# Deep Multi-Agent Reinforcement Learning


Jakob N. Foerster
Magdalen College
University of Oxford

A thesis submitted for the degree of
*Doctor of Philosophy*
Michaelmas 2018

To my parents,
Bärbel and Claus

# Acknowledgements

This thesis would not have been possible without the support of a large number of fantastic individuals and institutions. First and foremost I would like to thank my advisor Shimon Whiteson for having been supportive of my endeavours throughout my PhD, always providing useful advice and perspectives. I would also like to thank Nando de Freitas for co-advising me during the first year of my PhD, until our ways parted. I thank Peter Stone and Phil Blunsum for carefully examining this thesis and the constructive feedback. I thank Gregory Farquhar for having been an alter-ego for a lot of my PhD, happily discussing the wildest ideas. I thank Yannis Assael, Brendan Shillingford, and Oana Camburu for making the first year of my PhD unforgettable, successful, and fun. I also thank my collaborators and friends at WhiRL: Nantas Nardelli, Tabish Rashid, Christian Schroeder, Jelena Luketina, Max Igl, Luisa Zintgraf, Tim Rocktäschel, Wendelin Boehmer. I am extremely grateful to Francis Song from DeepMind for believing in the 'Bayesian Action Decoder' long before it started being a real thing. Without his support this section of the thesis would not have been possible. I am also thankful to Mike Bowling, Marc Bellemare, Marc Lanctot, Neil Burch, Nolan Bard, and others from DeepMind for the long lasting collaboration around Hanabi and to Thore Graepel for valuable advice. My thanks also go to Richard Chen and Maruan Al-Shedivat for being fantastic collaborators, and to Pieter Abbeel for his mentorship, during the LOLA project and beyond. I am indebted to DeepMind for the Oxford-Google-DeepMind scholarship. I am grateful to my close and/or long term friends for mostly bearing with me: Christoph, Jenny, the Rasumov & Ahnen families, Raphaele C. & M., Jakob, Julia, Brendan, Andreas, Dave, Dhruv, Molly, Kirill, Kasim, Linus, Adri, Yoni, Avital, Oiwi, Matthias, Cinjon, Paula, Fred, Paulina, Minqi, Valerie, Karol, Olesya, John, Javier, Katrina, Ashly, Oli, Tim, Lisa, Lars, Anne, Thais, Diego, Jaleh, James, Anshia, Ashish, Susanna, Ben, Fernanda, Will, Ian, Ryan, Tasha, Ronny, Leo, Ursin, Remo, and others. I thank Angelique for the walk we took in the alps in 2015, which inspired my thesis topic. I would also like to thank some of those that inspired and mentored me along the way: Odiê & Elias, Edgar, Harald and Sven, to name a few. Last not least I would like to thank my family: My parents, Bärbel and Claus, for always believing in me and encouraging me to be who I am. My brothers, Fridolin, Till, Moritz, and Peter, for continuously challenging me and helping me to grow. May the sun always shine on your paths.

# Abstract

A plethora of real world problems, such as the control of autonomous vehicles and drones, packet delivery, and many others consists of a number of agents that need to take actions based on local observations and can thus be formulated in the multi-agent reinforcement learning (MARL) setting. Furthermore, as more machine learning systems are deployed in the real world, they will start having impact on each other, effectively turning most decision making problems into multi-agent problems. In this thesis we develop and evaluate novel deep multi-agent RL (DMARL) methods that address the unique challenges which arise in these settings. These challenges include learning to *collaborate*, *to communicate*, and *to reciprocate* amongst agents. In most of these real world use cases, during *decentralised execution*, the final policies can only rely on local observations. However, in many cases it is possible to carry out *centralised training*, for example when training policies on a simulator or when using extra state information and free communication between agents during the training process.

The first part of the thesis investigates the challenges that arise when multiple agents need to learn to *collaborate* to obtain a common objective. One difficulty is the question of *multi-agent credit assignment*: Since the actions of all agents impact the reward of an episode, it is difficult for any individual agent to isolate the impact of their actions on the reward. In this thesis we propose *Counterfactual Multi-Agent Policy Gradients* (COMA) to address this issue. In COMA each agent estimates the impact of their action on the team return by comparing the estimated return with a counterfactual baseline. We also investigate the importance of *common knowledge* for learning coordinated actions: In *Multi-Agent Common Knowledge Reinforcement Learning* (MACKRL) we use a hierarchy of controllers that condition on the common knowledge of subgroups of agents in order to either act in the joint-action space of the group or delegate to smaller subgroups that have more common knowledge. The key insight here is that all policies can still be executed in a fully decentralised fashion, since each agent can independently compute the common knowledge of the group. In MARL, since all agents are learning at the same time, the world appears *nonstationary* from the perspective of any given agent. This can lead to learning difficulties in the context of off-policy reinforcement learning which relies on replay buffers. In order to overcome this problem we propose and evaluate a

metadata fingerprint that effectively disambiguates training episodes in the replay buffer based on the time of collection and the randomness of policies at that time.

So far we have assumed the agents act fully decentralised, *i.e.*, without directly communicating with each other. In the second part of the thesis we propose three different methods that allow agents to learn *communication protocols*. The first method, *Differentiable Inter-Agent Learning* (DIAL), uses differentiation across a discrete communication channel (specifically a *cheap-talk* channel) during centralised training to discover a communication protocol suited for solving a given task. The second method, *Reinforced Inter-Agent Learning* (RIAL), simply uses RL for learning the protocol, effectively treating the messages as actions. Neither of these methods directly reasons over the beliefs of the agents. In contrast, when humans observe the actions of others, they immediately form theories about why a given action was taken and what this indicates about the state of the world. Inspired by our insight, in our third method, the *Bayesian Action Decoder* (BAD), agents directly consider the beliefs of other agents using an approximate Bayesian update and learn to communicate both through observable actions and through grounded communication actions. Using BAD we obtain the best known performance on the imperfect information, cooperative card game *Hanabi*.

While in the first two parts of the thesis all agents are optimising a team reward, in the real world there commonly are conflicting interests between different agents. This can introduce learning difficulties for MARL methods, including unstable learning and the convergence to poorly performing policies. In the third part of the thesis we address these issues using *Learning with Opponent-Learning Awareness* (LOLA). In LOLA agents take into account the learning behaviour of the other agents in the environment and aim to find policies that shape the learning of their opponents in a way that is favourable to themselves. Indeed, instead of converging to the poorly performing defect-defect equilibrium in the iterated prisoner's dilemma, LOLA agents discover the tit-for-tat strategy. LOLA agents effectively *reciprocate* with each other, leading to overall higher returns. We also introduce the *Infinitely Differentiable Monte-Carlo Estimator* (DiCE), a new computational tool for estimating the higher order gradients that arise when one agent is accounting for the learning behaviour of other agents in the environment. Beyond being useful for LOLA, DiCE also is a general purpose objective that generates higher order gradient estimators for stochastic computation graphs, when differentiated in an auto-differentiation library.

To conclude, this thesis makes progress on broad range of the challenges that arise in multi-agent settings and also opens-up a number of exciting questions for future research. These include how agents can learn to account for the learning of other agents when their rewards or observations are unknown, how to learn communication protocols in settings of partial common interest, and how to account for the agency of humans in the environment.

# Contents

# List of Figures

# List of Abbreviations

**AC** . . . . . Actor-Critic.

**AI** . . . . . Artificial Intelligence.

**BAD** . . . . . Bayesian Action Decoder.

**COMA** . . . . Counterfactual Multi-Agent Policy Gradients.

**Dec-POMDP** Decentralised Partially Observable Markov Decision Process.

**DIAL** . . . . . Differentiable Inter-Agent Learning.

**DiCE** . . . . . Infinitely Differentiable Monte-Carlo Estimator.

**DMARL** . . . Deep Multi-Agent Reinforcement Learning.

**DQN** . . . . . Deep Q-Network.

**DRQN** . . . . . Deep Recurrent Q-Network.

**FF** . . . . . Feedforward Neural Network.

**FP** . . . . . Fingerprint.

**LOLA** . . . . . Learning with Opponent-Learning Awareness.

**IAC** . . . . . Independent Actor-Critic

**IPD** . . . . . Iterated Prisoner's Dilemma.

**IS** . . . . . Importance Sampling.

**IMP** . . . . . Iterated Matching Pennies.

**MACRKL** . . Multi-Agent Common-Knowledge Reinforcement Learning.

**MARL** . . . . Multi-Agent Reinforcement Learning.

**MDP** . . . . . Markov Decision Process.

**ML** . . . . . Machine Learning.

**MLP** . . . . . Multi-Layer Perceptron.

**NL** . . . . . Naive Learning.

**NN** . . . . . Deep Neural Network.

**OM** . . . . . Opponent-Modelling.

**PBT** . . . . . Population Based Training.

**PG** . . . . . Policy Gradient.

**RIAL** . . . . . Reinforced Inter-Agent Learning.

**RL** . . . . . Reinforcement Learning.

**RNN** . . . . . Recurrent Neural Network.

**SCG** . . . . . Stochastic Computation Graphs.

**SF** . . . . . Score Function.

**SL** . . . . . Surrogate Loss.

**STD** . . . . . Standard Deviation.

**TFT** . . . . . Tit-For-Tat

**XP** . . . . . Experience Replay.

# Notation Used

| | |
|---|---|
| $s \in \mathcal{S}$ | State and state space. |
| $n$ | Number of agents. |
| $t$ | Timestep in episode. |
| $T$ | Duration of episode. |
| $a \in \{1, ..n\}$ | Agent index. |
| $o_t^a = O(s_t, a)$ | 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 $t$. |
| $\gamma$ | Discount factor. |
| $R_t^a = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^a$ | Forward-looking return of agent $a$. |
| $\tau_t^a = \{o_0^a, u_0^a, .., o_t^a\}$ | Action observation history of agent $a$. |
| $\pi^a(u^a \vert \tau^a)$ | Policy of agent $a$. |
| $m_t^a$ | Message from agent $a$ sent at time $t$. |
| $\mathbf{u}, \boldsymbol{\pi}, \boldsymbol{\tau}$ | Joint- action, -policy, -observation history across all agents. |
| $\mathbf{u}^{-a}, \boldsymbol{\pi}^{-a}, \boldsymbol{\tau}^{-a}$ | Joint action etc. excluding agent $a$. |
| $P(s' \vert s, \mathbf{u})$ | State transition function. |
| $V^{\boldsymbol{\pi}}(s)$ | State value function for policy $\boldsymbol{\pi}$. |
| $Q^{\boldsymbol{\pi}}(s, \mathbf{u})$ | $Q$-function: State-action value function, conditioned on 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 joint policy $\boldsymbol{\pi}$. |
| $h_t^a$ | Hidden state of a recurrent network for agent $a$. |

# 1 Introduction

## Contents



## 1.1 The Industrial Revolution, Cognition, and Computers

The steam engine [1], and the industrial revolution that followed, lead to a rapid replacement of human labour by machines. These developments brought about drastic changes to every aspect of human life, leading to the creation of entirely new industries and societies. In particular, while in pre-industrial societies a vast majority of employment was in the producing sector, most workers in modern societies are employed in the service sector [2]. Importantly, rather than requiring physical strength, these jobs require cognitive skills. Examples include doctors and lawyers, but also taxi drivers, customer service and nurses.

What the steam engine, fossil fuels and other energy sources achieved for physical labour, computing technology should in principle be able to achieve for those tasks that require human cognition. However, while computation is the digital analog

of cognition, at first sight there are striking differences between computers and brains. While computers operate using deterministic binary gates implemented in silicon, the brain uses noisy biological neurones with probabilistic firing patterns [3]. Furthermore, there is a stark difference between the instruction sets that the brain and the computer execute: Computers require carefully constructed programs to carry out computation, in which any single bit can cause failure. In contrast, all neural-programs running on the brain are learned via repeated interaction with the environment, rather than provided by a programmer.

Historically, artificial intelligence (AI) research has been focused on recreating human-like reasoning abilities through expert systems that execute rule sets provided by the designer [4]. However, while recognising a dog in a picture is a trivial task for most humans, specifying a set of rules that achieve this reliably across a variety of different points of view and backgrounds has proven to be a superhuman challenge.

Machine Learning (ML) is an alternative approach for bringing cognitive abilities to machines. Importantly, in the ML paradigm the designer no longer needs to specify a set of rules for recognising a dog. Instead, it is sufficient to specify a set of *learning rules* which in conjunction with a labeled dataset of examples allow the algorithm to extract decision rules. Over the last 30 years, this approach has proven successful and transformed many areas of modern life. Example learning algorithms include linear regression, support-vector-machines [5], gaussian processes [6], and many others.

During the last decade, *Deep Learning* [7], in particular has shown tremendous success. Prominent success stories include speech-recognition [8], image recognition [9], lip reading [10], and language translation [11] amongst many others. All of these success stories consist of a large dataset of inputs and desired outputs, a setting commonly referred to as 'supervised learning'. Importantly, in supervised learning the training dataset is always assumed to by independent of the classification decisions that the algorithm makes.

This assumption is violated as soon as the classification decisions that the algorithm makes actively change the future training data. One area where this commonly occurs is when algorithms take actions that affect a *stateful* environment.

For example, when a self-driving car takes a specific action during training, this changes the kind of data and experiences that the car is exposed to in the later parts of the training process. A large number of real world problems fall into this category. For example, a ranking algorithm has impact on the kind of decisions users are facing and thus changes the future training data. Similarly when a cleaning robot knocks over a flower pot this changes the future state distribution.

All of these problems can be formalised in the Reinforcement Learning (RL) framework [12]. In RL, an agent, *e.g.*, the robot, sequentially interacts with the environment, *e.g.*, the living room, by taking actions based upon the observation it makes, *e.g.*, the camera input. The action space defines which actions are available to the agent, for the cleaning robot these might be navigation actions such as 'move left', 'move right', and so on. At every time-step the agent receives both an observation and a reward from the environment.

The agent's mapping from observations to actions is called the 'policy' and RL aims to find a policy that maximises the expected sum of discounted rewards across an episode. Here 'discounted' means that rewards occurring later in the episode, *i.e.*, further in the future, have less importance than early ones. An episode consists of a sequence of observations, rewards, and actions and ends whenever a 'terminal state' is reached, in which no further rewards are given and from which the agent is unable to leave. Importantly, the action chosen can change the immediate reward, but also changes the probability distribution over the next state, which in turn impacts future rewards. Furthermore, the agent is not a-priori provided with the rules governing the state-transition probabilities or the reward function, but rather has to learn from interaction with the environment.

Deep RL (DRL) refers to a subcategory of RL in which deep neural networks [7] are used as function approximators. In particular DRL allows agents to process high dimensional inputs and learn relevant feature representations as well as policies. This comes at the cost of requiring a large number of tuneable parameters. Fortunately these parameters can be trained efficiently using backpropagation. In recent years DRL has successfully been applied to a number of domains, including the playing

of Atari games [13], Go [14], and other challenging settings. All algorithms in this thesis can be applied to the DRL setting.

## 1.2 Deep Multi-Agent Reinforcement-Learning

For the most part, progress in DRL has been focused on settings where a single agent needs to solve an otherwise static task, corresponding to the single-agent setting.

In contrast, many real world challenges involve environments that contain a large number of learning agents, and are thus *multi-agent* in nature. Examples include self driving cars, garbage collection, packet delivery, and others. In all of these settings a large number of distributed agents need to be able to take independent decisions based on local observations in order to contribute most effectively to an overall goal or to maximise individual rewards, taking into account the presence of other agents in the environment. Notably, the other agents may consist both of other learning algorithms but may also include biological agents such as humans or other animals.

Multi-Agent RL (MARL) is the framework and field of study for addressing these kinds of problems. MARL is concerned with developing and analysing learning rules and algorithms that can discover effective policies in these multi-agent settings.

Beyond these settings, MARL is likely to play a major role in years to come due to the following two trends: First of all, applications of machine learning technology are becoming ubiquitous in our society. As such it is likely that AI systems will need to take into account the presence of other learning systems within their environment. When algorithms do not take into account the presence of other algorithmic decision makers in their environment, seemingly sensible rules can lead to drastic, unintended outcomes in their interaction, as could be observed with exploding prices for certain second hand books on Amazon, and in the 2010 flash crash, just to name a few. Furthermore, as the AI systems themselves are becoming more influential, they will likely start affecting the personality development of the human users themselves. For example, it is no longer sufficient to assume that the user's preferences are static and to attempt serving those preferences. Instead the agency of users needs to be considered by reasoning over beliefs, ideals, and desires.

Another reason to study MARL is that it can be regarded as a stepping stone towards developing systems that have human like reasoning abilities, even though such a goal is still in the distant future. This motivation is supported by the fact that the development of human level intelligence occurred in the social context of many agents interacting. Indeed, research indicates [15] that amongst primates the group size correlates strongly with the percentage of the brain occupied by the prefrontal cortex, an area responsible for higher level cognitive functions.

This is very intuitive: for many intelligent agents, such as humans and monkeys, the most complex relevant part of the environment are other agents and the interactions between them. It also suggests that higher level cognitive skills such as abstraction could naturally arise in these settings: Clearly when an agent has to account for the current state of mind of another agent as part of their state of mind, this corresponds to a level of abstract thought.

The same will hold true for future intelligent agents, which will likely have to interact with humans and other agents on an everyday basis. Beyond higher level reasoning abilities, humans have developed a large range of tools in order to allow for this interaction to be efficient and smooth. These tools include concepts such as 'reciprocity', 'contracts', 'beliefs', but also language, habits, culture, and dedicated representational and computational abilities, such as empathy.

They include the ability to change perspective when observing the actions of others and to understand why someone is taking a given action. The last part is commonly referred to as *theory of mind*, in which one agent can take account of the beliefs, desires or points of view of another agent. All of these can be regarded as part of the answer evolution discovered over hundreds of thousands, or even millions of years, in order to resolve the challenges that arise in multi-agent settings, allowing humans to coordinate an ever greater number of agents in the same environment.

In this thesis we develop novel deep multi-agent RL (DMARL) algorithms that allow groups of agents to acquire some of these abilities. In particular we focus on the following three challenges: *collaboration*, *communication*, and *reciprocity*. *Collaboration* describes the challenges that arise due to a number of

agents learning at the same time, rendering the learning problem continuously changing, or *nonstationary*. It also includes the problem of multi-agent *credit assignment*: Due to the large number of agents taking actions, it is commonly unclear whether a given action by a specific agent had an overall positive or negative effect. Effectively the other agents in the environment act as confounding factors in the reward attribution of a given agent. Furthermore, the optimal action of each agent can depend crucially on the (unobserved) action selection of other agents, making the learning of coordinated policies challenging.

*Communication* addresses the challenges associated with learning communication protocols. In many real world settings agents have to take decentralised actions but have access to a limited bandwidth discrete communication channel. Learning how to exchange information through this channel in a way that is useful for solving a given task is a hard problem.

While all of the challenges addressed so far appear in fully cooperative settings (*i.e.*, when agents are aiming to maximise a joint team-reward), *reciprocity* is a difficult challenge that appears in general-sum settings. In these settings agents can commonly obtain higher rewards if they manage to encourage other agents to collaborate with them. Humans naturally reciprocate with other humans, even in situations of military conflict [16], but bringing these abilities to learning algorithms is an open problem.

When addressing these problems we will commonly take advantage of *centralised training* with *decentralised execution*: For many real world problems training can be carried out on in a centralised fashion, for example by using a simulator or providing extra state information during the training process, while during execution each agent needs to chose their action independently based on local observations only.

Importantly, centralised training can greatly facilitate the learning process in multi-agent settings, assuming algorithms are able to exploit the central state information during training without requiring it during execution. This setting of centralised training and decentralised execution in MARL is thus an important avenue for deploying reinforcement learning algorithms in the real world and is

used throughout this thesis. Interestingly, we can use centralised training even in non-cooperative settings in order to learn better strategies, since the central state information is not required during execution.

## 1.3 Overall Structure

The thesis is divided into a background section followed by three main parts, each one addressing one of the MARL challenges outlined above. In the next subsections we provide a brief overview of the different sections.

### Background

In Chapter 2 we formally introduce the multi-agent RL setting and provide the necessary algorithmic and conceptual tools of deep reinforcement learning which are *common* to the rest of the thesis. In order to make it easier for readers to digest the background information, background concepts required only for a specific chapter are introduced within the corresponding chapter.

### Part 1: Learning to Collaborate

The most straightforward adaptation of single agent RL techniques to the MARL setting is called *naive learning* (NL). In this algorithm each agent carries out single agent RL, treating all other agents as a *static* part of the environment. NL offers the key advantage that it makes no assumptions about the learning behaviour of other agents. Furthermore, NL is commonly found to be a robust method that works surprisingly well in a number of multi-agent settings [17]. Across this thesis NL will serve as a benchmark to compare our algorithmic innovations against.

As mentioned above, one of the great challenges of MARL is the question of credit assignment: Since all agents are exploring and learning at the same time, it is difficult for any given agent to estimate the impact of their action on the overall return. For example, an agent might have chosen the optimal action in a given state, but the returns are lower than average since the teammate took an exploratory action. The agent will thus (falsely) learn to reduce the probability

of selecting this (optimal) action. This issue arises in particular in NL since each agent tracks the expected reward as a function of its own action selection, omitting the choices made by other agents.

In Chapter 3 we propose *Counterfactual Multi-Agent Policy Gradients* (COMA). COMA exploits the centralised training regime by using a centralised critic which learns a value function that conditions on the central state and the joint-action of all agents. Inspired by difference rewards [18], we use this value function to calculate a counterfactual baseline. This baseline is an estimate of what would have happened on average had the agent chosen a different action. Applied to a multi-agent version of StarCraft micromanagement, we find that COMA outperforms a set of strong baselines.

While COMA learns a joint value function, the policies of the agents are fully factorised. In other words, actions are sampled independently such that the probability of a joint action is the product of probabilities across the different agents. In some settings these kinds of factorised policies will not be able to learn optimal strategies.

In particular, whenever the optimal action selection depends crucially on the other agents also selecting the optimal action, the exploration of one agent can shift the best response of the other agent away from the optimal action.

These kinds of settings can in principle be solved by centralised controllers which learn to act in the joint action space. However, due to partial observability these centralised policies cannot in general be executed in a decentralised fashion.

In Chapter 4 we introduce *Multi-Agent Common Knowledge Reinforcement Learning* (MACKRL). MACKRL uses the *common knowledge* of a group of agents in order to learn a joint-action policy which can be executed in a fully decentralised fashion. Here the *common knowledge* of a group of agents are those things that all agents know and that all agents know that all agents know, ad infinitum. Interestingly, in a variety of MARL settings the agents can observe other agents and thereby form common knowledge. In particular MACKRL relies on hierarchical controllers, which can either assign a joint action for a group of agents or decide

on a partition of agents that should coordinate instead. Importantly, while each smaller subgroup will have less ability to coordinate the action selection, they will in general have more common knowledge. Thus the MACKRL controller learns to trade-off the need for coordinated action selection with the greater amount of information that is available to independent actors.

COMA and MACKRL are on-policy methods, *i.e.*, they use training data that was collected under the current policy. However, off-policy methods such as $Q$-learning can provide better sample efficiency. In order to stabilise learning, off-policy DRL relies heavily on using a replay memory: During training experiences are stored in the replay memory and then sampled randomly to provide a diverse range of state-action pairs to the agent.

However, the presence of multiple learning agents in the environment renders the learning problem nonstationary from the perspective of any given agent: The expected return for a given state-action pair for a given agent also depends on the kind of policies the other agents in the team are deploying. Since all agents are updating their policies, this expected return is continuously changing.

To address this fundamental challenge, in Chapter 5 we propose a novel method for stabilising experience replay during centralised training using a version of multi-agent importance weights and a metadata *fingerprint* which is added to the input of each agent during training. This metadata *fingerprint* disambiguates during which stage of training the episode was collected and thereby allows the agent to implicitly distinguish between different strategies of the teammates.

These chapters are based on following papers and pre-prints. Here and throughout the '*' indicates equal contribution:

- 'Counterfactual Multi-Agent Policy Gradients', Jakob Foerster*, Gregory Farquhar*, Triantafyllos Afouras, Nantas Nardelli, Shimon Whiteson, *AAAI Conference on Artificial Intelligence 2018*, **Outstanding Student Paper Award**

- 'Stabilising experience replay for deep multi-agent reinforcement learning' Jakob Foerster*, Nantas Nardelli*, Gregory Farquhar, Triantafyllos Afouras, Philip H. S. Torr, Pushmeet Kohli, Shimon Whiteson, *International Conference on Machine Learning*, 2017

- 'Multi-Agent Common-Knowledge Reinforcement Learning', Christian A. Schroeder de Witt*, Jakob N. Foerster*, Gregory Farquhar, Philip H. S. Torr, Wendelin Boehmer, Shimon Whiteson, *Advances in Neural Information Processing Systems*, 2019

## Part 2: Learning to Communicate

So far we have considered settings without any explicit communication between agents. However, in many real world applications reliable, limited bandwidth communication across a discrete channel is possible. In Chapter 6 we study how agents can learn to discover communication protocols in order to solve tasks when a discrete cheap-talk channel is available. Cheap-talk here means that the messages have no direct impact on the transition probabilities of the environment or the reward. In particular, we propose two methods, *Differentiable Inter-Agent Learning* (DIAL) and *Reinforced Inter-Agent Learning* (RIAL). During centralised training DIAL models the discrete messages as a continuous activation and passes gradients between different agents in order to learn what messages to send. By contrast, RIAL treats the messages as part of the action space and learns which messages to send using RL. In Chapter 7 we present the *Bayesian Action Decoder* (BAD), which is an extension of RIAL that introduces a public-Belief MDP. In contrast to DIAL, BAD can operate in settings without cheap talk channels. In particular, it allows agents to communicate through their environment actions when they are observed by other agents. By directly allowing agents to reason over the belief of other agents, BAD achieves a new state-of-the-art on the cooperative, partial information card game, Hanabi. The main challenge of Hanabi is to find efficient protocols that allow the players to communicate with each other through their actions.

This part of the thesis is based on the following publications and pre-prints:

- 'Learning to communicate with deep multi-agent reinforcement learning', Jakob N Foerster*, Iannis A Assael*, Nando de Freitas, Shimon Whiteson, *Advances in Neural Information Processing Systems*, 2016

- 'Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning', Jakob N. Foerster*, H. Francis Song*, Edward Hughes, Neil Burch, Iain Dunning, Shimon Whiteson, Matthew M. Botvinick, Mike Bowling, *International Conference on Machine Learning*, 2019

## Part 3: Learning to Reciprocate

All of the methods proposed so far assume a fully cooperative setting in which agents learn to cooperate and coordinate in order to maximise a team reward. However, in many real world problems agents aim to maximise diverse, individual rewards, potentially leading to a conflict of interest between different agents. For example, each driver typically wants to reach their destination as soon as possible, rather than to maximise the overall efficiency of the traffic. Game theory has a long history of studying optimal strategies in these settings. Here the core concept is a Nash equilibrium, which is achieved when none of the agents can improve their return through a unilateral change in policy. However, game theory commonly assumes that all Nash equilibria are known and can be computed, which is generally not the case in RL settings. In particular, in MARL the agents have to rely on interaction with the environment in order to learn any policy in the first place. While NL has a surprisingly strong track record in fully cooperative settings, in general sum settings issues can arise. First of all, all agents maximising their own objective can lead to unstable learning behaviour. Secondly, agents can fail to reciprocate, leading to convergence at Nash equilibria in which all agents are worse off. Both of these issues are due to the fact that the other agents are treated as a *static* part of the environment. *Learning with Opponent-Learning Awareness* (LOLA) aims to overcome these issues and to allow the agents to converge to Nash

equilibria with high returns. Rather than assuming that other agents are static, each agent instead assumes others are naive learners and optimises the expected return after one step of opponent-learning. Importantly, the agent can differentiate through the learning step of the opponent, leading to a shaping of their policy. LOLA agents manage to discover the famous *tit-for-tat* strategy in the iterated prisoner's dilemma. One technical difficulty is that differentiating through the learning step of an agent produces a higher order derivative which needs to be estimated using samples from the environment. In Chapter 9 we introduce *DiCE: The Infinitely Differentiable Monte-Carlo Estimator*(DiCE), a new way to estimate higher order gradients in stochastic computation graphs.

This part is based on the following publications:

- 'Learning with Opponent-Learning Awareness', Jakob N Foerster*, Richard Y Chen*, Maruan Al-Shedivat, Shimon Whiteson, Pieter Abbeel, Igor Mordatch, *International Conference on Autonomous Agents and Multiagent Systems*, 2018

- 'DiCE: The Infinitely Differentiable Monte-Carlo Estimator', Jakob N Foerster, Gregory Farquhar*, Maruan Al-Shedivat*, Tim Rocktäschel, Eric P Xing, Shimon Whiteson, *International Conference on Machine Learning*, 2018

# 2
# Background

## Contents



In this chapter we provide the necessary background information and formalisms which are common requirements for the rest of the thesis. In particular we introduce Reinforcement Learning, Section 2.1, the multi-agent setting, Section 2.2, value functions, Section 2.7, deep $Q$-Learning, Section 2.10, and actor-critic learning, Section 2.11. Concepts which are required only for specific chapters are introduced as additional background in those chapters, see, *e.g.*, stochastic computation graphs in Chapter 9. Furthermore, when appropriate, we revise some of the key concepts within the given chapters. Content in this chapter is based on all relevant papers and preprints mentioned in the introduction.

## 2.1 Reinforcement Learning

In Reinforcement Learning (RL) an agent interacts sequentially with an environment, $E$. The environment is characterised by the transition function, $P(s_{t+1}|s_t, u_t)$, the reward function $R(s_t, u_t)$, action space $U$, and discount factor, $\gamma$. The transition function, $P(s_{t+1}|s_t, u_t)\colon S \times U \times S \to \mathbb{R}$, specifies the probability distribution of the next state, $s_{t+1}$, given a current state, $s_t$, and action, $u_t$. Here $S$ is the state-space and $U$ is the action space. While some of the methods proposed in this thesis can be extended to continuous action spaces, we focus on discrete action spaces throughout. Since the probability of the next state conditions only on the last state and action, this is a Markov-process and RL is commonly framed as a Markov Decision Process (MDP). The, potentially stochastic, reward function, $R(s_t, u_t)\colon S \times U \to \mathbb{R}$, maps from state-action pairs, $\{s_t, u_t\}$, into rewards. In fully observable settings the agent observes the Markov state of the environment, $s_t$, and chooses an action $u_t \in U$ from their policy $\pi(u_t|s_t)\colon S \times U \to [0, 1]$. The environment then transitions to a next state, $s_{t+1} \sim P(s_{t+1}|s_t, u_t)$, and provides a reward, $r_t \sim R(s_t, u_t)$, to the agent. A standard assumption is that both the transition function and reward function are unknown to the agent, but instead have to be discovered by interacting with the environment. The task also contains a discount factor, $\gamma$, which specifies the relative importance of future rewards. The goal of the agent is to update the policy in order to maximise the total expected discounted return per episode $J = \mathbb{E}_{\tau \sim P(\tau)} R_0(\tau)$, where $\tau$ is the trajectory: $\tau = \{s_0, u_0, r_0, ...s_{T+1}\}$ and $R_t(\tau) = \sum_{t'=t} \gamma^{t'-t} r_{t'}$ is the forward-looking return from time step $t$ onwards. Using the Markov assumption we can express the probability of the trajectory as a multiplication of probabilities across time steps: $P(\tau) = P(s_0) \prod_{t=0}^{t=T} P(r_t|s_{t+1}, s_t, u_t) P(u_t|s_t) P(s_{t+1}|s_t, u_t)$. In this expression we recognise both the transition function, $P(s_{t+1}|s_t, u_t)$ and the policy $P(u_t|s_t) = \pi(u_t|s_t)$.

## 2.2 Multi-Agent Settings

Across this thesis we will be considering multi-agent settings. One way to formalise them is as a *stochastic game* [19], $G$, defined by a tuple $G = \langle S, U, P, r, Z, O, n, \gamma \rangle$, in which $n$ agents identified by $a \in A \equiv \{1, ..., n\}$ choose sequential actions. As in the single agent setting the environment has a true state $s \in S$. At each time step, each agent takes an action $u^a \in U$, forming a joint action $\mathbf{u} \in \mathbf{U} \equiv U^n$ which induces a transition in the environment according to the state transition function $P(s'|s, \mathbf{u}) : S \times \mathbf{U} \times S \to \mathbb{R}$. Here $U$ is the action space, the reward function specifies an agent specific reward, $r(s, \mathbf{u}, a) : S \times \mathbf{U} \times A \to \mathbb{R}$ and, as before, $\gamma \in [0, 1]$ is a discount factor.

We denote joint quantities over agents in bold, *e.g.*, $\mathbf{u}$, and joint quantities over agents other than a given agent $a$ with the superscript $-a$, *e.g.*, $\mathbf{u}^{-a}$.

## 2.3 Centralised vs Decentralised Control

In the fully observable setting it is in principle possible to learn a centralised controller, $\pi^C(\mathbf{u}|s_t)$, that maps from states into a probability distribution over *joint actions*, $\mathbf{u}$:

$$\pi^C(\mathbf{u}|s_t) : \mathbf{U} \times S \to [0, 1]. \tag{2.3.1}$$

However, there are two fundamental challenges with this approach: First of all the joint action space, $\mathbf{U}$, is exponential in the number of agents and, secondly, in many real world settings agents have to act based on their local observations, $o^a$, making centralised control impossible.

Due to the two reasons above, this thesis focuses on *decentralised control*. In this setting each agent has a local policy, $\pi^a(u^a|s_t)$, which maps from states (or observations) to a probability distribution over actions for the given agent. We note, this factorises the probability distribution over the joint-action:

$$P(\mathbf{u}|s_t) = \prod_a \pi^a(u^a|s_t). \tag{2.3.2}$$

This resolves both of the challenges stated above: Rather than considering an exponential action space, each agent only needs to consider their own action space. Furthermore, in partially observable settings (see below) each of the policies can condition on the local observations of the agent.

## 2.4 Cooperative, Zero-sum, and General-Sum

We use *cooperative* to refer to settings in which all agents receive the same reward, making it a team-reward:

$$r(s, \mathbf{u}, a) = r(s, \mathbf{u}, a'), \forall a, a'. \tag{2.4.1}$$

The opposite is *zero-sum*, when the reward summed across agents is 0 across all states:

$$\sum_a r(s, \mathbf{u}, a) = 0, \forall s, \mathbf{u}. \tag{2.4.2}$$

In zero-sum settings one agent's gain is the loss of the other agents in the environment. A middle ground are *general-sum* settings. These include cases in which agents are neither fully cooperative nor entirely adversarial.

Depending on the setting, different challenges arise: For example, partially observable, cooperative settings are well suited for investigating methods that learn communication protocols. In contrast, in general-sum settings, such as the *iterated prisoner's dilemma* (see Chapter 8) agents may need to learn to reciprocate and in zero-sum settings they need to avoid the instabilities arising from different agents optimising opposing losses.

The reward structure of the problem and the number of agents also change the properties of and the relationship between the different *Nash equilibria*, as explained in Section 2.8.

## 2.5 Partial Observability

We also generally consider partially observable settings, in which agents draw observations $o_t^a \in Z$ according to an observation function $O(s, a) : S \times A \to Z$. In

this setting each agent has an action-observation history $\tau^a \in T \equiv (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(u^a|\tau^a) : T \times U \to [0, 1]$ which maps from action-observation histories into a probability distribution over actions. Typically agents will need to learn a sufficient statistic for the action-observation history $\tau^a$, *e.g.*, using function approximators such as recurrent neural networks. Note that throughout the thesis we do not consider the rewards as being directly observable to the agent, unless they are part of the observation function.

Furthermore, in the background section of Chapter 4 we introduce featurised state spaces and the formalism for *common knowledge*. In Chapter 7 we introduce a *Public-Belief-MDP* that directly tracks the distribution of private features using common knowledge. We also note that cooperative, partially observable multi-agent settings can be formalised as *decentralised partially observable Markov decision processes* (Dec-POMDPs), as introduced by Oliehoek, Spaan, and Vlassis [20].

## 2.6 Centralised Training, Decentralised Execution

In this thesis we focus on settings in which agents need to take actions based on their local observations during *decentralised execution*. However, during *centralised training* they may utilise extra information (such as access to the Markov state) or free communication between the agents, as long as the final policies do not rely on this information. This is commonly possible in real world settings, *e.g.*, when training is carried out on a simulator while the final policies are later deployed in the real world. *Centralised Training with Decentralised Execution* is also a standard approach for Dec-POMDP solving [20] and has recently seen application in DMARL, with a number of examples provided in this thesis.

We note that even in general-sum settings it is possible and meaningful to carry out centralised training, as long as the final policies do not require access to extra information, *e.g.*, the opponents' policies.

## 2.7 Value Functions

The discounted return for agent $a$ across an episode is $R_t^a(\boldsymbol{\tau}) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r_{t'}^a$. The agents' joint policy induces a value function, i.e., an expectation over $R_t^a$ given a current state, $s_t$:

$$V_{\boldsymbol{\pi}}^a(s_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t:\infty}} \left[ R_t^a | s_t \right], \tag{2.7.1}$$

and an action-value function, also known as $Q$-function:

$$Q_{\boldsymbol{\pi}}^a(s_t, \mathbf{u}_t) = \mathbb{E}_{s_{t+1:\infty}, \mathbf{u}_{t+1:\infty}} \left[ R_t^a | s_t, \mathbf{u}_t \right]. \tag{2.7.2}$$

It also induces an expected total return, which is typically the maximisation objective:

$$J^a(\boldsymbol{\pi}) = \mathbb{E}_{s_{0:\infty}, \mathbf{u}_{0:\infty}} \left[ R_t^a \right]. \tag{2.7.3}$$

Commonly, we will drop the dependence on $a$ for notational convenience, when it is unambiguous (*e.g.*, in fully cooperative settings), and write, *e.g.*, $Q^{\boldsymbol{\pi}}$ or $Q$ instead. As introduced before, cooperative here means that all agents are receiving the same team-reward. Using this convention the advantage function is given by:

$$A^{\boldsymbol{\pi}}(s_t, \mathbf{u}_t) = Q^{\boldsymbol{\pi}}(s_t, \mathbf{u}_t) - V^{\boldsymbol{\pi}}(s_t). \tag{2.7.4}$$

This is called the advantage function since it measures the difference between the expected return given both the state and the action, compared to the expected return of just being in the state, $s_t$. In other words it measure the increase (or decrease) in expected return due to the agents having chosen action $\mathbf{u}_t$.

So far these value functions are presented as mathematical entities and definitions. In Section 2.10 and Section 2.11 we introduce methods for training value functions that are parameterised as deep neural networks, using samples generated via interaction with the environment. Throughout this thesis we are using model-free methods, building on the success of deep RL in the last few years. For more background on RL, including model-based approaches and planning please see [21].

## 2.8 Nash Equilibria

When multiple agents are learning to maximise their own reward, it is unclear what the right metric for measuring success is, see, *e.g.*, Shoham, Powers, Grenager, et al. [22] for more details on this. For example, in zero-sum settings the overall reward summed across agents is always zero, independent of the policy.

One important concept here is the *Nash-equilibrium* [23]: A Nash equilibrium describes any set of policies $\boldsymbol{\pi}^*$, such that the return for each agent $a$ given the policies of other agents, $\boldsymbol{\pi}^{*-a}$, is maximised under $\pi^{*a}$:

$$\forall a : J^a(\pi^{*a}, \boldsymbol{\pi}^{*-a}) \geq J^a(\pi^a, \boldsymbol{\pi}^{*-a}), \forall \pi^a \tag{2.8.1}$$

Clearly, the goal of converging to Nash equilibria is a good starting point for multi-agent learning algorithms. However, there commonly can be multiple Nash equilibria with different payouts, leading to greater difficulty in evaluating multi-agent learning compared to single agent RL. It can also lead to agents converging to Nash equilibria in which every single agent is worse off than in an alternative equilibrium. We provide example failure cases in Chapter 8.

The existence of multiple Nash equilibria with different payouts can also lead to local minima in the learning dynamics of multi-agent problems: Whenever agents have converged to a Nash equilibrium, there is no incentive for either of the agents to change their strategy unilaterally, even though everyone might be able to improve their return under a coordinated switch. This holds even in cooperative scenarios, a setting which we expand on and propose a solution to in Chapter 4.

Even multiple equilibria with identical payouts can be problematic: When different agents are playing their component of different Nash strategies, all agents can end up receiving lower returns. This shows that, *in general*, any single agent playing their component of a Nash strategy does not provide any performance guarantees.

A special class of games are two-player zero-sum settings. In these settings different Nash-equilibria are *interchangeable*. Playing their component, $\pi^{*a}$, of any

Nash profile, $\boldsymbol{\pi}^*$, is a *safe option* for agent $a$, independently of what strategy, $\boldsymbol{\pi}^{-a}$, the opponent is playing. Agent $a$ cannot do worse than the Nash payout:

$$J^a(\pi^{*a}, \boldsymbol{\pi}^{*-a}) \geq J^a(\pi^*, \boldsymbol{\pi}^{-a}), \forall \boldsymbol{\pi}^{-a}. \tag{2.8.2}$$

While in game theory it is common to assume that Nash equilibria are known or can be solved for, in MARL we typically investigate settings which are too complex for solving them exactly. Instead, policies have to be learned based on repeated interaction with the environment.

Another important concept is the *best response*: Given a set of policies $\boldsymbol{\pi}^{-a}$, the best response to $\boldsymbol{\pi}^{-a}$ is the policy $\hat{\pi}^a(\boldsymbol{\pi}^{-a})$ that maximises $J^a(\pi^a, \boldsymbol{\pi}^{-a})$:

$$\hat{\pi}^a(\boldsymbol{\pi}^{-a}) = \underset{\pi^a}{\operatorname{argmax}}\, J^a(\pi^a, \boldsymbol{\pi}^{-a}). \tag{2.8.3}$$

## 2.9 Deep Learning for MARL

Throughout this thesis we will be using state-of-the-art deep neural networks (NN) to represent both value functions and policies. For all intents and purposes we can think of these as function approximators with a large number of free parameters, typically indicated by $\theta$. Importantly, NNs can be trained efficiently using backpropagation and stochastic gradient descent on mini-batches (*i.e.*, small sets of examples). For more details on the burgeoning field of Deep Learning please see the recent "Deep Learning" book by Goodfellow et al. [24]. We recommend in particular Chapter 6 ("Deep Feedforward Networks") and Chapter 10 ("Sequence Modeling: Recurrent and Recursive Nets"), since both feedforward and recurrent neural networks are used extensively throughout the thesis.

There are a few properties of NNs that we exploit in the context of DMARL: In cooperative settings we commonly use parameter sharing across different agents, in which case the policies or value functions of all agents are parameterised by the same weights, $\theta$. Parameter sharing has the great advantage that the computational cost can be reduced by 'batching' the computation across different agents, *i.e.*, computing policies (or values) for all agents in parallel, and that it improves sample efficiency by aggregating experiences across different agents.

In settings where we do not share parameters, $\theta^a$ are the weights of the policy of agent $a$. We furthermore use superscripts, $\theta^\pi$and $\theta^C$, to distinguish between weights for the policy and the value function / critic when this is ambiguous.

Another property of deep learning that we use commonly in this thesis is that neural networks can be differentiated efficiently using the aforementioned backpropagation algorithm. We use this, for example, in Chapter 6, where agents learn to communicate by differentiating through a communication channel during training. In Chapter 9 we use the auto-differentiation mechanism in order to construct gradient estimators.

## 2.10 Q-Learning and DQN

While the $Q$-function defined above can in principle be evaluated for any arbitrary policy $\pi$, the optimal action-value function $Q^*(s,u) = \max_\pi Q^\pi(s,u)$ obeys the Bellman optimality equation:

$$Q^*(s,u) = \mathbb{E}_{s'}\left[r + \gamma \max_{u'} Q^*(s',u') \mid s,u\right]. \tag{2.10.1}$$

This is the identity underlying $Q$-learning. In deep $Q$-learning (DQN), [13], the $Q$-function is represented by a deep neural network parameterised by $\theta$, $Q(s,u;\theta)$. DQNs can be optimised by minimising the Bellman error, using the recursion above:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,u,r,s'}[(y_i^{DQN} - Q(s,u;\theta_i))^2], \tag{2.10.2}$$

at each iteration $i$, with target $y_i^{DQN} = r + \gamma \max_{u'} Q(s',u';\theta_i^-)$. Here, $\theta_i^-$ are the parameters of a target network that is kept constant for a number of iterations while updating the online network $Q(s,u;\theta_i)$ by gradient descent. The action $u$ is chosen from $Q(s,u;\theta_i)$ by an *action selector*, which typically implements an $\epsilon$-greedy policy that selects the action that maximises the Q-value with a probability of $1-\epsilon$ and chooses randomly with a probability of $\epsilon$ in order to ensure that agents keep exploring the environment. In order to stabilise learning and to improve sample efficiency DQN also uses *experience replay*: during learning, the agent builds a dataset of episodic experiences and then trains by sampling mini-batches of

experiences. Maintaining the experience replay memory also prevents the network from overfitting to recent experiences.

Next we discuss standard adaptations of DQN to both multi-agent and partially observable settings:

**Independent $Q$-learning.** Tan [25] introduced *Independent Q-learning* (IQL), an extension of $Q$-learning to multi-agent settings, in which each agent $a$ learns an independent $Q$-function, $Q^a$. Importantly, $Q^a$ only conditions on the action, $u^a$ of agent $a$, rather than the joint action, $\mathbf{u}$.

Tampuu et al. [26] address this setting with a framework that combines DQN with IQL: In their work, each agent $a$ independently and simultaneously learns its own $Q$-network, $Q^a(s, u^a; \theta_i^a)$. While independent $Q$-learning can in principle lead to convergence problems (since one agent's learning makes the environment appear nonstationary to other agents), it has a strong empirical track record [17, 27], and was successfully applied to two-player pong. Note that in IQL each agent independently estimates the total return for the episode given their action and the state.

**Deep Recurrent Q-Networks.** Both DQN and IQL assume full observability, *i.e.*, the agent receives the Markov state of the environment, $s_t$, as input. By contrast, in the partially observable environments we consider, $s_t$ is hidden and the agent receives only an observation $o_t^a$ that is correlated with $s_t$, but in general does not disambiguate it.

Hausknecht and Stone [28], propose *deep recurrent Q-networks* (DRQN) to address single-agent, partially observable settings. Instead of approximating $Q(s, u)$ with a feedforward network, they approximate $Q(\tau, u)$ with a recurrent neural network that can maintain an internal state and aggregate observations over time, learning a sufficient statistic for the action-observation history $\tau$. This can be achieved by adding the hidden state of the network as an extra input, $h_{t-1}$, and output $h_t$. During training gradients are back-propagated in time through the unrolled hidden states, $h_t$.

## 2.11 Reinforce and Actor-Critic

Next, we provide some background on policy gradient methods [29]. Unlike $Q$-learning (introduced above) which learns a $Q$-function and then obtains a policy by maximising the $Q$-function in a given state, these methods directly optimise an agent's policy, parameterised by $\theta^\pi$, by performing gradient ascent on an estimate of the expected discounted total reward $J = \mathbb{E}_\pi[R_t]$. Perhaps the simplest form of policy gradient is REINFORCE [30], in which the gradient is:

$$g = \mathbb{E}_{s_{0:\infty}, u_{0:\infty}}\left[\sum_{t=0}^{T} R_t \nabla_{\theta^\pi} \log \pi(u_t|s_t)\right]. \tag{2.11.1}$$

In Chapter 9 we introduce the framework of stochastic computation graphs [31] and provide further background on gradient estimation in these settings.

In *actor-critic* (AC) approaches [32–35], the *actor*, *i.e.*, the policy, is trained by following a gradient that depends on a *critic*, which usually estimates a value function. In particular, $R_t$ is replaced by any expression equivalent to $Q(s_t, u_t) - b(s_t)$, where $b(s_t)$ is a baseline designed to reduce variance [36]. A common choice is $b(s_t) = V(s_t)$, in which case $R_t$ is replaced by the advantage, $A(s_t, u_t)$. Another option is to replace $R_t$ with the *temporal difference* (TD) error $r_t + \gamma V(s_{t+1}) - V(s)$, which is an unbiased estimate of $A(s_t, u_t)$. In practice, the gradient must be estimated from trajectories sampled from the environment, and the (action-)value functions must be estimated with function approximators. Consequently, the bias and variance of the gradient estimate depends strongly on the exact choice of estimator [37].

In this thesis, critics for policy evaluation, $y^c(\cdot, \theta^c)$, are typically trained using a variant of TD($\lambda$) [12] adapted for use with deep neural networks. TD($\lambda$) uses a mixture of $n$-step returns $G_t^{(n)} = \sum_{l=1}^{n} \gamma^{l-1} r_{t+l} + \gamma^n y^c(\cdot_{t+n}, \theta^c)$. In particular, the critic parameters $\theta^c$ are updated to minimise the following loss:

$$\mathcal{L}_t(\theta^c) = (y^{(\lambda)} - y^c(\cdot_t, \theta^c))^2, \tag{2.11.2}$$

where $y^{(\lambda)} = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}$, and the $n$-step returns $G_t^{(n)}$ are calculated using a *target network* [13] with parameters copied periodically from $\theta^c$.

Below we list two adaptations of actor-critic algorithms to the partially observable and multi-agent setting respectively.

**Partially observable Actor-Critic.** In the most direct adaptation of AC to the partially observable setting both the policy and the value functions condition on the action-observation history $\tau$ instead of the true state, $s$. This can be achieved by using a recurrent neural network, *e.g.*, a GRU [38], that learns a hidden presentation, $h_t$. Like in DRQN, this $h_t$ can in principle learn a sufficient statistic for the action-observation history.

**Independent Actor-Critic.** In Independent Actor-Critic (IAC) each agent estimates the advantage using their local action-observation history, which is also used as an input to the policy. This is commonly achieved by sharing the parameters between the policy network and the value function. In IAC other agents and their policies are treated as a static part of the environment. In Chapter 3 we develop novel actor-critic methods in multi-agent settings.

Both IAC and IQL are instances of multi-agent methods in which each agent maximises their own return, treating other agents as a *static* part of the environment. We refer to these methods collectively as *naive learning* (NL). In this thesis they commonly serve as benchmarks for testing our algorithmic innovations.

This concludes the common background section, covering the most important common concepts for understanding the rest of the thesis. Further required background is introduced within the different chapters as needed.

# Part I

# Learning to Collaborate

# Abstract

In this part of the thesis we focus on methods that allow agents to learn to collaborate in cooperative, partially observable multi-agent systems. All methods are applied to a multi-agent version of StarCraft micromanagement. In this formulation of the problem each unit corresponds to a learning agent that needs to select an action based on local observations. In Chapter 3 we address *multi-agent credit assignment* using a centralised, *counterfactual baseline*. In Chapter 4 we propose a *common knowledge* based learning algorithm that allows agents to learn a centralised policy which can be executed in a decentralised fashion. While the first two chapters in this part are based on actor-critic algorithms, in Chapter 5 we address the nonstationarity that arises when using DQN in a multi-agent context.

# 3

# Counterfactual Multi-Agent Policy Gradients

## Contents



## 3.1 Introduction

One of the great challenges of multi-agent learning is *multi-agent credit assignment* [39]: in cooperative settings, joint actions typically generate only global rewards, making it difficult for each agent to deduce its own contribution to the team's success. In some settings it is possible to design individual reward functions for each agent in order to reduce the severity of this problem. However, these rewards are not generally available in cooperative settings and often fail to encourage individual agents to sacrifice for the greater good.

In this chapter, we propose a new multi-agent RL method called *counterfactual multi-agent* (COMA) policy gradients, in order to address these issues. COMA takes an *actor-critic* [37] approach, in which the *actor*, i.e., the policy, is trained by following a gradient estimated by a *critic*. COMA is based on three main ideas.

First, COMA uses a centralised critic. The critic is only used during learning, while only the actor is needed during execution. Since learning is centralised, we can therefore use a centralised critic that conditions on the joint action and all available state information, while each agent's policy conditions only on its own action-observation history.

Second, COMA uses a *counterfactual baseline*. The idea is inspired by *difference rewards* [18], in which each agent learns from a shaped reward that compares the global reward to the reward received when that agent's action is replaced with a *default action*. While difference rewards are a powerful way to perform multi-agent credit assignment, they typically require access to a simulator and, in many applications, it is unclear how to choose the default action. COMA addresses this by using the centralised critic to compute an *advantage function* that compares the value for the current action to a counterfactual baseline that marginalises out a single agent's action, while keeping the other agents' actions fixed. Hence, instead of relying on extra simulations, COMA computes a separate baseline for each agent that relies on the centralised critic to reason about counterfactuals in which only that agent's action changes.

Third, COMA uses a critic representation that allows the counterfactual baseline to be computed efficiently. In a single forward pass, it computes the $Q$-values for all the different actions of a given agent, conditioned on the actions of all the other agents.

We evaluate COMA in the testbed of *StarCraft unit micromanagement*[1], which has recently emerged as a challenging RL benchmark task with high stochasticity, a large state-action space, and delayed rewards. Previous works [40, 41] have made use of a centralised control policy that conditions on the entire state and can use powerful

[1]StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment™.

macro-actions, using StarCraft's built-in planner, that combine movement and attack actions. To produce a meaningfully decentralised benchmark that proves challenging for scenarios with even relatively few agents, we propose a variant that massively reduces each agent's field-of-view and removes access to these macro-actions.

Our empirical results on this new benchmark show that COMA can significantly improve performance over other multi-agent actor-critic methods, as well as ablated versions of COMA itself. In addition, COMA's best agents are competitive with state-of-the-art centralised controllers which are given access to full state information and macro-actions.

## 3.2 Related Work

Although multi-agent RL has been applied in a variety of settings [42, 43], it has often been restricted to tabular methods and simple environments. One exception is foundational work in DMARL, which can scale to high dimensional input and action spaces. Tampuu et al. [26] use a combination of DQN with independent $Q$-learning [17, 25, 27] to learn how to play two-player pong. More recently the same method has been used by Leibo et al. [44] to study the emergence of collaboration and defection in sequential social dilemmas.

Also related is work on the emergence of communication between agents, learned using multi-agent RL [45–48], also see Chapter 6 of the thesis. In this line of work, passing gradients between agents during training and sharing parameters are two common ways to take advantage of centralised training. However, these methods do not allow for extra state information to be used during learning and do not address the multi-agent credit assignment problem.

Gupta, Egorov, and Kochenderfer [49] investigate actor-critic methods for decentralised execution with centralised training. However, in their methods both the actors and the critic condition on local, per-agent, observations and actions, and multi-agent credit assignment is addressed only with hand-crafted local rewards.

Most previous applications of RL to StarCraft micromanagement use a centralised controller, with access to the full state, and control of all units, although the

architecture of the controllers exploits the multi-agent nature of the problem. Usunier et al. [40] use a *greedy MDP*, which at each timestep sequentially chooses actions for agents given all previous actions, in combination with zero-order optimisation, while Peng et al. [41] use an actor-critic method that relies on RNNs to exchange information between the agents. Since Usunier et al. [40] address similar scenarios to our experiments and implement a DQN baseline in a fully observable setting, in Section 3.5 we report our competitive performance against these state-of-the-art baselines, while maintaining decentralised control. Omidshafiei et al. [50] also address the stability of experience replay in multi-agent settings, but assume a fully decentralised training regime.

Rashid et al. [51] and Sunehag et al. [52] propose learning a centralised value function that factors into per-agent components to allow for decentralisation. Lowe et al. [53] propose a single centralised critic that conditions on all available information during training and is used to train decentralised actors, this work was done concurrently with the work presented here. None of these approaches explicitly address the question of multi-agent credit assignment.

[53] concurrently propose a multi-agent policy-gradient algorithm using centralised critics. Their approach does not address multi-agent credit assignment. Unlike our work, it learns a separate centralised critic for each agent and is applied to competitive environments with continuous action spaces.

Our work builds directly off of the idea of *difference rewards* [18]. The relationship of COMA to this line of work is discussed in Section 6.4.

## 3.3 Multi-Agent StarCraft Micromanagement

In this section, we describe our multi-agent variant of the StarCraft micromanagement problem as well as details of the state features. This problem setting is the common testbed for all methods in this part of the thesis.

**Decentralised StarCraft Micromanagement.** StarCraft is a rich environment with stochastic dynamics that cannot be easily emulated. Many simpler multi-agent settings, such as Predator-Prey [25] or Packet World [54], by contrast,

have full simulators with controlled randomness that can be freely set to any state in order to perfectly replay experiences. This makes it possible, though computationally expensive, to compute difference rewards via extra simulations. In StarCraft, as in the real world, this is not possible.

In this and the next two chapters, we focus on the problem of *micromanagement* in StarCraft, which refers to the low-level control of individual units' positioning and attack commands as they fight enemies. This task is naturally represented as a multi-agent system, where each StarCraft unit is replaced by a decentralised controller. We consider several scenarios with symmetric teams, formed of: 3 marines (3m), 5 marines (5m), 5 wraiths (5w), or 2 dragoons with 3 zealots (2d_3z). The enemy team is controlled by the StarCraft AI, which uses a set of reasonable but suboptimal hand-crafted heuristics.

We allow the agents to choose from a set of discrete actions: `move[direction]`, `attack[enemy_id]`, `stop`, and `noop`. In the StarCraft game, when a unit selects an attack action, it first moves into attack range before firing, using the game's built-in pathfinding to choose a route. These powerful *attack-move* macro-actions make the control problem considerably easier.

To create a more challenging benchmark that is meaningfully decentralised, we impose a restricted field of view on the agents, equal to the firing range of the weapons of the ranged units, shown in Figure 3.1. This departure from the standard setup for centralised StarCraft control has three effects.

**Figure 3.1:** Starting position with example local field of view for the 2d_3z map.

First, it introduces significant partial observability. Second, it means units can only attack when they are in range of enemies, removing access to the StarCraft macro-actions. Third, agents cannot distinguish between enemies who are dead and those who are out of range and so can issue invalid attack commands at such enemies, which results in no action being taken. This substantially

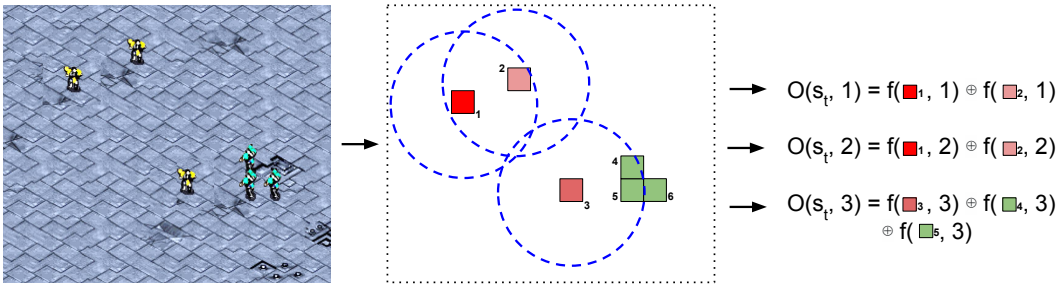



**Figure 3.2:** An example of the observations obtained by all agents at each time step $t$. The function f provides a set of features for each unit in the agent's field of view, which are concatenated. The feature set is {`distance`, `relative x`, `relative y`, `health points`, `weapon cooldown`}. Each quantity is normalised by its maximum possible value.

increases the average size of the action space, which in turn increases the difficulty of both exploration and control.

Under these difficult conditions, scenarios with even relatively small numbers of units become much harder to solve. As seen in Table 3.1, we compare against a simple hand-coded heuristic that instructs the agents to run forwards into range and then focus their fire, attacking each enemy in turn until it dies. This heuristic achieves a 98% win rate on m5v5 with a full field of view, but only 66% in our setting. To perform well in this task, the agents must learn to cooperate by positioning properly and focussing their fire, while remembering which enemy and ally units are alive or out of view.

All agents receive the same global reward at each time step, equal to the sum of damage inflicted on the opponent units minus half the damage taken. Killing an opponent generates a reward of 10 points, and winning the game generates a reward equal to the team's remaining total health plus 200. This damage-based reward signal is comparable to that used by Usunier et al. [40]. Unlike [41], our approach does not require estimating local rewards.

**State Features.** The actor and critic receive different input features, corresponding to local observations and global state, respectively. Both include features for allies and enemies. *Units* can be either allies or enemies, while *agents* are the decentralised controllers that command ally units.

The local observations for every agent are drawn only from a circular subset of the map centred on the unit it controls and include for each unit within this field of view: `distance`, `relative x`, `relative y`, `unit type` and `shield`.[2] All features are normalized by their maximum values. We do not include any information about the units' current target.

The global state representation consists of similar features, but for all units on the map regardless of fields of view. Absolute distance is not included, and $x$-$y$ locations are given relative to the centre of the map rather than to a particular agent. The global state also includes `health points` and `cooldown` for all agents. The representation fed to the centralised $Q$-function critic is the concatenation of the global state representation with the local observation of the agent whose actions are being evaluated. Our centralised critic that estimates $V(s)$, and is therefore agent-agnostic, receives the global state concatenated with all agents' observations. The observations contain no new information but include the egocentric distances relative to that agent.

## 3.4 Methods

In this section, we describe approaches for extending policy gradients to our multi-agent setting.

### 3.4.1 Independent Actor-Critic

The simplest way to apply policy gradients to multiple agents is to have each agent learn independently, with its own actor and critic, from its own action-observation history. This is essentially the idea behind *independent Q-learning* [25], which is perhaps the most popular multi-agent learning algorithm, but with actor-critic in place of $Q$-learning. Hence, we call this approach *independent actor-critic* (IAC).

In our implementation of IAC, we speed learning by sharing parameters among the agents, i.e., we learn only one actor and one critic, which are used by all agents.

[2] After firing, a unit's `cooldown` is reset, and it must drop before firing again. Shields absorb damage until they break, after which units start losing health. Dragoons and zealots have shields but marines do not.

The agents can still behave differently because they receive different observations, including an agent-specific ID, and thus evolve different hidden states. Learning remains independent in the sense that each agent's critic estimates only a local value function, i.e., one that conditions on $u^a$, not $\mathbf{u}$. Though we are not aware of previous applications of this specific algorithm, we do not consider it a significant contribution but instead merely a baseline algorithm.

We consider two variants of IAC. In the first, each agent's critic estimates $V(\tau^a)$ and follows a gradient based on the TD error, as described in Section 7.2. In the second, each agent's critic estimates $Q(\tau^a, u^a)$ and follows a gradient based on the advantage: $A(\tau^a, u^a) = Q(\tau^a, u^a) - V(\tau^a)$, where $V(\tau^a) = \sum_{u^a} \pi(u^a|\tau^a)Q(\tau^a, u^a)$. Independent learning is straightforward, but the lack of information sharing at training time makes it difficult to learn coordinated strategies that depend on interactions between multiple agents, or for an individual agent to estimate the contribution of its actions to the team's reward.

### 3.4.2 Counterfactual Multi-Agent Policy Gradients

The difficulties discussed above arise because, beyond parameter sharing, IAC fails to exploit the fact that learning is centralised in our setting. In this section, we propose *counterfactual multi-agent* (COMA) policy gradients, which overcome this limitation. Three main ideas underly COMA: 1) centralisation of the critic, 2) use of a counterfactual baseline, and 3) use of a critic representation that allows efficient evaluation of the baseline. The remainder of this section describes these ideas.

First, COMA uses a centralised critic. Note that in IAC, each actor $\pi(u^a|\tau^a)$ and each critic $Q(\tau^a, u^a)$ or $V(\tau^a)$ conditions only on the agent's own action-observation history $\tau^a$. However, the critic is used only during learning and only the actor is needed during execution. Since learning is centralised, we can therefore use a centralised critic that conditions on the true global state $s$, if it is available, or the joint action-observation histories $\boldsymbol{\tau}$ otherwise. Each actor conditions on its own action-observation histories $\tau^a$, with parameter sharing, as in IAC. Figure 5.1a illustrates this setup.

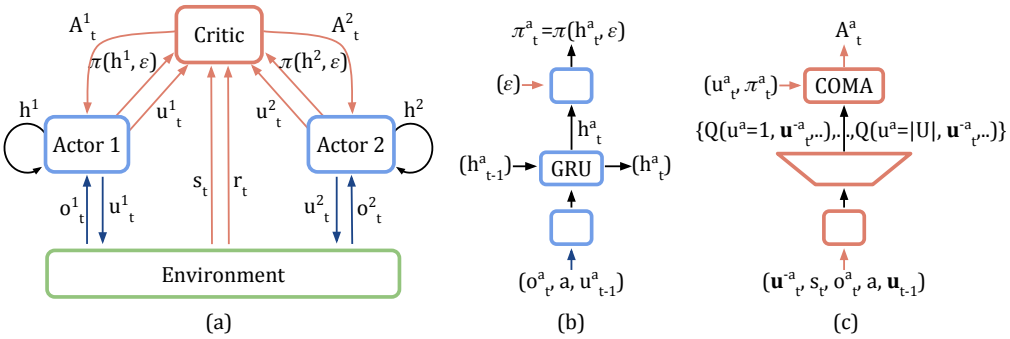



**Figure 3.3:** In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

A naive way to use this centralised critic would be for each actor to follow a gradient based on the TD error estimated from this critic:

$$g = \nabla_{\theta^\pi} \log \pi(u|\tau_t^a) \left(r + \gamma V(s_{t+1}) - V(s_t)\right). \tag{3.4.1}$$

However, such an approach fails to address a key credit assignment problem. Because the TD error considers only global rewards, the gradient computed for each actor does not explicitly reason about how that particular agent's actions contribute to that global reward. Since the other agents may be exploring, the gradient for that agent becomes very noisy, particularly when there are many agents.

Therefore, COMA uses a *counterfactual baseline*. The idea is inspired by *difference rewards* [55], in which each agent learns from a shaped reward $D^a = r(s, \mathbf{u}) - r(s, (\mathbf{u}^{-a}, c^a))$ that compares the global reward to the reward received when the action of agent $a$ is replaced with a *default action* $c^a$. Any action by agent $a$ that improves $D^a$ also improves the true global reward $r(s, \mathbf{u})$, because $r(s, (\mathbf{u}^{-a}, c^a))$ does not depend on agent $a$'s actions.

Difference rewards are a powerful way to perform multi-agent credit assignment. However, they typically require access to a simulator in order to estimate $r(s, (\mathbf{u}^{-a}, c^a))$. When a simulator is already being used for learning, difference rewards increase the number of simulations that must be conducted, since each agent's difference reward requires a separate counterfactual simulation. Proper and Tumer [56] and Colby, Curran, and Tumer [57] propose estimating difference

rewards using function approximation rather than a simulator. However, this still requires a user-specified default action $c^a$ that can be difficult to choose in many applications. In an actor-critic architecture, this approach would also introduce an additional source of approximation error.

A key insight underlying COMA is that a centralised critic can be used to implement difference rewards in a way that avoids these problems. COMA learns a centralised critic, $Q(s, \mathbf{u})$ that estimates $Q$-values for the joint action $\mathbf{u}$ conditioned on the central state $s$. For each agent $a$ we can then compute an advantage function that compares the $Q$-value for the current action $u^a$ to a counterfactual baseline that marginalises out $u^a$, while keeping the other agents' actions $\mathbf{u}^{-a}$ fixed:

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - \sum_{u'^a} \pi^a(u'^a|\tau^a) Q(s, (\mathbf{u}^{-a}, u'^a)). \tag{3.4.2}$$

Hence, $A^a(s, u^a)$ computes a separate baseline for each agent that uses the centralised critic to reason about counterfactuals in which only $a$'s action changes, learned directly from agents' experiences instead of relying on extra simulations, a reward model, or a user-designed default action.

This advantage has the same form as the *aristocrat utility* [55]. However, optimising for an aristocrat utility using value-based methods creates a self-consistency problem because the policy and utility function depend recursively on each other. As a result, prior work focused on difference evaluations using default states and actions. COMA is different because the counterfactual baseline's expected contribution to the gradient, as with other policy gradient baselines, is zero. Thus, while the baseline does depend on the policy, its expectation does not. Consequently, COMA can use this form of the advantage without creating a self-consistency problem.

While COMA's advantage function replaces potential extra simulations with evaluations of the critic, those evaluations may themselves be expensive if the critic is a deep neural network. Furthermore, in a typical representation, the number of output nodes of such a network would equal $|U|^n$, the size of the joint action space, making it impractical to train. To address both these issues, COMA uses a critic representation that allows for efficient evaluation of the baseline. In

particular, the actions of the other agents, $\mathbf{u}_t^{-a}$, are part of the input to the network, which outputs a $Q$-value for each of agent $a$'s actions, as shown in Figure 5.1c. Consequently, the counterfactual advantage can be calculated efficiently by a single forward pass of the actor and critic, for each agent. Furthermore, the number of outputs is only $|U|$ instead of $(|U|^n)$. While the network has a large input space that scales linearly in the number of agents and actions, deep neural networks can generalise well across such spaces.

In this paper, we focus on settings with discrete actions. However, COMA can be easily extended to continuous actions spaces by estimating the expectation in (3.4.2) with Monte Carlo samples or using functional forms that render it analytical, e.g., Gaussian policies and critic.

The following lemma establishes the convergence of COMA to a locally optimal policy. The proof follows directly from the convergence of single-agent actor-critic algorithms [29, 37], and is subject to the same assumptions.

**Lemma 3.4.1.** *For an actor-critic algorithm with a compatible TD(1) critic following a COMA policy gradient*

$$g_k = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_a \nabla_{\theta_k} \log \pi^a(u^a|\tau^a) A^a(s, \mathbf{u})\right] \tag{3.4.3}$$

*at each iteration $k$,*

$$\liminf_k ||\nabla J|| = 0 \quad w.p.\ 1. \tag{3.4.4}$$

*Proof.* The COMA gradient is given by

$$g = \mathbb{E}_{\boldsymbol{\pi}}\left[\sum_a \nabla_\theta \log \pi^a(u^a|\tau^a) A^a(s, \mathbf{u})\right], \tag{3.4.5}$$

$$A^a(s, \mathbf{u}) = Q(s, \mathbf{u}) - b(s, \mathbf{u}^{-a}), \tag{3.4.6}$$

where $\theta$ are the parameters of all actor policies, e.g. $\theta = \{\theta^1, \ldots, \theta^{|A|}\}$, and $b(s, \mathbf{u}^{-a})$ is the counterfactual baseline defined in equation 3.4.2.

First consider the expected contribution of the this baseline $b(s, \mathbf{u}^{-a})$:

$$g_b = -\mathbb{E}_{\boldsymbol{\pi}}\left[\sum_a \nabla_\theta \log \pi^a(u^a|\tau^a) b(s, \mathbf{u}^{-a})\right], \tag{3.4.7}$$

where the expectation $\mathbb{E}_{\boldsymbol{\pi}}$ is with respect to the state-action distribution induced by the joint policy $\boldsymbol{\pi}$. Now let $d^{\boldsymbol{\pi}}(s)$ be the discounted ergodic state distribution as defined by Sutton et al. [29]:

$$\begin{aligned} g_b &= -\sum_s d^{\boldsymbol{\pi}}(s) \sum_a \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}(\mathbf{u}^{-a}|\boldsymbol{\tau}-a) \cdot \\ &\quad \sum_{u^a} \pi^a(u^a|\tau^a) \nabla_\theta \log \pi^a(u^a|\tau^a) b(s, \mathbf{u}^{-a}) \qquad (3.4.8) \\ &= -\sum_s d^{\boldsymbol{\pi}}(s) \sum_a \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}(\mathbf{u}^{-a}|\boldsymbol{\tau}-a) \cdot \\ &\quad \sum_{u^a} \nabla_\theta \pi^a(u^a|\tau^a) b(s, \mathbf{u}^{-a}) \qquad (3.4.9) \\ &= -\sum_s d^{\boldsymbol{\pi}}(s) \sum_a \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}(\mathbf{u}^{-a}|\boldsymbol{\tau}-a) b(s, \mathbf{u}^{-a}) \nabla_\theta 1 \\ &= 0. \qquad (3.4.10) \end{aligned}$$

Clearly, the per-agent baseline, although it reduces variance, does not change the expected gradient, and therefore does not affect the convergence of COMA.

The remainder of the expected policy gradient is given by:

$$\begin{aligned} g &= \mathbb{E}_{\boldsymbol{\pi}} \left[ \sum_a \nabla_\theta \log \pi^a(u^a|\tau^a) Q(s, \mathbf{u}) \right] \qquad (3.4.11) \\ &= \mathbb{E}_{\boldsymbol{\pi}} \left[ \nabla_\theta \log \prod_a \pi^a(u^a|\tau^a) Q(s, \mathbf{u}) \right]. \qquad (3.4.12) \end{aligned}$$

Writing the joint policy as a product of the independent actors:

$$\boldsymbol{\pi}(\mathbf{u}|s) = \prod_a \pi^a(u^a|\tau^a), \qquad (3.4.13)$$

yields the standard single-agent actor-critic policy gradient:

$$g = \mathbb{E}_{\boldsymbol{\pi}} \left[ \nabla_\theta \log \boldsymbol{\pi}(\mathbf{u}|s) Q(s, \mathbf{u}) \right]. \qquad (3.4.14)$$

Konda and Tsitsiklis [37] prove that an actor-critic following this gradient converges to a local maximum of the expected return $J^{\boldsymbol{\pi}}$, given that:

1. the policy $\boldsymbol{\pi}$ is differentiable,

2. the update timescales for $Q$ and $\boldsymbol{\pi}$ are sufficiently slow, and that $\boldsymbol{\pi}$ is updated sufficiently slower than $Q$, and

3. $Q$ uses a representation compatible with $\boldsymbol{\pi}$,

amongst several further assumptions. The parameterisation of the policy (i.e., the single-agent joint-action learner is decomposed into independent actors) is immaterial to convergence, as long as it remains differentiable. Note however that COMA's centralised critic is essential for this proof to hold. □

Here is pseudo code for COMA:

**Algorithm 1** Counterfactual Multi-Agent (COMA) Policy Gradients

Initialise $\theta_1^c$, $\hat{\theta}_1^c, \theta^\pi$
**for** each training episode $e$ **do**
  Empty buffer
  **for** $e_c = 1$ **to** $\frac{\text{BatchSize}}{n}$ **do**
    $s_1$ = initial state, $t = 0$, $h_0^a = \mathbf{0}$ for each agent $a$
    **while** $s_t \neq$ terminal **and** $t < T$ **do**
      $t = t + 1$
      **for** each agent $a$ **do**
        $h_t^a = \text{Actor}\left(o_t^a, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i\right)$
        Sample $u_t^a$ from $\pi(h_t^a, \epsilon(e))$
      Get reward $r_t$ and next state $s_{t+1}$
    Add episode to buffer
  Collate episodes in buffer into single batch
  **for** $t = 1$ **to** $T$ **do** // from now processing all agents in parallel via single batch
    Batch unroll RNN using states, actions and rewards
    Calculate TD($\lambda$) targets $y_t^a$ using $\hat{\theta}_i^c$
  **for** $t = T$ **down to** 1 **do**
    $\Delta Q_t^a = y_t^a - Q\left(s_j^a, \mathbf{u}\right)$
    $\Delta\theta^c = \nabla_{\theta^c}(\Delta Q_t^a)^2$ // calculate critic gradient
    $\theta_{i+1}^c = \theta_i^c - \alpha\Delta\theta^c$ // update critic weights
    Every C steps reset $\hat{\theta}_i^c = \theta_i^c$
  **for** $t = T$ **down to** 1 **do**
    $A^a(s_t^a, \mathbf{u}) = Q(s_t^a, \mathbf{u}) - \sum_u Q(s_t^a, u, \mathbf{u}^{-a})\pi(u|h_t^a)$ // calculate COMA
    $\Delta\theta^\pi = \Delta\theta^\pi + \nabla_{\theta^\pi} \log \pi(u|h_t^a) A^a(s_t^a, \mathbf{u})$ // accumulate actor gradients
  $\theta_{i+1}^\pi = \theta_i^\pi + \alpha\Delta\theta^\pi$ // update actor weights

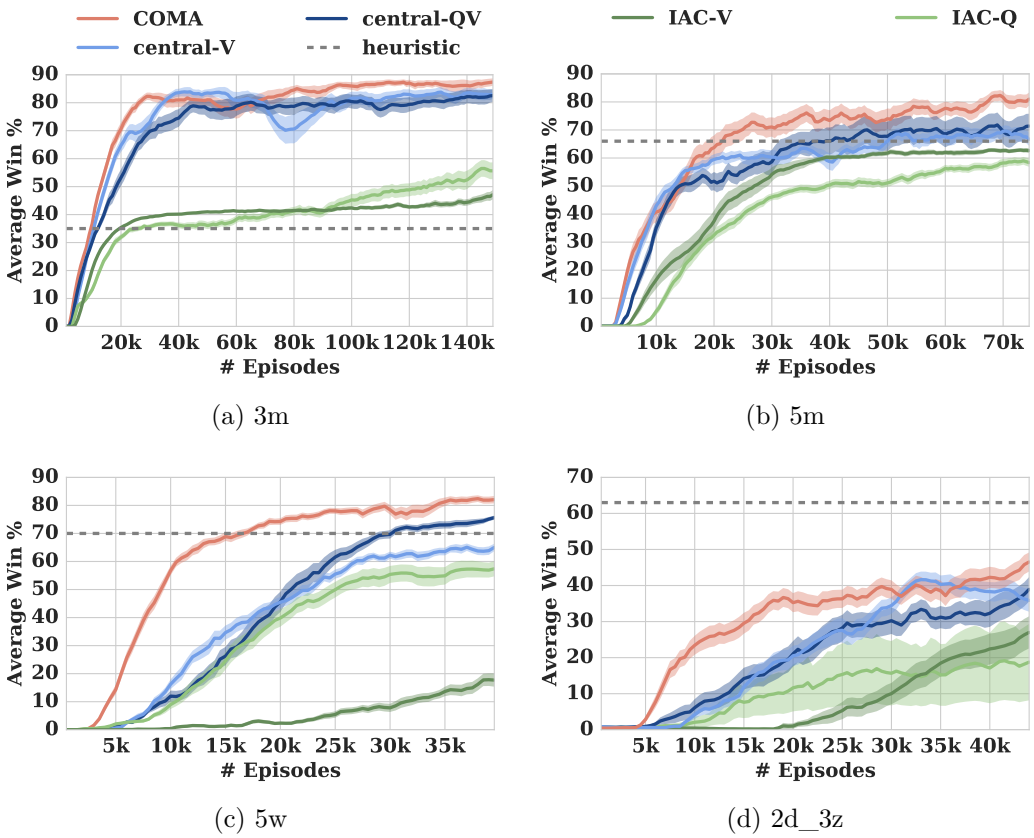
**Architecture & Training.** The actor consists of 128-bit *gated recurrent units* (GRUs) [38] that use fully connected layers both to process the input and to produce the output values from the hidden state, $h_t^a$. The IAC critics use extra output heads

appended to the last layer of the actor network. Action probabilities are produced from the final layer, $\mathbf{z}$, via a bounded softmax distribution that lower-bounds the probability of any given action by $\epsilon/|U|$: $P(u) = (1-\epsilon)\text{softmax}(\mathbf{z})_u + \epsilon/|U|$). We anneal $\epsilon$ linearly from 0.5 to 0.02 across 750 training episodes. The centralised critic is a feedforward network with multiple ReLU layers combined with fully connected layers. Hyperparameters were coarsely tuned on the m5v5 scenario and then used for all other maps. We found that the most sensitive parameter was TD($\lambda$), but settled on $\lambda = 0.8$ which worked best for both COMA and our baselines. Our implementation uses TorchCraft [58] and Torch 7 [59].

**Ablations.** We perform ablation experiments to validate three key elements of COMA. First, we test the importance of centralising the critic by comparing against two IAC variants, IAC-$Q$ and IAC-$V$. These take the same decentralised input as the actor, and share the actor network parameters up to the final layer. IAC-$Q$ then outputs $|U|$ $Q$-values, one for each action, while IAC-$V$ outputs a single state-value. Second, we test the significance of learning $Q$ instead of $V$. The method *central-V* still uses a central state for the critic, but learns $V(s)$, and uses the TD error to estimate the advantage for policy gradient updates. Third, we test the utility of our counterfactual baseline. The method *central-QV* learns both $Q$ and $V$ simultaneously and estimates the advantage as $Q - V$, replacing our counterfactual baseline with $V$. All methods use the same architecture and training scheme for the actors, and all critics are trained with TD($\lambda$).

| | Local Field of View (FoV) | | | | | | | Full FoV, Central Control | | |
|---|---|---|---|---|---|---|---|---|---|---|
| map | heur. | IAC-$V$ | IAC-$Q$ | cnt-$V$ | cnt-$QV$ | COMA mean | COMA best | heur. | DQN | GMEZO |
| 3m | .35 | .47 | .56 | .83 | .83 | **.87** | .98 | .74 | - | - |
| 5m | .66 | .63 | .58 | .67 | .71 | **.81** | .95 | .98 | .99 | 1. |
| 5w | .70 | .18 | .57 | .65 | .76 | **.82** | .98 | .82 | .70 | .74[3] |
| 2d_3z | **.63** | .27 | .19 | .36 | .39 | .47 | .65 | .68 | .61 | .90 |

**Table 3.1:** Mean win rates averaged across final 1000 evaluation episodes for the different maps, for all methods and the hand-coded heuristic in the decentralised setting with a limited field of view. The best result for this setting is in bold. Also shown, maximum win rates for COMA (decentralised), in comparison to the heuristic and published results (evaluated in the centralised setting).




(a) 3m (b) 5m

(c) 5w (d) 2d_3z

**Figure 3.4:** Win rates for COMA and competing algorithms on four different scenarios. COMA outperforms all baseline methods. Centralised critics also clearly outperform their decentralised counterparts. The legend at the top applies across all plots.

## 3.5 Results

Figure 3.4 shows average win rates as a function of episode for each method and each StarCraft scenario. For each method, we conducted 35 independent trials and froze learning every 100 training episodes to evaluate the learned policies across 200 episodes per method, plotting the average across episodes and trials. Also shown is one standard deviation in performance.

The results show that COMA is superior to the IAC baselines in all scenarios. Interestingly, the IAC methods also eventually learn reasonable policies in m5v5, although they need substantially more episodes to do so. This may seem counter-intuitive since in the IAC methods, the actor and critic networks share parameters

[3] 5w DQN and GMEZO benchmark performances are of a policy trained on a larger map and tested on 5w

in their early layers (see Section 3.4.2). This could be expected to speed learning, but these results suggest that the improved accuracy of policy evaluation made possible by conditioning on the global state outweighs the overhead of training a separate network.

Furthermore, COMA strictly dominates central-$QV$, both in training speed and in final performance across all settings. This is a strong indicator that our counterfactual baseline is crucial when using a central $Q$-critic to train decentralised policies.

Learning a state-value function has the obvious advantage of not conditioning on the joint action. Still, we find that COMA outperforms our baseline central-$V$ in final performance. Further, COMA typically achieves good policies faster, which is expected as COMA provides a shaped training signal. Training is also more stable than central-$V$, which is a consequence of the COMA gradient tending to zero as the policy becomes greedy. Overall, COMA is the best performing and most consistent method.

Usunier et al. [40] report the performance of their best agents trained with their state-of-the-art centralised controller labelled GMEZO (greedy-MDP with episodic zero-order optimisation), and for a centralised DQN controller, both given a full field of view and access to attack-move macro-actions. These results are compared in Table 3.1 against the best agents trained with COMA for each map. Clearly, in most settings these agents achieve performances comparable to the best published win rates despite being restricted to decentralised policies and a local field of view.

## 3.6 Conclusions & Future Work

In this chapter we presented COMA policy gradients, a method that uses a centralised critic in order to estimate a counterfactual advantage for decentralised policies in multi-agent RL. COMA addresses the challenges of multi-agent credit assignment by using a counterfactual baseline that marginalises out a single agent's action, while keeping the other agents' actions fixed. Our results in a decentralised *StarCraft unit micromanagement* benchmark show that COMA significantly improves final performance and training speed over other multi-agent

actor-critic methods and remains competitive with state-of-the-art centralised controllers under best-performance reporting. Future work will extend COMA to tackle scenarios with large numbers of agents, where centralised critics are more difficult to train and exploration is harder to coordinate. We also aim to develop more sample-efficient variants that are practical for real-world applications such as self-driving cars.

# 4

# Multi-Agent Common Knowledge Reinforcement Learning

## Contents



## 4.1 Introduction

In the previous chapter we addressed the question of credit assignment in cooperative multi-agent systems using a centralised value function which is only required during training. This is an example of a centralised training method with decentralised execution. While significant progress has been made in this direction [51, 60, 61], the requirement that policies must be fully decentralised severely limits the agents' ability to coordinate their behaviour. Often agents are forced to ignore information in

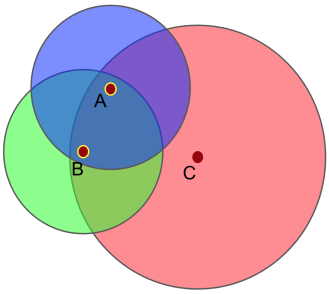



**Figure 4.1:** Three agents and their fields of view. A and B's locations are common knowledge to A and B as they are within each other's fields of view. However, even though C can see A and B, it shares no common knowledge with them.

their individual observations that would in principle be useful for maximising reward, because acting on it would make their behaviour less predictable to their teammates.

In this chapter, we propose *multi-agent common knowledge reinforcement learning* (MACKRL), which strikes a middle ground between these two extremes. The key insight is that, even in partially observable settings, subsets of the agents often possess some *common knowledge* that they can exploit to coordinate their behaviour. Common knowledge for a set 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" [62].

We formalise a multi-agent setting that suffices to give rise to common knowledge. The setting involves assumptions that, while restrictive, are naturally satisfied in a range of multi-agent problems. Intuitively, common knowledge can arise between two agents when each agent can observe the other, and doing so disambiguates what the other has observed. For example, if each agent can reliably observe things that are within its field of view and the agents know each other's fields of view, then two agents possess common knowledge whenever they see each other. Such a scenario, illustrated in Figure 4.1, arises in tasks such as robo-soccer [63] and multi-agent StarCraft Micromanagement [58]. It could also arise in a range of real-world scenarios, e.g. a self-driving car that knows which sensors other autonomous vehicles that it observes are equipped with.

Common knowledge is useful because, by definition, each agent in a group can independently deduce the common knowledge shared by that group. Consequently,

a centralised joint policy that conditions only on the common knowledge of all agents can be executed in a decentralised fashion: each agent simply queries the centralised policy for a joint action and then executes their part of that joint action. Since each agent supplies the same common knowledge as input, they return the same joint action, yielding coordinated behaviour.

However, the presence of common knowledge gives rise to a major challenge. Smaller groups of agents often possess more common knowledge than larger groups, making it unclear at what level agents should coordinate. On the one hand, coordination would ideally happen globally, that is, a fully centralised policy would select a joint action for all agents. However, the common knowledge shared among all agents may be weak, limiting what such a policy can condition on. On the other hand, the agents could be broken into subgroups that share more common knowledge within them. Coordination would no longer occur across the subgroups, but action selection within each group could condition on a richer common knowledge signal.

MACKRL addresses this challenge using a hierarchical approach. At each level of the hierarchy, a controller can either select a joint action for the agents in its subgroup, or propose a partition of the agents into smaller subgroups, whose actions are then selected by controllers at the next level of the hierarchy. During action selection, MACKRL simply samples sequentially from the hierarchy of common knowledge controllers. However, during training, the total probability of the joint action is evaluated by marginalising over all choices that could have been taken at every level of the hierarchy. Thus, even the parameters governing a subgroup that was not selected during the action sampling can receive gradients.

Using a matrix game we show results for a tabular version of MACKRL that outperforms both independent and joint-action learners, the code is available at `bit.ly/2PvFJAB`. Furthermore, we develop a DMARL variant called *pairwise MACKRL*, which uses pairwise coordinated strategies implemented with RNN policies and a centralised critic similar to what was proposed in Chapter 3. On a multi-agent variant of StarCraft II (SCII) micromanagement, similar to Section 3.3, pairwise MACKRL outperforms a baseline agent which uses a centralised critic.

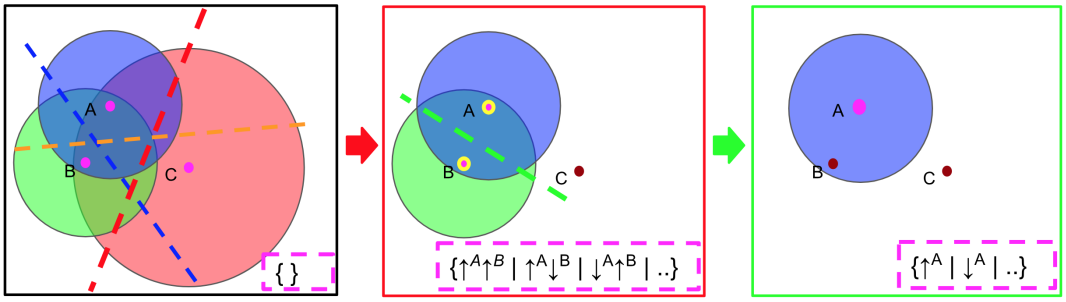



**Figure 4.2:** Pairwise MACKRL. Left: the pair selector must assign agents to pairs (plus a singleton in this case). Middle: the pair controller can either partition the pair or select among the pair's joint actions; Right, at the last level, the controller must select an action for a single agent.

We also show that the delegation decisions are semantically meaningful and relate to the amount of common knowledge between the agents.

## 4.2 Related Work

MARL has been studied extensively in small environments, see *e.g.*, [42, 43], but scaling it to large state spaces or many agents has proved problematic. One reason is that the joint action space grows exponentially in the number of agents, making learning intractable. Guestrin, Koller, and Parr [64] 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* [65] 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 are no conditional independencies to exploit and maximisation must still be performed over an intractably large joint action space.

Genter, Agmon, and Stone [66] and Barrett, Stone, and Kraus [67] examine *ad hoc teamwork*: how agents can cooperate with previously unseen teammates when there are a variable number of non-learning agents. Albrecht and Stone [68] address ad hoc teamwork by maintaining a belief over a set of parameterised hypothetical behaviours and updating them after each observation. Panella and Gmytrasiewicz

[69] treat the behaviours of teammates as a stochastic process and maintain beliefs over these in order to learn how to coordinate with previously unseen agents. Makino and Aihara [70] develop an algorithm that reasons over the beliefs over policies of other agents in fully observable settings. In ad-hoc teamwork, the learning agents learn to coordinate with other agents with fixed, non-learning behaviour; in our setting all agents learn at the same time, with the same known algorithm.

Thomas et al. [71] explore the psychology of common knowledge and coordination. Rubinstein [72] 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. [73] examine common knowledge in scenarios where agents use Facebook-like communication and show that a complete bipartite graph is required for common knowledge to be shared amongst a group. Brafman and Tennenholtz [74] 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. [75] introduce the concept of a *correlated equilibrium*, whereby a shared *correlation device* helps agents coordinate better. Cigler and Faltings [76] examine how the agents can reach such an equilibrium when given access to a simple shared *correlation vector* and a communication channel. Boutilier [77] 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. Instead of relying on a shared communication channel or full observability, MACKRL achieves coordination by utilising common knowledge.

Amato, Konidaris, and Kaelbling [78] propose MacDec-POMDPs, which use hierarchically optimal policies that allow agents to undertake temporally extended macro actions. Liu et al. [79] investigate how to learn such models in environments where the transition dynamics are not known. Makar, Mahadevan, and Ghavamzadeh [80] extend the MAXQ single-agent hierarchical framework by Dietterich [81] to the

multi-agent domain. They enable certain policies in the hierarchy to learn the joint action-value function, which allows for faster coordination across agents. However, unlike MACKRL this requires the agents to communicate during execution.

Kumar et al. [82] use a hierarchical controller that produces subtasks for each agent and chooses which pairs of agents should communicate in order to select their actions. In contrast to our approach, they allow communication during execution, and do not test on a sequential decision making task.

## 4.3 Dec-POMDP and Features

In this chapter, we consider a particular kind of *decentralised partially observable Markov decision processes* (Dec-POMDP) [20], that gives rise to common knowledge among agents.

In this Dec-POMDP, the state of the system $s \in \mathcal{S}$ is composed of a number of entities $e \in \mathcal{E}$, with state features $s^e$, i.e., $s = \{s^e \,|\, e \in \mathcal{E}\}$. A subset of the entities are agents: $a \in \mathcal{A} \subseteq \mathcal{E}$, where $|\mathcal{A}| = n$. Other entities could be enemies, obstacles, or goals.

At each timestep, each agent can select an action $u^a_{\text{env}} \in \mathcal{U}^a_{\text{env}}(s)$, where the subscript indicates an environment action. 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 joint reward from the function $r(s, \mathbf{u}_{\text{env}})$.

The agents have a particular form of partial observability. At each time step, each agent $a$ receives an observation $o^a \in \mathcal{Z}$ containing 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 $\mathcal{M}^a_s := \{e \,|\, \mu^a(s^a, s^e)\} \subseteq \mathcal{E}$, and the agent's observation is specified by the deterministic observation function $O(s, a)$ such that $o^a = O(s, a) = \{s^e \,|\, e \in \mathcal{M}^a_s\} \in \mathcal{Z}$.

The goal of the agents is to maximise the expected discounted forward looking return $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, \mathbf{u}_{t',\text{env}})$ experienced during trajectories of length $T$. The joint policy $\pi(\mathbf{u}_{\text{env}}|s)$ is restricted to a set of decentralised policies $\pi^a(u^a_{\text{env}}|\tau^a)$, that can be executed independently, i.e. each agent's policy conditions only on its own action-observation history $\tau^a$. We denote a policy across the joint action space $\mathcal{U}^{\mathcal{G}}_{\text{env}}$ of the group $\mathcal{G} \subseteq \mathcal{A}$ of agents as $\pi^{\mathcal{G}}$. Like in the rest of the thesis, we allow centralised learning of decentralised policies.

Our setting is thus a special case of the Dec-POMDP in which the state space factors into entities and the observation function is deterministic, yielding perceptual aliasing in which the state features of each entity are either accurately observed or completely occluded. In the next section, we leverage these properties to establish common knowledge among the agents. 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.

## 4.4 Common Knowledge

A key property of the binary mask $\mu^a$ 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 all agent $a$'s mask $\mu^a$ is common knowledge, then this means that another agent $b$, that can see $a$ and $e$, i.e., $a, e \in \mathcal{M}^b_s$, can deduce whether $a$ can also see $e$. Assuming all agents' binary masks are known can give rise to common knowledge about entities.

The *mutual knowledge* $\mathcal{M}^{\mathcal{G}}_s$ 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: $\mathcal{M}^{\mathcal{G}}_s := \cap_{a \in \mathcal{G}} \mathcal{M}^a_s$. However, mutual knowledge does not imply common knowledge. Instead, the common knowledge $\mathcal{I}^{\mathcal{G}}_s$ of group $\mathcal{G}$ in state $s$ is the set of entities such that all agents in $\mathcal{G}$ see $\mathcal{I}^{\mathcal{G}}_s$, know that all other agents in $\mathcal{G}$ see $\mathcal{I}^{\mathcal{G}}_s$, know that they know that all other agents see $\mathcal{I}^{\mathcal{G}}_s$, etc.

To know that another agent $b$ also sees $e \in \mathcal{E}$, agent $a$ must see $b$ and $b$ must see $e$, i.e., $\mu^a(s^a, s^b) \wedge \mu^b(s^b, s^e) = \top$. Common knowledge $\mathcal{I}^{\mathcal{G}}_s$ can then be formalised

recursively for every agent $a \in \mathcal{G}$ as:

$$\mathcal{I}_s^{\mathcal{G}} = \lim_{m\to\infty} \mathcal{I}_s^{a,m}, \qquad \mathcal{I}_s^{a,0} = \mathcal{M}_s^a, \tag{4.4.1}$$

$$\mathcal{I}_s^{a,m} = \bigcap_{b\in\mathcal{G}} \left\{ e \in \mathcal{I}_s^{b,m-1} \,|\, \mu^a(s^a, s^b) \right\}. \tag{4.4.2}$$

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.

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

$$\mathcal{I}_s^{\mathcal{G}} = \begin{cases} \mathcal{M}_s^{\mathcal{G}} & , \textit{if} \bigwedge\limits_{a,b\in\mathcal{G}} \mu^a(s^a, s^b) \\ \varnothing & , \textit{otherwise} \end{cases}. \tag{4.4.3}$$

*Proof.* The lemma follows by induction on $m$. The recursive definition of common knowledge (4.4.2) holds trivially if $\mathcal{I}_s^{\mathcal{G}} = \varnothing$. Starting from the knowledge of any agent $a$ in state $s$, $\mathcal{I}^{a,0} = \mathcal{M}_s^a$, definition (4.4.2) yields:

$$\mathcal{I}^{a,1} = \begin{cases} \mathcal{M}_s^{\mathcal{G}} & , \text{if} \bigwedge\limits_{b\in\mathcal{G}} \mu^a(s^a, s^b) \\ \varnothing & , \text{otherwise} \end{cases}.$$

Next we will show inductively that if all agents in group $\mathcal{G}$ know the mutual knowledge $\mathcal{M}_s^{\mathcal{G}}$ of state $s$ at some iteration $m$, that is, $\mathcal{I}_s^{c,m} \overset{\text{ind.}}{=} \mathcal{M}_s^{\mathcal{G}}$, then this mutual knowledge becomes common knowledge 2 iterations later. Applying the definition (4.4.2) for any agent $a \in \mathcal{G}$ twice yields:

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

where we used

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

Therefore, starting at the knowledge any agent of group $\mathcal{G}$, in which all agents can see each other, the mutual knowledge becomes the common knowledge for all $m \geq 3$. □

The common knowledge can be computed using only the visible set $\mathcal{M}_s^a$ 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 can also be considered common knowledge. The common knowledge of group $\mathcal{G}$ up to time $t$ is thus some common prior knowledge $\tau_0$ and the commonly known trajectory $\tau_t^{\mathcal{G}} = (\tau_0, o_1^{\mathcal{G}}, \mathbf{u}_1^{\mathcal{G}}, \ldots, o_t^{\mathcal{G}}, \mathbf{u}_t^{\mathcal{G}})$, with $o_k^{\mathcal{G}} = \{s_k^e \,|\, e \in \mathcal{I}_{s_k}^{\mathcal{G}}\}$. Knowing all binary masks $\mu^a$ makes it possible to derive $\tau_t^{\mathcal{G}} = \mathcal{I}^{\mathcal{G}}(\tau_t^a)$ from the observation trajectory $\tau_t^a = (\tau_0, o_1^a, \ldots, o_t^a)$ of every agent $a \in \mathcal{G}$, and a function that conditions on $\tau^{\mathcal{G}}$ can therefore be computed independently by every member of $\mathcal{G}$. This idea of common knowledge based on local observations is illustrated in Figure 4.1.

## 4.5 Multi-Agent Common Knowledge Reinforcement Learning

The key idea behind MACKRL is to learn decentralised policies that are nonetheless coordinated. By conditioning the joint policy $\pi^{\mathcal{G}}(\mathbf{u}_{\text{env}}^{\mathcal{G}} | \mathcal{I}^{\mathcal{G}}(\tau^a))$ over the joint action space $\mathcal{U}_{\text{env}}^{\mathcal{G}}$ of a group of agents $\mathcal{G}$ only on the common knowledge of that group, MACKRL learns centralised policies that can be executed in a decentralised fashion. Since all agents $a \in \mathcal{G}$ have access to $\mathcal{I}^{\mathcal{G}}(\tau^a)$ and the shared random seed, they can sample the same joint action $\mathbf{u}_{\text{env}}^{\mathcal{G}}$ and execute their individual component $u_{\text{env}}^a$.

If $\mathcal{I}^{\mathcal{G}}(\tau^a)$ is sufficiently informative, then in principle it is possible to learn a $\pi^{\mathcal{G}}$ that chooses high quality joint actions. However, if $\mathcal{I}^{\mathcal{G}}(\tau^a)$ is not sufficiently informative, it may be preferable for $\mathcal{G}$ to delegate action selection to smaller subgroups. Coordination would no longer occur across the subgroups, but action selection within each group could condition on a richer common knowledge signal. Furthermore, the choice between acting and delegating must itself be decentralisable amongst the agents in $\mathcal{G}$ and can thus only condition on $\mathcal{I}^{\mathcal{G}}(\tau^a)$. MACKRL tackles this problem using a hierarchy of controllers. At the top level is a controller that can either select a joint action across all agents, $\mathbf{u}_{\text{env}}$, or specify a partition of the agents to delegate action selection to. At intermediate levels are controllers

that can either select joint actions $\mathbf{u}_{\text{env}}^{\mathcal{G}}$ for a group $\mathcal{G}$ or a partition of $\mathcal{G}$. At the bottom level are controllers that select actions $u_{\text{env}}^{a}$ for all individual agents that have not yet been assigned actions. Figure 4.2 illustrates the hierarchy for the pairwise version of this model discussed in the next section. Formally, the controller for a group $\mathcal{G}$ can be described as:

$$u^{\mathcal{G}} \sim \pi^{\mathcal{G}}\Big(u^{\mathcal{G}} \,|\, \mathcal{I}^{\mathcal{G}}(\tau^{a})\Big) \qquad \text{with} \qquad u^{\mathcal{G}} \in \mathcal{U}^{\mathcal{G}}\,, \tag{4.5.1}$$

where $\mathcal{U}^{\mathcal{G}} = \{\mathcal{U}_{\text{env}}^{\mathcal{G}} \cup \mathcal{P}(\mathcal{G})\}$ and $\mathcal{P}(\mathcal{G})$ is the set of all partitions of $\mathcal{G}$. Algorithm 2 summarises MACKRL's hierarchical action selection. Since the hierarchical controllers condition only on the common knowledge as defined in (4.4.2), the actions can be executed by each agent. We use $\mathbf{u}^{\mathcal{P}} = \prod_{\mathcal{G}\in\mathcal{P}} u^{\mathcal{G}}$ to denote the joint action space of controllers in a partition $\mathcal{P}$.

**Algorithm 2** Action Selection in MACKRL

```
function GETACTION(τ_t^a)
  Set u_env = 0    ▷ Initialise joint action
  Set b = [A]    ▷ Initialise group buffer with entire group
  while not b.empty() do
    Set G = b.pop()    ▷ Get the next group from buffer
    Sample u^G ~ π^G(u^G|I^G(τ_t^a))
    if u^G ∈ U_env^G then    ▷ Sample a joint action for G
      u_env[G] = u^G    ▷ Assign joint action elements
    else    ▷ Delegate to selected partitions
      for G' ∈ u^G do
        b.push(G')    ▷ Add groups G' in partition u^G
  return u_env
```

However, rather than training a set of factorised policies, one for each agent, we train the marginalised joint policy $P(\mathbf{u}_{\text{env}}|s)$ induced by the hierarchical sampling process:

$$P(\mathbf{u}_{\text{env}}|s) = \sum_{\text{path}\in\text{Paths}} P(\mathbf{u}_{\text{env}}|s, \text{path})P(\text{path}|s), \tag{4.5.2}$$

where Paths is the set of all possible action assignments of the hierarchical controllers, that is, each path is a possible outcome of the action-selection in Algorithm 2. In general, sampling from a joint probability is problematic since the number of logits grows exponentially in the number of agents. Furthermore, evaluating the joint

probability requires central state information, and thus cannot be decentralised. The key insight of MACKRL is that we can use the hierarchical, decentralised process for sampling actions, while the marginal probabilities need to be computed only during training in order to obtain the joint probability of the joint action that was actually selected.

To train these policies, we use an actor-critic setup with a centralised baseline similar to Chapter 3. In contrast to the decentralised policy, we condition the critic on the central state and the last actions of all agents. Since MACKRL induces a correlated probability across the joint action space, it effectively turns training into a single agent problem and renders the COMA baseline proposed in Chapter 3 non-applicable.

The gradient w.r.t. the policy parameters $\theta$ is:

$$\nabla_\theta J_t \quad = \quad A_t \, \nabla_\theta \log(p(\mathbf{u}_{\text{env},t}|s_t)), \tag{4.5.3}$$

where $A_t = (r_t + \gamma V(s_{t+1}, \mathbf{u}_{\text{env},t}) - V(s_t, \mathbf{u}_{\text{env},t-1}))$ is an estimate of the advantage. The value function is learned by gradient descent on the TD($\lambda$) loss [21], as is standard in this thesis. As in Central-$V$, all critics are trained on-policy using samples from the environment, and all controllers within one layer of the hierarchy share parameters. The value function conditions both on the state, $s_t$, and on the last joint-action at each time step, $\mathbf{u}_{t-1}$, in order to account for durative actions.

In general, both the number of possible partitions and groups per partition can be large, making learning difficult. However, the next section describes an example implementation that illustrates how learning can be made tractable.

$$\begin{pmatrix} 1 & 0 & 0 & 0.4 & 0 \\ 0 & 0.2 & 0.4 & 0.8 & 0.4 \\ 0 & 0 & 0 & 0.4 & 0 \\ 0 & 0 & 0 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 4.3:** Matrix A

$$\begin{pmatrix} 0 & 0 & 0.2 & 0 & 1 \\ 0 & 0 & 0.4 & 0 & 0 \\ 0.2 & 0.4 & 0.8 & 0.4 & 0.2 \\ 0 & 0 & 0.4 & 0 & 0 \\ 1 & 0 & 0.2 & 0 & 0 \end{pmatrix}$$

**Figure 4.4:** Matrix B

## 4.6 Pairwise MACKRL

In this thesis, we focus on a simple implementation of MACKRL that we call *pairwise MACKRL*. The hierarchy consists of three levels. At the top level, the controller $\pi_{\text{ps}}$ is a *pair selector* that is not allowed to directly select a joint action but can only choose among pairwise partitions, i.e., it must group the agents into disjoint pairs. If there are an odd number of agents, then one agent is put in a singleton group. At the second level, each controller $\pi_{\text{pc}}^{aa'}$ is a *pair controller* whose action space consists of the joint action space of the pair of agents plus a delegate action $u_d$ that further partitions the pair into singletons: $\mathcal{U}^{\mathcal{G}} = \mathcal{U}_{\text{env}}^{a} \times \mathcal{U}_{\text{env}}^{a'} \cup \{u_d\}$, where $\mathcal{G} = \{a, a'\}$. At the third level, each controller selects an individual action $u_{\text{env}}^{a}$ for a single agent $a$. Figure 4.2 illustrates pairwise MACKRL.

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

**Table 4.1:** Hierarchy of pairwise MACKRL, where $h$ is the hidden state of RNNs and $o_t^a$ are observations. $\#\pi$ shows the number of controllers on this level for 3 agents.

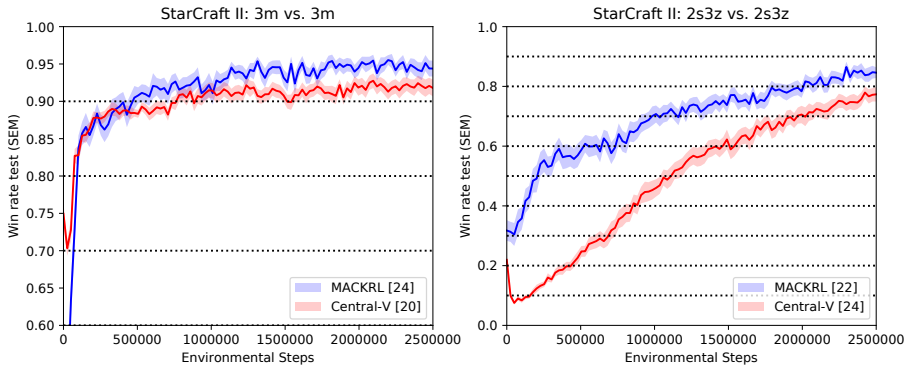



**Figure 4.6:** Mean and standard error of the mean of test win rates for two levels in StarCraft II: one with 5 marines (left), and one with 2 stalkers and 3 zealots on each side (right). Also shown is the number of runs [in brackets].

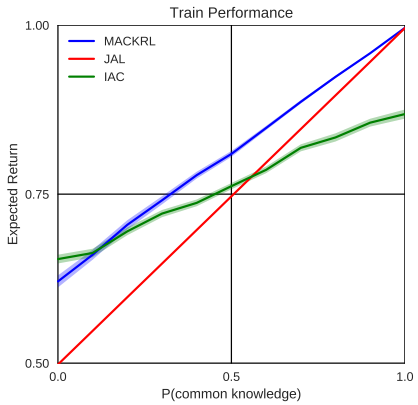



**Figure 4.5:** Results for the matrix game.

## 4.7 Experiments and Results

We evaluate MACKRL on two tasks. The first is a matrix game with partial observability. The state consists of two randomly sampled bits, which are drawn iid. The first bit indicates the *information state* and is always observable by both agents. The second bit selects which one of the two normal form games the agents are playing and is sampled 50%. When the first bit is in the 'common knowledge' state, which happens with a probability of $P$(common knowledge), the matrix bit is always observable by both agents and is thus common knowledge.

In contrast, when the first bit is in the 'obfuscation state', each of the agents observes the 'matrix bit' with a probability of 50% (iid), indicating whether the reward will be chosen using the matrix shown in Figure 4.3 or the matrix shown in Figure 4.4, and receives a 'no observation' otherwise. The code including a proof of principle implementation is available in the Supplementary Material and published online. We explore how MACKRL performs, for a range of values for $P$(common knowledge), in comparison to an always centralised policy using only the common knowledge and fully decentralised learners.

Figure 4.5 shows performance on the matrix game as a function of the probability of the common knowledge bit. When there is no common knowledge MACKRL reproduces the performance of independent actor critic (IAC), while a centralised policy, learned with *joint-action-learning* (JAL) [42], fails to take advantage of the private

observations. When common knowledge is always available MACKRL matches the performance of JAL. For intermediate probabilities, MACKRL outperforms both IAC and JAL. Inspecting the policy verifies that MACKRL learns to delegate to the independent learners whenever the common knowledge bit is absent but carries out a joint action when the bit is present. This corresponds to the optimal policy.

The second task is a challenging multi-agent version of StarCraft II micromanagement. It closely resembles our multi-agent StarCraft variant described in Section 3.3, *i.e.*, each unit is controlled by a decentralised agent and has to rely on local information for action selection. However, here we test on the StarCraft II version as described in [51] and use the maps where both sides have either 3 Marines (denoted '3m') or 2 Stalkers and 3 Zealots (denoted '2s3z'). The '3m' map was chosen as a simple reference map since it has been tackled in previous work. The '2s3z' consists of different unit types with different attack abilities and thus allows for more complex coordination between the agents. In particular Rashid et al. [51] showed that independent learners fail to perform well on the '2s3z' map, making it a good testbed for our method.

All policies are implemented as 2-layer recurrent neural networks (GRUs) with 64 hidden units, while the critic is feedforward and uses full state information. Table 4.1 describes the inputs and action spaces at each level of the hierarchy. Parameters are shared across controllers within each of the second and third levels of the hierarchy. Therefore, we also feed in the agent index or index pairs into the policy. For exploration we used 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 $\epsilon$. $\epsilon$ was annealed from 0.5 to the final value of 0.01 across the first 50k environment steps.

Episodes were collected using 8 parallel environments of SCII, and optimisation was done on GPU using ADAM with a learning rate of 0.0005 for both the agents and the critic. The policies were fully unrolled and updated in a large minibatch of $T \times B$ entries, where $T = 60$ and $B = 8$. In contrast the critic was optimised in small minibatches of size 8, one for each timestep, looping backwards in time.

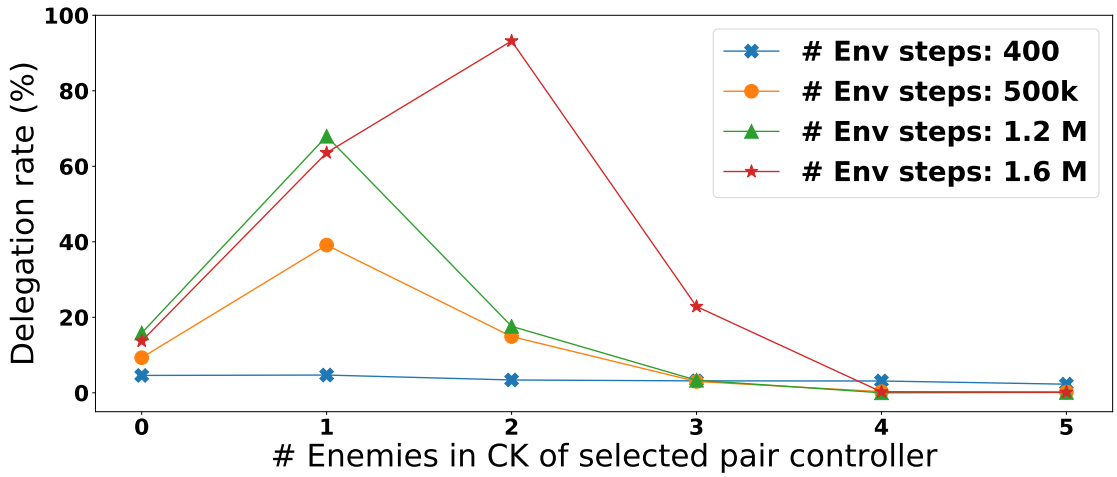



**Figure 4.7:** Delegation rate vs. number of enemies (2s3z) in the common knowledge of the pair controller over training.

We found that this both stabilised but also accelerated training compared to doing full batch updates for the critic. The target network for the critic was updated after every 200 update steps for the critic. We also used a TD-lambda of 0.8 to accelerate reward propagation across time.

Figure 4.6 shows the win rate of StarCraftII agents during test trajectories on our two maps. We omit independent learning since it is known to do poorly in this setting [51]. On the easier map (3m), both methods achieve near state-of-the-art performance above 90%. Since Central-V has around three times fewer parameters, it is able to initially learn faster on this simple map, but MACKRL achieves marginally higher final performance. On the more challenging map with mixed unit types (2s3z), MACKRL learns faster and achieves a higher final score. These results show that MACKRL can outperform a fully factored policy when all other aspects of training are the same.

To demonstrate that the pair controller can indeed learn to delegate strategically, we plot in Figure 4.7 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.

## 4.8 Conclusion & Future Work

We introduce MACKRL, a method which allows a team of agents to learn a policy in the joint action space using common knowledge, such that the policy can still be executed in a fully decentralised fashion. MACKRL shows strong performance on a matrix game where it clearly beats a joint-action-learner and independent learners. We also test MACKRL on our challenging multi-agent version of StarCraftII micromanagement and find that it outperforms a strong baseline in the harder of the two maps, and marginally on the easier one. In future work, we will investigate scaling MACKRL to settings with many agents and develop methods that allow agents to learn to deduce common knowledge from their trajectories, rather than having it presented separately. In the next chapter we focus on nonstationarity in the context of $Q$-learning based methods when applied to MARL.

# 5 Stabilising Experience Replay

## Contents



## 5.1 Introduction

In Chapter 3 and Chapter 4 we present multi-agent algorithm based on on-policy actor-critic methods. One of the challenges limiting the application of deep RL to real world problems is *sample efficiency*. Off-policy methods such as DQN need less data to learn since they can carry out multiple learning steps on the same samples.

A popular adaptation of DQN to MARL is *independent Q-learning* (IQL) [25], in which each agent independently learns its own policy, treating other agents as

part of the environment. While IQL avoids the scalability problems of centralised learning, it introduces a new problem: the environment becomes nonstationary from the point of view of each agent, as it contains other agents who are themselves learning, ruling out any convergence guarantees. Fortunately, substantial empirical evidence has shown that IQL often works well in practice [83].

In recent years, the use of deep neural networks has dramatically improved the scalability of single-agent RL [13]. However, one element key to the success of such approaches is the reliance on an *experience replay memory*, which stores experience tuples that are sampled during training. Experience replay not only helps to stabilise the training of a deep neural network, it also improves sample efficiency by repeatedly reusing experience tuples. Unfortunately, the combination of experience replay with IQL appears to be problematic: the nonstationarity introduced by IQL means that the dynamics that generated the data in the agent's replay memory no longer reflect the current dynamics in which it is learning. While IQL without a replay memory can learn well despite nonstationarity so long as each agent is able to gradually track the other agents' policies, that seems hopeless with a replay memory constantly confusing the agent with obsolete experience.

To avoid this problem, previous work on DMARL has limited the use of experience replay to short, recent buffers [44] or simply disabled replay altogether, *e.g.*, our work in Chapter 6. However, these workarounds limit the sample efficiency and threaten the stability of multi-agent RL. Consequently, the incompatibility of experience replay with IQL is emerging as a key stumbling block to scaling DMARL to complex tasks.

In this chapter, we propose two approaches for effectively incorporating experience replay into multi-agent RL. The first approach interprets the experience in the replay memory as *off-environment* data [84]. By augmenting each tuple in the replay memory with the probability of the joint action in that tuple, according to the policies in use at that time, we can compute an importance sampling correction when the tuple is later sampled for training. Since older data tends to generate lower

importance weights, this approach naturally decays data as it becomes obsolete, preventing the confusion that a nonstationary replay memory would otherwise create.

The second approach is inspired by *hyper Q-learning* [85], which avoids the nonstationarity of IQL by having each agent learn a policy that conditions on an estimate of the other agents' policies inferred from observing their behaviour. While it may seem hopeless to learn $Q$-functions in this much larger space, especially when each agent's policy is a deep neural network, we show that doing so is feasible as each agent need only condition on a low-dimensional *fingerprint* that is sufficient to disambiguate where in the replay memory an experience tuple was sampled from.

We evaluate these methods on our multi-agent variant of *StarCraft unit micromanagement*[1], see Section 3.3. Our results confirm that, thanks to our proposed methods, experience replay can indeed be successfully combined with multi-agent $Q$-learning to allow for stable training of deep multi-agent value functions.

## 5.2 Related Work

Beyond the related work mentioned in Chapter 3 our work is also broadly related to methods that attempt to allow for faster learning for value based methods. These include *prioritised experience replay* [86], a version of the standard replay memory that biases the sampling distribution based on the TD error. However, this method does not account for nonstationary environments and does not take into account the unique properties of the multi-agent setting.

Methods like hyper Q-learning [85] and AWESOME [87] try to tackle nonstationarity by tracking and conditioning each agent's learning process on their teammates' current policy, while Da Silva et al. [88] propose detecting and tracking different classes of traces on which to condition policy learning. Kok and Vlassis [89] show that coordination can be learnt by estimating a global $Q$-function in the classical distributed setting supplemented with a coordination graph. In general, these techniques have so far not successfully been scaled to high-dimensional state spaces.

[1] StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment™.

Lauer and Riedmiller [90] propose a variation of distributed Q-learning, a coordination-free method. However, they also argue that the simple estimation of the value function in the standard model-free fashion is not enough to solve multi-agent problems, and coordination through means such as communication [91] is required to ground separate observations to the full state function. Wang et al. [34] describe an importance sampling method for using off-policy experience in a single-agent actor-critic algorithm. However, to calculate policy-gradients, the importance ratios become products over potentially lengthy trajectories, introducing high variance that must be partially compensated for by truncation. By contrast, we address *off-environment* learning and show that the multi-agent structure results in importance ratios that are simply products over the agents' policies.

## 5.3 Methods

To avoid the difficulty of combining IQL with experience replay, previous work on DMARL has limited the use of experience replay to short, recent buffers [44] or simply disabled replay altogether, *e.g.*, our work in Chapter 6. However, these workarounds limit the sample efficiency and threaten the stability of multi-agent RL. In this section, we propose two approaches for effectively incorporating experience replay into multi-agent RL.

### 5.3.1 Multi-Agent Importance Sampling

We can address the nonstationarity present in IQL by developing an importance sampling scheme for the multi-agent setting. Just as an RL agent can use importance sampling to learn *off-policy* from data gathered when its own policy was different, so too can it learn *off-environment* [84] from data gathered in a different environment. Since IQL treats other agents' policies as part of the environment, off-environment importance sampling can be used to stabilise experience replay. In particular, since we know the policies of the agents at each stage of training, we know exactly the manner in which the environment is changing, and can thereby correct for it with importance weighting, as follows. We consider first a fully-observable multi-agent

setting. If the $Q$-functions can condition directly on the true state $s$, we can write the Bellman optimality equation for a single agent given the policies of all other agents:

$$Q_a^*(s, u^a|\boldsymbol{\pi}^{-a}) = \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|s)\Big[r(s, u^a, \mathbf{u}^{-a}) + \gamma \sum_{s'} P(s'|s, u^a, \mathbf{u}^{-a}) \max_{u'^a} Q_a^*(s', u'^a)\Big]. \quad (5.3.1)$$

The nonstationary component of this equation is $\boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|s) = \Pi_{a' \in -a}\pi^{a'}(u^{a'}|s)$, which changes as the other agents' policies change over time. Therefore, to enable importance sampling, at the time of collection $t_c$, we record $\boldsymbol{\pi}_{t_c}^{-a}(\mathbf{u}^{-a}|s)$ in the replay memory, forming an augmented transition tuple $\langle s, u^a, r, \pi(\mathbf{u}^{-a}|s), s'\rangle^{(t_c)}$.

At the time of replay $t_r$, we train off-environment by minimising an importance weighted loss function:

$$\mathcal{L}(\theta) = \sum_{i=1}^{b} \frac{\boldsymbol{\pi}_{t_r}^{-a}(\mathbf{u}^{-a}|s)}{\boldsymbol{\pi}_{t_i}^{-a}(\mathbf{u}^{-a}|s)}[(y_i^{DQN} - Q(s, u; \theta))^2], \quad (5.3.2)$$

where $t_i$ is the time of collection of the $i$-th sample.

The derivation of the nonstationary parts of the Bellman equation in the partially observable multi-agent setting is considerably more complex as the agents' action-observation histories are correlated in a complex fashion that depends on the agents' policies as well as the transition and observation functions.

To make progress, we can define an augmented state space $\hat{s} = \{s, \boldsymbol{\tau}^{-a}\} \in \hat{S} = S \times T^{n-1}$. This state space includes both the original state $s$ and the action-observation history of the other agents $\boldsymbol{\tau}^{-a}$. We also define a corresponding observation function $\hat{O}$ such that $\hat{O}(\hat{s}, a) = O(s, a)$. With these definitions in place, we define a new reward function $\hat{r}(\hat{s}, u) = \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|\boldsymbol{\tau}^{-a})r(s, \mathbf{u})$ and a new transition function,

$$\hat{P}(\hat{s}'|\hat{s}, u) = P(s', \tau'|s, \tau, u) = \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|\boldsymbol{\tau}^{-a})P(s'|s, \mathbf{u})P(\boldsymbol{\tau}'^{-a}|\boldsymbol{\tau}^{-a}, \mathbf{u}^{-a}, s'). \quad (5.3.3)$$

All other elements of the augmented game $\hat{G}$ are adopted from the original game $G$. This also includes $T$, the space of action-observation histories. The augmented game

is then specified by $\hat{G} = \langle \hat{S}, U, \hat{P}, \hat{r}, Z, \hat{O}, n, \gamma \rangle$. We can now write a Bellman equation for $\hat{G}$:

$$Q(\tau, u) = \sum_{\hat{s}} p(\hat{s}|\tau) \Bigg[ \hat{r}(\hat{s}, u) + \gamma \sum_{\tau', \hat{s}', u'} \hat{P}(\hat{s}'|\hat{s}, u) \pi(u', \tau') p(\tau'|\tau, \hat{s}', u) Q(\tau', u') \Bigg]. \quad (5.3.4)$$

Substituting back in the definitions of the quantities in $\hat{G}$, we arrive at a Bellman equation of a form similar to (5.3.1), where the right-hand side is multiplied by $\boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|\boldsymbol{\tau}^{-a})$:

$$Q(\tau, u) = \sum_{\hat{s}} p(\hat{s}|\tau) \sum_{\mathbf{u}^{-a}} \boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|\boldsymbol{\tau}^{-a}) \Bigg[ r(s, \mathbf{u}) + \gamma \sum_{\tau', \hat{s}', u'} P(s'|s, \mathbf{u}) p(\boldsymbol{\tau}'^{-a}|\boldsymbol{\tau}^{-a}, \mathbf{u}^{-a}, s') \cdot \pi(u', \tau') p(\tau'|\tau, \hat{s}', u) Q(\tau', u') \Bigg]. \quad (5.3.5)$$

This construction simply allows us to demonstrate the dependence of the Bellman equation on the same nonstationary term $\boldsymbol{\pi}^{-a}(\mathbf{u}^{-a}|s)$ in the partially-observable case. However, unlike in the fully observable case, the right-hand side contains several other terms that indirectly depend on the policies of the other agents and are to the best of our knowledge intractable. Consequently, the importance ratio defined above, $\frac{\boldsymbol{\pi}_{t_r}^{-a}(\mathbf{u}^{-a}|s)}{\boldsymbol{\pi}_{t_i}^{-a}(\mathbf{u}^{-a}|s)}$, is only an approximation in the partially observable setting.

### 5.3.2 Multi-Agent Fingerprints

While importance sampling provides an unbiased estimate of the true objective, it often yields importance ratios with large and unbounded variance [92]. Truncating or adjusting the importance weights can reduce the variance but introduces bias. Consequently, we propose an alternative method that embraces the nonstationarity of multi-agent problems, rather than correcting for it.

The weakness of IQL is that, by treating other agents as part of the environment, it ignores the fact that such agents' policies are changing over time, rendering its own $Q$-function nonstationary. This implies that the $Q$-function could be made

stationary if it conditioned on the policies of the other agents. This is exactly the philosophy behind *hyper Q-learning* [85]: each agent's state space is augmented with an estimate of the other agents' policies computed via Bayesian inference. Intuitively, this reduces each agent's learning problem to a standard, single-agent problem in a stationary, but much larger, environment.

The practical difficulty of hyper Q-learning is that it increases the dimensionality of the $Q$-function, making it potentially infeasible to learn. This problem is exacerbated in deep learning, when the other agents' policies consist of high dimensional deep neural networks. Consider a naive approach to combining hyper Q-learning with deep RL that includes the weights of the other agents' networks, $\theta^{-a}$, in the observation function. The new observation function is then $O'(s) = \{O(s), \theta^{-a}\}$. The agent could in principle then learn a mapping from the weights $\theta^{-a}$, and its own trajectory $\tau$, into expected returns. Clearly, if the other agents are using deep models, then $\theta^{-a}$ is far too large to include as input to the $Q$-function.

However, a key observation is that, to stabilise experience replay, each agent does not need to be able to condition on any possible $\theta^{-a}$, but only those values of $\theta^{-a}$ that actually occur in its replay memory. The sequence of policies that generated the data in this buffer can be thought of as following a single, one-dimensional trajectory through the high-dimensional policy space. To stabilise experience replay, it should be sufficient if each agent's observations disambiguate where along this trajectory the current training sample originated from.

The question then, is how to design a low-dimensional *fingerprint* that contains this information. Clearly, such a fingerprint must be correlated with the true value of state-action pairs given the other agents' policies. It should typically vary smoothly over training, to allow the model to generalise across experiences in which the other agents execute policies of varying quality as they learn. An obvious candidate for inclusion in the fingerprint is the training iteration number $e$. One potential challenge is that after policies have converged, this requires the model to fit multiple fingerprints to the same value, making the function somewhat harder to learn and more difficult to generalise from.

Another key factor in the performance of the other agents is the rate of exploration $\epsilon$. Typically an annealing schedule is set for $\epsilon$ such that it varies smoothly throughout training and is quite closely correlated to performance. Therefore, we further augment the input to the $Q$-function with $\epsilon$, such that the observation function becomes $O'(s) = \{O(s), \epsilon, e\}$. Our results in Section 5.5 show that even this simple fingerprint is remarkably effective.

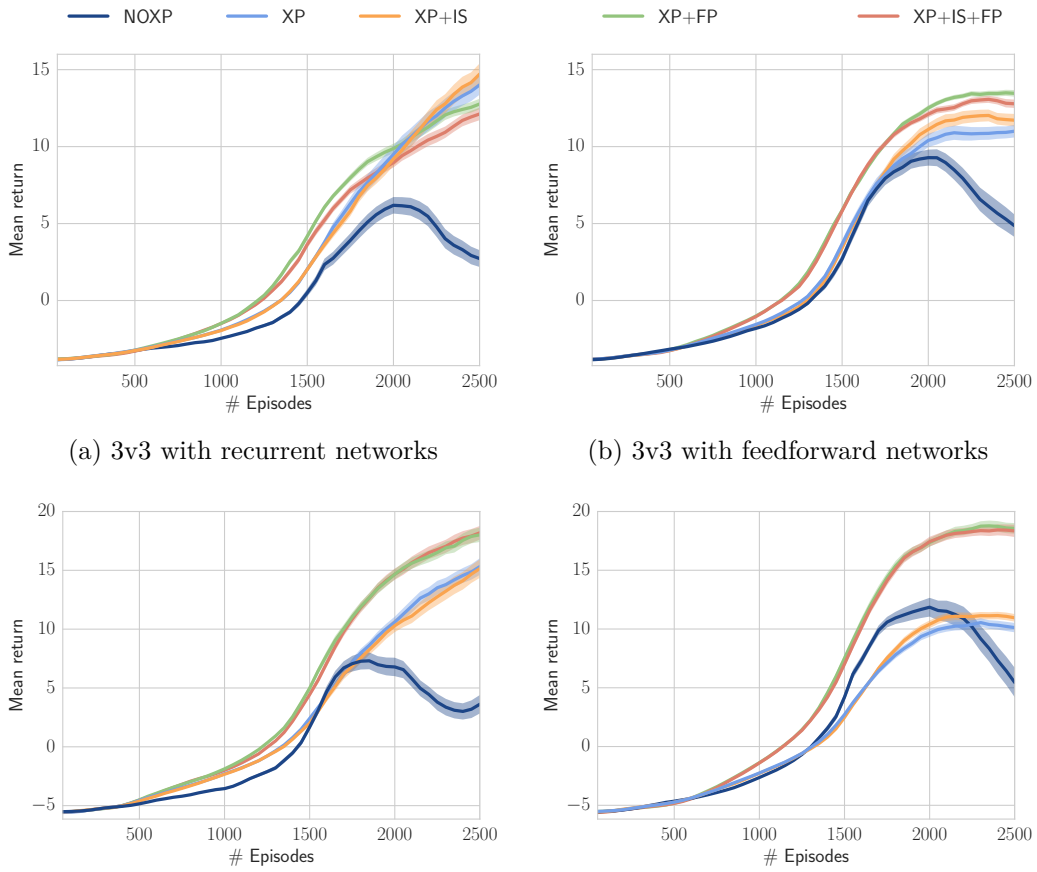
## 5.4 Experiments

In this section, we describe our experiments applying experience replay with fingerprints (XP+FP), with importance sampling (XP+IS), and with the combination (XP+IS+FP), to the StarCraft domain. We run experiments with both feedforward (FF) and recurrent (RNN) models, to test the hypothesis that in StarCraft recurrent models can use trajectory information to more easily disambiguate experiences from different stages of training.

### 5.4.1 Architecture

We use the recurrent DQN architecture described in Chapter 6 with a few modifications. We do not consider communicating agents, so there are no message connections. As mentioned above, we use two different different models: one with a feedforward model with two fully connected hidden layers, and another with a single-layer GRU. For both models, every hidden layer has 128 neurons.

We linearly anneal $\epsilon$ from 1.0 to 0.02 over 1500 episodes, and train the network for $e_{max} = 2500$ training episodes. In the standard training loop, we collect a single episode and add it to the replay memory at each training step. We sample batches of $\frac{30}{n}$ episodes uniformly from the replay memory and train on fully unrolled episodes. In order to reduce the variance of the multi-agent importance weights, we clip them to the interval $[0.01, 2]$. We also normalise the importance weights by the number of agents, by raising them to the power of $\frac{1}{n-1}$. Lastly, we divide the importance weights by their running average in order to keep the overall learning rate constant. All other hyperparameters are identical to the ones being used in Chapter 6.




(a) 3v3 with recurrent networks

(b) 3v3 with feedforward networks

(c) 5v5 with recurrent networks

(d) 5v5 with feedforward networks

**Figure 5.1:** Performance of our methods compared to the two baselines XP and NOXP, for both RNN and FF; (a) and (b) show the 3v3 setting, in which IS and FP are only required with feedforward networks; (c) and (d) show the 5v5 setting, in which FP clearly improves performance over the baselines, while IS shows a small improvement only in the feedforward setting. Overall, the FP is a more effective method for resolving the nonstationarity and there is no additional benefit from combining IS with FP. Confidence intervals show one standard deviation of the sample mean.

## 5.5 Results

In this section, we present the results of our StarCraft experiments, summarised in Figure 5.1. Across all tasks and models, the baseline without experience replay (NOXP) performs poorly. Without the diversity in trajectories provided by experience replay, NOXP overfits to the greedy policy once $\epsilon$ becomes small. When exploratory actions do occur, agents visit areas of the state space that have not had their $Q$-values updated for many iterations, and bootstrap off of values which have become stale or distorted by updates to the $Q$-function elsewhere. This

effect can harm or destabilise the policy. With a recurrent model, performance simply degrades, while in the feedforward case, it begins to drop significantly later in training. We hypothesise that full trajectories are inherently more diverse than single observations, as they include compounding chances for exploratory actions. Consequently, it is easier to overfit to single observations, and experience replay is more essential for a feedforward model.

With a naive application of experience replay (XP), the model tries to simultaneously learn a best-response policy to every historical policy of the other agents. Despite the nonstationarity, the stability of experience replay enables XP to outperform NOXP in each case. However, due to limited disambiguating information, the model cannot appropriately account for the impact of any particular policy of the other agents, or keep track of their current policy. The experience replay is therefore used inefficiently, and the model cannot generalise properly from experiences early in training.

### 5.5.1 Importance Sampling

The importance sampling approach (XP+IS) slightly outperforms XP when using feedforward models. While mathematically sound in the fully observable case, XP+IS is only approximate for our partially observable problem, and runs into practical obstacles. Early in training, the importance weights are relatively well behaved and have low variance. However, as $\epsilon$ drops, the importance ratios become multi-modal with increasing variance. The large majority of importance weights are less than or equal to $\epsilon(1-\epsilon) \approx \epsilon$, so few experiences contribute strongly to learning. In a setting that does not require as strongly deterministic a policy as StarCraft, $\epsilon$ could be kept higher and the variance of the importance weights would be lower.

### 5.5.2 Fingerprints

Our results show that the simple fingerprint of adding $e$ and $\epsilon$ to the observation (XP+FP) dramatically improves performance for the feedforward model. This fingerprint provides sufficient disambiguation for the model to track the quality

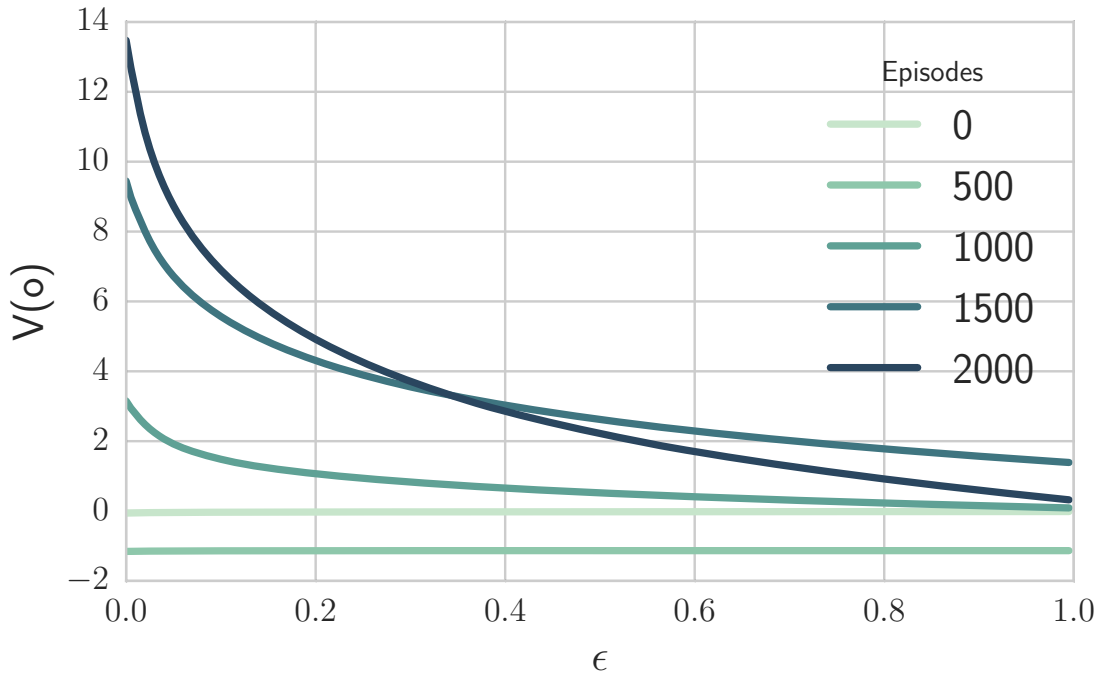



**Figure 5.2:** Estimated value of a single initial observation with different $\epsilon$ in its fingerprint input, at different stages of training. The network learns to smoothly vary its value estimates across different stages of training.

of the other agents' policies over the course of training, and make proper use of the experience buffer. The network still sees a diverse array of input states across which to generalise but is able to modify its predicted value in accordance with the known stage of training.

Figure 5.1 also shows that there is no extra benefit from combining importance sampling with fingerprints (XP+IS+FP). This makes sense given that the two approaches both address the same problem of nonstationarity, albeit in different ways.

Figure 5.2, which shows the estimated value for XP+FS of a single initial state observation with different $\epsilon$ inputs, demonstrates that the network learns to smoothly vary its value estimates across different stages of training, correctly associating high values with the low $\epsilon$ seen later in training. This approach allows the model to generalise between best responses to different policies of other agents. In effect, a larger dataset is available in this case than when using importance sampling, where most experiences are strongly discounted during training. The

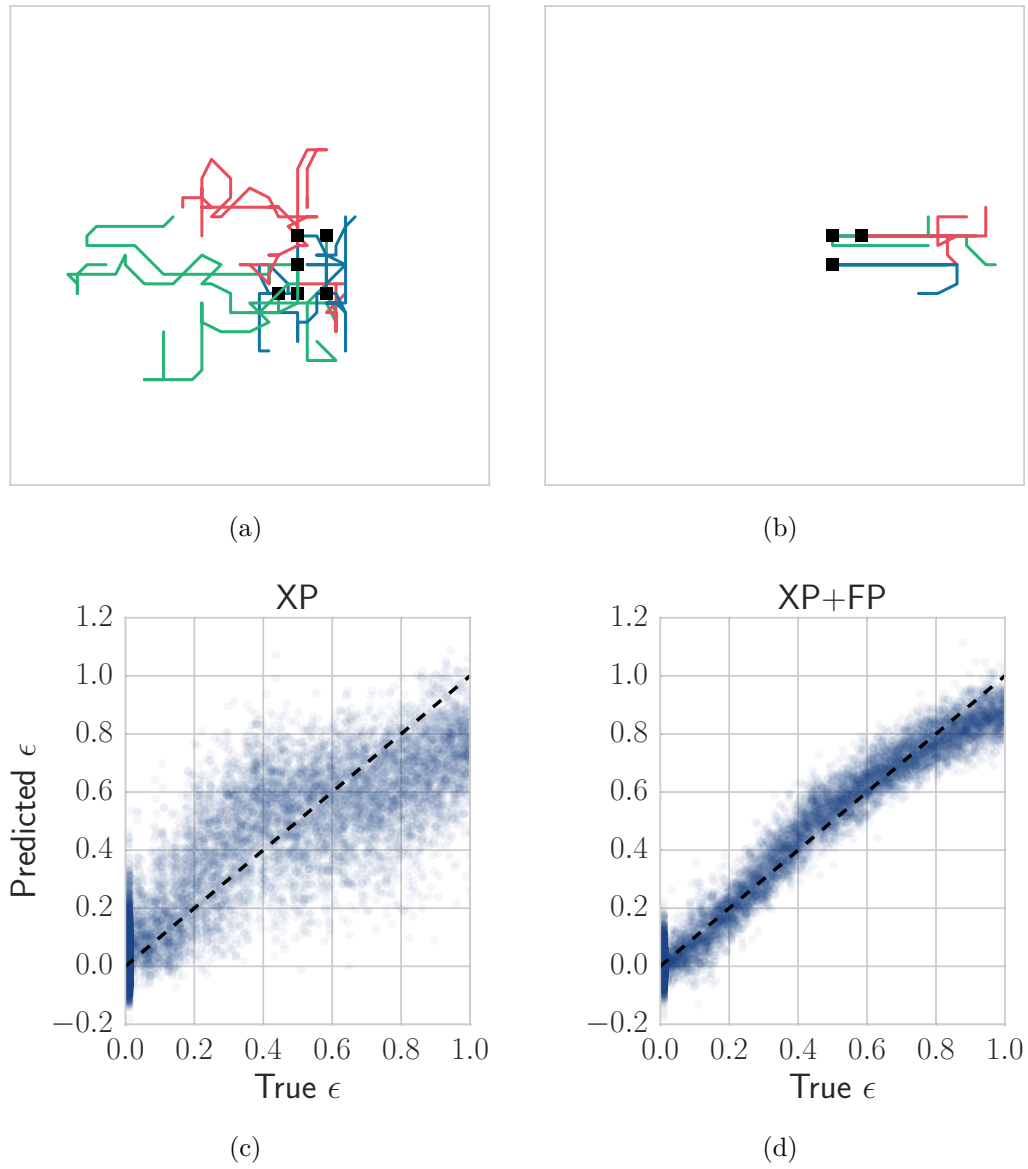



**Figure 5.3:** (upper) Sampled trajectories of agents, from the beginning (a) and end (b) of training. Each agent is one colour and the starting points are marked as black squares. (lower) Linear regression predictions of $\epsilon$ from the hidden state halfway through each episode in the replay buffer: (c) with only XP, the hidden state still contains disambiguating information drawn from the trajectories, (d) with XP+FP, the hidden state is more informative about the stage of training.

fingerprint enables the transfer of learning between diverse historical experiences, which can significantly improve performance.

### 5.5.3 Informative Trajectories

When using recurrent networks, the performance gains of XP+IS and XP+FP are not as large; in the 3v3 task, neither method helps. The reason is that, in StarCraft, the observed trajectories are significantly informative about the state of training, as shown in Figure 5.3a and 5.3b. For example, the agent can observe that it or its allies have taken many seemingly random actions, and infer that the sampled experience comes from early in training. This is a demonstration of the power of recurrent architectures in sequential tasks with partial observability: even without explicit additional information, the network is able to partially disambiguate experiences from different stages of training. To illustrate this, we train a linear model to predict the training $\epsilon$ from the hidden state of the recurrent model. Figure 5.3c shows a reasonably strong predictive accuracy even for a model trained with XP but no fingerprint, indicating that disambiguating information is indeed kept in the hidden states. However, the hidden states of a recurrent model trained with a fingerprint (Figure 5.3d) are even more informative.

## 5.6 Conclusion & Future Work

This chapter proposed two methods for stabilising experience replay in deep multi-agent reinforcement learning: 1) using a multi-agent variant of importance sampling to naturally decay obsolete data and 2) conditioning each agent's value function on a fingerprint that disambiguates the age of the data sampled from the replay memory. Results on a challenging decentralised variant of StarCraft unit micromanagement confirmed that these methods enable the successful combination of experience replay with multiple agents. In the future, we would like to apply these methods to a broader range of nonstationary training problems, such as classification on changing data, and extend them to multi-agent actor-critic methods. So far we have assumed that the agents have to execute their policies fully decentralised, *i.e.*, without having access to a communication channel. In the next part of the thesis we will investigate methods that allow agents to learn to use limited bandwidth

communication channels when they are available and to learn to communicate through informative actions when those are observable by their teammates.

# Part II

# Learning to Communicate

# Abstract

So far we have been focussed on settings which do not require implicit or explicit communication between the agents. In this part of the thesis we investigate different methods for learning communication protocols using DMARL. In Chapter 6 we investigate how to learn communication protocols in the presence of cheap-talk channels, *i.e.*, in settings where the messages sent do not have any direct impact on the reward or the transition probability. In these settings we can use differentiation across the communication channel in order to learn what messages to send. In contrast, in Chapter 7 we investigate a setting in which agents need to communicate through grounded hint-actions and by making their actions themselves informative, when observed by another agent. We solve this by allowing agents to reason over the beliefs of other agents in the environment.

# 6

# Learning to Communicate with Deep Multi-Agent Reinforcement Learning

## Contents



## 6.1 Introduction

So far we investigated settings which do not require any implicit or explicit communication. However, communication is an important feature of multi-agent settings. How language and communication emerge among intelligent agents also has long been a topic of intense debate. Among the many unresolved questions

are: Why does language use discrete structures? What role does the environment play? What is innate and what is learned? And so on. Some of the debates on these questions have been so fiery that in 1866 the French Academy of Sciences banned publications about the origin of human language.

The rapid progress in recent years of machine learning, and deep learning in particular, opens the door to a new perspective on this debate. How can agents use machine learning to automatically discover the communication protocols they need to coordinate their behaviour? What, if anything, can deep learning offer to such agents? What insights can we glean from the success or failure of agents that learn to communicate?

In this chapter, we take the first steps towards answering these questions. Our approach is programmatic: first, we propose a set of multi-agent benchmark tasks that require communication; then, we formulate several learning algorithms for these tasks; finally, we analyse how these algorithms learn, or fail to learn, communication protocols for the agents.

The tasks that we consider are fully cooperative, partially observable, sequential multi-agent decision making problems. All the agents share the goal of maximising the same discounted sum of rewards. While no agent can observe the underlying Markov state, each agent receives a private observation correlated with that state. In addition to taking actions that affect the environment, each agent can also communicate with its fellow agents via a discrete limited-bandwidth cheap-talk channel. Due to the partial observability and limited channel capacity, the agents must discover a communication protocol that enables them to coordinate their behaviour and solve the task.

As we do in the rest of the thesis, here we focus on settings with *centralised learning* but *decentralised execution*. In other words, communication between agents is not restricted during learning, which is performed by a centralised algorithm; however, during execution of the learned policies, the agents can communicate only via the limited-bandwidth channel. While not all real-world problems can be solved in this way, a great many can, e.g., when training a group of robots on a simulator.

Centralised planning and decentralised execution is also a standard paradigm for multi-agent planning [60]. For completeness, we also provide decentralised learning baselines.

To address these tasks, we formulate two approaches. The first, named *reinforced inter-agent learning* (RIAL), uses deep $Q$-learning [13] with a recurrent network to address partial observability. In one variant of this approach, *independent Q-learning*, the agents each learn their own network parameters, treating the other agents as part of the environment. Another variant trains a single network whose parameters are shared among all agents. Execution remains decentralised, at which point they receive different observations leading to different behaviour.

The second approach, which we call *differentiable inter-agent learning* (DIAL), is based on the insight that centralised learning affords more opportunities to improve learning than just parameter sharing. In particular, while RIAL is end-to-end trainable *within* an agent, it is not end-to-end trainable *across* agents, i.e., no gradients are passed between agents. The second approach allows real-valued messages to pass between agents during centralised learning, thereby treating communication actions as bottleneck connections between agents. As a result, gradients can be pushed through the communication channel, yielding a system that is end-to-end trainable even across agents. During decentralised execution, real-valued messages are discretised and mapped to the discrete set of communication actions allowed by the task. Because DIAL passes gradients from agent to agent, it is an inherently deep learning approach.

Our empirical study shows that these methods can solve our benchmark tasks, often discovering elegant communication protocols along the way. To our knowledge, this is the first time that either differentiable communication or reinforcement learning (RL) with deep neural networks have succeeded in learning communication protocols in complex environments involving sequences and raw input images. The results also show that deep learning, by better exploiting the opportunities of centralised learning, is a uniquely powerful tool for learning communication protocols. Finally, this study advances several engineering innovations, outlined in

the experimental section, that are essential for learning communication protocols in our proposed benchmarks.

## 6.2 Related Work

Research on communication spans many fields, e.g. linguistics, psychology, evolution and AI. In AI, it is split along a few axes: a) predefined or learned communication protocols, b) planning or learning methods, c) evolution or RL, and d) cooperative or competitive settings.

Given the topic of this chapter, here we focus on related work that deals with the cooperative learning of communication protocols. Out of the plethora of work on multi-agent RL with communication, *e.g.*, [25, 93–96], only a few fall into this category. Most assume a pre-defined communication protocol, rather than trying to learn protocols.

One exception is the work of Kasai, Tenmoto, and Kamiya [96], in which tabular Q-learning agents have to learn the content of a message to solve a predator-prey task with communication. Another example of open-ended communication learning in a multi-agent task is given in [97]. Here evolutionary methods are used for learning the protocols which are evaluated on a similar predator-prey task. Their approach uses a fitness function that is carefully designed to accelerate learning. In general, heuristics and handcrafted rules have prevailed widely in this line of research. Moreover, typical tasks have been necessarily small so that global optimisation methods, such as evolutionary algorithms, can be applied. The use of deep representations and gradient-based optimisation as advocated in this chapter is an important departure, essential for scalability and further progress. A similar rationale is provided in [98], another example of making an RL problem end-to-end differentiable.

Finally, we consider discrete communication channels. One of the key components of our methods is the signal binarisation during the decentralised execution. This is related to recent research on fitting neural networks in low-powered devices with memory and computational limitations using binary weights, e.g. [99], and previous works on discovering binary codes for documents [100].

## 6.3 Setting

Like the previous chapters, we consider cooperative RL problems with both multiple agents and partial observability. All the agents share the goal of maximising the same discounted sum of rewards $R_t$. While no agent can observe the underlying Markov state $s_t$, each agent receives a private observation $o_t^a$ correlated to $s_t$. However, in contrast to previous chapters, the environment includes a cheap-talk channel **??**: In each time-step, the agents select an *environment action* $u \in U$ that affects the environment, and a *communication action* $m \in M$ that is observed by other agents but has no direct impact on the environment or reward. We are interested in such settings because it is only when multiple agents and partial observability coexist that agents have the incentive to communicate. As no communication protocol is given a priori, the agents must develop and agree upon such a protocol to solve the task.

Since protocols are mappings from action-observation histories to sequences of messages, the space of protocols is extremely high-dimensional. Automatically discovering effective protocols in this space remains an elusive challenge. In particular, the difficulty of exploring this space of protocols is exacerbated by the need for agents to coordinate the sending and interpreting of messages. For example, if one agent sends a useful message to another agent, it will only receive a positive reward if the receiving agent correctly interprets and acts upon that message. If it does not, the sender will be discouraged from sending that message again. Hence, positive rewards are sparse, arising only when sending and interpreting are properly coordinated, which is hard to discover via random exploration.

We also focus on the setting of *centralised learning* but *decentralised execution*. In other words, communication between agents is not restricted during learning, which is performed by a centralised algorithm; however, during execution of the learned policies, the agents can communicate only via the limited-bandwidth channel.

## 6.4 Methods

In this section, we present two approaches for learning communication protocols.

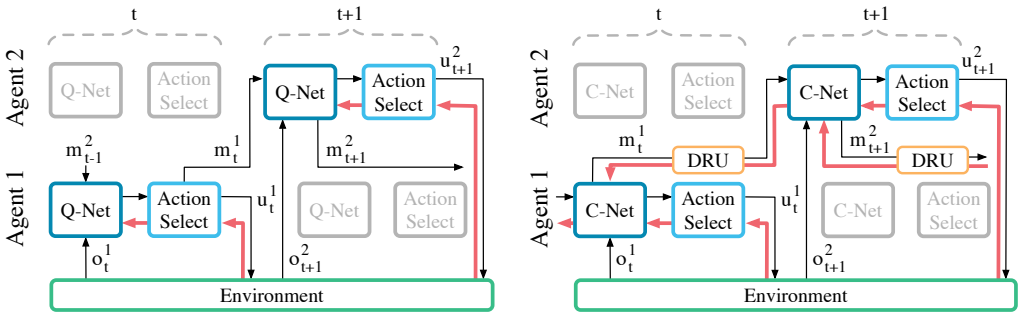
### 6.4.1 Reinforced Inter-Agent Learning

The most straightforward approach, which we call *reinforced inter-agent learning* (RIAL), is to combine DRQN with independent Q-learning for action and communication selection. Each agent's $Q$-network represents $Q^a(o_t^a, m_{t-1}^{a'}, h_{t-1}^a, u^a)$, which conditions on that agent's individual hidden state and observation.

To avoid needing a network with $|U||M|$ outputs, we split the network into $Q_u^a$ and $Q_m^a$, the Q-values for the environment and communication actions, respectively. Similarly to [101], the action selector separately picks $u_t^a$ and $m_t^a$ from $Q_u$ and $Q_m$, using an $\epsilon$-greedy policy. Hence, the network requires only $|U| + |M|$ outputs and action selection requires maximising over $U$ and then over $M$, but not maximising over $U \times M$.

Both $Q_u$ and $Q_m$ are trained using DQN with the following two modifications, which were found to be essential for performance. First, we disable experience replay to account for the nonstationarity that occurs when multiple agents learn concurrently, as it can render experience obsolete and misleading. Second, to account for partial observability, we feed in the actions $u$ and $m$ taken by each agent as inputs on the next time-step. Figure 6.1(a) shows how information flows between agents and the environment, and how Q-values are processed by the action selector in order to produce the action, $u_t^a$, and message $m_t^a$. Since this approach treats agents as independent networks, the learning phase is not centralised, even though our problem setting allows it to be. Consequently, the agents are treated exactly the same way during decentralised execution as during learning.

**Parameter Sharing.** RIAL can be extended to take advantage of the opportunity for centralised learning by sharing parameters among the agents. This variation learns only one network, which is used by all agents. However, the agents can still behave differently because they receive different observations and thus evolve different hidden states. In addition, each agent receives its own index $a$ as




(a) RIAL (b) DIAL

**Figure 6.1:** In RIAL (a), all Q-values are fed to the action selector, which selects both environment and communication actions. Gradients, shown in red, are computed using DQN for the selected action and flow only through the Q-network of a single agent. In DIAL (b), the message $m_t^a$ bypasses the action selector and instead is processed by the DRU (Section 6.4.2) and passed as a continuous value to the next C-network. Hence, gradients flow across agents, from the recipient to the sender. For simplicity, at each time step only one agent is highlighted, while the other agent is greyed out.

input, allowing them to specialise. The rich representations in deep Q-networks can facilitate the learning of a common policy while also allowing for specialisation. Parameter sharing also dramatically reduces the number of parameters that must be learned, thereby speeding learning. Under parameter sharing, the agents learn two $Q$-functions $Q_u(o_t^a, m_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, m_{t-1}^a, a, u_t^a)$ and $Q_m(\cdot)$, for $u$ and $m$, respectively, where $u_{t-1}^a$ and $m_{t-1}^a$ are the last action inputs and $m_{t-1}^{a'}$ are messages from other agents. During decentralised execution, each agent uses its own copy of the learned network, evolving its own hidden state, selecting its own actions, and communicating with other agents only through the communication channel.

### 6.4.2 Differentiable Inter-Agent Learning

While RIAL can share parameters among agents, it still does not take full advantage of centralised learning. In particular, the agents do not give each other feedback about their communication actions. Contrast this with human communication, which is rich with tight feedback loops. For example, during face-to-face interaction, listeners send fast nonverbal cues to the speaker indicating the level of understanding and interest. RIAL lacks this feedback mechanism, which is intuitively important for learning communication protocols.

To address this limitation, we propose *differentiable inter-agent learning* (DIAL). The main insight behind DIAL is that the combination of centralised learning and Q-networks makes it possible, not only to share parameters but to push gradients from one agent to another through the communication channel. Thus, while RIAL is end-to-end trainable *within* each agent, DIAL is end-to-end trainable *across* agents. Letting gradients flow from one agent to another gives them richer feedback, reducing the required amount of learning by trial and error, and easing the discovery of effective protocols.

DIAL works as follows: during centralised learning, communication actions are replaced with direct connections between the output of one agent's network and the input of another's. Thus, while the task restricts communication to discrete messages, during learning the agents are free to send real-valued messages to each other. Since these messages function as any other network activation, gradients can be passed back along the channel, allowing end-to-end backpropagation across agents.

In particular, the network, which we call a C-Net, outputs two distinct types of values, as shown in Figure 6.1(b), a) $Q(\cdot)$, the Q-values for the environment actions, which are fed to the action selector, and b) $m_t^a$, the real-valued message to other agents, which bypasses the action selector and is instead processed by the *discretise/regularise unit* (DRU($m_t^a$)). The DRU regularises it during centralised learning, DRU($m_t^a$) = Logistic($\mathcal{N}(m_t^a, \sigma)$), and discretises it during decentralised execution, DRU($m_t^a$) = $\mathbb{1}\{m_t^a > 0\}$, where $\sigma$ is the standard deviation of the noise added to the channel. Figure 6.1 shows how gradients flow differently in RIAL and DIAL. The gradient chains for $Q_u$, in RIAL and $Q$, in DIAL, are based on the DQN loss. However, in DIAL the gradient term for $m$ is the backpropagated error from the recipient of the message to the sender. Using this inter-agent gradient for training provides a richer training signal than the DQN loss for $Q_m$ in RIAL. While the DQN error is nonzero only for the selected message, the incoming gradient is a $|m|$-dimensional vector that can contain more information, here $|m|$ is the length of $m$. It also allows the network to directly adjust messages

in order to minimise the downstream DQN loss, reducing the need for trial and error exploration to learn good protocols.

While we limit our analysis to discrete messages, DIAL naturally handles continuous protocols, as they are part of the differentiable training. While we limit our analysis to discrete messages, DIAL naturally handles continuous message spaces, as they are used anyway during centralised learning. DIAL can also scale naturally to large discrete message spaces, since it learns binary encodings instead of the one-hot encoding in RIAL, $|m| = O(\log(|M|)$.

## 6.5 DIAL Details

Algorithm 3 formally describes DIAL. At each time-step, we pick an action for each agent $\epsilon$-greedily with respect to the $Q$-function and assign an outgoing message

$$Q(\cdot), m_t^a = \text{C-Net}\left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a; \theta_i\right). \tag{6.5.1}$$

We feed in the previous action, $u_{t-1}^a$, the agent index, $a$, along with the observation $o_t^a$, the previous internal state, $h_{t-1}^a$ and the incoming messages $\hat{m}_{t-1}^{a'}$ from other agents. After all agents have taken their actions, we query the environment for a state update and reward information.

When we reach the final time-step or a terminal state, we proceed to the backwards pass. Here, for each agent, $a$, and time-step, $j$, we calculate a target Q-value, $y_j^a$, using the observed reward, $r_t$, and the discounted target network. We then accumulate the gradients, $\nabla\theta$, by regressing the Q-value estimate

$$Q(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i), \tag{6.5.2}$$

against the target Q-value, $y_t^a$, for the action chosen, $u_t^a$. We also update the message gradient chain $\mu_t^a$ which contains the derivative of the downstream bootstrap error $\sum_{m,t'>t}\left(\Delta Q_{t+1}^{a'}\right)^2$ with respect to the outgoing message $m_t^a$.

To allow for efficient calculation, this sum can be broken out into two parts: The first part, $\sum_{m'\neq m}\frac{\partial}{\partial \hat{m}_t^a}\left(\Delta Q_{t+1}^{a'}\right)^2$, captures the impact of the message on the total estimation error of the next step. The impact of the message $m_t^a$ on all other future

**Algorithm 3** Differentiable Communication (DIAL)

Initialise $\theta_1$ and $\theta_1^-$
**for** each episode $e$ **do**
  $s_1$ = initial state, $t = 0$, $h_0^a = \mathbf{0}$ for each agent $a$
  **while** $s_t \neq$ terminal **and** $t < T$ **do**
    $t = t + 1$
    **for** each agent $a$ **do**
      Get messages $\hat{m}_{t-1}^{a'}$ of previous time-steps from agents $m'$ and evaluate C-Net:
      $Q(\cdot), m_t^a = \text{C-Net}\left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a; \theta_i\right)$
      With probability $\epsilon$ pick random $u_t^a$, else $u_t^a = \max_a Q\left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i\right)$
      Set message $\hat{m}_t^a = \text{DRU}(m)$, where $\text{DRU}(m) = \begin{cases} \text{Logistic}(\mathcal{N}(m, \sigma)), \text{ if training, else} \\ \mathbb{1}\{m > 0\} \end{cases}$
      Get reward $r_t$ and next state $s_{t+1}$
  Reset gradients $\nabla\theta = 0$
  **for** $t = T$ **to** 1, $-1$ **do**
    **for** each agent $a$ **do**
      $y_t^a = \begin{cases} r_t, \text{ if } s_t \text{ terminal, else} \\ r_t + \gamma \max_u Q\left(o_{t+1}^a, \hat{m}_t^{a'}, h_t^a, u_t^a, a, u; \theta_i^-\right) \end{cases}$
      Accumulate gradients for action:
      $\Delta Q_t^a = y_t^a - Q\left(o_j^a, h_{t-1}^a, \hat{m}_{t-1}^{a'}, u_{t-1}^a, a, u_t^a; \theta_i\right)$
      $\nabla\theta = \nabla\theta + \frac{\partial}{\partial\theta}(\Delta Q_t^a)^2$
      Update gradient chain for differentiable communication:
      $\mu_j^a = \mathbb{1}\{t < T - 1\} \sum_{m' \neq m} \frac{\partial}{\partial \hat{m}_t^a}\left(\Delta Q_{t+1}^{a'}\right)^2 + \mu_{t+1}^{a'} \frac{\partial \hat{m}_{t+1}^{a'}}{\partial \hat{m}_t^a}$
      Accumulate gradients for differentiable communication:
      $\nabla\theta = \nabla\theta + \mu_t^a \frac{\partial}{\partial m_t^a} \text{DRU}(m_t^a) \frac{\partial m_t^a}{\partial\theta}$
  $\theta_{i+1} = \theta_i + \alpha\nabla\theta$
  Every $C$ steps reset $\theta_i^- = \theta_i$

rewards $t' > t + 1$ can be calculated using the partial derivative of the outgoing messages from the agents at time $t + 1$ with respect to the incoming message $m_t^a$, multiplied with their message gradients, $\mu_{t+1}^{a'}$. Using the message gradient, we can calculate the derivative with respect to the parameters, $\mu_t^a \frac{\partial \hat{m}_t^a}{\partial\theta}$.

Having accumulated all gradients, we conduct two parameter updates, first $\theta_i$ in the direction of the accumulated gradients, $\nabla\theta$, and then every C steps $\theta_i^- = \theta_i$. During decentralised execution, the outgoing activations in the channel are mapped

into a binary vector, $\hat{m} = \mathbb{1}\{m_t^a > 0\}$. This ensures that discrete messages are exchanged, as required by the task.

In order to minimise the discretisation error when mapping from continuous values to discrete encodings, two measures are taken during centralised learning. First, Gaussian noise is added in order to limit the number of bits that can be encoded in a given range of $m$ values. Second, the noisy message is passed through a logistic function to restrict the range available for encoding information. Together, these two measures regularise the information transmitted through the bottleneck.

Furthermore, the noise also perturbs values in the middle of the range, due to the steeper slope, but leaves the tails of the distribution unchanged. Formally, during centralised learning, $m$ is mapped to $\hat{m} = \text{Logistic}\left(\mathcal{N}(m, \sigma)\right)$, where $\sigma$ is chosen to be comparable to the width of the logistic function. In Algorithm 3, the mapping logic from $m$ to $\hat{m}$ during training and execution is contained in the $\text{DRU}(m_t^a)$ function.

## 6.6 Experiments

In this section, we evaluate RIAL and DIAL with and without parameter sharing in two multi-agent problems and compare it with a no-communication shared parameters baseline (NoComm). Results presented are the average performance across several runs, where those without parameter sharing (-NS), are represented by dashed lines. Across plots, rewards are normalised by the highest average reward achievable given access to the true state (Oracle).

In our experiments, we use an $\epsilon$-greedy policy with $\epsilon = 0.05$, the discount factor is $\gamma = 1$, and the target network is reset every 100 episodes. To stabilise learning, we execute parallel episodes in batches of 32. The parameters are optimised using RMSProp with momentum of 0.95 and a learning rate of $5 \times 10^{-4}$. The architecture makes use of *rectified linear units* (ReLU), and *gated recurrent units* (GRU) [38], which have similar performance to *long short-term memory* [102] (LSTM) [103, 104]. Unless stated otherwise we set $\sigma = 2$, which was found to be essential for good performance. We intent to published the source code online.

### 6.6.1 Model Architecture

RIAL and DIAL share the same individual model architecture. For brevity, we describe only the DIAL model here. As illustrated in Figure 6.2, each agent consists of a recurrent neural network (RNN), unrolled for $T$ time-steps, that maintains an internal state $h$, an input network for producing a task embedding $z$, and an output network for the $Q$-values and the messages $m$. The input for agent $a$ is defined as a tuple of $(o_t^a, m_{t-1}^{a'}, u_{t-1}^a, a)$. The inputs $a$ and $u_{t-1}^a$ are passed through lookup tables, and $m_{t-1}^{a'}$ through a 1-layer MLP, both producing embeddings of size 128. $o_t^a$ is processed through a task-specific network that produces an additional embedding of the same size. The state embedding is produced by element-wise summation of these embeddings, $z_t^a = \Big(\text{TaskMLP}(o_t^a) + \text{MLP}[|M|, 128](m_{t-1}) + \text{Lookup}(u_{t-1}^a) + \text{Lookup}(a)\Big)$. We found that performance and stability improved when a batch normalisation layer [105] was used to preprocess $m_{t-1}$. $z_t^a$ is processed through a 2-layer RNN with GRUs, $h_{1,t}^a = \text{GRU}[128, 128](z_t^a, h_{1,t-1}^a)$, which is used to approximate the agent's action-observation history. Finally, the output $h_{2,t}^a$ of the top GRU layer, is passed through a 2-layer MLP $Q_t^a, m_t^a = \text{MLP}[128, 128, (|U| + |M|)](h_{2,t}^a)$.

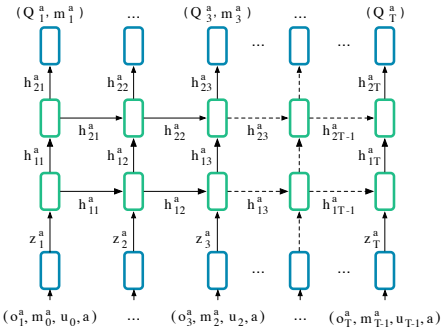


**Figure 6.2:** DIAL architecture.

### 6.6.2 Switch Riddle

The first task is inspired by a well-known riddle described as follows: *"One hundred prisoners have been newly ushered into prison. The warden tells them that starting tomorrow, each of them will be placed in an isolated cell, unable to communicate amongst each other. Each day, the warden will choose one of the prisoners uniformly at random with replacement, and place him in a central interrogation room containing only a light bulb with a toggle switch. The prisoner will be able to observe the current state of the light bulb. If he wishes, he can toggle the light bulb. He also has the option of announcing that he believes all prisoners have visited the interrogation*

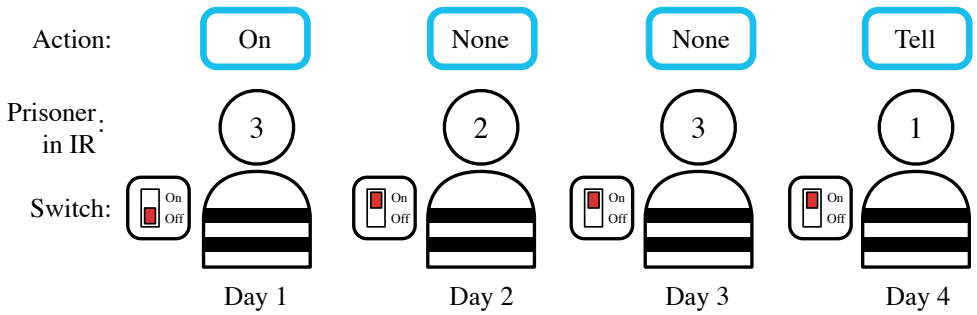



**Figure 6.3:** *Switch:* Every day one prisoner gets sent to the interrogation room where he sees the switch and chooses from "On", "Off", "Tell" and "None".

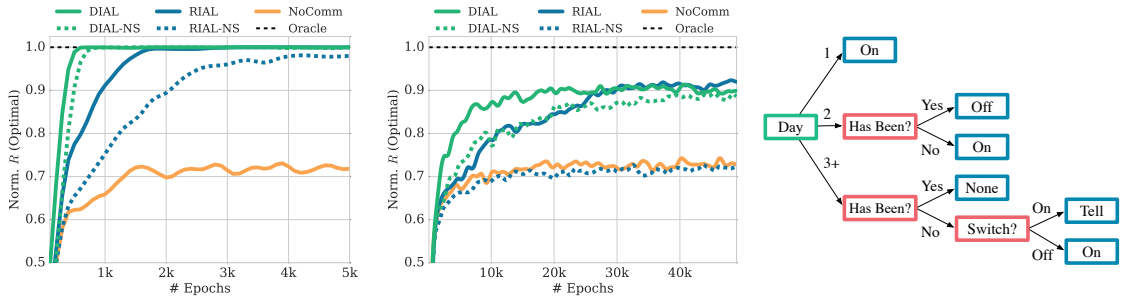
*room at some point in time. If this announcement is true, then all prisoners are set free, but if it is false, all prisoners are executed. The warden leaves and the prisoners huddle together to discuss their fate. Can they agree on a protocol that will guarantee their freedom?"* [106].

**Architecture.** In our formalisation, at time-step $t$, agent $a$ observes $o_t^a \in 0, 1$, which indicates if the agent is in the interrogation room. Since the switch has two positions, it can be modelled as a 1-bit message, $m_t^a$. If agent $a$ is in the interrogation room, then its actions are $u_t^a \in \{$"None","Tell"$\}$; otherwise the only action is "None". The episode ends when an agent chooses "Tell" or when the maximum time-step, $T$, is reached. The reward $r_t$ is 0 unless an agent chooses "Tell", in which case it is 1 if all agents have been to the interrogation room and $-1$ otherwise. Following the riddle definition, in this experiment $m_{t-1}^a$ is available only to the agent $a$ in the interrogation room. Finally, we set the time horizon $T = 4n - 6$ in order to keep the experiments computationally tractable.

**Complexity.** The switch riddle poses significant protocol learning challenges. At any time-step $t$, there are $|o|^t$ possible observation histories for a given agent, with $|o| = 3$: the agent either is not in the interrogation room or receives one of two messages when he is. For each of these histories, an agent can chose between $4 = |U||M|$ different options, so at time-step $t$, the single-agent policy space is $(|U||M|)^{|o|^t} = 4^{3^t}$. The product of all policies for all time-steps defines the total policy space for an agent: $\prod 4^{3^t} = 4^{(3^{T+1}-3)/2}$, where $T$ is the final time-step.

The size of the multi-agent policy space grows exponentially in $n$, the number of agents: $4^{n(3^{T+1}-3)/2}$. We consider a setting where $T$ is proportional to the number of agents, so the total policy space is $4^{n3^{O(n)}}$. For $n = 4$, the size is $4^{354288}$. Our approach using DIAL is to model the switch as a continuous message, which is binarised during decentralised execution.

**Experimental results.** Figure 6.4(a) shows our results for $n = 3$ agents. All four methods learn an optimal policy in 5k episodes, substantially outperforming the NoComm baseline. DIAL with parameter sharing reaches optimal performance substantially faster than RIAL. Furthermore, parameter sharing speeds both methods. Figure 6.4(b) shows results for $n = 4$ agents. DIAL with parameter sharing again outperforms all other methods. In this setting, RIAL without parameter sharing was unable to beat the NoComm baseline. These results illustrate how difficult it is for agents to learn the same protocol independently. Hence, parameter sharing can be crucial for learning to communicate. DIAL-NS performs similarly to RIAL, indicating that the gradient provides a richer and more robust source of information.




(a) n=3 (b) n=4 (c) n=3 protocol

**Figure 6.4:** *Switch:* (a-b) Performance of DIAL and RIAL, with and without ( -NS) parameter sharing, and NoComm-baseline, for $n = 3$ and $n = 4$ agents. (c) The decision tree extracted for $n = 3$ to interpret the communication protocol discovered by DIAL

We also analysed the communication protocol discovered by DIAL for $n = 3$ by sampling 1K episodes, for which Figure 6.4(c) shows a decision tree corresponding to an optimal strategy. When a prisoner visits the interrogation room after day two, there are only two options: either one or two prisoners may have visited the room

r

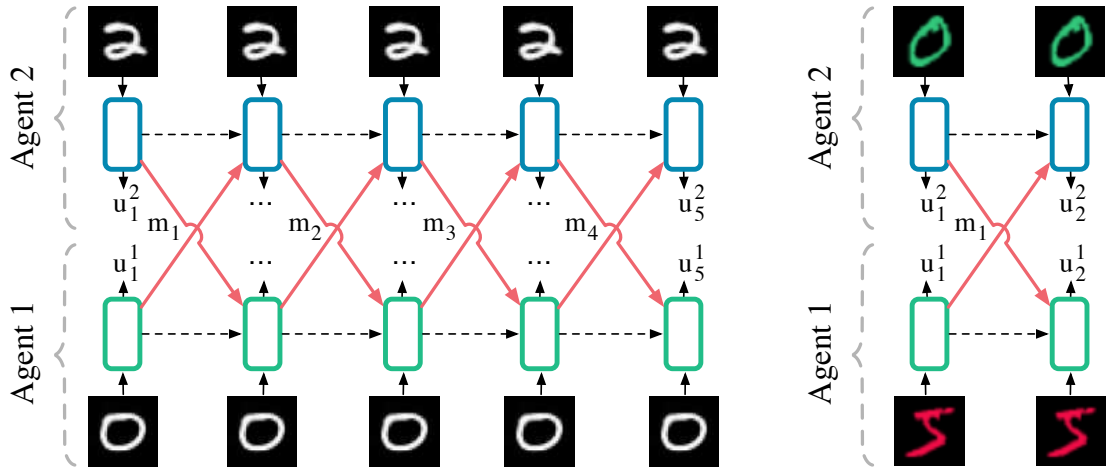



**Figure 6.5:** MNIST games architectures.

before. If three prisoners had been, the third prisoner would have finished the game. The other options can be encoded via the "On" and "Off" position respectively.

### 6.6.3 MNIST Games

In this section, we consider two tasks based on the well known MNIST digit classification dataset [7].

**Colour-Digit MNIST** is a two-player game in which each agent observes the pixel values of a random MNIST digit in red or green of size $2 \times 28 \times 28$, while the colour label, $c^a \in 0, 1$, and digit value, $d^a \in 0..9$, are hidden. For each agent, reward consists of two components that are antisymmetric in the action, colour, and parity (odd, even) of the digits. Only one bit of information can be sent, so agents must agree to encode/decode either colour or parity, with parity yielding greater rewards. The game has two steps; in the first step, both agents send a 1-bit message, in the second step they select a binary action $u_2^a$. The reward for each agent is $r(a) = 2(-1)^{a_2^a + c^a + d^{a'}} + (-1)^{a_2^a + d^a + c^{a'}}$ and the total cooperative reward is $r_2 = r(1) + r(2)$.

**Multi-Step MNIST** is a grayscale variant that requires agents to develop a communication protocol which integrates information across many time-steps: Each step the agents send a message, $m_t^a$, and take an action $u_t^a \in \{0, \ldots, 9\}$.

Only at the final step, $t = 5$, is reward given, $r_5 = 0.5$ for each correctly guessed digit, $u_5^a = d^{a'}$. As only 1-bit is sent per step, agents must find a protocol that integrates information across the four messages they exchange (the last message is not received). The protocol can be trained using gradients in DIAL, but also needs to have a low discretisation error.

**Architecture.** The input processing network is a 2-layer MLP TaskMLP$[(|c| \times 28 \times 28), 128, 128](o_t^a)$. Figure 6.5 depicts the generalised setting for both games. Our experimental evaluation showed improved training time using batch normalisation after the first layer.

**Experimental results.** Figures 6.6(a) and 6.6(b) show that DIAL substantially outperforms the other methods on both games. Furthermore, parameter sharing is crucial for reaching the optimal protocol. In multi-step MNIST, results were obtained with $\sigma = 0.5$. In this task, RIAL fails to learn, while in colour-digit MNIST it fluctuates around local minima in the protocol space; the NoComm baseline is stagnant at zero. DIAL's performance can be attributed to directly optimising the messages in order to reduce the global DQN error while RIAL must rely on trial and error. DIAL can also optimise the message content with respect to rewards taking place many time-steps later, due to the gradient passing between agents, leading to optimal performance in multi-step MNIST. To analyse the protocol that DIAL learned, we sampled 1K episodes. Figure 6.6(c) illustrates the communication bit sent at time-step $t$ by agent 1, as a function of its input digit. Thus, each agent has learned a binary encoding and decoding of the digits. These results illustrate that differentiable communication in DIAL is essential to fully exploiting the power of centralised learning and thus is an important tool for studying the learning of communication protocols.

Our results show that DIAL deals more effectively with the high dimensional input space in the *colour-digit MNIST* game than RIAL. To better understand why, as a thought-experiment, consider a simpler two-agent problem with a structurally similar reward function $r = (-1)^{(s^1+s^2+u^2)}$, which is antisymmetric in the observations and action of the agents. Here random digits $s^1, s^2 \in 0, 1$ are input

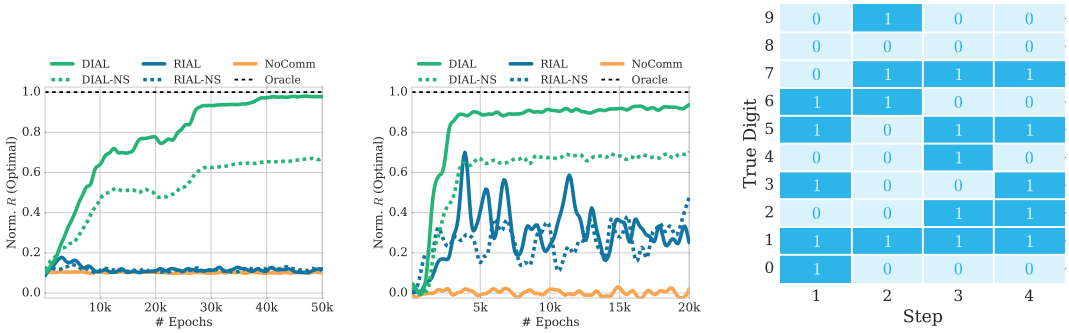



**Figure 6.6:** *MNIST Games:* (a,b) Performance of DIAL and RIAL, with and without (-NS) parameter sharing, and NoComm, for both MNIST games. (c) Extracted coding scheme for multi-step MNIST.

to agent 1 and agent 2 and $u^2 \in 1, 2$ is a binary action. Agent 1 can send a single bit message, $m^1$. Until a protocol has been learned, the average reward for any action by agent 2 is 0, since averaged over $s_1$ the reward has an equal probability of being $+1$ or $-1$. Equally the TD error for agent 1, the sender, is zero for any message $m$:

$$\mathbb{E}\left[\Delta Q(s^1, m^1)\right] = Q(s^1, m^1) - \mathbb{E}\left[r(s^2, u^2, s^1)\right]_{s^2,u^2} = 0 - 0, \tag{6.6.1}$$

By contrast, DIAL allows for learning. Unlike the TD error, the gradient is a function of the action and the observation of the receiving agent, so summed across different $+1/-1$ outcomes the gradient updates for the message $m$ no longer cancel:

$$\mathbb{E}\left[\nabla\theta\right] = \mathbb{E}\left[\left(Q(s^2, m^1, u^2) - r(s^2, u^2, s^1)\right)\frac{\partial}{\partial m}Q(s^2, m^1, u^2)\frac{\partial}{\partial\theta}m^1(s^1)\right]_{<s^2,u^2>}. \tag{6.6.2}$$

The question of why language evolved to be discrete has been studied for centuries, see *e.g.*, the overview in [107]. Since DIAL learns to communicate in a continuous channel, our results offer an illuminating perspective on this topic.

In particular, Figure 6.7 shows that, in the switch riddle, DIAL without noise in the communication channel learns centred activations. By contrast, the presence of noise forces messages into two different modes during learning. Similar observations have been made in relation to adding noise when training document models [100] and performing classification [99]. In our work, we found that adding noise was essential for successful training.

### 6.6.4 Effect of Channel Noise

Given that the amount of noise, $\sigma$, is a hyperparameter that needs to be set, it is useful to understand how it impacts the amount of information that can pass through the channel. A first intuition can be gained by looking at the width of the sigmoid: Taking the decodable range of the logistic function to be $x$ values corresponding to $y$ values between 0.01 and 0.99, an initial estimate for the range is $\approx 10$. Thus, requiring distinct $x$ values to be at least six standard deviations apart, with $\sigma = 2$, only two bits can be encoded reliably in this range. To get a better understanding of the required $\sigma$ we can visualise the capacity of the channel including the logistic function and the Gaussian noise. To do so, we must first derive an expression for the probability distribution of outgoing messages, $\hat{m}$, given incoming activations, $m$, $P(\hat{m}|m)$:

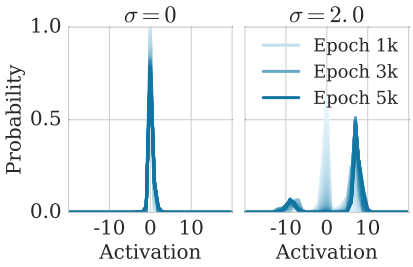



**Figure 6.7:** DIAL's learned activations with and without noise in DRU.

$$P(\hat{m}|m) = \frac{1}{\sqrt{2\pi\sigma}\hat{m}(1-\hat{m})} \exp\left(-\frac{\left(m - \log(\frac{1}{\hat{m}} - 1)\right)^2}{\sigma^2}\right). \tag{6.6.3}$$

For any $m$, this captures the distribution of messages leaving the channel. Two $m$ values $m_1$ and $m_2$ can be distinguished when the outgoing messages have a small probability of overlapping. Given a value $m_1$ we can thus pick a next value $m_2$ to be distinguishable when the highest value $\hat{m}_1$ that $m_1$ is likely to produce is less than the lowest value $\hat{m}_2$ that $m_2$ is likely to produce. An approximation for when this happens is when $(\max_{\hat{m}} s.t. P(\hat{m}|m_1) > \epsilon) = (\min_{\hat{m}} s.t. P(\hat{m}|m_2) > \epsilon)$. Figure 6.8 illustrates this for three different values of $\sigma$. For $\sigma > 2$, only two options can be reliably encoded using $\epsilon = 0.1$, resulting in a channel that effectively transmits only one bit of information.

Interestingly, the amount of noise required to regularise the channel depends greatly on the benefits of over-encoding information. More specifically, as illustrated

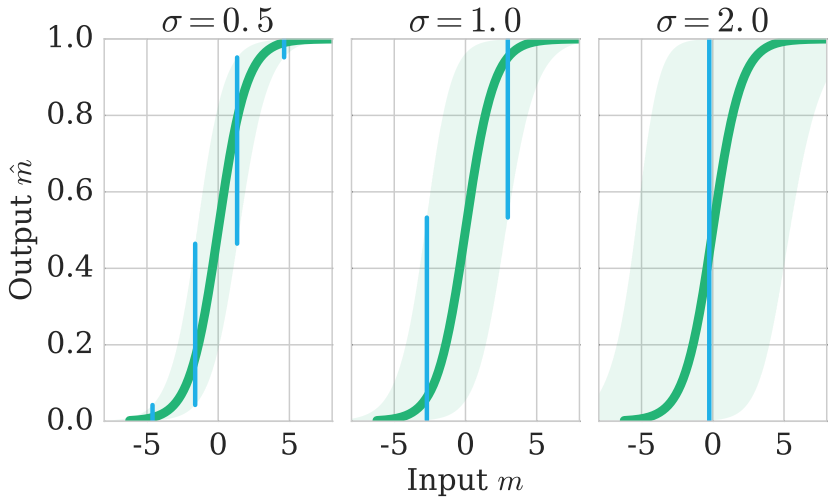



**Figure 6.8:** Distribution of regularised messages, $P(\hat{m}|m)$ for different noise levels. Shading indicates $P(\hat{m}|m) > 0.1$. Blue bars show a division of the $x$-range into intervals s.t. the resulting $y$-values have a small probability of overlap, leading to decodable values.

in Figure 6.9, in tasks where sending more bits does not lead to higher rewards, small amounts of noise are sufficient to encourage discretisation, as the network can maximise reward by pushing activations to the tails of the sigmoid, where the noise is minimised. The figure illustrates the final average evaluation performance normalised by the training performance of three runs after 50K of the multi-step MNIST game, under different noise regularisation levels $\sigma \in \{0, 0.5, 1, 1.5, 2\}$, and different numbers of steps $step \in [2, \ldots, 5]$. When the lines exceed"Regularised", the test reward, after discretisation, is higher than the training reward, i.e., the channel is properly regularised and getting used as a single bit at the end of learning. Given that there are 10 digits to encode, four bits are required to get full rewards. Reducing the number of steps directly reduces the number of bits that can be communicated, $\#bits = steps - 1$, and thus creates an incentive for the network to "over-encode" information in the channel, which leads to greater discretisation error. This is confirmed by the normalised performance for $\sigma = 0.5$, which is around 0.7 for 2 steps (1 bit) and then goes up to $> 1$ for 5 steps (4 bits). We also note that, without noise, regularisation is not possible and that with enough noise the channel is always regularised, even if it would yield higher training rewards to over-encode information.

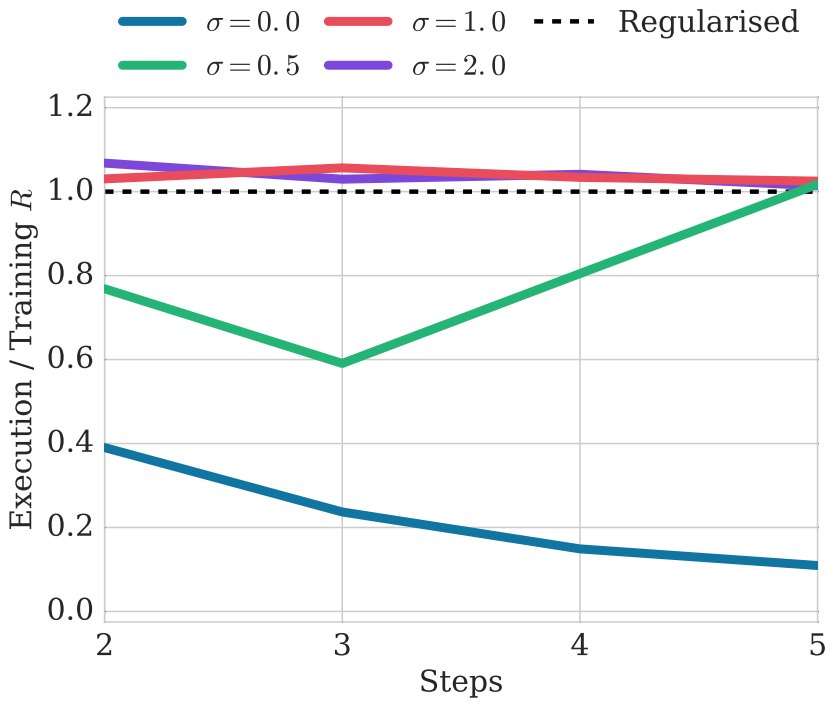



**Figure 6.9:** Final evaluation performance on multi-step MNIST of DIAL normalised by training performance after 50K epochs, under different noise regularisation levels $\sigma \in \{0, 0.5, 1, 1.5, 2\}$, and different numbers of steps $step \in [2, \ldots, 5]$.

## 6.7 Conclusion & Future Work

This chapter advanced novel environments and successful techniques for learning communication protocols. It presented a detailed comparative analysis covering important factors involved in the learning of communication protocols with deep networks, including differentiable communication, neural network architecture design, channel noise, tied parameters, and other methodological aspects.

This chapter should be seen as a first attempt at learning communication and language with deep learning approaches. The gargantuan task of understanding communication and language in their full splendour, covering compositionality, concept lifting, conversational agents, and many other important problems still lies ahead. We are however optimistic that the approaches proposed in this paper can play a substantial role in tackling these challenges. In the next chapter we address settings in which agents have to communicate through their actions, rather than relying on cheap-talk channels.

# 7

# Bayesian Action Decoder

## Contents

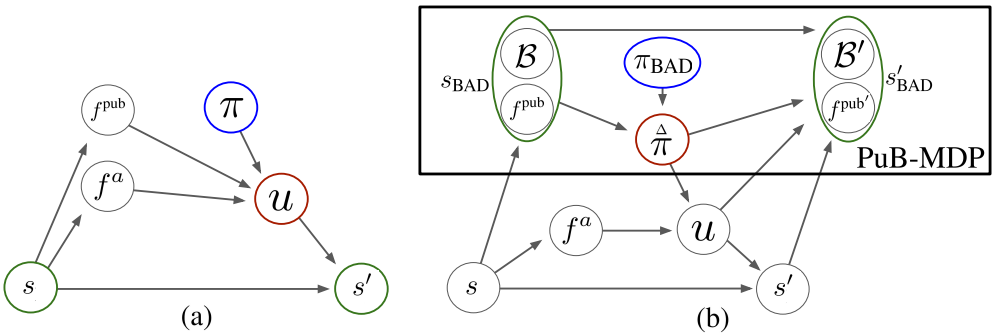



**Figure 7.1:** a) In an MDP the action $u$ is sampled from a policy $\pi$ that conditions on the state features (here separated into $f^{\text{pub}}$ and $f^a$). The next state is sampled from $P(s'|s,u)$. b) In a PuB-MDP, public features $f^{\text{pub}}$ generated by the environment and the public belief together constitute the Markov state $s_{\text{BAD}}$. The 'action' sampled by the BAD agent is in fact a deterministic partial policy $\hat{\pi} \sim \pi_{\text{BAD}}(\hat{\pi}|s_{\text{BAD}})$ that maps from private observations $f^a$ to actions. Only the acting agent observes $f^a$ and deterministically computes $u = \hat{\pi}(f^a)$. $u$ is provided to the environment, which transitions to state $s'$ and produces the new observation $f^{\text{pub}'}$. BAD then uses the public belief update to compute a new belief $\mathcal{B}'$ conditioned on $u$ and $\hat{\pi}$ (Equation 7.3.1), thereby completing the state transition.

## 7.1 Introduction

State-of-the-art DMARL methods for learning communication protocols, *e.g.*. DIAL proposed in the previous chapter, mostly use backpropagation across a communication channel (also see [48]). This approach has two limitations. First, it can only be applied to *cheap-talk* channels in which the communication action has no effect on the environment. Second, it misses the conceptual connection between communication and reasoning over the beliefs of others, which is known to be important to how humans learn to communicate [108, 109].

A well-known domain that highlights these challenges is Hanabi, a popular, fully cooperative card game of incomplete information that is difficult even for humans (Hanabi won the *Spiel des Jahres* award in 2013). A distinguishing feature of the game is that players can see everyone's hands but their own. To succeed, players must find effective conventions for communication. Since there is no cheap-talk channel, most recent methods for emergent communication are inapplicable, necessitating a novel approach. Taking advantage of these unique features, Hanabi has recently been established as a new benchmark for multi-agent learning [**bard2019hanabi**].

The goal in Hanabi is to play a legal sequence of cards and, to aid this process, players are allowed to give each other hints indicating which cards are of a specific rank or colour. These hints have two levels of semantics. The first level is the surface-level content of the hint, which is grounded in the properties of the cards that they describe. This level of semantics is independent of any possible intent of the agent in providing the hint, and would be equally meaningful if provided by a random agent. For example, knowing which cards are of a specific colour often does not indicate whether they can be safely played or discarded.

A second level of semantics arises from information contained in the actions themselves, i.e., the very fact that an agent decided to take a particular action and not another, rather than the information resulting from the state transition induced by the action. This is essential to the formation of conventions and to discovering good strategies in Hanabi.

To address these challenges, we propose the *Bayesian action decoder* (BAD), a novel multi-agent RL algorithm for discovering effective communication protocols and policies in cooperative, partially observable multi-agent settings. Inspired by the work of Nayyar, Mahajan, and Teneketzis [110], BAD uses all publicly observable features in the environment to compute a *public belief* over the players' private features. This effectively defines a new Markov process, the *public belief Markov decision process* (PuB-MDP), in which the action space is the set of deterministic partial policies, parameterised by deep neural networks, that can be sampled for a given public state. By acting in the space of deterministic partial policies that map from private observations into environment actions, an agent acting only on this public belief state can still learn an optimal policy. Using approximate, factorised Bayesian updates and deep neural networks, we show for the first time how a method using the public belief of Nayyar, Mahajan, and Teneketzis [110], can scale to large state spaces and allow agents to carry out a form of counterfactual reasoning.

When an agent observes the action of another agent, the public belief is updated by sampling a set of possible private states from the public belief and filtering for those states in which the teammate chose the observed action. This process is closely

related to the kind of *theory of mind* reasoning that humans routinely undertake [111]. Such reasoning seeks to understand why a person took a specific action among several, and what information this contains about the distribution over private observations.

We experimentally validate an exact version of BAD on a simple two-step matrix game, showing that it outperforms policy gradient methods. We then apply an approximate version to Hanabi, where BAD achieves an average score of 24.174 points in the two-player setting, surpassing the best previously published results for learning agents by around 9 points and approaching the best known performance of 24.9 points for (cheating) open-hand gameplay. BAD thus establishes a current state-of-the-art on the Hanabi-Learning-Environment [**bard2019hanabi**] for the two player self-play setting. We further show that the beliefs obtained via Bayesian reasoning have 40% less uncertainty over possible hands than those using only grounded information.

## 7.2 Setting

Here we consider a partially observable, cooperative multi-agent setting where the Markov state $s_t$ consists of a set of discrete features $f_t$ composed of public features $f_t^{\text{pub}}$, which are common knowledge to all agents, and private features $f_t^{\text{pri}}$. Each of the private features is observable by at least one, but not all, of the agents. $f_t^a$ are the private features observable by agent $a$. For example, in a typical card game the cards being played openly on the table are part of $f_t^{\text{pub}}$, while the cards being held by each player are in $f_t^{\text{pri}}$, with the cards visible to a particular agent in $f_t^a$. We assume that this separation of state features is common knowledge to all agents. An example of this separation for the case of an MDP is illustrated in Figure 7.1a.

## 7.3 Method

Below we introduce the *Bayesian Action Decoder* (BAD). BAD scales the public belief of Nayyar, Mahajan, and Teneketzis [110] to large state spaces using

factorised beliefs, an approximate Bayesian update, and sampled deterministic policies parameterised by deep neural networks.

### 7.3.1 Public belief

In single-agent partially observable settings, it is clearly useful for an agent to maintain beliefs about the hidden environment state, since this is a sufficient statistic for its action-observation history [112]. In multi-agent settings, however, it is not obvious what the beliefs should be over. It is not enough to maintain beliefs over the environment state alone, as other agents also have unobservable internal states. In interactive POMDPs (I-POMDPs; Gmytrasiewicz and Doshi 2005), agents model each other's beliefs, beliefs over these beliefs, and so on, but this is often computationally intractable.

Fortunately, in our setting the common knowledge described above makes it possible to compute a *public belief*, [110], that makes the recursion of I-POMDPs unnecessary. In our case the public belief $\mathcal{B}_t$ is the posterior over all of the private state features given only the public features, i.e., $\mathcal{B}_t = P(f_t^{\text{pri}}|f_{\leq t}^{\text{pub}})$, where $\leq t$ indicates history: $f_{\leq t}^{\text{pub}} = (f_0^{\text{pub}}, .., f_t^{\text{pub}})$. Because $\mathcal{B}_t$ conditions only on publicly available information, it can be computed independently by every agent via a common algorithm, yielding the same result for all agents. Furthermore, since all agents know $f^{\text{pub}}$, it suffices for $\mathcal{B}_t$ to be a posterior over $f^{\text{pri}}$, not $f_t = \{f^{\text{pri}}, f^{\text{pub}}\}$.

While the public belief avoids recursive reasoning, it is not obvious how it can be used to guide behaviour: agents that condition their actions only on the public belief will never exploit their private observations. As Nayyar, Mahajan, and Teneketzis [110] propose, we can construct a special *public agent* whose policy $\pi_{\text{BAD}}$ conditions on the public observation and the public belief but which nonetheless can generate optimal behaviour.[1] This is possible because an action selected by $\pi_{\text{BAD}}$ specifies a *partial policy*, $\hat{\pi} : \{f^a\} \rightarrow \mathcal{U}$, for the acting agent, deterministically mapping private observations to environment actions. The sampling of a deterministic partial policy also addresses a fundamental tension in using policy gradients to

[1] $\pi_{\text{BAD}}$ conditions on the public observation because the public belief is a sufficient statistic for the public observation, but only over the private features.

learn communication protocols, namely, differentiation and exploration require high-entropy policies, while communication requires low-entropy policies. By sampling in the space of deterministic policies, both can be achieved.

Intuitively, the public agent can be viewed as a third party that can observe only the public observation and belief. While $\pi_{\text{BAD}}$ cannot observe the private state, it can tell each agent what to do for any private observation it might receive. Thus at each timestep, the public agent selects $\hat{\pi}$ based on $\mathcal{B}_t$ and $f_t^{\text{pub}}$; the acting agent then selects the action $u_t^a = \hat{\pi}(f^a)$ by supplying the private observation hidden from the public agent; the public agent then uses the observed action $u_t^a$ to construct the new belief $\mathcal{B}_{t+1}$.

### 7.3.2 Public Belief MDP

Since $\hat{\pi}$ and $u_t^a$ are public information, observing $u_t^a$ induces a posterior belief over the possible private state features $f_t^{\text{pri}}$ given by the *public belief update*:

$$P(f_t^a|u_t^a, \mathcal{B}_t, f_t^{\text{pub}}, \hat{\pi}) = \frac{P(u_t^a|f_t^a, \hat{\pi})P(f_t^a|\mathcal{B}_t, f_t^{\text{pub}})}{P(u_t^a|\mathcal{B}_t, f_t^{\text{pub}}, \hat{\pi})} \tag{7.3.1}$$

$$\propto \mathbb{1}(\hat{\pi}(f_t^a), u_t^a)P(f_t^a|\mathcal{B}_t, f_t^{\text{pub}}). \tag{7.3.2}$$

Using this Bayesian belief update, we can define a new Markov process, the *public belief MDP* (PuB-MDP), as illustrated in Figure 7.1b. The state $s_{\text{BAD}} \in S_{\text{BAD}}$ of the PuB-MDP consists of the public observation and public belief; the action space is the set of deterministic partial policies that map from private observations to environment actions; and the transition function is given by $P(s'_{\text{BAD}}|s_{\text{BAD}}, \hat{\pi})$. The next state contains the new public belief calculated using the public belief update. The reward function marginalises over the private state features:

$$r_{\text{BAD}}(s_{\text{BAD}}, \hat{\pi}) = \sum_{f^{\text{pri}}} \mathcal{B}(f^{\text{pri}})r(s, \hat{\pi}(f^{\text{pri}})), \tag{7.3.3}$$

where $s_{\text{BAD}} = \{\mathcal{B}, f^{\text{pub}}\}$. Since $s'_{\text{BAD}}$ includes the new public belief, and that belief is computed via an update which conditions on $\hat{\pi}$, the PuB-MDP transition function conditions on all of $\hat{\pi}$, not just the selected action $u_t^a$. Thus the state transition

depends not just on the executed action, but on the *counterfactual actions*, i.e., those specified by $\hat{\pi}$ for private observations other than $f_t^a$.

In the remainder of this section, we describe how factorised beliefs and policies can be used to learn a public policy $\pi_{\text{BAD}}$ for the PuB-MDP efficiently.

### 7.3.3 Sampling Deterministic Partial Policies

For each public state, $\pi_{\text{BAD}}$ must select a distribution $\pi_{\text{BAD}}(\hat{\pi}|s_{\text{BAD}})$ over deterministic partial policies. The size of this space is exponential in the number of possible private observations $|f^a|$, but we can reduce this to a linear dependence by assuming a distribution across $\hat{\pi}$ that is factorised across the different private observations, i.e., for all $\hat{\pi}$,

$$\pi_{\text{BAD}}(\hat{\pi}|\mathcal{B}_t, f^{\text{pub}}) := \prod_{f^a} \pi_{\text{BAD}}(\hat{\pi}(f^a)|\mathcal{B}_t, f^{\text{pub}}, f^a). \tag{7.3.4}$$

With this restriction, we can easily parameterise $\pi_{\text{BAD}}$ with factors of the form $\pi^\theta_{\text{BAD}}(u^a|\mathcal{B}_t, f^{\text{pub}}, f^a)$ using a function approximator such as a deep neural network.

In order for all of the agents to perform the public belief update, the sampled $\hat{\pi}$ must be public. We resolve this by having $\hat{\pi}$ sampled deterministically from a given $\mathcal{B}_t$ and $f_t^{\text{pub}}$, using a common knowledge random seed $\xi_t$. The seeds are then shared prior to the game so that all agents sample the same $\hat{\pi}$: this resembles the way humans share common ways of reasoning in card games and allows the agents to explore alternative policies jointly as a team.

### 7.3.4 Factorised Belief Updates.

In general, representing exact beliefs is intractable in all but the smallest state spaces. For example, in card games the number of possible hands is typically exponential in the number of cards held by all players. To avoid this unfavourable scaling, we can instead represent an approximate factorised belief state

$$P(f_t^{\text{pri}}|f_{\leq t}^{\text{pub}}) \approx \prod_i P(f_t^{\text{pri}}[i]|f_{\leq t}^{\text{pub}}) := \mathcal{B}_t^{\text{fact}}. \tag{7.3.5}$$

From here on we drop the superscript and use $\mathcal{B}$ exclusively to refer to the factorised belief. In a card game each factor represents per-card probability distributions,

assuming approximate independence across the different cards both within a hand and across players. This approximation makes it possible to represent and reason over the otherwise intractably large state spaces that commonly occur in many settings, including card games.

To carry out the public belief update with a factorised representation we maintain factorised likelihood terms $\mathcal{L}_t[f[i]]$ for each private feature that we update recursively:

$$\mathcal{L}_t[f[i]] := P(u^a_{\leq t}|f[i], \mathcal{B}_{\leq t}, f^{\text{pub}}_{\leq t}, \hat{\pi}_{\leq t}) \tag{7.3.6}$$

$$\approx \mathcal{L}_{t-1}[f[i]] \cdot P(u^a_t|f[i], \mathcal{B}_t, f^{\text{pub}}_t, \hat{\pi}_t) \tag{7.3.7}$$

$$= \mathcal{L}_{t-1}[f[i]] \cdot \frac{\mathbb{E}_{f_t \sim \mathcal{B}_t}\Big[\mathbb{1}(f_t[i], f[i])\mathbb{1}(\hat{\pi}(f^a_t), u^a_t)\Big]}{\mathbb{E}_{f_t \sim \mathcal{B}_t}\Big[\mathbb{1}(f_t[i], f[i])\Big]}, \tag{7.3.8}$$

where (7.3.7) assumes that actions are (approximately) conditionally independent of the future given the past. As indicated, these likelihood terms are calculated by sampling, and the larger number of samples the better. We sampled $S = 3,000$ hands during training and $S = 20,000$ hands for the final test games.

### 7.3.5 Self-Consistent Beliefs

This factorisation is only an approximation, even in very simple card games — knowledge that a player is holding a specific card clearly influences the probability that another player is holding that same card. Furthermore, using our approximation can result in beliefs that are not even self-consistent, i.e., they are not the marginalisation of any belief over joint features. While not central to the key ideas behind BAD, we introduce a general iterative procedure that can account for feature interactions in factorised models. Starting with a public belief $\mathcal{B}_t$ we can iteratively update the belief to make it more self-consistent through re-marginalisation:

$$\mathcal{B}^0 = \mathcal{B}_t, \tag{7.3.9}$$

$$\mathcal{B}^{k+1}(f[i]) = \sum_{f[\text{-}i]} \mathcal{B}^k(f[\text{-}i]) P(f[i]|f[\text{-}i], f^{\text{pub}}_{\leq t}, u^a_{\leq t}, \hat{\pi}_{\leq t}) \tag{7.3.10}$$

$$\propto \mathbb{E}_{f[\text{-}i] \sim \mathcal{B}^k}\Big[\mathcal{L}_t(f[i]) P(f[i]|f[\text{-}i], f^{\text{pub}}_t)\Big], \tag{7.3.11}$$

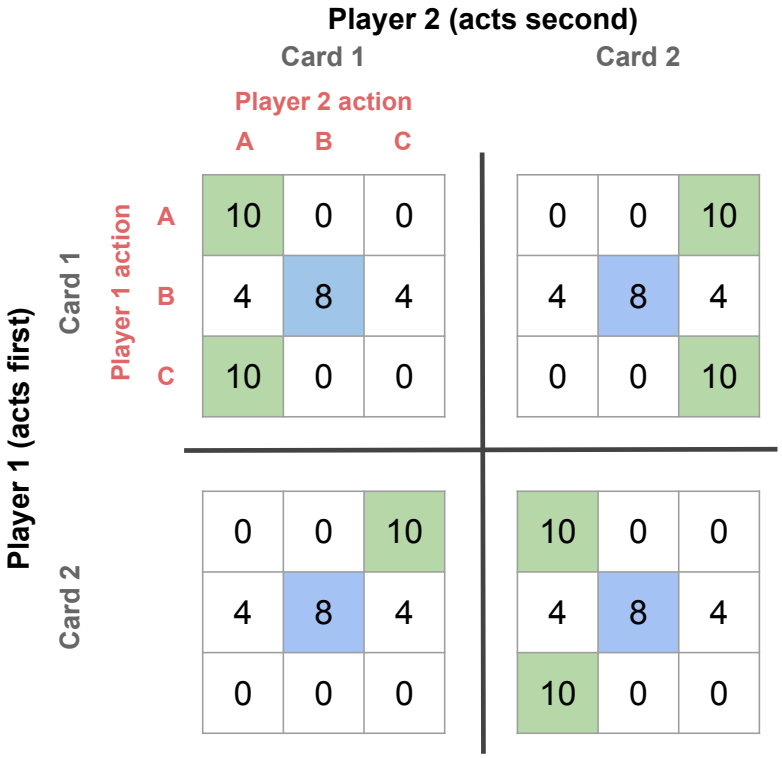



**Figure 7.2:** Payoffs for the toy matrix-like game. The two outer dimensions correspond to the card held by each player, the two inner dimensions to the action chosen by each player. Payouts are structured such that Player 1 must encode information about their card in the action they chose in order to obtain maximum payoffs. Although presented here in matrix form for compactness, this is a two-step, turn-based game, with Player 1 always taking the first action and Player 2 taking an action after observing Player 1's action.

where $f[\text{-}i]$ denotes all features excluding $f[i]$. In the last step we used the factorised likelihood terms from above and converted to an expectation, so that we can use samples to approximate the intractable sum across features. The notion of refining the probability across one feature while keeping the probability across all other features fixed is similar to the Expectation Propagation algorithm used in factor graphs [114]. However, the card counts constitute a global factor, which renders the factor graph formulation less useful. While this iterative update can in principle be carried out until convergence, in practice we terminate after a fixed number of iterations.

## 7.4 Experiments and Results

### 7.4.1 Matrix Game

We first present proof-of-principle results for a two-player, two-step partially observable matrix-like game (Figure 7.2). The state consists of 2 random bits (the cards for Player 1 and 2) and the action space consists of 3 discrete actions. Each player observes its own card, with Player 2 also observing Player 1's action before acting, which in principle allows Player 1 to encode information about its card with its action. The reward is specified by a payoff tensor, $r = \text{Payoff}[\text{card}^1][\text{card}^2][u^1][u^2]$, where $\text{card}^a$ and $u^a$ are the card and action of the two players, respectively. The payout tensor is structured such that the optimal reward can only be achieved if the two players establish a convention, in particular if Player 1 chooses informative actions that can be decoded by Player 2.

As shown in Figure 7.3, BAD clearly outperforms the baseline policy-gradient method on the toy matrix game. In this small, exact setting, it is also possible to estimate counterfactual (CF) policy gradients that reinforce not only the action taken, but also these counterfactual actions. This can be achieved by replacing $\log \pi^a(u_t^a|\tau^a)$ with $\log P(\hat{\pi}|\mathcal{B}_t, f^{\text{pub}})$ in the estimation of the policy gradient. However, the additional improvement in performance from using CF gradients is minor compared to the initial performance gain from using a counterfactual belief state.

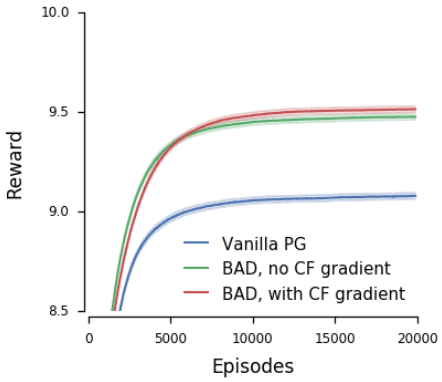



**Figure 7.3:** BAD, both with and without counterfactual gradients, outperforms vanilla policy gradient on the matrix game. Shown is mean of 1000 games.

Code for the matrix game with a proof-of-principle implementation of BAD is available at `https://bit.ly/2P3YOyd`.

### 7.4.2 Hanabi

Here we briefly describe the rules of Hanabi.

Let $N_h$ the number of cards in a hand and $n$ the number of players. In the standard game of Hanabi, $N_h = 5$ for $n = 2$ or 3 and $N_h = 4$ for $n = 4$ or 5 players. For generality, we consider that for each colour there are three cards with rank = 1, one rank = $N_{\text{rank}}$, and two each of rank = $2, \ldots, (N_{\text{rank}} - 1)$, i.e., $2N_{\text{rank}}$ cards per colour for a total of $N_{\text{deck}} = 2N_{\text{color}}N_{\text{rank}}$ cards in the deck. In the standard game of Hanabi, $N_{\text{color}} = N_{\text{rank}} = 5$ for a total of $N_{\text{deck}} = 50$. While this is a modestly large number of cards, even for 2 players it leads to $6.2 \times 10^{13}$ possible joint hands at the beginning of the game.

### 7.4.3 Observations and Actions

Each player observes the hands of all other players, but not their own. The action space consists of $N_h \times 2$ options for discarding and playing cards, and $N_{\text{color}} + N_{\text{rank}}$ options per teammate for hinting colours and ranks. Hints reveal all cards of a specific rank or colour to one of the teammates, e.g., 'Player 2's card 3 and 5 are red'. Hinting for colours and ranks not present in the hand of the teammate (so-called 'empty hints') is not allowed.

Each hint costs one hint token. The game starts with 8 hint tokens, which can be recovered by discarding cards. After a player has played or discarded a card, it draws a new card from the deck. When a player picks up the last card, everyone (including that player) gets to take one more action before the game terminates. Legal gameplay consists of building $N_{\text{color}}$ fireworks, which are piles of ascending numbers, starting at 1, for each colour. When the $N_{\text{rank}}$-th card has been added to a pile the firework is complete and the team obtains another hint token (unless they already have 8). Each time an agent plays an illegal card the players lose a life token, after three mistakes the game also terminates. Players receive 1 point after playing any playable card, with a perfect score being $N_{\text{color}}N_{\text{rank}}$.

The number of hint and life tokens at any time are observed by all players, as are the played and discarded cards, the last action of the acting player and any hints provided.

### 7.4.4 Beliefs in Hanabi

The basic belief calculation in Hanabi is straightforward: $f_t^{\text{pub}}$ consists of a vector of 'candidates' $C$ containing counts for all remaining cards, and a 'hint mask' HM, an $nN_h \times (N_{\text{color}}N_{\text{rank}} + 1)$ binary matrix that is 1 if in a given 'slot' the player could be holding a specific card according to the hints so far, and 0 otherwise; the additional 1 accounts for the possibility that the card may not exist in the final round of play. Slots correspond to the features of the private state space $f[i]$, for example the 3rd card of the second player. Hints contain both positive and negative information: for example, the statement 'the 2nd and 4th cards are red' also implies that all other cards are not red.

The basic belief $B^0$ can be calculated as

$$B^0(f[i]) = P(f[i]|f^{\text{pub}}) \propto C(f) \times \text{HM}(f[i]). \tag{7.4.1}$$

We call this the 'V0 belief', in which the belief for each card depends only on the publicly available information for that card. In our experiments, we focus on baseline agents that receive this basic belief, rather than the raw hints, as public observation inputs; while the problem of simply remembering all hints and their most immediate implication for card counts is potentially challenging for humans in recreational play, we are here more interested in the problem of forming effective conventions for high-level play.

As noted above, this basic belief misses an important interaction between the hints for different slots. We can calculate an approximate version of the self-consistent beliefs that avoids the potentially expensive and noisy sampling step in Equation 7.3.11 (note that this sampling is distinct from the sampling required to compute the marginal likelihood in Equation 7.3.8).

The basic per-card belief is simply:

$$B^0(f[i]) \propto C(f) \times \text{HM}(f[i]) \times \mathcal{L}(f[i]), \tag{7.4.2}$$

$$B^0(f[i]) = \frac{C(f) \times \text{HM}(f[i]) \times \mathcal{L}(f[i])}{\sum_g C(g) \times \text{HM}(g[i]) \times \mathcal{L}(g[i])} \tag{7.4.3}$$

$$= \beta_i \Big( C(f) \times \text{HM}(f[i]) \times \mathcal{L}(f[i]) \Big). \tag{7.4.4}$$

In the last two lines we are normalising the probability, since the probability of the $i$-th feature being one of the possible values must sum to 1. For convenience we also introduced the notation $\beta_i$ for the normalisation factor.

Next we apply the same logic to the iterative belief update. The key insight here is to note that conditioning on the features $f[\text{-}i]$, i.e., the other cards in the slots, corresponds to reducing the card counts in the candidates. Below we use $M(f[i]) = \text{HM}(f[i]) \times \mathcal{L}(f[i])$ for notational convenience:

$$\mathcal{B}^{k+1}(f[i])$$

$$= \sum_{f[-i]} \mathcal{B}^k(f[\text{-}i]) P(f[i] | f[\text{-}i], f^{\text{pub}}_{\leq t}, u^a_{\leq t}, \hat{\pi}_{\leq t}) \tag{7.4.5}$$

$$= \sum_{g[-i]} \mathcal{B}^k(g[\text{-}i]) \beta_i \left( C(f) - \sum_{j \neq i} \mathbb{1}(g[j] = f) \right) M(f[i]). \tag{7.4.6}$$

In the last line we relabelled the dummy index $f[\text{-}i]$ to $g[\text{-}i]$ for clarity and used the result from above. Next we substitute the factorised belief assumption across the features, $\mathcal{B}^k(g[\text{-}i]) = \prod_{j \neq i} \mathcal{B}^k(g[j])$ :

$$\mathcal{B}^{k+1}(f[i])$$

$$= \sum_{g[-i]} \mathcal{B}^k(g[\text{-}i]) \beta_i \left( C(f) - \sum_{j \neq i} \mathbb{1}(g[j] = f) \right) M(f[i]) \tag{7.4.7}$$

$$= \sum_{g[-i]} \prod_{j \neq i} \mathcal{B}^k(g[j]) \beta_i \left( C(f) - \sum_{j \neq i} \mathbb{1}(g[j] = f) \right) M(f[i]) \tag{7.4.8}$$

$$\simeq \beta_i \sum_{g[-i]} \prod_{j \neq i} \mathcal{B}^k(g[j]) \left( C(f) - \sum_{j \neq i} \mathbb{1}(g[j] = f) \right) M(f[i]). \tag{7.4.9}$$

In the last line we have omited the dependency of $\beta_i$ on the sampled hands $f[\text{-}i]$. It corresponds to calculating the average across sampled hands first and then normalising (which is approximate but tractable) rather than normalising

and then averaging (which is exact but intractable). We can now use product-sum rules to simplify the expression.

$$\begin{aligned}
&\mathcal{B}^{k+1}(f[i]) \\
&\simeq \beta_i\left(C(f) - \sum_{g[-i]}\prod_{j\neq i}\mathcal{B}^k(g[j])\sum_{j\neq i}\mathbb{1}(g[j]=f)\right)M(f[i]) && (7.4.10)\\
&= \beta_i\left(C(f) - \sum_{j\neq i}\sum_{g}\mathcal{B}^k(g[j])\mathbb{1}(g[j]=f)\right)M(f[i]) && (7.4.11)\\
&= \beta_i\left(C(f) - \sum_{j\neq i}\mathcal{B}^k(f[j])\right)M(f[i]) && (7.4.12)\\
&\propto \left(C(f) - \sum_{j\neq i}\mathcal{B}^k(f[j])\right)M(f[i]). && (7.4.13)
\end{aligned}$$

This concludes the derivation.

Therefore we can iteratively compute an approximate self consistent beliefs without any sampling:

$$B^{k+1}(f[i]) \propto \left(C(f) - \sum_{j\neq i}B^k(f[j])\right) \times \mathrm{HM}(f[i]). \tag{7.4.14}$$

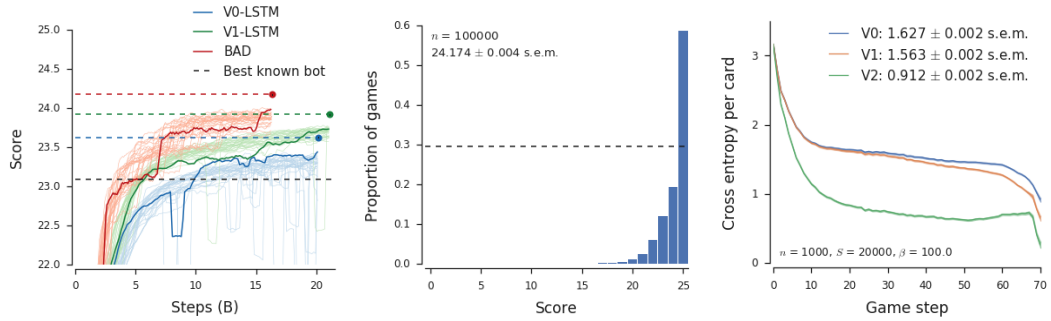
We call the resulting belief at convergence (or after a maximum number of iterations) the 'V1 belief'. It does not condition on the Bayesian probabilities but considers interactions between hints for different cards. In essence, at each iteration the belief for a given slot is updated by reducing the candidate count by the number of cards believed to be held across all other slots.

By running the same algorithm but including $\mathcal{L}$, we obtain the Bayesian beliefs, BB, that lie at the core of BAD:

$$\mathrm{BB}^0(f[i]) \propto C(f) \times \mathrm{HM}(f[i]) \times \mathcal{L}(f[i]), \tag{7.4.15}$$

$$\begin{aligned}
\mathrm{BB}^{k+1}(f[i]) \propto &\left(C(f) - \sum_{j\neq i}B^k(f[j])\right) \\
&\times \mathrm{HM}(f[i]) \times \mathcal{L}(f[i]).
\end{aligned} \tag{7.4.16}$$

In practice, to ensure stability, the final 'V2 belief' that we use is an interpolation between the Bayesian belief and the V1 belief: $\mathrm{V2} = (1-\alpha)\mathrm{BB} + \alpha\mathrm{V1}$ with $\alpha = 0.01$ (we found $\alpha = 0.1$ to also work).




(a) (b) (c)

**Figure 7.4:** a) Training curves for BAD on Hanabi and the V0 and V1 baseline methods using LSTMs rather than the Bayesian belief. The thick line for each agent type shows the final evaluated agent for each type; upward kinks are generally due to agents 'evolving' in PBT by copying its weights and hyperparameters (plus perturbations) from a superior agent. b) Distribution of game scores for BAD on Hanabi under testing conditions. BAD achieves a perfect score in nearly 60% of the games. The dashed line shows the proportion of perfect games reported for SmartBot, the best known heuristic for two-player Hanabi. c) Per-card cross entropy with the true hand for different belief mechanisms in Hanabi during BAD play. V0 is the basic belief based on hints and card counts, V1 is the self-consistent belief, and V2 is the BAD belief which also includes the Bayesian update. The BAD agent conveys around 40% of the information via conventions, rather than grounded information.

### 7.4.5 Architecture Details for Baselines and Method

Advantage actor-critic agents were trained using the Importance-Weighted Actor-Learner Architecture [115], in particular the multi-agent implementation described in [116]. In this framework, 'actors' continually generate trajectories of experience (sequences of states, actions, and rewards) by having agents (self-)playing the game, which are then used by 'learners' to perform batched gradient updates (batch size was 32 for all agents). Because the policy used to generate the trajectory can be several gradient updates behind the policy at the time of the gradient update, V-trace was applied to correct for the off-policy trajectories. The length of the trajectories, or rollouts, was 65, the maximum length of a winning game.

In the V0-LSTM and V1-LSTM agents, all observations were first processed by an MLP with a single 256-unit hidden layer and ReLU activations, then fed into a 2-layer LSTM with 256 units in each layer. The policy $\pi$ was a linear softmax readout of the LSTM output. The baseline network was an MLP with a single

256-unit hidden layer and ReLU activations, which then projected linearly to a single value. Since the baseline network is only used to compute gradient updates, we followed the centralised critic from Chapter 3 in feeding each agent's own hand (i.e., the other agent's private observation) into the baseline by concatenating it with the LSTM output; thus we make the common assumption of centralised training and decentralised execution. We note that the V0 and V1-LSTM agents differed *only* in their public belief inputs.

The BAD agent consisted of an MLP with two 384-unit hidden layers and ReLU activations that processed all observations, followed by a linear softmax policy readout. To compute the baseline, we used the same MLP as the policy but included the agent's own hand in the input (this input was present but zeroed out for the computation of the policy).

For all agents, illegal actions (such as hint for a red card when there are no red cards) were masked out by setting the corresponding policy logits to a large negative value before sampling an action. In particular, for the non-acting agent at each turn the only allowed action was the 'no-action'.

### 7.4.6 Hyperparamters

For the toy matrix game, we used a batch size of 32 and the Adam optimiser with all default TensorFlow settings; we did not tune hyperparameters for any runs.

For Hanabi, we used the RMSProp optimiser with $\epsilon = 10^{-10}$, momentum 0, and decay 0.99. The RL discounting factor $\gamma$ was set to 0.999. The baseline loss was multiplied by 0.25 and added to the policy-gradient loss. We used population-based training (PBT) [116, 117] to 'evolve' the learning rate and entropy regularisation parameter during the course of training, with each training run consisting of a population of 30 agents. For the LSTM agents, learning rates were sampled log-uniformly from the interval $[1, 4) \times 10^{-4}$ while the entropy regularisation parameter was sampled log-uniformly from the interval $[1, 5) \times 10^{-2}$. For the BAD agents, learning rates were sampled log-uniformly from the interval $[9 \times 10^{-5},\ 3 \times 10^{-4})$ while the entropy regularisation parameter was sampled log-uniformly from the interval

$[3, 7) \times 10^{-2}$. Agents evolved within the PBT framework by copying weights and hyperparameters (plus perturbations) according to each agent's rating, which was an exponentially moving average of the episode rewards with factor 0.01. An agent was considered for copying roughly every 200M steps if a randomly chosen copy-to agent had a rating at least 0.5 points higher. To allow the best hyperparameters to manifest sufficiently, PBT was turned off for the first 1B steps of training.

The BAD agent was trained with 100 self-consistent iterations, a V1 mix-in of $\alpha = 0.01$, inverse temperature 1.0, and 3000 sampled hands. Since sampling from card-factorised beliefs can result in hands that are not compatible with the deck, we sampled 5 times the number of hands and accepted the first 3000 legal hands, zeroing out any hands that were illegal.

### 7.4.7 Results on Hanabi

The BAD agent achieves a new state-of-the-art mean performance of 24.174 points on two-player Hanabi. In Figure 7.4a we show training curves and test performance for BAD and two LSTM-based baseline methods, as well as the performance of the the best known hand-coded bot for two-player Hanabi. For the LSTM agents, test performance was obtained by using the greedy version of the trained policy, resulting in slightly higher scores than during training. To select the agent, we first performed a sweep over all agents for 10,000 games, then carried out a final test run of 100,000 games on the best agent from the sweep, since taking the maximum of a sweep introduces bias in the score. We carried out a similar procedure for the BAD agent but with additional hyperparameters, also varying the V1 mix-in factor, and number of sampled hands.

The results for other learning methods from the literature perform well below the range of the $y$-axis (far below 20 points) and are omitted for readability. We note that, under a strict interpretation of the rules of Hanabi, games in which all three error tokens are exhausted should be awarded a score of zero. Under these rules the same BAD agent achieves $23.917 \pm 0.009$ s.e.m., still better than the hand-coded

| Agent | Learning steps | Mean ± s.e.m. | Prop. perfect |
|---|---|---|---|
| SmartBot | - | 23.09 | 29.52% |
| V0-LSTM | 20.2B | 23.622 ± 0.005 | 36.5% |
| V1-LSTM | 21.1B | 23.919 ± 0.004 | 47.5% |
| BAD | 16.3B | **24.174± 0.004** | **58.6%** |

**Table 7.1:** Test scores on 100K games. The LSTM agents were tested with a greedy version of the trained policy, while the final BAD agent was evaluated with V1 mix-in $\alpha = 0.01$, 20K sampled hands, and inverse softmax temperature 100.0.

bot (for whom only results in which all points are counted have been reported). This is true even though our agents were not trained under these conditions.

While not all of the game play BAD learns is easy to follow, some conventions can be understood simply from inspecting the game. Printouts of 100 random games can be found at `https://bit.ly/2zeEShh`. One convention stands out: Hinting for 'red' or 'yellow' indicates that the newest card of the other player is playable. We found that in over 80% of cases when an agent hints 'red' or 'yellow', the next action of the other agent is to play the newest card. This convention is very powerful: Typically agents know the least about the newest card, so by hinting 'red' or 'yellow', agents can use a single hint to tell the other agent that the card is playable. Indeed, the use of two colours to indicate 'play newest card' was present all of the highest-performing agents we studied. Hinting 'white' and 'blue' are followed by a discard of the newest card in over 25% of cases. We also found that the agent sometimes attempts to play cards which are not playable in order to convey information to their teammate. In general, unlike human players, agents play and discard predominantly from the last card.

Figure 7.4c) shows the quality of the different beliefs. While the iterated belief update leads to a reduction in cross entropy compared to the basic belief, a much greater reduction in cross entropy is obtained using counterfactual beliefs. This clearly demonstrates the importance of learning conventions for successful gameplay in Hanabi: Roughly 40% of the information is obtained through conventions rather than through the grounded information and card counting.

## 7.5 Related Work

### 7.5.1 Learning to Communicate

Many works have addressed problem settings where agents must learn to communicate in order to cooperatively solve a toy problem. These tasks typically involve a cheap-talk communication channel that can be modeled as a continuous variable during training, which allows differentiation through the channel. This was first proposed by us, in Chapter 6, and Sukhbaatar, Fergus, et al. [48], and has since been applied to a number of different settings. In this work we focused on the case where, rather than relying on a cheap-talk channel, agents must learn to communicate via grounded hinting actions and observable environment actions. This setting is closest to the 'hat game' in Foerster et al. [118]. In this work we proposed a simple extension to recurrent deep Q-networks rather than explicitly modeling action-conditioned Bayesian beliefs. An idea very similar to the Pub-MDP was introduced in the context of decentralised stochastic control by Nayyar, Mahajan, and Teneketzis [110], who also formulated a coordinator that uses 'common information' to map local controller information to actions. However, they did not provide a concrete solution method that can scale to a high-dimensional problem like Hanabi.

### 7.5.2 Research on Hanabi

A number of papers have been published on Hanabi. Baffier et al. [119] showed that optimal gameplay in Hanabi is NP-hard even when players can observe their own cards. Encoding schemes similar to the hat game essentially solves the 5-player case [120], but only achieve 17.8 points in the two-player setting [121]. Walton-Rivers et al. [122] developed a variety of Monte Carlo tree search and rule-based methods for learning to play the game, but the reported scores were roughly 50% lower than those achieved by BAD. Osawa [123] defined a number of heuristics for the two-player case that reason over possible hands given the other player's action. While this is similar in spirit to our approach, the work was limited to hand-coded heuristics, and the reported scores were around 8 points lower than our results.

Eger, Martens, and Cordoba [124] investigated humans playing with hand-coded agents, but no pairing resulted in scores higher than 15 points on average.

The best result for two-player Hanabi we could find was for the 'SmartBot' described at `github.com/Quuxplusone/Hanabi`, which has been reported to achieve an average of 23.09 points (29.52% perfect games). While SmartBot uses the same game rules as those used in our work, it is entirely hand-coded and involves no learning.

### 7.5.3 Belief State Methods

The continual re-solving (nested solving) algorithm used by DeepStack [125] and Libratus [126] for poker also used a belief state space. Like BAD, when making a decision in a player state, continual re-solving considers the belief state associated with the current player and generates a joint policy across all player states consistent with this belief. The policy for the actual player is then selected from this joint policy. Continual re-solving also does a Bayesian update of the beliefs after an action. There are key differences, however. Continual re-solving performed exact belief updates, which requires that the joint policy space be small enough to enumerate; belief states were also augmented with opponent values. Continual re-solving is a value-based method, where the training process consists of learning the values of belief states under optimal play. Finally, the algorithm was designed for two-player, zero-sum games, where it can independently consider player state values while guaranteeing that an optimal choice for the joint action policy can be found.

## 7.6 Conclusion & Future Work

We presented the *Bayesian action decoder* (BAD), a novel algorithm for multi-agent reinforcement learning in cooperative partially observable settings. BAD uses a factorised, approximate belief state that allows agents to efficiently learn informative actions, leading to the discovery of conventions. We showed that BAD outperforms policy gradients in a proof-of-principle matrix game, and achieves a state-of-the-art performance of 24.174 points on average in the card game Hanabi. We also showed

that using the Bayesian update leads to a reduction in uncertainty across the private hands in Hanabi by around 40%. To the best of our knowledge, this is the first instance in which DMARL has been successfully applied to a problem setting that both requires the discovery of communication protocols and was originally designed to be challenging for humans. BAD also illustrates clearly that using an explicit belief computation achieves better performance in such settings than current state-of-the-art RL methods using implicit beliefs, such as recurrent neural networks.

In the future we would like to apply BAD to more than 2 players and further generalise BAD by learning more of the components. While the belief update necessarily involves a sampling step, most of the other components can likely be learned end-to-end. We also plan to extend the BAD mechanism to value-based methods and further investigate the relevance of counterfactual gradients.

# Part III

# Learning to Reciprocate

# Abstract

So far we have assumed fully cooperative settings, in which all agents work together to maximise a common return. However, in the real world, commonly a number of different agents in the same environment pursue their own goals, leading to potential conflicts of interest. In Chapter 8 we propose a novel framework for allowing agents to consider the learning behaviour of other agents in the environment as part of their policy-optimisation. We show that when each agent anticipates one step of opponent learning, we can obtain drastically different learning outcomes for self-interested agents. When accounting for the learning behaviour of others in RL settings, higher order gradients need to be estimated using samples. In Chapter 9 we propose a new objective which generates the correct gradient estimators under automatic differentiation.

# 8

# Learning with Opponent-Learning Awareness

## Contents



## 8.1 Introduction

In the previous two parts of the thesis we have been focused on methods that address the learning challenges associated with fully cooperative multi-agent RL.

However, hierarchical reinforcement learning, generative adversarial networks and decentralised optimisation can also be regarded as multi-agent problems. In all these settings the presence of multiple learning agents renders the training problem nonstationary and often leads to unstable training or undesired final results, especially when we allow for different agents to pursue different objectives. Furthermore, considering future applications of multi-agent RL, such as self-driving cars, it is obvious that many of these will be only partially cooperative and contain elements of competition and conflict.

The human ability to maintain cooperation in a variety of complex social settings has been vital for the success of human societies. Emergent reciprocity has been observed even in strongly adversarial settings such as wars [16], making it a quintessential and robust feature of human life.

In the future, artificial learning agents are likely to take an active part in human society, interacting both with other learning agents and humans in complex partially competitive settings. Failing to develop learning algorithms that lead to emergent reciprocity in these artificial agents would lead to disastrous outcomes.

How reciprocity can emerge among a group of learning, self-interested, reward maximising RL agents is thus a question both of theoretical interest and of practical importance. Game theory has a long history of studying the learning outcomes in games that contain cooperative and competitive elements. In particular, the tension between cooperation and defection is commonly studied in the iterated prisoner's dilemma. In this game, selfish interests can lead to an outcome that is overall worse for all participants, while cooperation maximises social welfare, one measure of which is the sum of rewards for all agents.

Interestingly, in the simple setting of an infinitely repeated prisoner's dilemma with discounting, randomly initialised RL agents pursuing independent gradient descent on the exact value function learn to defect with high probability. This shows that current state-of-the-art learning methods in DMARL can lead to agents that fail to cooperate reliably even in simple social settings with explicit actions to cooperate and defect. One well-known shortcoming is that they fail to consider

the learning process of the other agents and simply treat the other agent as a *static* part of the environment [127].

As a step towards reasoning over the learning behaviour of other agents in social settings, we propose *Learning with Opponent-Learning Awareness* (LOLA). The LOLA learning rule includes an additional term that accounts for the impact of one agent's parameter update on the learning step of the other agents. For convenience we use the word 'opponents' to describe the other agents, even though the method is not limited to zero-sum games and can be applied in the general-sum setting. We show that this additional term, when applied by both agents, leads to emergent reciprocity and cooperation in the iterated prisoner's dilemma (IPD). Experimentally we also show that in the IPD, each agent is incentivised to switch from naive learning to LOLA, while there are no additional gains in attempting to exploit LOLA with higher order gradient terms. This suggests that within the space of local, gradient-based learning rules both agents using LOLA is a stable equilibrium. This is further supported by the good performance of the LOLA agent in a round-robin tournament, where it successfully manages to shape the learning of a number of multi-agent learning algorithms from literature. This leads to the overall highest average return on the IPD and good performance on Iterated Matching Pennies (IMP).

We also present a version of LOLA adopted to the DMARL setting using likelihood ratio policy gradients, making LOLA scalable to settings with high dimensional input and parameter spaces.

We evaluate the policy gradient version of LOLA on the IPD and iterated matching pennies (IMP), a simplified version of rock-paper-scissors. We show that LOLA leads to cooperation with high social welfare, while independent policy gradients, a standard multi-agent RL approach, does not. The policy gradient finding is consistent with prior work, e.g., Sandholm and Crites [128]. We also extend LOLA to settings where the opponent policy is unknown and needs to be inferred from state-action trajectories of the opponent's behaviour.

Finally, we apply LOLA with and without opponent modelling to a grid-world task with an embedded underlying social dilemma. This task has temporally extended actions and therefore requires high dimensional recurrent policies for agents to learn to reciprocate. Again, cooperation emerges in this task when using LOLA, even when the opponent's policy is unknown and needs to be estimated.

## 8.2 Related Work

The study of general-sum games has a long history in game theory and evolution. Many papers address the iterated prisoner's dilemma (IPD) in particular, including the seminal work on the topic by Axelrod [16]. This work popularised tit-for-tat (TFT), a strategy in which an agent cooperates on the first move and then copies the opponent's most recent move, as an effective and simple strategy in the IPD.

A number of methods in multi-agent RL aim to achieve convergence in self-play and rationality in sequential, general sum games. Seminal work includes the family of WoLF algorithms [129], which uses different learning rates depending on whether an agent is winning or losing, joint-action-learners (JAL), and AWESOME [87]. Unlike LOLA, these algorithms typically have well understood convergence behaviour given an appropriate set of constraints. However, none of these algorithm have the ability to shape the learning behaviour of the opponents in order to obtain higher payouts at convergence. AWESOME aims to learn the equilibria of the one-shot game, a subset of the equilibria of the iterated game.

Detailed studies have analysed the dynamics of JALs in general sum settings: This includes work by Uther and Veloso [130] in zero-sum settings and by Claus and Boutilier [131] in cooperative settings. Sandholm and Crites [128] study the dynamics of independent Q-learning in the IPD under a range of different exploration schedules and function approximators. Wunder, Littman, and Babes [132] and Zinkevich, Greenwald, and Littman [133] explicitly study the convergence dynamics and equilibria of learning in iterated games. Unlike LOLA, these papers do not propose novel learning rules.

Littman [134] propose a method that assumes each opponent either to be a friend, i.e., fully cooperative, or foe, i.e., fully adversarial. Instead, LOLA considers general sum games.

By comparing a set of models with different history lengths, Chakraborty and Stone [135] propose a method to learn a best response to memory bounded agents with fixed policies. In contrast, LOLA assumes learning agents, which effectively correspond to unbounded memory policies.

Brafman and Tennenholtz [136] introduce the solution concept of an efficient learning equilibrium (ELE), in which neither side is encouraged to deviate from the learning rule. The algorithm they propose applies to settings where all Nash equilibria can be computed and enumerated; LOLA does not require either of these assumptions.

By contrast, most work in DMARL focuses on fully cooperative or zero-sum settings, in which learning progress is easier to evaluate, [50] and emergent communication in particular [45–48], see also Part I and Part II of this thesis. As an exception, Leibo et al. [44] analyse the outcomes of naive learning in partially observable, general sum settings using feedforward neural networks as policies. Lowe et al. [53] propose a centralised actor-critic architecture for efficient training in these general sum environments. However, none of these methods explicitly reasons about the learning behaviour of other agents. Lanctot et al. [137] generalise the ideas of game-theoretic best-response-style algorithms, such as NFSP [138]. In contrast to LOLA, these best-response algorithms assume a given set of opponent policies, rather than attempting to shape the learning of the other agents.

The problem setting and approach of Lerer and Peysakhovich [139] is closest to ours. They directly generalise tit-for-tat to complex environments using DMARL. The authors explicitly train a fully cooperative and a defecting policy for both agents and then construct a tit-for-tat policy that switches between these two in order to encourage the opponent to cooperate. Similar in spirit to this work, Munoz de Cote and Littman [140] propose a Nash equilibrium algorithm for repeated stochastic games that explicitly attempts to find the egalitarian equilibrium by switching

between competitive and cooperative strategies. A similar idea underlies M-Qubed, [141], which balances best-response, cautious, and optimistic learning biases.

Reciprocity and cooperation are not emergent properties of the learning rules in these algorithms but rather directly coded into the algorithm via heuristics, limiting their generality.

Our work also relates to opponent modelling, such as fictitious play [142] and action-sequence prediction [143, 144]. Mealing and Shapiro [145] also propose a method that finds a policy based on predicting the future action of a memory bounded opponent. Furthermore, Hernandez-Leal and Kaisers [146] directly model the distribution over opponents. While these methods model the opponent strategy, or distribution thereof, and use look-ahead to find optimal response policies, they do not address the learning dynamics of opponents. For further details we refer the reader to excellent reviews on the subject [42, 127].

By contrast, Zhang and Lesser [147] carry out policy prediction under one-step learning dynamics. However, the opponents' policy updates are assumed to be given and only used to learn a best response to the anticipated updated parameters. By contrast, a LOLA agent directly shapes the policy updates of all opponents in order to maximise its own reward. Differentiating through the opponent's learning step, which is unique to LOLA, is crucial for the emergence of tit-for-tat and reciprocity. To the best of our knowledge, LOLA is the first method that aims to shape the learning of other agents in a multi-agent RL setting.

With LOLA, each agent differentiates through the opponents' policy update. Similar ideas were proposed by Metz et al. [148], whose training method for generative adversarial networks differentiates through multiple update steps of the opponent. Their method relies on an end-to-end differentiable loss function, and thus does not work in the general RL setting. However, the overall results are similar: differentiating through the opponent's learning process stabilises the training outcome in a zero sum setting.

Outside of purely computational studies the emergence of cooperation and defection in RL settings has also been studied and compared to human data [149].

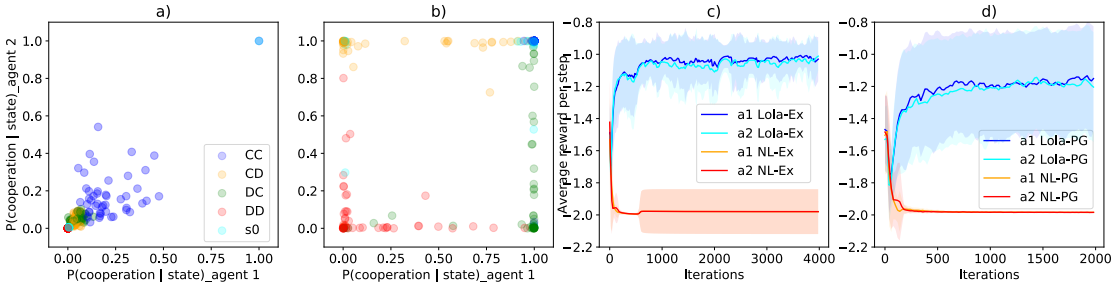



**Figure 8.1:** a) shows the probability of cooperation in the iterated prisoners dilemma (IPD) at the end of 50 training runs for both agents as a function of state under naive learning (NL-Ex) and b) displays the results for LOLA-Ex when using the exact gradients of the value function. c) shows the normalised discounted return for both agents in NL-Ex vs. NL-Ex and LOLA-Ex vs. LOLA-Ex, with the exact gradient. d) plots the normalised discounted return for both agents in NL-PG vs. NL-PG and LOLA-PG vs. LOLA-PG, with policy gradient approximation. We see that NL-Ex leads to DD, resulting in an average reward of ca. $-2$. In contrast, the LOLA-Ex agents play tit-for-tat in b): When in the last move agent 1 defected and agent 2 cooperated (DC, green points), most likely in the next move agent 1 will cooperate and agent 2 will defect, indicated by a concentration of the green points in the bottom right corner. Similarly, the yellow points (CD), are concentrated in the top left corner. While the results for the NL-PG and LOLA-PG with policy gradient approximation are more noisy, they are qualitatively similar. Best viewed in colour.

## 8.3 Methods

In this section, we review the naive learner's strategy and introduce the LOLA learning rule. We first derive the update rules when agents have access to exact gradients and Hessians of their expected discounted future return in Sections 8.3.1 and 8.3.2. In Section 8.3.3, we derive the learning rules purely based on policy gradients, thus removing access to exact gradients and Hessians. This renders LOLA suitable for DMARL. However, we still assume agents have access to opponents' policy parameters in policy gradient-based LOLA. Next, in Section 8.3.4, we incorporate opponent modelling into the LOLA learning rule, such that each LOLA agent only infers the opponent's policy parameter from experience. Finally, we discuss higher order LOLA in Section 8.3.5.

For simplicity, here we assume the number of agents is $n = 2$ and display the update rules for agent 1 only. The same derivation holds for arbitrary numbers of agents.

### 8.3.1 Naive Learner

Suppose each agent's policy $\pi^a$ is parameterised by $\theta^a$ and $J^a(\theta^1, \theta^2)$ is the expected total discounted return for agent $a$ as a function of both agents' policy parameters $(\theta^1, \theta^2)$. A Naive Learner (NL) iteratively optimises for its own expected total discounted return, such that at the $i$th iteration, $\theta_i^a$ is updated to $\theta_{i+1}^a$ according to

$$\theta_{i+1}^1 = \text{argmax}_{\theta^1} J^1(\theta^1, \theta_i^2)$$
$$\theta_{i+1}^2 = \text{argmax}_{\theta^2} J^2(\theta_i^1, \theta^2).$$

In the reinforcement learning setting, agents do not have access to $\{J^1, J^2\}$ over all parameter values. Instead, we assume that agents only have access to the function values and gradients at $(\theta_i^1, \theta_i^2)$. Using this information the naive learners apply the gradient ascent update rule $\boldsymbol{f}_{\text{nl}}^1$:

$$\theta_{i+1}^1 = \theta_i^1 + \boldsymbol{f}_{\text{nl}}^1(\theta_i^1, \theta_i^2),$$
$$\boldsymbol{f}_{\text{nl}}^1 = \nabla_{\theta_i^1} J^1(\theta_i^1, \theta_i^2) \cdot \delta, \tag{8.3.1}$$

where $\delta$ is the step size.

### 8.3.2 Learning with Opponent Learning Awareness

A LOLA learner optimises its return under one step look-ahead of opponent learning: Instead of optimizing the expected return under the current parameters, $J^1(\theta_i^1, \theta_i^2)$, a LOLA agent optimises $J^1(\theta_i^1, \theta_i^2 + \Delta\theta_i^2)$, which is the expected return after the opponent updates its policy with one naive learning step, $\Delta\theta_i^2$. Going forward we drop the subscript $i$ for clarity. Assuming small $\Delta\theta^2$, a first-order Taylor expansion results in:

$$J^1(\theta^1, \theta^2 + \Delta\theta^2) \approx J^1(\theta^1, \theta^2) + (\Delta\theta^2)^T \nabla_{\theta^2} J^1(\theta^1, \theta^2). \tag{8.3.2}$$

The LOLA objective (8.3.2) differs from prior work, e.g., Zhang and Lesser [147], that predicts the opponent's policy parameter update and learns a best response. LOLA learners attempt to actively influence the opponent's future policy update, and explicitly differentiate through the $\Delta\theta^2$ with respect to $\theta^1$. Since LOLA

focuses on this shaping of the learning direction of the opponent, the dependency of $\nabla_{\theta^2} J^1(\theta^1, \theta^2)$ on $\theta^1$ is dropped during the backward pass. Investigation of how differentiating through this term would affect the learning outcomes is left for future work.

By substituting the opponent's naive learning step:

$$\Delta\theta^2 = \nabla_{\theta^2} J^2(\theta^1, \theta^2) \cdot \eta \tag{8.3.3}$$

into (8.3.2) and taking the derivative of (8.3.2) with respect to $\theta^1$, we obtain our LOLA learning rule:

$$\theta^1_{i+1} = \theta^1_i + \boldsymbol{f}^1_{\text{lola}}(\theta^1_i, \theta^2_i),$$

which includes a second order correction term

$$\begin{aligned} \boldsymbol{f}^1_{\text{lola}}(\theta^1, \theta^2) &= \nabla_{\theta^1} J^1(\theta^1, \theta^2) \cdot \delta \\ &+ \Big(\nabla_{\theta^2} J^1(\theta^1, \theta^2)\Big)^T \nabla_{\theta^1} \nabla_{\theta^2} J^2(\theta^1, \theta^2) \cdot \delta\eta, \end{aligned} \tag{8.3.4}$$

where the step sizes $\delta, \eta$ are for the first and second order updates. Exact LOLA and NL agents (LOLA-Ex and NL-Ex) have access to the gradients and Hessians of $\{J^1, J^2\}$ at the current policy parameters $(\theta^1_i, \theta^2_i)$ and can evaluate (8.3.1) and (8.3.4) exactly.

### 8.3.3 Learning via Policy Gradient

When agents do not have access to exact gradients or Hessians, we derive the update rules $f_{\text{nl, pg}}$ and $f_{\text{lola, pg}}$ based on approximations of the derivatives in (8.3.1) and (8.3.4).

We denote an episode of horizon $T$ as $\tau = (s_0, u^1_0, u^2_0, .., s_{T+1}, u^1_T, u^2_T, r^1_T, r^2_T)$ and its corresponding discounted return for agent $a$ at timestep $t$ as $R^a_t(\tau) = \sum_{t'=t}^{T} \gamma^{t'-t} r^a_{t'}$. Given this definition, the expected episodic return conditioned on the agents' policies $(\pi^1, \pi^2)$, $\mathbb{E}\, R^1_0(\tau)$ and $\mathbb{E}\, R^2_0(\tau)$, approximate $J^1$ and $J^2$ respectively, as do the gradients and Hessians.

The gradient of $\mathbb{E}\, R_0^1(\tau)$ follows from the policy gradient derivation:

$$\begin{aligned}
&\nabla_{\theta^1} \mathbb{E}\, R_0^1(\tau) = \mathbb{E}\left[R_0^1(\tau)\nabla_{\theta^1} \log \pi^1(\tau)\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T} \nabla_{\theta^1} \log \pi^1(u_t^1|s_t) \cdot \sum_{l=t}^{T} \gamma^l r_l^1\right] \\
&= \mathbb{E}\left[\sum_{t=0}^{T} \nabla_{\theta^1} \log \pi^1(u_t^1|s_t)\gamma^t \Big(R_t^1(\tau) - b(s_t)\Big)\right],
\end{aligned}$$

where $b(s_t)$ is a baseline for variance reduction. Then the update rule $\boldsymbol{f}_{\text{nl, pg}}$ for the policy gradient-based naive learner (NL-PG) is

$$\boldsymbol{f}_{\text{nl, pg}}^1 = \nabla_{\theta^1} \mathbb{E}\, R_0^1(\tau) \cdot \delta. \tag{8.3.5}$$

For the LOLA update, we derive the following estimator of the second-order term in (8.3.4) based on policy gradients. The derivation (omitted) closely resembles the standard proof of the policy gradient theorem, exploiting the fact that agents sample actions independently. We further note that this second order term is exact in expectation. In Chapter 8 we provide novel tools for algorithmically generating these gradient estimators:

$$\begin{aligned}
&\nabla_{\theta^1}\nabla_{\theta^2} \mathbb{E}\, R_0^2(\tau) \\
&\quad= \mathbb{E}\left[R_0^2(\tau)\nabla_{\theta^1} \log \pi^1(\tau)\Big(\nabla_{\theta^2} \log \pi^2(\tau)\Big)^T\right] \\
&\quad= \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t^2 \cdot \Big(\sum_{l=0}^{t} \nabla_{\theta^1} \log \pi^1(u_l^1|s_l)\Big)\right. \\
&\qquad\left.\Big(\sum_{l=0}^{t} \nabla_{\theta^2} \log \pi^2(u_l^2|s_l)\Big)^T\right].
\end{aligned} \tag{8.3.6}$$

The complete LOLA update using policy gradients (LOLA-PG) is

$$\begin{aligned}
&\boldsymbol{f}_{\text{lola, pg}}^1 = \nabla_{\theta^1} \mathbb{E}\, R_0^1(\tau) \cdot \delta + \\
&\Big(\nabla_{\theta^2} \mathbb{E}\, R_0^1(\tau)\Big)^T \nabla_{\theta^1}\nabla_{\theta^2} \mathbb{E}\, R_0^2(\tau) \cdot \delta\eta.
\end{aligned} \tag{8.3.7}$$

### 8.3.4 LOLA with Opponent modelling

Both versions (8.3.4) and (8.3.7) of LOLA learning assume that each agent has access to the exact parameters of the opponent. However, in adversarial settings the opponent's parameters are typically obscured and have to be inferred from the opponent's state-action trajectories. Our proposed opponent modelling is similar to

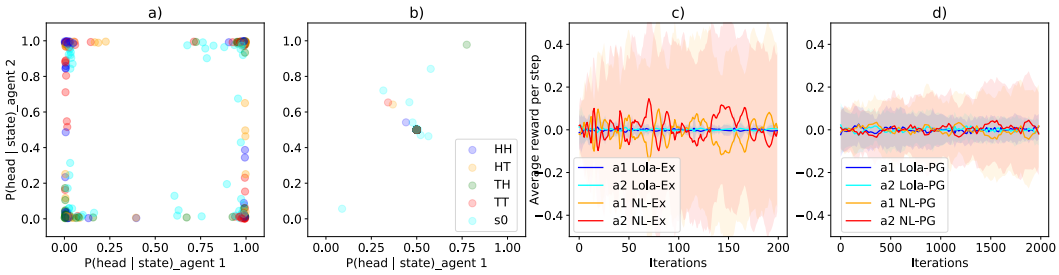



**Figure 8.2:** a) the probability of playing heads in the iterated matching pennies (IMP) at the end of 50 independent training runs for both agents as a function of state under naive learning NL-Ex. b) the results of LOLA-Ex when using the exact gradients of the value function. c) the normalised discounted return for both agents in NL-Ex vs. NL-Ex and LOLA-Ex vs. LOLA-Ex with exact gradient. d) the normalised discounted return for both agents in NL-PG vs. NL-PG and LOLA-PG vs. LOLA-PG with policy gradient approximation. We can see in a) that NL-Ex results in near deterministic strategies, indicated by the accumulation of points in the corners. These strategies are easily exploitable by other deterministic strategies leading to unstable training and high variance in the reward per step in c). In contrast, LOLA agents learn to play the only Nash strategy, 50%/50%, leading to low variance in the reward per step. One interpretation is that LOLA agents anticipate that exploiting a deviation from Nash increases their immediate return, but also renders them more exploitable by the opponent's next learning step. Best viewed in colour.

behavioural cloning [150, 151]. Instead of accessing agent 2's true policy parameters $\theta^2$, agent 1 models the opponent's behaviour with $\hat{\theta}^2$, where $\hat{\theta}^2$ is estimated from agent 2's trajectories using maximum likelihood:

$$\hat{\theta}^2 = \underset{\theta^2}{\operatorname{argmax}} \sum_t \log \pi_{\theta^2}(u_t^2|s_t). \tag{8.3.8}$$

Then, $\hat{\theta}^2$ replaces $\theta^2$ in the LOLA update rule, both for the exact version (8.3.4) using the value function and the gradient based approximation (8.3.7). We compare the performance of policy-gradient based LOLA agents (8.3.7) with and without opponent modelling in our experiments. In particular we can obtain $\hat{\theta}^2$ using the past action-observation history. In our experiments we incrementally fit to the most recent data in order to address the nonstationarity of the opponent.

### 8.3.5 Higher-Order LOLA

By substituting the naive learning rule (8.3.3) into the LOLA objective (8.3.2), the LOLA learning rule so far assumes that the opponent is a naive learner. We

call this setting *first-order LOLA*, which corresponds to the first-order learning rule of the opponent agent. However, we can also consider a higher-order LOLA agent that assumes the opponent applies a first-order LOLA learning rule, thus replacing (8.3.3). This leads to third-order derivatives in the learning rule. While the third-order terms are typically difficult to compute using policy gradient method, due to high variance, when the exact value function is available it is tractable. We examine the benefits of higher-order LOLA in our experiments.

## 8.4 Experimental Setup

In this section, we summarise the settings where we compare the learning behaviour of NL and LOLA agents. The first setting (Sec. 8.4.1) consists of two classic infinitely iterated games, the iterated prisoners dilemma (IPD), [152] and iterated matching pennies (IMP) [153]. Each round in these two environments requires a single action from each agent. We can obtain the discounted future return of each player given both players' policies, which leads to exact policy updates for NL and LOLA agents. The second setting (Sec. 8.4.2), Coin Game, a more difficult two-player environment, where each round requires the agents to take a sequence of actions and exact discounted future reward can not be calculated. The policy of each player is parameterised with a deep recurrent neural network.

In the policy gradient experiments with LOLA, we assume off-line learning, i.e., agents play many (batch-size) parallel episodes using their latest policies. Policies remain unchanged within each episode, with learning happening between episodes. One setting in which this kind of offline learning naturally arises is when policies are trained on real-world data. For example, in the case of autonomous cars, the data from a fleet of cars is used to periodically train and dispatch new policies.

### 8.4.1 Iterated Games

We first review the two iterated games, the IPD and IMP, and explain how we can model iterated games as a memory-1 two-agent MDP.

| | C | D |
|---|---|---|
| C | (-1, -1) | (-3, 0) |
| D | (0, -3) | (-2, -2) |

**Table 8.1:** Payoff matrix of prisoner's dilemma.

Table 8.1 shows the per-step payoff matrix of the prisoner's dilemma. In a single-shot prisoner's dilemma, there is only one Nash equilibrium [154], where both agents defect. In the infinitely iterated prisoner's dilemma, the folk theorem [155] shows that there are infinitely many Nash equilibria. Two notable ones are the always defect strategy (DD), and tit-for-tat (TFT). In TFT each agent starts out with cooperation and then repeats the previous action of the opponent. The average returns per step in self-play are $-1$ and $-2$ for TFT and DD respectively.

Matching pennies [156] is a zero-sum game, with per-step payouts shown in Table 8.2. This game only has a single mixed strategy Nash equilibrium which is both players playing 50%/50% heads / tails.

| | Head | Tail |
|---|---|---|
| Head | (+1, -1) | (-1, +1) |
| Tail | (-1, +1) | (+1, -1) |

**Table 8.2:** Payoff matrix of matching pennies.

Agents in both the IPD and IMP can condition their actions on past history. Agents in an iterated game are endowed with a memory of length $K$, i.e., the agents act based on the results of the last $K$ rounds. Press and Dyson [157] proved that agents with a good memory-1 strategy can effectively force the iterated game to be played as memory-1. Thus, we consider memory-1 iterated games in our work.

We can model the memory-1 IPD and IMP as a two-agent MDP, where the state at time 0 is empty, denoted as $s_0$, and at time $t \geq 1$ is both agents' actions from $t-1$:

$$s_t = (u^1_{t-1}, u^2_{t-1}) \quad \text{for } t \geq 1.$$

Each agent's policy is fully specified by 5 probabilities. For agent $a$ in the case of the IPD, they are the probability of cooperation at game start $\pi^a(C|s_0)$, and the cooperation probabilities in the four memory states: $\pi^a(C|CC)$, $\pi^a(C|CD)$,

| | IPD | | IMP | |
|---|---|---|---|---|
| | %TFT | R(std) | %Nash | R(std) |
| NL-Ex. | 20.8 | -1.98(0.14) | 0.0 | 0(0.37) |
| LOLA-Ex. | 81.0 | -1.06(0.19) | 98.8 | 0(0.02) |
| NL-PG | 20.0 | -1.98(0.00) | 13.2 | 0(0.19) |
| LOLA-PG | 66.4 | -1.17(0.34) | 93.2 | 0(0.06) |

**Table 8.3:** We summarise results for NL vs. NL and LOLA vs. LOLA settings with either exact gradient evaluation (-Ex) or policy gradient approximation (-PG). Shown is the probability of agents playing TFT and Nash for the IPD and IMP respectively as well as the average reward per step, R, and standard deviation (std) at the end of training for 50 training runs.

$\pi^a(C|DC)$, and $\pi^a(C|DD)$. By analytically solving the multi-agent MDP we can derive each agent's future discounted reward as an analytical function of the agents' policies and calculate the exact policy update for both NL-Ex and LOLA-Ex agents.

We also organise a round-robin tournament where we compare LOLA-Ex to a number of state-of-the-art multi-agent learning algorithms, both on the and IMP.

### 8.4.2 Coin Game

Next, we study LOLA in a more high-dimensional setting called Coin Game. This is a sequential game and the agents' policies are parameterised as deep neural networks. Coin Game was first proposed by Lerer and Peysakhovich [139] as a higher dimensional alternative to the IPD with multi-step actions. As shown in Figure 8.3, in this setting two agents, 'red' and 'blue', are tasked with collecting coins.

The coins are either blue or red, and appear randomly on the grid-world. A new coin with random colour and random position appears after the last one is picked up. Agents pick up coins by moving onto the position where the coin is located. While every agent receives a point for picking up a coin of any colour, whenever an picks up a coin of different colour, the other agent loses 2 points.

As a result, if both agents greedily pick up any coin available, they receive 0 points in expectation. Since the agents' policies are parameterised as a recurrent neural network, one cannot obtain the future discounted reward as a function of both agents' policies in closed form. Policy gradient-based learning is applied for

both NL and LOLA agents in our experiments. We further include experiments of LOLA with opponent modelling (LOLA-OM) in order to examine the behaviour of LOLA agents without access to the opponent's policy parameters.

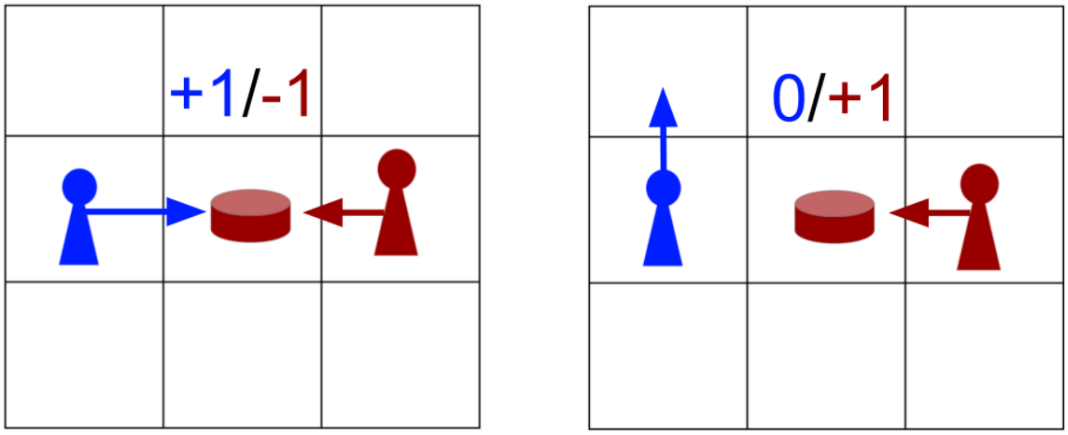



**Figure 8.3:** In the Coin Game, two agents, 'red' and 'blue', get 1 point for picking up any coin. However, the 'red agent' loses 2 points when the 'blue agent' picks up a red coin and vice versa. Effectively, this is a world with an embedded social dilemma where cooperation and defection are temporally extended.

### 8.4.3 Training Details

In policy gradient-based NL and LOLA settings, we train agents with an actor-critic method [21] and parameterise each agent with a policy actor and -critic for variance reduction during policy updates.

During training, we use gradient descent with step size, $\delta$, of 0.005 for the actor, 1 for the critic, and the batch size 4000 for rollouts. The discount rate $\gamma$ is set to 0.96 for the prisoner's dilemma and Coin Game and 0.9 for matching pennies. The high value of $\gamma$ for Coin Game and the IPD was chosen in order to allow for long time horizons, which are known to be required for cooperation. We found that a lower $\gamma$ produced more stable learning on IMP.

For Coin Game the agent's policy architecture is a recurrent neural network with 32 hidden units and 2 convolutional layers with $3 \times 3$ filters, stride 1, and ReLU activation for input processing. The input is presented as a 4-channel grid, with 2 channels encoding the positions of the 2 agents and 2 channels for the red and blue coins respectively.

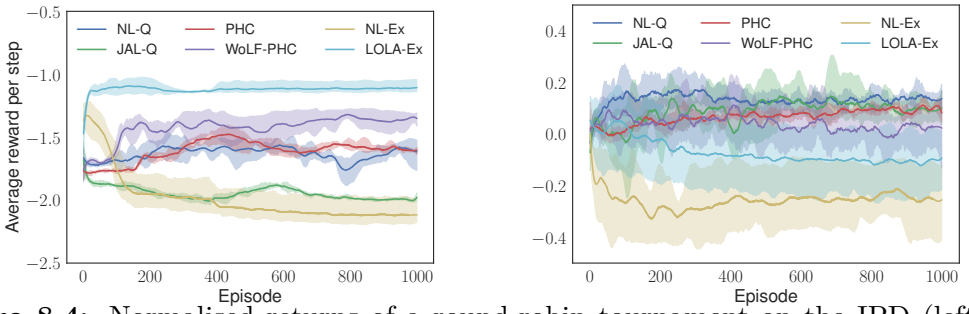



**Figure 8.4:** Normalised returns of a round-robin tournament on the IPD (left) and IMP (right). LOLA-Ex agents achieve the best performance in the IPD and are within error bars for IMP. Shading indicates a 95% confidence interval of the error of the mean. Baselines from [129]: naive Q-learner (NL-Q), joint-action Q-learner (JAL-Q), policy hill-climbing (PHC), and "Win or Learn Fast" (WoLF).

For the tournament, we use baseline algorithms and the corresponding hyperparameter values as provided in the literature [129]. The tournament is played in a round-robin fashion between all pairs of agents for 1000 episodes, 200 steps each.

## 8.5 Results

In this section, we summarise the experimental results. We denote LOLA and naive agents with exact policy updates as LOLA-Ex and NL-Ex respectively. We abbreviate LOLA and native agents with policy updates with LOLA-PG and NL-PG. We aim to answer the following questions:

1. How do pairs of LOLA-Ex agents behave in iterated games compared with pairs of NL-Ex agents?

2. Using policy gradient updates instead, how to LOLA-PG agents and NL-PG agents behave?

3. How do LOLA-Ex agents fair in a round robin tournament involving a set of multi-agent learning algorithms from literature?

4. Does the learning of LOLA-PG agents scale to high-dimensional settings where the agents' policies are parameterised by deep neural networks?

5. Does LOLA-PG maintain its behaviour when replacing access to the exact parameters of the opponent agent with opponent modelling?

6. Can LOLA agents be exploited by using higher order gradients, i.e., does LOLA lead to an arms race of ever higher order corrections or is LOLA / LOLA stable?

We answer the first three questions in Sec. 8.5.1, the next two questions in Sec. 8.5.2, and the last one in Sec. 8.5.3.

### 8.5.1 Iterated Games

We first compare the behaviours of LOLA agents with NL agents, with either exact policy updates or policy gradient updates.

Figures 8.1a) and 8.1b) show the policy for both agents at the end of training under NL-Ex and LOLA-Ex when the agents have access to exact gradients and Hessians of $\{J^1, J^2\}$. Here we consider the settings of NL-Ex vs. NL-Ex and LOLA-Ex vs. LOLA-Ex. We study mixed learning of one LOLA-Ex agent vs. an NL-Ex agent in Section 8.5.3. Under NL-Ex, the agents learn to defect in all states, indicated by the accumulation of points in the bottom left corner of the plot. However, under LOLA-Ex, in most cases the agents learn TFT. In particular agent 1 cooperates in the starting state $s_0$, $CC$ and $DC$, while agent 2 cooperates in $s_0$, $CC$ and $CD$. As a result, Figure 8.1c) shows that the normalised discounted reward[1] is close to $-1$ for LOLA-Ex vs. LOLA-Ex, corresponding to TFT, while NL-Ex vs. NL-Ex results in an normalised discounted reward of $-2$, corresponding to the fully defective ($DD$) equilibrium. Figure 8.1d) shows the normalised discounted reward for NL-PG and LOLA-PG where agents learn via policy gradient. LOLA-PG also demonstrates cooperation while agents defect in NL-PG.

We conduct the same analysis for IMP in Figure 8.2. In this game, under naive learning the agents' strategies fail to converge. In contrast, under LOLA the agents' policies converge to the only Nash equilibrium, playing 50%/50% heads/tails.

Table 8.3 summarises the numerical results comparing LOLA with NL agents in both the exact and policy gradient settings in the two iterated game environments. In the IPD, LOLA agents learn policies consistent with TFT with a much higher

[1]We use following definition for the normalised discounted reward: $(1-\gamma)\sum_{t=0}^{T}\gamma^t r_t$.

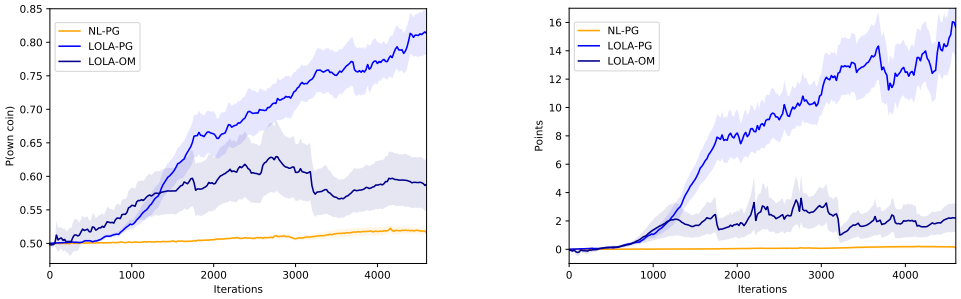



**Figure 8.5:** The percentage of all picked up coins that match in colour (left) and the total points obtained per episode (right) for a pair of naive learners using policy gradient (NL-PG), LOLA-agents (LOLA-PG), and a pair of LOLA-agents with opponent modelling (LOLA-OM). Also shown is the standard error of the mean (shading), based on 30 training runs. While LOLA-PG and LOLA-OM agents learn to cooperate, LOLA-OM is less stable and obtains lower returns than LOLA-PG. Best viewed in colour.

probability and achieve higher normalised discounted rewards than NL ($-1.06$ vs $-1.98$). In IMP, LOLA agents converge to the Nash equilibrium more stably while NL agents do not. The difference in stability is illustrated by the high variance of the normalised discounted returns for NL agents compared to the low variance under LOLA (0.37 vs 0.02).

In Figure 8.4 we show the average normalised return of our LOLA-Ex agent against a set of learning algorithms from the literature. We find that LOLA-Ex receives the highest normalised return in the IPD, indicating that it successfully shapes the learning outcome of other algorithms in this general sum setting.

In the IMP, LOLA-Ex achieves stable performance close to the middle of the distribution of results.

### 8.5.2 Coin Game

We summarise our experimental results in the Coin Game environment. To examine the scalability of LOLA learning rules, we compare NL-PG vs. NL-PG to LOLA-PG vs. LOLA-PG. Figure 8.5 demonstrates that NL-PG agents collect coins indiscriminately, corresponding to defection. In contrast, LOLA-PG agents learn to pick up coins predominantly (around 80%) of their own colour, showing that the LOLA learning rule leads to cooperation in the Coin Game as well.

Removing the assumption that agents can access the exact parameters of opponents, we examine LOLA agents with opponent modelling (Section 8.3.4). Figure 8.5 demonstrates that without access to the opponent's policy parameters, LOLA agents with opponent modelling pick up coins of their own colour around 60% of the time, inferior to the performance of LOLA-PG agents. We emphasise that with opponent modelling neither agent can recover the exact policy parameters of the opponent, since there is a large amount of redundancy in the neural network parameters. For example, each agent could permute the weights of their fully connected layers. Opponent modelling introduces noise in the opponent agent's policy parameters, thus increasing the variance and bias of the gradients (8.3.7) during policy updates, which leads to inferior performance of LOLA-OM vs. LOLA-PG in Figure 8.5.

### 8.5.3 Exploitability of LOLA

In this section we address the exploitability of the LOLA learning rule. We consider the IPD, where one can calculate the exact value function of each agent given the policies. Thus, we can evaluate the higher-order LOLA terms. We pitch a NL-Ex or LOLA-Ex agent against NL-Ex, LOLA-Ex, and a 2nd-order LOLA agent. We compare the normalised discounted return of each agent in all settings and address the question of whether there is an arms race to incorporate ever higher orders of LOLA correction terms.

Table 8.4 shows that a LOLA-Ex learner can achieve higher payouts against NL-Ex. Thus, there is an incentive for either agent to switch from naive learning to first order LOLA. Furthermore, two LOLA-Ex agents playing against each other both receive higher normalised discounted reward than a LOLA-Ex agent playing against a NL-Ex. This makes LOLA a dominant learning rule in the IPD compared to naive learning. We further find that 2nd-order LOLA provides no incremental gains when playing against a LOLA-Ex agent, leading to a reduction in payouts for both agents. These experiments were carried out with a $\delta$ of 0.5. While it is

beyond the scope of this work to prove that LOLA vs LOLA is a dominant learning rule in the space of gradient-based rules, these initial results are encouraging.

| | NL-Ex | LOLA-Ex | 2nd-Order |
|---|---|---|---|
| NL-Ex | (-1.99, -1.99) | (-1.54, -1.28) | (-1.46, -1.46) |
| LOLA-Ex | (-1.28, -1.54) | (-1.04, -1.04) | (-1.14, -1.17) |

**Table 8.4:** Higher-order LOLA results on the IPD. A LOLA-Ex agent obtains higher normalised return compared to a NL-Ex agent. However in this setting there is no incremental gain from using higher-order LOLA in order to exploit another LOLA agent in the IPD. In fact both agents do worse with the 2nd order LOLA (incl. 3rd order corrections).

## 8.6 Conclusion & Future Work

We presented Learning with Opponent-Learning Awareness (LOLA), a learning method for multi-agent settings that considers the learning processes of other agents. We show that when both agents have access to exact value functions and apply the LOLA learning rule, cooperation emerges based on tit-for-tat in the infinitely repeated iterated prisoner's dilemma while independent naive learners defect. We also find that LOLA leads to stable learning of the Nash equilibrium in IMP. In our round-robin tournament against other multi-agent learning algorithms we show that exact LOLA agents achieve the highest average returns on the IPD and respectable performance on IMP. We also derive a policy gradient-based version of LOLA, applicable to a DMARL setting. Experiments on the IPD and IMP demonstrate similar learning behaviour to the setting with exact value function.

In addition, we scale the policy gradient-based version of LOLA to the *Coin Game*, a multi-step game that requires deep recurrent policies. LOLA agents learn to cooperate, as agents pick up coins of their own colour with high probability while naive learners pick up coins indiscriminately. We further remove agents' access to the opponent agents' policy parameters and replace with opponent modelling. While less reliable, LOLA agents with opponent modelling also learn to cooperate.

We briefly address the exploitability of LOLA agents. Empirical results show that in the IPD both agents are incentivised to use LOLA, while higher order exploits show no further gain.

In the future, we would like to continue to address the exploitability of LOLA, when adversarial agents explicitly aim to take advantage of a LOLA learner using global search methods rather than just gradient-based methods. Just as LOLA is a way to exploit a naive learner, there should be means of exploiting LOLA learners in turn, unless LOLA is itself an equilibrium learning strategy.

# 9

# DiCE: The Infinitely Differentiable Monte Carlo Estimator

## Contents



## 9.1 Introduction

In the previous chapter we presented *Learning with Opponent-Learning Awareness* (LOLA), a method in which an agent accounts for the learning behaviour of other agents in the environment in its own optimisation step. In order to do so, the agents need to differentiate through the learning step of other agents, leading to the appearance of higher order gradients in the optimisation. Since LOLA is placed

in the context of RL, which optimises a non-differentiable objective, these terms need to be estimated using samples from the environment.

Estimating the first order gradients in these settings is computationally and conceptually simple: We can simply produce Monte Carlo estimates of gradients, *e.g.*, using to score function trick. While the gradient estimators can be directly defined, it is often more convenient to define an objective whose derivative is the gradient estimator and let the powerful automatic-differentiation (auto-diff) toolbox as implemented in deep learning libraries do the work for you. This is the method used by the *surrogate loss* (SL) approach [31], which provides a recipe for building a surrogate objective from a *stochastic computation graph* (SCG). When differentiated, the SL produces an estimator for the first order gradient of the original objective.

However, estimating higher order gradients is more challenging. Beyond being required for LOLA, such estimators are also useful for a number of optimisation techniques, accelerating convergence in supervised settings [158]. Furthermore, they are vital for gradient-based meta-learning [159–161], which differentiates an objective after some number of first order learning steps.

Unfortunately, the first order gradient estimators mentioned above are fundamentally ill-suited for calculating higher order derivatives via auto-diff. Due to the dependency on the sampling distribution, higher order gradient estimators require repeated application of the score function trick. Simply differentiating the first order estimator again, as was for example done by Finn, Abbeel, and Levine [159], leads to missing terms.

To obtain higher order score function gradient estimators, there are currently two unsatisfactory options. The first is to analytically derive and implement the estimators. However, this is laborious, error prone, and does not comply with the auto-diff paradigm. The second is to repeatedly apply the SL approach to construct new objectives for each further gradient estimate. However, constructing each of these new objectives involves increasingly complex graph manipulations, defeating the appeal of using a differentiable surrogate loss.

Moreover, to match the first order gradient after a single differentiation, the SL treats part of the cost as a fixed sample, severing the dependency on the parameters. We show that this yields missing and incorrect terms in higher order gradient estimators. We believe that these difficulties have limited the usage and exploration of higher order methods in reinforcement learning tasks and other application areas that may be formulated as SCGs.

Therefore, we propose a novel technique, the *Infinitely **Di**fferentiable Monte-**C**arlo **E**stimator* (DiCE), to address all these shortcomings. DiCE constructs a single objective that evaluates to an estimate of the original objective, but can also be differentiated repeatedly to obtain correct gradient estimators of any order. Unlike the SL approach, DiCE relies on auto-diff as implemented for instance in TensorFlow [162] or PyTorch [163] to automatically perform the complex graph manipulations required for these higher order gradient estimators.

DiCE uses a novel operator, MagicBox ($\boxdot$), which wraps around the set of those stochastic nodes $\mathcal{W}_c$ that influence each of the original losses in an SCG. Upon differentiation, this operator generates the correct gradients associated with the sampling distribution:

$$\nabla_\theta \boxdot(\mathcal{W}_c) = \boxdot(\mathcal{W}_c) \nabla_\theta \sum_{w \in \mathcal{W}_c} \log(p(w;\theta)),$$

while returning 1 when evaluated: $\boxdot(\mathcal{W}) \rightarrowtail 1$. The MagicBox-operator can easily be implemented in standard deep learning libraries as follows:

$$\boxdot(\mathcal{W}) = \exp\Big(\tau - \bot(\tau)\Big),$$
$$\tau = \sum_{w \in \mathcal{W}} \log(p(w;\theta)),$$

where $\bot$ is an operator that sets the gradient of the operand to zero, so $\nabla_x \bot(x) = 0$. In addition, we show how to use a baseline for variance reduction in our formulation.

We verify the correctness of DiCE both through a proof and through numerical evaluation of the DiCE gradient estimates. We also open-source our code in TensorFlow. Last not least, to demonstrate the utility of DiCE, we also propose a novel approach for LOLA. Using DiCE we can express LOLA in a formulation

that closely resembled meta-learning. We hope this powerful and convenient novel objective will unlock further exploration and adoption of higher order learning methods in meta-learning, reinforcement learning, and other applications of SCGs.

## 9.2 Background

Suppose $x$ is a random variable, $x \sim p(x;\theta)$, $f$ is a function of $x$ and we want to compute $\nabla_\theta \mathbb{E}_x[f(x)]$. If the analytical gradients $\nabla_\theta f$ are unavailable or nonexistent, we can employ the *score function* (SF) estimator:

$$\nabla_\theta \mathbb{E}_x[f(x)] = \mathbb{E}_x[f(x)\nabla_\theta \log(p(x;\theta))] \tag{9.2.1}$$

If instead $x$ is a deterministic function of $\theta$ and another random variable $z$, the operators $\nabla_\theta$ and $\mathbb{E}_z$ commute, yielding the *pathwise derivative estimator* or *reparameterisation trick*. In this work, we focus on the SF estimator, which can capture the interdependency of both the objective and the sampling distribution on the parameters $\theta$, and therefore requires careful handling for higher order gradient estimates.

### 9.2.1 Stochastic Computation Graphs

Gradient estimators for single random variables can be generalised using the formalism of a stochastic computation graph [SCG, 31]. An SCG is a directed acyclic graph with four types of nodes: *input nodes*, $\Theta$; *deterministic nodes*, $\mathcal{D}$; *cost nodes*, $\mathcal{C}$; and *stochastic nodes*, $\mathcal{S}$. Input nodes are set externally and can hold parameters we seek to optimise. Deterministic nodes are functions of their parent nodes, while stochastic nodes are distributions conditioned on their parent nodes. The set of cost nodes $\mathcal{C}$ are those associated with an objective $\mathcal{L} = \mathbb{E}[\sum_{c\in\mathcal{C}} c]$.

Let $v \prec w$ denote that node $v$ *influences* node $w$, *i.e.*, there exists a path in the graph from $v$ to $w$. If every node along the path is deterministic, $v$ influences $w$ deterministically which is denoted by $v \prec^D w$. See Figure 9.1 (top) for a simple SCG with an input node $\theta$, a stochastic node $x$ and a cost function $f$. Note that $\theta$ influences $f$ deterministically ($\theta \prec^D f$) as well as stochastically via $x$ ($\theta \prec f$).

### 9.2.2 Surrogate Losses

In order to estimate gradients of a sum of cost nodes, $\sum_{c \in \mathcal{C}} c$, in an arbitrary SCG, Schulman et al. [31] introduce the notion of a *surrogate loss* (SL):

$$\mathrm{SL}(\Theta, \mathcal{S}) := \sum_{w \in \mathcal{S}} \log p(w \mid \textsc{deps}_w) \hat{Q}_w + \sum_{c \in \mathcal{C}} c(\textsc{deps}_c)$$

Here $\textsc{deps}_w$ are the "dependencies" of $w$: the set of stochastic or input nodes that deterministically influence the node $w$. Furthermore, $\hat{Q}_w$ is the sum of *sampled* costs $\hat{c}$ corresponding to the cost nodes influenced by $w$.

The SL produces a gradient estimator when differentiated once [31, Corollary 1]:

$$\nabla_\theta \mathcal{L} = \mathbb{E}[\nabla_\theta \mathrm{SL}(\Theta, \mathcal{S})]. \tag{9.2.2}$$

The hat notation on $\hat{Q}_w$ indicates that, inside the SL, these costs are treated as fixed samples, thus severing the functional dependency on $\theta$ that was present in the original stochastic computation graph. This ensures that the first order gradients of the SL match the score function estimator, which does not contain a term of the form $\log(p)\nabla_\theta Q$.

Although Schulman et al. [31] focus on first order gradients, they argue that the SL gradient estimates themselves can be treated as costs in an SCG and that the SL approach can be applied repeatedly to construct higher order gradient estimators. However, the use of sampled costs in the SL leads to missing dependencies and wrong estimates when calculating such higher order gradients, as we discuss in Section 9.3.2.

## 9.3 Higher Order Gradients

In this section, we illustrate how to estimate higher order gradients via repeated application of the score function (SF) trick and show that repeated application of the surrogate loss (SL) approach in stochastic computation graphs (SCGs) fails to capture all of the relevant terms for higher order gradient estimates.

### 9.3.1 Higher Order Gradient Estimators

We begin by revisiting the derivation of the score function estimator for the gradient of the expectation $\mathcal{L}$ of $f(x;\theta)$ over $x \sim p(x;\theta)$:

$$\begin{aligned}
\nabla_\theta \mathcal{L} &= \nabla_\theta \mathbb{E}_x \left[ f(x;\theta) \right] \\
&= \nabla_\theta \sum_x p(x;\theta) f(x;\theta) \\
&= \sum_x \nabla_\theta \Big( p(x;\theta) f(x;\theta) \Big) \\
&= \sum_x \Big( f(x;\theta) \nabla_\theta p(x;\theta) + p(x;\theta) \nabla_\theta f(x;\theta) \Big) \\
&= \sum_x \Big( f(x;\theta) p(x;\theta) \nabla_\theta \log(p(x;\theta)) \\
&\qquad + p(x;\theta) \nabla_\theta f(x;\theta) \Big) \\
&= \mathbb{E}_x \left[ f(x;\theta) \nabla_\theta \log(p(x;\theta)) + \nabla_\theta f(x;\theta) \right] \qquad (9.3.1) \\
&= \mathbb{E}_x [g(x;\theta)].
\end{aligned}$$

The estimator $g(x;\theta)$ of the gradient of $\mathbb{E}_x \left[ f(x;\theta) \right]$ consists of two distinct terms: (1) The term $f(x;\theta)\nabla_\theta \log(p(x;\theta))$ originating from $f(x;\theta)\nabla_\theta p(x;\theta)$ via the SF trick, and (2) the term $\nabla_\theta f(x;\theta)$, due to the direct dependence of $f$ on $\theta$. The second term is often ignored because $f$ is often only a function of $x$ but not of $\theta$. However, even in that case, the gradient estimator $g$ depends on both $x$ and $\theta$. We might be tempted to again apply the SL approach to $\nabla_\theta \mathbb{E}_x[g(x;\theta)]$ to produce estimates of higher order gradients of $\mathcal{L}$, but below we demonstrate that this fails. In Section 9.4, we subsequently introduce a practical algorithm for correctly producing such higher order gradient estimators in SCGs.

### 9.3.2 Higher Order Surrogate Losses

While Schulman et al. [31] focus on the first order gradients, they state that a recursive application of SL can generate higher order gradient estimators. However, as we demonstrate in this section, because the SL approach treats part of the objective as a sampled cost, the corresponding terms lose a functional dependency on the sampling distribution. This leads to missing terms in the estimators of the higher order gradients.

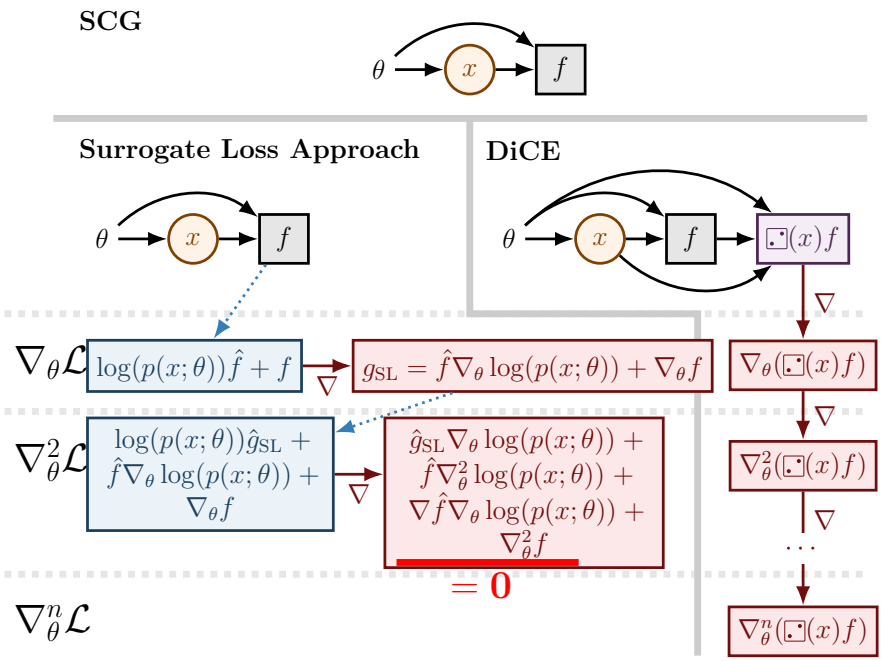



**Figure 9.1:** Simple example illustrating the difference of the Surrogate Loss (SL) approach to DiCE. Stochastic nodes are depicted in orange, costs in gray, surrogate losses in blue, DiCE in purple, and gradient estimators in red. Note that for second-order gradients, SL requires the construction of an intermediate stochastic computation graph and due to taking a sample of the cost $\hat{g}_{\text{SL}}$, the dependency on $\theta$ is lost, leading to an incorrect second-order gradient estimator. Arrows from $\theta, x$ and $f$ to gradient estimators omitted for clarity.

Consider the following example, where a single parameter $\theta$ defines a sampling distribution $p(x;\theta)$ and the objective is $f(x,\theta)$.

$$\begin{aligned}
\text{SL}(\mathcal{L}) &= \log p(x;\theta)\hat{f}(x) + f(x;\theta) \\
(\nabla_\theta \mathcal{L})_{\text{SL}} &= \mathbb{E}_x[\nabla_\theta \text{SL}(\mathcal{L})] \\
&= \mathbb{E}_x[\hat{f}(x)\nabla_\theta \log p(x;\theta) + \nabla_\theta f(x;\theta)] \\
&= \mathbb{E}_x[g_{\text{SL}}(x;\theta)].
\end{aligned} \tag{9.3.2}$$

The corresponding SCG is depicted at the top of Figure 9.1. Comparing (9.3.1) and (9.3.2), note that the first term, $\hat{f}(x)$ has lost its functional dependency on $\theta$, as indicated by the hat notation and the lack of a $\theta$ argument. While these terms evaluate to the same estimate of the first order gradient, the lack of the functional dependency yields a discrepancy between the exact derivation of the

second order gradient and a second application of SL.

$$\begin{aligned}\mathrm{SL}(g_{\mathrm{SL}}(x;\theta)) &= \log p(x;\theta)\hat{g}_{\mathrm{SL}}(x) + g_{\mathrm{SL}}(x;\theta)\\ (\nabla_\theta^2\mathcal{L})_{\mathrm{SL}} &= \mathbb{E}_x[\nabla_\theta \mathrm{SL}(g_{\mathrm{SL}})]\\ &= \mathbb{E}_x[\hat{g}_{\mathrm{SL}}(x)\nabla_\theta \log p(x;\theta) + \nabla_\theta g_{\mathrm{SL}}(x;\theta)]. \end{aligned} \tag{9.3.3}$$

By contrast, the exact derivation of $\nabla_\theta^2\mathcal{L}$ results in the following expression:

$$\begin{aligned}\nabla_\theta^2\mathcal{L} &= \nabla_\theta \mathbb{E}_x[g(x;\theta)]\\ &= \mathbb{E}_x[g(x;\theta)\nabla_\theta \log p(x;\theta) + \nabla_\theta g(x;\theta)]. \end{aligned} \tag{9.3.4}$$

Since $g_{\mathrm{SL}}(x;\theta)$ differs from $g(x;\theta)$ only in its functional dependencies on $\theta$, $g_{\mathrm{SL}}$ and $g$ are identical when *evaluated*. However, due to the missing dependencies in $g_{\mathrm{SL}}$, the *gradients* w.r.t. $\theta$, which appear in the higher order gradient estimates in (9.3.3) and (9.3.4), differ:

$$\begin{aligned}\nabla_\theta g(x;\theta) &= \nabla_\theta f(x;\theta)\nabla_\theta \log(p(x;\theta))\\ &\quad + f(x;\theta)\nabla_\theta^2 \log(p(x;\theta))\\ &\quad + \nabla_\theta^2 f(x;\theta),\\ \nabla_\theta g_{\mathrm{SL}}(x;\theta) &= \hat{f}(x)\nabla_\theta^2 \log(p(x;\theta))\\ &\quad + \nabla_\theta^2 f(x;\theta).\end{aligned}$$

We lose the term $\nabla_\theta f(x;\theta)\nabla_\theta \log(p(x;\theta))$ in the second order SL gradient because $\nabla_\theta \hat{f}(x) = 0$ (see left part of Figure 9.1). This issue occurs immediately in the second order gradients when $f$ depends directly on $\theta$. However, as $g(x;\theta)$ always depends on $\theta$, the SL approach always fails to produce correct third or higher order gradient estimates even if $f$ depends only indirectly on $\theta$.

### 9.3.3 Simple Failing Example

Here is a toy example to illustrate a possible failure case. Let $x \sim \mathrm{Ber}(\theta)$ and $f(x,\theta) = x(1-\theta) + (1-x)(1+\theta)$. For this simple example we can exactly

evaluate all terms:

$$\begin{aligned}\mathcal{L} &= \theta(1-\theta) + (1-\theta)(1+\theta) \\ &= -2\theta^2 + \theta + 1 \\ \nabla_\theta \mathcal{L} &= -4\theta + 1 \\ \nabla^2_\theta \mathcal{L} &= -4\end{aligned}$$

Evaluating the expectations for the SL-gradient estimators analytically results in the following terms:

$$\begin{aligned}(\nabla_\theta \mathcal{L})_{\text{SL}} &= -4\theta + 1 \\ (\nabla^2_\theta \mathcal{L})_{\text{SL}} &= -2\end{aligned}$$

Even with an infinite number of samples, the SL estimator produces the wrong second order gradient. If, for example, these wrong estimates were used in combination with the Newton-Raphson method for optimising $\mathcal{L}$, then $\theta$ would never converge to the correct value. In contrast, this method would converge in a single step using the correct gradients.

The failure mode seen in this toy example will appear whenever the objective includes a regularisation term that depends on $\theta$, and is also impacted by the stochastic samples. One example in a practical algorithm is soft $Q$-learning for RL [164], which regularises the policy by adding an entropy penalty to the rewards. This penalty encourages the agent to maintain an exploratory policy, reducing the probability of getting stuck in local optima. Clearly the penalty depends on the policy parameters $\theta$. However, the policy entropy will also depends on the states visited, which in turn depend on the stochastically sampled actions. As a result, the entropy regularised RL objective in this algorithm will have the exact property leading to the failure of the SL approach shown above. Unlike our toy analytic example, the consequent errors will not just appear as a rescaling of the proper

higher order gradients, but will depend in a complex way on the parameters $\theta$. Any second order methods with such a regularised objective will therefore require an alternative strategy for generating gradient estimators, even setting aside the awkwardness of repeatedly generating new surrogate objectives.

## 9.4 Correct Gradient Estimators with DiCE

In this section, we propose the *Infinitely **Di**fferentiable Monte-**C**arlo **E**stimator* (DiCE), a practical algorithm for programmatically generating correct gradients of any order in arbitrary SCGs. The naive option is to recursively apply the update rules in (9.3.1) that map from $f(x;\theta)$ to the estimator of its derivative $g(x;\theta)$. However, this approach has two deficiencies: First, by defining gradients directly, it fails to provide an objective that can be used in standard deep learning libraries. Second, these naive gradient estimators violate the auto-diff paradigm for generating further estimators by repeated differentiation since in general $\nabla_\theta f(x;\theta) \neq g(x;\theta)$. Our approach addresses these issues, as well as fixing the missing terms from the SL approach.

As before, $\mathcal{L} = \mathbb{E}[\sum_{c\in\mathcal{C}} c]$ is the objective in an SCG. The correct expression for a gradient estimator that preserves all required dependencies for further differentiation is:

$$\nabla_\theta \mathcal{L} = \mathbb{E}\left[\sum_{c\in\mathcal{C}}\left(c\sum_{w\in\mathcal{W}_c}\nabla_\theta \log p(w \mid \text{DEPS}_w) + \nabla_\theta c(\text{DEPS}_c)\right)\right], \tag{9.4.1}$$

where $\mathcal{W}_c = \{w \mid w \in \mathcal{S}, w \prec c, \theta \prec w\}$, *i.e.* the set of stochastic nodes that depend on $\theta$ and influence the cost $c$. For brevity, from here on we suppress the DEPS notation, assuming all probabilities and costs are conditioned on their relevant ancestors in the SCG.

Note that (9.4.1) is the generalisation of (9.3.1) to arbitrary SCGs. The proof is given by Schulman et al. [31, Lines 1-10, Appendix A]. Crucially, in Line 11 the authors then replace $c$ by $\hat{c}$, severing the dependencies required for correct higher

order gradient estimators. As described in Section 9.2.2, this was done so that the SL approach reproduces the score function estimator after a single differentiation and can thus be used as an objective for backpropagation in a deep learning library.

To support correct higher order gradient estimators, we propose DiCE, which relies heavily on a novel operator, MagicBox ($\boxdot$). MagicBox takes a set of stochastic nodes $\mathcal{W}$ as input and has the following two properties by design:

1. $\boxdot(\mathcal{W}) \rightarrowtail 1$,

2. $\nabla_\theta \boxdot(\mathcal{W}) = \boxdot(\mathcal{W}) \sum_{w \in \mathcal{W}} \nabla_\theta \log(p(w; \theta))$.

Here, $\rightarrowtail$ indicates "*evaluates to*" in contrast to full equality, $=$, which includes equality of all gradients. In the auto-diff paradigm, $\rightarrowtail$ corresponds to a forward pass evaluation of a term. Meanwhile, the behaviour under differentiation in property (2) indicates the new graph nodes that will be constructed to hold the gradients of that object. Note that that $\boxdot(\mathcal{W})$ reproduces the dependency of the gradient on the sampling distribution under differentiation through the requirements above. Using $\boxdot$, we can next define the DiCE objective, $\mathcal{L}_{\boxdot}$:

$$\mathcal{L}_{\boxdot} = \sum_{c \in \mathcal{C}} \boxdot(\mathcal{W}_c) c. \tag{9.4.2}$$

Below we prove that the DiCE objective indeed produces correct arbitrary order gradient estimators under differentiation.

**Theorem 1.** $\mathbb{E}[\nabla_\theta^n \mathcal{L}_{\boxdot}] \rightarrowtail \nabla_\theta^n \mathcal{L}, \forall n \in \{0, 1, 2, \dots\}$.

*Proof.* For each cost node $c \in \mathcal{C}$, we define a sequence of nodes, $c^n, n \in \{0, 1, \dots\}$ as follows:

$$\begin{aligned} c^0 &= c, \\ \mathbb{E}[c^{n+1}] &= \nabla_\theta \mathbb{E}[c^n]. \end{aligned} \tag{9.4.3}$$

By induction it follows that $\mathbb{E}[c^n] = \nabla_\theta^n \mathbb{E}[c]\ \forall n$, *i.e.* that $c^n$ is an estimator of the $n$th order derivative of the objective $\mathbb{E}[c]$.

We further define $c^n_{\boxdot} = c^n \boxdot(\mathcal{W}_{c^n})$. Since $\boxdot(x) \rightarrowtail 1$, clearly $c^n_{\boxdot} \rightarrowtail c^n$. Therefore $\mathbb{E}[c^n_{\boxdot}] \rightarrowtail \mathbb{E}[c^n] = \nabla^n_\theta \mathbb{E}[c]$, *i.e.*, $c^n_{\boxdot}$ is also a valid estimator of the $n$th order derivative of the objective. Next, we show that $c^n_{\boxdot}$ can be generated by differentiating $c^0_{\boxdot}$ $n$ times. This follows by induction, if $\nabla_\theta c^n_{\boxdot} = c^{n+1}_{\boxdot}$, which we prove as follows:

$$\begin{aligned}
\nabla_\theta c^n_{\boxdot} &= \nabla_\theta (c^n \boxdot(\mathcal{W}_{c^n})) \\
&= c^n \nabla_\theta \boxdot(\mathcal{W}_{c^n}) + \boxdot(\mathcal{W}_{c^n}) \nabla_\theta c^n \\
&= c^n \boxdot(\mathcal{W}_{c^n}) \left( \sum_{w \in \mathcal{W}_{c^n}} \nabla_\theta \log(p(w;\theta)) \right) \\
&\quad + \boxdot(\mathcal{W}_{c^n}) \nabla_\theta c^n \\
&= \boxdot(\mathcal{W}_{c^n}) \left( \nabla_\theta c^n + c^n \sum_{w \in \mathcal{W}_{c^n}} \nabla_\theta \log(p(w;\theta)) \right) && (9.4.4) \\
&= \boxdot(\mathcal{W}_{c^{n+1}}) c^{n+1} = c^{n+1}_{\boxdot}. && (9.4.5)
\end{aligned}$$

To proceed from (9.4.4) to (9.4.5), we need two additional steps. First, we require an expression for $c^{n+1}$. Substituting $\mathcal{L} = \mathbb{E}[c^n]$ into (9.4.1) and comparing to (9.4.3), we find the following map from $c^n$ to $c^{n+1}$:

$$c^{n+1} = \nabla_\theta c^n + c^n \sum_{w \in \mathcal{W}_{c^n}} \nabla_\theta \log p(w;\theta). \tag{9.4.6}$$

The term inside the brackets in (9.4.4) is identical to $c^{n+1}$. Secondly, note that (9.4.6) shows that $c^{n+1}$ depends only on $c^n$ and $\mathcal{W}_{c^n}$. Therefore, the stochastic nodes which influence $c^{n+1}$ are the same as those which influence $c^n$. So $\mathcal{W}_{c^n} = \mathcal{W}_{c^{n+1}}$, and we arrive at (9.4.5).

To conclude the proof, recall that $c^n$ is the estimator for the $n$th derivative of $c$, and that $c^n_{\boxdot} \rightarrowtail c^n$. Summing over $c \in \mathcal{C}$ then gives the desired result. □

**Implementation of DiCE.** DiCE is easy to implement in standard deep learning libraries:

$$\begin{aligned}
\boxdot(\mathcal{W}) &= \exp\Big(\tau - \bot(\tau)\Big), \\
\tau &= \sum_{w \in \mathcal{W}} \log(p(w;\theta)),
\end{aligned}$$

where $\perp$ is an operator that sets the gradient of the operand to zero, so $\nabla_x \perp(x) = 0$. [1]

Since $\perp(x) \rightarrowtail x$, clearly $\boxdot(\mathcal{W}) \rightarrowtail 1$. Furthermore:

$$\begin{aligned}
\nabla_\theta \boxdot(\mathcal{W}) &= \nabla_\theta \exp\Big(\tau - \perp(\tau)\Big) \\
&= \exp\Big(\tau - \perp(\tau)\Big)\nabla_\theta(\tau - \perp(\tau)) \\
&= \boxdot(\mathcal{W})(\nabla_\theta \tau + 0) \\
&= \boxdot(\mathcal{W}) \sum_{w \in \mathcal{W}} \nabla_\theta \log(p(w;\theta)).
\end{aligned}$$

With this implementation of the $\boxdot$-operator, it is now straightforward to construct $\mathcal{L}_{\boxdot}$ as defined in (9.4.7). This procedure is demonstrated in Figure 9.2, which shows a reinforcement learning use case. In this example, the cost nodes are rewards that depend on stochastic actions, and the total objective is $J = \mathbb{E}[\sum r_t]$. We construct a DiCE objective $J_{\boxdot} = \sum_t \boxdot(\{a_{t'}, t' \leq t\})r_t$. Now $\mathbb{E}[J_{\boxdot}] \rightarrowtail J$ and $\mathbb{E}[\nabla_\theta^n J_{\boxdot}] \rightarrowtail \nabla_\theta^n J$, so $J_{\boxdot}$ can both be used to estimate the return and to produces estimators for any order gradients under auto-diff, which can be used for higher order methods such as TRPO [165].

**Causality.** In Theorem 1 of Schulman et al. [31], two expressions for the the gradient estimator are provided:

1. The first expression handles causality by summing over stochastic nodes, $w$, and multiplying $\nabla \log(p(w))$ for each stochastic node with a sum of the *downstream* costs, $\hat{Q}_w$. This is *forward looking* causality.

2. In contrast, the second expression sums over costs, $c$, and multiplies each cost with a sum over the gradients of log-probabilities from *upstream* stochastic nodes, $\sum_{w \in \mathcal{W}_c} \nabla \log(p(w))$. We can think of this as *backward looking* causality.

In both cases, integrating causality into the gradient estimator leads to reduction of variance compared to the naive approach of multiplying the full sum over costs with the full sum over grad-log-probabilities.

[1] This operator exists in PyTorch as `detach` and in TensorFlow as `stop_gradient`.

While the SL approach is based on the first expression, DiCE uses the second formulation. As shown by Schulman et al. [31], both expressions result in the same terms for the gradient estimator. However, the second formulation leads to greatly reduced conceptual complexity when calculating higher order terms, which we exploit in the definition of the DiCE objective. This is because each further gradient estimator maintains the same backward looking dependencies for each term in the original sum over costs, *i.e.*, $\mathcal{W}_{c^n} = \mathcal{W}_{c^{n+1}}$. In contrast, the SL approach is centred around the stochastic nodes, which each become associated with a growing number of downstream costs after each differentiation. Consequently, we believe that our DiCE objective is more intuitive, as it is conceptually centred around the original objective and remains so under repeated differentiation.

**First Order Variance Reduction.** So far, the construction of the DiCE objective addresses variance reduction only via implementing causality, since each cost term is associated with the $\boxdot$ that captures all causal dependencies. However, we can also include a baseline term in the definition of the DiCE objective:

$$\mathcal{L}_{\boxdot} = \sum_{c \in \mathcal{C}} \boxdot(\mathcal{W}_c) c + \sum_{w \in \mathcal{S}} (1 - \boxdot(\{w\})) b_w. \tag{9.4.7}$$

The baseline $b_w$ is a design choice and can be any function of nodes not influenced by $w$. As long as this condition is met, the baseline will not change the expectation of the gradient estimates, but can considerably reduce the variance (including of higher order gradient estimators). A common choice is the average cost.

Since $(1 - \boxdot(\{w\})) \rightarrowtail 0$, the addition of the baseline leaves the evaluation of the estimator $\mathcal{L}_{\boxdot}$ of the original objective unchanged, while reproducing the baseline term stated by Schulman et al. [31] after differentiation.

**Hessian-Vector Product.** The Hessian-vector, $v^\top H$, is useful for a number of algorithms, such as estimation of eigenvectors and eigenvalues of $H$ [166]. Using DiCE, $v^\top H$ can be implemented efficiently without having to compute the full

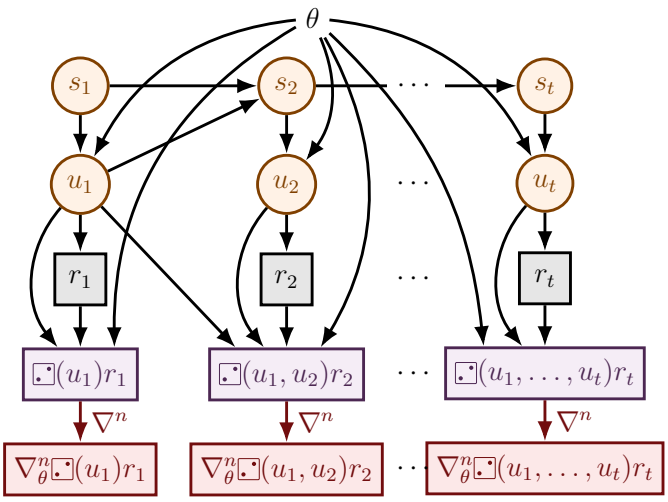



**Figure 9.2:** DiCE applied to a reinforcement learning problem. A stochastic policy conditioned on $s_t$ and $\theta$ produces actions, $u_t$, which lead to rewards $r_t$ and next states, $s_{t+1}$. Associated with each reward is a DiCE objective that takes as input the set of all causal dependencies that are functions of $\theta$, *i.e.*, the actions. Arrows from $\theta, u_i$ and $r_i$ to gradient estimators omitted for clarity.

Hessian. Assuming $v$ does not depend on $\theta$ and using $^\top$ to indicate the transpose:

$$\begin{aligned} v^\top H &= v^\top \nabla^2 \mathcal{L}_{\boxdot} \\ &= v^\top (\nabla^\top \nabla \mathcal{L}_{\boxdot}) \\ &= \nabla^\top (v^\top \nabla \mathcal{L}_{\boxdot}). \end{aligned}$$

In particular, $(v^\top \nabla \mathcal{L}_{\boxdot})$ is a scalar, making this implementation well suited for auto-diff.

## 9.5 Case Studies

While the main contribution of this chapter is to provide a novel general approach for any order gradient estimation in SCGs, we also provide a proof-of-concept empirical evaluation for a set of case studies, carried out on the *iterated prisoner's dilemma* (IPD) as introduced in the previous chapter. This setting is useful because (1) it has a nontrivial but analytically calculable value function, allowing for verification of gradient estimates, and (2) differentiating through the learning steps of other agents in multi-agent RL is a highly relevant application of higher order policy gradient estimators in RL.

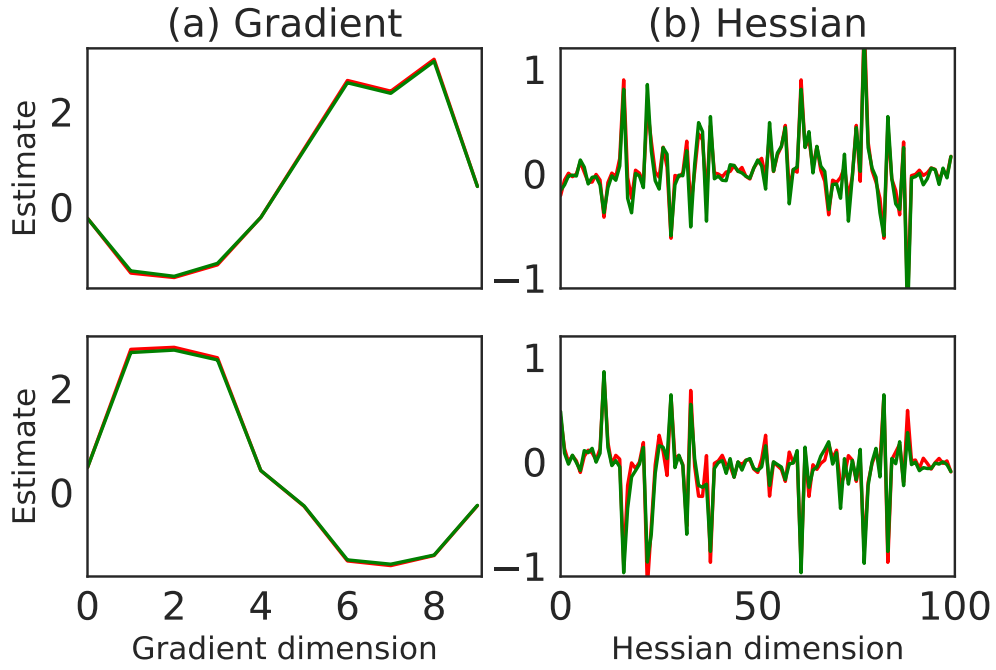



**Figure 9.3:** For each of the two agents (1 top row, 2 bottom row) in the iterated prisoner's dilemma, shown is the flattened true (red) and estimated (green) Gradient (left) and Hessian (right) using the first and second derivative of DiCE and the exact value function respectively. The correlation coefficients are 0.999 for the gradients and 0.97 for the Hessian; the sample size is 100k.

**Empirical Verification.** We first verify that DiCE successfully recovers gradients and Hessians in stochastic computation graphs. To do so, we use DiCE to estimate gradients and Hessians of the expected return for fixed policies in IPD.

As shown in Figure 9.3, we find that indeed the DiCE estimator matches both the gradients (a) and the Hessians (b) for both agents accurately. Furthermore, Figure 9.4 shows how the estimate of the gradient improve as the value function becomes more accurate during training, in (a). Also shown is the quality of the gradient estimation as a function of sample size with and without a baseline, in (b). Both plots show that the baseline is a key component of DiCE for accurate estimation of gradients.

**DiCE for multi-agent RL.** In the previous chapter we introduced *learning with opponent-learning awareness* (LOLA), and showed that agents which differentiate through the learning step of their opponent converge to Nash equilibria with higher social welfare in the IPD.

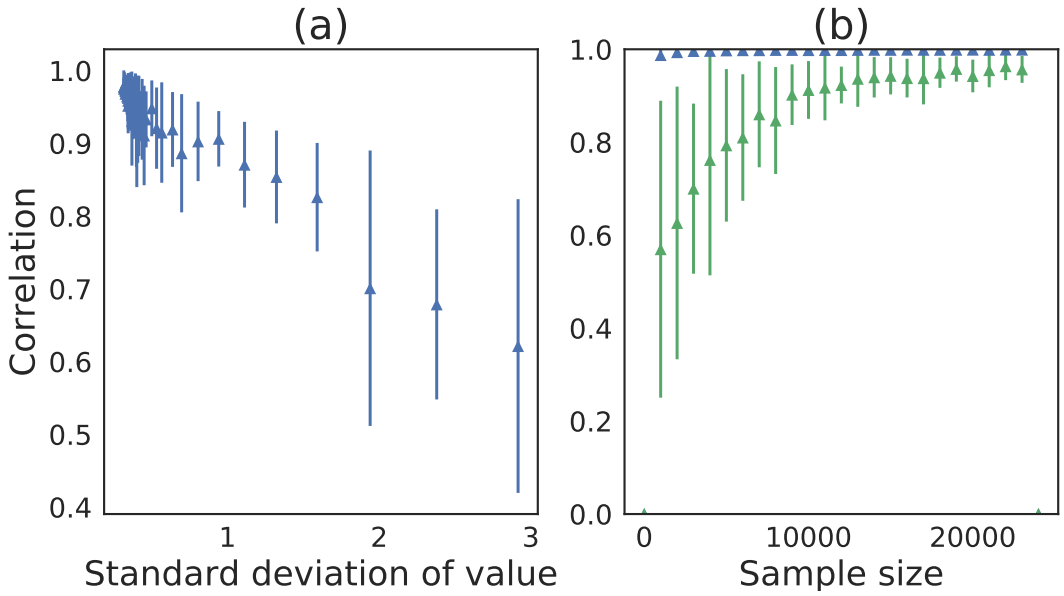



**Figure 9.4:** Shown in (a) is the correlation of the gradient estimator (averaged across agents) as a function of the estimation error of the baseline when using a sample size of 128 and in (b) as a function of sample size when using a converged baseline (in blue) and no baseline (in green). In both plots errors bars indicate the standard deviation.

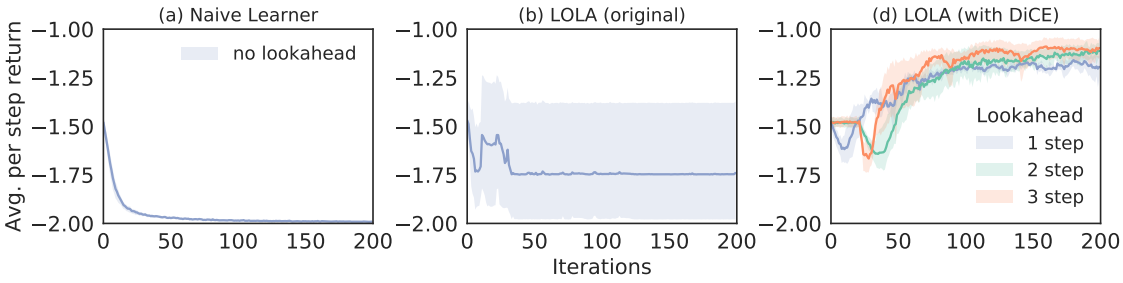



**Figure 9.5:** Joint average per step returns for different training methods. (a) Agents naively optimize expected returns w.r.t. their policy parameters only, without lookahead steps. (b) The original LOLA algorithm, see Chapter 8, that uses gradient corrections. (c) LOLA-DiCE with lookahead of up to 3 gradient steps. Shaded areas represent the 95% confidence intervals based on 5 runs. All agents used batches of size 64, which is more than 60 times smaller than the size required in the original LOLA method.

Since the standard policy gradient learning step for one agent has no dependency on the parameters of the other agent (which it treats as part of the environment), LOLA relies on a Taylor expansion of the expected return in combination with an analytical derivation of the second order gradients to be able to differentiate through the expected return after the opponent's learning step.

In this chapter we take a more direct approach, made possible by DiCE. Let

$\pi_{\theta^1}$ be the policy of the LOLA agent and let $\pi_{\theta^2}$ be the policy of its opponent and vice versa. Assuming that the opponent learns using policy gradients, LOLA-DiCE agents learn by directly optimising the following stochastic objective w.r.t. $\theta^1$:

$$\begin{aligned} \mathcal{L}^1(\theta^1, \theta^2)_{\text{LOLA}} &= \mathbb{E}_{\pi_{\theta^1}, \pi_{\theta^2 + \Delta\theta^2(\theta^1,\theta^2)}} \left[\mathcal{L}^1\right], \text{where} \\ \Delta\theta^2(\theta^1, \theta^2) &= \alpha \nabla_{\theta^2} \mathbb{E}_{\pi_{\theta^1}, \pi_{\theta^2}} \left[\mathcal{L}^2\right], \end{aligned} \tag{9.5.1}$$

Here $\alpha$ is a scalar step size and $\mathcal{L}^a = \sum_{t=0}^{T} \gamma^t r_t^a$ is the sum of discounted rewards for agent $a$. To follow the SCG formalism, here and for the rest of the the case study we use $\mathcal{L}^a$ rather than $J^a$ to describe the return.

To evaluate these terms directly, our variant of LOLA unrolls the learning process of the opponent, which is functionally similar to model-agnostic meta-learning [MAML, 159]. In the MAML formulation, the gradient update of the opponent, $\Delta\theta^2$, corresponds to the inner loop (typically training objective) and the gradient update of the agent itself to the outer loop (typically test objective).

Using the following DiCE-objective to estimate gradient steps for agent $a$, we are able preserve all dependencies:

$$\mathcal{L}^a_{\boxdot(\theta^1,\theta^2)} = \sum_t \boxdot \left( \left\{ u_{t' \leq t}^{a' \in \{1,2\}} \right\} \right) \gamma^t r_t^a, \tag{9.5.2}$$

where $\left\{ u_{t' \leq t}^{a' \in \{1,2\}} \right\}$ is the set of all actions taken by both agents up to time $t$. We note that for computational reasons, we cache the $\Delta\theta^a$ of the inner loop when unrolling the outer loop policies in order to avoid recalculating them at every time step.

Importantly, using DiCE, differentiating through $\Delta\theta^2$ produces the correct higher order gradients, which is vital for LOLA to function. In contrast, simply differentiating through the SL-based first order gradient estimator again, as was done for MAML by Finn, Abbeel, and Levine [159], results in omitted gradient terms and a biased gradient estimator, as pointed out by Al-Shedivat et al. [160] and Stadie et al. [167].

Figure 9.5 shows a comparison between the LOLA-DiCE agents and the original formulation of LOLA. In our experiments, we use a time horizon of 150 steps and a reduced batch size of 64; the lookahead gradient step, $\alpha$, is set to 1 and the learning

rate is 0.3. Importantly, given the approximation used, the LOLA method was restricted to a single step of opponent learning. In contrast, using DiCE we can differentiate through an arbitrary number of opponent learning steps.

The original LOLA implemented via second order gradient corrections shows no stable learning, as it requires much larger batch sizes ($\sim 4000$). In contrast, LOLA-DiCE agents discover strategies of high social welfare, replicating the results of the original LOLA method in a way that is both more direct, efficient and establishes a common formulation between MAML and LOLA.

## 9.6 Related Work

Gradient estimation is well studied, although many methods have been named and explored independently in different fields, and the primary focus has been on first order gradients. Fu [168] provides an overview of methods from the point of view of simulation optimisation.

The score function (SF) estimator, also referred to as the likelihood ratio estimator or REINFORCE, has received considerable attention in many fields. In reinforcement learning, policy gradient methods [30] have proven highly successful, especially when combined with variance reduction techniques [36, 169]. The SF estimator has also been used in the analysis of stochastic systems [170], as well as for variational inference [171, 172].

Kingma and Welling [173] and Rezende, Mohamed, and Wierstra [174] discuss Monte-Carlo gradient estimates in the case where the stochastic parts of a model are amenable to reparameterisation.

To easily make use of these estimates for optimizing neural network models, automatic differentiation for backpropagation has been widely used [175].

These approaches are formalized for arbitrary computation graphs by Schulman et al. [31], but to our knowledge our paper is the first to present a practical and correct approach for generating higher order gradient estimators utilizing auto-diff.

One rapidly growing application area for such higher order gradient estimates is meta-learning for reinforcement learning. Finn, Abbeel, and Levine [159] compute

a loss after a number policy gradient learning steps, differentiating through the learning step to find parameters that can be quickly fine-tuned for different tasks. Li et al. [161] extend this work to also meta-learn the fine-tuning step direction and magnitude.

Al-Shedivat et al. [160] and Stadie et al. [167] derive the proper higher order gradient estimators for their work by reapplying the score function trick. We also presented an analytical derivation of the higher order gradient estimator in the previous section. No previous research presents a general strategy for constructing higher order gradient estimators for arbitrary graphs.

## 9.7 Conclusion & Future Work

We presented DiCE, a general method for computing any order gradient estimators for stochastic computation graphs. DiCE resolves the deficiencies of current approaches for computing higher order gradient estimators: analytical calculation is error-prone and incompatible with auto-diff, while repeated application of the Surrogate Loss approach is cumbersome and, as we show, leads to incorrect estimators in many cases. We prove the correctness of DiCE estimators, introduce a simple practical implementation of DiCE for use in deep learning frameworks, and validate its correctness and utility in a multi-agent reinforcement learning problem. We believe DiCE will unlock further exploration and adoption of higher order learning methods in meta-learning, reinforcement learning, and other applications of stochastic computation graphs. In the future we would like to extend DiCE to include action-dependent baselines and a general, higher-order, baseline generating function.

# 10
# Afterword

In this thesis we motivated, proposed and investigated a number of novel DMARL challenges that focus on different aspects of multi-agent learning. These include the learning of communication protocols, how agents can learn to collaborate, and how agents can learn to reciprocate with others. We also proposed and evaluated methods for the different challenges, clearly demonstrating progress on a number of different fronts: Chapter 6 is the first application of DMARL to the learning of communication protocols and one of the first applications of deep RL to any multi-agent setting overall. Chapter 7 is the first instance of DMARL agents learning communication protocols in a setting that was designed to be difficult for humans. Furthermore, we achieve this by allowing agents to reason over the beliefs of other agents, something that had not been demonstrated before. The work on StarCraft in Chapter 3, 4, and 5 demonstrates one of the first applications of DMARL to settings with complex game dynamics rather than to made-up toy-tasks, such as grid worlds. Chapter 8 is the first instance of a DMARL method that allows agents to directly consider and shape the learning behaviour of others and which discovers tit-for-tat in the iterated prisoner's dilemma.

While this is fantastic progress, important open challenges remain:

- Many real world settings are both partially observable, general-sum and involve opponents with unknown or partially known rewards and policies. In these settings accounting for the learning behaviour of other agents is a difficult and relevant challenge. Applications including online marketplaces and financial markets, where different participants interact aiming to maximise their own reward. However, there currently are no promising approaches that extend LOLA to these settings.

- The partially observable, general sum setting also relates to the question of *strategic communication*: Humans commonly communicate with others in settings of partial common interest, for example in order to formulate a compromise, form a coalition or to negotiate an agreement. Currently this is a vastly under-explored area in the context of agents that learn to communicate.

- While we show proof-of-principle results for agents that discover their own communication protocols, these protocols are extremely simplistic and lack the compositional, structural, and linguistic richness of human language. How to obtain rich emergent language is thus an important challenge.

- Modelling of other agents is a key requirement for broadening the application of MARL from settings where all agents are using known policies to settings that consist of unknown or partially known agents including humans. In principle we can use centralised training in order to generate a large number of diverse strategies and then train agents that can adapt to other agents strategies quickly using meta-learning and opponent modelling, rather than known models. Assuming enough diversity in the training pool, these fast adapting agents should then be able to perform ad-hoc teamplay with other agents, including humans. Potential areas of application include conversational interfaces or chat bots but also self-driving cars that can properly account for the beliefs, intentions and point of view of other agents in traffic.

- One of the key innovations that advanced machine learning over the last decade was the move to graphic-processing-units which allowed for great parallelism across a number different compute-cores. However, communication constraints between different computing processes present challenges to further parallelism. One of the appeals of multi-agent learning is that it might be able to provide a long term solution to the problem of parallelism: Each agent can in principle be instantiated on a different datacenter and carry out training in a decentralised form. These different agents could then use a learned protocol to exchange important high level information or coordinate whenever it is required.

All of these constitute formidable challenges with the potential to not only keep scientists engaged for the years and decades to come, but also to profoundly change the way we think about agency and, ultimately, consciousness.

# References


[1] John Ruggles. *Locomotive steam-engine for rail and other roads.* US Patent 1. July 1836.

[2] Jeremy Rifkin. *The end of work: The decline of the global labor force and the dawn of the post-market era.* ERIC, 1995.

[3] William M Siebert. "Frequency discrimination in the auditory system: Place or periodicity mechanisms?" In: *Proceedings of the IEEE* 58.5 (1970), pp. 723–730.

[4] Donald Waterman. "A guide to expert systems". In: (1986).

[5] Marti A. Hearst et al. "Support vector machines". In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.

[6] Carl Edward Rasmussen. "Gaussian processes in machine learning". In: *Advanced lectures on machine learning.* Springer, 2004, pp. 63–71.

[7] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[8] Geoffrey Hinton et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *IEEE Signal processing magazine* 29.6 (2012), pp. 82–97.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems.* 2012, pp. 1097–1105.

[10] Brendan Shillingford et al. "Large-scale visual speech recognition". In: *arXiv preprint arXiv:1807.05162* (2018).

[11] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems.* 2014, pp. 3104–3112.

[12] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3.1 (1988), pp. 9–44.

[13] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (2015), pp. 529–533.

[14] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529.7587 (2016), pp. 484–489.

[15] Robin IM Dunbar. "Neocortex size as a constraint on group size in primates". In: *Journal of human evolution* 22.6 (1992), pp. 469–493.

[16] Robert M Axelrod. *The evolution of cooperation: revised edition.* Basic books, 2006.

[17] Erik Zawadzki, Asher Lipson, and Kevin Leyton-Brown. "Empirically evaluating multiagent learning algorithms". In: *arXiv preprint arXiv:1401.8074* (2014).

[18] 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*. ACM. 2007, p. 255.

[19] Lloyd S Shapley. "Stochastic games". In: *Proceedings of the national academy of sciences* 39.10 (1953), pp. 1095–1100.

[20] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. "Optimal and Approximate Q-value Functions for Decentralized POMDPs". In: 32 (2008), pp. 289–353.

[21] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Vol. 1. 1. MIT press Cambridge, 1998.

[22] Yoav Shoham, Rob Powers, Trond Grenager, et al. "If multi-agent learning is the answer, what is the question?" In: *Artificial Intelligence* 171.7 (2007), pp. 365–377.

[23] John F Nash et al. "Equilibrium points in n-person games". In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.

[24] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[25] Ming Tan. "Multi-agent reinforcement learning: Independent vs. cooperative agents". In: *Proceedings of the tenth international conference on machine learning*. 1993, pp. 330–337.

[26] Ardi Tampuu et al. "Multiagent cooperation and competition with deep reinforcement learning". In: *arXiv preprint arXiv:1511.08779* (2015).

[27] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York: Cambridge University Press, 2009.

[28] Matthew Hausknecht and Peter Stone. "Deep recurrent q-learning for partially observable mdps". In: *arXiv preprint arXiv:1507.06527* (2015).

[29] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPS*. Vol. 99. 1999, pp. 1057–1063.

[30] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[31] John Schulman et al. "Gradient Estimation Using Stochastic Computation Graphs". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*. 2015, pp. 3528–3536.

[32] Hajime Kimura, Shigenobu Kobayashi, et al. "An analysis of actor-critic algorithms using eligibility traces: reinforcement learning with imperfect value functions". In: *Journal of Japanese Society for Artificial Intelligence* 15.2 (2000), pp. 267–275.

[33] John Schulman et al. "High-Dimensional Continuous Control Using Generalized Advantage Estimation". In: *CoRR* abs/1506.02438 (2015). URL: http://arxiv.org/abs/1506.02438.

[34] Ziyu Wang et al. "Sample Efficient Actor-Critic with Experience Replay". In: *arXiv preprint arXiv:1611.01224* (2016).

[35] Roland Hafner and Martin Riedmiller. "Reinforcement learning in feedback control". In: *Machine learning* 84.1 (2011), pp. 137–169.

[36] Lex Weaver and Nigel Tao. "The optimal reward baseline for gradient-based reinforcement learning". In: *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2001, pp. 538–545.

[37] Vijay R Konda and John N Tsitsiklis. "Actor-Critic Algorithms." In: *NIPS*. Vol. 13. 2000, pp. 1008–1014.

[38] Kyunghyun Cho et al. "On the properties of neural machine translation: Encoder-decoder approaches". In: *arXiv preprint arXiv:1409.1259* (2014).

[39] Yu-Han Chang, Tracey Ho, and Leslie Pack Kaelbling. "All learning is Local: Multi-agent Learning in Global Reward Games." In: *NIPS*. 2003, pp. 807–814.

[40] Nicolas Usunier et al. "Episodic Exploration for Deep Deterministic Policies: An Application to StarCraft Micromanagement Tasks". In: *arXiv preprint arXiv:1609.02993* (2016).

[41] Peng Peng et al. "Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games". In: *arXiv preprint arXiv:1703.10069* (2017).

[42] Lucian Busoniu, Robert Babuska, and Bart De Schutter. "A comprehensive survey of multiagent reinforcement learning". In: *IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews* 38.2 (2008), p. 156.

[43] Erfu Yang and Dongbing Gu. *Multiagent reinforcement learning for multi-robot systems: A survey*. Tech. rep. tech. rep, 2004.

[44] Joel Z Leibo et al. "Multi-agent Reinforcement Learning in Sequential Social Dilemmas". In: *arXiv preprint arXiv:1702.03037* (2017).

[45] Abhishek Das et al. "Learning Cooperative Visual Dialog Agents with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1703.06585* (2017).

[46] Igor Mordatch and Pieter Abbeel. "Emergence of Grounded Compositional Language in Multi-Agent Populations". In: *arXiv preprint arXiv:1703.04908* (2017).

[47] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. "Multi-agent cooperation and the emergence of (natural) language". In: *arXiv preprint arXiv:1612.07182* (2016).

[48] Sainbayar Sukhbaatar, Rob Fergus, et al. "Learning multiagent communication with backpropagation". In: *Advances in Neural Information Processing Systems*. 2016, pp. 2244–2252.

[49] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. "Cooperative Multi-Agent Control Using Deep Reinforcement Learning". In: (2017).

[50] Shayegan Omidshafiei et al. "Deep Decentralized Multi-task Multi-Agent RL under Partial Observability". In: *arXiv preprint arXiv:1703.06182* (2017).

[51] Tabish Rashid et al. “QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning”. In: *Proceedings of The 35th International Conference on Machine Learning*. 2018.

[52] Peter Sunehag et al. “Value-Decomposition Networks For Cooperative Multi-Agent Learning”. In: *arXiv preprint arXiv:1706.05296* (2017).

[53] Ryan Lowe et al. “Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments”. In: *arXiv preprint arXiv:1706.02275* (2017).

[54] Danny Weyns, Alexander Helleboogh, and Tom Holvoet. “The packet-world: A test bed for investigating situated multi-agent systems”. In: *Software Agent-Based Applications, Platforms and Development Kits*. Springer, 2005, pp. 383–408.

[55] David H Wolpert and Kagan Tumer. “Optimal payoff functions for members of collectives”. In: *Modeling complexity in economic and social systems*. World Scientific, 2002, pp. 355–369.

[56] Scott Proper and Kagan Tumer. “Modeling difference rewards for multiagent learning”. In: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems-Volume 3*. International Foundation for Autonomous Agents and Multiagent Systems. 2012, pp. 1397–1398.

[57] Mitchell K Colby, William Curran, and Kagan Tumer. “Approximating difference evaluations with local information”. In: *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2015, pp. 1659–1660.

[58] Gabriel Synnaeve et al. “TorchCraft: a Library for Machine Learning Research on Real-Time Strategy Games”. In: *arXiv preprint arXiv:1611.00625* (2016).

[59] R. Collobert, K. Kavukcuoglu, and C. Farabet. “Torch7: A Matlab-like Environment for Machine Learning”. In: *BigLearn, NIPS Workshop*. 2011.

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

[61] Emilio Jorge, Mikael Kageback, and Emil Gustavsson. “Learning to Play Guess Who? and Inventing a Grounded Language as a Consequence”. In: *arXiv preprint arXiv:1611.03218* (2016).

[62] Martin J Osborne and Ariel Rubinstein. *A course in game theory*. MIT press, 1994.

[63] Katie Genter, Tim Laue, and Peter Stone. “Three years of the RoboCup standard platform league drop-in player competition”. In: *Autonomous Agents and Multi-Agent Systems* 31.4 (2017), pp. 790–820.

[64] Carlos Guestrin, Daphne Koller, and Ronald Parr. “Multiagent planning with factored MDPs”. In: *Advances in neural information processing systems*. 2002, pp. 1523–1530.

[65] Jelle R Kok and Nikos Vlassis. “Sparse cooperative Q-learning”. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 61.

[66] Katie Genter, Noa Agmon, and Peter Stone. “Ad hoc teamwork for leading a flock”. In: *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2013, pp. 531–538.

[67] Samuel Barrett, Peter Stone, and Sarit Kraus. "Empirical evaluation of ad hoc teamwork in the pursuit domain". In: *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems. 2011, pp. 567–574.

[68] Stefano V Albrecht and Peter Stone. "Reasoning about hypothetical agent behaviours and their parameters". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 547–555.

[69] Alessandro Panella and Piotr Gmytrasiewicz. "Interactive POMDPs with finite-state models of other agents". In: *Autonomous Agents and Multi-Agent Systems* 31.4 (2017), pp. 861–904.

[70] Takaki Makino and Kazuyuki Aihara. "Multi-agent reinforcement learning algorithm to handle beliefs of other agents' policies and embedded beliefs". In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM. 2006, pp. 789–791.

[71] Kyle A Thomas et al. "The psychology of coordination and common knowledge." In: *Journal of personality and social psychology* 107.4 (2014), p. 657.

[72] Ariel Rubinstein. "The Electronic Mail Game: Strategic Behavior Under" Almost Common Knowledge"". In: *The American Economic Review* (1989), pp. 385–391.

[73] Gizem Korkmaz et al. "Collective action through common knowledge using a facebook model". In: *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 253–260.

[74] Ronen I. Brafman and Moshe Tennenholtz. "Learning to Coordinate Efficiently: A Model-based Approach". In: *Journal of Artificial Intelligence Research*. Vol. 19. 2003, pp. 11–23.

[75] Robert J Aumann et al. "Subjectivity and correlation in randomized strategies". In: *Journal of mathematical Economics* 1.1 (1974), pp. 67–96.

[76] Ludek Cigler and Boi Faltings. "Decentralized anti-coordination through multi-agent learning". In: *Journal of Artificial Intelligence Research* 47 (2013), pp. 441–473.

[77] Craig Boutilier. "Sequential optimality and coordination in multiagent systems". In: *IJCAI*. Vol. 99. 1999, pp. 478–485.

[78] 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*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 1273–1280.

[79] Miao Liu et al. "Learning for Multi-robot Cooperation in Partially Observable Stochastic Environments with Macro-actions". In: *arXiv preprint arXiv:1707.07399* (2017).

[80] Rajbala Makar, Sridhar Mahadevan, and Mohammad Ghavamzadeh. "Hierarchical multi-agent reinforcement learning". In: *Proceedings of the fifth international conference on Autonomous agents*. ACM. 2001, pp. 246–253.

[81] Thomas G. Dietterich. “Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition”. In: *J. Artif. Int. Res.* 13.1 (Nov. 2000), pp. 227–303.

[82] Saurabh Kumar et al. “Federated Control with Hierarchical Multi-Agent Deep Reinforcement Learning”. In: *arXiv preprint arXiv:1712.08266* (2017).

[83] Laetitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. “Independent reinforcement learners in cooperative Markov games: a survey regarding coordination problems”. In: *The Knowledge Engineering Review* 27.01 (2012), pp. 1–31.

[84] Kamil Ciosek and Shimon Whiteson. “OFFER: Off-Environment Reinforcement Learning”. In: (2017).

[85] Gerald Tesauro. “Extending Q-Learning to General Adaptive Multi-Agent Systems.” In: *NIPS*. Vol. 4. 2003.

[86] Tom Schaul et al. “Prioritized Experience Replay”. In: *CoRR* abs/1511.05952 (2015).

[87] Vincent Conitzer and Tuomas Sandholm. “AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents”. In: *Machine Learning* 67.1-2 (2007), pp. 23–43.

[88] Bruno C Da Silva et al. “Dealing with non-stationary environments using context detection”. In: *Proceedings of the 23rd international conference on Machine learning*. ACM. 2006, pp. 217–224.

[89] Jelle R Kok and Nikos Vlassis. “Collaborative multiagent reinforcement learning by payoff propagation”. In: *Journal of Machine Learning Research* 7.Sep (2006), pp. 1789–1828.

[90] 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.

[91] Maja J Mataric. “Using communication to reduce locality in distributed multiagent learning”. In: *Journal of experimental & theoretical artificial intelligence* 10.3 (1998), pp. 357–369.

[92] CP Robert and G Casella. “Monte Carlo Statistical Methods Springer”. In: *New York* (2004).

[93] F. S. Melo, M. Spaan, and S. J. Witwicki. “QueryPOMDP: POMDP-based communication in multiagent systems”. In: *Multi-Agent Systems*. 2011, pp. 189–204.

[94] L. Panait and S. Luke. “Cooperative multi-agent learning: The state of the art”. In: *Autonomous Agents and Multi-Agent Systems* 11.3 (2005), pp. 387–434.

[95] C. Zhang and V. Lesser. “Coordinating multi-agent reinforcement learning with limited communication”. In: vol. 2. 2013, pp. 1101–1108.

[96] T. Kasai, H. Tenmoto, and A. Kamiya. “Learning of communication codes in multi-agent reinforcement learning problem”. In: *IEEE Soft Computing in Industrial Applications*. 2008, pp. 1–6.

[97] C. L. Giles and K. C. Jim. "Learning communication for multi-agent systems". In: *Innovative Concepts for Agent-Based Systems*. Springer, 2002, pp. 377–390.

[98] Karol Gregor et al. "DRAW: A recurrent neural network for image generation". In: *arXiv preprint arXiv:1502.04623* (2015).

[99] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training deep neural networks with weights and activations constrained to +1 or -1". In: *arXiv preprint arXiv:1602.02830* (2016).

[100] Geoffrey Hinton and Ruslan Salakhutdinov. "Discovering binary codes for documents by learning deep generative models". In: *Topics in Cognitive Science* 3.1 (2011), pp. 74–91.

[101] Karthik Narasimhan, Tejas Kulkarni, and Regina Barzilay. "Language understanding for text-based games using deep reinforcement learning". In: *arXiv preprint arXiv:1506.08941* (2015).

[102] Sepp Hochreiter and Jurgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[103] Junyoung Chung et al. "Empirical evaluation of gated recurrent neural networks on sequence modeling". In: *arXiv preprint arXiv:1412.3555* (2014).

[104] Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. "An empirical exploration of recurrent network architectures". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*. 2015, pp. 2342–2350.

[105] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *arXiv preprint arXiv:1502.03167* (2015).

[106] W. Wu. *100 prisoners and a lightbulb*. Tech. rep. OCF, UC Berkeley, 2002.

[107] Michael Studdert-Kennedy. "How Did Language go Discrete?" In: *Language Origins: Perspectives on Evolution*. Ed. by Maggie Tallerman. Oxford University Press, 2005. Chap. 3.

[108] H. P. Grice. "Logic and Conversation". In: *Syntax and Semantics: Vol. 3: Speech Acts*. Ed. by Peter Cole and Jerry L. Morgan. New York: Academic Press, 1975, pp. 41–58. URL: http://www.ucl.ac.uk/ls/studypacks/Grice-Logic.pdf.

[109] Michael C. Frank and Noah D. Goodman. "Predicting pragmatic reasoning in language games". In: *Science (80-. ).* 336.6084 (2012), p. 998. arXiv: 0602092 [physics].

[110] Ashutosh Nayyar, Aditya Mahajan, and Demosthenis Teneketzis. "Decentralized stochastic control with partial history sharing: A common information approach". In: *IEEE Trans. Automat. Contr.* 58.7 (2013), pp. 1644–1658. arXiv: 1209.1695. URL: https://arxiv.org/abs/1209.1695.

[111] Chris L. Baker et al. "Rational quantitative attribution of beliefs, desires and percepts in human mentalizing". In: *Nat. Hum. Behav.* 1.4 (2017), pp. 1–10. URL: http://dx.doi.org/10.1038/s41562-017-0064.

[112] L P Kaelbling, M L Littman, and A R Cassandra. "Planning and acting in partially observable stochastic domains". In: *Artif. Intell.* 101.1-2 (1998), pp. 99–134. URL: http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

[113] Piotr J. Gmytrasiewicz and Prashant Doshi. "A framework for sequential planning in multi-agent settings". In: *J. Artif. Intell. Res.* 24 (2005), pp. 49–79. arXiv: 1109.2135.

[114] Thomas P Minka. "Expectation Propagation for Approximate Bayesian Inference". In: *Uncertain. Artif. Intell.* 17.2 (2001), pp. 362–369. arXiv: 1301.2294. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.86.1319%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[115] Lasse Espeholt et al. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *arXiv:1802.01561* (2018). arXiv: 1802.01561. URL: http://arxiv.org/abs/1802.01561.

[116] Max Jaderberg et al. "Human-level performance in first-person multiplayer games with population-based deep reinforcement learning". In: *arXiv:1807.01281* (2018). arXiv: 1807.01281. URL: http://arxiv.org/abs/1807.01281.

[117] Max Jaderberg et al. "Population Based Training of Neural Networks". In: *arXiv:1711.09846* (2017). arXiv: 1711.09846. URL: http://arxiv.org/abs/1711.09846.

[118] Jakob N Foerster et al. "Learning to communicate to solve riddles with deep distributed recurrent q-networks". In: *arXiv preprint arXiv:1602.02672* (2016).

[119] Jean-Francois Baffier et al. "Hanabi is NP-complete, even for cheaters who look at their cards". In: (2016).

[120] Christopher Cox et al. "How to Make the Perfect Fireworks Display : Two Strategies for Hanabi". In: *Math. Mag.* 88 (2015), p. 323. URL: http://www.jstor.org/stable/10.4169/math.mag.88.5.323.

[121] Bruno Bouzy. "Playing Hanabi Near-Optimally". In: *Advances in Computer Games*. Springer. 2017, pp. 51–62.

[122] Joseph Walton-Rivers et al. "Evaluating and modelling Hanabi-playing agents". In: *Evolutionary Computation (CEC), 2017 IEEE Congress on*. IEEE. 2017, pp. 1382–1389.

[123] Hirotaka Osawa. "Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information." In: *AAAI workshop: Computer Poker and Imperfect Information*. 2015, pp. 37–43.

[124] Markus Eger, Chris Martens, and Marcela Alfaro Cordoba. "An intentional AI for hanabi". In: *2017 IEEE Conf. Comput. Intell. Games, CIG 2017* (2017), pp. 68–75.

[125] Matej Moravcik et al. "DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker". In: *arXiv preprint arXiv:1701.01724* (2017).

[126] Noam Brown and Tuomas Sandholm. "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals". In: *Science* 359.6374 (2018), pp. 418–424.

[127] Pablo Hernandez-Leal et al. "A Survey of Learning in Multiagent Environments: Dealing with Non-Stationarity". In: *arXiv preprint arXiv:1707.09183* (2017).

[128] Tuomas W Sandholm and Robert H Crites. "Multiagent reinforcement learning in the iterated prisoner's dilemma". In: *Biosystems* 37.1-2 (1996), pp. 147–166.

[129] Michael Bowling and Manuela Veloso. "Multiagent learning using a variable learning rate". In: *Artificial Intelligence* 136.2 (2002), pp. 215–250.

[130] William Uther and Manuela Veloso. *Adversarial reinforcement learning*. Tech. rep. Technical report, Carnegie Mellon University, 1997. Unpublished, 1997.

[131] C. Claus and C. Boutilier. "The Dynamics of Reinforcement Learning Cooperative Multiagent Systems". In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. June 1998, pp. 746–752.

[132] Michael Wunder, Michael L Littman, and Monica Babes. "Classes of multiagent q-learning dynamics with epsilon-greedy exploration". In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 1167–1174.

[133] Martin Zinkevich, Amy Greenwald, and Michael L Littman. "Cyclic equilibria in Markov games". In: *Advances in Neural Information Processing Systems*. 2006, pp. 1641–1648.

[134] Michael L Littman. "Friend-or-foe Q-learning in general-sum games". In: *ICML*. Vol. 1. 2001, pp. 322–328.

[135] Doran Chakraborty and Peter Stone. "Multiagent learning in the presence of memory-bounded agents". In: *Autonomous agents and multi-agent systems* 28.2 (2014), pp. 182–213.

[136] Ronen I. Brafman and Moshe Tennenholtz. "Efficient Learning Equilibrium". In: *Advances in Neural Information Processing Systems*. Vol. 9. 2003, pp. 1635–1643.

[137] Marc Lanctot et al. "A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning". In: *Advances in Neural Information Processing Systems (NIPS)*. 2017.

[138] Johannes Heinrich and David Silver. "Deep reinforcement learning from self-play in imperfect-information games". In: *arXiv preprint arXiv:1603.01121* (2016).

[139] Adam Lerer and Alexander Peysakhovich. "Maintaining cooperation in complex social dilemmas using deep reinforcement learning". In: *arXiv preprint arXiv:1707.01068* (2017).

[140] Enrique Munoz de Cote and Michael L. Littman. "A Polynomial-time Nash Equilibrium Algorithm for Repeated Stochastic Games". In: *24th Conference on Uncertainty in Artificial Intelligence (UAI'08)*. 2008. URL: `http://uai2008.cs.helsinki.fi/UAI_camera_ready/munoz.pdf`.

[141] Jacob W Crandall and Michael A Goodrich. "Learning to compete, coordinate, and cooperate in repeated games using reinforcement learning". In: *Machine Learning* 82.3 (2011), pp. 281–314.

[142] George W Brown. "Iterative solution of games by fictitious play". In: (1951).

[143] Richard Mealing and Jonathan Shapiro. "Opponent Modelling by Expectation-Maximisation and Sequence Prediction in Simplified Poker". In: *IEEE Transactions on Computational Intelligence and AI in Games* (2015).

[144] Neil C Rabinowitz et al. "Machine Theory of Mind". In: *arXiv preprint arXiv:1802.07740* (2018).

[145] Richard Mealing and Jonathan L Shapiro. “Opponent Modelling by Sequence Prediction and Lookahead in Two-Player Games.” In: *ICAISC (2)*. 2013, pp. 385–396.

[146] Pablo Hernandez-Leal and Michael Kaisers. “Learning against sequential opponents in repeated stochastic games”. In: (2017).

[147] Chongjie Zhang and Victor R Lesser. “Multi-Agent Learning with Policy Prediction.” In: *AAAI*. 2010.

[148] Luke Metz et al. “Unrolled generative adversarial networks”. In: *arXiv preprint arXiv:1611.02163* (2016).

[149] Max Kleiman-Weiner et al. “Coordinate to cooperate or compete: abstract goals and joint intentions in social interaction”. In: *COGSCI*. 2016.

[150] Stephane Ross, Geoffrey J Gordon, and J Andrew Bagnell. “No-regret reductions for imitation learning and structured prediction”. In: *In AISTATS*. Citeseer. 2011.

[151] Mariusz Bojarski et al. “End to end learning for self-driving cars”. In: *arXiv preprint arXiv:1604.07316* (2016).

[152] R Duncan Luce and Howard Raiffa. “Games and Decisions: Introduction and Critical Survey”. In: (1957).

[153] King Lee and K Louis. “The Application of Decision Theory and Dynamic Programming to Adaptive Control Systems”. PhD thesis. 1967.

[154] Drew Fudenberg and Jean Tirole. “Game theory, 1991”. In: *Cambridge, Massachusetts* 393 (1991), p. 12.

[155] B Myerson Roger. *Game theory: analysis of conflict*. 1991.

[156] Robert Gibbons. *Game theory for applied economists*. Princeton University Press, 1992.

[157] William H Press and Freeman J Dyson. “Iterated Prisoner’s Dilemma contains strategies that dominate any evolutionary opponent”. In: *Proceedings of the National Academy of Sciences* 109.26 (2012), pp. 10409–10413.

[158] John E Dennis Jr and Jorge J More. “Quasi-Newton methods, motivation and theory”. In: *SIAM review* 19.1 (1977), pp. 46–89.

[159] Chelsea Finn, Pieter Abbeel, and Sergey Levine. “Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks”. In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*. 2017, pp. 1126–1135.

[160] Maruan Al-Shedivat et al. “Continuous Adaptation via Meta-Learning in Nonstationary and Competitive Environments”. In: *CoRR* abs/1710.03641 (2017). arXiv: 1710.03641.

[161] Zhenguo Li et al. “Meta-SGD: Learning to Learn Quickly for Few Shot Learning”. In: *CoRR* abs/1707.09835 (2017). arXiv: 1707.09835.

[162] Martin Abadi et al. “TensorFlow: A System for Large-Scale Machine Learning”. In: *12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016, Savannah, GA, USA, November 2-4, 2016*. 2016, pp. 265–283.

[163] Adam Paszke et al. “Automatic differentiation in PyTorch”. In: (2017).

[164] John Schulman, Pieter Abbeel, and Xi Chen. "Equivalence Between Policy Gradients and Soft Q-Learning". In: *CoRR* abs/1704.06440 (2017). arXiv: 1704.06440.

[165] John Schulman et al. "Trust region policy optimization". In: *International Conference on Machine Learning*. 2015, pp. 1889–1897.

[166] Barak A Pearlmutter. "Fast exact multiplication by the Hessian". In: *Neural computation* 6.1 (1994), pp. 147–160.

[167] Bradly Stadie et al. *Some Considerations on Learning to Explore via Meta-Reinforcement Learning*. 2018. URL: https://openreview.net/forum?id=Skk3Jm96W.

[168] Michael C Fu. "Gradient estimation". In: *Handbooks in operations research and management science* 13 (2006), pp. 575–616.

[169] Ivo Grondman et al. "A survey of actor-critic reinforcement learning: Standard and natural policy gradients". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307.

[170] Peter W Glynn. "Likelihood ratio gradient estimation for stochastic systems". In: *Communications of the ACM* 33.10 (1990), pp. 75–84.

[171] David Wingate and Theophane Weber. "Automated Variational Inference in Probabilistic Programming". In: *CoRR* abs/1301.1299 (2013). arXiv: 1301.1299.

[172] Rajesh Ranganath, Sean Gerrish, and David M. Blei. "Black Box Variational Inference". In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*. 2014, pp. 814–822.

[173] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114 (2013). arXiv: 1312.6114.

[174] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: (2014), pp. 1278–1286.

[175] Atilim Gunes Baydin, Barak A. Pearlmutter, and Alexey Andreyevich Radul. "Automatic differentiation in machine learning: a survey". In: *CoRR* abs/1502.05767 (2015). arXiv: 1502.05767.