, $\forall i \neq j$ and $\cup_{i=1}^{k} \mathcal{G}_i = \mathcal{G}$. Choosing a partition $\mathbf{u}^{\mathcal{G}} \neq \mathcal{U}^{\mathcal{G}}_{\text{env}}$ yields control to the sub-policies $\pi^{\mathcal{G}_i}$ of the partition's subgroups $\mathcal{G}_i \in \mathbf{u}^{\mathcal{G}}$. This can be an advantage in states where the common history $\tau^{\mathcal{G}_i}_t$ of the subgroups is more informative than $\tau^{\mathcal{G}}_t$. All action spaces have to be specified in advance, which induces the hierarchical tree structure of the joint policy. Algorithm 4 shows the decentralised sampling of environmental actions from the hierarchical joint policy as seen by an individual agent $a \in \mathcal{A}$.

As the common knowledge of a group with only one agent $\mathcal{G} = \{a\}$ is $\mathcal{I}^{\{a\}}(\tau^a, \xi) = \tau^a$, fully decentralised policies are a special case of MACKRL policies: in this case, the root policy $\pi^{\mathcal{A}}$ has only one action $\mathcal{U}^{\mathcal{A}} := \{\mathbf{u}^{\mathcal{A}}\}$, $\mathbf{u}^{\mathcal{A}} := \{\{1\}, \ldots, \{n\}\}$, and all leaf policies $\pi^{\{a\}}$ have only environmental actions $\mathcal{U}^{\{a\}} := \mathcal{U}^a_{\text{env}}$.

Algorithm 4 Decentralised action selection for agent $a \in \mathcal{A}$ in MACKRL

function SELECT_ACTION(a, τ^a_t, ξ) ▷ random seed in ξ is common knowledge
  $\mathcal{G} := \mathcal{A}$ ▷ initialise the group $\mathcal{G}$ of all agents
  $\mathbf{u}^{\mathcal{G}}_t \sim \pi^{\mathcal{G}}\Big(\cdot \,|\, \mathcal{I}^{\mathcal{G}}(\tau^a_t, \xi)\Big)$ ▷ $\mathbf{u}^{\mathcal{G}}_t$ is either a joint environmental action in $\mathcal{U}^{\mathcal{G}}_{\text{env}}$...
  while $\mathbf{u}^{\mathcal{G}}_t \notin \mathcal{U}^{\mathcal{G}}_{\text{env}}$ **do** ▷ ... or a set of disjoint subgroups $\{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$
    $\mathcal{G} := \{\mathcal{G}' \,|\, a \in \mathcal{G}', \mathcal{G}' \in \mathbf{u}^{\mathcal{G}}_t\}$ ▷ select subgroup containing agent a
    $\mathbf{u}^{\mathcal{G}}_t \sim \pi^{\mathcal{G}}\Big(\cdot \,|\, \mathcal{I}^{\mathcal{G}}(\tau^a_t, \xi)\Big)$ ▷ draw an action for that subgroup
  return u^a_t ▷ return environmental action $u^a_t \in \mathcal{U}^a_{\text{env}}$ of agent a

5.4.1 Pairwise MACKRL

To give an example of one possible MACKRL architecture, we define *Pairwise MACKRL*, illustrated in Figure 5.2. As joint action spaces grow exponentially in the number of agents, we restrict ourselves to *pairwise* joint policies and define a three-level hierarchy of controllers.

The root of this hierarchy is the *pair selector* $\pi^{\mathcal{A}}_{\text{ps}}$, with an action set $\mathcal{U}^{\mathcal{A}}_{\text{ps}}$ that contains all possible partitions of agents into pairs $\{\{a_1, \bar{a}_1\}, \ldots, \{a_{n/2}, \bar{a}_{n/2}\}\} =: \mathbf{u}^{\mathcal{A}} \in \mathcal{U}^{\mathcal{A}}_{\text{ps}}$, but no environmental actions. If there are an odd number of agents, then one agent is put in a singleton group. At the second level, each *pair controller* $\pi^{a\bar{a}}_{\text{pc}}$ of the pair $\mathcal{G} = \{a, \bar{a}\}$ can choose between joint actions $\mathbf{u}^{a\bar{a}}_{\text{env}} \in \mathcal{U}^{a}_{\text{env}} \times \mathcal{U}^{\bar{a}}_{\text{env}}$ and one delegation action $u^{a\bar{a}}_{d} := \{\{a\}, \{\bar{a}\}\}$, i.e., $\mathcal{U}^{a\bar{a}}_{\text{pc}} := \mathcal{U}^{a}_{\text{env}} \times \mathcal{U}^{\bar{a}}_{\text{env}} \cup \{u^{a\bar{a}}_{d}\}$. At the third level, individual controllers π^{a} select an individual action $u^{a}_{\text{env}} \in \mathcal{U}^{a}_{\text{env}}$ for a single agent a. This architecture retains manageable joint action spaces, while considering all possible pairwise coordination configurations. Fully decentralised policies are the special case when all pair controllers always choose partition $u^{a\bar{a}}_{d}$ to delegate.

Unfortunately, the number of possible pairwise partitions is $O(n!)$, which limits the algorithm to medium sized sets of agents. For example, $n = 11$ agents induce $|\mathcal{U}^{\mathcal{A}}_{\text{ps}}| = 10395$ unique partitions. To scale our approach to tasks with many agents, we share network parameters between all pair controllers with identical action spaces, thereby greatly improving sample efficiency. We also investigate a more scalable variant in which the action space of the pair selector $\pi^{\mathcal{A}}_{\text{ps}}$ is only a fixed random subset of all possible pairwise partitions. This restricts agent coordination to a smaller set of predefined pairs, but only modestly affects MACKRL's performance (see Section 5.5.4).

To give a better intuition about Pairwise MACKRL, we detail an example of three agents $\mathcal{A} := \{1, 2, 3\}$. The explicitly written-out joint policy of Pairwise MACKRL for these agents is:

$$
\begin{aligned}
\boldsymbol{\pi}_\theta(u^1_{\text{env}}, u^2_{\text{env}}, u^3_{\text{env}}) = {} & \pi^{\mathcal{A}}_{\text{ps},\theta}(u^{\mathcal{A}}_{\text{ps}}{=}\{\{1\},\{2,3\}\}|\mathcal{I}^{1,2,3}_s) \cdot \pi^1_\theta(u^1_{\text{env}}|\tau^1) \\
& \quad \cdot \Big(\pi^{2,3}_{\text{pc},\theta}(u^{2,3}_{\text{env}}|\mathcal{I}^{2,3}_s) + \pi^{2,3}_{\text{pc},\theta}(u^{2,3}_{d}|\mathcal{I}^{2,3}_s) \cdot \pi^2_\theta(u^2_{\text{env}}|\tau^2) \cdot \pi^3_\theta(u^3_{\text{env}}|\tau^3)\Big) \\
& + \pi^{\mathcal{A}}_{\text{ps},\theta}(u^{\mathcal{A}}_{\text{ps}}{=}\{\{2\},\{1,3\}\}|\mathcal{I}^{1,2,3}_s) \cdot \pi^2_\theta(u^2_{\text{env}}|\tau^2) \\
& \quad \cdot \Big(\pi^{1,3}_{\text{pc},\theta}(u^{1,3}_{\text{env}}|\mathcal{I}^{1,3}_s) + \pi^{1,3}_{\text{pc},\theta}(u^{1,3}_{d}|\mathcal{I}^{1,3}_s) \cdot \pi^1_\theta(u^1_{\text{env}}|\tau^1) \cdot \pi^2_\theta(u^3_{\text{env}}|\tau^3)\Big) \\
& + \pi^{\mathcal{A}}_{\text{ps},\theta}(u^{\mathcal{A}}_{\text{ps}}{=}\{\{3\},\{1,2\}\}|\mathcal{I}^{1,2,3}_s) \cdot \pi^3_\theta(u^3_{\text{env}}|\tau^3) \\
& \quad \cdot \Big(\pi^{1,2}_{\text{pc},\theta}(u^{1,2}_{\text{env}}|\mathcal{I}^{1,2}_s) + \pi^{1,2}_{\text{pc},\theta}(u^{1,2}_{d}|\mathcal{I}^{1,2}_s) \cdot \pi^1_\theta(u^1_{\text{env}}|\tau^1) \cdot \pi^2_\theta(u^2_{\text{env}}|\tau^2)\Big)\,.
\end{aligned}
$$

Conditional variables beyond common knowledge $\mathcal{I}^{\mathcal{G}}$ and action-observation histories τ^a have been omitted for brevity. See Table 5.1 for a detailed depiction of Pairwise MACKRL's hierarchical controllers.

[A] The following section is taken from the paper's appendix.

Level	Policy / Controller	$\#\pi$
1	$\pi_{\text{ps}}(u^{\text{ps}}\|\mathcal{I}^{\mathcal{A}}_{s_t}, u^{\text{ps}}_{t-1}, h^{\text{ps}}_{t-1})$	1
2	$\pi^{aa'}_{\text{pc}}(u^{aa'}\|\mathcal{I}^{aa'}_{s_t}, u^{aa'}_{t-1}, h^{aa'}_{t-1}, aa')$	3
3	$\pi^{a}(u^{a}\|z^{a}_{t}, h^{a}_{t-1}, u^{a}_{t-1}, a)$	3

Table 5.1: Hierarchy of pairwise MACKRL, where h is the hidden state of RNNs and z^a_t are observations. $\#\pi$ shows the number of controllers at this level for 3 agents.

Further details on Pairwise MACKRL

However the sampling of each agent's actions $u^a_{\text{env}} \in \mathcal{U}^a$ only needs to traverse one branch of the tree, as shown in Figure 5.3. At the top level, an agent id partition u_{ps} is categorically sampled from the pair selector policy $\pi_{\text{ps},\theta}$. At the second level, the pair selector policy for the pair contained in the partition $\pi^{a,b}_{\text{pc}}$ is categorically sampled from in order to receive $u^{a,b}$ where $a, b \in u_{\text{ps}}$. If the delegation action d is sampled, then both u^a and u^b are categorically resampled from their respective independent policies π^a and π^b. Otherwise, u^a and u^b are determined by $u^{a,b}$. The leftover agent $c \notin u_{\text{ps}}$ samples its action from its corresponding independent policy π^c. Note that this sampling scheme naturally generalised for $n > 3$.

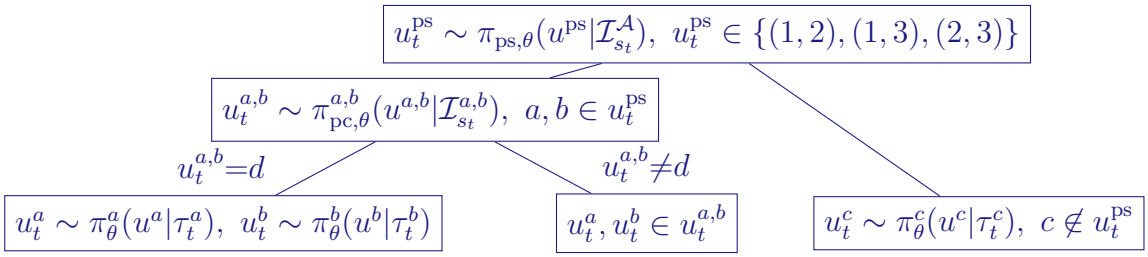


Figure 5.3: Action sampling for MACKRL for $n = 3$ agents.

5.4.2 Training

The training of policies in the MACKRL family is based on *Central-V* (Foerster et al., 2018), a stochastic policy gradient algorithm (Williams, 1992) with a centralised critic. Unlike the decentralised policy, we condition the centralised critic on the state $s_t \in \mathcal{S}$ and the last actions of all agents $\mathbf{u}_{\text{env},t-1} \in \mathcal{U}_{\text{env}}$. We do not use the multi-agent counterfactual baseline proposed by Foerster et al. (2018), because

Algorithm 5 Compute joint policies for a given $\mathbf{u}^{\mathcal{G}}_{\text{env}} \in \mathcal{U}^{\mathcal{G}}_{\text{env}}$ of a group of agents $\mathcal{G}$ in MACKRL

function JOINT_POLICY($\mathbf{u}^{\mathcal{G}}_{\text{env}} | \mathcal{G}, \{\tau^a_t\}_{a\in\mathcal{G}}, \xi$) ▷ random seed in ξ is common knowledge

$a' \sim \mathcal{G}$; $\mathbf{I}^{\mathcal{G}} := \mathcal{I}^{\mathcal{G}}(\tau^{a'}_t, \xi)$ ▷ common knowledge $\mathbf{I}^{\mathcal{G}}$ is identical for every agent $a' \in \mathcal{G}$

$p_{\text{env}} := 0$ ▷ initialise probability for choosing environmental joint action $\mathbf{u}^{\mathcal{G}}_{\text{env}}$

for $\mathbf{u}^{\mathcal{G}} \in \mathcal{U}^{\mathcal{G}}$ **do** ▷ add probability to choose $\mathbf{u}^{\mathcal{G}}_{\text{env}}$ for all outcomes of $\pi^{\mathcal{G}}$

if $\mathbf{u}^{\mathcal{G}} = \mathbf{u}^{\mathcal{G}}_{\text{env}}$ **then** ▷ if $\mathbf{u}^{\mathcal{G}}$ is the environmental joint action in question

$p_{\text{env}} := p_{\text{env}} + \pi^{\mathcal{G}}\Big(\mathbf{u}^{\mathcal{G}}_{\text{env}} | \mathbf{I}^{\mathcal{G}}\Big)$

if $\mathbf{u}^{\mathcal{G}} \notin \mathcal{U}^{\mathcal{G}}_{\text{env}}$ **then** ▷ if $\mathbf{u}^{\mathcal{G}} = \{\mathcal{G}^1, \ldots, \mathcal{G}^k\}$ is a set of disjoint subgroups

$p_{\text{env}} := p_{\text{env}} + \pi^{\mathcal{G}}\Big(\mathbf{u}^{\mathcal{G}} | \mathbf{I}^{\mathcal{G}}\Big) \prod_{\mathcal{G}'\in\mathbf{u}^{\mathcal{G}}} \text{JOINT\_POLICY}\Big(\mathbf{u}^{\mathcal{G}'}_{\text{env}} | \mathcal{G}', \{\tau^a_t\}_{a\in\mathcal{G}'}, \xi\Big)$

return p_{env} ▷ return probability that controller $\pi^{\mathcal{G}}$ would have chosen $\mathbf{u}^{\mathcal{G}}_{\text{env}}$

MACKRL effectively turns training into a single agent problem by inducing a correlated probability across the joint action space. Algorithm 5 shows how the probability of choosing a joint environmental action $\mathbf{u}^{\mathcal{G}}_{\text{env}} \in \mathcal{U}^{\mathcal{G}}_{\text{env}}$ of group $\mathcal{G}$ is computed: the probability of choosing the action in question is added to the recursive probabilities that each partition $\mathbf{u}^{\mathcal{G}} \notin \mathcal{U}^{\mathcal{G}}_{\text{env}}$ would have selected it. Algorithm 4 only traverses one branch of the tree during decentralised execution, during which we employ deterministic policies that do not require any additional means of coordination such as shared random seeds.

At time t, the gradient with respect to the parameters θ of the joint policy $\pi(\mathbf{u}_{\text{env}}|\{\tau^a_t\}_{a\in\mathcal{A}}, \xi)$ is:

$$\nabla_\theta J_t = \underbrace{\Big(r(s_t, \mathbf{u}_{\text{env},t}) + \gamma V(s_{t+1}, \mathbf{u}_{\text{env},t}) - V(s_t, \mathbf{u}_{\text{env},t-1})\Big)}_{\text{sample estimate of the advantage function}} \nabla_\theta \log\Big(\underbrace{\pi(\mathbf{u}_{\text{env},t}|\{\tau^a_t\}_{a\in\mathcal{A}}, \xi)}_{\text{JOINT\_POLICY}(\mathbf{u}_{\text{env},t}|\mathcal{A}, \{\tau^a_t\}_{a\in\mathcal{A}}, \xi)}\Big), \tag{5.3}$$

The value function V is learned by gradient descent on the TD(λ) loss (Sutton and Barto, 2018).

As the hierarchical MACKRL policy tree computed by Algorithm 5 is fully differentiable and MACKRL trains a joint policy in a centralised fashion, the standard convergence results for actor-critic algorithms (Konda and Tsitsiklis, 1999) with compatible critics (Sutton et al., 1999) apply.

Note that the convergence proof only holds if the critic explicitly conditions on the agent's histories. In the single-agent case, this fact has been pointed out by (Baisero and Amato, 2021).

According to (Lyu et al., 2021), removing the agent's history from the critic inputs

may nevertheless hit a better bias/variance sweet spot in practice.

5.5 Experiments and Results

We evaluate Pairwise MACKRL (henceforth referred to as MACKRL) on two environments: first, we use a matrix game with special coordination requirements to illustrate MACKRL's ability to surpass both IL and JAL. Secondly, we employ MACKRL with deep recurrent neural network policies in order to outperform state-of-the-art baselines on a number of challenging StarCraft II unit micromanagement tasks. Finally, we analyse MACKRL's robustness to sensor noise and its scalability to large numbers of agents to illustrate its applicability to real-world tasks. All source code is available at `https://github.com/schroederdewitt/mackrl`.

5.5.1 Single-step matrix game

To demonstrate how MACKRL trades off between independent and joint action selection, we evaluate a two-agent matrix game with partial observability. In each round, a fair coin toss decides which of the two matrix games in Figure 5.5 [left] is played. Both agents can observe which game has been selected if the observable *common knowledge bit* is set. If the bit is not set, each agent observes the correct game only with probability p_σ, and is given *no observation* otherwise. Crucially, whether agents can observe the current game is in this case determined independently of each other. Even if both agents can observe which game is played, this observation is no longer common knowledge and cannot be used to infer the choices of the other agent. To compare various methods we adjusted p_σ such that the independent probability of each agent to observe the current game is fixed at 75%.

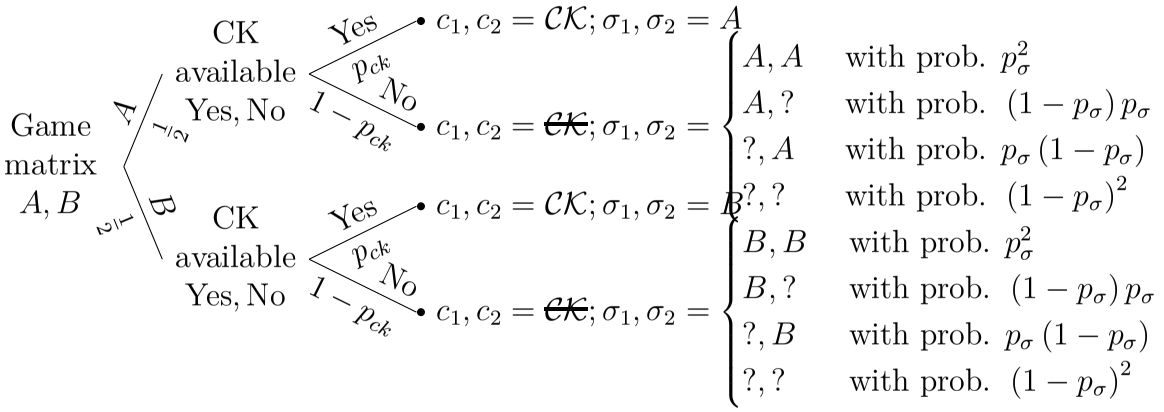


Figure 5.4: Probability tree for our simple single-step matrix game. The game chooses randomly between matrix A or B, and whether common knowledge is available or not. If common knowledge is available, both agents can condition their actions on the game matrix chosen. Otherwise, both agents independently only have a random chance of observing the game matrix choice. Here, p_{ck} is the probability that common knowledge exists and p_σ is the probability that an agent independently observes the game matrix choice. The observations of each agent 1 and 2 are given by tuples (c_1, σ_1) and (c_2, σ_2), respectively, where $c_1, c_2 \in \{\mathcal{CK}, \cancel{\mathcal{CK}}\}$ and $\sigma_1, \sigma_2 \in \{A, B, ?\}$.

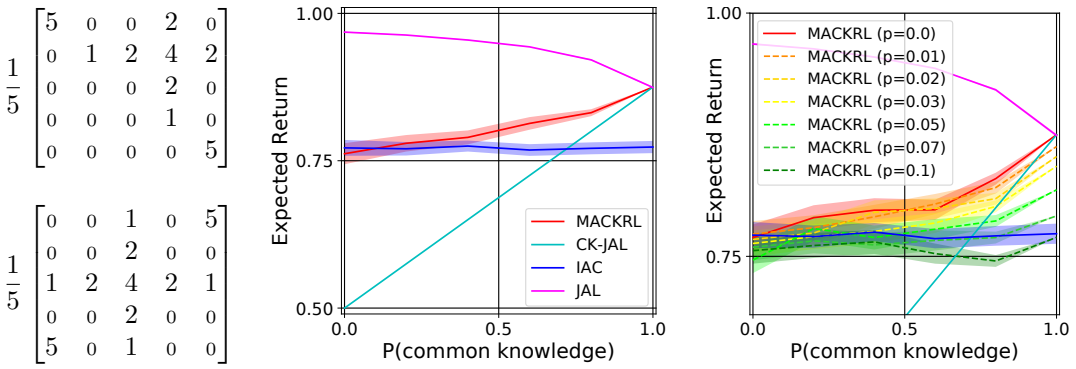


Figure 5.5: Game matrices A (top) and B (bottom) [left]. MACKRL almost always outperforms both IL and CK-JAL and is upper-bounded by JAL [middle]. When the common knowledge is noised by randomly flipping the CK-bit with probability p, MACKRL degrades gracefully [right].

Plots are displaying means, with shaded areas denoting standard error of the mean over 10 random seeds.

In order to illustrate MACKRL's performance, we compare it to three other methods: Independent Actor Critic (IAC) (Foerster et al., 2018) is a variant of Independent Learning where each agent conditions both its decentralized actor and critic only on its own observation. Joint Action Learning (JAL) learns a centralized joint policy that conditions on the union of both agent's observations. CK-JAL is a variant of JAL in which each agent learns its own joint policy conditioning only on the available common knowledge.

Note that JAL is implemented as a single-agent centralised policy gradient with $TD(\lambda)$ loss.

Figure 5.5 [middle] plots MACKRL's performance relative to IL, CK-JAL, and JAL against the fraction of observed games that is caused by a set CK-bit. As expected, the performance of CK-JAL linearly increases as more common knowledge becomes available, whereas the performance of IAC remains invariant. MACKRL's performance matches the one of IAC if no common knowledge is available and matches those of JAL and CK-JAL in the limit of all observed games stemming from common knowledge. In the regime between these extremes, MACKRL outperforms both IAC and CK-JAL, but is itself upper-bounded by JAL, which gains the advantage due to central execution.

To assess MACKRL's performance in the case of *probabilistic common knowledge* (see Section 5.2), we also consider the case where the observed *common knowledge bit* of individual agents is randomly flipped with probability p. This implies that both agents do not share true common knowledge with respect to the game matrix played. Instead, each agent a can only form a belief $\tilde{\mathcal{I}}_a^{\mathcal{G}}$ over what is commonly

known. The commonly known pair controller policy can then be conditioned on each agent's belief, resulting in agent-specific pair controller policies $\tilde{\pi}^{aa'}_{pc,a}$.

As $\tilde{\pi}^{aa'}_{pc,a}$ and $\tilde{\pi}^{aa'}_{pc,a'}$ are no longer guaranteed to be consistent, agents need to sample from their respective pair controller policies in a way that minimizes the probability that their outcomes disagree in order to maximise their ability to coordinate. Using their access to a shared source of randomness ξ, the agents can optimally solve this *correlated sampling* problem using Holenstein's strategy.

[A] The following section is partially taken from the paper's appendix.

5.5.2 Holenstein's Strategy

Given two agent-specific pair controller policies $\tilde{\pi}^{aa'}_{pc,a}$, both agents can optimally minimise disagreement when sampling independently form their respective policies by following Holenstein's strategy (Holenstein, 2007; Bavarian et al., 2020): With a suitably chosen $\gamma > 0$, each agent a is assigned a set

$$\mathcal{H}_a = \{(u,p) \in \mathcal{U}^{aa'}_{pc} \times \Gamma : p < \tilde{\pi}^{aa'}_{pc,a}(u)\},\ \Gamma = \{0, \gamma, 2\gamma, \dots, 1\} \tag{5.4}$$

Let ζ be a shared ξ-seeded random permutation of the elements in $\mathcal{U}^{aa'}_{pc} \times \Gamma$, then agent a samples $\zeta(i_a)$, where i_a is the smallest index such that $\zeta(i_a) \in \mathcal{H}_a$ (and agent a' proceeds analogously). The whole process is illustrated for two pair controllers with each two actions in Figure 5.6.

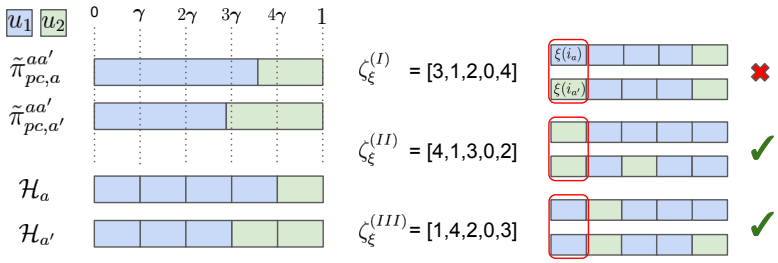


Figure 5.6: A graphical depiction of Holenstein sampling. We discretise the two slightly different pair controller policies into γ^{-1} chunks (left). Then, for each joint sampling, we shuffle the pair controller policies according to a shared random seed. Both agents agree on the same action if the resultant first partitions agree (mid and bottom), otherwise agents commit a coordination error (top).

Given the total variational distance δ between the categorical probability distributions defined by $\tilde{\pi}^{aa'}_{pc,a}$ and $\tilde{\pi}^{aa'}_{pc,a'}$, the disagreement probability of agents a, a' is then guaranteed to be at most $2\delta/(1+\delta)$ (Bavarian et al., 2020).

5.5.3 Approximation to Holenstein's strategy

However, Holenstein's strategy requires to evaluate a significantly larger set of actions and quickly becomes computationally expensive. We therefore instead make use of a simple correlated sampling heuristic: given a shared uniformly drawn variable $0 \leq \delta \sim \xi \leq 1$, agent a samples an action u_a such that

$$\sum_{u=1}^{u_a-1} \tilde{\pi}^{aa'}_{pc,a}(u) \leq \delta \leq \sum_{u=1}^{u_a} \tilde{\pi}^{aa'}_{pc,a}(u) \tag{5.5}$$

(agent a' proceeds analogously). While suboptimal, we find that this heuristic nevertheless performs satisfactorily in practice and can be trivially extended to groups of more than two agents.

Figure 5.5 [right] shows that MACKRL's performance declines remarkably gracefully with increasing observation noise. Note that as real-world sensor observations tend to tightly correlate with the true observations, noise levels of $p \geq 0.1$ in the context of the single-step matrix game are in comparison rather extreme as they indicate a completely different game matrix. This illustrates MACKRL's applicability to real-world tasks with noisy observation sensors.

5.5.4 StarCraft II micromanagement

To demonstrate MACKRL's ability to solve complex coordination tasks, we evaluate it on a challenging multi-agent version of StarCraft II (SCII) micromanagement. To this end, we report performance on three challenging coordination tasks from the established multi-agent benchmark SMAC (Samvelyan et al., 2019).

The first task, map *2s3z*, contains mixed unit types, where both the MACKRL agent and the game engine each control two Stalkers and three Zealots. Stalkers are ranged-attack units that take heavy damage from melee-type Zealots. Consequently, a winning strategy needs to be able to dynamically coordinate between letting one's own Zealots attack enemy Stalkers, and when to backtrack in order to defend one's own Stalkers against enemy Zealots. The challenge of this coordination task results in a particularly poor performance of Independent Learning (Samvelyan et al., 2019).

The second task, map *3m*, presents both sides with three Marines, which are medium-ranged infantry units. The coordination challenge on this map is to reduce enemy fire power as quickly as possible by focusing unit fire to defeat each enemy unit in turn. The third task, map *8m*, scales this task up to eight Marines on both

sides. The relatively large number of agents involved poses additional scalability challenges.

On all maps, the units are subject to partial observability constraints and have a circular field of view with fixed radius. Common knowledge $\mathcal{I}^{\mathcal{G}}$ between groups $\mathcal{G}$ of agents arises through entity-based field-of-view common knowledge.

We compare MACKRL to Central-V (Foerster et al., 2018), as well as COMA (Foerster et al., 2018) and QMIX (Rashid et al., 2020b), where the latter is an off-policy value-based algorithm that is the current state-of-the-art on all maps. We omit IL results since it is known to do comparatively poorly (Samvelyan et al., 2019). All experiments use SMAC settings for comparability (see Samvelyan et al. (2019)). In addition, MACKRL and its within-class baseline Central-V share equal hyper-parameters as far as applicable.

[A] The following section is taken from the paper's appendix.

5.5.5 Experimental setup - StarCraft II

All policies are implemented as two-layer recurrent neural networks (GRUs) with 64 hidden units, while the critic is feed forward and uses full state information. Parameters are shared across controllers within each of the second and third levels of the hierarchy. We also feed into the policy the agent index or index pairs. For exploration, we use a bounded softmax distribution in which the agent samples from a softmax over the policy logits with probability $(1 - \epsilon)$ and samples randomly with probability ϵ. We anneal ϵ from 0.5 to 0.01 across the first 50k environment steps.

Episodes are collected using eight parallel StarCraftII environments. Optimisation is carried out on a single GPU with Adam and a learning rate of 0.0005 for both the agents and the critic. The policies are fully unrolled and updated in a large mini-batch of $T \times B$ entries, where $T = 60$ and $B = 8$. By contrast, the critic is optimised in small mini-batches of size 8, one for each time-step, looping backwards in time. We found that this stabilised and accelerated training compared to full batch updates for the critic. The target network for the critic is updated after every 200 critic updates. We use $\lambda = 0.8$ in TD(λ) to accelerate reward propagation.

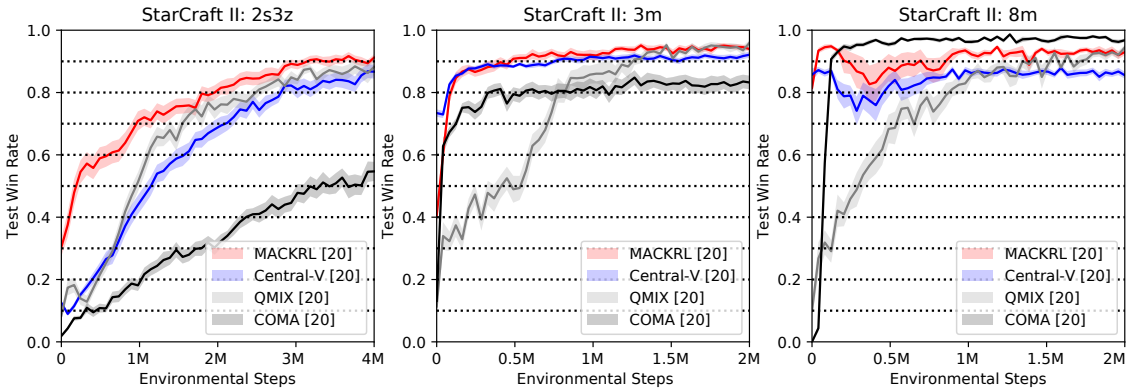


Figure 5.7: Win rate at test time across StarCraft II scenarios: 2 Stalkers & 3 Zealots [left], 3 Marines [middle] and 8 Marines [right]. Plots show means and their standard errors with [number of runs].

Square brackets denote the number of random seeds averaged over.

MACKRL outperforms the Central-V baseline in terms of sample efficiency and limit performance on all maps (see Figure 5.7). All other parameters being equal, this suggests that MACKRL's superiority over Central-V is due to its ability to exploit common knowledge. Below we confirm this conclusion by showing that the policies learnt by the pair controllers are almost always preferred over individual controllers whenever agents have access to substantial amounts of common knowledge.

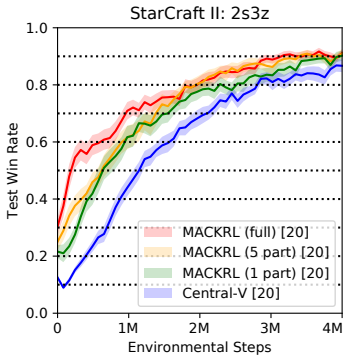


Figure 5.8: Illustrating MACKRL's scalability properties using partition subsamples of different sizes. Plot displays means over 20 seeds, with shaded areas indicating standard errors of the mean.

MACKRL also significantly outperforms COMA and QMIX on all maps in terms of sample efficiency, with a similar limit performance to QMIX (see Figure 5.7). These results are particularly noteworthy as MACKRL employs neither a sophisticated

multi-agent baseline, like COMA, nor an off-policy replay buffer, like QMIX.

As mentioned in Section 5.4.1, the number of possible agent partitions available to the pair selector $\pi_{\text{ps}}^{\mathcal{A}}$ grows as $\mathcal{O}(n!)$. We evaluate a scalable variant of MACKRL that constrains the number of partitions to a fixed subset, which is drawn randomly before training. Figure 5.8 shows that sample efficiency declines gracefully with subsample size. MACKRL's policies appear able to exploit any common knowledge configurations available, even if the set of allowed partitions is not exhaustive.

Pair controller introspection

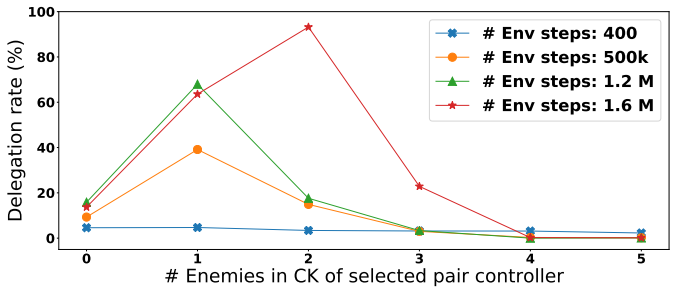


Figure 5.9: Delegation rate vs. number of enemies (2s3z) in the common knowledge of the pair controller over training. The numbers displayed stem from a single run.

We now test the hypothesis that MACKRL's superior performance is indeed due to its ability to learn how to use common knowledge for coordination. To demonstrate that the pair controller can indeed learn to delegate strategically, we plot in Figure 5.9 the percentage of delegation actions u_d against the number of enemies in the common knowledge of the selected pair controller, in situations where there is at least some common knowledge.

Since we start with randomly initialised policies, at the beginning of training the pair controller delegates only rarely to the decentralised controllers. As training proceeds, it learns to delegate in most situations where the number of enemies in the common knowledge of the pair is small, the exception being no visible enemies, which happens too rarely (5% of cases). This shows that MACKRL can learn to delegate in order to take advantage of the private observations of the agents, but

also learns to coordinate in the joint action space when there is substantial common knowledge.

5.6 Related Work

Multi-agent reinforcement learning (MARL) has been studied extensively in small environments (Busoniu et al., 2008; Yang and Gu, 2004), but scaling it to large state spaces or many agents has proved problematic. Guestrin et al. (2002a) propose the use of *coordination graphs*, which exploit conditional independence properties between agents that are captured in an undirected graphical model, in order to efficiently select joint actions. *Sparse cooperative Q-learning* (Kok and Vlassis, 2004) also uses coordination graphs to efficiently maximise over joint actions in the Q-learning update rule. Whilst these approaches allow agents to coordinate optimally, they require the coordination graph to be known and for the agents to either observe the global state or to be able to freely communicate. In addition, in the worst case there is no conditional independence to exploit and maximisation must still be performed over an intractably large joint action space.

There has been much work on scaling MARL to handle complex, high dimensional state and action spaces. In the setting of fully centralised training and execution, Usunier et al. (2016) frame the problem as a *greedy MDP* and train a centralised controller to select actions for each agent in a sequential fashion. Sukhbaatar et al. (2016) and Peng et al. (2017) train factorised but centralised controllers that use special network architectures to share information between agents. These approaches assume unlimited bandwidth for communication.

One way to decentralise the agents' policies is to learn a separate Q-function for each agent as in *Independent Q-Learning* (Tan, 1993). Foerster et al. (2017) and Omidshafiei et al. (2017) examine the problem of instability that arises from the nonstationarity of the environment induced by both the agents' exploration and their changing policies. Rashid et al. (2020b) and Sunehag et al. (2018) propose learning a centralised value function that factors into per-agent components. Gupta et al. (2017) learn separate policies for each agent in an actor-critic framework, where the critic for each agent conditions only on per-agent information. Foerster et al. (2018) and Lowe et al. (2017) propose a single centralised critic with decentralised actors. None of these approaches explicitly learns a policy over joint actions and hence are limited in the coordination they can achieve.

Thomas et al. (2014) explore the psychology of common knowledge and coordination. Rubinstein (1989) shows that any finite number of reasoning steps, short of the infinite number required for common knowledge, can be insufficient for achieving coordination. Korkmaz et al. (2014) examine common knowledge in scenarios where agents use Facebook-like communication. Brafman and Tennenholtz (2003) use a common knowledge protocol to improve coordination in common interest stochastic

games but, in contrast to our approach, establish common knowledge about agents' action sets and not about subsets of their observation spaces.

Aumann et al. (1974) introduce the concept of a *correlated equilibrium*, whereby a shared *correlation device* helps agents coordinate better. Cigler and Faltings (2013) examine how the agents can reach such an equilibrium when given access to a simple shared *correlation vector* and a communication channel. Boutilier (1999) augments the state space with a coordination mechanism, to ensure coordination between agents is possible in a fully observable multi-agent setting. This is in general not possible in the partially observable setting we consider.

Amato et al. (2014) propose MacDec-POMDPs, which use hierarchically optimal policies that allow agents to undertake temporally extended macro-actions. Liu et al. (2017) investigate how to learn such models in environments where the transition dynamics are not known. Makar et al. (2001) extend the MAXQ single-agent hierarchical framework (Dietterich, 2000) to the multi-agent domain. They allow certain policies in the hierarchy to be *cooperative*, which entails learning the joint action-value function and allows for faster coordination across agents. Kumar et al. (2017) use a hierarchical controller that produces subtasks for each agent and chooses which pairs of agents should communicate in order to select their actions. Oh and Smith (2008) employ a hierarchical learning algorithm for cooperative control tasks where the outer layer decides whether an agent should coordinate or act independently, and the inner layer then chooses the agent's action accordingly. In contrast with our approach, these methods require communication during execution and some of them do not test on sequential tasks.

Nayyar et al. (2013) show that common knowledge can be used to reformulate decentralised planning problems as POMDPs to be solved by a central coordinator using dynamic programming. However, they do not propose a method for scaling this to high dimensions. By contrast, MACKRL is entirely model-free and learns trivially decentralisable control policies end-to-end.

Guestrin et al. (2002c) represent agents' value functions as a sum of context-specific value rules that are part of the agents' fixed a priori common knowledge. By contrast, MACKRL learns such value rules dynamically during training and does not require explicit communication during execution.

Despite using a hierarchical policy structure, MACKRL is not directly related to the family of hierarchical reinforcement learning algorithms (Vezhnevets et al., 2017), as it does not involve temporal abstraction.

5.7 Conclusion and Future Work

This paper proposed a way to use common knowledge to improve the ability of decentralised policies to coordinate. To this end, we introduced MACKRL, an algorithm which allows a team of agents to learn a fully decentralised policy that nonetheless can select actions jointly by using the common knowledge available.

MACKRL uses a hierarchy of controllers that can either select joint actions for a pair or delegate to independent controllers.

In evaluation on a matrix game and a challenging multi-agent version of StarCraft II micromanagement, MACKRL outperforms strong baselines and even exceeds the state of the art by exploiting common knowledge. We present approximate versions of MACKRL that can scale to greater numbers of agents and demonstrate robustness to observation noise.

In future work, we would like to further increase MACKRL's scalability and robustness to sensor noise, explore off-policy variants of MACKRL and investigate how to exploit limited bandwidth communication in the presence of common knowledge. We are also interested in utilising SIM2Real transfer methods (Tobin et al., 2017; Tremblay et al., 2018) in order to apply MACKRL to autonomous car and unmanned aerial vehicle coordination problems in the real world.

This concludes our quote.

5.8 Extensions

MACKRL can be extended in a variety of ways. First of all, our paper includes, next to SMAC results, the study of a simple two-player matrix game. Despite its simplicity, our matrix game has proven very useful in studying the unique ground between joint and independent learning that MACKRL occupies. *Extending this matrix game to more than two players* would allow similar analysis of how to best learn policy delegation across the group hierarchy of common knowledge. This would also allow to study the performance of Holenstein sampling in comparison to the simple correlated bucket sampling strategy employed by us.

Secondly, it would be interesting to see MACKRL's performance under *noisy observations in a more complex environment*, such as SMAC. We only include experiments on our matrix toy environment, which does constitute a general proof of principle, but does not allow to calibrate the tolerable noise levels to practical applications.

Thirdly, to scale MACKRL to larger numbers of agents, our algorithm randomly subsamples group partitions. An alternative way of achieving scalability is to instead *prune the backward graph* of the joint agent policy. A possible way of doing this would be to start randomly pruning subtrees during training, and then, over time, transition to pruning subtrees based on probability thresholds.

Fourthly, it would be interesting to see how MACKRL could be extended to *continuous action spaces*. One possible avenue would be to parametrise all group-wise controllers and independent policies by Gaussians, which would render the multi-level joint policy into a mixture of Gaussians.

There also remains an interesting theoretical problem: The optimality proof (Bavarian et al., 2020) for optimal correlated sampling (Holenstein sampling) is only based on two players. *Optimality guarantees for more than two players* seem to be a hitherto open problem.

Finally, MACKRL may be relevant to settings that do allow for some *limited bandwidth communication between agents*: The available bandwidth may be used in order to replace the upper levels of the joint policy by communication protocols

that negotiate which groups of agents should coordinate with each other under which circumstances. This would allow group partitioning to be optimised even if higher-level group knowledge is not available. Unlike other approaches, MACKRL would not require agents to exchange their actual high-dimensional observations through their limited-bandwidth communication channels.

Coming up next...

In this chapter, we have developed MACKRL, a novel DMARL method that allows agents to leverage group-wise common knowledge in order to train joint policies that are nevertheless fully decentralised, allowing agents to improve their coordinative abilities. In the next chapter, we study settings where a receiver agent can observe the full trajectory of a sender agent, and this fact is commonly known between both agents. This allows the sender agent to transmit messages to the receiver agent using the spare capacity of the sender's policy through a learnt communication protocol.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Multi-agent Common Knowledge Reinforcement Learning
Publication Status	x Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	***C. Schroeder de Witt****, J. Foerster*, G. Farquhar, P. Torr, W. Böhmer, and S. Whiteson. Multi-agent Common Knowledge Reinforcement Learning (2019).* Advances in Neural Information Processing Systems, volume 32, pages 9927–9939 (*: equal contribution)

Student Confirmation

Student Name:	Christian Schroeder de Witt		
Contribution to the Paper	• Generation of Ideas with co-authors • Software Development • Designed and performed experiments • Wrote the paper with co-authors		
Signature		Date	30.04.2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Professor Philip H.S. Torr			
Supervisor comments			
Signature		Date	04.05.2021

This completed form should be included in the thesis, at the end of the relevant chapter.

Part III

Implicit Communication

6

Communicating via Markov Decision Processes

Contents

In this Chapter, we study settings where a receiver agent can observe the full trajectory of a sender agent, and this fact is commonly known between both agents. This allows the sender agent to transmit messages to the receiver agent using the spare capacity of the sender's policy through a learnt communication protocol. We introduce *GME*, a novel control algorithm that separates learning the sender's task from encoding information in its actions. Compared to end-to-end approaches,

GME scales to much larger message spaces.

We now quote from (Sokota et al., 2021), with the quoted text visually indicated by reduced linespacing. Please note that the quote is literal except for where original supplementary material and appendices have been inserted in-place and internal document references have been updated accordingly.

This work introduces a novel problem setting called Markov coding games (MCGs). MCGs are two-player decentralized Markov decision processes (Oliehoek and Amato, 2016) that proceed in four steps. In the first step, one agent (called the sender) receives a special private observation (called the message), which it is tasked with communicating. In the second step, the sender plays out an episode of a Markov decision process (MDP). In the third, the other agent (called the receiver) receives the sender's MDP trajectory as its observation. In the fourth, the receiver decodes the received trajectory and guesses the message. The shared payoff to the sender and receiver is a weighted sum of the cumulative reward yielded by the MDP and an indicator specifying whether or not the receiver correctly guessed the message.

Among the reasons that MCGs are of interest is the fact that they generalize other important settings. The first of these is referential games. In a referential game, a sender attempts to communicate a message to a receiver using cheap talk actions—i.e., communicatory actions that do not have externalities on the transition or reward functions. Referential games have been a subject of academic interest dating back at least as far as Lewis's seminal work *Convention* (Lewis, 1969). Since then, various flavors of referential games have been studied in game theory (Skyrms, 2010), artificial life (Steels, 2013), evolutionary linguistics (Smith, 2002), cognitive science (Spike et al., 2017), and machine learning (Lazaridou et al., 2018). MCGs can be viewed as a generalization of referential games to a setting where we drop the often unrealistic assumption that the sender's actions do not incur costs.

A second problem setting generalized by MCGs is source coding (MacKay, 2002). In source coding (also known as data compression) the objective is to construct an injective mapping from a space of messages to the set of sequences of symbols (for some finite set of symbols) such that the expected output length is minimized. Source coding has a myriad of real world applications involving the compression of images, video, audio, and genetic data. MCGs can be viewed as a generalization of the source coding problem to a setting where the cost of an encoding may involve complex considerations, rather than simply being equal to the sequence length.

Yet another reason to be interested in MCGs is that they isolate an important subproblem of decentralized control. In particular, achieving good performance in an MCG requires the sender's actions to simultaneously perform control in an MDP and communicate information (i.e., to communicate implicitly). This presents a challenge due to the fact that approximate dynamic programming, the foundation for preeminent approaches to constructing control policies (Sutton and Barto, 2018), is ill suited to constructing communication protocols because their values depend

on counterfactuals. In other words, the information conveyed by an action depends on the policy at other contemporaneous states, violating the locality assumption of approximate dynamic programming approaches.

To address MCGs, we propose a theoretically grounded algorithm called greedy minimum entropy coupling (GME). GME leverages a union of maximum entropy reinforcement learning (MaxEnt RL) (Ziebart et al., 2008) and minimum entropy coupling (MEC) (Kovačević et al., 2015). The key insight is that maximizing the returns of the MDP can be disentangled from learning a good communication protocol by realizing that the entropy of a policy corresponds (in an informal sense) to its capacity to communicate. GME leverages this insight in two steps. In the first step, GME constructs a MaxEnt policy for the MDP, balancing between maximizing expected return and maximizing cumulative conditional entropy. In the second step, which occurs at each decision point, GME uses MEC to pair messages with actions in such a way that the sender selects actions with the same probabilities as the MaxEnt RL policy (thereby guaranteeing the same expected return from the MDP) and the receiver's uncertainty about the message is greedily reduced as much as possible.

To demonstrate the efficacy of GME, we present experiments for MCGs based on a gridworld, Cartpole, and Pong (Bellemare et al., 2013), which we call CodeGrid, CodeCart, and CodePong, respectively. For CodeGrid, we show that with a message space in the 10s or 100s, GME significantly is able to outperform a relevant baseline. For CodeCart and CodePong, we use a message space of binary images and a uniform distribution over messages, meaning that a randomly guessing receiver has an astronomically small probability of guessing correctly. Remarkably, we show that GME is able to achieve an optimal expected return in Cartpole and Pong while simultaneously losslessly communicating images to the receiver, demonstrating that GME has the capacity to be scaled to extremely large message spaces and complex control tasks. Moreover, we find that the performance of GME decays gracefully as the amount of actuator noise in the environment increases.

6.1 Related Work

The works that are most closely related to this one can be taxonomized as coming from literature on referential games, source coding, multi-agent reinforcement learning, and diverse skill learning.

Referential Games Among work on referential games, Foerster et al. (2016)'s work is perhaps most similar in that it is concerned with directly optimizing the performance of a communication protocol. They propose DIAL, an algorithm that optimizes the sender's protocol by performing gradient ascent through the parameters of the receiver. Foerster et al. show that DIAL outperforms methods based on independent Q-learning on a variety of communication tasks. However, DIAL-based approaches are not directly applicable to MCGs, as they would require differentiating through trajectories.

Channel Coding Another body of related work concerns extensions of the source coding problem. Length limited coding (Larmore and Hirschberg, 1990) considers a problem setting in which the objective is to minimize the expected sequence length (as before), subject to a maximum length constraint. Coding with unequal symbol costs (Golin et al., 2002; Iwata et al., 1997) considers the problem in which the goal is to minimize the expected cumulative symbol cost of the sequence to which the message is mapped. The cost of a symbol may differ from the cost of other symbols arbitrarily, making it a strictly more general problem setting than standard source coding (which can also be thought of as minimizing cumulative symbol cost with equally costly symbols). Both length limited coding and coding with unequal costs are subsumed by Markov coding games. And while existing algorithms for both standard source coding and the extensions above are well-established and widely commercialized, they are unable to address the more general MCG setting.

MCGs are also related to finite state Markov channel settings (Wang and Moayeri, 1995). In such settings, the fidelity of the channel by which the sender communicates to a receiver is controlled by a Markov process, which, in contrast to our work, transitions *independently* of the sender's decisions. Another related setting is intersymbol interference, where the sender's previously selected symbols (i.e., actions) may cause interference with subsequently selected symbols, making them less likely to be faithfully transmitted to the receiver (Lathi, 1998). MCGs differ from both Markov channel and intersymbol interference settings in that the Markov system controls the cost paid by the sender, rather than interfering with the quality of the channel. MCGs are more resemblant of a setting in which the channel is reliable, but subject to natural variation in costs, such as based on weather or third party usage, as well as variation based on the sender's own usage.

Multi-Agent Reinforcement Learning A third related area comes from MARL literature. Strouse et al. (2018) investigate directly embedding a reward for taking actions with high mutual information into policy gradient objectives. They find that this approach can improve expected return in cooperative settings with asymmetric information. The baseline for our CodeGrid experiments loosely resembles Strouse et al.'s algorithm. More recently, Bhatt and Buro (2021) investigate an alternative approach whereby the sender's behavioral policy deterministically selects the action that maximizes the receiver's posterior probability of the correct message, when computed using the target policy. They show that this modification empirically yields significantly improved convergence properties as compared to other variations of independent reinforcement learning. However, this approach is not directly applicable to settings in which a single action must be used for both communication and control.

Diverse Skill Learning A fourth area of related research is that of diverse skill learning (Eysenbach et al., 2018). Eysenbach et al. (2018) propose an unsupervised learning method for discovering diverse, identifiable skills. Their objective, called DIAYN, seeks to learn diverse, discriminable skills. This paradigm resembles our work in the sense that skills can be interpreted as messages and discriminability can be interpreted as maximizing the mutual information between the skill and the state. The baseline used in our CodeGrid experiments can also be viewed as an

adaptation of an idealized version of DIAYN to the MCG setting.

We now include some additional, perhaps less closely related work that nevertheless merits being compared and/or contrasted to.

MCGs themselves are a special subclass of *Dec-POMDPs* (Oliehoek and Amato, 2016) (or, more precisely, *Dec-POMDP-Coms* (Goldman and Zilberstein, 2003)). Spaan et al. (2006) introduce a slight variation of Dec-POMDP-Coms in which communication actions are not broadcast, but instead form part of other agents' local observations. Unlike in MCGs, the corresponding observations are factored out of the domain-level observation space.

(Winstein and Balakrishnan, 2013) develop an end-to-end congestion control mechanism for a multi-user network. In such settings, network endpoints need to decide when to transmit a given packet of data under conditions that make it difficult to estimate whether the required capacity is available. While this setting features decentralised communication under a complex, non-stationary cost function and, if only considering one sending and one receiving endpoint under full-observability, might be reduced to a MCG, it does not seem entirely clear why a sender's decision to send or not send a packet at a given time should be used to transmit additional information to the receiver.

We note that MCGs can be interpreted as a form of *Dec-POMDP with delayed communication* (Aicardi et al., 1987; Ooi and Wornell, 1996),(Oliehoek et al., 2008a, Appendix A.2.1), as the sender may need to take many actions before the receiver can decode the full message.

MCGs can be seen as a certain two-player instantiation of a ρDec-POMDP (Lauri et al., 2017, 2019) where the sender's reward depends on both the sender's own state and action, as well as the receiver's *belief entropy*. However, compared with active sensing applications commonly associated with ρ-POMDPs (Araya et al., 2010), i.e. (Hero et al., 2008; Thrun, 2000), or ρDec-POMDPs(Lauri et al., 2017), MCGs pose a distinctly more artificial setup.

Lauri and Oliehoek (2020) introduce multi-agent active perception tasks, where a team of agents cooperatively gathers observations to compute a joint estimate of a hidden variable, the accuracy of which is quantified by a centralised observer in the very end. This setting does not seem to be reducible to a MCG, as in a MCG, the receiver agent is not able to influence its own observations.

6.2 Background and Notation

We will require the following background and notation material to introduce Markov coding games and greedy minimum entropy coupling.

Markov Decision Processes To represent our task formalism, we use finite Markov decision processes (MDPs). We notate MDPs using tuples $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T} \rangle$ where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{R}\colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\mathcal{T}\colon \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function. An agent's interactions with an MDP are dictated by a policy $\pi\colon \mathcal{S} \to \Delta(\mathcal{A})$ mapping states to distributions over actions. We focus on episodic MDPs, meaning that after a finite number of transitions have occurred, the MDP will terminate. The history of states and actions is notated using $h = (s^0, a^0, \dots, s^t)$. We use the notation $\mathcal{R}(h) = \sum_j \mathcal{R}(s^j, a^j)$ to denote the amount of reward accumulated over the course of a history. When a history is terminal, we use z to notate it, rather than h. The objective of an MDP is to determine a policy $\arg\max_\pi \mathbb{E}_\pi \mathcal{R}(Z)$ yielding a large cumulative reward in expectation.

Entropy To help us quantify the idea of uncertainty, we introduce entropy. Symbolically, the entropy of a random variable X is $\mathcal{H}(X) = -\mathbb{E} \log \mathcal{P}(X)$. Because the

logarithm function is concave, the entropy of X is maximized when the mass of $\mathcal{P}_X$ is spread as evenly as possible and minimized when the mass of $\mathcal{P}_X$ is concentrated at a single point.

In the context of decision-making, entropy can be used to describe the uncertainty regarding which action will be taken by an agent. When a policy spans multiple decision-points, the uncertainty regarding the agent's actions given that the state is known is naturally described by conditional entropy. Conditional entropy is the entropy of a random variable, conditioned upon the fact that the realization of another random variable is known. More formally, conditional entropy is defined by $\mathcal{H}(X \mid Y) = \mathcal{H}(X, Y) - \mathcal{H}(Y)$ where the joint entropy $\mathcal{H}(X, Y) = -\mathbb{E} \log \mathcal{P}(X, Y)$ is defined as the entropy of (X, Y) considered as a random vector.

In some contexts, it is desirable for a decision-maker's policy to be highly stochastic. In such cases, an attractive alternative to the expected cumulative reward objective is the maximum entropy RL objective (Ziebart et al., 2008)

$$\max_{\pi} \mathbb{E}_{\pi} \left[\sum_{t} \mathcal{R}(S^t, A^t) + \alpha \mathcal{H}(A^t \mid S^t) \right],$$

which trades off between maximizing expected return and pursuing trajectories along which its actions have large cumulative conditional entropy, using the temperature hyperparameter α.

Mutual Information A closely related concept to entropy is mutual information. Mutual information describes the strength of the dependence between two random variables. The greater the mutual information between two random variables, the more the outcome of one affects the conditional distribution of the other. Symbolically, mutual information is defined by $\mathcal{I}(X; Y) = \mathcal{H}(Y) - \mathcal{H}(Y \mid X) = \mathcal{H}(X) - \mathcal{H}(X \mid Y)$. From this definition, we see explicitly that the mutual information of two random variables can be interpreted as the amount of uncertainty about one that is eliminated by observing the realization of the other.

Mutual information is important for communication because we may only be able to share the realization of an auxiliary random variable, rather than that of the random variable of interest. In such cases, maximizing the amount of communicated information amounts to maximizing the mutual information between the auxiliary random variable and the random variable of interest.

The Data Processing Inequality The independence relationships among random variables play an important role in determining their mutual information. If random variables X and Z are conditionally independent given Y (that is, $X \perp Z \mid Y$), the data processing inequality states that $\mathcal{I}(X; Y) \geq \mathcal{I}(X; Z)$. Less formally, the data processing inequality states that if Z does not provide additional information about X given Y, then the dependence between X and Z cannot be stronger than the dependence between X and Y.

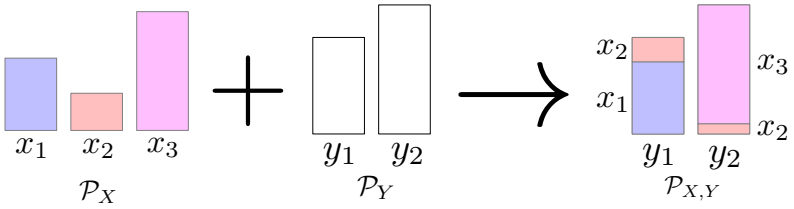


Figure 6.1: An approximate minimum entropy coupling. Given marginal distributions $\mathcal{P}_X$ and $\mathcal{P}_Y$, minimum entropy coupling constructs the a joint distribution $\mathcal{P}_{X,Y}$ having minimal joint entropy. In other words, it finds a joint distribution which allows to encode as much information as possible about X into a given marginal distribution $\mathcal{P}_Y$.

Minimum Entropy Coupling In some cases, we may wish to maximize the mutual information between two random variables subject to fixed marginals. That is, we are tasked with determining a joint distribution $\mathcal{P}_{X,Y}$ that maximizes the mutual information $\mathcal{I}(X;Y)$ between X and Y subject to the constraints that $\mathcal{P}_{X,Y}$ marginalizes to $\mathcal{P}_X$ and $\mathcal{P}_Y$, where $\mathcal{P}_X$ and $\mathcal{P}_Y$ are given as input. Invoking the relationship between mutual information and joint entropy $\mathcal{I}(X;Y) = \mathcal{H}(X)+\mathcal{H}(Y)-\mathcal{H}(X,Y)$, we see that this problem is equivalent to that of minimizing the joint entropy of X and Y. As a result, this problem is often referred to as the minimum entropy coupling problem. A visual example is shown in Figure 6.1. While minimum entropy coupling is NP-hard (Kovačević et al., 2015), Cicalese et al. (2019) recently showed that there exists a polynomial time algorithm that is suboptimal by no more than one bit.

[A] The following algorithmic description is taken from the paper's appendix.

6.2.1 Min Entropy Joint Distribution Algorithm outputting a sparse representation of M (Cicalese et al., 2019)

Algorithm 6 Lemma3-Sparse

Input: real $z > 0$, $x \geq 0$, and priority queue $\mathcal{Q}$ s.t. $\left(\sum_{(m,l)\in\mathcal{Q}} m\right) = qsum$ and $qsum = x \geq z$

Output: $z^{(d)}, z^{(r)} \geq 0$, and $I \subseteq \mathcal{Q}$ s.t. $z^{(d)} + z^{(r)} = z$, and $z^{(d)} + \sum_{(m,l)\in I} m = x$

$I \leftarrow \emptyset$, $sum \leftarrow 0$

while $\mathcal{Q} \neq \emptyset$ **and** $sum + Min(\mathcal{Q}) < x$ **do**

$(m,l) \leftarrow \text{ExtractMin}(\mathcal{Q})$, $qsum \leftarrow qsum - m$

$I \leftarrow I \cup \{(m,l)\}$, $z^{(r)} \leftarrow z - z^{(d)}$

$z^{(d)} \leftarrow x - sum$, $z^{(r)} \leftarrow z - z^{(d)}$ **return** $\left(z^{(d)}, z^{(r)}, I, qsum\right)$

Algorithm 7 Min Entropy Joint Distribution Cicalese et al. (2019)

Input: prob. distributions $\mathbf{p} = (p_1, \ldots, p_n)$ and $\mathbf{q} = (q_1, \ldots, q_n)$
Output: A Coupling $\mathbf{M} = [m_{ij}]$ of $\mathbf{p}$ and $\mathbf{q}$ in sparse representation $\mathbf{L} = \left\{\left(m_{ij}, (i,j)\right) | m_{ij} \neq 0\right\}$

if $\mathbf{p} \neq \mathbf{q}$, let $i = \max\left\{j | p_j \neq q_j\right\}$; **if** $p_i < q_i$ **then swap** $\mathbf{p} \leftrightarrow \mathbf{q}$.
$\mathbf{z} = (z_1, \ldots, z_n) \leftarrow \mathbf{p} \wedge \mathbf{q}$, $\mathbf{L} \leftarrow \emptyset$
CreatePriorityQueue$\left(\mathcal{Q}^{(row)}\right)$, qrowsum $\leftarrow 0$
CreatePriorityQueue$\left(\mathcal{Q}^{(col)}\right)$, qcolsum $\leftarrow 0$
for $i = n$ **downto** 1 **do**
  $z_i^{(d)} \leftarrow z_i$, $z_i^{(r)} \leftarrow 0$
  if $qcolsum + z_i > q_i$ **then**
    $\left(z_i^{(d)}, z_i^{(r)}, I, qcolsum\right) \leftarrow$ Lemma3-Sparse $\left(z_i, q_i, \mathcal{Q}^{(col)}, qcolsum\right)$
  else
    while $\mathcal{Q}^{(col)} \neq \emptyset$ **do**
      $(m, l) \leftarrow$ ExtractMin$(\mathcal{Q}^{(col)})$,
      $qcolsum \leftarrow qcolsum - m$,
      $\mathbf{L} \leftarrow \mathbf{L} \cup \{(m, (l, i))\}$
  if $qrowsum + z_i > p_i$ **then**
    $(z_i^{(d)}, z_i^{(r)}, I, qrowsum) \leftarrow$ Lemma3-Sparse$(z_i, p_i, \mathcal{Q}^{(row)}, qrowsum)$
    for each $(m, l) \in I$ **do** $\mathbf{L} \rightarrow \mathbf{L} \cup \{(m, (i, l))\}$
    if $z_i^{(r)} > 0$ **then** Insert $\left(\mathcal{Q}^{(row)}, (z_i^{(r)}, i)\right)$
    $qrowsum \leftarrow qrowsum + z_i^{(r)}$
  else
    while $\mathcal{Q}^{(row)} \neq \emptyset$ **do**
      $(m, l) \leftarrow$ ExtractMin$(\mathcal{Q}^{(row)})$
      $qrowsum \leftarrow qrowsum - m$
      $\mathbf{L} \leftarrow \mathbf{L} \cup \{(m, (i, l))\}$
  $\mathbf{L} \leftarrow \mathbf{L} \cup \left\{(z_i^{(d)}, (i, i))\right\}$

6.3 Markov Coding Games

We are now ready to introduce Markov coding games (MCGs). An MCG is a tuple $\langle(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}), \mathcal{M}, \mu, \zeta\rangle$, where $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ is a Markov decision process, $\mathcal{M}$ is a set of messages, μ is a distribution over $\mathcal{M}$, and ζ is a non-negative real number. An Markov coding game proceeds in the following steps:

1. First, a message $M \sim \mu$ is sampled from the prior over messages and revealed to the sender.
2. Second, the sender uses a message conditional policy to generate a trajectory $Z \sim (\mathcal{T}, \pi_{|M})$.

3. Third, the sender's terminal trajectory Z is given to the receiver as an observation.
4. Fourth, the receiver uses a trajectory conditional policy to guess the message $\hat{M} \sim \pi_{|Z}(Z)$.

The objective of the agents is to maximize the expected weighted sum of the return and the accuracy of the receiver's guess $\mathbb{E}\left[\mathcal{R}(Z) + \zeta \mathbb{I}[M = \hat{M}]\right]$. Optionally, in cases in which a reasonable distance function is available, we allow for the objective to be modified to minimizing the distance between the message and the guess $d(M, \hat{M})$, rather than maximizing the probability that the guess is correct.

Figure 6.2: Graphical representation of an MCG. First, the sender is given a message. Second, the sender is tasked with a MDP, unrelated to the message. Third, the receiver observes the sender's trajectory. Fourth, the receiver guesses the message.

6.3.1 An Example

Figure 6.3: A payoff matrix for a simple MCG.

Message	Sender	Receiver $\hat{m}$	Receiver $\hat{m}'$
m	a_1	(4, ζ)	(4, 0)
	a_1'	(3, ζ)	(3, 0)
	a_1''	(0, ζ)	(0, 0)
m'	a_1	(4, 0)	(4, ζ)
	a_1'	(3, 0)	(3, ζ)
	a_1''	(0, 0)	(0, ζ)

A payoff matrix for a simple MCG is shown in Figure 6.3. In this game, the sender is given one of two messages m or m', with equal probability. It then chooses among three actions a_1, a_1' and a_1'', for which the rewards are 4, 3, and 0, respectively. The receiver observes the sender's action and guesses the message using actions $\hat{m}$ and $\hat{m}'$, corresponding to guessing to m and m', respectively. The receiver accrues a reward of 4 for guessing correctly. The payoff entries in table denote $(\mathcal{R}(Z), \zeta \mathbb{I}[M = \hat{M}])$ for each outcome.

As is generally true of MCGs, this MCG is difficult for independent approximate dynamic programming-based approaches because their learning dynamics are subject to local optima. Consider a run in which the sender first learns to maximize its immediate reward by always selecting a_1. Now, the receiver has no reason to condition its guess on the sender's action because the sender is not communicating any information about the message. As a result, thereafter, the sender has no incentive to communicate any information in its message, because the receiver would

ignore it anyways. This outcome, sometimes called a babbling equilibrium, leads to a total expected return of $4 + \zeta/2$ (sender always selects a_1, receiver guesses randomly). In cases in which ζ is small (communication is unimportant), the babbling equilibrium performs well. However, it can perform arbitrarily poorly as ζ becomes large.

6.3.2 Special Cases

We can express both (a large class of) referential games and various source coding settings as special cases of the MCG formalism by describing the MDPs to which they correspond.

(A Large Class of) Referential Games We can express a T step referential game as follows.

- The state space $\mathcal{S} = \{s^0, s^1, \ldots, s^T\}$.
- The transition function deterministically maps $s^t \mapsto s^{t+1}$ and terminates at input s^T.
- The reward function maps to zero for every state action pair.

Standard Source Coding We can express the standard source coding problem as follows.

- The state space $\mathcal{S} = \{s\}$.
- The action space $\mathcal{A} = \tilde{\mathcal{A}} \cup \{\emptyset\}$.
- The transition function deterministically maps to s to s for $a \in \tilde{A}$ and terminates the game on $\emptyset$.
- The reward function maps to -1 for $a \in \tilde{\mathcal{A}}$ and maps $\emptyset$ to 0.

Length Limited Source Coding Length limited source coding can be captured in the same way as standard source coding with the modifications that $\mathcal{S} = \{s^0, s^1, \ldots, s^T\}$, the transition function terminates on s^T, and the reward function yields 0 from s^T, where T is the length limit.

Source Coding with Unequal Symbol Costs Source coding with unequal symbol costs can be captured in the same way as standard source coding with the modification that $\mathcal{R}(\cdot, a)$ returns the negative symbol cost of a, rather than returning -1, for $a \in \tilde{\mathcal{A}}$.

6.4 Greedy Minimum Entropy Coupling

To address MCGs, we propose a novel algorithm we call greedy minimum entropy coupling (GME). GME (and more broadly, any algorithm geared toward MCGs) is faced with two competing incentives. On one hand, it needs to maximize expected reward $\mathcal{R}(Z)$ generated by the MDP. On the other hand, it needs to maximize the amount of information $\mathcal{I}(M; Z)$ communicated to the receiver about the message, so as to maximize the probability of a correct guess. GME handles this trade-off using a two step process for constructing the sender's policy. In the first step, it

computes an MDP policy that balances between high cumulative reward and large cumulative entropy. In the second step, it couples the probability mass of this policy with the posterior over messages in such a way that the expected return does not decrease and the amount of mutual information between the message and trajectory is greedily maximized. We describe these steps below. Thereafter, we show this procedure possesses desirable guarantees and discuss intuition for the method.

6.4.1 Method Description

Step One In the first step, GME uses MaxEnt RL to construct an MDP policy π. This policy is an MDP policy, not an MCG policy, and therefore does not depend on the message. Note that this policy depends on the choice of temperature α used for the MaxEnt RL algorithm.

Step Two In the second step, at execution time, GME constructs a message-conditional policy online. Say that, up to time t, the sender is in state s^t, history h^t and has played according to the message conditional policy $\pi^{:t}_{|M}$. Let

$$b^t = \mathcal{P}(M \mid h^t, \pi^{:t}_{|M})$$

be the posterior over the message, conditioned on the history and the historical policy. GME performs a MEC between the posterior over the message b^t and distribution over actions $\pi(s^t)$, as determined by the MDP policy. Let $\nu = \text{MEC}(b^t, \pi(s^t))$ denote joint distribution over messages and actions resulting from the coupling. Then GME sets the sender to act according to the message conditional distribution

$$\pi^t_{|M}(s^t, m) = \nu(A^t \mid M = m)$$

of the coupling distribution $\text{MEC}(b^t, \pi(s^t))$.

After the MDP episode terminates, the receiver guesses the maximum a posteriori message.

Pseudocode for this procedure is included in the appendix.

[A] The following algorithmic description is taken from the paper's appendix.

Algorithm 8 GME

// Step 1: Compute MDP policy
Input: MDP G_{MDP}, temperature α
MaxEnt MDP policy $\pi \leftarrow \text{MaxEntRL}(G_{\text{MDP}}, \alpha)$
Return: π

// Step 2: Play MCG episode
Input: MDP policy π, MCG G_{MCG}
message $m \leftarrow \text{sample}(\mu)$
belief $b \leftarrow \mu$
state $s \leftarrow s^0$
while s non-terminal **do**
  joint distribution $\nu \leftarrow \text{minimum\_entropy\_coupling}(b, \pi(s))$
  decision rule $\pi_{|M} \leftarrow \nu(A \mid M)$
  sender action $a \sim \pi_{|M}(m)$
  new belief $b \leftarrow \text{posterior\_update}(b, \pi_{|M}, a)$
  next state $s \leftarrow \text{sample}(\mathcal{T}(s, a))$

6.4.2 Method Analysis

GME possesses guarantees both concerning the return $\mathcal{R}(Z)$ generated by the MDP and concerning the amount of information communicated $\mathcal{I}(M; Z)$.

Proposition 1. At each state of the MDP, the message conditional policy $\pi_{|M}$ selects actions with the same probabilities as the MDP policy π.

Proof. Fix an arbitrary state s. Let b be a posterior over the message induced by the sender's message conditional policy on the way to s. Then recall that GME uses the distribution $\mathcal{P}_{\text{MEC}(b,\pi(s))}$ induced by a MEC between b and $\pi(s)$ to select its action. Because MECs guarantee that the resultant joint distribution marginalizes correctly, it follows directly that GME must select its actions with the same probabilities as $\pi(s)$. □

Proposition 2. The expected return generated from the MDP by the message conditional policy $\pi_{|M}$ is equal to that of the MaxEnt policy π. That is,

$$\mathbb{E}_{M\sim\mu}\mathbb{E}_{\pi_{|M}}\mathcal{R}(Z) = \mathbb{E}_{\pi}\mathcal{R}(Z).$$

Proof. It follows from Proposition 1 that all trajectories are generated with the same probabilities and therefore that the expected returns are the same. □

Proposition 3. At each decision point, GME greedily maximizes the mutual information $I(M; H^{t+1} \mid b^t, h^t)$ between the message M the history at the next time step H^{t+1}, given the contemporaneous posterior and history, subject to Proposition 1.

Proof. For conciseness, we leave the conditional dependence on b^t and h^t implicit in the argument. First, we claim that $\mathcal{I}(M; H^{t+1}) = \mathcal{I}(M; A^t)$. To see this, first consider that $H^{t+1} \equiv (h^t, A^t, S^{t+1})$. This means we have $\mathcal{I}(M; H^{t+1}) = \mathcal{I}(M; (A^t, S^{t+1}))$ since we are conditioning on h^t. Now, consider that because the message influences the next state only by means of the action, we have the causal graph $M \to A^t \to (A^t, S^{t+1})$, which implies that $M \perp (A^t, S^{t+1}) \mid A^t$. Also, we trivially have $M \perp A^t \mid (A^t, S^{t+1})$. These conditional independence properties allow us to apply the data processing inequality: $X \perp Z \mid Y \Rightarrow \mathcal{I}(X; Y) \geq \mathcal{I}(X; Z)$. Applying it in both directions yields $\mathcal{I}(M; (A^t, S^{t+1})) = \mathcal{I}(M; A^t)$.

Now consider that we can rewrite mutual information using the equality

$$\mathcal{I}(M; A^t) = \mathcal{H}(M) + \mathcal{H}(A^t) - \mathcal{H}(M, A^t). \tag{6.1}$$

The first term $\mathcal{H}(M)$ is exogenous by virtue of being determined by b^t. The second term is exogenous by virtue of being subject to Proposition 1. The third term is the joint entropy between M and A^t, which is exactly what a minimum entropy coupling minimizes. □

6.4.3 Method Discussion

One aspect of GME that went uncommented upon in the analysis section is the choice of MaxEnt RL for step one. The reason that GME uses MaxEnt RL here is to control the $\mathcal{H}(A^t)$ term in equation (6.1). If the temperature α is large, GME places more emphasis on maximizing $\mathcal{H}(A^t)$ (and thereby $\mathcal{I}(M; Z)$); if the temperature α is small, GME places more emphasis on maximizing $\mathcal{R}(Z)$. The appropriate choice of α will depend on the value of ζ. Code for GME that illustrates the functionality of α applied to the example from Figure 6.3 can be found at `https://bit.ly/36I3LDm`.

A second aspect of GME that we have yet to discuss is its scalability. Performing an approximate MEC takes $O(N \log N)$ time, making it expensive to scale to very large message spaces. To accommodate this fact, for large message spaces we recommend using a factored representation $\mathbb{M} \subset \mathbb{M}_1 \times \cdots \times \mathbb{M}_k$ and a corresponding factored belief $(b_1, \ldots, b_k)$, where each b_j tracks the posterior over $\mathbb{M}_j$. At each time step, we suggest performing a MEC between $\pi(s^t)$ and the block $b_j = \arg\max_{b_j} \mathcal{H}(b_j)$ having the largest entropy. By doing so, we can substantially reduce the size the of the space over which we need to perform a minimum entropy coupling at any one time, allowing us to scale to extremely large message spaces.

[A] The following algorithmic description is taken from the paper's appendix. Note that Factored GME is not solely factoring the message space, but also updates

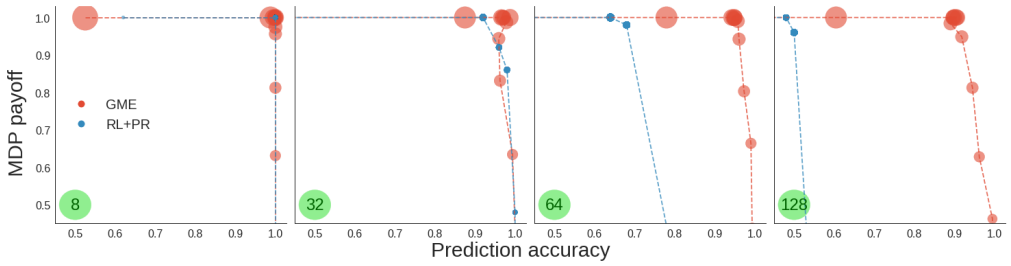


Figure 6.4: Results for GME and RL+PR on CodeGrid with varying message space sizes.

factors according to their current entropy in the optimisation process.

Algorithm 9 Factored GME

// Step 1: Compute MDP policy
Input: MDP G_{MDP}, temperature α
MaxEnt MDP policy $\pi \leftarrow \text{MaxEntRL}(G_{\text{MDP}}, \alpha)$
Return: π

// Step 2: Play MCG episode
Input: MDP policy π, MCG G_{MCG}
message $m \leftarrow \text{sample}(\mu)$
factored beliefs $b \leftarrow \text{factorize}(\mu)$
state $s \leftarrow s^0$
while s non-terminal **do**
  active block index $i \leftarrow \arg\max_j \left\{\mathcal{H}(b_j) | b_j \in b\right\}$
  joint distribution $\nu \leftarrow \text{minimum\_entropy\_coupling}(b_i, \pi(s))$
  decision rule $\pi_{|M} \leftarrow \nu(A \mid M_i)$
  sender action $a \sim \pi_{|M}(m_i)$
  new belief $b_i \leftarrow \text{posterior\_update}(b_i, \pi_{|M}, a)$
  next state $s \leftarrow \text{sample}(\mathcal{T}(s, a))$

One may asked whether our baseline RL+PR could be factorised similarly to GME for very large message spaces. However, the only way this seems to be doable is by giving the sender an auxiliary reward once each chunk has been successfully encoded. This, however, does not resolve the fundamental limitis of end-to-end training in a setting where there are many more tokens than can be trained over.

6.5 Experiments

We investigate the efficacy of GME on MCGs based on a gridworld, Cartpole and Pong. For all three we use MaxEnt Q-learning for our MDP policy. Details regarding our implementation can be found in the supplementary material and in the appendix.

[A] The following section is taken from the paper's appendix.

6.6 Experimental Details

For CodeGrid, we use a policy parameterized by neural network with two fully-connected layers of hidden dimension 64, each followed by a ReLu activation (Nair and Hinton, 2010). For CodePong and CodeCart, we use a convolutional encoder with three layers of convolutions (number of channels, kernel size, stride) as follows: (32,8,4), (64,4,2), (64,3,1). This is followed by a fully connected layer (Mnih et al., 2015). We use ReLu activations after each layer, note that we do not use any max-pooling. For CodeGrid and CodePong, layer weights are randomly initialized using PyTorch 1.7 (Paszke et al., 2017) defaults. For CodeCart, we initialise weights according to an optimally-trained DQN policy included in *rl-baselines3-zoo*[1] and individually finetune for each β for $15k$ steps. For all environments, we use the Adam optimizer with learning rate 10^{-4}, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and no weight decay.

CodeGrid See Figure 6.5.

CodeCart For CodeCart we use the standard action space (move left and move right). There are no redundant actions that the agent can use to communicate.

CodePong For CodePong we use a reduced action space of size two (move up and move down). There are no redundant actions that the agent can use to communicate.

[1] `https://github.com/DLR-RM/rl-baselines3-zoo`

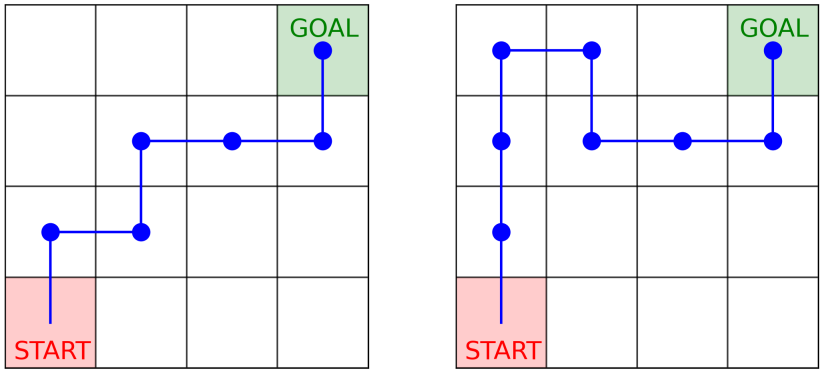


Figure 6.5: An illustration of two possible CodeGrid trajectories, one of length 6 (left) and one of length 8 (right).

CodeGrid In our gridworld MCG, which we call CodeGrid, the sender is placed on a 4×4 grid in which it can move left, right, up and down. The sender starts at position $(1, 1)$ and must to move to $(4, 4)$ within 8 time steps to achieve the maximal MDP return of 1. Otherwise, the game terminates after 8 time steps and the sender receives a payoff of 0.

For our baseline, we trained an RL agent to play as the MCG sender. We paired this RL agent with an perfect receiver, meaning that, for each episode, the receiver computed the exact posterior over the message based on the sender's current policy and guessed the MAP, both during training and testing. We abbreviate this baseline as RL+PR (where PR stands for perfect receiver). RL+PR is one way to adapt related work such as (Strouse et al., 2018; Eysenbach et al., 2018) to the MCG setting.

Pseudocode for the RL+PR baseline can be found in the appendix.

[A] The following algorithmic description is taken from the paper's appendix.

Algorithm 10 RL+PR baseline

// Play Episode
Input: MCG G_{MCG}
state $s \leftarrow s^0$
message $m \leftarrow \text{sample}(\mu)$
belief $b \leftarrow \mu$
while True **do**
  action $a \leftarrow \text{sample}(\pi(s, m))$
  belief $b \leftarrow \text{posterior\_update}(b, \pi, a)$
  new state $s' \leftarrow \text{sample}(\mathcal{T}(s, a))$
  if s' non-terminal **then**
    $\text{add\_to\_buffer}(s, a, \mathcal{R}(s, a), s')$
    state $s \leftarrow s'$
  else
    break
$\text{add\_to\_buffer}(s, a, \mathcal{R}(s, a) + \zeta \max_{m' \in \mathcal{M}} b(m'), \emptyset)$

We show results for GME and RL+PR across a variety of settings in Figure 6.4. The column of the figure corresponds to the cardinality of the message space; the exact size is listed in the green bubble. We use a uniform marginal over messages in each case. The x-axis corresponds to the proportion of the time that the receiver correctly guesses the message. The y-axis corresponds to the MDP payoff (in this case whether the sender reached the opposing corner of the grid within the time limit). Both GME and RL+PR possess mechanisms to trade-off between these two goals. GME can adjust its temperature α, while RL+PR can adjust the value of ζ used during training. Figure 6.4 show the Pareto curve for each. For GME, the size of the circle corresponds to the inverse temperature $\beta = 1/\alpha$.

For the settings with 8 messages and 32 messages, we observe that both GME and RL+PR achieve strong performance—achieving optimal or nearly optimal MDP returns and optimal or nearly optimal transmission. However, as the size of the message space increases, the performance of RL+PR falls off sharply. Indeed, for the 128 message setting, RL+PR is only able to correctly transmit the message slightly more than half the time. RL+PR's inability to achieve strong performance in these cases may be a result of the fact that communication protocols violate the locality assumption of approximate dynamic programming approaches. In contrast, we observe that GME, which constructs its protocol using MEC, remains optimal or near optimal both the 64 and the 128 message settings.

CodeCart and CodePong While the CodeGrid experiments that GME can outperform an obvious approach to MCGs, it remains to be determined whether GME can perform well at scale. Toward the end of making this determination, we consider MCGs based on Cartpole and Pong. In these MCGs, the message spaces are the sets of 16×16 and 22×22 binary images, respectively, each with a uniform prior. The cardinality of these spaces ($> 10^{77}$ and $> 10^{145}$) is astronomical. In fact,

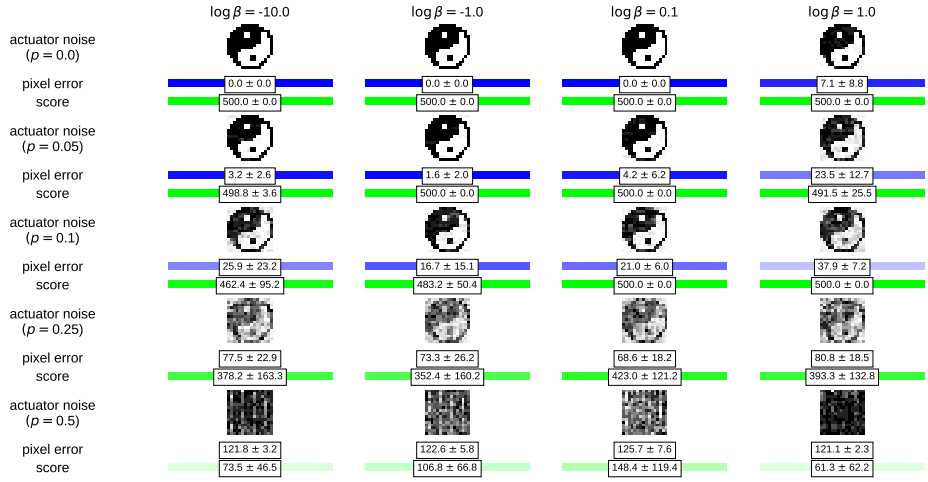


Figure 6.6: Results for GME on CodeCart with varying amounts of actuator noise and temperatures.

it is not clear how an RL+PR-like approach could even be scaled to this setting. On the other hand, GME is easily adaptable to this setting, using the factorization scheme suggested in the method section.

We also include results with variable amounts of *actuator noise*, i.e., with some probability p, the environment executes a random action, rather than the one it intended. Actuator noise models the probability of error during transmission and arises naturally in many settings involving communication, such as noisy channel coding. In this setting, the receiver may only observe the action executed by the environment, rather than the one intended by the sender. While this setting is not technically an MCG, GME can still be directly applied. And while the derivation of Proposition 3 no longer holds, we suggest that GME nevertheless offers a strong heuristic in such settings.

We show the results in Figure 6.6 and Figure 6.7 for CodeCart and CodePong, respectively. For both plots, the y-axis corresponds to the amount of actuator noise (lower is more noise). The x-axis corresponds the inverse temperature value $\beta = 1/\alpha$ (further right is colder temperature, meaning there is more emphasis on MDP expected return). Each entry contains a decoded yin-yang symbol with the corresponding temperature and actuator noise. Each entry also lists the ℓ_1 pixel error (blue) and the MDP expected return (green), along with corresponding standard errors over 10 runs; a brighter color corresponds to better performance.

Remarkably, for both CodeCart and CodePong, we observe that, when there is no actuator noise, GME is able to perfectly transmit images while achieving the maximum expected return in the MDP. For CodeCart, this occurs at $\log \beta \in \{-10, -1, 0.1\}$; for CodePong, it occurs at $\log \beta \in \{4, 5\}$. Interestingly, as the amount of actuator noise increases to a non-zero value, the effect on performance differs between CodeCart and CodePong. In CodeCart, GME continues to achieve maximal performance in the MDP, but begins to accumulate errors in the transmission. In contrast, in CodePong, GME continues to transmit the message with perfect fidelity, but begins

Figure 6.7: Results for GME on CodePong with varying amounts of actuator noise and temperatures.

to lose expected return from the MDP. This suggests that accidentally taking a random action is costlier in Pong than it is in Cartpole, but that, in an informal sense, Pong has more bandwidth to transmit information. However, in both cases, the decay in performance is graceful—for example, with $p = 0.05$, the decrease in the visual quality of the transmission for Cartpole ($\log \beta = -1$) is difficult to even perceive, while the CodePong sender still manages to win games roughly 80% of the time ($\log \beta = 5$). The performance in both settings continues to deteriorate up to $p = 1/2$, at which point GME is neither able to perform adequately on the MDP nor to transmit a clear symbol.

Videos of the agent playing are included in the supplemental material.

6.7 Conclusions

This work introduces a new problem setting called Markov coding games, which generalize referential games and source coding and are related to channel coding problems and decentralized control. We contribute an algorithm for this setting called GME and show that GME possesses provable guarantees regarding both the return generated from the MDP and the amount of information content communicated, subject to some constraints. On the experimental side, we show that GME significantly outperforms an RL baseline with a perfect receiver. Finally, we show that GME is able to scale to extremely large message spaces and transmit these messages with high fidelity, even with some actuator noise, suggesting that it could be robust in real world settings.

This concludes our quote.

While generalising a multitude of problem settings, MCGs themselves are a special subclass of *Dec-POMDPs* (Oliehoek and Amato, 2016) (or, more precisely, *Dec-POMDP-Coms* (Goldman and Zilberstein, 2003)), or, alternatively, *Factored Observation Stochastic Games* (Kovařík et al., 2020, FOSG). This perspective poses the question of whether various extensions of MCG settings admit learning algorithms that, similarly to GME, allow treating communication separately from task completion.

For example, seen from a FOSG-perspective, would GME (or an extension thereof) be applicable to settings where the sender's private information (i.e. the message token) in some way conditions on its trajectory? If so, such algorithms might be applicable to Hanabi-like settings (Bard et al., 2020). Similarly, could MCGs be meaningfully extended to Dec-POMDPs with more than two players? Investigating such extensions might allow the use of novel, scalable, hybrid end-to-end learning algorithms for larger classes of real-world applications.

The End.

We have spent the first two Parts of this thesis developing novel algorithms for collaborative deep multi-agent reinforcement learning. We have used value decomposition in order to refine the centralised learning of decentralised policies both in discrete and continuous action space settings, and we have introduced novel benchmark suites for each case. We then showed that agents can leverage group-wise common knowledge in order to learn joint policies that are nevertheless decentralisable. Finally, in this Chapter, we have shown how to construct implicit communication protocols that scale to very large message spaces.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	*Communicating via Markov Decision Processes*
Publication Status	☐ Published ☐ Accepted for Publication x Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	*S. Sokota*, **C. Schroeder de Witt***, M. Igl, L. Zintgraf, P. Torr, S. Whiteson, and J. Foerster. Communicating via Markov Decision Processes (2021).* arXiv:2107.08295 [cs.AI] (*: equal contribution, order determined by coin flip)

Student Confirmation

Student Name:	Christian Schroeder de Witt		
Contribution to the Paper	• Generation of Ideas with co-authors • Software Development • Designed and performed experiments • Wrote the paper with co-authors		
Signature	C. Schroeder de Witt	Date	30.04.2021

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title: Professor Philip H.S. Torr			
Supervisor comments			
Signature	Philip Torr	Date	04.05.2021

This completed form should be included in the thesis, at the end of the relevant chapter.

7
Afterword

Contents

In this thesis, we have presented a variety of novel cooperative DMARL methods, making progress on various fronts: Chapter 3 introduced *QMIX*, which has since turned out very influential on cooperative DMARL research, and *SMAC*, which has arguably become the most popular cooperative DMARL benchmark suite for discrete control at the time of writing. Chapter 4 introduces a novel state-of-the-art multi-agent algorithm, FACMAC, that not only excels in discrete control settings, but also extends value decomposition to continuous control settings. A novel benchmark environment, *MAMujoco*, for multi-agent robotic control settings is introduced in addition. In Chapter 5, we show how group-wise common knowledge between agents can allow them to develop complex coordination in fully decentralised settings, even if observations are noisy. And in Chapter 6, we introduce *GME*, a novel method that allows agents to learn implicit communication protocols that scale to very large message spaces. All contributions above significantly advance the field of cooperative multi-agent systems research.

At the same time, our work opens up exciting spaces for future research in the field of decentralised control. While some of these have already been discussed in the respective chapters, we here provide a concise overview of select opportunities that we have identified:

Value decomposition. Both QMIX (Chapter 3) and FACMAC (Chapter 4) may serve as starting points for further research on value factorisation under function approximation. For example, there is still a lack of theoretical guarantees surrounding convergence of such methods, and the limits of non-monotonic critic decomposition are also still underexplored.

Since publication of (Rashid et al., 2020b), significant progress has been made in understanding under what conditions such state-action value function decomposition may be a reasonable assumption. Oliehoek et al. (2021) show that for any *factored Dec-POMDP* for which a set of *local form models* (Oliehoek et al., 2021, Definition 10) can be created, the joint *value* function can indeed be decomposed into per-agent factors as long as each factor conditions on a sufficient statistic of respective agent's belief state. However, this notion does not directly carry over to *state-action* value functions. The empirical success of methods based on factored state-action value functions on environments like SMAC (Rashid et al., 2020b) or Multi-Agent Mujoco (Peng et al., 2021) show it to approximately hold across a number of real-world setups, however, we are currently unable to explain these observations conceptually. Another interesting hypothesis to investigate is the claim that on-policy algorithms with state-action value decomposition should ultimately perform better than off-policy algorithms Oliehoek et al. (2021, Section 8.1).

We argue that suitable benchmark environments are indispensable for the study of decentralised control. Both the StarCraft Multi-Agent Challenge, and, increasingly, Multi-Agent Mujoco, have received much attention by research on decentralised control. Developing novel benchmark environments that more closely resemble tangible real-world applications is an urgent avenue for further research in decentralised control. At the same time, as environments become more realistic, it

becomes increasingly important to ensure that the resulting research is for the benefit of humanity: Autonomous control has many benevolent use cases, for example in disaster response, delivery or industrial manufacture, however, the increasing proliferation of lethal autonomous weapons raises legitimate concerns.

Common knowledge. MACKRL (Chapter 5) offers a wealth of directions for follow-up research: The simple two-player probabilistic matrix setting could be extended to more players and different forms of noise, allowing the algorithm, as well as the performance of Holenstein sampling to be more accurately investigated, in such settings. To increase semblance to real-world applicability, it would similarly be interesting to see MACKRL's performance under *noisy observations in a more complex environment*, such as SMAC. To increase the method's scalability, different forms of backward graph pruning should be studied. Furthermore, it would be interesting to see how MACKRL can be scaled to off-policy algorithms, or to continuous control environments, such as Multi-Agent Mujoco. Settings where limited-bandwidth communication is available could be exploited to replace higher-level controllers for which common knowledge might not be available. Last but not least, the precise conceptual relationship between MACKRL's policy factorisation and value decomposition remains open.

Implicit Communication. GME (Chapter 6), particularly its factored variant, is a powerful illustration of how the learning of communication protocols can benefit from combining end-to-end gradient-based optimisation with efficient numerical heuristics. We believe that, given the pathologies of optimisation landscapes involved in learning communication protocols end-to-end, such hybrid approaches may be required to deliver both scalability and robustness properties necessary for real-world deployment. A next step in expanding GME's capabilities would be to study settings in which the sender's private information may change over time due to interaction with the environment or other agents. Such settings arise naturally in benchmark environments such as Hanabi(Bard et al., 2020).

Having reflected on our thesis, what now is the overall conclusion? First of all, work on DMARL, and in particular, cooperative DMARL, has only just begun. There is an infinite space of exciting research left to be done within decentralised control. But we should be careful in navigating this exciting space responsibly, and efficiently. One of the most important challenges in DMARL research right now is a lack of relevant benchmark environments that can guide meaningful method development. What could these benchmark environments look like?

Beyond decentralised control, we encourage young researchers entering the field to also think about DMARL applications to some of the world's most pressing issues. With respect to climate change applications, I propose looking into *scenario and pathway analysis* (van Vuuren et al., 2011): A new generation of agent-based Integrated Assessment Models that take both social and physical non-linearities into account (e.g. tipping points) is under construction as we speak. DMARL promises new avenues to solving these new models. As an example, consider some of our recent interdisciplinary collaboration on optimal decarbonisation pathway strategies for fossil fuel majors (Radovic et al., 2020). The Climate Change Initiative (CGI) [1] is poised to help making AI a positive force on climate change.

[1] `https://www.climategym.org`

Bibliography

Johannes Ackermann, Volker Gabler, Takayuki Osa, and Masashi Sugiyama. Reducing Overestimation Bias in Multi-agent Domains using Double Centralized Critics. *arXiv preprint arXiv:1910.01465*, 2019.

M. Aicardi, F. Davoli, and R. Minciardi. Decentralized Optimal Control of Markov Chains with a Common Past Information Set. *IEEE Transactions on Automatic Control*, 32(11):1028–1031, November 1987. ISSN 1558-2523. doi: 10.1109/TAC.1987.1104483. Conference Name: IEEE Transactions on Automatic Control.

Martin Allen and Shlomo Zilberstein. Complexity of Decentralized Control: Special Cases. In *Advances in Neural Information Processing Systems*, volume 22. Curran Associates, Inc., 2009.

Christopher Amato and Frans Oliehoek. Bayesian Reinforcement Learning for Multiagent Systems with State Uncertainty. In *Proceedings of the Ninth AAMAS Workshop on Multi-Agent Sequential Decision Making in Uncertain Domains (MSDM)*, pages 76–83, January 2013.

Christopher Amato and Frans Oliehoek. Scalable Planning and Learning for Multiagent POMDPs. In *AAAI 2015*, January 2015.

Christopher Amato, George D Konidaris, and Leslie P Kaelbling. Planning with Macro-actions in Decentralized POMDPs. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1273–1280. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

Brandon Amos, Lei Xu, and J. Zico Kolter. Input Convex Neural Networks. *arXiv:1609.07152 [cs, math]*, September 2016. arXiv: 1609.07152.

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, January 2020. ISSN 0278-3649. doi: 10.1177/0278364919887447. Publisher: SAGE Publications Ltd STM.

T. Anthony, Tom Eccles, A. Tacchetti, János Kramár, I. Gemp, Thomas C. Hudson, Nicolas Porcel, Marc Lanctot, Julien Pérolat, R. Everett, S. Singh, T. Graepel, and Yoram Bachrach. Learning to Play No-Press Diplomacy with Best Response Policy Iteration. *NeurIPS*, 2020.

Mauricio Araya, Olivier Buffet, Vincent Thomas, and Françcois Charpillet. A POMDP Extension with Belief-dependent Rewards. In *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010.

Ronald C Arkin. Motor Schema—based Mobile Robot Navigation. *The International journal of robotics research*, 8(4):92–112, 1989.

Sanjeev Arora, Nadav Cohen, Noah Golowich, and Wei Hu. A Convergence Analysis of Gradient Descent for Deep Linear Neural Networks. *arXiv preprint arXiv:1810.02281*, 2018.

Robert J Aumann et al. Subjectivity and Correlation in Randomized Strategies. *Journal of mathematical Economics*, 1(1):67–96, 1974.

Andrea Baisero and Christopher Amato. Unbiased Asymmetric Actor-Critic for Partially Observable Reinforcement Learning. *arXiv:2105.11674 [cs]*, May 2021. arXiv: 2105.11674.

Bram Bakker, Shimon Whiteson, L.J.H.M. Kester, and Frans Groen. Traffic Light Control by Multiagent Reinforcement Learning Systems. In *Studies in Computational Intelligence*, volume 281, pages 475–510. Springer, March 2010. ISBN 978-3-642-11687-2. doi: 10.1007/978-3-642-11688-9_18.

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi Challenge: A New Frontier for AI Research. *Artificial Intelligence*, 280: 103216, March 2020. ISSN 00043702. doi: 10.1016/j.artint.2019.103216. arXiv: 1902.00506.

L. E. Barnes, M. A. Fields, and K. P. Valavanis. Swarm Formation Control Utilizing Elliptical Surfaces and Limiting Functions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(6):1434–1445, 2009. doi: 10.1109/TSMCB.2009.2018139.

Mohammad Bavarian, Badih Ghazi, Elad Haramaty, Pritish Kamath, Ronald L. Rivest, and Madhu Sudan. Optimality of Correlated Sampling Strategies. *Theory of Computing*, 16:1–18, November 2020. doi: 10.4086/toc.2020.v016a012. Number: 12 Publisher: Theory of Computing.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013. ISSN 1076-9757. doi: 10.1613/jair.3912.

Marc G Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR, 2017.

Richard Bellman. *Dynamic Programming*. Princeton University Press, 1957.

Daniel S. Bernstein, Shlomo Zilberstein, and Neil Immerman. The Complexity of Decentralized Control of Markov Decision Processes. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI'00, pages 32–37, San Francisco, CA, USA, June 2000. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-709-5.

Marilena Bescuca. Factorised Critics in Deep Multi-agent Reinforcement Learning. In *Master Thesis, University of Oxford*, 2019.

Varun Bhatt and M. Buro. Inference-based Deterministic Messaging For Multi-Agent Communication. *ArXiv*, abs/2103.02150, 2021.

Vincent D. Blondel and John N. Tsitsiklis. A Survey of Computational Complexity Results in Systems and Control. *Automatica*, 36(9):1249–1274, September 2000. ISSN 0005-1098. doi: 10.1016/S0005-1098(00)00050-9.

Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep Coordination Graphs. In *International Conference on Machine Learning*, 2020.

Craig Boutilier. Sequential Optimality and Coordination in Multiagent Systems. In *IJCAI*, volume 99, pages 478–485, 1999.

Ronen I. Brafman and Moshe Tennenholtz. Learning to Coordinate Efficiently: a Model-based Approach. In *Journal of Artificial Intelligence Research*, volume 19, pages 11–23, 2003.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016. arXiv: 1606.01540.

Rodney Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE journal on robotics and automation*, 2(1):14–23, 1986.

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

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

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

Jacopo Castellini, Frans A. Oliehoek, Rahul Savani, and Shimon Whiteson. Analysing Factorizations of Action-Value Networks for Cooperative Multi-Agent Reinforcement Learning. *Autonomous Agents and Multi-Agent Systems*, 35(2):25, June 2021. ISSN 1573-7454. doi: 10.1007/s10458-021-09506-w.

Yong Chen, Ming Zhou, Ying Wen, Yaodong Yang, Yufeng Su, Weinan Zhang, Dell Zhang, Jun Wang, and Han Liu. Factorized Q-Learning for Large-Scale Multi-Agent Systems. *arXiv:1809.03738 [cs]*, September 2018. arXiv: 1809.03738.

Shushman Choudhury, Jayesh K. Gupta, Peter Morales, and Mykel J. Kochenderfer. Scalable Anytime Planning for Multi-Agent MDPs. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '21, pages 341–349, Richland, SC, May 2021. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-8307-3.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning*, 2014.

F. Cicalese, L. Gargano, and U. Vaccaro. Minimum-Entropy Couplings and Their Applications. *IEEE Transactions on Information Theory*, 65(6):3436–3451, 2019. doi: 10.1109/TIT.2019.2894519.

Ludek Cigler and Boi Faltings. Decentralized Anti-Coordination through Multi-Agent Learning. *Journal of Artificial Intelligence Research*, 47:441–473, 2013.

R. Cogill, M. Rotkowitz, Benjamin Van Roy, and S. Lall. An Approximate Dynamic Programming Approach to Decentralized Control of Stochastic Systems. *Lecture Notes in Control and Information Sciences*, pages 243–256, 2006.

Christopher Cox, Jessica De Silva, Philip Deorsey, Franklin H. J. Kenter, Troy Retter, and Josh Tobin. How to Make the Perfect Fireworks Display: Two Strategies for Hanabi. *Mathematics Magazine*, 88(5):323–336, 2015. ISSN 0025-570X. doi: 10.4169/math.mag.88.5.323. Publisher: Mathematical Association of America.

Allan Dafoe, Edward Hughes, Yoram Bachrach, Tantum Collins, Kevin McKee, Joel Leibo, Kate Larson, and Thore Graepel. *Open Problems in Cooperative AI*. Google DeepMind, December 2020.

Pieter-Tjerk de Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, 134(1):19–67, February 2005. ISSN 1572-9338. doi: 10.1007/s10479-005-5724-z.

Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowe. Learning Multi-Agent State Space Representations. In *AAMAS 2010*, volume 2, pages 715–722, January 2010. doi: 10.1145/1838206.1838301.

Yann-Michaël De Hauwere, Peter Vrancx, and Ann Nowe. Adaptive State Representations for MARL. *Belgian/Netherlands Artificial Intelligence Conference*, January 2011.

Christian Schroeder de Witt and Thomas Hornigold. Stratospheric Aerosol Injection as a Deep Reinforcement Learning Problem. *arXiv:1905.07366 [physics, stat]*, May 2019. arXiv: 1905.07366.

Thomas G. Dietterich. Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal Artificial Intelligence Research*, 13(1):227–303, November 2000. ISSN 1076-9757.

Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. LIIR: Learning Individual Intrinsic Reward in Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Charles Dugas, Yoshua Bengio, Franois Blisle, Claude Nadeau, and Ren Garcia. Incorporating Functional Knowledge in Neural Networks. *Journal of Machine Learning Research*, 10:1239–1262, 2009.

Benjamin Eysenbach and Sergey Levine. If MaxEnt RL is the Answer, What is the Question? *arXiv:1910.01913 [cs, stat]*, October 2019. arXiv: 1910.01913.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is All You Need: Learning Skills without a Reward Function. *CoRR*, abs/1802.06070, 2018.

A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings. Decentralised Coordination of Low-power Embedded Devices Using the Max-sum Algorithm. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2*, AAMAS '08, pages 639–646, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-1-6.

Alessandro Farinelli, Alex Rogers, and Nick R. Jennings. Bounded Approximate Decentralised Coordination using the Max-Sum Algorithm. In *In Distributed Constraint Reasoning Workshop*, 2009.

Jakob Foerster, Yannis M Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 2137–2145, 2016.

Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Philip Torr, Pushmeet Kohli, Shimon Whiteson, et al. Stabilising Experience Replay for Deep Multi-Agent Reinforcement Learning. In *Proceedings of The 34th International Conference on Machine Learning*, 2017.

Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Counterfactual Multi-Agent Policy Gradients*, 2018.

Jakob N. Foerster, H. Francis Song, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew Botvinick, and Michael Bowling. Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 36th International Conference on Machine Learning*, pages 1942–1951. PMLR, 2019.

Haotian Fu, Hongyao Tang, Jianye Hao, Zihan Lei, Yingfeng Chen, and Changjie Fan. Deep Multi-Agent Reinforcement Learning with Discrete-Continuous Hybrid Action Spaces. *arXiv:1903.04959 [cs, stat]*, March 2019. arXiv: 1903.04959.

Katie Genter, Tim Laue, and Peter Stone. Three Years of the RoboCup Standard Platform League Drop-in Player Competition. *Autonomous Agents and Multi-Agent Systems*, 31(4):790–820, 2017.

Mohammad Ghavamzadeh, Shie Mannor, Joelle Pineau, and Aviv Tamar. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 8 (5-6):359–483, November 2015. ISSN 1935-8237. doi: 10.1561/2200000049.

Claudia V. Goldman and Shlomo Zilberstein. Optimizing Information Exchange in Cooperative Multi-agent Systems. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '03, pages 137–144, New York, NY, USA, 2003. ACM. ISBN 978-1-58113-683-8. doi: 10.1145/860575.860598.

Mordecai J. Golin, Claire Kenyon, and Neal E. Young. Huffman Coding with Unequal Letter Costs. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, STOC '02, page 785–791, New York, NY, USA, 2002. Association for Computing Machinery. ISBN 1581134959. doi: 10.1145/509907.510020.

Anirudh Goyal and Yoshua Bengio. Inductive Biases for Deep Learning of Higher-Level Cognition. *arXiv:2011.15091 [cs, stat]*, February 2021. arXiv: 2011.15091.

Jonathan Gray, Adam Lerer, Anton Bakhtin, and Noam Brown. Human-Level Performance in No-Press Diplomacy via Equilibrium Search. *arXiv:2010.02923 [cs]*, October 2020. arXiv: 2010.02923.

Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous Deep Q-Learning with Model-based Acceleration. *arXiv:1603.00748 [cs]*, March 2016. arXiv: 1603.00748.

Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent Planning with Factored MDPs. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 1523–1530. MIT Press, 2002a.

Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. Coordinated Reinforcement Learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 227–234, San Francisco, CA, USA, 2002b. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-873-3.

Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-Specific Multiagent Coordination and Planning with Factored MDPs. In *AAAI/IAAI*, 2002c.

Arthur Guez, David Silver, and Peter Dayan. Efficient Bayes-adaptive Reinforcement Learning using Sample-based Search. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1025–1033, Red Hook, NY, USA, December 2012. Curran Associates Inc.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative Multi-agent Control using Deep Reinforcement Learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 66–83. Springer, 2017.

David Ha, Andrew Dai, and Quoc V. Le. HyperNetworks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement Learning with Deep Energy-Based Policies. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1352–1361. JMLR.org, 2017.

Joseph Y. Halpern and Yoram Moses. Knowledge and Common Knowledge in a Distributed Eenvironment. *arXiv:cs/0006009*, June 2000. arXiv: cs/0006009.

Matthew Hausknecht and Peter Stone. Deep Recurrent Q-Learning for Partially Observable MDPs. *arXiv:1507.06527 [cs]*, July 2015. arXiv: 1507.06527.

Fritz Heider and Marianne Simmel. An Experimental Study of Apparent Behavior. *The American Journal of Psychology*, 57(2):243–259, 1944. ISSN 0002-9556. doi: 10.2307/1416950.

Johannes Heinrich and David Silver. Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. *CoRR*, abs/1603.01121, 2016.

Alfred Olivier Hero, David Castanon, Doug Cochran, and Keith Kastella, editors. *Foundations and Applications of Sensor Management*. Signals and Communication Technology. Springer US, 2008. ISBN 978-0-387-27892-6. doi: 10.1007/978-0-387-49819-5.

Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable Baselines. `https://github.com/hill-a/stable-baselines`, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Comput.*, 9 (8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735.

Matt Hoffman, Bobak Shahriari, John Aslanides, Gabriel Barth-Maron, Feryal Behbahani, Tamara Norman, Abbas Abdolmaleki, Albin Cassirer, Fan Yang, Kate Baumli, Sarah Henderson, Alex Novikov, Sergio Gómez Colmenarejo, Serkan Cabi, Caglar Gulcehre, Tom Le Paine, Andrew Cowie, Ziyu Wang, Bilal Piot, and Nando de Freitas. Acme: A Research Framework for Distributed Reinforcement Learning. *arXiv preprint arXiv:2006.00979*, 2020.

Thomas Holenstein. Parallel Repetition: Simplifications and the No-signaling Case. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 411–419. ACM, 2007. ISBN 978-1-59593-631-8. doi: 10.1145/1250790.1250852.

Hengyuan Hu and Jakob N Foerster. Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning. In *International Conference on Learning Representations*, 2020.

Hengyuan Hu, Adam Lerer, Noam Brown, and Jakob Nicolaus Foerster. Learned Belief Search: Efficiently Improving Policies in Partially Observable Settings. *OpenReview*, September 2020.

Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Deep Reinforcement Learning for Swarm Systems. *arXiv:1807.06613 [cs, stat]*, July 2018. arXiv: 1807.06613.

Shariq Iqbal and Fei Sha. Actor-Attention-Critic for Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

Ken-ichi Iwata, Masakatu Morii, and T. Uyematsu. An Efficient Universal Coding Algorithm for Noiseless Channel with Symbols of Unequal Cost. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, pages 2232–2237, 01 1997.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical Reparameterization with Gumbel-Softmax. *arXiv:1611.01144 [cs, stat]*, November 2016. arXiv: 1611.01144.

Joe Booth. Richard Sutton SuperDyna talk NeurIPS 2019, December 2019.

Emilio Jorge, Mikael Kageback, and Emil Gustavsson. Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence. *arXiv preprint arXiv:1611.03218*, 2016.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. *arXiv:1806.10293 [cs, stat]*, June 2018. arXiv: 1806.10293.

Sammie Katt, Frans A. Oliehoek, and Christopher Amato. Learning in POMDPs with Monte Carlo Tree Search. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML 2017, pages 1819–1827, Sydney, NSW, Australia, August 2017. JMLR.org.

Sammie Katt, Frans A. Oliehoek, and Christopher Amato. Bayesian Reinforcement Learning in Factored POMDPs. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS 2019, pages 7–15, Richland, SC, May 2019. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-6309-9.

Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, Eiichi Osawa, and Hitoshi Matsubara. Robocup: A Challenge Problem for ai. *AI magazine*, 18(1):73–73, 1997.

Jelle R Kok and Nikos Vlassis. Sparse Cooperative Q-learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 61. ACM, 2004.

Jelle R. Kok and Nikos Vlassis. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *J. Mach. Learn. Res.*, 7:1789–1828, December 2006. ISSN 1532-4435.

Jelle R. Kok, Pieter Hoen, Bram Bakker, and Nikos Vlassis. Utile Coordination: Learning Interdependencies Among Cooperative Agents. In *CIG 2005*, January 2005.

Daphne Koller and Ronald Parr. Computing Factored Value Functions for Policies in Structured mdps. In *Proceedings of IJCAI*, pages 1332–1339, 1999.

Vijay R Konda and John N Tsitsiklis. Actor-Critic Algorithms. In *NIPS*, volume 13, pages 1008–1014, 1999.

Gizem Korkmaz, Chris J Kuhlman, Achla Marathe, Madhav V Marathe, and Fernando Vega-Redondo. Collective Action through Common Knowledge using a Facebook Model. In *Proceedings of the 2014 international conference on Autonomous agents and*

multi-agent systems, pages 253–260. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

Mladen Kovačević, Ivan Stanojević, and Vojin Šenk. On the Entropy of Couplings. *Information and Computation*, 242:369–382, 2015. ISSN 0890-5401. doi: https://doi.org/10.1016/j.ic.2015.04.003.

Vojtěch Kovařík, Martin Schmid, Neil Burch, Michael Bowling, and Viliam Lisý. Rethinking Formal Models of Partially Observable Multiagent Decision Making. *arXiv:1906.11110 [cs]*, October 2020. arXiv: 1906.11110.

Landon Kraemer and Bikramjit Banerjee. Multi-Agent Reinforcement Learning as a Rehearsal for Decentralized Planning. *Neurocomputing*, 190:82–94, May 2016.

Paul Krasucki, Rohit Parikh, and Gilbert Ndjatou. Probabilistic Knowledge and Probabilistic Common Knowledge. In *Probabilistic knowledge and probabilistic common knowledge*, pages 1–8, May 1991.

Akshat Kumar and S. Zilberstein. Anytime Planning for Decentralized POMDPs using Expectation Maximization. In *UAI*, 2010.

Akshat Kumar, Shlomo Zilberstein, and Marc Toussaint. Scalable Multiagent Planning using Probabilistic Inference. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2140–2146, Barcelona, Catalonia, Spain, July 2011. AAAI Press. ISBN 978-1-57735-515-1.

Saurabh Kumar, Pararth Shah, Dilek Hakkani-Tur, and Larry Heck. Federated Control with Hierarchical Multi-Agent Deep Reinforcement Learning. *arXiv preprint arXiv:1712.08266*, 2017.

Haruhisa Kurokawa, Kohji Tomita, Akiya Kamimura, Shigeru Kokaji, Takashi Hasuo, and Satoshi Murata. Distributed Self-Reconfiguration of M-TRAN III Modular Robotic System. *The International Journal of Robotics Research*, 27(3-4):373–386, 2008.

Lawrence L. Larmore and Daniel S. Hirschberg. Length-Limited Coding. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 310–318, USA, 1990. Society for Industrial and Applied Mathematics. ISBN 0898712513.

B. P. Lathi. *Modern Digital and Analog Communication Systems 3e Osece*. Oxford University Press, Inc., USA, 3rd edition, 1998. ISBN 0195110099.

Martin Lauer and Martin Riedmiller. An Algorithm for Distributed Reinforcement Learning in Cooperative Multi-Agent Systems. In *In Proceedings of the Seventeenth International Conference on Machine Learning*. Citeseer, 2000.

M. Lauri and F. Oliehoek. Multi-Agent Active Perception with Prediction Rewards. *NeurIPS*, 2020.

M. Lauri, Eero Heinänen, and S. Frintrop. Multi-Robot Active Information Gathering with Periodic Communication. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017. doi: 10.1109/ICRA.2017.7989104.

M. Lauri, J. Pajarinen, and Jan Peters. Information Gathering in Decentralized POMDPs by Policy Graph Improvement. In *AAMAS*, 2019.

Angeliki Lazaridou, Karl Moritz Hermann, Karl Tuyls, and Stephen Clark. Emergence of Linguistic Communication from Referential Games with Symbolic and Pixel Input. *arXiv:1804.03984 [cs]*, April 2018. arXiv: 1804.03984.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *Nature*, 521(7553): 436–444, May 2015. ISSN 1476-4687. doi: 10.1038/nature14539. Number: 7553 Publisher: Nature Publishing Group.

Joel Z. Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, pages 464–473, Richland, SC, May 2017. International Foundation for Autonomous Agents and Multiagent Systems.

Adam Lerer, Hengyuan Hu, Jakob N. Foerster, and Noam Brown. Improving Policies via Search in Cooperative Partially Observable Games. *AAAI*, 2020. doi: 10.1609/AAAI.V34I05.6208.

David K. Lewis. *Convention: A Philosophical Study.* Wiley-Blackwell, 1969.

Kevin Leyton-Brown and Yoav Shoham. *Essentials of Game Theory: A Concise, Multidisciplinary Introduction.* Morgan and Claypool Publishers, 1st edition, 2008. ISBN 1598295934.

Shihui Li, Yi Wu, Xinyue Cui, Honghua Dong, Fei Fang, and Stuart Russell. Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4213–4220, 2019.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. In *International Conference on Machine Learning (ICML)*, 2018.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous Control with Deep Reinforcement Learning. In *Continuous control with deep reinforcement learning.*, January 2016.

Alex Tong Lin, Mark J Debord, Katia Estabridis, Gary Hewer, and Stanley Osher. Cesma: Centralized Expert Supervises Multi-Agents. *arXiv preprint arXiv:1902.02311*, 2019.

Long-Ji Lin. Reinforcement Learning for Robots using Neural Networks. In *Dissertation, Carnegie Mellon University*, 1992a.

Long-Ji Lin. Self-improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching. *Machine Learning*, 8(3-4):293–321, May 1992b. ISSN 0885-6125, 1573-0565. doi: 10.1007/BF00992699.

Michael L. Littman, Thomas L. Dean, and Leslie Pack Kaelbling. On the Complexity of Solving Markov Decision Problems. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, UAI'95, pages 394–402, San Francisco, CA, USA, August 1995. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-385-1.

Miao Liu, Kavinayan Sivakumar, Shayegan Omidshafiei, Christopher Amato, and Jonathan P How. Learning for Multi-robot Cooperation in Partially Observable Stochastic Environments with Macro-actions. *arXiv preprint arXiv:1707.07399*, 2017.

Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. Emergent Coordination through Competition. *arXiv preprint arXiv:1902.07151*, 2019a.

Yong Liu, Yujing Hu, Yang Gao, Yingfeng Chen, and Changjie Fan. Value Function Transfer for Deep Multi-Agent Rreinforcement Learning Based on N-Step Returns. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pages 457–463, 2019b.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems*, pages 6379–6390, 2017.

Xueguang Lyu and Christopher Amato. Decentralized Likelihood Quantile Networks for Improving Performance in Deep Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:1812.06319v4*, 2019.

Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting Centralized and Decentralized Critics in Multi-Agent Reinforcement Learning. In *Proceedings of the 20th International Conference on Autonomous Agents and Multi-Agent Systems*, 2021.

David J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, USA, 2002. ISBN 0521642981.

Aditya Mahajan and Mehnaz Mannan. Decentralized Stochastic Control. *Annals of Operations Research*, 241:109–126, 2016.

Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-Agent Variational Exploration. In *Advances in Neural Information Processing Systems*, pages 7613–7624, 2019.

Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. Hierarchical multi-agent reinforcement learning. In *Proceedings of the fifth international conference on Autonomous agents*, pages 246–253. ACM, 2001.

Aleksandra Malysheva, Tegg Taekyong Sung, Chae-Bong Sohn, Daniel Kudenko, and Aleksei Shpilman. Deep Multi-Agent Reinforcement Learning with Relevance Graphs. *arXiv:1811.12557 [cs]*, November 2018. arXiv: 1811.12557.

Andreï Andreevitch Markov. Extension of the Law of Large Numbers to Quantities that Depend on each other. *Proceedings of the Physics and Mathematics Society at Kazan University*, 15(2):135–156, 1906.

Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Hysteretic Q-learning: An Algorithm for Decentralized Reinforcement Learning in Cooperative Multi-Agent Teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007.

Francisco Melo and Manuela Veloso. Learning of Coordination: Exploiting Sparse Interactions in Multiagent Systems. In *AAMAS 2009*, pages 773–780, January 2009. doi: 10.1145/1558109.1558118.

Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. ISSN 0021-9606. doi: 10.1063/1.1699114. Publisher: American Institute of Physics.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proceedings of the 20th National Conference on Artificial Intelligence - Volume 1*, AAAI'05, pages 133–139, Pittsburgh, Pennsylvania, 2005. AAAI Press. ISBN 978-1-57735-236-5.

Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, Madison, WI, USA, June 2010. Omnipress. ISBN 978-1-60558-907-7.

Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. Decentralized Stochastic Control with Partial History Sharing: A Common Information Approach, 2013.

Jean Oh and Stephen F. Smith. A Few Good Agents: Multi-Agent Social Learning. In *AAMAS*, 2008. doi: 10.1145/1402383.1402434.

Frans Oliehoek, Stefan Witwicki, and Leslie Kaelbling. A Sufficient Statistic for Influence in Structured Multiagent Environments. *Journal of Artificial Intelligence Research*, 70: 789–870, February 2021. ISSN 1076-9757. doi: 10.1613/jair.1.12136.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. SpringerBriefs in Intelligent Systems. Springer International Publishing, 2016. ISBN 978-3-319-28927-4. doi: 10.1007/978-3-319-28929-8.

Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and Approximate Q-Value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32(1):289–353, May 2008a. ISSN 1076-9757.

Frans A. Oliehoek, Matthijs T. J. Spaan, Shimon Whiteson, and Nikos Vlassis. Exploiting Locality of Interaction in Factored Dec-POMDPs. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 1*, AAMAS '08, pages 517–524, Richland, SC, May 2008b. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-0-9817381-0-9.

Frans A. Oliehoek, Shimon Whiteson, and Matthijs T. J. Spaan. Exploiting Structure in Cooperative Bayesian Games. *arXiv:1210.4886 [cs]*, October 2012a. arXiv: 1210.4886.

Frans A. Oliehoek, Stefan J. Witwicki, and Leslie P. Kaelbling. Influence-Based Abstraction for Multiagent Systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, AAAI'12, pages 1422–1428, Toronto, Ontario, Canada, July 2012b. AAAI Press.

Frans A. Oliehoek, Shimon Whiteson, and Matthijs T.J. Spaan. Approximate Solutions for Factored Dec-POMDPs with Many Agents. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '13, pages 563–570, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5.

Frans A. Oliehoek, Matthijs T. J. Spaan, and Stefan J. Witwicki. Factored Upper Bounds for Multiagent Planning Problems Under Uncertainty with Non-factored Value Functions. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1645–1651, Buenos Aires, Argentina, 2015. AAAI Press. ISBN 978-1-57735-738-4.

Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian. Deep Decentralized Multi-task Multi-Agent RL under Partial Observability. *arXiv preprint arXiv:1703.06182*, 2017.

J.M. Ooi and G.W. Wornell. Decentralized Control of a Multiple Access Broadcast Channel: Performance Bounds. In *Proceedings of 35th IEEE Conference on Decision and Control*, volume 1, pages 293–298 vol.1, December 1996. doi: 10.1109/CDC.1996.574318. ISSN: 0191-2216.

OpenAI. OpenAI/Baselines, May 2020. original-date: 2017-05-24T01:58:13Z.

Dirk Ormoneit and Śaunak Sen. Kernel-Based Reinforcement Learning. *Machine Learning*, 49(2):161–178, November 2002. ISSN 1573-0565. doi: 10.1023/A:1017928328829.

Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep Exploration via Bootstrapped DQN. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

Martin J Osborne and Ariel Rubinstein. *A Course in Game Theory*. MIT press, 1994.

Simon Pageaud, Véronique Deslandres, Vassilissa Lehoux, and Salima Hassas. Multiagent Learning and Coordination with Clustered Deep Q-Network. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 2156–2158, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

Joni Pajarinen and Jaakko Peltonen. Efficient Planning for Factored Infinite-Horizon DEC-POMDPs. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume One*, IJCAI'11, pages 325–331, Barcelona, Catalonia, Spain, July 2011. AAAI Press. ISBN 978-1-57735-513-7.

Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient Multi-Agent Deep Reinforcement Learning. *arXiv*, July 2017. arXiv: 1707.04402.

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, pages 43–51, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems. event-place: Montreal QC, Canada.

Ling Pan, Tabish Rashid, Bei Peng, Longbo Huang, and Shimon Whiteson. Softmax with Regularization: Better Value Estimation in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2103.11883*, 2021.

Liviu Panait, Karl Tuyls, and Sean Luke. Theoretical Advantages of Lenient Learners: An Evolutionary Game Theoretic Perspective. *Journal of Machine Learning Research*, 9(Mar):423–457, 2008.

Christos H. Papadimitriou. Computational Complexity. In *Encyclopedia of Computer Science*, pages 260–265. John Wiley and Sons Ltd., GBR, January 2003. ISBN 978-0-470-86412-8.

Sébastien Paquet, Ludovic Tobin, and Brahim Chaib-draa. An Online POMDP Algorithm for Complex Multiagent Environments. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '05, page 970–977, New York, NY, USA, 2005. Association for Computing Machinery. ISBN 1595930930. doi: 10.1145/1082473.1082620.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic Differentiation in PyTorch. In *NIPS Workshop on Automatic Differentiation*, 2017.

Bei Peng, Tabish Rashid, Christian A. Schroeder de Witt, Pierre-Alexandre Kamienny, Philip H. S. Torr, Wendelin Böhmer, and Shimon Whiteson. FACMAC: Factored Multi-Agent Centralised Policy Gradients. *arXiv:2003.06709 [cs, stat]*, May 2021. arXiv: 2003.06709.

Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games. *arXiv preprint arXiv:1703.10069*, 2017.

Leonid Peshkin. Reinforcement Learning by Policy Search, 2000.

Allan Pinkus. Approximation Theory of the MLP Model in Neural Networks. *Acta numerica*, 8:143–195, 1999.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv:1802.09464 [cs]*, March 2018. arXiv: 1802.09464.

Elise van der Pol and F. Oliehoek. Coordinated Deep Reinforcement Learners for Traffic Light Control, 2016.

PriceWaterhouseCoopers. PwC's Global Artificial Intelligence Study: Sizing the Prize, 2017.

Zinovi Rabinovich, C. Goldman, and J. Rosenschein. The Complexity of Multiagent Systems: the Price of Silence. In *AAMAS '03*, 2003. doi: 10.1145/860575.860816.

Neil Rabinowitz, Frank Perbet, Francis Song, Chiyuan Zhang, SM Ali Eslami, and Matthew Botvinick. Machine Theory of Mind. In *International Conference on Machine Learning*, pages 4218–4227, 2018.

Dylan Radovic, Lucas Kruitwagen, and Christian Schroeder de Witt. Revealing the Oil Majors' Adaptive Capacity to the Energy Transition with Deep Multi-Agent Reinforcement Learning. *Climate Change AI Workshop @ NeurIPS 2020*, page 9, 2020.

Tom Rainforth. Automating Inference, Learning, and Design using Probabilistic Programming. In *DPhil Thesis*, 2017.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, pages 4295–4304. PMLR 80, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted QMIX: Expanding Monotonic Value Function Factorisation. *arXiv preprint arXiv:2006.10800*, 2020a.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder de Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *Journal of Machine Learning Research*, 21 (178):1–51, 2020b. ISSN 1533-7928.

Amir Rasouli, Iuliia Kotseruba, and John K. Tsotsos. Agreeing to Cross: How Drivers and Pedestrians Communicate. *arXiv:1702.03555 [cs]*, February 2017. arXiv: 1702.03555.

Martin Riedmiller, Thomas Gabel, Roland Hafner, and Sascha Lange. Reinforcement Learning for Robot Soccer. *Autonomous Robots*, 27(1):55–73, 2009.

Ariel Rubinstein. The Electronic Mail Game: Strategic Behavior Under "Almost Common Knowledge". *The American Economic Review*, 79(3):385–391, 1989. ISSN 0002-8282.

Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real Single-Image Flight without a Single Real Image. *arXiv:1611.04201 [cs]*, June 2017. arXiv: 1611.04201.

A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):211–229, 1959.

Mikayel Samvelyan, Tabish Rashid, Christian Schroeder de Witt, Gregory Farquhar, Nantas Nardelli, Tim G. J. Rudner, Chia-Man Hung, Philiph H. S. Torr, Jakob Foerster, and Shimon Whiteson. The StarCraft Multi-Agent Challenge. *CoRR*, abs/1902.04043, 2019.

Christian Schroeder de Witt, Jakob Foerster, Gregory Farquhar, Philip Torr, Wendelin Boehmer, and Shimon Whiteson. Multi-Agent Common Knowledge Reinforcement Learning. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, pages 9927–9939. Curran Associates, Inc., 2019.

Christian Schroeder de Witt, Bei Peng, Pierre-Alexandre Kamienny, Philip H. S. Torr, Wendelin Böhmer, and S. Whiteson. Deep Multi-Agent Reinforcement Learning for Decentralized Continuous Cooperative Control. *arXiv:2003.06709v4 [cs.LG]*, 2020.

Sven Seuken and Shlomo Zilberstein. Memory-Bounded Dynamic Programming for DEC-POMDPs. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2009–2015, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

David Silver and Joel Veness. Monte-Carlo Planning in Large POMDPs. In *Proceedings of the 23rd International Conference on Neural Information Processing Systems - Volume 2*, NIPS'10, pages 2164–2172, Red Hook, NY, USA, December 2010. Curran Associates Inc.

Brian Skyrms. *Signals: Evolution, Learning, and Information*. Oxford University Press, 2010.

Kenny Smith. The Cultural Evolution of Communication in a Population of Neural Networks. *Connection Science*, 14, 04 2002. doi: 10.1080/09540090210164306.

Samuel Sokota, Christian Schroeder de Witt, Luisa Zintgraf, Maximilian Igl, Philip H. S. Torr, Shimon Whiteson, and Jakob Foerster. Communicating via Markov Decision Processes. *arXiv:2107.08295 [cs.AI]*, 2021.

Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 5887–5896, 2019.

Kyunghwan Son, Sungsoo Ahn, Roben Delos Reyes, Jinwoo Shin, and Yung Yi. Qtran++: Improved Value Transformation for Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2006.12010*, 2020.

M. Spaan, G. J. Gordon, and N. Vlassis. Decentralized Planning under Uncertainty for Teams of Communicating Agents. In *International joint conference on Autonomous agents and multiagent systems*, pages 249–256, 2006.

Matthew Spike, Kevin Stadler, Simon Kirby, and Kenny Smith. Minimal Requirements for the Emergence of Learned Signaling. *Cognitive Science*, 41(3):623–658, 2017. doi: https://doi.org/10.1111/cogs.12351.

Statista. Biggest Companies in the World by Market Cap 2020, 2020.

Luc Steels. Self-organization and Selection in Cultural Language Evolution. *Experiments in Cultural Language Evolution*, 03 2013. doi: 10.1075/ais.3.02ste.

Peter Stone and Richard S. Sutton. Scaling Reinforcement Learning toward RoboCup Soccer. In *Icml*, volume 1, pages 537–544. Citeseer, 2001.

Peter Stone, Gregory Kuhlmann, Matthew E Taylor, and Yaxin Liu. Keepaway Soccer: From Machine Learning Testbed to Benchmark. In *Robot Soccer World Cup*, pages 93–105. Springer, 2005.

Adam Stooke and Pieter Abbeel. rlpyt: A Research Code Base for Deep Reinforcement Learning in PyTorch. *arXiv preprint arXiv:1909.01500*, 2019.

D J Strouse, Max Kleiman-Weiner, Josh Tenenbaum, Matt Botvinick, and David Schwab. Learning to Share and Hide Intentions Using Information Regularization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 10270–10281, Red Hook, NY, USA, 2018. Curran Associates Inc.

Jianyu Su, Stephen Adams, and Peter A Beling. Value-Decomposition Multi-Agent Actor-Critics. In *Proceedings of the Thirty-Fifth AAAI Conference on Artificial Intelligence*, 2021.

Sainbayar Sukhbaatar, Arthur Szlam, and Rob Fergus. Learning Multiagent Communication with Backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning Based On Team Reward. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, pages 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems. event-place: Stockholm, Sweden.

Daniel Susskind and Richard Susskind. *The Future of the Professions: How Technology Will Transform the Work of Human Experts*. Oxford University Press, Oxford, New York, January 2016. ISBN 978-0-19-871339-5.

Jamie Susskind. *Future Politics: Living Together in a World Transformed by Tech*. Oxford University Press, Oxford, New York, October 2018. ISBN 978-0-19-882561-6.

Jamie Susskind. *Online Courts and the Future of Justice*. Oxford University Press, Oxford, New York, December 2019. ISBN 978-0-19-883836-4.

Richard Sutton. The Bitter Lesson, 2019.

Richard S Sutton. Learning to Predict by the Methods of Temporal Differences. *Machine learning*, 3(1):9–44, 1988.

Richard S. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. In *Advances in Neural Information Processing Systems 8*, pages 1038–1044. MIT Press, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 978-0-262-03924-6.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation. In *NIPS*, volume 99, pages 1057–1063, 1999.

Gabriel Synnaeve, Nantas Nardelli, Alex Auvolat, Soumith Chintala, Timothée Lacroix, Zeming Lin, Florian Richoux, and Nicolas Usunier. TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games. *arXiv preprint arXiv:1611.00625*, 2016.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent Cooperation and Competition with Deep Reinforcement Learning. *arXiv:1511.08779 [cs, q-bio]*, November 2015. arXiv: 1511.08779.

Ming Tan. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 330–337, 1993.

Moshe Tennenholtz. Program Equilibrium. *Games and Economic Behavior*, 49(2): 363–373, November 2004. ISSN 0899-8256. doi: 10.1016/j.geb.2004.02.002.

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`, 2012. Accessed: 2018-11-16.

Kyle A Thomas, Peter DeScioli, Omar Sultan Haque, and Steven Pinker. The Psychology of Coordination and Common Knowledge. *Journal of Personality and Social Psychology*, 107(4):657, 2014.

Rob Thomas. IBM BrandVoice: How AI Is Driving The New Industrial Revolution, 2020. Section: Innovation.

Sebastian Thrun. Probabilistic Algorithms in Robotics. *AI Magazine*, 21(4):93–93, December 2000. ISSN 2371-9621. doi: 10.1609/aimag.v21i4.1534. Number: 4.

Zheng Tian, Shihao Zou, Ian Davies, Tim Warr, Lisheng Wu, Haitham Bou-Ammar, and Jun Wang. Learning to Communicate Implicitly by Actions. 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*, pages 7261–7268. AAAI Press, 2020.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, September 2017. doi: 10.1109/IROS.2017.8202133. ISSN: 2153-0866.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A Physics Engine for Model-Based Control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. *arXiv:1804.06516 [cs]*, April 2018. arXiv: 1804.06516.

Kagan Tumer and Adrian Agogino. Distributed Agent-based Air Traffic Flow Management. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 255. ACM, 2007.

Nicolas Usunier, Gabriel Synnaeve, Zeming Lin, and Soumith Chintala. Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks. *arXiv preprint arXiv:1609.02993*, 2016.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *Thirtieth AAAI conference on artificial intelligence*, 2016.

Detlef P. van Vuuren, Jae Edmonds, Mikiko Kainuma, Keywan Riahi, Allison Thomson, Kathy Hibbard, George C. Hurtt, Tom Kram, Volker Krey, Jean-Francois Lamarque, Toshihiko Masui, Malte Meinshausen, Nebojsa Nakicenovic, Steven J. Smith, and Steven K. Rose. The Representative Concentration Pathways: An Overview. *Climatic Change*, 109(1):5, August 2011. ISSN 1573-1480. doi: 10.1007/s10584-011-0148-z.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. *arXiv:1703.01161 [cs]*, March 2017. arXiv: 1703.01161.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, John Quan, Stephen Gaffney, Stig Petersen, Karen Simonyan, Tom Schaul, Hado van Hasselt, David Silver, Timothy Lillicrap, Kevin Calderone, Paul Keet, Anthony Brunasso, David Lawrence, Anders Ekermo, Jacob Repp, and Rodney Tsing. Starcraft II: A New Challenge for Reinforcement Learning. *arXiv preprint arXiv:1708.04782*, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning. *Nature*, 575(7782):350–354, 2019.

Hong Shen Wang and N. Moayeri. Finite-state Markov Channel - a Useful Model for Radio Communication Channels. *IEEE Transactions on Vehicular Technology*, 44(1): 163–171, 1995. doi: 10.1109/25.350282.

Jianhao Wang, Zhizhou Ren, Beining Han, and Chongjie Zhang. Towards Understanding Linear Value Decomposition in Cooperative Multi-Agent Q-learning. *arXiv preprint arXiv:2006.00587*, 2020a.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. Qplex: Duplex Dueling Multi-Agent Q-learning. *arXiv preprint arXiv:2008.01062*, 2020b.

Tingwu Wang, Renjie Liao, Jimmy Ba, and Sanja Fidler. NerveNet: Learning Structured Policy with Graph Neural Networks. In *6th International Conference on Learning Representations, ICLR*, 2018.

Tonghan Wang, Jianhao Wang, Chongyi Zheng, and Chongjie Zhang. Learning Nearly Decomposable Value Functions Via Communication Minimization. *ICLR*, 2020c.

Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. Action Semantics Network: Considering the Effects of Actions in Multiagent Systems. *arXiv*, 2019a.

Weixun Wang, Tianpei Yang, Yong Liu, Jianye Hao, Xiaotian Hao, Yujing Hu, Yingfeng Chen, Changjie Fan, and Yang Gao. From Few to More: Large-scale Dynamic Multiagent Curriculum Learning. *arXiv*, 2019b.

Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Dop: Off-Policy Multi-Agent Decomposed Policy Gradients. In *9th International Conference on Learning Representations, ICLR*, 2021.

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

Ermo Wei and Sean Luke. Lenient Learning in Independent-Learner Stochastic Cooperative Games. *Journal of Machine Learning Research*, 17(84):1–42, 2016. ISSN 1533-7928.

Rudolf Paul Wiegand. *An Analysis of Cooperative Coevolutionary Algorithms*. phd, George Mason University, USA, 2004. AAI3108645.

Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696.

Keith Winstein and Hari Balakrishnan. TCP Ex Machina: Computer-generated Congestion Control. *ACM SIGCOMM Computer Communication Review*, 43(4): 123–134, August 2013. ISSN 0146-4833. doi: 10.1145/2534169.2486020.

Jeff Wu. WTFWThat. `https://github.com/WuTheFWasThat/hanabi.rs`, 2018.

Erfu Yang and Dongbing Gu. Multiagent Reinforcement Learning for Multi-Robot Systems: A survey. Technical report, tech. rep, 2004.

Jiachen Yang, Igor Borovikov, and Hongyuan Zha. Hierarchical Cooperative Multi-Agent Reinforcement Learning with Skill Discovery. *arXiv preprint arXiv:1912.03558*, 2019.

Yaodong Yang, Rui Luo, Minne Li, M. Zhou, Weinan Zhang, and J. Wang. Mean Field Multi-Agent Reinforcement Learning. In *ICML*, 2018.

Yaodong Yang, Jianye Hao, Ben Liao, Kun Shao, Guangyong Chen, Wulong Liu, and Hongyao Tang. Qatten: A General Framework for Cooperative Multiagent Reinforcement Learning. *arXiv preprint arXiv:2002.03939*, 2020.

Xinghu Yao, Chao Wen, Yuhui Wang, and Xiaoyang Tan. SMIX(λ): Enhancing Centralized Value Functions for Cooperative Multi-Agent Reinforcement Learning. *arXiv*, 2019.

William Yeoh, Akshat Kumar, and Shlomo Zilberstein. Automated Generation of Interaction Graphs for Value-Factored Decentralized POMDPs. *IJCAI '13: Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9*, pages 411–417, August 2013.

Mark Yim, Ying Zhang, and David Duff. Modular Robots. *IEEE Spectrum*, 39(2):30–34, 2002.

Kaiqing Zhang, Zhuoran Yang, and Tamer Basar. Multi-agent reinforcement learning: A Selective Overview of Theories and Algorithms. *CoRR*, abs/1911.10635, 2019a.

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*, pages 3230–3239, 2019b.

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*, pages 2321–2323, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

Lianmin Zheng, Jiacheng Yang, Han Cai, Weinan Zhang, Jun Wang, and Yong Yu. MAgent: A Many-Agent Reinforcement Learning Platform for Artificial Collective Intelligence. *arXiv preprint arXiv:1712.00600*, 2017.

Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning Implicit Credit Assignment for Multi-Agent Actor-Critic. In *Advances in neural information processing systems*, 2020.

Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd national conference on Artificial intelligence - Volume 3*, AAAI'08, pages 1433–1438, Chicago, Illinois, July 2008. AAAI Press. ISBN 978-1-57735-368-3.

Luisa M. Zintgraf, K. Shiarlis, Maximilian Igl, Sebastian Schulze, Y. Gal, Katja Hofmann, and S. Whiteson. VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning. *ICLR*, 2020.