Improving Cooperation in Multi-agent Deep Reinforcement Learning for Mobile Robotics

Maxime Toquebiau

under the supervision of Faïz Ben Amar, Nicolas Bredeche, and Jae Yun Jun Kim

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

at
Institut des Systèmes Intelligents et de Robotique (ISIR), Sorbonne Université & ECE Paris

École doctorale Sciences mécaniques, acoustique, électronique et robotique de Paris (SMAER) - ED 391

Katrien Beuls	University of Namur	Rapportrice
Clément Moulin-Frier	INRIA	Rapporteur
Sylvain Chevalier	Univ. Paris-Saclay, CNRS	Examinateur
Aurélie Beynier	LIP6, Sorbonne Université	Examinatrice
Olivier Simonin	INSA Lyon, INRIA	Examinateur
Faïz Ben Amar	ISIR, Sorbonne Université	Co-directeur
Nicolas Bredeche	ISIR, Sorbonne Université	Co-directeur
Jae Yun Jun Kim	École Centrale d'Électronique (ECE)	Co-encadrant

Abstract

Improving Cooperation in Multi-agent Deep Reinforcement Learning for Mobile Robotics

Multi-agent deep reinforcement learning (MADRL) algorithms propose ways of learning multi-agent behaviour in a variety of different settings. Despite significant advancements in the past decade, it still faces important challenges for learning skills required in robotic settings. This work examines the current state of MADRL, identifies some of its shortcomings, and proposes novel methods to improve the existing approaches. Specifically, the thesis focuses on two critical challenges: exploration and communication. For exploration, traditional approaches may struggle to discover optimal joint behaviours due to inefficient local exploration strategies. We introduce a method based on intrinsic motivation that encourages agents to explore new coordinated actions, thereby improving the convergence to optimal joint strategies. For tackling communication, while differentiable emergent communication is a popular approach, we argue that it may not fully meet the needs of artificial agents, especially for working towards robotic applications. We advocate for language-augmented agents to be deployed in multi-agent environments. We propose a training method for these language-augmented agents and present experiments demonstrating their effectiveness. Teaching agents a predefined language offers multiple benefits: efficient communication, grounded representation learning, enhanced generalization, and improved interaction with humans.

Résumé

Améliorer les Stratégies de Coopération dans l'Apprentissage par Renforcement Profond Multi-Agent en Robotique Mobile

L'apprentissage par renforcement profond permet l'apprentissage de stratégies multi-agents dans divers contextes. Malgré des avancées significatives au cours de la dernière décennie, il rencontre encore d'importants défis pour l'apprentissage des compétences nécessaires dans les contextes robotiques. Ce travail examine l'état actuel du MADRL, identifie certaines de ses lacunes et propose des méthodes innovantes pour améliorer les approches existantes. La thèse se concentre sur deux défis critiques : l'exploration et la communication. Les approches d'apprentissage multi-agent reconnues ont des difficultés à découvrir des comportements conjoints optimaux en raison de stratégies d'exploration locales inefficaces. Pour résoudre ce problème, nous introduisons une méthode basée sur la motivation intrinsèque qui encourage les agents à explorer de nouvelles actions coordonnées, améliorant ainsi la convergence vers des stratégies conjointes optimales. Dans le cas de l'apprentissage de la communication, bien que la communication émergente différentiable soit une approche populaire, nous soutenons qu'elle pourrait ne pas répondre pleinement aux besoins des agents artificiels, notamment pour les applications robotiques. À la place, nous proposons d'enseigner une langue prédéfinie aux agents. Nous proposons un algorithme d'apprentissage pour enseigner un langage à des agents et présentons des expériences démontrant l'efficacité de cette approche. Ceci permet d'acquérir de multiples avantages : communication efficace, apprentissage de représentation ancré dans le langage, généralisation améliorée et interaction plus facile avec les êtres humains.

Acknowledgements

Working on this thesis has been the most challenging, exciting, and enriching experience of my life. This could not have been achieved without the help of several people.

First of all, my dear supervisors. Thank you for your guidance and implication in my research.

My parents for their abundant support. Thank you for everything.

The friends made along the way, both at ECE and ISIR. Thank you for your support, wise insights, and great moments spent together.

Contents

List of Abbreviations

ANN	**A**rtificial **N**eural **N**etwork	2.4.1
CLIP	**C**ontrastive **L**anguage-**I**mage **P**re-training	5.3.3
CNN	**C**onvolutional **N**eural **N**etwork	2.4.3
CTDE	**C**entralised **T**raining with **D**ecentralised **E**xecution	3.4.1
CTE	**C**entralised **T**raining and **E**xecution	3.4.1
(D)DPG	(**D**eep) **D**eterministic **P**olicy **G**radient	2.3.2, 2.5.2
Dec-POMDP	**Dec**entralised **P**artially-**O**bservable **M**arkov **D**ecision **P**rocess	3.2.1
DQN	**D**eep **Q**-**N**etwork	2.5.1
DTE	**D**ecentralised **T**raining and **E**xecution	3.4.1
E3B	**E**xploration via **E**lliptical **E**pisodic **B**onuses	4.4.3
EEC	**E**pisodic **E**xploration **C**riterion	4.5.1
GRU	**G**ated **R**ecurrent **U**nit	2.4.2
HRI	**H**uman-**R**obot **I**nteraction	3.5.4
IGM	**I**ndividual-**G**lobal-**M**ax	3.4.4
JIM	**J**oint **I**ntrinsic **M**otivation	4
KL	**K**ullback-**L**eibler	2.5.2
LIM	**L**ocal **I**ntrinsic **M**otivation	4.6
LLEC	**L**ife-**L**ong **E**xploration **C**riterion	4.5.1
LLM	**L**arge **L**anguage **M**odel	5.2.3
MADDPG	**M**ulti-**A**gent **D**eep **D**eterministic **P**olicy **G**radient	3.4.3
MADRL	**M**ulti-**A**gent **D**eep **R**einforcement **L**earning	3
MAPPO	**M**ulti-**A**gent **P**roximal **P**olicy **O**ptimisation	3.4.3
MAS	**M**ulti-**A**gent **S**ystem	3
MDP	**M**arkov **D**ecision **P**rocess	2.2.1
MLP	**M**ulti-**L**ayer **P**erceptron	2.4.3
MPE	**M**ulti-agent **P**article **E**nvrionment	3.5.1
MSBE	**M**ean **S**quared **B**ellman **E**rror	2.5.1
NovelD	**Novel**ty **D**iversity	4.4.2
POMDP	**P**artially-**O**bservable **M**arkov **D**ecision **P**rocess	3.2.1
PPO	**P**roximal **P**olicy **O**ptimisation	2.5.2
RL	**R**einforcement **L**earning	2.1
RLHF	**R**einforcement **L**earning from **H**uman **F**eedback	2.1
RNN	**R**ecurrent **N**eural **N**etwork	2.4.2
RND	**R**andom **N**etwork **D**istillation	4.4.1
SAC	**S**oft **A**ctor-**C**ritic	2.5.2
SMAC	**S**tarcraft **M**ulti-**A**gent **C**hallenge	3.5.1
TD	**T**emporal **D**ifference	2.3.1
TD3	**T**win **D**elayed **N**etwork **D**eep **D**eterministic **P**olicy **G**radient	2.5.2
TRPO	**T**rust **R**egion **P**olicy **O**ptimisation	2.5.2
VDN	**V**alue **D**ecomposition **N**etwork	3.4.4

Chapter 1

Introduction

Intelligence in a Group

One important component of intelligence resides in the ability to behave in a group. As an individual in a group, one must be able to observe others and react to incoming information to fulfil their own objective. This may require cooperating with others or entering into conflicts. In one case or the other, the individual must be able to align their behaviour with their known capacities, personal intentions, and expected capacities and intentions of others. Learning in a group also requires specific abilities. An individual can acquire knowledge about their environment by observing it and can learn to adapt their behaviour from the results of their actions. But, it can also observe others and internalise their experiences to modify its own behaviour. However, in both cases, effective learning requires being able to assert what elements caused the observed results, and how to modify these elements to obtain different results. Specifically, in a group, any outcome can be produced by a combination of individual behaviours. Thus, learning in a group requires identifying both the impact of individual behaviours and how to combine them to achieve certain desired outcomes.

These challenges are faced by living beings. First, because being alone is often not an option. Most ecological settings include a large variety of different living beings, with both unrelated or interconnected objectives, and with different kinds of intelligence. Being able to cope with this diversity is a matter of survival. Second, because living in a group often comes with great advantages. By composing the individual's capabilities, a group can gain strength, resilience, longevity, and many other advantages that are profitable to the concerned individuals. This is common in nature, where most living beings are organised in populations of individuals sharing common biological traits, as a product of evolution. This can also take the form of social relations, with groups of individuals relying on each other to fulfil the needs of the group together.

In an attempt to model intelligence with computational tools, and without a clear definition of intelligence, nor a recipe for obtaining it, artificial intelligence research has studied the many known or presumed components of intelligence. Thus, this is no surprise that the study of groups of artificial entities is an important matter in the field. Previously mentioned aspects of individuals in groups can all be applied to artificial beings that live in environments populated with other beings. So-called multi-agent systems have been developed to describe these interactions between multiple agents – i.e., beings, artificial or natural, that live in a given environment –, and propose methods for providing them intelligent ways of behaving and learning.

Groups of Artificial Intelligent Agents

While the learning part was not always preeminently considered for building artificial intelligence, it has now been widely accepted as central to the problem. Enabling artificial learning provides a mechanism for artificial agents to acquire knowledge by themselves. Without it, building artificial intelligence would require transmitting the required knowledge and reasoning capacities – again, not clearly defined – from humans to the lucky recipients, through any way of coding this information. Thus, for the past few decades, the field has largely focused on developing various ways of enabling computational models to learn from human-generated data or from their own experience. The latter case is the one we will concentrate on throughout this thesis.

The idea of enabling artificial agents to learn by themselves through trial-and-error is a longstanding ambition of artificial intelligence research. First because, conceptually, reproducing the way living beings learn seems both logical and extremely appealing. Second because, more practically, supervised training of intelligent agents requires time, effort – these two, we will put anyway –, and, crucially, a lot of data describing all aspects of intelligence we are expecting. Instead, a more preferable avenue could be to define the tools that allow the emergence of intelligence and let agents learn from their experience. This is the general idea behind **reinforcement learning**, that mathematically formulates the trial-and-error problem. With reinforcement learning, agents are placed in a learning environment where they can freely experiment different series of actions. Resulting of the explored behaviours, they will receive rewards, either positive or negative, from which they will learn by modifying their behaviour. Actions that lead to larger returns will be performed more often, and conversely. The rewards define the objectives given to the agents, with positive rewards being assigned to actions that work towards the given objective. The study of reinforcement learning comprises the definition of mathematical tools allowing this process and the design of learning algorithms that use them.

To investigate how intelligence can emerge in multi-agent systems, reinforcement learning tools can be adapted to allow multiple agents to interact and learn together. This implicates a series of important challenges, all resulting from having multiple agents learning together in an environment. Agents must now handle more dynamic environments, where other agents live by their side, and observed outcomes may be independent of one's actions. But, this also provides the benefits to the group. More complex tasks can be envisioned, with simple agent combining their capacities to achieve greater results. Realistic settings where various types of entities live together can be described more precisely to try learning better agents.

Thesis Subject: Improving Multi-Agent Learning for Robotic Settings

In the last decade, deep learning techniques have been introduced in the fields of single-agent and multi-agent reinforcement learning. They have greatly increased the potential of reinforcement learning for tackling more realistic environments. **Multi-agent deep reinforcement learning** approaches have been proposed to handle the numerous challenges of multi-agent learning. But, there are still many issues remaining to solve for building efficient agents capable of interacting with other agents and with human beings.

In this thesis, we take the perspective of robotics by looking at how multi-agent learning approaches can work towards robotic applications. While robotics has always been an important drive of artificial intelligence and reinforcement learning research,

there are still many challenges to overcome for learning robotic behaviour with multi-agent deep reinforcement learning. Therefore, we will explore the implications of building agents for the robotic setting, asking

What constraints should be put on learning?

and

What skills should be sought in our agents?

These questions have many important implications on how and agents and learning algorithms should be designed. For now, research on multi-agent deep reinforcement learning has mainly avoided these issues to focus on tackling issues of multi-agent learning and building algorithms that perform better on the existing benchmarks. While this is justified by the many problems brought by learning in a multi-agent setting, this also limits the relevance and potential of current research. Taking requirements of the robotic domain into consideration will certainly bring additional issues, but may also help focus research on the right problems.

Following this idea, our objective is to provide an analysis of the current state of research on multi-agent deep reinforcement learning algorithms and provide ways for improving them. We will focus on the cooperative setting that corresponds to most robotic applications in our daily lives, identify key challenges of these settings and possible approaches for tackling them. Consequently, we will propose two contributions for improving cooperative multi-agent learning algorithms, specifically targeting the problems of exploration and communication.

Reading Guide

The following manuscript will be organised into four main chapters. **Chapter** 2 presents the reinforcement learning problem and its proposed approaches. This problem and the learning tools that result from it are at the core of this work: they define the challenges we face and the solutions we can employ. Therefore, it is crucial to introduce them formally before any other matter. We start by presenting the field of reinforcement learning research, with its foundational tools and algorithms. Next, we introduce the techniques of deep learning that will be employed extensively throughout this thesis. Lastly, we present a short review of important deep reinforcement learning algorithms, focusing on the kind of approaches used in our works.

In **Chapter** 3, we present an overview of the field of multi-agent deep reinforcement learning research, from the perspective of robotics. We formally introduce the tools used in the multi-agent reinforcement learning algorithms. Then, we introduce the challenges faced in both multi-agent learning and robotic environments, looking at how they are often closely related. We then present a review of the main approaches featured in recent research on multi-agent deep reinforcement learning, describing state-of-the-art algorithms that will be used in further chapters. Finally, we propose a critical analysis of the current state of research on multi-agent deep reinforcement learning, specifically regarding problems related to robotic applications: how the field tackles these issues and what could be improved.

In **Chapter** 4, we tackle the problem of multi-agent exploration. As in the single-agent case, reinforcement learning tools require efficient exploration strategies to work well, especially in environments where rewards do not provide enough guidance. We demonstrate how, in the multi-agent case, exploring locally can be inefficient as it can fail to unveil optimal joint behaviours. Thus, tackling this issue requires a specific, multi-agent strategy for exploration. We propose a method based on intrinsic

motivation, that incites agents to explore new coordinated behaviours. We show that state-of-the-art approaches fail to learn highly coordinated behaviours, and can benefit from our joint intrinsic motivation method to improve their convergence properties.

Lastly, in **Chapter** 5, we tackle the problem of learning to communicate in multi-agent systems. A popular approach to this problem in multi-agent deep reinforcement learning is to learn differentiable emergent communication, where agents develop their own communication system through the optimisation of their reinforcement learning objectives. While this conveniently fits the gradient-based learning paradigm of deep reinforcement learning, we argue that it fails to fulfil the needs of communicating artificial agents. We instead advocate for the deployment of language-augmented agents in multi-agent robotic settings. Teaching a pre-defined language to agents provides them with multiple advantages: efficient communication, grounding of representation learning, better generalisation abilities, and interactions with humans. We propose a method for training language-augmented agents and a series of experiments to demonstrate their qualities.

Communications and Publications

In the course of my thesis, I had multiple opportunities to share my work with other researchers and students:

- Our work on joint intrinsic motivation, presented in Chapter 4, has been **published** in the *Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)* in 2024 (link), following a preliminary presentation in the *Adaptive and Learning Agents (ALA)* workshop at AAMAS 2023.
- I took part in the redaction of a journal article investigating different approaches to communication in swarm robotics, the article is currently **under review** in the *Philosophical Transactions of Royal Society A*.
- Chapter 3 was the subject of a course on multi-agent deep reinforcement learning for master's students at *Sorbonne Université*.

In addition, our work on language-based communication, described in Chapter 5, is currently in preparation for a submission to a conference next January.

Chapter 2

Introduction to Reinforcement Learning

Definitions and techniques

2.1 Introduction: Definition, Trends and Limitations

Reinforcement Learning (RL) is a machine learning approach that enables an artificial agent to learn how to act in order to maximise some behaviour-dependent reward signal. The core mechanic of RL is trial-and-error, where the learner sequentially tries actions, obtains rewards, and updates their strategy based on their findings. Throughout many of these cycles, the learner progressively acquires knowledge about the task and devises a strategy to solve it. In this regard, it differs from other machine learning paradigms, namely *supervised* and *unsupervised* learning, that use some fixed set of data points and learn to optimise some objective over this data. With RL, there is no data beforehand, only some definition of the learning environment and the learning agent[1]. From there, the agent will generate its own training data by acting in the environment, which will then influence its learning journey.

Another key difference is in the optimised objective. In supervised and unsupervised learning, the learning task is defined with some mathematical cost function computed over the model's prediction and the data points. This cost function is usually defined to be easily differentiable so that the model can find a way to minimise it using gradient descent. In RL, the learning task is defined solely by the rewards, that should be maximised. While we will often refer to the reward as a function, in practice it is not a mathematical function of the learner's actions. The reward is a signal given by the environment to the agent: it may be the score in a video game, a note given by a human observer, or a win state (e.g., 1 for "game won", -1 for "game lost", and 0 for "unfinished game") in a tabletop game. Thus, RL algorithms use a series of mathematical tricks to translate the objective of "maximising the rewards" into a learnable objective. This makes learning significantly more difficult. In practice, supervised learning is always preferred to RL if the context allows it. But, it also opens a whole range of possibilities where a learning algorithm can try to optimise any given numerical signal.

A particularly appealing aspect of RL lies in its way of emulating how humans and other animals learn in the real world. In fact, while the term "reinforcement learning" now refers widely to a class of learning algorithms, the idea of learning through reinforcement is also used beyond the scope of computer science. Many concepts

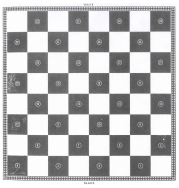
[1] Some forms of RL, like *offline RL*, use previously gathered data to learn RL objectives, but generally this is not the case.

of RL derive from neuroscience and are still used to describe the functioning of our brain (Friston, 2010; Schultz et al., 1997) or to study mental illness (Montague et al., 2012). Reinforcement experiments are used in biology and psychology to characterise intelligence and investigate how learning occurs in animals (Brembs, 2010; Gardner & Gardner, 1984; Rescorla & Wagner, 1972). RL is also closely tied to evolutionary biology, where evolution can be defined as finding the best strategy to adapt to the environment through natural selection. This definition translates to a whole subclass of RL, suitably called "Evolutionary Algorithms" (Eiben & Smith, 2003).

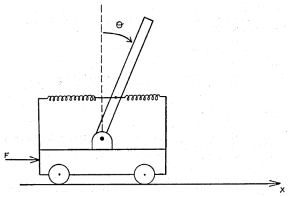
Coming back to computer science, RL was introduced during the second half of the 20th century, from various fields of research in neuroscience, optimisation, control theory, and electrical engineering (Sutton & Barto, 2018). Two main threads can be recognised as direct origins of RL theories. One stems from the ethologist study of learning by trial-and-error, and especially with reinforcing events (Rescorla & Wagner, 1972; Thorndike, 1911). These ideas influenced early works in artificial intelligence, later integrating trial-and-error learning into electrical engineering (Walter, 1950) and computer science (Minsky, 1961). The second branch is *dynamic programming* (Bellman, 1966), which investigated solving a complex problem by breaking it into easier sub-problems. This involved the definition of many mathematical tools that became the foundations of RL algorithms (see Section 2.2).

Throughout these developments, various types of problems were studied, starting from recreational games like checkers (see Figure 2.1.a; Samuel, 1959) and tic-tac-toe (Michie & Chambers, 1968). Such games offer a convenient setting for reinforcement learning: the rules and actions are well-defined, the reward function is easy to define (e.g., 1 if the game is won, 0 otherwise), and they are simple enough to be simulated on computers. This last point is crucial, as RL usually requires a large number of experiences to find the optimal strategy, a problem referred to as "*sample inefficiency*". Being able to run fast experiments on simulated environments is critical. With more powerful computers, increasingly difficult games have been tackled with RL methods. Notable instances are TD-Gammon for backgammon (Tesauro, 1994) and AlphaGo (see Figure 2.1.f; Silver et al., 2016) which demonstrated how RL techniques could be used to achieve superhuman performance in the challenging game of Go. Following tabletop games, video games have recently been used as playgrounds for RL research. They offer similar advantages (i.e., simulated, scores as rewards) while providing increased complexity in the variety of tasks and environments. The Atari benchmark (see Figure 2.1.e; Bellemare et al., 2013) has been broadly adopted for its wide range of different games that agents play by looking at the pixel images, just like humans do. Recently, the popular video game Minecraft has been invested as a new challenge for artificial intelligence research, thanks to its extremely diverse environment and vast set of acquirable skills (see Figure 2.1.h; Guss et al., 2019; Oh et al., 2016), showing the great potential of modern video games for artificial intelligence research.

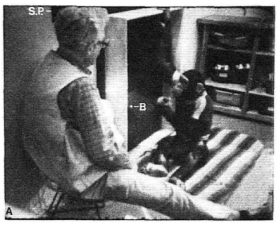
Concurrently to games, RL has also been employed in optimal control and robotics. An early example of this is the classical pole-balancing problem (Michie & Chambers, 1968) that features a pole balancing on a moving cart (see Figure 2.1.h): the cart can move right or left and the reward is made only of a failure signal when the pole falls or the cart reaches the end of the track. While controlling robots is a major drive of RL research, it remains a great challenge. Robotics problems feature high-dimensional, continuous states and actions, which contrast with the discrete, finite states and actions of tabletop games. To overcome this, solutions include partitioning the space of states or actions to reduce complexity using linear function approximation (Busoniu



(a) Checkers
(Samuel, 1959)

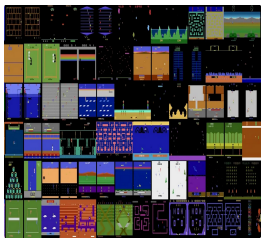


(b) Pole-balancing problem
(Michie & Chambers, 1968)

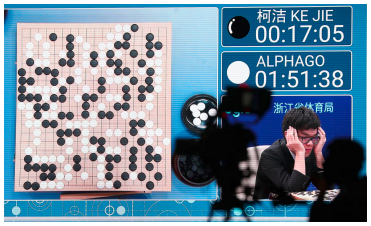


(c) Teaching word to a chimpazee
(Gardner & Gardner, 1984)

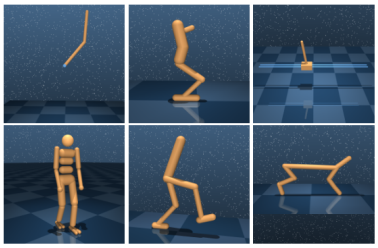
(d) Flying an helicopter
(Bagnell & Schneider, 2001)



(e) Atari benchmark
(Bellemare et al., 2013)

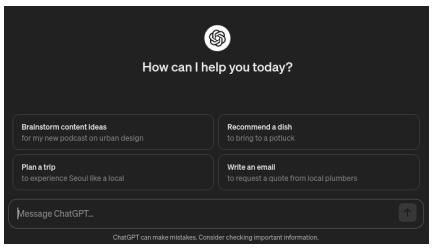


(f) AlphaGo vs. best human
(Silver et al., 2016)
(Photograph by Wu Hong, in The Guardian)



(g) Deepmind Control Suite in MuJoCo
(Tassa et al., 2018)

(h) Learning skills in Minecraft
(Guss et al., 2019)



(i) ChatGPT
(chat.openai.com)
(Ouyang et al., 2022)

Figure 2.1: Reinforcement learning environments in various domains of research.

et al., 2017), or using neural networks to learn to extract features from continuous states automatically (Bishop, 2006) (see Section 2.4.3). Another attribute of robotic environments is partial observability, where robots only have incomplete information about the current state of the environment. This has been tackled by learning a model of the dynamics of the environment (Bagnell & Schneider, 2001) (see Section 2.3.3) or by providing the learner with memory to keep track of past information (Wierstra et al., 2007). Another challenge is the design of a reward function. This requires translating the robot's objective into a numerical signal, which can be tedious. Intuitively, a robot could be rewarded positively when it completes its task. But this kind of sparse reward is difficult to learn. To fix this, reward shaping methods (Laud, 2004; Ng et al., 1999) construct rewards to consistently guide the learner towards the objective in a limited number of experiences. Finally, a last great challenge for learning robotic control with RL is the cost of experimenting with robots. Sample inefficiency and learning by trial-and-error make experimenting with physical hardware expensive and time-consuming. For these reasons, some approaches start by learning from demonstrations, before training with RL methods (Abbeel et al., 2006; Vecerik et al., 2017). Learning from demonstrated sequences of actions simplifies the learning problem by recasting it as a supervised learning problem. Another solution is to turn to simulated environments like ROS (Macenski et al., 2022) or MuJoCo (see Figure 2.1.g; Todorov et al., 2012). However, using a model learnt in simulation in the real world is proven to be a difficult task because of the inevitable gap between the best simulations we have and the real world, a problem termed the "*reality gap*" (see Section 3.3.2 for more details on this issue).

While control has been a main focus of RL, it has also been studied in other domains of artificial intelligence research. A recent success has been the use of Reinforcement Learning from Human Feedback (RLHF; Christiano et al., 2017; Stiennon et al., 2020) in language modelling. Language modelling is the task of learning to predict the next word in a given sequence. In this context, RLHF re-trains initially supervised language models to better fit human preferences. This technique is made possible by using reward modelling (Leike et al., 2018), where a model learns to predict the rewards given by humans and is then used to train the RL model on a large number of experiences. This allowed the development of conversational agents like the now widely used ChatGPT (Ouyang et al., 2022). The success of RLHF approaches demonstrates the great potential of RL for learning to fit human needs.

All these examples show the tremendous potential of RL to shape artificial intelligence research further. This is largely due to the advent of deep reinforcement learning that sparked an ongoing revolution in the field during the last decade. Today, RL is widely considered an important block to building more advanced forms of artificial intelligence, if not the main tool to do so (Silver et al., 2021). However, this view is not shared by all (Mitchell, 2021) and RL still suffers from many limitations. We have already mentioned the problems of sample inefficiency, safety, and the high cost of RL training. They all generally hinder performance and prevent RL algorithms from being used extensively in robotics (Ibarz et al., 2021; Sünderhauf et al., 2018) and autonomous driving (Chen et al., 2023; Kiran et al., 2022), with supervised alternatives like imitation learning being far superior (Hester et al., 2018). These issues motivate various practical solutions (Ibarz et al., 2021), like the aforementioned reward shaping strategy (Laud, 2004), as well as whole lines of research such as safe RL that explicitly constrain RL to learn from safe states (García et al., 2015), and curriculum learning that focuses on designing a schedule of increasingly difficult setups (Bengio et al., 2009; Uchendu et al., 2023). RL also suffers from more

practical issues inherent to its definitions and the tools it uses: reproducibility, high variance, and intricacy of the algorithms and their implementations (Henderson et al., 2018). Finally, as with the rest of methods based on deep neural networks, there is an issue of explainability and interpretability of deep RL. Deep neural networks act as black boxes that have no explicit mean for interpreting their reasoning and, thus, explaining their results (Rudin, 2019; Samek et al., 2021). With deep RL agents, interpreting the learnt behaviours and understanding how training shaped them this way is often difficult. Some solutions are pursued, like hierarchical RL (Hafner et al., 2022; Pateria et al., 2021; Shu et al., 2018) or language-augmented RL (see Section 5.2.4 for a more in-depth review of the literature on this matter).

2.2 Elements of Reinforcement Learning

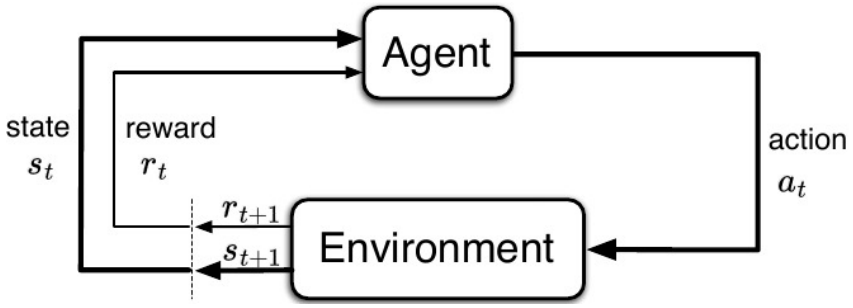


Figure 2.2: Diagram of the reinforcement learning interaction between the agent and its environment, from Sutton and Barto, 2018.

Reinforcement learning is the process of learning how to complete a particular **task** from trial-and-error. In this process, we refer to the learner as an **agent**. The agent interacts with its **environment** over a sequence of discrete time steps denoted by $t = \{0, 1, 2, ...\}$. Each interaction consists in the agent receiving information on its **state** s_t from the environment and consequently choosing an **action** a_t to perform. Following the agent's action, the environment produces a **reward** r_{t+1} and a new state s_{t+1}. This process, illustrated in Figure 2.2, repeats indefinitely or stops after a finite number of steps T. In the latter case, we call this finite sequence of steps an **episode**. In the context of RL, the agent's goal is to pick the right actions to maximise the cumulative rewards in the long run.

This terminology serves to describe any possible RL setting. The agent refers to the central entity of the experiment, defined concretely as a computer system. It is often considered "*intelligent*" for being: (i) **reactive** to its environment, perceiving information from its surroundings and acting in response to these input signals; (ii) **proactive**, shaping its behaviour to satisfy its current goal; (iii) able to **learn** from its experience (Wooldridge & Jennings, 1995). The goal of the agent is defined primarily by the task. It can be as straightforward as "picking up an object", or be a more abstract objective like "give this person what they need". The environment refers to the setting where this task is conducted, encompassing all elements except for the agent. The frontier between the agent and the environment is not a physical boundary but rather a conceptual one: everything that cannot be changed arbitrarily by the agent is considered part of the environment. For example, sensors, motors and mechanical joints of a robot should be considered as parts of the environment, while the agent in this case is the program controlling the robot. The states and actions depend on the agent's capabilities in the environment. Like the goal, they can

take different forms, with various levels of abstraction. For example, the state of a robot could be concrete information about its surroundings coming from its sensors (e.g., camera, lidar, inertial measurement unit), or more symbolic information like the state of being in a particular room or not. Similarly, actions can be as concrete as "turning 4.5 degrees left", or more high-level like "flip the pancake". The RL framework accounts for all these levels of abstraction, allowing its use in many different settings.

2.2.1 Markov Decision Processes

The universally accepted framework for building RL algorithms is the **Markov Decision Process** (MDP). The MDP is a mathematical formalisation of the RL problem. It defines this problem as a tuple $\langle \mathbf{S}, \mathbf{A}, \mathcal{T}, \mathcal{R} \rangle$. In this tuple, $\mathbf{S}$ represents the set of all possible states in the environment: $\mathbf{S} = \{S_0, S_1, S_2, ...\}$. Similarly, $\mathbf{A}$ is the set of all possible actions: $\mathbf{A} = \{A_0, A_1, A_2, ...\}$. In the example of Figure 2.3, $\mathbf{S}$ is the set of all eleven possible positions in the grid, numbered from 0 to 10, and $\mathbf{A}$ contains four possible actions: $\mathbf{A} = \{\texttt{move\_up}, \texttt{move\_right}, \texttt{move\_down}, \texttt{move\_left}\}$. This setup describes the particular case of a *finite* MDP, where the number of possible states and actions are finite. However, an MDP could also allow infinite states and actions if needed. For example, an autonomous driving system could have its state defined as a continuous GPS position and its action as the continuously defined rotation angle to apply to the steering wheel.

Next is the **transition function** $\mathcal{T}$ that defines the probability of transitioning from state s to state s' by taking action a:

$$\mathcal{T}(s' \mid s, a) \coloneqq \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}. \tag{2.1}$$

It dictates how the environment changes after performing an action. The transition function translates the robot's move in terms of probability: for example, in the setup of Figure 2.3, $\mathcal{T}(S_1|S_0, \texttt{move\_right}) = 1$ and $\mathcal{T}(S_4|S_0, \texttt{move\_rigth}) = 0$. These probabilities describe a *deterministic* environment, where a given action performed in a particular state always leads to the same outcome. On the other hand, in *stochastic* environments, an action may have multiple possible outcomes. For example, an action may fail with a probability of 0.01: $\mathcal{T}(S_1|S_0, \texttt{move\_rigth}) = 0.99$ and $\mathcal{T}(S_0|S_0, \texttt{move\_rigth}) = 0.01$. In this synthetic environment, we control the probability of each transition. However, in more realistic settings, the transition function is unknown.

Finally, the **reward function** $\mathcal{R}$ is the expected value of the reward obtained during a transition from state s to s' with action a:

$$\begin{aligned} \mathcal{R}(s, a) &\coloneqq \mathbb{E}[r_{t+1} \mid s_t = s, a_t = a] \\ &= \sum_{s' \in \mathbf{S}} \mathcal{T}(s' \mid s, a) r_{t+1} \end{aligned} \tag{2.2}$$

In Figure 2.3, the reward function is made of two elements: a positive reward signal if the agent reaches the goal and a negative signal for all other states. The former is straightforward: it indicates that the task is complete. The latter penalises the agent for entering any state that is not the terminal state. This kind of penalty is used often to urge the agent to complete the task as fast as possible: the more steps are taken, the larger the accumulated penalty. The reward depicted in Figure 2.3 can be described as *sparse* because there are very few positive reward signals to guide the

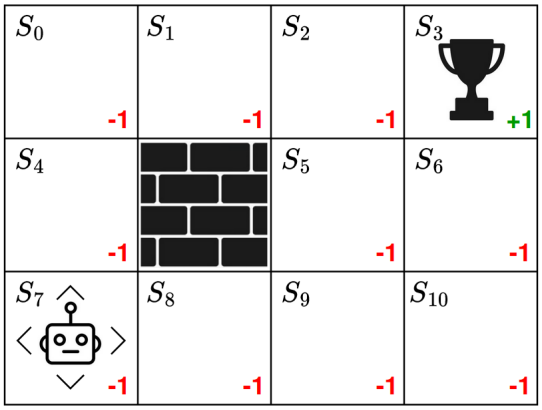


Figure 2.3: Simple reinforcement learning environment where the goal is for a robot to navigate in the grid and find the trophy cell. The robot can move in all four directions. The episode ends if the robot reaches the trophy or if the maximum number of steps T is reached. States are defined by the position of the robot in the grid. Rewards for entering each state are indicated in the bottom-right corner of each cell.

agent towards the objective. In this simple environment, this should not be an issue. But, in more complex settings with a larger number of possible states, sparse rewards can become problematic.

Often, we see the transition probability and reward function brought together in a single function p that defines the **environment dynamics**, with:

$$p(s', r \mid s, a) \coloneqq \Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t = s, a_t = a\}, \tag{2.3}$$

which is the probability of being in state $s' \in \mathbf{S}$ and receiving reward $r \in \mathbb{R}$, after performing action $a \in \mathbf{A}$ in state $s \in \mathbf{S}$.

2.2.2 Modeling the Agent

In the context of an MDP, the agent is modelled as a **policy** that selects actions depending on the current state of the environment. This policy can be either *deterministic* or *stochastic*. A deterministic policy is a function π that directly maps the current state s to an action a to perform: $\pi : \mathbf{S} \to \mathbf{A}$, $\pi(s) = a$. A stochastic policy π is defined as a function that maps states to probabilities over possible actions: the probability of choosing each action $a \in \mathbf{A}$ in state s is $\pi(a|s) \in [0, 1]$, with $\sum_{a \in \mathbf{A}} \pi(a|s) = 1$. The policy function is used to describe how the agent selects its actions. In some algorithms, the policy is explicitly learnt during training. But, the policy might also be arbitrarily based on other learnt elements, or even random or unknown.

The agent uses the policy function to act in the environment, which results in a sequence of states, actions, and rewards that is called a **trajectory**, denoted τ:

$$\tau = (s_0, a_0, r_1, s_1, a_1, r_2, s_2, a_2, ...). \tag{2.4}$$

The goal of an RL algorithm is to find a policy that maximises the accumulated rewards in the trajectories it generates. To that end, the policy must choose actions to maximise the sum of future rewards which is called the **return** G_t, starting from any step t:

$$G_t \coloneqq r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T, \tag{2.5}$$

with the episode ending at step T. In the case where T is infinite (i.e., the episode never ends) this formulation is problematic as it can result in an infinite return. This motivates another formulation of the return where future rewards are discounted by their distance in time to the current step. This **discounted return** is defined as:

$$G_t \coloneqq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}, \tag{2.6}$$

with $\gamma \in [0, 1]$ the *discount factor*. The discounted return favours rewards that are closer in time. The value of the discount factor controls how far in time the reward should affect the decision of the agent: a γ close to zero will favour immediate rewards, while a γ close to one will give importance to long-term returns. In practice, almost all RL methods use the discounted return, even if the episode ends in a finite number of steps T.

As the return cannot be computed before the end of the episode, many RL algorithms learn **value functions** to predict the expected return at any point during the episode. Thus, the **state-value function** associated with policy π, is defined as the expected return starting from any state s and following policy π:

$$V_\pi(s) \coloneqq \mathbb{E}_\pi[G_t \mid s_t = s], \tag{2.7}$$

where $\mathbb{E}_\pi[.]$ is the expected value of any random variable given that the agent always follows the policy π. Following the value of a particular state, we can also learn the value of performing a particular action in a particular state. This is the **action-value function** of policy π, defined as the expected return starting from state s, performing action a, and then following policy π:

$$Q_\pi(s, a) \coloneqq \mathbb{E}_\pi[G_t \mid s_t = s, a_t = a]. \tag{2.8}$$

As we will see in further sections, these value functions can be learnt from experience and then used to choose actions accordingly or to do planning over multiple time steps.

An important property of the discounted return defined in Equation 2.6 is its successiveness: the return at step t can be expressed in function of the return of the following steps:

$$\begin{aligned} G_t &\coloneqq r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + ... \\ &= r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} + ... \\ &= r_{t+1} + \gamma G_{t+1}. \end{aligned} \tag{2.9}$$

The same property can therefore be found for the state-value function:

$$\begin{aligned} V_\pi(s) &\coloneqq \mathbb{E}_\pi[G_t \mid s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma G_{t+1} \mid s_t = s] \\ &= \mathbb{E}_\pi[r_{t+1} + \gamma V_\pi(s') \mid s_t = s, s_{t+1} = s']. \end{aligned} \tag{2.10}$$

The *optimal policy* is defined as the policy whose expected return is greater than or equal to that of all other possible policies, for all possible states. In other words, a policy π is the optimal policy if $V_\pi(s) \geq V_{\pi'}(s)$, for all $s \in \mathbf{S}$ and all possible other policies π'. The optimal policy, denoted π^* is associated with the optimal value

functions V^* and Q^*. By definition, the optimal policy always chooses the action with the largest action-value. That is,

$$\pi^*(s) = \arg\max_{a \in \mathbf{A}} Q^*(s, a). \tag{2.11}$$

Following this, the optimal state-value function of any state s is equal to the maximum value of Q^* in this state:

$$\begin{aligned} V^*(s) &= \max_{a \in \mathbf{A}} Q^*(s, a) \\ &= \max_{a \in \mathbf{A}} \mathbb{E}_{\pi^*}[G_t \mid s_t = s, a_t = a] \\ &= \max_{a \in \mathbf{A}} \mathbb{E}_{\pi^*}[r_{t+1} + \gamma V^*(s') \mid s_t = s, a_t = a, s_{t+1} = s']. \end{aligned} \tag{2.12}$$

This last result is the *Bellman optimality equation* (Bellman, 1957) for the state-value function. By unfolding the expected value, we get:

$$V^*(s) = \max_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} \mathcal{T}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma V^*(s')\right]. \tag{2.13}$$

This is another form the Bellman optimality equation that will be used to define RL algorithms in the next section. It can also be expressed for the optimal action-value function as:

$$\begin{aligned} Q^*(s) &= \mathbb{E}_{\pi^*}[r_{t+1} + \gamma \max_{a' \in \mathbf{A}} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a] \\ &= \sum_{s' \in \mathbf{S}} \mathcal{T}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma \max_{a' \in \mathbf{A}} Q^*(s', a')\right]. \end{aligned} \tag{2.14}$$

Bellman optimality equations are the basis of most RL algorithms, as they allow to gradually learn the optimal value functions through an iterative process described in Section 2.3.1. In finite MDPs, the optimal value functions are proven to exist and be unique. But, that is not the case for more complex environments. Therefore, RL algorithms usually learn to estimate these functions through various techniques.

Finally, another tool used in RL is a **model** of the environment. A model predicts the outcomes of the agent's action in terms of changes in the environment and resulting rewards. Concretely, a model learns to approximate the environment dynamics p defined in Equation 2.3. Thus, the model is a function $\widehat{p}$ that maps the current state and action to the next state and reward. It can output a direct prediction, with $\widehat{p} : S \times A \to S \times \mathbb{R}$, or a probability distribution over states and rewards (like in Equation 2.3). These two forms of models can be used in different ways for doing *planning* over one or multiple time steps. As we will see in Section 2.3.3, planning usually involves looking ahead by predicting future outcomes and then using these predictions to compute or improve value function predictions, which, in turn, can be used to select actions that maximise the expected return.

2.3 Foundational RL Algorithms

After introducing the core elements used in RL, we present different approaches to learning from reinforcement. These approaches rely on learning value functions, policies, models, or a combination of the three, to devise an efficient strategy of actions.

In this section, we present the foundational RL algorithms that serve as a basis for building deep RL and multi-agent RL methods.

2.3.1 Value Estimation

As presented in the last section, value functions are a fundamental component of RL as they aim to predict the most important thing for the agent: the expected return of its actions. In RL algorithms, value functions are used for two different purposes:

- First, to evaluate a given policy, which may be unknown, by estimating the expected return obtained following this policy.
- Second, to explicitly learn a strategy to control the agent, in which case the agent's policy will be derived from the learnt value function.

In both cases, the algorithms presented learn estimates of the value functions defined in the last section (see Equations 2.7 and 2.8).

Value Iteration

To learn a value function estimate, a basic technique is *value iteration*. It learns through an iterative process of gathering experience and improving the value function. Value iteration turns the Bellman optimality equations (see Equations 2.13 and 2.14) into an update rule. That is, given a value function V at our disposal, we compute a new value function based on the result of the Bellman equation:

$$V(s) \leftarrow \max_{a \in \mathbf{A}} \sum_{s' \in \mathbf{S}} \mathcal{T}(s' \mid s, a) \left[\mathcal{R}(s, a, s') + \gamma V(s') \right], \tag{2.15}$$

for each state $s \in \mathbf{S}$. The arrow indicates that we replace the current value $V(s)$, typically stored in a table with one value for each state, with a new one resulting from the calculation. In finite MDPs, executing this one time yields a new value function that is necessarily closer to the optimal value function. Thus, starting with an arbitrary initial value function (e.g., $V(s) = 0$ for all $s \in \mathbf{S}$), repeating the value iteration operation ensures that V converges towards V^*. In Figure 2.4, value iteration is executed on the simple RL environment previously introduced. This example shows how value iteration can learn the optimal value function quickly in a simple environment with a limited number of states.

An interesting property of such algorithms is the fact that they use the current estimate to compute the new one. This is called *bootstrapping*. As we will see, this feature is at the core of many value estimation methods. Bootstrapping can greatly improve the convergence speed of these algorithms: in Figure 2.4, using the current estimate V allows each iteration to have more impact (depending on the order in which states are processed). But, this can also induce difficulties: in more complex settings, using imperfect estimates can sometimes produce even worse new estimates, thus compromising the learning process.

Monte Carlo Methods

A crucial drawback of the value iteration algorithm is that it assumes complete knowledge of the environment dynamics. Only because we know the probabilities of all transitions and their resulting reward, are we able to compute the expected value in the update rule (2.15). In most RL settings, the environment dynamics are unknown. Therefore, it is impossible to compute the exact value iteration update rule. What

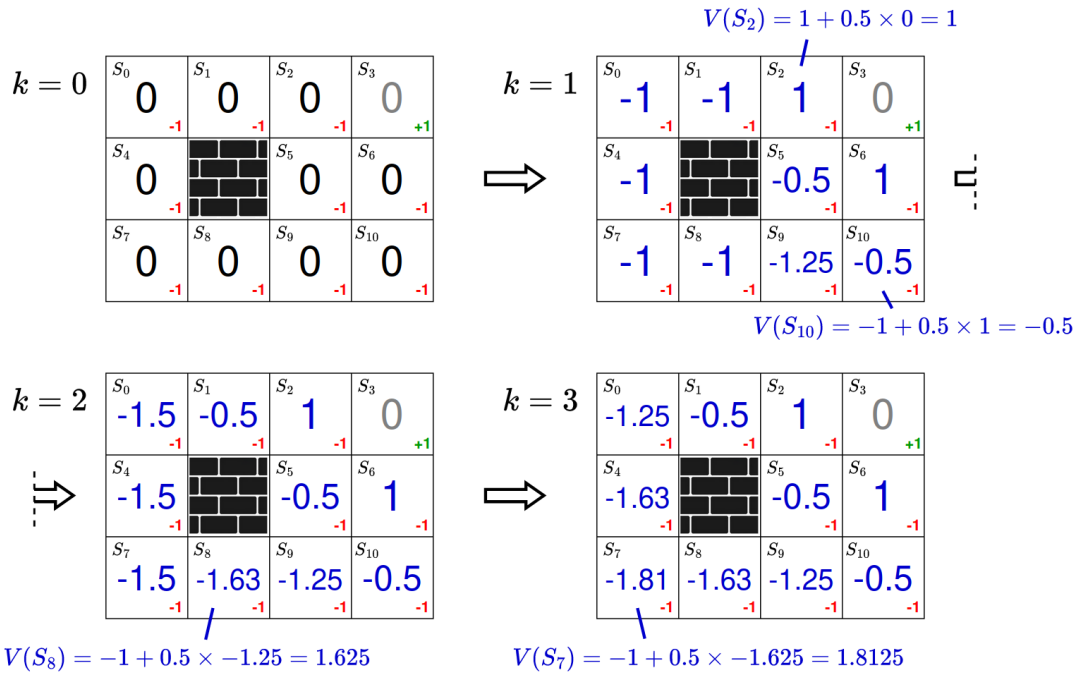


Figure 2.4: Value iteration algorithm executed on the example of Figure 2.3. We start with an initial value function $V(s) = 0$ for all $s \in \mathbf{S}$. Then, each iteration k applies the update rule of Equation 2.15, with $\gamma = 0.5$, with the result $V(S_t)$ shown in the corresponding cell. As state S_3 is terminal, its value is considered to be 0. In three iterations, we converge to the optimal value function. Note that, because we compute the new V for each state successively, the order of the states affects the result of each iteration and the number of iterations required for converging.

is often possible, however, is to run simulated experiments and gather sequences of states, actions and rewards. This is called "*sampling*" trajectories from the environment. These sampled trajectories can then be used to compute approximations of what we want to predict. For example, given a trajectory τ generated using the sampling policy π, we can compute the return from any state $s \in \tau$ (according to Equation 2.6) and use the result as an estimation of the expected return from state s, i.e., $V_\pi(s)$. This is the core idea behind **Monte Carlo** methods: estimating the value function of a given policy by averaging sampled returns. This can be performed iteratively, by keeping a running estimate of the value function and updating it after each episode with:

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha \left[G_t - V_\pi(s_t) \right], \tag{2.16}$$

for all $t < T$, with G_t the experienced return from state s_t and $\alpha \in]0, 1]$ a constant parameter called the **learning rate**. This update modifies the value of $V_\pi(s_t)$ by "moving it" towards the experienced return. The learning rate α controls the magnitude of this modification: $\alpha = 1$ would mean the new value of $V_\pi(s_t)$ is G_t, $\alpha < 1$ yields a new value between the old $V_\pi(s_t)$ and G_t. Using a learning rate is important because one sampled return is a poor approximation of the actual state-value function. Thus, repeating this update many times will allow to gradually move towards a better estimate of $V_\pi(s_t)$.

Temporal Difference Learning

A problem with Monte Carlo methods is that they require waiting for the end of each episode to compute the return of all states. In some environments, the episode might

last for several hours or even never end, making Monte Carlo methods impractical. A solution for this is to use **Temporal-Difference** (TD) learning (Sutton, 1988). TD methods similarly learn an estimate of the value function, but using only a limited number of transitions. The simplest TD method, called *TD(0)*, updates the value after only one time step, with

$$V_\pi(s_t) \leftarrow V_\pi(s_t) + \alpha \left[r_{t+1} + \gamma V_\pi(s_{t+1}) - V(s_t) \right]. \tag{2.17}$$

The return in the Monte Carlo update rule (Equation 2.16) is replaced by its estimation using the current value function. The expression inside the brackets is often referred to as the *TD-error* or *Bellman error* as it is derived from the Bellman equation (2.13). It measures the difference between the current estimated value of s and the better estimate $r_{t+1} + \gamma V_\pi(s_{t+1})$ given by TD(0), called the *TD-target*. The TD-target, can be extended to use the experience of multiple time steps to have a better estimate of the return (Sutton, 1988).

Q-Learning

For now, we have looked at methods to learn estimates of the state-value function. The *Q-learning* algorithm (Watkins, 1989) applies TD learning to the action-value function to directly learn an efficient way to control the agent. Given many transition tuples of form $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$, we can learn an action-value function Q with:

$$Q_\pi(s_t, a_t) \leftarrow Q_\pi(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q_\pi(s_{t+1}, a) - Q_\pi(s_t, a_t) \right]. \tag{2.18}$$

This update rule uses the current estimate Q as an approximation of the optimal Q^* to compute the Bellman error. With this update, Q-learning learns the action-value function of the "*greedy* policy with respect to Q", i.e., the policy that always chooses the action with the highest action-value according to Q: $\pi(s) = \arg\max_a Q_\pi(s, a)$. Given enough training, the learnt action-value function can be used to choose actions following the greedy policy.

This method is shown to effectively learn a good approximation of Q^* as long as all states are explored enough. Thus, it requires a sampling policy to explore the environment and gather the transition tuples. The simplest possible policy is to randomly select actions in all steps. But, this means that some irrelevant states will be explored as much as others. Instead, we might want to focus on the states and actions that have more potential. This is a common dilemma in RL algorithms, where we need to balance a trade-off between *exploration* and *exploitation*:

- We want to explore different behaviours to better understand the environment dynamics and improve our estimations.
- But, at the same time, we want to exploit our past findings to learn faster.

In the case of Q-learning, exploration is done by randomly selecting actions. And, exploitation is done by using the greedy policy to select action.

To balance the exploration-exploitation trade-off, Q-learning employs the ϵ-*greedy* policy. At each time step, ϵ-greedy decides between choosing an action randomly or following the greedy policy. It uses a parameter ϵ between 0 and 1 that determines the probability of choosing the action randomly. Typically, this parameter starts at 1 and decreases throughout training towards 0. This allows a progressive shift between random exploration at the start when the estimations are bad, and exploitation when estimations start getting better. While this is far from being a perfect exploration

strategy (more on that in Chapter 4), it offers a simple solution for exploration in Q-learning. An algorithm for executing Q-learning with ϵ-greedy is described here:

Algorithm 1 Q-learning with ϵ-greedy policy

Initialize $Q(s, a)$ arbitrarily
Initialize $\epsilon = 1$
Initialize α (learning rate), γ (discount factor)
for each episode **do**
  Initialize state s_0
  for t=0,...,T **do**
    Execute ϵ-greedy policy:
      With probability ϵ, choose a random action a_t
      Otherwise, choose $a_t = \arg\max_a Q(s_t, a)$
    Perform action a_t, observe reward r_{t+1} and new state s_{t+1}
    $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\right]$
    $s_t \leftarrow s_{t+1}$
    $\epsilon \leftarrow \max(\epsilon - 0.001, 0)$
  end for
end for

An important observation to make about all value-based methods presented in this section is the fact that they all rely on a list of values to keep track of the current estimations. In Q-learning, for example, the action-value function is represented as a table with states as rows and actions as columns, where each cell (s, a) contains the current estimate $Q(s, a)$. For this reason, these approaches are qualified as "tabular". This aspect highlights a clear drawback of these methods: they are limited in the number of single values they can keep track of. To learn a perfect policy or value function, we would need to exhaustively try all possible actions in all possible states. But, this is impossible in rich environments with very large, if not continuous, state and action spaces. Thus, tabular value functions are limited to environments where they can realistically explore all states or state-action pairs enough times to build good value estimates. In fact, they are even incapable of dealing with continuous states and actions, as they would require infinitely large value tables, which makes their application to real world tasks significantly more difficult.

A solution for this is to change the form of the value function and represent it as a parameterised function. Instead of keeping track of all values in a table, the value function is defined as a mathematical function of states that outputs the value of the input state. We would write the value functions as $V(s, \varphi) : S \times \mathbb{R}^{d^\varphi} \rightarrow \mathbb{R}$ or $Q(s, a, \varphi) : S \times A \times \mathbb{R}^{d^\varphi} \rightarrow \mathbb{R}$, with $\varphi \in \mathbb{R}^{d^\varphi}$ a vector of parameters of dimension d^φ used to define the value function. The actual definition of the function is arbitrary. It can be as simple as a linear transformation of the input state, or use more complex definitions such as artificial neural networks (see Section 2.4). In any case, φ is the set of real-valued parameters that dictates the outputs of the value function. An advantage of having a parameterised value function is that it can now deal with continuous states. There is no more table keeping track of all possible values. The new value function should now be able to predict the value of states it has never seen before, by generalising from its training experience (see Section 2.42 for more detail). But, this also comes with downsides. Learning a set of parameters that generalises well can be difficult. From the designer's point of view, the value function is now a black box that computes values from a series of mathematical operations, which can

be hard to interpret.

Despite their limitations, value-based approaches are still widely used across RL research. Their simple definition and low computational resources make them easy to implement. In the next sections, we will describe ways to alleviate some of the drawbacks of value-based algorithms. Furthermore, we will see that, when combined with other techniques, value functions play a crucial role in building good RL agents.

On-Policy vs. Off-Policy Learning

Q-learning, as defined in Algorithm 1, is characterised as an **off-policy** algorithm, because its update does not depend on the policy used to generate the experience, which is referred to as the *behaviour* policy. This is in contrast with **on-policy** algorithms, such as SARSA (Rummery & Niranjan, 1994), that learns an action-value function, similarly to Q-learning, but based on an experience tuple $\langle s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1} \rangle$:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]. \tag{2.19}$$

The difference with the Q-learning update (see Equation 2.18) is in the choice for action at state s_{t+1}, used in the TD-target:

- SARSA uses the action a_{t+1} that was selected by the behaviour policy. Therefore, its update depends on the current behaviour policy of the agent (i.e., on-policy).
- Q-learning uses the learnt action-value to select a greedy action that serves as a_{t+1} in the TD-target: $r_{t+1} + \gamma \max_a Q(s_{t+1}, a)$. This TD-target depends only on Q and is independent of the behaviour policy (in this case the ϵ-greedy policy). Thus, Q-learning is off-policy.

In other words, off-policy value learning learns the value of a policy (in this case, the greedy policy with respect to Q), using another policy to gather experiences (here, ϵ-greedy). On the other hand, on-policy value learning directly learns the value of the behaviour policy. Note that, while we illustrate these properties with action-value learning algorithms, these can be used to characterise state-value or policy learning algorithms (see Sections 2.3.2 and 2.5.2).

The distinction between on- and off-policy is subtle but has one major implication. Off-policy algorithms can learn functions (value or policy) based on data generated by other, totally separate processes. For example, Q-learning could learn the optimal action-value function based on experiences gathered by a random policy, a pool of various independent policies, or even human experts. This particular feature allows using **experience replay** (Lin, 1992), where, during training, experiences are stored in a memory, called the *replay buffer*, and are then reused any number of times in later updates. To make an update, a batch of past experiences is drawn randomly from the replay buffer and the learning update is performed on all these samples. Training on past experiences allows learning on more diverse data, thus specialising less on the current behaviour policy. In this case, off-policy learning is required because the replay buffer is full of past experiences generated by previous (different because trained fewer times) versions of the behaviour policy.

2.3.2 Policy Gradients

Learning a Parameterised Policy Function

In Section 2.2.2, we said that an RL agent is primarily defined by its policy function π, which maps states to actions. We saw in the last section that value functions can be used to evaluate an unknown policy or even to learn a policy by being greedy with respect to an action-value function. Another possible approach is to directly learn the policy function. This can be done by representing the policy function as a parameterised mathematical function of states which outputs either a probability over possible actions, in the case of a stochastic policy, or directly the action to perform, for a deterministic policy. For these two possible cases, we write:

$$\pi(a \mid s, \theta) = Pr\{a_t = a \mid s_t = s, \theta_t = \theta\}, \tag{2.20}$$

or

$$\pi(s; \theta) = a, \tag{2.21}$$

where $\theta \in \mathbb{R}^{d^\theta}$ is the set of learnt parameters of the policy function. We also denote such policy π_θ, as the policy defined by parameters θ.

To maximise future rewards, we have to find the set of values for θ that produces the best sequences of actions. To do so, we can sample experiences from the environment and accordingly modify θ to improve the policy. This can be done using **gradient ascent**, which gradually modifies the parameters to maximise a given measure of performance $J(\theta)$:

$$\theta_{t+1} = \theta_t + \alpha \nabla J(\theta_t), \tag{2.22}$$

where $\alpha \in]0, 1]$ is a small learning rate and $\nabla J(\theta_t)$ is the gradient of the objective $J(\theta_t)$ with regards to the policy parameters θ at time step t. It is a vector that shows in which direction in the parameter space to move θ to maximise the objective. Thus, Equation 2.22 replaces the current parameters θ_t with a new set of values θ_{t+1} that yields better values of J.

The Policy Gradient Theorem

In RL, the objective to maximise is the sum of future rewards. Thus, we can define the objective using the true value function of our policy: $J(\theta) = V_{\pi_\theta}(s_0)$, with s_0 being the initial state of the episode. Given this, for a discrete MDP and a given stochastic policy π_θ, we can devise the **policy gradient theorem** that yields:

$$\nabla J(\theta) = \sum_s \rho_{\pi_\theta}(s) \sum_a Q_{\pi_\theta}(s, a) \nabla \pi_\theta(a \mid s), \tag{2.23}$$

with Q_{π_θ} the true action-value function of policy π_θ and $\rho_{\pi_\theta}(s)$ the probability of experiencing state s if we follow policy π_θ.

If policy π_θ is followed, then the theorem can be formulated with an expected value:

$$\nabla J(\theta) = \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\sum_a Q_{\pi_\theta}(s,a)\nabla\pi_\theta(a \mid s)\right] \tag{2.24}$$

$$= \mathbb{E}_{s\sim\rho_{\pi_\theta}}\left[\sum_a \pi_\theta(a \mid s)Q_{\pi_\theta}(s,a)\frac{\nabla\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}\right]$$

$$= \mathbb{E}_{s\sim\rho_{\pi_\theta},a\sim\pi_\theta}\left[Q_{\pi_\theta}(s,a)\frac{\nabla\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}\right]. \tag{2.25}$$

The expected value in Equation 2.24 is obtained by sampling $s \sim \rho_{\pi_\theta}$. Likewise, in Equation 2.25, the sum is absorbed in the expectation by sampling $a \sim \pi_\theta$. Because $Q_{\pi_\theta}(s,a) = \mathbb{E}_\pi[G_{s,a}|s,a]$, by definition, we can rewrite the policy gradient as:

$$\nabla J(\theta) = \mathbb{E}_{\pi_\theta}\left[G_{s,a}\frac{\nabla\pi_\theta(a \mid s)}{\pi_\theta(a \mid s)}\right]. \tag{2.26}$$

This expectation can be approximated by sampling many trajectories. The algorithm REINFORCE (Williams, 1992) uses this to update the parameters θ:

$$\theta_{t+1} \leftarrow \theta_t + \alpha G_t \frac{\nabla\pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}, \tag{2.27}$$

for any state s_t and action a_t from which the agent has experienced the return G_t. In this update, the gradient of the policy indicates the direction, in parameter space, that most increases the probability of choosing action a_t. By taking the product of this gradient with the return G_t, this update increases the probability of actions depending on the returns they generate. High positive returns will result in high increases in the corresponding actions, and conversely for negative returns. Then, dividing by the probability of the action ensures that actions that are selected more frequently are not over-estimated by getting more updates of the same magnitude. Therefore, the REINFORCE update conveniently increases the probability of choosing actions that yield good returns and decreases the probability of actions that yield bad ones.

The Actor-Critic Architecture

As it uses the complete return, REINFORCE is a Monte Carlo method for learning a parameterised policy function. Like the Monte Carlo approach presented in the last section, REINFORCE requires waiting for the end of the episode to update the policy. But, again, we can use the principle of TD learning (see Section 2.3.1) to solve this issue, by replacing the return by its estimation with a value function:

$$\theta_{t+1} \leftarrow \theta_t + \alpha\left(r_{t+1} + \gamma V(s_{t+1})\right)\frac{\nabla\pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}, \tag{2.28}$$

or,

$$\theta_{t+1} \leftarrow \theta_t + \alpha Q(s_t,a_t)\frac{\nabla\pi_\theta(a_t \mid s_t)}{\pi_\theta(a_t \mid s_t)}, \tag{2.29}$$

where V and Q would be learnt value functions that estimate the true value functions of the current policy π. This kind of approach is referred to as *actor-critic* methods (Barto et al., 1983; Sutton, 1984), where the policy is the actor who learns to

select action, and the value function is the critic who evaluates the policy's actions. In this case, the value function is used only to guide the learning process of the policy.

In the last section, we introduced parameterised value functions as a way to use value functions with continuous states. However, a parameterised version of Q-learning would still not be able to deal with continuous actions efficiently. That is because finding a greedy policy with respect to a continuous action-value function would be an optimisation problem in itself: finding the global maximum of a given function. But, thanks to an actor-critic architecture, a solution to this issue can be found. By extending the policy gradient theorem defined in Equation 2.24 to continuous actions and a deterministic policy, we can find that

$$\nabla J(\theta) = \mathbb{E}_{s\sim\rho_{\pi_\theta}} \left[\nabla_\theta \pi_\theta(s) \nabla_a Q_\varphi(s,a) \mid a = \pi_\theta(s)\right]. \tag{2.30}$$

This the **deterministic policy gradient theorem** (DPG), as defined by Silver et al., 2014, which gives the following update for the parameters θ:

$$\theta_{t+1} \leftarrow \theta_t + \alpha_\theta \nabla_\theta \pi_\theta(s_t) \nabla_a Q_\varphi(s_t, a_t). \tag{2.31}$$

The parameters of the action-value function φ are updated as well to minimise the Bellman error:

$$\delta_t = r_t + \gamma Q_\varphi(s_{t+1}, a_{t+1}) - Q_\varphi(s_t, a_t) \tag{2.32}$$

$$\varphi_{t+1} \leftarrow \varphi_t + \alpha_\varphi \delta_t \nabla_\varphi Q_\varphi(s_t, a_t). \tag{2.33}$$

Intuitively, DPG can be understood as an extension of Q-learning for continuous actions, where the ϵ-greedy policy is replaced by a learnt policy trained to select the actions that maximise Q_φ.

All RL approaches that derive from the policy gradient theorem are referred to as *policy-based*, even the actor-critic that also learns value functions. A first appeal to policy-based methods is intuitive: it seems logical to explicitly learn to perform actions more often if they produce better returns, and conversely. The great advantage of policy-based methods, however, is in their capacity to handle continuous state and action spaces (Williams, 1992). Parameterised value functions can deal with continuous states but, as we have seen, are still unable to handle continuous actions without being paired with a learnt policy or using discretisation methods. Thus, policy-based methods are a logical choice for more fine-grained control environments. That said, value-based still have their advantages. Depending on the setting, it might be simpler to learn a value function than a policy. In this sense, actor-critic algorithms offer a convenient integration of both. We will see in Section 2.5 that most state-of-the-art RL algorithms use this approach.

2.3.3 Planning with Models

The two previous sections presented algorithms that learn a policy, a value, or both, and rely on the learnt elements to select actions in each step. These methods are often called **model-free** because they do not use a model of the environment dynamics. That is in contrast with **model-based** methods that rely primarily on a model to learn an efficient strategy, as defined in Section 2.2. Having a model that predicts future outcomes allows to simulate experiences and use these to devise a strategy. This may involve learning value and policy functions as well. The key difference with model-free techniques lies in the exploitation of a learnt model of the environment to

improve the learning and usage of policy and value functions.

Learning the Model

Learning the model is a supervised task where the agent learns to approximate the environment dynamics using its own experienced transitions as training examples. The goal is to learn a good approximation $\widehat{p}$ to use for planning. As $\widehat{p}$ learns two distinct elements, the transition probability $\mathcal{T}$ and the reward function $\mathcal{R}$, the prediction task is often separated in two as well, with $\widehat{\mathcal{T}}$ and $\widehat{\mathcal{R}}$.

The simplest way to learn these functions is by learning a *tabular model* (Sutton, 1991). In a discrete MDP, we can store the number of occurrences of all transitions $n(s, a, s')$ and use it to output a probability for each possible next state:

$$\widehat{\mathcal{T}}(s' \mid s, a) = \frac{n(s, a, s')}{\sum_{s" \in \mathbf{S}} n(s, a, s")}. \tag{2.34}$$

Similarly, for the reward, we can store the rewards obtained in each transition $r(s, a, s')$ and use them to approximate the expected reward of each transition by computing a mean over sampled transitions:

$$\widehat{\mathcal{R}}(s, a, s') = \frac{1}{N} \sum_{k=0}^{N} r_k(s, a, s'), \tag{2.35}$$

with $N = n(s, a, s')$. This tabular solution can be effective in small discrete MDPs. However, it does not scale well to environments with many possible transitions.

To tackle this issue, another approach is to define the model as a parametric function $\widehat{p}(s, a, \psi)$ with parameters $\psi \in \mathbb{R}^{d^\psi}$. Like for value and policy functions, this parameterised function can take many different forms. Parameterised models have been defined using various machine learning algorithms like linear regression (Sutton et al., 2008), random forest (Hester & Stone, 2012), and neural networks (Narendra & Parthasarathy, 1990; Oh et al., 2015).

Using the Model

We introduced models as a tool to perform planning by simulating experience. Planning itself can serve different purposes. One approach is to use the model to learn better approximations of value and policy functions. Instead of gathering experiences in the environment, a model can be queried to simulate transitions. These simulated transitions can then be used for computing the learning objectives of any algorithm. For example, in Dyna (Sutton, 1991), a tabular model is used to perform many Q-learning updates between each environment step. This improves sample efficiency by increasing the number of updates for each call to the environment. Other learning algorithms require the environment dynamics to perform their update, like value iteration (see Section 2.3.1) or guided policy search (Levine & Koltun, 2013). In such cases, learning a model enables using these algorithms in complex environments with unknown dynamics (Abbeel et al., 2006; Levine & Abbeel, 2014).

A second approach focuses the model on the decision part of the algorithm. In this sense, planning is used to predict many different paths of states and actions, and ultimately choose the path leading to the best return. This can be done in an exhaustive manner, by simulating all possible paths until the completion of the episode. Then, the value of each observed state can be computed as a mean of all

returns obtained from this state (Tesauro & Galperin, 1996), similarly to Monte Carlo methods (see Section 2.3.1). Planning in this way is particularly effective, especially in environments where a perfect model is given like Chess or Go. But, it requires a great amount of computation to simulate all trajectories and evaluate each state and action. Doing an exhaustive search of all possible outcomes is unrealistic in environments with a large state space. In the game of Go, the number of possible states is beyond any computing limitation. To handle this, the Monte Carlo Tree Search algorithm (MCTS; Coulom, 2006) had to greatly optimise the search by parallelising the computation of different paths of actions, excluding states that are probably not good, and learning both an action-value and a policy function. Thanks to efficient planning, MCTS is the core algorithm responsible for a series of breakthroughs in playing board games, such as in the game Go considered very hard for its large count of possible board situations (Coulom, 2006; Finnsson & Björnsson, 2008; Schrittwieser et al., 2019; Silver et al., 2016).

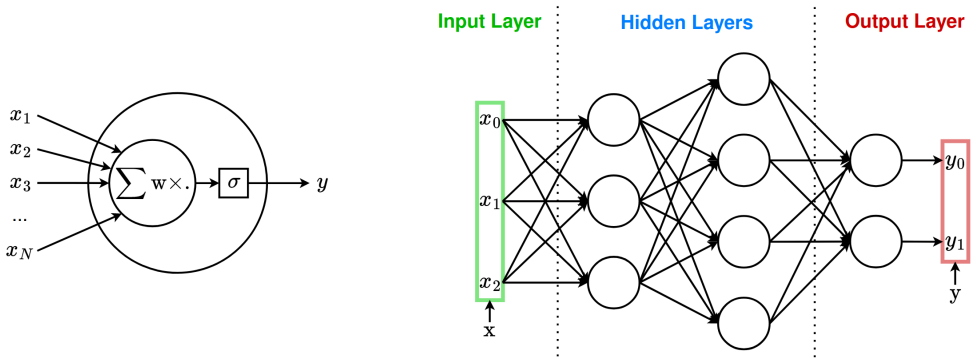
Overall, model-based methods have many advantages. Intuitively, learning the environment dynamics seems logical, as the model learns general knowledge about the environment that should generalise well to all parts of the environment, which is not necessarily the case for value and policy functions. Model-based approaches are often more sample efficient than model-free ones. By using the model's prediction to train policies and values, they can require less experience in the real environment to come up with a good strategy. However, models still have their limitations. Learning the model is not straightforward and many successful model-based approaches, like MCTS, are given a true model of the environment, which is not possible in more complex environments. Also, relying on planning for selecting actions is very computationally intensive, which makes it tough to implement in real-time applications.

2.4 Artificial Neural Networks and Deep Learning

Artificial neural networks are a core component of recent machine learning techniques. Given rather large amounts of data, they allow learning non-linear mappings between different data spaces. They can learn how to understand a given data point, thanks to the automatic analysis of many other similar data points. This enables solving many different learning challenges, such as data classification, image segmentation, and natural language generation. It also allows building more capable RL algorithms, by using neural networks as models, policies or value functions. The great capacities of neural networks have improved the potential of RL, allowing it to deal with more complex environments and learn more intricate policies. In this section, we define the components of neural networks and their learning mechanisms. We define recurrent neural networks used for emulating memory. And, we present the field of deep neural networks that are now widely used across machine learning and RL research.

2.4.1 Artificial Neural Networks

The term **artificial neural network** (ANN) designates a type of parameterised architecture that can be used to solve machine learning problems. Concretely, an ANN is a mathematical function that transforms the input vector, into an output vector of another form: $f : \mathbb{R}^N \to \mathbb{R}^M : f(x;\theta) = y$, with x and y respectively the input and output vectors of dimensions N and M, and $\theta \in \mathbb{R}^{d^\theta}$ the vector of parameters of the neural network. The name "artificial neural network" comes from the actual form of this function. It can be decomposed into many simpler functions, each one of



(a) Perceptron (b) Artificial neural network

Figure 2.5: Architecture of an artificial neural network (ANN). (a) The perceptron, i.e., the neuron-like element, transforming an input vector into a single output value. It does so with a summed vector multiplication with the neuron's weights w and by using the activation function σ that determines the activation of the neuron. (b) Example of ANN, with each circle representing a single perceptron. As there are more than one hidden layer, it is a multi-layer perceptron.

them acting as a neuron-like activation. The aggregation of multiple of these neurons in a computational graph builds the network architecture of the global function, as illustrated in Figure 2.5b.

The neuron element of this network, illustrated in Figure 2.5a, is called a **perceptron** (Rosenblatt, 1958). It is a simple function that transforms a given input vector $x \in \mathbb{R}^N$ into a single output value $h(x; w) \in \mathbb{R}$ through a weighted sum. In this operation, each element of the input vector is multiplied by a learnable parameter $w_i \in \mathbb{R}$, also called a *weight*, before being all summed together:

$$h(x; w) = \sum_{i=1}^{N} w_i x_i + b, \tag{2.36}$$

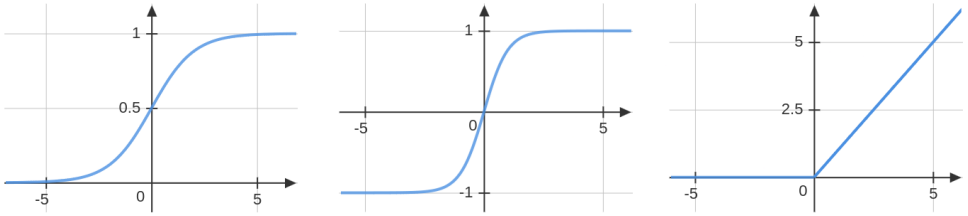
with the input vector $x = \{x_i\}_{0<i\leq N}$, the weights $w = \{w_i\}_{0<i\leq N}$, and a bias parameter $b \in \mathbb{R}$. This bias, which is also a learnable parameter, is needed to learn more complex transformations.

The result of this operation is then passed through an activation function σ that determines the final output of the neuron:

$$y = \sigma\left(h(x; w)\right) = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right). \tag{2.37}$$

This activation function is a non-linear transformation of the input. Without the non-linear activation, an aggregation of perceptrons could be rewritten as a linear transformation of the input. Thus, the purpose of σ is to allow the neural network to model non-linear functions. Three examples of frequently used activation functions are the *sigmoid* function, the hyperbolic tangent (*Tanh*), and the Rectified Linear Unit (*ReLU*; Glorot et al., 2011), all shown in Figure 2.6.

Multiple perceptrons can be used to build a neural network, like in Figure 2.5b. In the classical ANN architecture, the neurons are structured in *layers*. Each layer is composed of a fixed number of neurons, each neuron taking as input all the outputs of the previous layer. For this reason, this kind of layer is often called *fully-connected*.



$$Sigmoid(x) = \frac{1}{1+e^{-x}} \qquad Tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad ReLU(x) = \max(0,\ x)$$

Figure 2.6: Mathematical definitions and representations of three activation functions: sigmoid, hyperbolic tangent (Tanh), and rectified linear unit (ReLU).

Three distinct parts of the network are often identified: the input layer, the hidden layers, and the output layer. The input layer is made by the input vector. The output layer is the layer of neurons that will generate the output vector. Thus, both the dimensions of the input and output layers are determined by the required dimensions of the input and output vectors. On the other hand, between the input and output layers, the number of hidden layers and neurons in each one of them can be defined arbitrarily. Each neuron has its set of learnable parameters, whose dimension depends on the size of the last layer: one weight for each input, plus the bias. Thus, the parameters of the whole ANN, defined previously as θ, is the set of all parameters (weights and biases) of all neurons in the network.

Learning these parameters can be done with gradient descent. When training the network, training examples x are fed into the network, generating predicted outputs: $\widehat{y} = f(x;\theta)$. This is called the *forward propagation*. The predictions are evaluated by a loss function that measures their quality with regard to the training objective. For example, in a supervised task, the objective of the model is to predict a target value y associated with x in the training data. In this case, the loss function measures the prediction error, which can be defined as the *mean squared error*:

$$L(\widehat{y};\theta) = \frac{1}{M}\sum_{i=1}^{M}(y_i - \widehat{y}_i)^2, \tag{2.38}$$

with M the dimension of the output layer. All operations made inside the network (i.e., f) are differentiable[2]. This allows computing the gradient of the loss $L(\widehat{y};\theta)$ with regard to each parameter in the network, through a series of chain rules. Thus, the parameters can be updated to minimise the loss:

$$\theta \leftarrow \theta - \alpha\nabla_\theta L(\theta), \tag{2.39}$$

with a learning rate $\alpha \in]0,1]$. Notice the negative sign of the gradient used in gradient descent to minimise the loss, conversely to gradient ascent presented in Section 2.3.2, which was intended for maximising the RL objective. This process of going back to each parameter in the graph to compute its gradient and update it is often referred to as *backward propagation*, or back-propagation.

[2]The derivatives of some activation functions may not be defined everywhere (e.g., ReLU in 0), when that is the case, a value is arbitrarily defined for the missing derivatives (e.g., $\text{ReLU}'(0) = 0$).

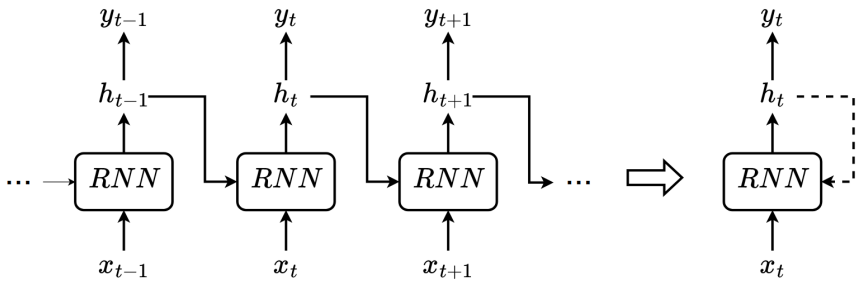


Figure 2.7: Illustration of a recurrent neural network (RNN) unit handling the sequence of inputs x. The left version shows the process unfolded in time, with the hidden state of step t passed to the next step $t+1$. On the right is shown a condensed illustration of the RNN, with the dashed arrow indicating that h_t is carried out to the next step.

2.4.2 Recurrent Neural Networks

The ANN architecture presented previously is one of many other more complex structures that specialise in particular tasks. One limitation of the ANN is that it struggles to handle sequential data, where inputs are arranged in time and can depend on previous inputs. For example, text is a form of sequential data where words and punctuation marks are arranged in a sequence. In this sequence, some words may have a strong correlation with others. In such cases, a neural network would benefit from learning these potential correlations by memorising some information from previous inputs. This is the idea behind **recurrent neural networks** (RNNs) that propose an architecture for handling sequential data, illustrated in Figure 2.7. At each time step t, the RNN unit takes two inputs: the input data point $x_t \in \mathbb{R}^N$ and the hidden state of the last step $h_{t-1} \in \mathbb{R}^M$. The hidden state is a vector that carries information between each step of the process. At each step, it is updated with:

$$h_t = \sigma\left(W_x x_t + W_h h_{t-1} + b\right), \tag{2.40}$$

with $W_x \in \mathbb{R}^{N\times M}$ and $W_h \in \mathbb{R}^{M\times M}$ two matrices of learnable parameters, a bias vector $b \in \mathbb{R}^M$, and an activation function σ, which is often a Tanh in RNNs. The output is then a function of the hidden state: $y_t = g(h_t)$. Here, function g can be a layer of perceptrons to output a vector in the required dimension. The RNN unit serves as a fundamental block that is often used inside a larger architecture.

Notice that, all the computations made from the beginning of the sequence can be traced back in the final hidden state h_T and written as a function of all inputs of the sequence: $h_T = g(x_T, h_{T-1}) = g(x_T, g(x_{T-1}, h_{T-2}))$, and so forth. Thus, as for the ANN, we can compute the gradients of a given loss function with regard to each parameter of the RNN unit. The specificity here is that the gradients are "back-propagated through time", as the loss at step T depends on calculations done in all previous steps. This allows the RNN to learn the temporal correlation in the input sequences.

While the hidden state of the RNN carries some information to the next step, in practice, this simple RNN struggles to memorise long-term information. In addition to that, it suffers from algorithmic flaws that prevent its use in many applications. However, better recurrent architectures have been developed to address these issues. The **Long Short-Term Memory** (LSTM) architecture (Hochreiter & Schmidhuber, 1997) is one example (see Figure 2.8). Inside the LSTM unit, the information flows into various *gates* that ensure the conservation or forgetting of some parts of the inputs.

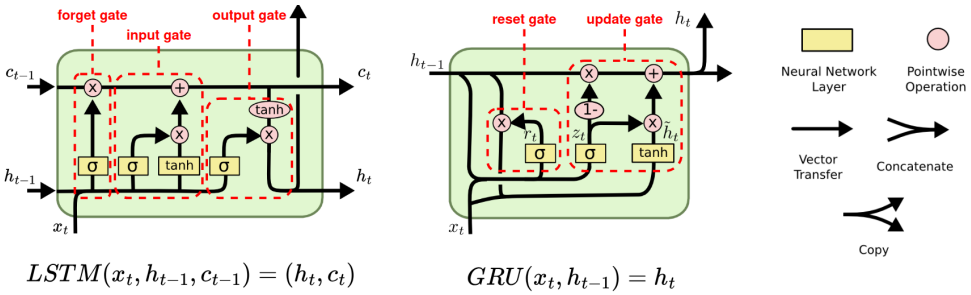


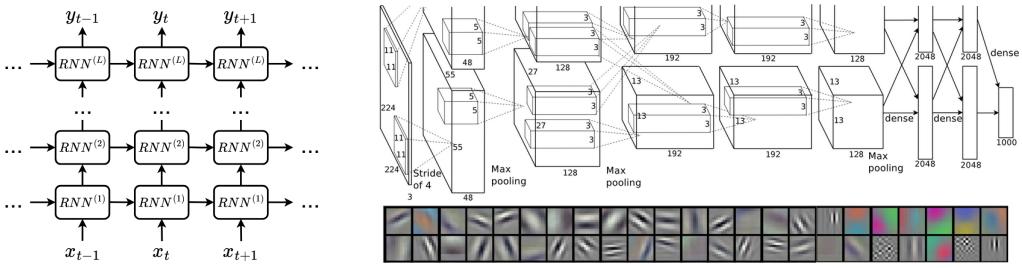
Figure 2.8: Illustration of the Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) neural network architecture[3]. Input information flows along the black arrows and passes through different "gates". These gates are neural networks that compute the outputs by updating the memory, forgetting some information and adding some new from the input x_t. In addition to the hidden state, the LSTM carries the cell state c between each step to extend the memory capacities.

In addition to the hidden state, a "cell state" is also kept between steps. Intuitively, the cell state should carry more long-term memory than the hidden state. Another similar recurrent architecture is the **Gated Recurrent Unit** (GRU; Chung et al., 2014). The GRU takes inspiration from the LSTM, but simplifies its architecture while keeping similar performance. In practice, both the LSTM and GRU are used in recent works to deal with sequential data.

2.4.3 Deep Neural Networks

The great strength of neural networks is their ability to automatically learn hidden relations in the training data. Take for example a computer vision task, where the objective is to determine the presence of a car in images. Classical machine learning algorithms usually require high-level features to solve such tasks: the presence of wheels in the picture, the number of such wheels, the presence of reflective material, and other common attributes of cars. With neural networks, such high-level features can be extracted automatically by looking at the training data. This is called *representation learning*, where the neural network learns a set of weights (as described in Section 2.4.1) that produces accurate internal representations of the input data. Through careful training, the network learns how to understand the information given in the picture, to give an accurate prediction of the presence of a car. Because this is done automatically, this relieves the otherwise painful work of manually extracting and annotating important features in all training samples. This also allows the neural network to discover hidden features that could have been omitted in human-generated data (e.g., the fact that cars are often pictured on a road, so the presence of road signs might increase the probability of having a car in the image). However, there are also downsides to this. First, this automatic process of learning representations is done in a black box that is out of the control of human observers. The representation capacities of the neural network are embedded in the weights of the networks and are hard to understand and interpret. Second, these representation capacities still have a cost, now measured in the amount of training data at our disposal. To learn the complex relations in a very large space of input values and obtain good performance, neural networks usually require very large datasets of standardised data. Still, obtaining these datasets is often less difficult than manually extracting valuable features by hand.

[3]Illustrations adapted from: http://colah.github.io/posts/2015-08-Understanding-LSTMs/.



(a) Deep RNN

(b) Deep CNN

Figure 2.9: Two examples of deep neural network architectures. (a) A deep RNN, where the input x_t passes through L RNN units to compute the output y_t, with each unit passing its hidden state $h_t^{(i)}$ to the next layer $i+1$ and to the next step $t+1$. (b) Architecture of a deep convolutional network on top, and the bottom shows the learnt filters of the first convolutional layer that detect basic shapes in the images (Simonyan & Zisserman, 2015).

It has been widely recognised that the number of layers in a neural network can greatly influence its performance in solving the task. In many cases, adding hidden layers can improve the representations learnt by the network. This can be understood intuitively by looking at each layer as a representation unit, where the first layer extracts low-level information, and subsequent layers each use the representation given by the last layer to extract more high-level features. This has led to the use of *deep neural networks* that have more than one hidden layer, as in Figure 2.5b. These networks, often called **Multi-Layer Perceptrons** (MLPs), are the fundamental block of what is now called *deep learning*, corresponding to any machine learning architecture that features deep neural networks.

The MLP is the most basic form of deep neural architecture. It is often used to handle any input that can be represented as a vector. But, other architectures have been developed for specific types of inputs. For sequential data, the RNNs can be extended to have a deep architecture, with many recurrent units stacked on top of each other, as in Figure 2.9a. For images, the **Convolutional Neural Networks** (CNNs; Krizhevsky et al., 2012; LeCun et al., 1989; Simonyan and Zisserman, 2015) learn to detect complex patterns of pixel values that can then be used to recognise objects in images (see Figure 2.9b). Recently, the Transformer architecture (Vaswani et al., 2017) have been introduced to handle sequential data, with better long-term memory than recurrent architectures. Transformers have been widely adopted in the machine learning community as a foundation for building better models. Extremely large Transformers with billions of learnable parameters have demonstrated superior performance in text generation (Ouyang et al., 2022) and computer vision (Dosovitskiy et al., 2021).

Overall, the impact of deep neural networks on machine learning has been tremendous. In the next section, we will see that deep learning is now a common technique in reinforcement learning research. But it also comes with its share of complications. First, training deep neural networks requires large amounts of data to train on. A larger neural network usually needs more data to train efficiently (Kaplan et al., 2020) and a common trend of greatly increasing model sizes to improve performance has been observed (Villalobos et al., 2022). Training larger models requires considerable computational power and access to computers equipped with Graphics Processing Units (GPUs) that can efficiently compute large matrix operations. This makes deep learning expensive and not accessible to anyone with a restricted budget. Even executing

a trained model requires some computational capacities, which makes deployment on small platforms (e.g., robots) complicated. Lastly, deep neural networks are often complex and depend on many design choices to function well. They usually come with a long list of parameters, often called *hyperparameters* to distinguish them from learnable parameters, that define their architecture and training setting: the number of layers, the dimension of the hidden layers, the learning rate, etc. Each one of these hyperparameters is a value that can impact the performance of the model and that needs to be tuned. But, despite these obstacles, deep learning is now widely used in artificial intelligence research and its applications, and is now a common building block of all state-of-the-art approaches in domains like computer vision and natural language processing.

2.5 Model-Free Deep Reinforcement Learning

As in other branches of machine learning, deep learning has been a revolutionary technique for RL. It has enabled extending existing RL algorithms to operate in much more complex settings and has facilitated the development of new approaches for learning models, policies, and value functions. In this section, we will present the foundational works in model-free deep RL published in the last decade. These works form the basis for the majority of studies in both single-agent and multi-agent RL. While there is also significant research on model-based deep RL methods (Moerland et al., 2023), we will not cover them here as they are not typically used in multi-agent settings and are thus outside the scope of this thesis.

2.5.1 Value-Based Methods

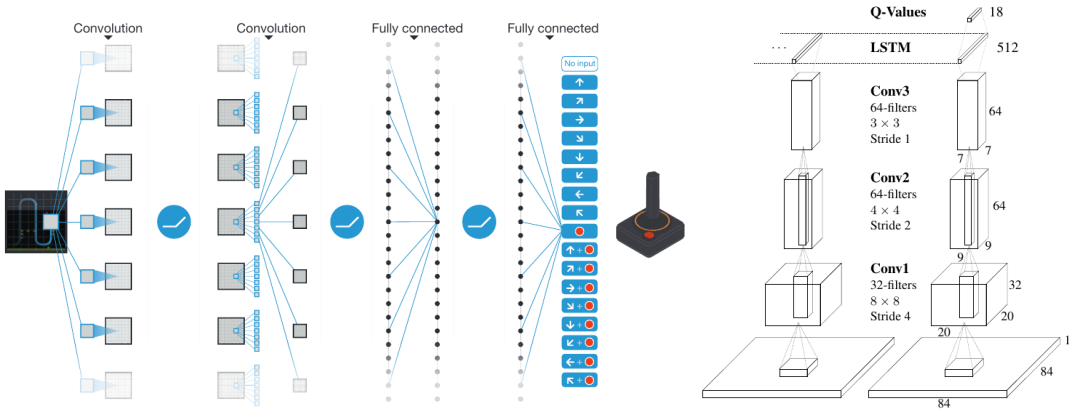
Deep Q-Learning

In Section 2.3.1, we presented the Q-learning algorithm for learning a control policy using only an action-value function. We introduced the main limitations of this algorithm: scalability and generalisation. First, tabular Q-learning scales poorly to environments with large state-action spaces, as it needs to keep track of the learnt values of all state-action pairs. Second, Q-learning struggles to generalise its training knowledge to make good predictions in situations it has never experienced. It needs a thorough exploration of the environment to ensure that its predicted values are relevant. For these reasons, Q-learning is difficult to use in realistic settings where the number of possible states is large or infinite.

These issues can be alleviated using a deep neural network in place of the value table. The **Deep Q-Network** (DQN), proposed by Mnih et al., 2013, successfully adapts Q-learning to learn from visual data. To do so, it models the action-value as a CNN (see Section 2.4.3) that takes as input frames from a video game and outputs the values of possible actions, as illustrated in Figure 2.10a. Note that, while this first iteration of the DQN architecture uses a CNN, the term DQN is now used to describe any deep neural network architecture that computes action-values.

The DQN is learnt completely end-to-end from the Q-learning objective. Concretely, the parameters of the network θ are optimised to minimise the following squared Bellman error:

$$L(\theta) = \left(r_{t+1} + \gamma \max_a Q_{\theta^-}(s_{t+1}, a) - Q_\theta(s_t, a_t)\right)^2, \tag{2.41}$$



(a) Deep Q-Network (b) Deep Recurrent Q-Network

Figure 2.10: DQN architectures. (a) The original DQN (Mnih et al., 2015), with a series of convolution layers that analyse the input frame, followed by fully connected layers that output the value of each action. (b) Recurrent version of a DQN (Hausknecht & Stone, 2015), that adds an LSTM unit after the convolutions to allow some information to carry to future steps.

for any transition $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$ and with the discount factor γ. Notice the TD-target is computed using another network Q_{θ^-}, named the *target network*. The parameters θ^- are initialised as a copy of the main parameters θ. Then, the target is fixed for a pre-defined number of training iterations, after which it is copied from the main network again, and so on. Using a slightly older, fixed version of Q_θ helps to stabilise training by giving a fixed target to aim for when minimising the Bellman error, instead of having it modified after each training iteration (Mnih et al., 2015).

Another important feature of the DQN is its use of experience replay (Lin, 1992). Because deep Q-learning is off-policy, it can capitalise on all previously acquired experiences to train its deep neural network. The deep Q-learning algorithm thus alternates between acquiring experiences by interacting with the environment (typically executing the ϵ-greedy policy), storing each experience in the replay buffer $\mathcal{B}$, and later training the DQN on a batch of samples randomly drawn from $\mathcal{B}$. Therefore, the DQN is trained to minimise the Mean Squared Bellman Error (MSBE) on the full batch:

$$L(\theta) = \mathbb{E}_{\langle s,a,r,s' \rangle \sim \mathcal{B}} \left[\left(r + \gamma \max_{a'} Q_{\theta^-}(s', a') - Q_\theta(s, a) \right)^2 \right]. \tag{2.42}$$

This batched loss is crucial for ensuring efficient training of a deep neural network. Plus, experience replay allows reusing each gathered experience multiple times and having more diverse batches, thus improving sample efficiency and avoiding overfitting the current policy.

DQN was the first successful instance of coupling RL with deep learning. It showed that deep RL was able to surpass human performance in video games, by learning only from visual data (Mnih et al., 2013). It sparked a wave of research focused on using deep learning in existing RL algorithms and developing new learning methods specific to deep RL.

DQN Extensions

Following the introduction of the DQN, many works focused on improving this architecture using both old and new techniques. In their paper, Hessel et al. (2018)

summarise the most important extensions and combine them into their method called *Rainbow*. Some of these improvements concern the core functioning of the Q-learning algorithm:

- *Double Q-learning* (van Hasselt, 2010) modifies Equation 2.42 to decouple the selection of the action to its evaluation when computing the Bellman error:

$$L(\theta) = \left(r_{t+1} + \gamma Q_{\theta^-}(s_{t+1}, \arg\max_{a'} Q_\theta(s_{t+1}, a')) - Q_\theta(s_t, a_t)\right)^2. \tag{2.43}$$

  In the TD-target, the action is selected by the main network Q_θ and evaluated by the target network Q_{θ^-}. van Hasselt et al. (2016) apply this idea to the DQN to improve its performance.

- The *Dueling DQN* (Wang, Schaul, et al., 2016) proposes to decompose the action-value of an action $Q_(s, a)$ into two separate predictions: the value of the current state $V(s)$ and the *advantage* of the action defined as $A(s, a) = Q(s, a) - V(s)$. They use a neural network architecture with two outputs, the state-value and advantage, which are then combined into the action-value. This allows them to learn a state-value and an action-value function in a single learning process with a deep Q-learning algorithm.

- *Distributional value learning* (Bellemare et al., 2017) proposes to learn a probability distribution over possible future returns, instead of learning to estimate the expected return of an action. Doing this allows to better account for multimodality in value distributions and generally improves learning stability.

Other techniques focus on algorithmic and architectural improvements to DQN:

- *Prioritised experience replay*, proposed by Schaul et al. (2016), improves on experience replay by sampling experiences more often if they lead to bad value predictions (i.e., high Bellman errors) during training. This allows prioritising samples that the network fails to evaluate well, thus greatly improving sample efficiency.

- *Noisy DQN* (Fortunato et al., 2018) tackles the problem of exploration by adding noise to the parameters of the DQN. Note that Plappert et al. (2018) concurrently proposed a similar approach working with other deep RL algorithms.

Rainbow shows that combining all these extensions greatly improves the performance and sample efficiency of DQN in a large variety of video games (Hessel et al., 2018).

Deep Recurrent Q-Network

A remaining limitation of DQN is its lack of memory of previous events. This becomes a problem in partially observable environments where the agent does not get the full state of the environment, but rather a local observation describing its surroundings. In such settings, a memory-equipped agent could explore its surroundings to build a better understanding of the current state of the environment. To achieve this, Hausknecht and Stone (2015) augment the DQN architecture with an LSTM unit (see Section 2.4.2) to learn to memorise important information for future steps (see Figure 2.10b). They show that this makes agents more robust to losses in their observations. This approach has been combined with distributed training by Kapturowski et al. (2019), largely surpassing previous works on some Atari games. As we will see in Chapter 3, using recurrent units has been widely adopted in multi-agent deep RL to deal with partial information given to each agent.

2.5.2 Policy-Based Methods

Deep Deterministic Policy Gradient

In Section 2.3.2, we introduced the actor-critic architecture that combines a policy (i.e., actor) and a value (i.e., critic) to open up the possibilities of RL training. Following the introduction of the DQN, similar ideas were implemented to extend actor-critics with deep learning. The **Deep Deterministic Policy Gradient** (DDPG; Lillicrap et al., 2015) was proposed to extend the DQN to work with continuous actions. Like in the original DPG (see Section 2.3.2), a deterministic policy function π_θ is learnt to select actions in a continuous space and an action-value function Q_ϕ is used to predict the quality of the selected actions. Here, both of these are modelled using deep neural networks. As in Double Q-learning (see Section 2.5.1), both the policy and value functions are doubled with target functions π_{θ^-} and Q_{ϕ^-}, respectively. The targets are used to compute the TD-target for computing the MSBE loss:

$$L(\Theta) = \mathbb{E}_{\langle s,a,r,s'\rangle\sim\mathcal{B}}\left[\left(r + \gamma Q_{\phi^-}(s', \pi_{\theta^-}(s')) - Q_\phi(s,a)\right)^2\right]. \tag{2.44}$$

The target parameters $\Theta^- = \{\theta^-, \phi^-\}$ are initialised as copies of the main parameters Θ and later slowly moved towards the main networks with a "soft" update: $\Theta^- \leftarrow \tau\Theta^- + (1-\tau)\Theta$, with $0 < \tau \ll 1$. Using this approach, combined with experience replay and other small tricks, DDPG managed to greatly improve on the non-deep DPG.

Fujimoto et al. (2018) pointed out that DDPG, as all Q-learning methods, suffers from an overestimation of action-values induced by using a bad value estimate for computing the optimisation objective. While Double Q-learning helps address this issue, it does not completely solve it. They propose an extension of DDPG, named *Twin Delayed DDPG* (TD3), that better mitigates the overestimation problem with three tricks:

- First (*Twin*), TD3 learns two action-value functions instead of one and uses the smaller values of the two when computing the TD-target.
- Second (*Delayed*), the policy function is trained less often than the action-value functions (one policy update for every two value updates) to make the policy less prone to value errors.
- Third, the value estimates are smoothed by adding a small clipped noise on the target policy actions when computing the TD-target. This prevents having high spikes of value on some particular actions that the deterministic policy would then overfit to.

Therefore, TD3 computes the TD-target by taking the minimum value given by the twin DQNs and adding the clipped noise to the policy's action:

$$y(r, s') = r + \gamma \min_{i=1,2} Q_{\phi_i^-}(s', \pi_{\theta^-}(s') + \epsilon), \tag{2.45}$$

$$\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c), \tag{2.46}$$

for some transition $\langle s, a, r, s'\rangle$, with hyperparameters σ and c controlling the width of the smoothing noise. The twin DQNs are both trained to minimise their MSBE:

$$L(\phi_i) = \mathbb{E}_{\langle s,a,r,s'\rangle\sim\mathcal{B}}\left[\left(y(r,s') - Q_{\phi_i}(s,a)\right)^2\right], \text{ with } i \in \{1,2\}. \tag{2.47}$$

These modifications are shown to improve the stability and performance of DDPG.

Trust Region Policy Updates

In Section 2.3.2, we presented the policy gradient method used in most policy-based algorithms, which aims at maximising the following objective for learning a stochastic policy:

$$L^{PG}(\theta) = \mathbb{E}_{\pi}\left[\log \pi_\theta(a|s) Q_{\pi_\theta}(s,a)\right].^{4} \tag{2.48}$$

This objective has two important limitations. First, the size of the learning step is difficult to define. In the context of learning a policy that generates the data it is trained on, a too big learning step could be catastrophic if it engenders a bad policy. In supervised learning, a bad learning step can be recovered later because the training data is still good. Here, a bad policy would change the distribution of samples and could then be very hard to recover from. Second, this objective theoretically limits the algorithm to use each gathered experience only one time (Schulman et al., 2017 shows doing multiple updates damages performance), which makes policy gradient algorithms less sample efficient.

Schulman et al. (2015) propose to modify the policy gradient objective to tackle these issues. The objective is replaced by the following surrogate objective:

$$L^{surr}(\theta) = \mathbb{E}_{\pi_\theta}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} A_{\pi_\theta}(s,a)\right], \tag{2.49}$$

where $\pi_{\theta_{old}}$ is another policy and A_{π_θ} is the advantage function defined as $A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s)$ (Baird, 1995). Using the advantage is equivalent to the action-value but reduces the variance of the value estimate. This objective is shown to be a good approximation of the policy gradient one, only if $\pi_{\theta_{old}}$ is close to π_θ. Thus, for this to work, a constraint must be added to ensure that π_θ does not diverge too much from the old version:

$$\mathbb{E}_{\pi_\theta}\left[KL(\pi_{\theta_{old}}(\cdot|s), \pi_\theta(\cdot|s))\right] \leq \delta, \tag{2.50}$$

with KL measuring the Kullback-Leibler divergence between the two distributions and δ a hyperparameter for controlling the amount of allowed divergence. Maximising L^{surr} while respecting the constraint of (2.50) ensures that the update stays in a "trust region". The resulting **Trust Region Policy Optimisation** (TRPO) algorithm can do multiple updates on one sampled experience, with $\pi_{\theta_{old}}$ being the policy used when gathering the experience, to improve π_θ as much as possible for each interaction with the environment.

While the approach TRPO has great advantages, its implementation is difficult, mainly because of the constraint on KL divergence second-order optimisation. To simplify this, **Proximal Policy Optimisation** (PPO; Schulman et al., 2017) removes the constraint and instead clips the objective to limit the magnitude of the updates. The new, clipped objective is defined as follows:

$$L^{clip}(\theta) = \mathbb{E}_{\pi_\theta}\left[\min(r(\theta)A_{\pi_\theta}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A_{\pi_\theta}(s,a)\right], \tag{2.51}$$

with $r(\theta)$ the probability ratio from TRPO, $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)}$, and ϵ a hyperparameter typically close to 0. This objective first clips the ratio to be close to 1 to ensure that the new policy does not diverge too much from the old one. Then, they take the minimum of the clipped and unclipped objectives to make a lower bound of the unclipped objective. This objective is simpler to compute and optimise, making PPO

[4] If we compute the gradient of this objective, we find the formulation of Equation 2.25 as, by chain rule, $\nabla_x \log f(x) = \frac{\nabla_x f(x)}{f(x)}$.

simpler to implement. The full objective maximised by PPO is the following:

$$L^{PPO}(\theta, \phi) = \mathbb{E}_{\pi_\theta}\left[L^{clip}(\theta) - c_1 L^{VF}(\phi) + c_2 H(\pi_\theta)\right] \tag{2.52}$$

where c_1 and c_2 are coefficients for the two secondary losses. L^{VF} is the MSBE loss for learning the value function used to compute the advantage in (2.51), using a technique called Generalised Advantage Estimation (Schulman et al., 2016). The third term uses the entropy of the policy, defined as $H(\pi) = \mathbb{E}_\pi[-\log \pi(a|s)]$, to induce exploration. Maximising entropy means increasing the randomness of the policy. This bonus ensures random exploration of new policies. The main objective L^{clip} evaluates each modification with regard to expected future returns.

PPO is considered one of, if not the best existing deep RL algorithm. It has been deeply studied and improved in various ways since its first release. Many implementation tricks have been identified to be crucial to the performance of the algorithm (Engstrom et al., 2020; Henderson et al., 2018).

Soft Actor-Critic

Finally, we present the Soft Actor-Critic (SAC) algorithm (Haarnoja et al., 2018) that tackles the poor sample efficiency of on-policy algorithms and the brittleness of off-policy algorithms with regard to their hyperparameters. To do so, it builds an off-policy actor-critic algorithm with a stochastic policy and a "soft" action-value function. The policy is trained on samples drawn from the replay memory $\mathcal{B}$ to maximise the expected future rewards and an entropy bonus for exploration, as in PPO:

$$L^{\pi}(\theta) = \mathbb{E}_{s\sim\mathcal{B}, a\sim\pi_\theta}\left[Q_\phi(s,a) + \alpha H(\pi_\theta)\right], \tag{2.53}$$

with *alpha* controlling the importance of the entropy bonus. As in DDPG, the action is sampled from the policy and evaluated by the value function. Maximising this objective operates a trade-off between generating actions with high action-values and inducing diverse behaviours.

To learn the action-value function, the authors take inspiration from soft Q-learning (Haarnoja et al., 2017), where entropy maximisation is included in the value learning algorithm. The soft action-value is trained to minimise the soft MSBE for any transition drawn from the buffer:

$$L^{Q}(\phi) = \mathbb{E}_{\langle s,a,r,s'\rangle\sim\mathcal{B}}\left[\frac{1}{2}\big(Q_\phi(s_t,a_t) - y^{soft}(r,s')\big)^2\right], \tag{2.54}$$

where

$$y^{soft}(r,s') = r + \gamma \mathbb{E}_{a'\sim\pi_\theta}[Q_\phi(s',a') - \alpha \log \pi_\theta(a'|s')]. \tag{2.55}$$

The target in this objective also has a bonus on the entropy of the policy, which makes the action-value function favour actions where the policy is uncertain. They show that this use of entropy maximisation with a stochastic policy greatly improves the stability and robustness of their algorithm.

2.6 Conclusion

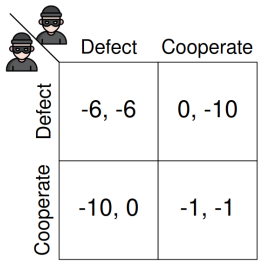
In this chapter, we have presented the field of reinforcement learning for training a single agent to solve tasks from its own experience. Understanding the different techniques behind reinforcement learning algorithms is important to understand the

challenges we will face later: *how having multiple agents will impact learning* and *how we can use and shape these learning algorithms to fit the requirements of robotic settings*. In the next chapters, we will investigate these issues and propose new approaches for tackling them.

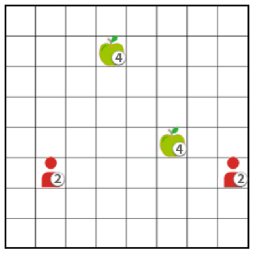
Chapter 3

Multi-Agent Deep Reinforcement Learning in the context of Robotics

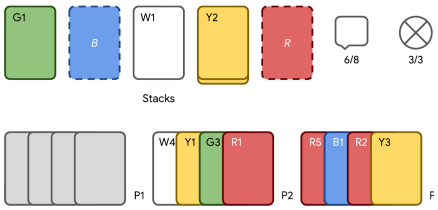
Bridging the gap between MADRL research and robotics



(a) Social dilemma

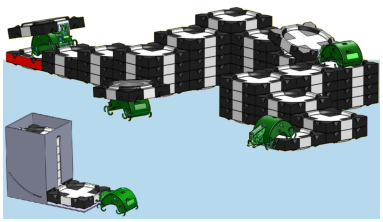


(b) Two-dimensional grid world

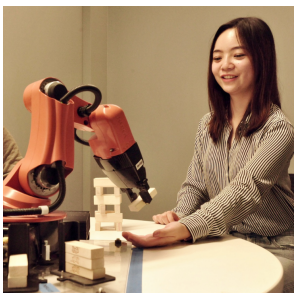


(c) Card game

(d) Multi-player video game



(e) Multi-robot system



(f) Human-robot interaction

Figure 3.1: Types of multi-agent environments. (a) Social dilemmas, here the famous *prisoners' dilemma*, extensively used in the game theory literature to devise multi-agent learning concepts and exact solutions. (b) Two-dimensional grid environments (here, level-based foraging; Albrecht and Ramamoorthy, 2013) that allow studying various multi-agent tasks in simplified settings. (c) Card and tabletop games (here, Hanabi; Bard et al., 2020) that often require learning complex strategies. (d) Multi-player video games (here, Dota 2, tackled by OpenAI et al., 2019) that offer rich environments demanding complex team-play. (e) Multi-robot systems (here, the TERMES construction robots; Petersen et al., 2012), having to deal with high-dimensional robotic settings. (f) Human-robot interaction, with the need to build interfaces and adapt to human partners (Jung et al., 2020).

3.1 Introduction

In most realistic environments, multiple entities interact with each other to fulfil an individual or a collective goal. With multiple entities, each with their personal reasoning, the outcome of one's actions also depends on the others' actions, thus increasing the difficulty of learning how to best behave. If entities are all capable of adapting and learning, then the environment becomes an ever-evolving sum of intersecting strategies. The problem of finding the optimal strategy becomes even more complex, as the best response to previously observed behaviours might not be true in future tries. The single-agent learning setting, used throughout Chapter 2, does not explicitly model these new problems and, thus, falls short in most of these cases. For this reason, the concept of multi-agent system (MAS) has been defined to better describe the dynamics of environments containing multiple intelligent entities, as found in human societies (Bousquet & Le Page, 2004; Doran & Palmer, 1995; Hamill & Gilbert, 2015), games (Nowé et al., 2012; Owen, 2013), or robotics (Parker et al., 2016; Rizk et al., 2019) (see Figure 3.1 for example of multi-agent environments). Multi-agent learning specifically tackles how learning takes place in MASs, how it can be harmed by having multiple agents, and how it can benefit from it. Multi-agent learning research comes from the intersection of many different views in software engineering (Ben-Ari, 2006), distributed artificial intelligence (Stone & Veloso, 2000), and game theory (Leyton-Brown & Shoham, 2008; Rosenschein & Zlotkin, 1994); each one having proposed ways of modelling multi-agent interactions (Wooldridge, 2009). One possible way is to adapt and extend single-agent RL tools to fit the needs of MASs. Recent years have seen the development of multi-agent deep reinforcement learning (MADRL) algorithms, with a wide range of new approaches for tackling richer multi-agent environments.

Many robotic applications involve interactions between multiple robots and/or humans (Parker et al., 2016). In this context, MADRL is a potentially valuable tool for learning complex multi-agent behaviours in realistic environments (Orr & Dutta, 2023). But, to progress towards this objective, we need to ask ourselves: *What does it mean to have robots operating in the real world? What should be the requirements when building intelligent robots and how should this impact the design of learning algorithms?* This thesis intends to provide answers to these questions by studying existing approaches and proposing new solutions for improving cooperative MADRL in the context of robotics. In this particular regard, we are faced with several observations about the related literature:

(1) Learning with multiple agents implies multiple theoretical and technical issues that are often treated separately or even ignored.

(2) Similarly, robotic domains establish several constraints and challenges that are not thoroughly investigated in MADRL research, despite a prevalent aspiration of applying these algorithms to robotic settings.

(3) The multi-agent (deep) reinforcement learning literature is extremely rich, with many orthogonal subjects of interest. This results in a domain that is hard to grasp and fully understand, with many works that are difficult to compare and evaluate.

This chapter aims to propose a clarified view of the literature, providing insights and avenues for reflection on these three starting observations. In Section 3.2, we start by formally defining important concepts and mathematical tools used in multi-agent RL.

In Section 3.3, we introduce points (1) and (2), defining the challenges met in multi-agent learning and robotic settings, and looking at how they can overlap. To answer point (3), in Section 3.4, we present a short review of recent works in the MADRL literature. Finally, in Section 3.5, we provide a personal reflection on the remaining shortfalls of MADRL research, specifically when dealing with robotic applications. We aim to highlight the major flaws of this line of work and identify potential areas of improvement.

3.2 Multi-Agent Learning: Definitions

Multi-agent learning studies how learning can take place in environments where there are two or more intelligent agents. There is no theoretical limit to the number of agents in a MAS, but studies in this field are often limited to rather small groups of agents (say, from 2 to around 20), with larger groups being the subject of swarm robotics (Hamann, 2018). With a limited number of agents, multi-agent learning research can study more complex agent definitions and more intricate social dynamics.

Agents in a MAS are close to the RL agents defined in Section 2.2. But, because they are now in a group, their definition may be completed accordingly. In addition to the three characteristics of *reactiveness*, *proactiveness*, and *ability to learn*, defined in Section 2.2, an intelligent agent is now also a **social** entity. This translates into having interactive abilities, which can be either *explicit abilities*: e.g., communication (see Sections 3.2.3 and 3.4.5), prediction of multi-agent outcomes (see Section 3.2.4), or agent modelling (see Section 3.4.6); or *implicit abilities*: e.g., cooperation, coordination, attack/defence, negotiation, or bluffing. Taking this social aspect into consideration will influence the design of multi-agent learning algorithms.

3.2.1 Learning Framework: Dec-POMDP

Because we want to study cooperation in robotic-like environments, we use the decentralized partially-observable Markov decision process (Dec-POMDP) (Oliehoek & Amato, 2016) as our learning framework. The Dec-POMDP is an extension of the MDP (as defined in Section 2.2.1) that allows the presence of multiple intelligent agents and models the fact that the environment is not fully observable: agents only observe a part of it through their sensors. Formally, it is defined as a tuple $\langle \mathbf{S}, \mathbf{A}, \mathcal{T}, \mathbf{O}, \mathcal{O}, \mathcal{R}, n, \gamma \rangle$ in which:

- n is the number of agents;
- $\mathbf{S}$ is the set of all possible states of the environment, often referred to as "global states" as they describe the environment entirely;
- $\mathbf{O}$ is the set of joint observations, with one joint observation $\mathbf{o} = \{o^1, ..., o^n\} \in \mathbf{O}$ being a set of local observations (one for each agent);
- $\mathbf{A}$ is the set of joint actions, with one joint action $\mathbf{a} = \{a^1, ..., a^n\} \in \mathbf{A}$ being a set of local actions (one for each agent);
- $\mathcal{T}$ is the transition function defining the probability $P(s'|s, \mathbf{a})$ to transition from state s to next state s' with the joint action $\mathbf{a}$;
- $\mathcal{O}$ is the observation function defining the probability $P(\mathbf{o}|\mathbf{a}, s')$ to observe the joint observation $\mathbf{o}$ after taking joint action $\mathbf{a}$ and ending up in s';
- $\mathcal{R} : \mathbf{S} \times \mathbf{A} \to \mathbb{R}$ is the reward function producing a single reward for all agents at each time steps;

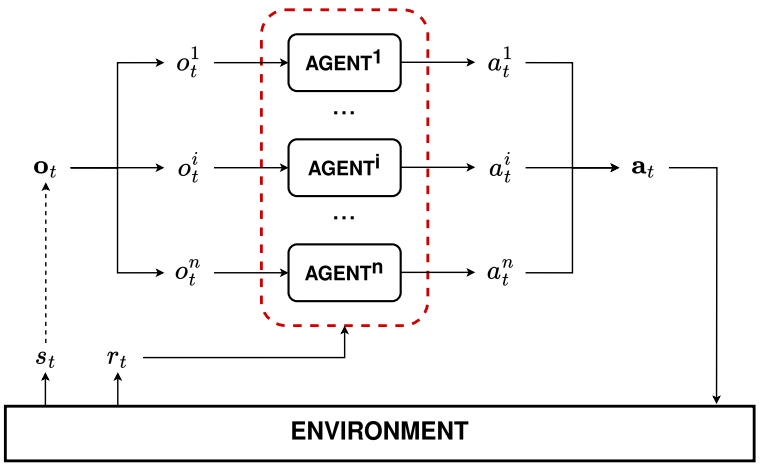


Figure 3.2: Diagram illustrating the Dec-POMDP framework. At each step t, the environment is in a state s_t that may be unknown by the agents. The joint observation $\mathbf{o}_t$ is produced by the environment, containing one local observation o_t^i for each agent i. Each agent produces an action a_t^i, all actions being gathered in the joint action $\mathbf{a}_t$ that is to be executed in the environment. A single reward r_t is produced and shared by all agents for evaluating step $t-1$. The red dotted line symbolises the multi-agent learning algorithm that can use the reward in various ways to train the agents.

- $\gamma \in [0, 1)$ is the discount factor controlling the importance of immediate rewards against future gains.

Figure 3.2 illustrates a time step in the Dec-POMDP framework. An important thing modelled in this framework is that agents may not have access to the global state of the environment s_t (it is often even not defined). They only observe a sub-part of this global state through their sensors, which is represented by the local observations o_t^i. In a robotic environment, the local observation will be the result of the robot's sensors (e.g., camera, lidar, etc.). In simulation, the content of the observations is typically defined arbitrarily by deciding what an agent should be able to observe from the environment (e.g., its position, the relative positions of other agents, etc.). The joint observation $\mathbf{o}_t$, denoted in bold, is the concatenation of all local observations at step t (and similarly for the joint action $\mathbf{a}_t$).

3.2.2 Multi-Agent Reinforcement Learning Tools

The Dec-POMDP framework allows the development of RL agents, with the basic RL tools adapted to take into consideration the multi-agent context. With multiple agents, the goal of the learning algorithm is to find the optimal **joint policy** $\pi = (\pi^1, ..., \pi^n)$, with one **local policy** π_i for each agent i, that maximises expected future returns. In the Dec-POMDP, local policies are conditioned on the action-observation history $h_t^i = (o_0^i, a_0^i, ..., a_{t-1}^i, o_t^i) \in \mathbf{H} = (\mathbf{O} \times \mathbf{A})^*$ that contains all previous local observations and actions in the current episode. Thus, we have one local policies $\pi^i(a^i|h^i) : \mathbf{H} \times \mathbf{A} \to [0, 1]$ for each agent. The **joint value functions**, related to the joint policy can be written as:

$$V_\pi(\mathbf{h}_t) \coloneqq \mathbb{E}_\pi[G_t \mid \mathbf{h}_t] \text{ and } Q_\pi(\mathbf{h}_t, \mathbf{a}_t) \coloneqq \mathbb{E}_\pi[G_t \mid \mathbf{h}_t, \mathbf{a}_t]; \tag{3.1}$$

and similarly for the **local value functions** related to the local policy π_i:

$$V_{\pi^i}(h_t^i) \coloneqq \mathbb{E}_{\pi^i}[G_t \mid h_t^i] \text{ and } Q_{\pi^i}(h_t^i, a_t^i) \coloneqq \mathbb{E}_{\pi^i}[G_t \mid h_t^i, a_t^i], \tag{3.2}$$

with the discounted return G_t defined as in RL (see Equation 2.6) and $\mathbf{h}_t = \{h_t^1, ..., h_t^n\}$ the joint action-observation history. We will also denote the local policy of any agent i as π_{θ_i}, with θ_i the learnt parameters of agent i. Similarly, the joint policy can be written π_θ, with the whole set of parameters $\theta = (\theta_1, ..., \theta_n)$.

We will see that MADRL algorithms usually do not explicitly use the action-observation history to condition the local policies. Some simply do not use it and condition only on the local observation. Others implement some kind of memory in the agent's policy (e.g., a recurrent neural network) to implicitly save some information about past steps in the agent's reasoning.

3.2.3 Communication

Multi-agent interactions may benefit from the ability to communicate information to other agents. Communicating efficiently requires knowing what information should be shared and how to share this information to be understood correctly. Learning these skills is a challenge in itself that has been thoroughly studied in dedicated lines of work (Austin, 1975; Brighton et al., 2005; Farrell & Rabin, 1996; Galke et al., 2022) and in the context of MADRL, as we will see in Section 3.4.5 and more deeply in Chapter 5. In a MAS, communication can take multiple forms. Agents might develop implicit communication abilities, using physical actions to convey abstract information (e.g., pointing a finger towards an object). Agents might use a shared archive, similar to a blackboard in a room, to store information accessible to everyone. Or, they might be provided a collection of mechanisms for exchanging messages between agents. We focus on such explicit, message-based communication to describe how agents can learn to communicate.

Communication mechanisms can be defined in many different ways depending on when agents communicate, how they share information, and to whom they are allowed to communicate. Here, we define a formal framework for communication to describe its use in MADRL algorithms. Note that this framework might not fit all communication architectures perfectly, but it can be adapted if needed.

We define communication as taking place during the action-selection process of the agents. After receiving their local observations, agents take part in a *communication turn* where they can generate a message $m_t^i = f_{comm}^i(h_t^i, \theta_i)$ and send it to the other agents. We consider f_{comm}^i as a learnt module of agent i, using a subset of θ_i (e.g., a dedicated neural network) to generate the message. The message can be sent to either all other agents, referred to as *broadcasting*, or to a limited subset, e.g., within a certain range or choosing particular agents to target. Agents then receive incoming messages and use them according to their architecture. Some methods might allow repeating such communication turn multiple times to allow some form of discussion. Then, the incoming information is used to compute the generated action: $a_t^i \sim \pi_i(.|h_t^i, m_t)$, where m_t refers to the incoming messages.

3.2.4 Nash Equilibrium

An important concept from game theory is the Nash equilibrium, introduced by Nash, 1950. It represents a stable joint policy state where no player can gain an advantage by changing their individual strategy, provided that the other players' strategies remain unchanged. This equilibrium can take various forms. A deterministic Nash equilibrium requires agents to always choose a particular joint action. A stochastic Nash equilibrium is one describing a stochastic joint policy, where agents select their

actions given a specific equilibrium distribution. Nash equilibria are pivotal in multi-agent systems research, as they help predict the behaviour of agents interacting in competitive or cooperative environments.

Importantly, there may be multiple Nash equilibria in a particular environment, with some equilibria yielding better returns than others. Thus, a problem of equilibrium selection arises to avoid suboptimal equilibria and efficiently converge to the optimal one (Bowling & Veloso, 2002; Conitzer & Sandholm, 2007; Harsanyi & Selten, 1988; Kalai & Lehrer, 1993). While equilibria are instructive for understanding valuable long-term strategies, their direct application may be flawed in dynamic and unpredictable environments (Shoham et al., 2007). Additionally, they are impractical, if not impossible to compute in high-dimensional and non-simulated environments, as they require knowing the outcomes of all actions in all possible states. Nonetheless, Nash equilibria remain a significant concept in multi-agent learning, helpful for illustrating the dynamics of some multi-agent scenarios.

3.3 Context: Challenges in Multi-Agents Robotic Domains

Working towards learning behavioural strategies for multi-robot systems requires first understanding the implications of learning in such environments. The multiplicity of intelligent agents in an environment has many important impacts on how these agents are able to learn, disrupting basic RL techniques and imposing strong architectural choices. Similarly, learning in robotics faces several obstacles that hinder the applicability of algorithms designed to learn in games and simulations (Pierson & Gashler, 2017). In this section, we define the challenges of both domains, look at how they intersect, mention some basic or foundational approaches to tackle them, and discuss their impact on the design of learning algorithms.

3.3.1 Learning with Multiple Agents

Non-Stationarity

The problem of non-stationarity refers to the continuously changing nature of the learning environment. It occurs in single-agent RL, as training the agent changes its behaviour, thus modifying the nature of its future interactions with the environment. But, in the multi-agent setting, non-stationarity is greatly amplified by having all agents change their behaviour after each training phase. From the point of view of an agent, the optimisation problem (i.e., "finding the best policy to maximise future returns") changes after each training phase, because the distribution of states and the outcome of actions change. This leads to a moving target problem, where continuous co-adaptation can induce unstable training. This is especially bad for RL algorithms, as they bootstrap the learnt estimates to determine the best solution to the optimisation problem. If this problem changes too much after each training phase, the learnt estimates can end up ineffective. In addition, the optimisation problem being constantly evolving, past experiences are made irrelevant for future training phases as they depict obsolete interactions. This makes the use of experience replay unusable without specific adaptation (Foerster et al., 2016a).

Non-stationarity is a major issue preventing single-agent RL algorithms from working in multi-agent settings. To tackle this problem, multiple approaches have been proposed. In Section 3.4, we will see that using centralised information during training can help constrain the effects of non-stationarity. Because experience replay is such an important part of making DRL function well, some works have proposed

solutions to adapt it for MADRL by ensuring that all agents train on concurrent time steps (Omidshafiei et al., 2017) and giving less weight to older experiences in the replay memory (Foerster et al., 2017). While the concurrent replay method has been widely adopted, this does not completely solve the issue. Replay memories in MADRL are often set to discard old experiences earlier than in single-agent RL to prevent learning from too irrelevant data.

Credit Assignment

Credit assignment refers to the problem of finding how each past action contributed to the obtained rewards. As for non-stationarity, this problem exists in single-agent RL, with actions contributing to rewards obtained in the future, requiring a form of *temporal credit assignment*. But this becomes harder with multiple agents as, at each step, multiple actions are performed simultaneously by different agents. Knowing the marginal contribution of each action is a crucial requirement for evaluating each local policy accordingly. This **multi-agent credit assignment** problem can be examined from the point of view of the whole MAS, looking at *"how much each agent contributed to the obtained return?"*, or from the point of view of a single agent, answering the question *"was my success (or failure) due to my actions, or to some other agent's actions?"*. This problem is particularly difficult in the Dec-POMDP framework where all agents share a common reward signal. In such cases, a "lazy" agent could be reinforced into doing nothing because the other agents manage to solve the task on their own. However, it is worth noting that the problem persists if each agent gets its own individual reward (and, similarly, in competitive settings), because a local reward might have been caused by another agent's actions.

To tackle the multi-agent credit assignment problem, some methods have been developed for trying to model the marginal contributions of each agent. Exact solutions exist in simple, controlled environments, like the *Shapley value* (Shapley, 1953) that evaluates an agent with the expected gain in utility of having this agent contribute in any possible coalition of agents. But this value is computationally very expensive and even approximations are hardly applicable to environments outside the game-theoretic framework or with a large number of agents (Fatima et al., 2008; Michalak et al., 2014; Wang, Zhang, et al., 2020; Wang et al., 2022). Another approach is to compute the marginal contribution of each action as the difference between the observed outcome and a counterfactual outcome where the evaluated action was not performed. The *wonderful life utility* (WLU; Wolpert and Tumer, 1999) replaces the action by a "null" action and simulates the presumed return: $WLU(a_i) = G(\mathbf{a}) - G(\mathbf{a}_{-i}, a_i = \text{null})$, with G the evaluation function and $\mathbf{a}_{-i}$ the joint action without action a_i. The *aristocrat utility* (AU; Wolpert and Tumer, 2002) takes as counterfactual metric the expected outcome from the truncated joint action: $AU(a_i) = G(\mathbf{a}) - \mathbb{E}(G|\mathbf{a}_{-i}, s)$, with s being the state in which action a_i was performed. However, these utilities are impractical to compute in high-dimensional, stochastic environments. In Sections 3.4.3 and 3.4.4, we will introduce recent approaches, based on MADRL, to the credit assignment problem that better fit more realistic settings.

Coordination

In many cooperative multi-agent tasks, agents are required to coordinate their actions to fulfil the objective. Robots might have to lift a heavy object together. In games played in teams, coordination between teammates is often crucial to successful

tactics (Bard et al., 2020; Samvelyan et al., 2019). Thus, coordination is often a pursued ability in cooperative multi-agent learning. Coordination can be defined, in the multi-agent context, as the ability of an agent to synchronise its local actions with the anticipated behaviour of other agents to achieve a particular objective. This ability encompasses a wide set of knowledge the agents must acquire: knowledge about the environment dynamics, about how the task is completed, and about the other agents' strategies. Learning all this can be extremely complicated, especially in partially observable environments where local information is often insufficient to understand the full state of the environment.

Coordination skills may be learnt with different approaches. First, by acquiring a thorough knowledge of the joint policy search space to know how to behave in any situations. This can be related to a problem of multi-agent exploration of the joint policy space, which we study more deeply in Chapter 4. Another approach would be to have an explicit mechanism for coordinating actions during execution. This can be tackled in various ways. In Section 3.4.6, we review methods that enable agents to learn a model of the other agents' policies, allowing them to coordinate their choice. Communication can be a handy mechanism for exchanging local information and reaching a consensus on the best way to act (see Section 3.4.5 and Chapter 5). Finally, coordination graphs (Böhmer et al., 2020; Guestrin et al., 2002; Li et al., 2021) more explicitly model coordination by learning pairwise joint value functions.

Scaling

A major issue of all multi-agent learning algorithms is how they deal with scaling to larger state and action spaces and, especially in the case of MASs, to a larger number of agents in the system. Being able to handle more agents efficiently makes an algorithm more applicable to various settings. But, this is not elementary, as scaling exacerbates all issues faced by multi-agent learning. Non-stationarity is increased because having more agents in the system means the environment changes more after each training step. Credit assignment is harder because the outcomes depend on more local actions. Learning value and policy functions is more difficult, as they depend on more independent elements and, thus, the search space is larger. On a more technical note, having more agents makes training more computationally expensive. As in RL, the use of deep learning techniques can help for dealing with scaling. It helps generalising to unseen configurations of the environment, partly compensating for the larger search spaces. In Section 3.4.2, we will see that having agents learn independently can help with scaling, but it also implies some important downsides.

3.3.2 Learning in Robotic Domains

Domain Complexity

Going towards learning in robotic environments, an important obstacle to successful learning is the complexity of realistic environments. Many multi-agent environments simplify the definition of states and actions substantially to focus on particular multi-agent dynamics (e.g., two-dimensional grid level-based foraging, as shown in Figure 3.1b, to study how coordination can arise). But, to apply learning algorithms to robotic domains, they must be capable of handling the full complexity of realistic state and action definitions. This complexity takes three different forms: high-dimensionality, continuity, and multi-modality.

High-dimensional states and actions are made of many components that each carry some information. Take, for example, a robotic hand manipulating objects. For observing its environment, it might use an RGB (red-green-blue) camera producing images made of thousands of three-valued pixels (one value for each colour). Understanding each image requires knowing how these numerous values relate with each other to compose high-level information. For manipulating objects, the robotic hand must control a large number of motors at once to achieve the desired motion. Each motor necessitates its own policy, but all policies must coordinate to achieve the desired behaviour. Note that, such high-dimensional action space can benefit from being formulated as a multi-agent problem, with each joint being handled by one agent (Sartoretti et al., 2019).

Robots usually deal with **continuous** inputs and outputs. For example, pixel values can be represented as a continuous value of colour intensity. Continuous angles of rotation are used to precisely articulate joints. Having continuous states and actions implies that the respective search spaces are infinite. This requires the ability to generalise well to handle previously unseen states: in a continuous state space, a robot will never experience the same state twice so it needs to use its experience in very similar states to know how to react. Handling continuous states and actions also requires specific techniques. In Chapter 2, we have seen how tabular methods are unable to handle continuous state and action spaces, as they require learning estimates for each possible state-action pair (see Section 2.3.1). Similarly, deep Q-learning is limited to discrete action spaces as it learns the values of a discrete set of pre-defined actions. Thus, dealing with continuous spaces imposes strong design constraints.

Finally, robots often encounter **multi-modal** states and actions, which are composed of multiple different types of information. In addition to the RGB camera, the robotic hand might have tactile sensors on each finger that generate precise haptic information on how the object is being grasped. Actions may also be distributed on different types of actuators: e.g., wheels to navigate in a room, hands to grasp objects, and voice to communicate. Multi-modality requires the controller to understand different types of information and recognise their relationships and interactions.

These input and output complexities each require specific care and altogether make learning more difficult. Deep learning techniques are an important tool for dealing with this level of complexity. They allow automatic learning of high-level representations from high-dimensional data (Simonyan & Zisserman, 2015), generalisation in continuous spaces (Schulman et al., 2016), and fusing of multi-modal data (Driess et al., 2023; Radford et al., 2021). With the right learning approach and enough resources, deep learning enables efficient learning of complex robotic behaviour (Andrychowicz et al., 2020; Pinto & Gupta, 2016). However, using deep learning has some notable drawbacks such as sample inefficiency and computational cost. When used in the context of RL, deep learning techniques are useful to open the range of potential applications, but RL techniques still require extensive work to suit robotic domains well (Ibarz et al., 2021; Sünderhauf et al., 2018).

Partial Observability

All realistic robotic environments are inherently partially observable. It is practically impossible to capture the full complexity of a real-world setting within a single state vector. A robot typically accesses information about its environment by observing it with its own sensors, providing it with a subjective view of its immediate surroundings. This limited perspective does not offer a complete description of the full state of the

environment. To achieve its objective, a robot must infer some information based on its past experiences and observations from previous steps.

In the context of multi-agent RL, the Dec-POMDP, defined in Section 3.2.1, is useful for modelling this environmental uncertainty. Similar to realistic robotic settings, agents within a Dec-POMDP only observe a subjective subset of the complete state of the environment. The resulting uncertainty significantly increases the difficulty of learning optimal policies. To overcome this, robots require specific tools. In single-agent POMDPs, agents can learn to infer the current state of the environment from their incomplete observations (Abbeel et al., 2006; Lee, Nagabandi, et al., 2019). However, in a multi-agent setting, this is significantly harder as the state also depends on the actions of other agents (Oliehoek & Amato, 2016; Papadimitriou & Tsitsiklis, 1987). Memory can be a valuable tool in dealing with uncertainty, enabling agents to remember previous observations during an episode to better infer the current state (Hausknecht & Stone, 2015). Communication can also play a crucial role, allowing agents to share their individual subjective knowledge, thus cooperating to build a more accurate representation of the current state (see Section 3.4.5 and Chapter 5 for more detail).

Reality Gap

Because RL requires thousands of experiences to converge to an efficient strategy, a promising approach to learning robotic tasks is to train in simulation and then apply the learnt policy on the real robots. However, this transfer is challenging due to numerous subtle differences between the simulated environment and the real world, a problem known as the **reality gap** (Jakobi et al., 1995). Even the most sophisticated simulations fall short of accurately reproducing real-world dynamics. Real robotic sensors and actuators have imperfections that are often not modelled in simulations. These minor discrepancies accumulate, causing a policy that performs well in simulation to fail when deployed on a real robot. To leverage extensive training in simulation, techniques must be developed to enable efficient simulation-to-reality (sim-to-real) transfer (Ju et al., 2022).

One possible sim-to-real approach is to develop models that adapt quickly and effectively to new domains (Rusu et al., 2017). This can be facilitated by learning the critical characteristics shared between the training and execution domains (Gupta, Devin, et al., 2017; James et al., 2019). Another strategy, which can complement the first, is domain randomisation. This technique involves slightly randomising observations and actions during training in simulation, making RL agents more robust to minor discrepancies in their inputs and outputs (Andrychowicz et al., 2020; Chebotar et al., 2019; Tobin et al., 2017). Lastly, hierarchical learning offers a promising approach to developing transferable policies (D'Ambrosio et al., 2024; Nachum et al., 2020). Assuming that the reality gap affects lower-level actions more, it should be easier to learn a transferable high-level policy that selects macro-actions (e.g., find an apple) (Amato et al., 2019). Low-level policies for executing these macro-actions with basic actions (e.g., move forward) can be learnt more easily within the target domain.

Embodiment

Contrary to most computer programs, the controller of a robot is **embodied**: it is situated within a concrete body that lives inside an environment (Pfeifer & Bongard, 2006). There are many different views of embodiment from philosophy, psychology,

cognitive science, and artificial intelligence, describing the role of embodiment in learning and intelligence (Barsalou et al., 2003; Kiverstein, 2012; Lakoff & Johnson, 1999; Sünderhauf et al., 2018). Here, we define some useful notions for appreciating the various implications of control and learning in robotics. **Physical** embodiment relates to the instantiation of the controller in a physical body that can sense and act upon its environment. **Temporal** embodiment refers to the fact that robots experience their environment through sequences of strongly correlated states. This has important implications for the treatment of both past states, on which the present state depends, and future states, which can be influenced by the robot's actions. What we call **composite** embodiment refers to the composite nature of a robot, made of many interconnected parts that all serve a specific purpose in the life of the robot. Ziemke, 2003 calls this "organismoid" to relate the composite body of robots to that of a living organism in which the presence of organs may play a major role in the development of commonsense intelligence (Lakoff & Johnson, 1999). Finally, **social** embodiment describes the fundamental relation between intelligent entities in the environment, and how these social relations may shape intelligence (Barsalou et al., 2003). In the context of robotics, other entities may be other (potentially heterogeneous) robots, humans, or even animals. Such interactions are extremely diverse and may be dictated by implicit or explicit social rules. All these notions manifest in robotics, but not necessarily in other domains of computer science and artificial intelligence. Thus, it is instructive to consider these notions to build better robotic systems.

Embodiment is not a problem in itself but an essential perspective for understanding the challenges of robotic applications. Being embodied in all the forms described above has many important implications. The physical and temporal views of embodiment, put together, imply a capacity to interact with the environment: the robot needs to perform actions to gather information and alter the environment. This introduces safety concerns, as actions in a physical environment may result in catastrophic outcomes. It also implies a highly dynamic range of environmental situations: in the real world, the people, objects, and furniture present in the room might change over time; a single type of object can be associated with many different forms and colours; and one task can be performed in different environmental settings (e.g., different ground textures or weather conditions) which may change during the robot's life. The robot's composite form entails the need to account for the inherent characteristics of its robotic parts: a particular sensor might require a specific behaviour to gather information properly, or any sensor or actuator may have flaws or momentarily malfunction. Finally, social embodiment implies the need to study the dynamics of interaction between multiple robots, as in MAS research, and between robots and humans (see Section 3.5.4).

It is important to note that, while physicality is an important aspect of embodiment, simulations are still a useful tool for studying embodiment. Simulated environments accurately replicate many challenges linked with embodiment: interaction with the environment, temporal embodiment, and social interactions. However, some other aspects are harder to emulate: accurate physical dynamics, variety and dynamicity of the environment, malfunctions, and human-robot interactions. Thus, simulations are still relevant for addressing learning in robotics, but their shortages should not be overlooked and, if possible, addressed accordingly with real-world experiments.

-

Challenges in multi-agent learning and robotics are numerous and, when combined, often exacerbate each other, making multi-robot environments particularly challenging.

Due to this great multiplicity of issues, it is common for some problems to be addressed separately. However, this leads to the domain of multi-agent learning research being highly fragmented, with many concurrent lines of work that are difficult to compare. To advance towards the learning of complex behaviours in multi-robot environments, multi-agent learning needs to integrate the inherent challenges of robotics and develop methods that efficiently tackle all issues of multi-agent learning. In the next two sections, we will review the main directions of multi-agent reinforcement learning research and then reflect on how they address the specific problems of robotics.

3.4 Methods in Multi-Agent (Deep) Reinforcement Learning

To tackle multi-agent environments, RL algorithms have been adapted and extended in various ways. As with single-agent RL, deep learning has enabled addressing more complex multi-agent environments. In this section, we present a survey of state-of-the-art MADRL algorithms. We focus on some important techniques employed in the past few years to produce efficient algorithms. Our objective is to present the different approaches and explain the functioning of state-of-the-art algorithms that are frequently found in the literature and which we may use in subsequent chapters of this thesis.

3.4.1 Multi-Agent Learning Paradigms

Designing a multi-agent learning algorithm requires first choosing how we consider one agent in relation to the MAS. This choice will dictate how information can be used for training the agent's policy and for executing it. In MASs, the gathering of information about the environment is always done locally by each agent, through the local observations o_t^i (see Figure 3.2). The processing of this information, however, can be done in various ways depending on our assumptions of how information can flow in the MAS. These assumptions will greatly impact the design of the resulting algorithms, as they dictate what information is available to them during training or execution. Here, we examine how, what, and when information can or should be centralised. By centralising information, we mean that the algorithm gathers the local observations together (i.e., it has access to the joint observation) and uses them during either the training phase to improve the learning update, the execution phase to improve local policies, or both. Note that this does not include communication, as defined in Section 3.2.3, that involves the agents actively choosing what information they should share with others. Centralising some information can greatly improve the efficiency of multi-agent learning algorithms. But, centralisation, if it is even possible, comes at several costs in terms of algorithmic conception and complexity. Thus, different MA(D)RL algorithms may prefer different levels of centralisation depending on environmental constraints and design decisions. They can be classified into three main categories that define how information is allowed to flow in the MAS during training and execution.

The **centralised training and execution** (CTE) paradigm allows information to be shared between agents at all times. This means that the agents' policies might be conditioned on information coming from other agents. Centralising information can take many different forms: sharing other agents' observations, internal hidden states, learnt parameters, generated value estimates, or policy outputs. This allows agents to generate better predictions and to have more information at hand during

training to stabilise training. For example, a local policy conditioned on the joint observation, $a_{i,t} \sim \pi_i(.|\mathbf{o}_t)$, will have more information about the environment, allowing better prediction of the best action to choose. A more extreme version of this would have a single controller centralising all local observations and generating the joint action: $\mathbf{a}_t \sim \pi_{central}(.|\mathbf{o}_t)$. This would reduce the problem to a single-agent one, preventing the issues of multi-agent non-stationarity and credit assignment. But, these methods imply that the search spaces (for learnt estimates) grow exponentially with the number of agents, making such solutions poorly scalable. Additionally, in many environments, it might not even be possible to share information between agents. In realistic scenarios, agents are often independent and might not be connected to a central unit that can centralise and redistribute information freely.

At the opposite side of the spectrum, **decentralised training and execution** (DTE) considers that no information can ever be centralised. This is a far more realistic assumption as it does not rely on an algorithmic-level connection required to pass information between agents. It is also more computationally efficient to learn estimates in tighter observation and action spaces. So it allows better scaling to a large number of agents. However, because less information is at their disposal to train and compute estimation functions, these algorithms can learn less efficiently and suffer more from non-stationarity. Note, again, that this does not forbid having communication between agents. In fact, decentralised agents would largely benefit from learning an efficient way to discuss with partners to share local information and coordinate their actions (Cao et al., 2018).

To improve on DTE without losing the decentralised execution, a middle ground can be found with **centralised training and decentralised execution** (CTDE). In CTDE, information is allowed to be shared during training, but agents are kept totally independent during execution. Because training is usually done in a controlled environment, we can often make the reasonable assumption of being able to centralise some information during training. This allows a range of simplifications, with varying degrees of assumptions on how information can be centralised: from ensuring agents are trained on concurrent steps (Omidshafiei et al., 2017), to learning centralised value functions (see Section 3.4.3). In general, sharing some information makes RL training easier by reducing non-stationarity and compensating for partial observability. To ensure that execution is decentralised, the agents' policies are required to be conditioned only on local information, i.e., $a_i \sim pi_i(.|o_i)$. Because decentralised execution is such an important requirement and centralising information during training allows great improvements in RL training, the CTDE paradigm is the most preferred one today for MADRL algorithms.

3.4.2 Independent Learning

A seemingly simplistic approach to the multi-agent problem is to consider each agent totally independently of the rest of the MAS, ignoring other agents, as if they were parts of the environment. This is often referred to as Independent Learning (IL; Claus and Boutilier, 1998; Tampuu et al., 2017; Tan, 1993), where each agent can be trained independently with single-agent RL methods. But, a fully decentralised version of IL suffers badly from non-stationarity as the environment, composed of other learning agents, is continuously changing (Foerster et al., 2016a; Tan, 1993). Jiang and Lu, 2022 solve the problem of non-stationarity by learning from a surrogate transition probability that considers other agents will act optimally. But this requires strong

assumptions on the environment, especially that it is deterministic. Thus, some assumptions from CTDE might be adopted to facilitate IL. For example, IL agents often share the parameters of their learnt policy or value functions (Foerster et al., 2016a; Gupta, Egorov, & Kochenderfer, 2017; Schroeder de Witt et al., 2020; Tan, 1993), allowing faster convergence. To maintain diverse behaviours, the policies are conditioned on a unique identification number corresponding to the agent. To enable the use of experience replay in a decentralised algorithm, Omidshafiei et al., 2017 introduced concurrent experience replay for training agents on the same environment steps during each update. This ensures that independent learners have a common training schedule, thus mitigating the effects of non-stationarity. While IL seems simplistic, it has actually been shown to be competitive with more centralised alternatives (Jiang et al., 2024; Lyu et al., 2021; Schroeder de Witt et al., 2020). This simplicity allows a great deal of freedom when designing training procedures. The self-sufficiency of agents makes it possible to modify the teams at will. For example, Jaderberg et al. (2019) trains a large population of independent agents, changing the team regularly so they learn to be robust to different partners and opponents.

3.4.3 Multi-Agent Actor-Critics

The CTDE framework requires only that the local policies are decentralised, that is, they must be conditioned on local information only (i.e., observations or history). Centralised information may be used, however, to improve the learning of these local policies. Therefore, a natural implementation of this paradigm is to use the actor-critic framework with decentralised actors and a centralised critic. Because the critic is used only to train the actor (see Section 2.3.2), we can use centralised information to improve the value estimates learnt during training. The **multi-agent DDPG** (MADDPG; Lowe et al., 2017) introduced this approach by having one DDPG-based actor-critic (see Section 2.5.2) for each agent, but allowing the critics to use the joint observation to compute the local action-value estimates (see Figure 3.3). By using the information collected by all agents instead of only the local observation, the critic has more information on the current state of the environment to better estimate the action-value function. This also decreases the effects of non-stationarity: because the critic knows the state of other agents, it is less sensitive to changes in other agents' policies.

This centralised critic idea has been widely used for its intuitive advantages. It has been extended to include memory with a recurrent neural network used in both the decentralised policies and the centralised critic (Wang, Everett, & How, 2020). Other actor-critic algorithms have been implemented this way, with multi-agent PPO (MAPPO; Yu et al., 2021b), multi-agent TD3 (MATD3; Ackermann et al., 2019) and multi-agent SAC (MASAC; Yu et al., 2021a). Iqbal and Sha (2019b) added an attention mechanism in the centralised critic to better combine the centralised information. Finally, Foerster et al. (2018) used a centralised critic to improve credit assignment with their counterfactual multi-agent policy gradient (COMA). Having a centralised critic allows them to efficiently compute a marginal contribution for each local action, by computing the advantage of taking the action compared to all other possible actions: $A^i(\mathbf{h}, \mathbf{a}) = Q(\mathbf{h}, \mathbf{a}) - \sum_{a^{i\prime}} \pi^i(a^{i\prime}|h^i) Q(\mathbf{h}, (\mathbf{a}^{-i}, a^{i\prime}))$.

The nature of the centralised information used as input to the critic can vary. The joint observation is used often (Iqbal & Sha, 2019b; Lowe et al., 2017). Memory-equipped agents can extend this with the joint history (Foerster et al., 2018; Wang, Everett, & How, 2020; Yu et al., 2021b). But, it has also been proposed to use the

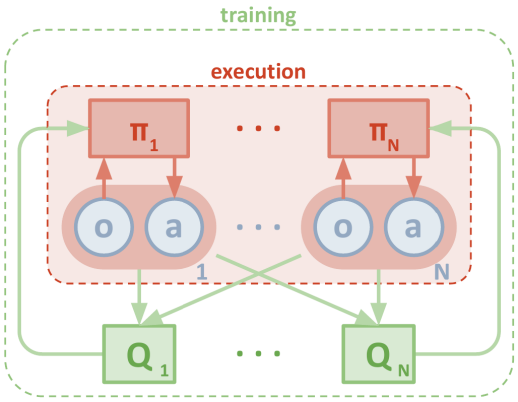


Figure 3.3: The MADDPG architecture (Lowe et al., 2017) illustrating the centralised critics training decentralised policies. Each agent has a local policy used during training and execution, and a centralised critic, conditioned on the joint observation, used only during training.

global state of the environment s_t, if it is defined and accessible (Foerster et al., 2018; Lowe et al., 2017). This is often described as preferable because the global state would be a more condensed and complete description of the environment compared to the joint observation. However, it has been shown that using the global state can result in a higher variance in training as a result of introducing bias (Lyu et al., 2023). Additionally, defining the global state is a problem on its own, which might not be straightforward in some environments. In fact, in rich and complex settings (e.g., the real world), it might be impossible to describe the state fully. Thus, choosing a particular state definition is a complex design choice that can have important impacts on training.

3.4.4 Value Factorisation

In Section 3.3.1, we presented the multi-agent credit assignment problem that arises when multiple agents share a common reward signal and need a way to measure the contribution of their behaviour towards the observed common outcome. In such settings with a common reward signal, learning the value of local action a^i, with respect to the global return G (as in Equation 3.2), can be problematic because G does not depend only on a^i, but also on the other agents' actions. In Section 3.3.1, we introduced methods that tackled this issue by computing a marginal contribution of each action. But, defining an effective marginal contribution can be tricky and computing it is usually expensive. Instead, another approach would be to learn the action-values of each action with respect to their real, unknown contribution, knowing that they are related in some way to the known common return. In other words, given the joint action-value $Q(\mathbf{h}, \mathbf{a})$ estimating the expected global return (see Equation 3.1) and local action-values $Q^i(h^i, a^i) := \mathbb{E}_{\pi^i}[u^i \mid h^i, a^i]$, with u^i the *local utility* measuring the contribution of agent i in G, we need to find how the local values compose the joint value:

$$Q(\mathbf{h}, \mathbf{a}) = f(\{Q^i(h^i, a^i)\}_{1 \leq i \leq n}, \mathbf{h}). \tag{3.3}$$

The function f describes how each local value contributes to the joint value, depending on the joint history. Given the global return G, **value factorisation** (or "value decomposition") approaches learn the joint action-value and a way to decompose it into local action-values (i.e., function f). Value factorisation is a form of implicit

credit assignment where we learn local value functions by learning how they compose the global value.

Learning this properly allows having local action-value functions that can be used for choosing greedy local actions for each agent. Because the objective is to maximise the global return, this requires that greedy local actions lead to optimal joint actions. This has been referred to as the **individual-global-max** (IGM; Rashid et al., 2018) property, requiring that choosing the greedy joint action with respect to the joint action-value corresponds to choosing greedy local actions with respect to each local action-value, i.e.:

$$\arg\max_{\mathbf{a}} Q(\mathbf{h}, \mathbf{a}) = \{\arg\max_{a^i} Q^i(h^i, a^i)\}_{1 \leq i \leq n}. \tag{3.4}$$

Respecting this property is essential to be able to use the learnt local action-values for local action selection. Thus, a multi-agent value-based learning algorithm that follows the IGM principle can fit into the CTDE paradigm, with decentralised local action-values used during execution, trained in a centralised manner with the help of a learnt joint action-value.

Sunehag et al., 2018 introduced this idea with their **value decomposition network** (VDN), making the simple assumption that local action-values should sum up to the joint action-value:

$$Q(\mathbf{h}, \mathbf{a}) \approx \sum_{i=1}^{n} Q^i(h^i, a^i). \tag{3.5}$$

With this assumption, Q is differentiable with regard to each Q^i. Thus, VDN is able to learn individual action-values, parameterised as deep neural networks, from the DQN objective (see Equation 2.42) completely end-to-end. This linear version of value factorisation has the advantage of simplicity and being computationally lightweight, which allows excellent scalability.

While this linear decomposition is intuitively logical and follows the IGM property, there is no guarantee that the true factorisation function f is a linear transformation of local utilities. Therefore, the formulation of VDN limits the factorisation operation and the learnt representations of joint and local action-values. To address this, **QMIX** (Rashid et al., 2018) introduces a separate "mixing" neural network that takes the local action-values and the global state as input, and outputs the joint action-value: $Q(s, \mathbf{a}) = f_{MIX}(\{Q^i(h^i, a^i)\}_{1 \leq i \leq n}, s)$ (see Figure 3.4). Note that, in their implementation, they consider that the global state s is available, but it can be replaced by the joint observation of history if needed. The mixing network learns a factorisation function that depends on the current state of the environment, allowing much richer factorisation capacities. To ensure that the IGM is respected, the mixing of local values must be monotonic: if a local value increases, the joint value must increase too, i.e., $\frac{\partial f_{MIX}}{\partial Q^i} \geq 0$. This monotonic constraint is ensured by having the weights of the mixing network be positive. But, this constraint must be applied only for the local action-values, not for the state. To allow this, QMIX employs a hypernetwork (Ha et al., 2017), which uses a separate MLP, conditioned on s to generate the weights of the MLP used for factorising the local action values. The absolute value of the generated weights is taken to ensure the monotonic constraint described above. Using a hypernetwork allows to depend on the state in a non-monotonic way and to learn more complex dependencies between the action values and the state (Zhou et al., 2020).

Many subsequent works have extended QMIX to improve its performance (Hong

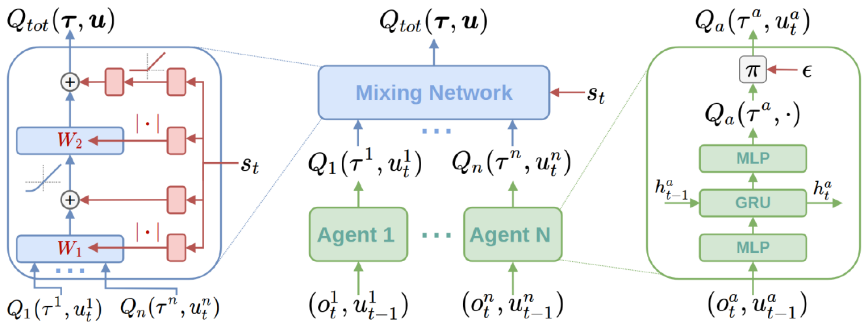
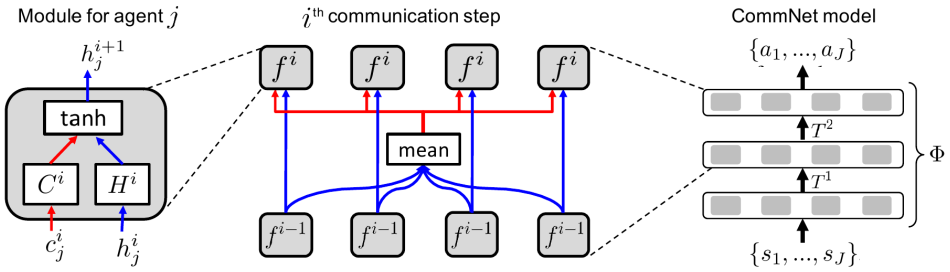


Figure 3.4: The architecture of QMIX (Rashid et al., 2018) illustrating the learnt monotonic value factorisation. The middle shows the overall architecture with local action-value functions, mixed in to compute the joint action-value. On the left is shown the mixing network that takes in the local action-values to compute the joint action-value, with the weights generated by the hypernetwork (in red) conditioned on the global state. On the right, the architecture for a local action-value function is described, with a recurrent network (GRU) used for memory.

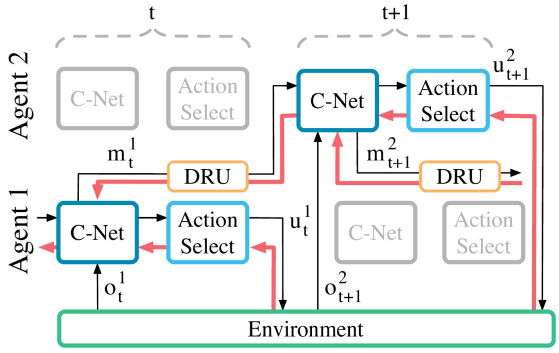
et al., 2022; Iqbal et al., 2021; Peng et al., 2021; Rashid et al., 2020; Son et al., 2019; Sun et al., 2023; Wang, Ren, Liu, et al., 2021; Xu et al., 2023; Yang et al., 2020; Zhou et al., 2022; Zhou et al., 2020). The main issue is the monotonic constraint that limits the potential of QMIX for modelling some factorisation functions, which might induce poor performance in some scenarios. Yang et al. (2020) reformulate the factorisation as a weighted sum that can be learnt with an attention mechanism. Zhou et al. (2020) and Peng et al. (2021) both extend QMIX to be used in an actor-critic algorithm. Having local policies for action-selection allows relaxing the monotonic constraint imposed in QMIX, required only because the local action-values were used for action-selection. Without this constraint, more accurate factorisation functions can be modelled. Additionally, using an actor-critic framework enables working with continuous actions (Peng et al., 2021) and learning stochastic policies (Zhou et al., 2020). QPLEX (Wang, Ren, Liu, et al., 2021) reformulates the problem by making the IGM property based on the advantage function instead of the action-value: given that $Q = V + A$ (as defined in Section 2.5.1) and that the action selection does not depend on V, the IGM constraint can be transferred onto the advantage function, rewriting Equation 3.4 with the joint and individual advantages instead of action-values. This allows easier learning of a value factorisation. Finally, Sun et al. (2023) propose a distributional extension of QMIX to better handle stochastic environments. While extensions provide several theoretical advantages, QMIX is still widely considered as state-of-the-art, and seen more often as a baseline. Also, it is interesting to note that the flexibility granted by QMIX for modelling more complex value factorisation might not be needed in many simpler tasks. In fact, VDN has been shown to outperform QMIX in some scenarios (Papoudakis, Christianos, Schäfer, & Albrecht, 2021; Wang, Ren, Han, et al., 2021).

3.4.5 Differentiable Emergent Communication

An important part of human interactions is communication. We often use our multi-modal communication abilities (e.g., speech, language, hand gestures, facial expressions) to share our knowledge, coordinate our actions, negotiate, or express goals and feelings. It is therefore natural to study communication in the context of multi-agent learning. We have seen, in Section 3.3.1, that the decentralised aspect of multi-agent settings gives rise to many issues that can be alleviated with efficient multi-agent



(a) CommNet (Sukhbaatar et al., 2016)



(b) DIAL (Foerster et al., 2016b)

Figure 3.5: Two architectures of differentiable communication. (a) CommNet learns a centralised communication network that shares local information in multiple rounds of communication before choosing local actions. (b) DIAL learns decentralised communication with centralised training, by allowing gradients to flow between agents. Because messages result from differentiable operations of neural networks, communication can emerge from back-propagation of gradients from the RL objective.

communication. To achieve this, agents need to learn how to communicate. This may be summarised as learning *what to communicate* and *how to communicate it*. A popular approach for learning communication in a multi-agent setting is to let agents develop their own communication system in the process of learning to complete a given task. This gives rise to an emergent communication system that specifically fills the communication needs of the task.

Recently, a successful approach to emergent communication with MADRL has been to implement communication as a differentiable sub-step of the action-selection process (Foerster et al., 2016b; Sukhbaatar et al., 2016). By using only differentiable operations – i.e., neural networks – to generate and process the messages, the gradients of the RL objective can be back-propagated through the message-generating modules so they participate in maximising future returns. Importantly, having differentiable messages allows for gradients to be passed between agents: as a message generated by agent i will impact the choices made by agent j, agent i can be trained to generate messages that maximise agent j's returns. This way, the communication mechanisms emerge from the task requirements. CommNet (Sukhbaatar et al., 2016) introduced a centralised approach of differentiable communication (see Figure 3.5a), with all agents sharing a communication network enabling information sharing and trained to maximise the joint return. DIAL (Foerster et al., 2016b) took this into the CTDE paradigm (see Figure 3.5b), with each agent's communication network (C-Net in the figure) trained from the other agent's learning objective. In these two approaches, neural networks generate differentiable messages comprised of a vector of continuous values, which may carry information to other agents. By learning to maximise the

RL objective, the message-generating networks are trained to generate messages that allow other agents to select better actions.

This approach of differentiable communication has been extended in various ways for more targeted information sharing (Das et al., 2019; Hoshen, 2017; Jiang & Lu, 2018) or to limit bandwidth usage (Han et al., 2023; Singh et al., 2019; Wang, He, et al., 2020; Zhang et al., 2019). While previously cited works use *continuous vectors* as messages, others have developed techniques to use *discrete symbols* for communicating (Cao et al., 2018; Jaques et al., 2019; Kim et al., 2019; Lazaridou et al., 2018; Rita, Tallec, et al., 2022). Discrete symbols are advantageous because they limit the bandwidth of transmitted messages. They also incite the emergence of certain qualities of natural languages that make human communications so efficient (Chaabouni et al., 2019; Mordatch & Abbeel, 2018).

While emergent communication allows efficient learning of information transmission with deep RL, it has important limitations. As with all deep learning approaches, it acts as a black box that lacks practical ways of interpreting and measuring its efficiency (Lazaridou & Baroni, 2020; Lowe et al., 2019). Because it emerges from task-oriented training in a closed group of agents, the resulting communication mechanism will be hardly usable for solving different tasks and communicating with other agents. For these reasons, methods are investigated to learn more interpretable and generalisable communication skills. In Chapter 5, we will look into communication and language more in depth, to see how multi-agent communication can be improved.

3.4.6 Agent Modelling

Previously presented approaches rely on learning policy and value functions to learn multi-agent behaviour. This model-free approach predominates in multi-agent learning because learning a model in a multi-agent setting is made extremely difficult by the fact that, from one agent's perspective, other agents contribute to the environment dynamics: the transition probability and reward function. One step towards solving this is to learn a model of other agents' policies, based on previous observations (Albrecht & Stone, 2018). In fictitious play (Brown, 1951; Fudenberg & Levine, 1995; Hofbauer & Sandholm, 2002; Robinson, 1951), each agent keeps track of action counts by other policies to compute potential action probabilities, then choosing an action accordingly. Recent deep RL techniques have been employed to improve fictitious play and allow its use in more complex, partially observable environments (Heinrich & Silver, 2016; Jing et al., 2024; Papoudakis, Christianos, & Albrecht, 2021; Rahman et al., 2023; Strouse et al., 2021). Bayesian learning goes further by tracking probabilities over possible policies for other agents, allowing to model uncertainty about their current reasoning (Bowling & Veloso, 2001; Foerster et al., 2019; Hu & Foerster, 2020; Jordan, 1991; Kalai & Lehrer, 1993). Such agent modelling approaches are promising for learning intricate multi-agent interactions, as they allow to adapt to the observed behaviour of other agents instead of trying to learn a policy able to effectively answer to any situation. Additionally, this approach has the intuitive advantage of emulating the way human beings approach their interactions with other intelligent entities, as described by the *theory of mind* literature (Apperly, 2011; Aru et al., 2023; Heyes & Frith, 2014).

3.5 Robotic Perspectives on MADRL Research: Open Challenges and Shortcomings

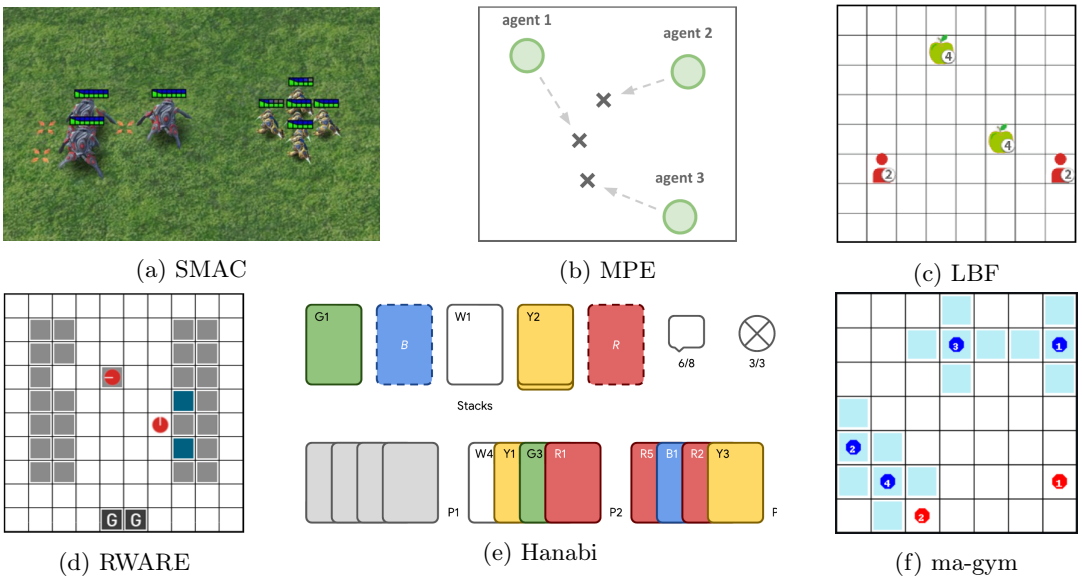
One of the key objectives of MADRL research is to facilitate the integration of robots into our daily lives. The real world is inherently multi-agent, as almost all conceivable situations involve interactions with other intelligent entities. Consequently, MADRL research aims to extend RL algorithms to be applied in complex multi-agent settings that more accurately reflect everyday scenarios. However, despite this objective, the current state of research has not yet come this far. The methods presented in the previous section are hardly applicable to robotic settings without significant modifications. This is partly because these methods are designed to be general multi-agent learning approaches, rather than being specifically designed for robotics. However, it may also result from some inherent limitations in their learning techniques or shortcomings of MADRL research.

In this section, we present four important challenges faced in MADRL research that must be addressed to enable the progress of MADRL algorithms in robotics. We define these challenges, examine how they are typically addressed, explore specific approaches to overcome them, and discuss potential improvements. While there may be other obstacles to overcome, we believe these challenges represent the main avenues for improving robotic control in complex multi-agent environments.

3.5.1 Benchmarking MADRL

Rigorous evaluation and comparison of different MADRL methods have been difficult to carry out due to several key challenges. Firstly, there is a large variety of learning environments and tasks, with little consensus on which setting should be used for studying which multi-agent problem. The most frequently found environments are the **Starcraft multi-agent challenge** (SMAC, see Figure 3.6a; Samvelyan et al., 2019) and the **multi-agent particle environment** (MPE, see Figure 3.6b; Lowe et al., 2017), but many others are also studied (see Figures 3.6c-f). Most environments have multiple tasks available for training and testing algorithms. But, it is often unclear what multi-agent learning problems are featured in a given task. Thus, different works choose different environments and tasks arbitrarily based on their preferences, available computing power, and the performance of their method. This complicates the comparison of different works that tackle different environments and tasks. Additionally, the value and rigour of these environments are seldom questioned, as shown by the recent revision of SMAC after it was found to be solvable by only observing the current time step (Ellis et al., 2023). Some other interesting environments are often proposed, for more efficient computation (Lechner et al., 2023; Michalski et al., 2023), human-agent teaming (Carroll et al., 2019), for allowing more agents (Lechner et al., 2023), more various tasks (Leroy et al., 2023), or more realistic settings (Kurach et al., 2020; Vinitsky et al., 2022); but there are seldom included in new studies and benchmarks.

Secondly, a thorough comparison with all existing methods is difficult. Learning multi-agent policies generally takes time and computing power. Among the available implementations, multiple versions of the same methods may have slight differences that are not always clearly stated. Different works might use different programming tools. While the programming language Python is widely adopted in Machine Learning, various Python libraries exist for implementing learning algorithms. No single



(a) SMAC (b) MPE (c) LBF

(d) RWARE (e) Hanabi (f) ma-gym

Figure 3.6: The most used MADRL environments. (a) The *Starcraft multi-agent challenge* (SMAC; Samvelyan et al., 2019), with teams of units fighting opponents in various scenarios. (b) The *multi-agent particle environment* (MPE; Lowe et al., 2017), a two-dimensional continuous environment with various tasks studying cooperative navigation and communication. (c) The *level-based foraging* (LBF; Albrecht and Ramamoorthy, 2013) task, studying coordination in a grid world. (d) The *multi-robot warehouse* (RWARE; Christianos et al., 2020), a robotic task in a grid world. (e) The cooperative card game *Hanabi* (Bard et al., 2020) for studying complex team strategy learning and adaptation to teammates. (f) The *ma-gym* two-dimensional grid world environment, with various cooperative tasks.

library is universally preferred[1], leading to significant differences that prevent easy adaptation from one library to another.

Lastly, the implementation of each method may differ from one work to another. All methods come with a very large set of hyperparameters, with some having a great impact on performance. Deep RL methods, which serve as the foundation for MADRL methods, can be implemented differently, with some implementation tricks having a major impact on performance. This variability makes comparison across different studies challenging.

For these reasons, assessing the progress of MADRL research is difficult. Performance reported in papers is hard to take at face value because of untold discrepancies hidden in the implementations and reported results (Singh et al., 2023). For example, Gorsane et al., 2022 show inconsistencies in the performance of QMIX reported in different papers. The consequence of this lack of standardised benchmark is concerning: it is unclear which methods are the best for any given purpose, and therefore what method should be used as a baseline in any given setting. To advance learning in multi-robot environments, it would be difficult to determine the most valuable MADRL algorithms to use in these environments.

Some benchmarks have been presented to try tackling this issue (Bettini et al., 2023; Ellis et al., 2023; Papoudakis, Christianos, Schäfer, & Albrecht, 2021; Yu et al., 2021a). They help clarify the field by providing a common ground for comparing

[1]Pytorch (https://pytorch.org/) is the most used in research, but some still use TensorFlow (https://www.tensorflow.org/), and a growing number of people prefer JAX for its computational efficiency (**Jax2018**).

important methods. However, there is limited variety in the environments used in these benchmarks, so the results might not hold in other tasks or more complex environments. It is also unclear what exact skill sets are required in each task, with only a rough measure of difficulty based on the returns obtained by all methods. This makes it difficult to discern the specific advantages of each method over others. Nevertheless, there are attempts to propose standardised evaluation protocols for new works (Gorsane et al., 2022; Singh et al., 2023), which is a promising avenue for building stronger and more progressive research in MADRL.

To move forward, better practices should be adopted. Proposed methods should all disclose hyperparameters and specific code-level optimisations. Evaluation protocols and metrics should be standardised across all new publications (Gorsane et al., 2022). Benchmarks should include more diverse environments, integrating the wide range of potential learning problems studied across multi-agent learning: continuous and discrete settings, various degrees of centralisation allowed during training and execution, communication between agents, and different degrees of environmental complexity with environments closer to robotics. There is no doubt that MADRL research would immensely benefit from improving its evaluation protocols as such, allowing less biased comparison between methods and deeper analysis of their abilities. This is essential for efficiently advancing the field and ensuring that proposed methods are robust and generalisable across tasks and environments.

3.5.2 Exploration

Exploration is arguably one of the most important problems in single-agent RL (Hao et al., 2024). It is particularly crucial when dealing with sparse rewards, where only a few interactions in the environment yield positive reinforcement signals. Such settings are often termed "hard-exploration" problems, requiring techniques allowing consistent discovery of the infrequent rewarding states. In multi-agent RL, the problem is exacerbated. Performance depends on the joint behaviour of all agents, requiring exploration of the space of joint policies to identify the best approaches. Exploration becomes a multi-agent problem, especially when coordination is necessary, as agents need to explore different ways to act in unison. Exploration is also a major subject in robotic environments. Partial observability complicates the issue, as one environment state may be observed from many different perspectives. Moreover, the safety concerns are both a constraint and an expectation for exploration. When exploration is conducted on robots, it should only involve safe states to ensure that robots do not injure themselves or others (Ding et al., 2021; Koller et al., 2018). At the same time, exploration is a way to find the optimal strategies that are safer for the robots. In this sense, exploration might be conducted in simulation to identify safe behaviours to execute on physical robots (Brunke et al., 2022; García et al., 2015).

Most single and multi-agent RL approaches treat exploration arbitrarily by infusing randomness into the behavioural policy during training. Q-learning-based approaches employ the epsilon-greedy strategy (see Section 2.3.1), policy-based approaches either add noise to actions, as seen in DDPG, or maximise the entropy of the policy, as in PPO (see Section 2.5.2). However, these methods are often insufficient for dealing with hard exploration problems (Burda et al., 2019; Ostrovski et al., 2017; Pathak et al., 2017). In multi-agent environments, random exploration often leads to the problem of relative overgeneralisation, where agents are attracted towards suboptimal Nash equilibria because the optimal strategy is too marginal to be found consistently through random exploration (Wiegand, 2003). In Chapter 4, we will

tackle this issue of multi-agent exploration by describing the problem of *relative overgeneralisation* that results from poorly coordinated exploration, reviewing the related literature, and exploring intrinsic motivation to explicitly induce joint exploration.

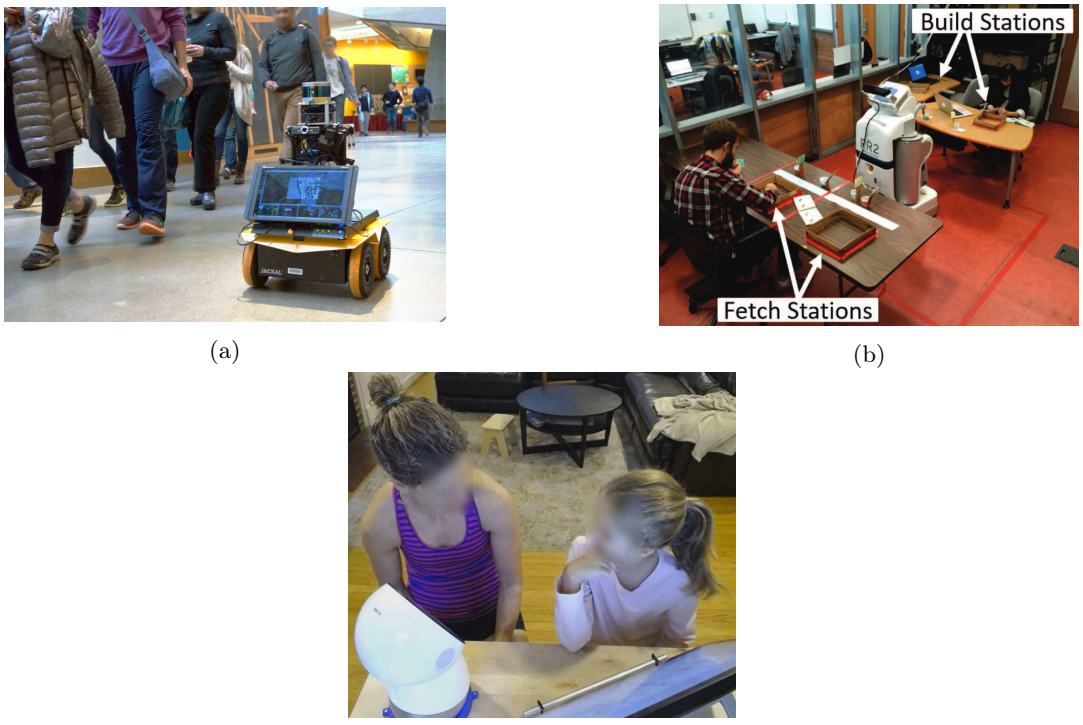
3.5.3 Generalisation

Generalisation is a significant problem in machine learning and single-agent RL, concerning the robustness of learnt models to situations unseen during training. In RL, this may correspond to different initial conditions or new environmental settings. A good model is one that maintains its training performance in these new situations. In the multi-agent setting, the problem persists and even evolves with multiple agents, requiring to handle changes in the strategies of other agents.

In machine learning, good generalisation is typically achieved through extensive training on very large amounts of data. However, this becomes challenging when faced with embodiment issues (see Section 3.3.2). In RL, the training data is generated by the agents themselves. Thus, in multi-agent RL, acquiring a comprehensive understanding of the joint policy space is a challenging exploration problem, as discussed in the previous section. In multi-agent RL, agents usually train in "self-play", with a fixed team of agents learning by trying to solve the task together. However, this often leads to agents converging to an arbitrary convention on collective behaviour, which may not hold with new partners. This is a problem in robotic settings, where the environment is dynamic and robots are expected to efficiently and safely handle new robotic or human partners.

Generalising to new environmental situations can be facilitated by having diverse environmental settings, such as procedural maps, which allow training in many different scenarios and, hopefully, learning more general policies (Cobbe et al., 2020; Jaderberg et al., 2019). For generalising to different partners, one approach is population-based training, involving a large number of agents trained in dynamic teams to face various strategies during training (Jaderberg et al., 2019; Liu, Lever, et al., 2019; Zhao et al., 2023). While this can be very effective (Jaderberg et al., 2019), it requires extensive training sessions to converge to general strategies. This may be impractical in high-dimensional and dynamic robotic environments, where training is expensive and accounting for all possible modifications of a real environment is impossible. Moreover, if humans are involved, it is impossible to train for all possible changes in human behaviour.

A promising approach is to learn to quickly adapt to any situation. In single-agent RL, *zero-shot generalisation* methods aim to adapt to unseen environmental settings without being retrained (Haarnoja et al., 2024; Kirk et al., 2023). In multi-agent RL, this concept extends to zero-shot, or *ad hoc*, teaming, where agents are evaluated with new partners (Stone et al., 2010). Agent modelling is promising for such settings, allowing to learn to model the "type" of policy observed in other agents (Rahman et al., 2023; Strouse et al., 2021; Xie et al., 2021; Yan et al., 2023), or even the exact agents faced (Barrett et al., 2017; Lanctot et al., 2023), allowing better reactions to the observed behaviour. If learned properly, zero-shot teaming can be a valuable tool for human-agent teaming. Training with humans is expensive, so it may be more efficient to learn to adapt quickly to any human partner (Shih et al., 2021; Strouse et al., 2021; Xie et al., 2021; Yan et al., 2023; Yu et al., 2023).



(a) (b)

(c)

Figure 3.7: Examples of human-robot interaction. (a) Navigation of a robot in a human-populated environment (Chen et al., 2017). (b) A robot assisting humans in a building task by fetching objects (Gombolay et al., 2017). (c) A robot helping a child with autism spectrum disorder to learn social interaction (Scassellati et al., 2018).

3.5.4 Interaction

Having robots in real environments implies the need for handling interactions with human beings. Figure 3.7 shows different examples of human-robot interaction (HRI), illustrating various levels of HRI with differing degrees of connection between robots and humans, ranging from simply living in a human-populated environment to deep social interactions between robots and humans. Across all these potential scenarios, we can identify three sub-problems of HRI. First, to enable interaction between humans and computer systems in general, we need to understand how humans behave. This means studying how they go about solving a task, to know how robots could help them and how they should not (Shih et al., 2021). This means understanding how they interact with each other, to understand what makes a successful human interaction and how robots could be a part of them (Tseng et al., 2016). This also means studying how humans react to robots when they interact with them, to understand the differences between human-human and human-robot interactions (Jung et al., 2020; Roesler et al., 2024). And, this means studying how humans communicate with each other, investigating different tools like natural language, body language, and a large variety of social cues (Feine et al., 2019). Analysing how humans behave in social interactions and when cooperating to fulfil a task can help design better robots and more effective learning approaches for HRI.

Second, HRI requires designing proper interfaces to enable smooth interactions. This involves physical interfaces, including the ability to sense, grasp, move, point and look at particular objects. These physical abilities are not needed only for interacting with human beings, but the HRI component might influence the design of

these skills. Social interfaces are also required, with the help of human-like features such as voice, eyes, articulated faces, and gestures (Złotowski et al., 2014). Being able to communicate with humans is also an important requirement to allow information sharing, teaching, strategy evaluation and correction (Crandall et al., 2018; Mikolov et al., 2018), and, more generally to bond more easily with artificial agents (Liu et al., 2022). In Chapter 5, we will explore in greater depth the role of communication and language for human-agent interaction.

Finally, HRI requires learning to behave around humans and cooperate with them. This is where MADRL research becomes relevant, as it enables learning in complex environments with multiple intelligent entities interacting. However, the main MADRL algorithms, discussed in Section 3.4, are not specifically intended for interacting with humans. They need to be extended using various techniques to efficiently address the problems linked with embodiment (see Section 3.3.2). The generalisation problem, described previously, must be tackled to enable robots to interact with any given human partner. And, to enable the use of MADRL algorithms on real robots, we need efficient approaches for bridging the reality gap (see Section 3.3.2).

3.6 Conclusion

This chapter explored the domain of MADRL research, from the perspective of robotics. One of the objectives of MADRL research is to learn behavioural policies for controlling robots in the real world. Thus, it is important to reflect on the current state of progress in this domain, analyse the relevance of recent studies for robotics, and see how the challenges of robotic environments are addressed. After formally defining the tools of MADRL, we have presented the main challenges faced in multi-agent learning and robotics. Some of these challenges are specific to one of the two areas, but many overlap on many aspects. This analysis shows that connecting these two domains means dealing with many different forms of complexity, in environmental settings, interactions, and unpredictable situations.

In Section 3.4, we presented a survey of the main avenues of MADRL research, describing how state-of-the-art methods learn policies in multi-agent settings. While some techniques have been shown to tackle increasingly complex tasks and environments, the main approaches are still far from being applicable in a robotic scenario without significant adaptation. This can be attributed to the fact that MADRL research is focused on the optimisation problem of multi-agent learning, i.e., finding an efficient multi-agent strategy. Doing so, it often overlooks some problems faced in realistic scenarios and thus fails to progress towards more efficient learning algorithms.

In Section 3.5, we introduced multiple challenges for MADRL research that should be addressed for improving the field, especially for moving towards robotic applications. The benchmarking issue faced in the domain is especially important to solve rapidly to ensure a more reliable research field and more steady progress. Next, the problems of exploration, generalisation, and interaction, are all key to improving the efficiency and applicability of new MADRL algorithms. We believe these challenges represent the main directions for advancing the control of robots in complex multi-agent environments, and that MADRL algorithms would benefit from investigating them further.

Following this analysis of MADRL algorithms, in the next chapters, we propose two new methods for tackling some of the challenges defined previously. First, in Chapter 4, we investigate the problem of exploration further, by highlighting the

weaknesses of existing MADRL algorithms and proposing a method for improving them by exploring in a coordinated fashion. Next, in Chapter 5, we tackle the problem of communication in multi-agent systems, and propose an algorithm for learning to communicate with a pre-defined language which, as we demonstrate, serves multiple purposes of the robotic setting.

Chapter 4

Joint Intrinsic Motivation

Inducing coordinated exploration in multi-agent environments.

*This chapter extends the conference paper "*Joint Intrinsic Motivation for Coordinated Exploration in Multi-Agent Deep Reinforcement Learning*", published in the Conference on Autonomous Agents and Multi-Agent Systems (AAMAS) 2024*[1].

4.1 Introduction

In Chapter 3, we reviewed the multi-agent deep reinforcement learning (MADRL) literature, its state-of-the-art approaches, and some important challenges that still need to be tackled for building better learning algorithms. One major issue in multi-agent learning is the problem of relative overgeneralisation (Wei & Luke, 2016; Wiegand, 2003) where agents struggle to find the optimal joint policy because local policies are attracted towards suboptimal areas of the search space. This makes most algorithms inefficient in tasks where the optimal strategy requires strong coordination among agents. Relative overgeneralisation can be described as a problem of exploration of the joint-observation space: as the success of the MAS depends on the coordination of multiple agents, exploring the joint-observation space is required to discover optimal joint behaviours. In this chapter, we address the question of how to explore the joint-state space to efficiently discover superior coordinated strategies for solving the task at hand.

In single-agent RL, the problem of exploration has been studied to solve hard exploration tasks where positive reward signals are sparse. One solution is to use intrinsic motivation (Lehman & Stanley, 2011; Oudeyer & Kaplan, 2007; Schmidhuber, 1991) to incite agents to explore unknown parts of the environment. In addition to the environment reward, agents are given an auxiliary reward related to the novelty of encountered states. Maximising this intrinsic reward leads agents to visit previously unexplored regions of the environment, ultimately discovering new solutions to the task. These methods have shown great success in helping RL agents solve hard exploration tasks (Badia, Sprechmann, et al., 2020; Pathak et al., 2017).

In the multi-agent setting, intrinsic objectives have also been studied to induce different kinds of behaviours in agents such as coordinated exploration (Iqbal & Sha, 2019a), social influence (Jaques et al., 2019; Wang, Wang, et al., 2020) or alignment with other agents' expectations (Ma et al., 2022). However, previous works have only used local observations to generate intrinsic rewards. With partial observability, local

[1]Full paper available at: https://arxiv.org/abs/2402.03972

observations often lack crucial information to fully understand the current configuration of the environment. In the context of exploration, an intrinsic reward based only on local observations will lead to each agent exploring their own observation space, without considering the current state of other agents. This can result in inefficient exploration in cooperative tasks where the success of the MAS depends on the coordination of all agents.

In this chapter, we introduce a novel multi-agent exploration approach called **Joint Intrinsic Motivation** (JIM) which can be combined with any MADRL algorithm that follows the centralised training with decentralised execution (CTDE) paradigm. JIM exploits centralised information during training to motivate agents to explore new coordinated behaviours. In order to compute joint novelty, JIM builds from two state-of-the-art single-agent intrinsic rewards: NovelD (Zhang et al., 2021) for exploring unknown parts of the environment, and E3B (Henaff et al., 2022) for having more diverse trajectories. Adding this auxiliary reward to the agents' objective incites them to diversify their collective behaviour until they have a fair knowledge of the environment and can focus on the main task at hand.

To demonstrate the advantages of our approach, we first design a simple test environment to showcase a clear example of relative overgeneralisation. We show that the state-of-the-art algorithm QMIX (Rashid et al., 2018) struggles in this scenario and that motivating the exploration of coordinated behaviour helps solve the task. Next, we validate these results in a continuous virtual environment, showing that coordination tasks benefit from joint exploration. Finally, further analysis is conducted to confirm the strength and scalability of our approach.

4.2 The Multi-Agent Exploration Problem

4.2.1 Random Exploration Strategies

To effectively learn with RL, algorithms need to balance an exploration-exploitation trade-off. Exploration of the environment is required to collect knowledge about the task. But, exploiting the learnt strategy allows focusing on parts of the environment that were discovered valuable, potentially making learning more efficient. Most RL methods tackle this issue by adding randomness in the agents' behaviour during training. Algorithms based on Q-learning use the ϵ-greedy strategy, described in Section 2.3.1, where agents start training by executing random actions and progressively shift to choosing actions only using the learnt action-value. In policy-based approaches, multiple exploration strategies are possible. In DDPG (Lillicrap et al., 2015) and the multi-agent version MADDPG (Lowe et al., 2017), exploration is performed by adding a Gaussian noise on the action generated by the deterministic formula: $a_t^i = \pi_i(o_t^i) + \epsilon$, with the noise $\epsilon \sim \mathcal{N}(0, \sigma)$ and σ a hyperparameter controlling the standard variation of the exploration noise. In PPO (Schulman et al., 2017) and MAPPO (Yu et al., 2021b), because the policy is stochastic, actions can be drawn randomly from the generated action distribution during training: $a_t^i \sim \pi(.|o_t^i)$. In addition to this, exploration is also handled at the policy learning level, by maximising the entropy of the policy in the PPO objective (see Equation 2.51).

All these approaches rely on randomness to ensure that agents gather essential knowledge about the environment and the task. However, in environments with very few positive reward signals, random exploration is not sufficient (Burda et al., 2019; Ostrovski et al., 2017; Pathak et al., 2017). In hard exploration problems, finding the positive reward signal requires specific sequences of hundreds of actions with no

	A	B	C
A	10	-5	-5
B	-5	7	7
C	-5	7	7

Figure 4.1: A social dilemma featuring the problem of relative overgeneralisation. Two agents have to choose between three actions. The optimal joint behaviour is both agents choosing action A. But, because actions B and C have a better average return when looked at locally, they will often be preferred.

guidance. Discovering these trajectories by acting randomly is extremely unlikely. And, to be able to learn an efficient behaviour, RL algorithms require experiencing this behaviour many times, learning a little bit from each experience. Thus, random exploration is unlikely to enable efficient discovery and learning of optimal strategies in hard exploration problems.

4.2.2 Relative Overgeneralisation

The hard exploration problem becomes even worse in MAS as the completion of a task depends on the actions of multiple independent agents. In the context of MAS, a task that requires strong coordination between multiple agents can be considered a hard exploration problem. As defined in Section 3.3.1, coordination requires the synchronisation of multiple agents' actions to achieve their common goal. Thus, a task requiring strong coordination is one where very few joint behaviours will produce the optimal returns. As in single-agent hard exploration problems, consistently discovering these optimal joint trajectories through random exploration will be extremely inefficient.

In such cases, agents learning by randomly exploring their local state-action spaces will often struggle to find the optimal coordination strategy and settle for an easier suboptimal joint strategy. This is a problem known as **relative overgeneralisation** (Wei & Luke, 2016; Wiegand, 2003), illustrated in Figure 4.1. In this example, there are two Nash equilibria: the optimal equilibrium with both agents choosing action A, and a suboptimal equilibrium where agents choose randomly between actions B and C with any probability. While the joint action (A, A) is clearly optimal, if agents explore their actions locally they will quickly find that actions B and C yield better returns on average. Only if all joint actions are explored and evaluated against each other, will the agents discover the optimal joint strategy. Thus, a joint-exploration approach can help solve the issue of relative overgeneralisation.

4.3 Explicit Exploration Strategies: Related Works

4.3.1 Exploration in Single-Agent Reinforcement Learning

To improve the exploration strategies of RL agents, multiple techniques have been employed. Inducing noise in learnt estimates can help RL agents having more diverse behaviour during training (Chiappa et al., 2023; Fortunato et al., 2018; Osband et al., 2016, 2018; Plappert et al., 2018). Hierarchical learning algorithms can be driven by the search for new skills, based on the idea that different skills generate different trajectories (Eysenbach et al., 2019; Gehring et al., 2021; Karol Gregor, 2017). Other works use a learnt model of the environment to plan exploration trajectories (Hu et al., 2023; Sekar et al., 2020; Shyam et al., 2019). Ecoffet et al., 2021 achieve great

results on hard exploration problems with a more handcrafted exploration method based on replaying past trajectories that resulted in unknown states, and starting exploring randomly from these states. Similarly, Pislar et al., 2022 devise a strategy to decide when agents should explore with random actions based on their knowledge of the environment.

However, the most successful approach for solving hard exploration problems has been inciting curiosity with intrinsic motivation. **Intrinsic motivation** is a technique for shaping the behaviour of RL agents by adding an auxiliary reward for them to maximise (Lehman & Stanley, 2011; Oudeyer & Kaplan, 2007; Schmidhuber, 1991). This reward is computed by the agent, hence the name "intrinsic reward", based on its current state. Curiosity can be induced by computing a reward based on the novelty of the agent's trajectory. By learning to maximise this curiosity objective, the agent should learn to search for novel states. For measuring novelty, several methods use the error of trainable prediction models. The *Intrinsic Curiosity Module* (ICM) (Pathak et al., 2017) trains a model of environment dynamics and uses the prediction error as a measure of novelty. *Random Network Distillation* (RND) (Burda et al., 2019) uses a neural network with fixed, randomly initialised parameters to produce a random encoding of the state, and trains a predictor network to generate the same encoding, the prediction error being the measure of novelty. With these two approaches, the prediction models will yield low novelty for states similar to what they have trained on, while producing high novelty for unknown parts of the environment. RIDE (Raileanu & Rocktäschel, 2020) and NovelD (Zhang et al., 2021) use respectively ICM and RND to compute a reward from the difference of novelty between the next state and the current state, pushing the agents to always seek novel states. Similarly, NGU (Badia, Sprechmann, et al., 2020) and E3B (Henaff et al., 2022) use clustering techniques to reward states that are distant from previously observed states. Agent57 (Badia, Piot, et al., 2020) manages to solve the hard exploration games in the Atari benchmark by extending NGU to learn two value functions: one for the extrinsic reward and one for the intrinsic one; and by learning a meta-controller that decides whether to explore or exploit during training. Finally, AGAC (Flet-Berliac et al., 2021) trains an adversarial policy to predict the main policy's output, the latter being rewarded with the former's prediction error.

4.3.2 Exploration in Multi-Agent Reinforcement Learning

To address the problem of relative overgeneralisation, better exploration strategies must be implemented in MADRL algorithms. Not all single-agent exploration strategies can be used efficiently in multi-agent settings. The noise-inducing approach could be another source of non-stationarity in a multi-agent algorithm. Also, planning-based techniques are not usable as learning a model in a multi-agent setting is challenging. However, some other approaches have been successfully pursued. MAVEN (Mahajan et al., 2019) extends QMIX with a hierarchical policy that chooses goals common to all agents, and then explores the space of these joint goals to try different joint behaviours. Similarly, CMAE (Liu et al., 2021) have agents sharing common goals, defined as states to reach, and use a curriculum approach for choosing joint goals of increased difficulty based on the number of times each goal-state has been experienced. Lupu et al., 2021 promote policy diversity by learning a population of agents and maximising the divergence of each policy with regards to the rest of the population, which is shown to help agents be more versatile. Finally, EMAX (Schäfer et al., 2023) learn multiple joint value functions at once and explore by maximising the disagreement of the ensemble of joint values.

4.3.3 Multi-Agent Intrinsic Motivation

In MASs, there is a large variety of different ways to complete tasks, interact with teammates, and handle opponents. Different definitions of intrinsic motivation can stimulate the emergence of different types of behaviour. Social influence has been shown to help learn interesting multi-agent behaviour (Jaques et al., 2019; Wang, Wang, et al., 2020), by rewarding actions that have a significant impact on other agents. Learning to influence other agents helps train agents that actively look for interactions, thus avoiding lazy agents, and that try new behaviours during training, helping the exploration of joint behaviours. Ma et al., 2022 propose an intrinsic reward based on the average alignment with other agents' expectations. Depending on how this alignment reward is used, as a bonus or a penalty, this can promote more predictable or more surprising behaviours.

Curiosity has also been studied for multi-agent exploration. Iqbal and Sha, 2019a propose an approach for coordinated exploration using several metrics for estimating the novelty of observations that depend on all agents' past experiences. Zheng et al., 2021 extend VDN with a separate network that predicts local action-values, with an intrinsic curiosity reward based on the prediction error of these separate values. Finally, in a concurrent work, COIN (Li, Kuang, et al., 2023) proposed an intrinsic reward inspired by ICM that measures the novelty of the joint trajectories. Apart from COIN, other multi-agent intrinsic rewards use local observations for computing their intrinsic rewards. As shown in Section 4.2.2, local exploration is not sufficient for exploring coordinated behaviours. Therefore, in the following sections, we present an approach for exploring the space of joint observations with an intrinsic reward inspired by recent works in single-agent curiosity.

4.4 Background on Intrinsic Reward Definitions

In Section 4.3, we introduced intrinsic motivation as a way to incite agents to actively explore their environment. To this end, at each time step t, agents receive an augmented reward $r_t = r_t^{\text{ext}} + \beta r_t^{\text{int}}$, where r_t^{ext} is the extrinsic reward given by the environment, r_t^{int} is the intrinsic reward, and β is a hyperparameter controlling the weight of the intrinsic reward in the agents' objective. We describe three methods of intrinsic motivation from the literature that we will use later in Section 4.5.

4.4.1 Random Network Distillation (RND)

In *random network distillation* (RND), Burda et al., 2019 compute novelty using two neural networks with the same architecture: a target network ϕ and a predictor network ϕ'. The target's parameters are initialised randomly and fixed. It takes as input the state s_t and produces a random embedding $\phi(s_t)$. The predictor is trained to output the same embedding, minimising the Euclidean distance:

$$RND(s_t) = \|\phi(s_t) - \phi'(s_t)\|_2. \tag{4.1}$$

By training the predictor network to produce the same embeddings as the one generated by the target, the algorithm essentially tries to copy the parameters of the target in the predictor, or "distil" them. This distance $RND(s_t)$ is also used as a measure of the novelty of state s_t and is given as an intrinsic reward to the agent. By training the predictor network along the agent, the predictor will get better at predicting

the embeddings of states that are often seen by the agent during its trials. Thus, to accumulate more intrinsic reward, the agent will have to discover new states.

4.4.2 Novelty Diversity (NovelD)

Zhang et al. (Zhang et al., 2021) build upon RND to devise a novelty criterion termed *NovelD*, for "novelty diversity". It is defined as follows:

$$N(s_t, s_{t+1}) = \max[RND(s_{t+1}) - \alpha RND(s_t), 0] \times \mathbb{1}\{N_e(s_{t+1}) = 1\}, \tag{4.2}$$

with α a scaling factor, N_e an episodic count of visited states, and $\mathbb{1}\{\cdot\}$ the indicator function that outputs 1 if the condition is true. The first part is the core of the novelty criterion. It uses RND to reward agents for positive gains in novelty between the current and the next states. Thus, NovelD rewards the agent for always going towards newer states. The second part is an episodic restriction that ensures the reward is given only when state s_{t+1} is observed for the first time in this episode. This ensures that agents cannot exploit one particularly novel state by staying in it. However, because this restriction is based on an explicit count of the occurrence of states during the episode, it limits the use of NovelD to discrete state spaces, as continuous states will likely never be reached more than once.

4.4.3 Exploration via Elliptical Episodic Bonuses (E3B)

Henaff et al., 2022 propose *exploration via elliptical episodic bonuses* (E3B), an episodic bonus based on the position of the observed state with respect to an ellipse that fits all states previously encountered in the current episode. Formally, it is computed as follows:

$$b(s_t) = \psi(s_t)^\top C_{t-1}^{-1} \psi(s_t), \tag{4.3}$$

with

$$C_{t-1} = \sum_{i=1}^{t-1} \psi(s_i)\psi(s_i)^\top + \lambda I, \tag{4.4}$$

where I is the identity matrix and λ a scalar coefficient. ψ is an embedding network trained using an inverse dynamics model (Pathak et al., 2017): embeddings of following states $\psi(s_t)$ and $\psi(s_{t+1})$ are used by a separate neural network trained to predict the action a_t taken between these states. As a result of this training process, ψ encodes parts of the observation that are controllable by the agents (see the original paper for more detail, Henaff et al., 2022). Intuitively, b can be understood as a generalisation of a count-based episodic bonus for a continuous state space. States that are close to previously encountered states in the current episode will yield low bonuses, whereas states that are very different will produce high bonuses. This incites the agent to have diverse trajectories.

4.5 Joint Intrinsic Motivation Algorithm

In this section, we introduce the Joint Intrinsic Motivation (JIM) exploration criterion for coordinated multi-agent exploration. We take inspiration from state-of-the-art intrinsic motivation techniques developed for single-agent RL and propose a new method for efficient multi-agent exploration. First, we define the main components of the intrinsic reward. Then, we explain how it is used in a multi-agent setting for exploring the space of joint observations.

4.5.1 Double-timescale Intrinsic Reward

Similarly to previous works on single-agent intrinsic motivation (Badia, Sprechmann, et al., 2020), we define a novelty metric that combines two exploration criteria working at different timescales:

- A **life-long exploration criterion (LLEC)** that captures how novel is the current observation with respect to all observations since the beginning of training.
- An **episodic exploration criterion (EEC)** that captures the difference between the current observation and all previous observations in the current episode.

Intuitively, the *life-long reward* motivates agents to search for never-experienced parts of the environment. Meanwhile, the *episodic bonus* induces more diverse trajectories. These two elements will work together to reinforce agents to efficiently explore their environment.

We first define this intrinsic reward as it would be in a single-agent case, measuring the novelty of an agent's state. For each transition from state s_t to the next state s_{t+1}, we define the double-timescale intrinsic reward as follows:

$$r_t(s_t, s_{t+1}) = N_{LLEC}(s_t, s_{t+1}) \times N_{EEC}(s_{t+1}), \tag{4.5}$$

with the life-long novelty N_{LLEC} inspired from NovelD (Zhang et al., 2021) (see Eq. (4.2)):

$$N_{LLEC}(s_t, s_{t+1}) = \max[RND(s_{t+1}) - \alpha RND(s_t), 0], \tag{4.6}$$

with α a scaling factor and RND the novelty measure (see Eq. (4.1)). Further, the episodic novelty N_{EEC} uses the bonus from E3B (Henaff et al., 2022) (see Eq. (4.3)):

$$N_{EEC}(s_{t+1}) = \sqrt{2b(s_{t+1})}. \tag{4.7}$$

We make two modifications compared to previous works. First, we modify NovelD by removing the count-based episodic restriction that was limited to discrete state spaces. We replace it with the elliptical episodic bonus b from E3B (Henaff et al., 2022). This bonus acts as an episodic restriction by scaling N_{LLEC} up or down, depending on the novelty of the current state compared to what has been observed during the current episode. Second, we take $\sqrt{2b(s_{t+1})}$ instead of simply b to smooth the values given by this bonus, increasing the small ones and decreasing the large ones. This is because we observed that, in continuous state spaces (which was not studied in the original E3B paper), b yielded extremely high bonuses at the start of the episode and quickly dropped to low bonuses after this. Thus, the smoothing function allows having a more controlled range of bonuses.

Combining these two rewards makes it possible to take the benefits of both. N_{LLEC} pushes agents to explore regions of the state space that are not well-known to agents, considering states observed since the beginning of training. Meanwhile, N_{EEC} favours diverse trajectories, inciting agents to always seek new observations during a single episode. As the agents explore their environment, the prediction error of RND (see Eq. (4.1)) slowly decreases. Thus, N_{LLEC} decreases as well, tending toward zero. This allows to naturally shift from high exploration at the beginning of training, to progressively focusing on the extrinsic reward. Finally, as the episodic restriction does not rely on any explicit count of visited states, it can be used in continuous state spaces.

4.5.2 The Joint Intrinsic Motivation Algorithm

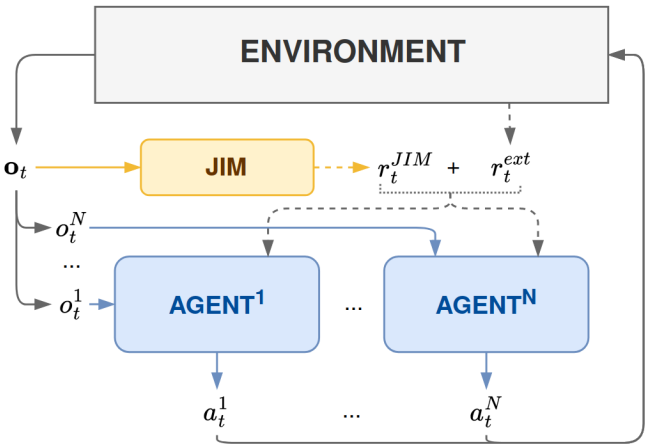


Figure 4.2: Architecture for the Joint Intrinsic Motivation (JIM) algorithm. JIM has only one intrinsic motivation module for the whole multi-agent system, computing novelty of the joint observation $\mathbf{o}_t$. However, agents only use their local observation to choose their actions.

Building from the intrinsic reward introduced previously, we propose the **Joint Intrinsic Motivation** (JIM) algorithm to incite MADRL agents to explore the joint-observation space. At each time step, all agents receive the same global reward $r_t = r_t^{\text{ext}} + \beta r_t^{JIM}$, where r_t^{ext} is the extrinsic reward given by the environment, r_t^{JIM} is our joint exploration criterion, and β is a hyperparameter controlling the weight of the intrinsic reward. The exploration criterion in JIM uses the double-timescale intrinsic reward defined previously to compute the novelty of the joint observation:

$$r_t^{JIM}(\mathbf{o}_t, \mathbf{o}_{t+1}) = N_{LLEC}(\mathbf{o}_t, \mathbf{o}_{t+1}) \times N_{EEC}(\mathbf{o}_{t+1}), \tag{4.8}$$

where $\mathbf{o}_t = \{o_t^i\}_{0 \leq i \leq N}$, i.e., the concatenation of all local observations. Figure 4.2 shows the architecture for JIM. Compared to a local method that would use one intrinsic motivation module per agent, JIM computes only one intrinsic reward. This makes it possible to capture novelty at the team level, rather than at the individual level only, while requiring fewer parameters and less computation. As agents are rewarded by the novelty of the joint observation, they will learn to search for new combinations of observations with other agents of the system, rather than only exploring their local-observation space. This will induce the exploration of new configurations of the environment and thus help find better coordinated strategies.

As JIM uses joint observations for computing the intrinsic reward, it can be associated with any MADRL algorithm that fits in the CTDE paradigm. These algorithms usually employ a centralised value function (Lowe et al., 2017; Rashid et al., 2018; Yu et al., 2021b) that looks at the joint observation to predict the value of the agents' actions. Such centralised value functions will be able to associate rewards provided by JIM to new configurations in the joint observation space, thus inducing agents to actively search for these configurations.

One could note that the joint observation has two notable drawbacks: the number of dimensions grows linearly with the number of agents and there is a risk of capturing redundant information. These issues are both alleviated by using embedding networks to encode the joint observation into a more condensed latent representation. Both N_{LLEC} and N_{EEC} use embedding networks, respectively ϕ and ψ (as described in

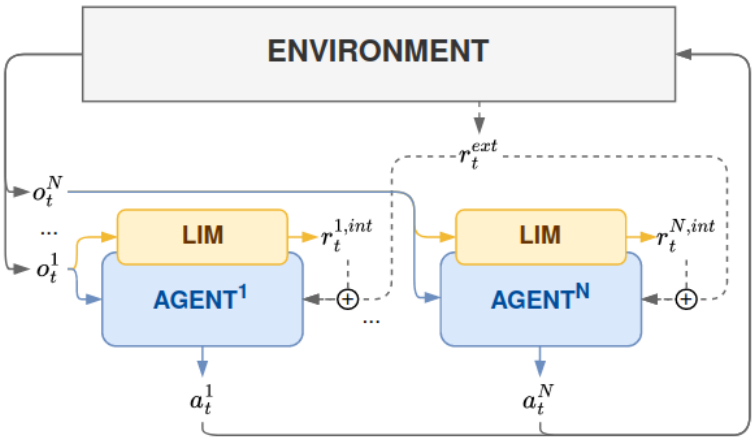


Figure 4.3: Architecture for local intrinsic motivation (LIM), used as a baseline. Each agent has its own module for computing an intrinsic reward based on its local observation.

Section 4.4), to encode the joint observation. This allows for a more controllable number of parameters in JIM, as only the dimension of the input layers of ϕ and ψ depend on the size of the joint observation. Furthermore, embedding networks learn to cast away useless or redundant information in order to produce a more compact representation of the joint observation. Note that both ϕ and ψ were originally used (respectively in RND (Burda et al., 2019) and E3B (Henaff et al., 2022)) with raw pixel images as input, showing the significant capability of dimensionality reduction of these techniques.

4.6 Implementation Details

In the next section, we use JIM with QMIX (Rashid et al., 2018). We use the default QMIX architecture and hyperparameters, along with prioritised experience replay (Schaul et al., 2016). In all experiments, we compare three algorithms:

- **QMIX+JIM**, augmenting QMIX with joint exploration, as shown in Figure 4.2 and described in Section 4.4.
- **QMIX+LIM**, a degraded version of QMIX+JIM where local intrinsic motivation is used. Each agent generates its own intrinsic reward based solely on its local observation, using the same reward definition as JIM (see Section 4.4). The architecture for LIM (Local Intrinsic Motivation) is described in Figure 4.3.
- The original state-of-the-art **QMIX** algorithm (Rashid et al., 2018) with no intrinsic motivation, used as a baseline.

Note that the only difference between these three algorithms is the definition of the reward function given to each agent during training. The actual training and execution algorithms are identical.

To ensure a fair comparison between JIM and LIM, we use different values for some specific hyperparameters in the two versions in order for them to have a similar number of trainable parameters. All hyperparameters used in our experiments are listed in Appendix A.1. The code used to run all experiments is freely available online[2].

[2] https://github.com/MToquebiau/Joint-Intrinsic-Motivation

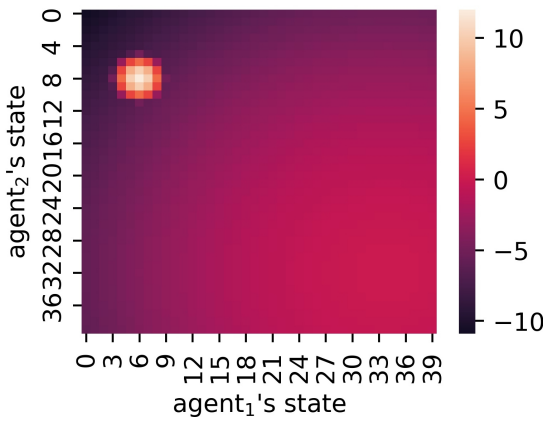


Figure 4.4: Heat map of the rewards given for each state in the two-dimensional climbing game. Each axis show the positions of one agent. The combined position of the two agents yields a reward that is defined by Equation 4.9. We distinguish clearly the high reward spike in the top-left of the environment and the low reward plateau in the bottom-right. The optimal strategy is to navigate towards the joint position corresponding to the high reward spike.

4.7 Experiments

In this section, we present a set of experiments to evaluate the exploration criterion of JIM when used along the state-of-the-art QMIX algorithm (Rashid et al., 2018). First, we show the results in a synthetic discrete environment where the problem of relative overgeneralisation can be artificially tuned and observe that JIM helps alleviate this issue. Then, we test our approach on pseudo-realistic robotic tasks in a continuous environment and show that exploring the joint-observation space helps solve cooperative tasks with strong coordination requirements. Next, we present an ablation study by comparing JIM with two simpler versions that each lack one of the two exploration criteria described in Section 4.4, showing the advantage of combining the two. We show that JIM remains efficient with more than two agents. And, finally, we show that the centralised intrinsic motivation is more computationally efficient than the local version.

4.7.1 Addressing Relative Overgeneralisation

Environment Definition

To demonstrate how joint exploration helps solve the problem of relative overgeneralisation, we design a simple test environment that expands the example shown in Figure 4.1. This environment is a two-dimensional climbing game, inspired by previous works (Wei et al., 2018). In this environment, two agents can move on a discrete one-dimensional axis with D possible positions. The two agents are denoted by their position, namely x (for the first agent) and y (for the second agent). At each time step, agents observe their position as a one-hot vector (e.g., for agent x, $o_t^{\mathrm{x}} = \{o_t^{\mathrm{x},i} = 1 \text{ if } \mathrm{x} = i,\ 0 \text{ otherwise}\}_{0 \leq i < D}$) and can choose between three actions:

move in one direction or the other, or stay in position. They receive a reward corresponding to their combined position:

$$r_t^{\text{ext}}(x, y; \delta) = \max\Big(R^+ - \frac{\delta}{D}\big[(x - r_x^+)^2 + (y - r_y^+)^2\big], \\ R^- - \frac{1}{8D}\big[(x - r_x^-)^2 + (y - r_y^-)^2\big]\Big). \tag{4.9}$$

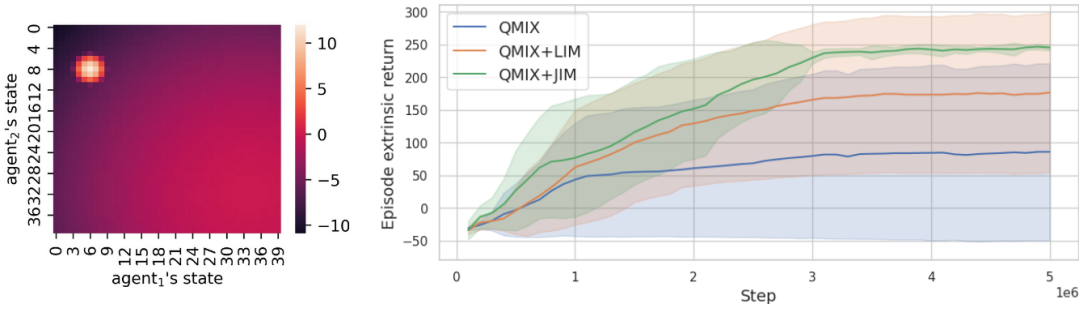
The result of this formula is displayed in Figure 4.4. The reward combines two hyperboles in opposite corners: one narrow that culminates at R^+ at position (r_x^+, r_y^+), and another much wider that plateaus at R^- at position (r_x^-, r_y^-). We set the optimal reward $R^+ = 12$ and the suboptimal $R^- = 0$. The width of the optimal reward spike is controlled by the parameter δ: a higher δ value yields a narrower spike.

At the beginning of each episode, the agents are placed randomly on their respective axes. Each episode lasts D steps, ensuring that all starting positions can reach all other positions on the axis. The goal of the agents is to find where to go to maximise the global return. The wide suboptimal hyperbole is deceptive as it is an obvious path for agents to minimise their loss. The optimal reward spike is difficult to find because it covers a small portion of the state space and is surrounded by very low rewards, but it guarantees much greater returns. We can vary the difficulty of the task by changing the width of this optimal reward spike: the narrower the spike, the harder it is to find.

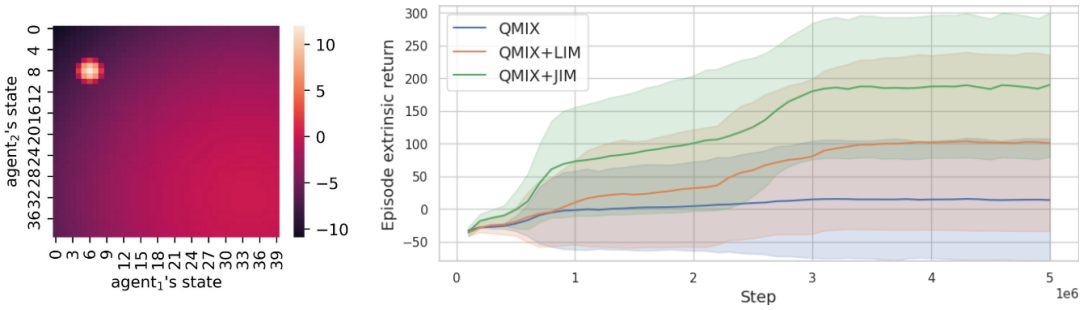
In this environment, we expect MADRL methods to struggle to find the optimal reward spike. Exploring local states could help but would not be sufficient to consistently solve the task. As the dimension D of the local-state space is fairly small, local novelty rewards will quickly vanish and will not help agents find the optimal reward spike. Exploring the joint-observation space adequately is required in order to consistently find optimal rewards. As JIM will reward exploration until all combined positions (x, y) are visited several times, agents will visit the optimal reward spike more often, thus helping them to learn the optimal coordinated strategy.

Results

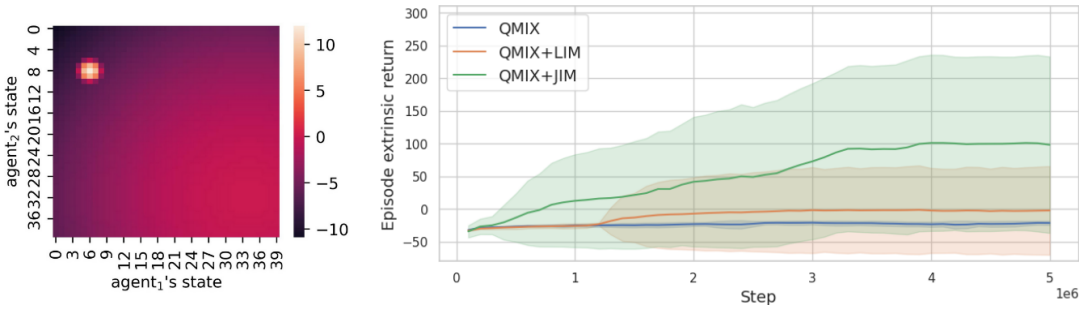
The results shown in Figure 4.5 confirm the hypotheses formulated in the previous section. We show the performance of QMIX, QMIX+LIM, and QMIX+JIM across 15 independent runs each. Further, we present results in three difficulty levels dictated by the width of the optimal reward spike. The results clearly demonstrate the importance of exploring the joint-state space. QMIX alone manages to get a positive reward on the easy scenario, but its performance is lower and with a larger standard deviation compared to the two other algorithms, showing that some runs did not manage to find the optimal strategy. In the harder scenarios, QMIX's performance degrades strongly, never finding any positive reward in the hardest case. JIM clearly improves the performance of QMIX. In the easy scenario, QMIX+JIM consistently goes for the optimal reward spike. In the harder settings, it still performs well on average, even in the "very hard" scenario where the optimal reward spike covers only 0.013% of all combined positions. The results of QMIX+LIM show that exploring the local-observation space helps agents find the optimal reward spike more often. However, it performs worse than JIM as it does not ensure that all combined positions are sufficiently explored. This shows that exploring the joint-observation space is crucial to allow agents to discover optimal coordinated behaviours.



(a) $\delta = 30$



(b) $\delta = 40$



(c) $\delta = 50$

Figure 4.5: Performance of variants of QMIX in the climbing game, with three levels of difficulty. Each row corresponds to a different difficulty level, where difficulty is controlled by the width coefficient of the optimal reward spike δ (as defined in Eq. (4.9)). On the left are the heat maps representing the reward function in each instance. Increasing δ leads to a smaller optimal reward spike, thus making the task more difficult. On the right is shown the performance during training of QMIX with no intrinsic reward (QMIX), local intrinsic motivation (QMIX+LIM), and joint intrinsic motivation (QMIX+JIM) (mean and standard deviation shown for 15 runs each). We see that a slight decrease in the size of the optimal reward spike results in a considerable increase in the difficulty of the task.

4.7.2 Coordination Task in a Continuous Environment

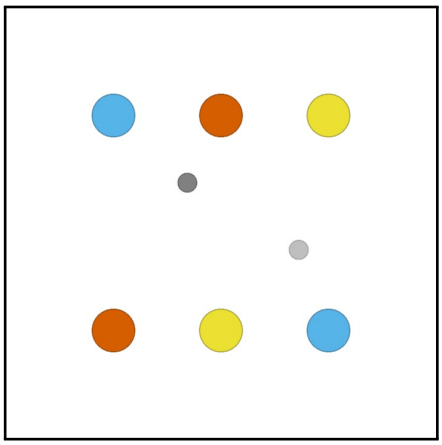


Figure 4.6: Coordinated placement task, agents are the small grey circles, the coloured circles represent landmarks the agents have to navigate on to gain rewards.

Environment Definition

Next, we study how JIM scales to more realistic continuous environments and more complex tasks. We use the multi-agent particle environment (MPE; Lowe et al., 2017) to simulate a cooperative robotic task that requires a high degree of coordination. The state space of MPE is continuous: agents receive as observation a vector with their position in the two-dimensional space and relative positions and velocities of other entities. Agents navigate in a closed two-by-two-meter area by choosing between five discrete actions: move in any four cardinal directions or stay in place. The observation range is limited: agents only have information about entities that are in a range of 60 centimetres around them.

We design a coordination task, named "*coordinated placement*", where agents must position themselves over landmarks in order to maximise their returns. As shown in Figure 4.6, there are two sets of three coloured landmarks. The reward given at each time step depends on the placement of agents on the landmarks. The optimal state is having both agents placed on the orange landmarks, yielding a reward of +10 at each time step. The blue and yellow landmarks act as deceiving rewards, yielding much smaller rewards (+2 for blue, +1 for yellow). To increase the deceiving aspect of the blue and yellow landmarks, we also reward agents collectively by +0.5 if only one of them stands on one of these two colours. This leads to a relative overgeneralisation situation, as they will locally find that going on blue or yellow landmarks systematically leads to a reward, while this is not the case for orange. Only if agents explore their environment in a coordinated fashion, will they discover that they need to be both on orange to get the optimal reward signal. Importantly, this scenario features partial observability, with agents only having information about entities close to them. This means that agents do not necessarily see which landmark the other agent goes to.

In this task, the observations of each agent consist in:

- its personal information: its position and velocity $\texttt{pos}_{\texttt{self},x}$, $\texttt{pos}_{\texttt{self},y}$, $\texttt{vel}_{\texttt{self},x}$, $\texttt{vel}_{\texttt{self},y}$,

- information about the other agent: a boolean indicating if the other agent is visible or not, the relative position, and the velocity of this agent $\texttt{is\_visible}_{\texttt{agent}}$, $\texttt{dist}_{\texttt{agent},x}$, $\texttt{dist}_{\texttt{agent},y}$, $\texttt{vel}_{\texttt{agent},x}$, $\texttt{vel}_{\texttt{agent},y}$,
- for each landmark in the environment: a boolean indicating if the landmark is visible or not, the relative position of this landmark, and its colour as a one-hot encoding: $\texttt{is\_visible}_{\texttt{landmark}}$, $\texttt{dist}_{\texttt{landmark},x}$, $\texttt{dist}_{\texttt{landmark},y}$, $\texttt{is\_red}$, $\texttt{is\_blue}$, $\texttt{is\_yellow}$.

Thus, the observation is a vector of dimension 43 containing this information. Relative positions of other entities (agent or landmarks) are actually the distance to the agent, normalised by their range of observation, i.e.,

$$\texttt{dist}_{\texttt{agent},x} = \frac{\texttt{pos}_{\texttt{agent},x} - \texttt{pos}_{\texttt{self},x}}{\texttt{obs\_range}}.$$

Results

Experiments in the coordinated placement task demonstrate well the importance of exploring in a coordinated fashion. Figure 4.7 shows the performance across training of QMIX, QMIX+LIM, and QMIX+JIM, with 11 independent runs each. On the right is displayed the performance of each independent run at the last iteration of training. This helps to visualise the multiple modes in potential returns in this scenario. The coloured dashed lines give an insight into the level of strategy learnt by each run. These levels of strategy can be visualised with example trajectories displayed in Figure 4.8.

QMIX alone almost always goes for the blue landmarks, while sometimes settling for the yellow ones. This indicates that without actively exploring the environment, QMIX gets stuck because of deceptive rewards and is unable to find the optimal strategy. While QMIX+LIM seems slightly better than QMIX on the training curves, the individual run performance shows that LIM arguably performs worse. Two runs manage to find the optimal strategy, but LIM often performs poorly with only one agent on a blue or yellow landmark. This demonstrates that exploring the space of local observations can be helpful, as it pushes agents to explore the environment. However, it can also be misleading as they do not contain all the information about the current state of the environment. With JIM, exploring the joint-observation space clearly improves the quality of the chosen strategies. More than half of the time, QMIX+JIM finds the optimal reward signal and learns an effective strategy to go on orange landmarks, showing that JIM allows for more efficient exploration of coordinated behaviours. When agents do not find the optimal strategy, they stick with the best sub-optimal strategy to go both on blue. This shows that agents benefit from exploring the space of joint observations as they are directly linked to the obtained reward, whereas local observations lack crucial information to understand the global reward.

The high standard deviation in the graph of Figure 4.7 is justified by the extreme gap in returns produced by the different levels of strategy. Note that this task is extremely sparse:

- from a local perspective, because no guidance is given to agents to navigate towards any landmark;
- and from a joint perspective, as the optimal joint strategy is extremely marginal.

The deceptive rewards given by standing on blue or yellow make the suboptimal strategies very attractive. Without an explicit exploration strategy, QMIX is always attracted to the local optimum, similar to results found in the climbing game. Proper

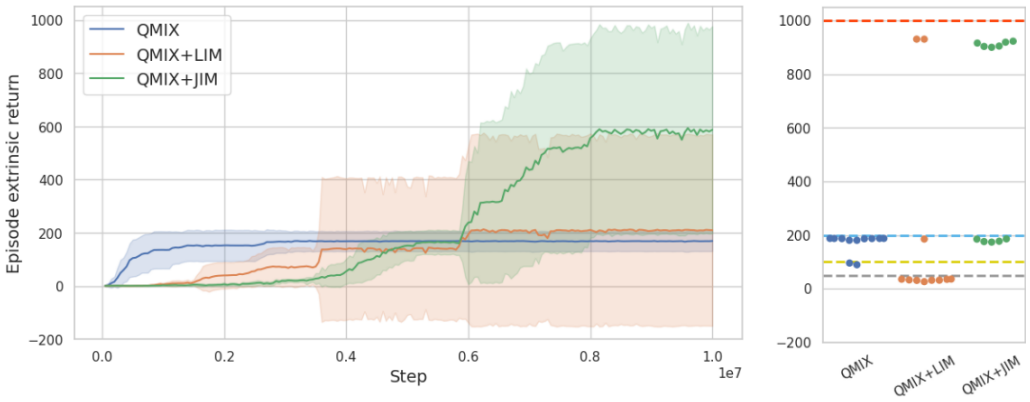
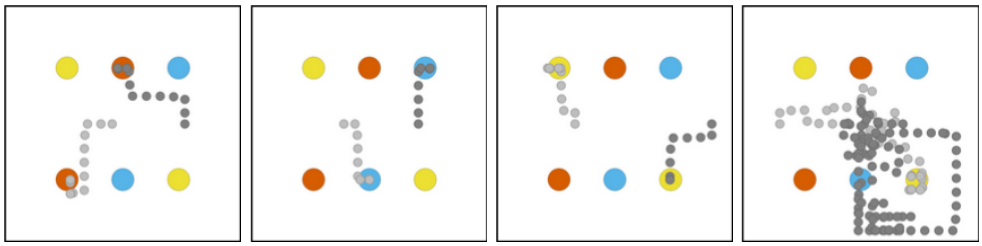


Figure 4.7: Performance of the three variants of QMIX in the coordinated placement task. The left shows the training curves with the mean and standard deviation across 11 independent runs each. The right graph displays the performance of each independent run at the last iteration of training. Dashed lines indicate the level of return obtained with different strategies: orange, blue, and yellow lines represent the return obtained if both agents are on landmarks of the related colour during 100 steps (the duration of an episode), and the grey line is for when only one agent is either on blue or yellow.



(a) (b) (c) (d)

Figure 4.8: Examples of trajectories in the coordinated placement task. Each image displays a different level of strategy, with (a) > (b) > (c) > (d): (a) optimal strategy with both agents on orange, (b) both on blue, (c) both on yellow, and (d) one on blue/yellow.

multi-agent exploration allows JIM to significantly improve the chances for QMIX to find the optimal strategy.

4.7.3 Further Analysis

Ablation Study

Next, we propose an ablation study to compare JIM with two derived versions of the reward: one with only the *episodic exploration criterion* N_{EEC} (JIM-EEC) and one with only the *life-long exploration criterion* N_{LLEC} (JIM-LLEC). Note that JIM-LLEC is actually equivalent to NovelD (Zhang et al., 2021) in this environment as the episodic restriction of NovelD (see Section 4.4) would be ineffective in a continuous environment such as MPE.

Results Figure 4.9 shows the results of training these versions in the coordinated placement task, with 11 independent runs each. Both ablated algorithms perform significantly worse than JIM. First, the episodic bonus of JIM-EEC alone lacks the motivation for discovering unseen configurations. Thus, it explores less and is not able to find the optimal solution to the task. Meanwhile, without the episodic restriction,

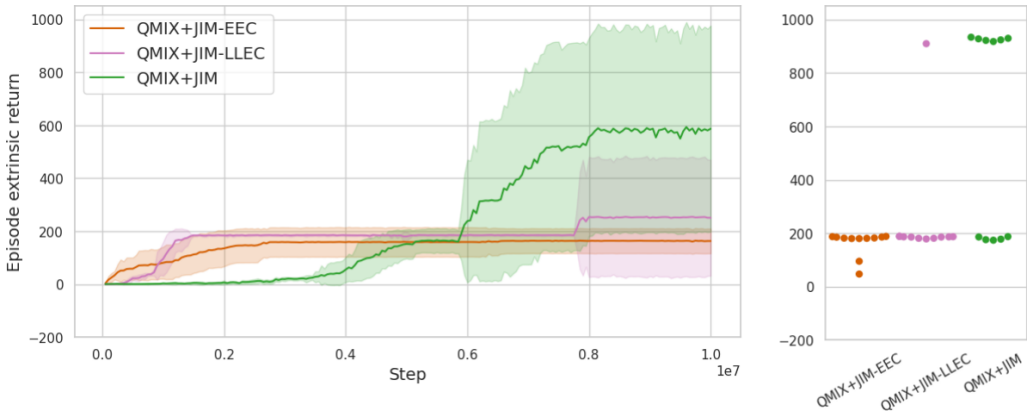


Figure 4.9: Ablation study of JIM in the coordinated placement task. The left graph shows the training curves with the mean and standard deviation across 11 independent runs each. The right graph displays the performance of each independent run at the last iteration of training. The two ablated versions only feature one of the two exploration criteria defined in Section 4.5: JIM-EEC for N_{EEC} and JIM-LLEC for N_{LLEC}. The results show the importance of combining the two criteria.

JIM-LLEC is less driven to have diverse trajectories, hindering its exploration abilities. This confirms that, as shown in a recent study (Andres et al., 2022), the episodic restriction implemented in NovelD and other intrinsic rewards (Badia, Sprechmann, et al., 2020; Raileanu & Rocktäschel, 2020) is crucial for developing efficient exploration strategies. Overall, this proves the importance of combining the two stages of exploration defined in N_{LLEC} and N_{EEC}.

Scaling Up to More Agents

We evaluate JIM in a scenario with four agents using a modified version of the two-dimensional climbing game introduced in Section 4.7.1. We modify the reward definition given Equation 4.9 to allow N dimensions:

$$r_t^{\text{ext}}(\mathbf{p};\delta) = \max\left(R^+ - \frac{\delta}{D}\sum_{i=0}^{N}(p_i - r_i^+)^2, R^- - \frac{1}{8D}\sum_{i=0}^{N}(p_i - r_i^-)^2\right),$$

with $\mathbf{p} = \{p_i\}_{0<i\leq N}$ the positions of the agents, δ the coefficient controlling the size of the optimal reward spike, D the dimension of each agent's state, R^+ the maximum value of the optimal reward spike placed at position $\mathbf{r}^+ = \{r_i^+\}_{0<i\leq N}$, and R^- the maximum value of the suboptimal plateau placed at position $\mathbf{r}^- = \{r_i^-\}_{0<i\leq N}$. This formula yields the same results as the two-dimensional example shown in Figure 4.4 but in a N-dimensional space.

We use this extended version to experiment with a four-agent climbing game. As in the two-agent version, the reward function has a high reward spike in one corner of the space and a low reward plateau in the other corner, making relative overgeneralisation prone to arise. With more agents, the number of dimensions of the joint observation increases linearly (in this case: from 80 dimensions with two agents to 160 with four agents) while the number of possible states increases exponentially, making the search for the optimal strategy significantly more challenging.

To compensate for the increased size of the state space, we have to make the optimal reward spike larger for the task to be solvable by QMIX. In the four-agent experiments, we use $\delta = 0.9$. Also, with four agents we had to lower the initial value

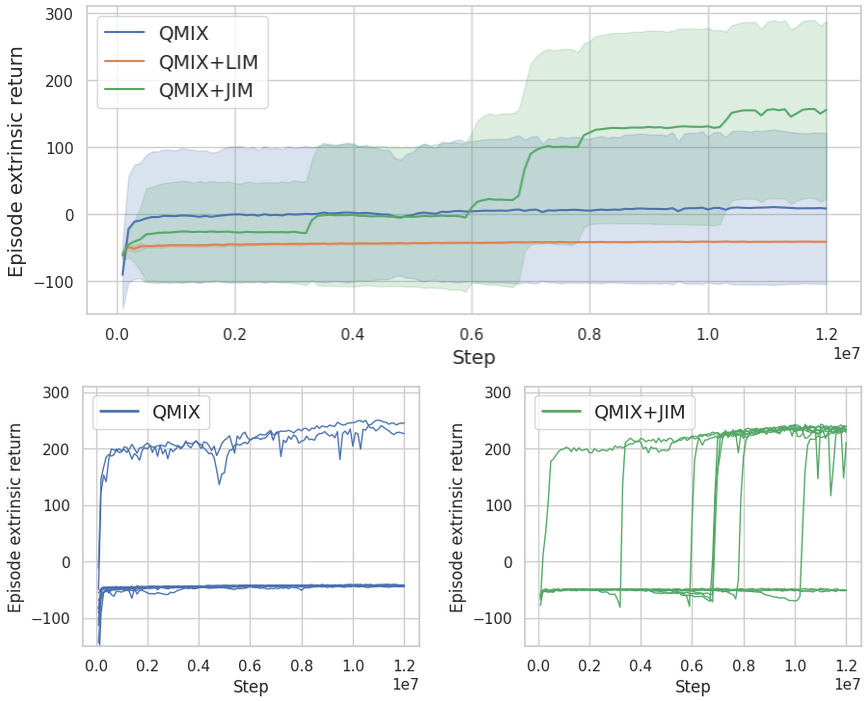


Figure 4.10: Training performance of QMIX, QMIX+LIM and QMIX+JIM in the four-agent climbing game. The top graph shows the mean and standard deviation across 11 runs each, the bottom graphs display all single runs for QMIX (left) and QMIX+JIM (right).

of the ϵ parameter of QMIX for its ϵ-greedy strategy. We found that increasing the number of agents led to bad results with the default 0.3 initial value of ϵ. With this hyperparameter set to 0.1, the results were significantly better. This is likely due to the fact that agents choose separately if they explore or exploit, meaning that increasing the number of agents leads to more randomness in the selection of each joint action.

Results Figure 4.10 shows the results in this scenario for 11 independent runs each. As with previous experiments, JIM improves the performance of QMIX by upgrading its exploration capabilities. Taking a closer look at the results reveals that QMIX and QMIX+LIM find the optimal solution in respectively 2 and 0 runs out of the 11, while QMIX+JIM finds the optimal solution in 8 out of 11 runs in the allocated time (see Figure 4.10-bottom). The two positive results for QMIX can be attributed to beneficial initial conditions as QMIX never reaches the optimal performance otherwise. This is not the case with QMIX+JIM, which shows robustness to initial conditions thanks to active exploration of the environment. In fact, the impact of JIM becomes evident when looking at curves from individual runs, where we consistently observe significant performance enhancements subsequent to a minor initial drop in efficiency. This phenomenon reflects a deliberate shift towards exploring novel approaches when the system would otherwise remain stagnant. This is unique to JIM, as QMIX+LIM never succeeds in finding the optimal solution, advocating for the benefits of using a global, rather than local, intrinsic reward for exploration.

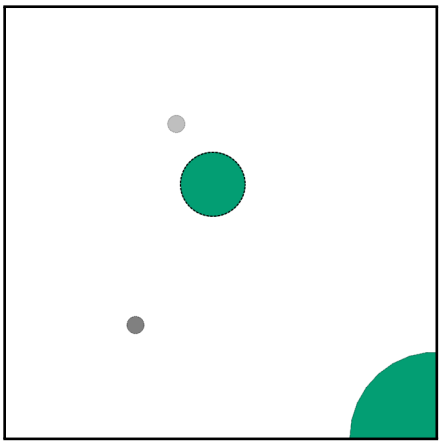


Figure 4.11: Cooperative box pushing task, agents are the small grey circles, the green circle in the middle is an object to deliver to the landmark in the bottom right corner.

Cooperative Task with no Coordination Need

To study coordinated exploration further, we experiment with a cooperative task that depends less on coordination. We design a cooperative box-pushing task that requires agents to push an object and place it on top of a landmark. Figure 4.11 shows a screenshot of this scenario. At the start of each episode, the landmark is randomly placed in any one of the four corners. The initial positions of the agents and the object are randomly set. If the agents manage to push the object and place it on the landmark, the episode ends early and they receive a reward of +100. Agents also get a small penalty of -1 at each time step to reward faster strategies.

In this setting, observations are defined as:

- its personal information: its position and velocity $\texttt{pos}_{\texttt{self},x}$, $\texttt{pos}_{\texttt{self},y}$, $\texttt{vel}_{\texttt{self},x}$, $\texttt{vel}_{\texttt{self},y}$,
- information about the other agent: a boolean indicating if the other agent is visible or not, the relative position, and the velocity of this agent $\texttt{is\_visible}_{\texttt{agent}}$, $\texttt{dist}_{\texttt{agent},x}$, $\texttt{dist}_{\texttt{agent},y}$, $\texttt{vel}_{\texttt{agent},x}$, $\texttt{vel}_{\texttt{agent},y}$,
- information about the object: a boolean indicating if the object is visible or not, the relative position, and the velocity of this object: $\texttt{is\_visible}_{\texttt{object}}$, $\texttt{dist}_{\texttt{object},x}$, $\texttt{dist}_{\texttt{object},y}$, $\texttt{vel}_{\texttt{object},x}$, $\texttt{vel}_{\texttt{object},y}$,
- and information about the landmark: a boolean indicating if the landmark is visible or not and the number of the corner it is located into (from 1 to 4): $\texttt{is\_visible}_{\texttt{landmark}}$, $\texttt{corner}_{\texttt{landmark}}$.

The observation is a vector of dimension 16 containing this information. Similarly to the coordinated placement task, this setting is also partially observable.

Results Figure 4.12 shows the results in the cooperative push scenario (median and confidence interval shown for 11 runs each). First, we observe that QMIX alone performs very poorly as it is unable to find the solution to the task. The high sparsity of the reward function makes it impossible for agents to discover the objective with random exploration of the environment. Second, we see that JIM and LIM achieve similar levels of performance. While coordination can help agents perform well, it is actually not a requirement for this task. In fact, one agent alone is able to push the

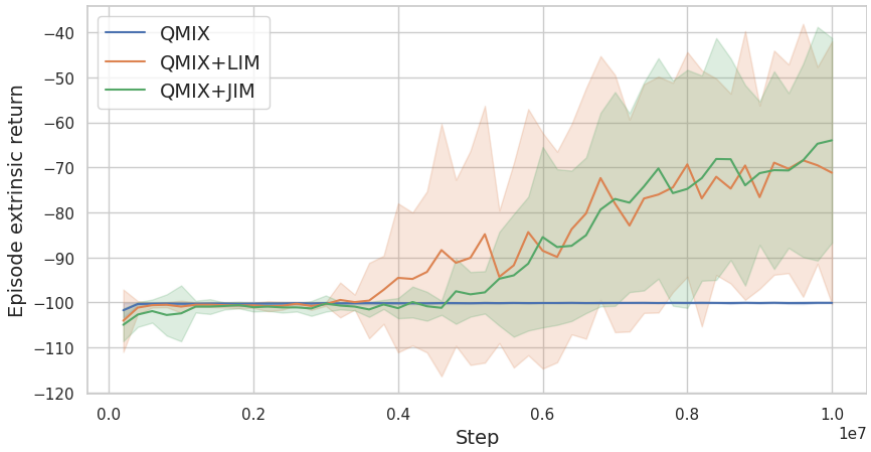


Figure 4.12: Training curves of the three variants of QMIX in the cooperative box pushing task, with the mean and standard deviation across 11 runs each.

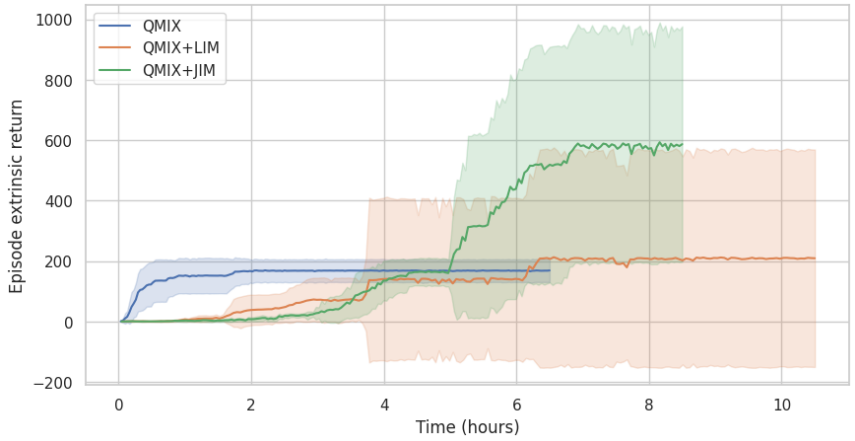


Figure 4.13: Training curves of QMIX, QMIX+LIM, and QMIX+JIM in the coordinated placement task with execution time on the x-axis. In our implementation, using JIM increases training time by 31%, and with LIM by 62%.

object and place it on the landmark. Thus, exploring the space of joint configurations is not helpful in this scenario. This shows however that actively exploring the environment is crucial in tasks where the reward function is very sparse.

Time Consumption

Finally, we show another advantage of having a single, centralised intrinsic motivation. While LIM and similar approaches in previous works (Du et al., 2019; Iqbal & Sha, 2019a; Wang, Wang, et al., 2020) require computing one intrinsic reward for each agent, JIM only computes one intrinsic reward for the whole group of agents. This makes JIM significantly more efficient to run, with LIM being approximately 24% slower than JIM to train, as shown in Figure 4.13.

4.8 Conclusion

In this chapter, we present an algorithm for joint intrinsic motivation (JIM), which is the first method to reward active exploration of the joint-observation space. It can be integrated to enhance any MADRL algorithm that uses centralised training with

decentralised execution. By combining JIM with the state-of-the-art QMIX algorithm, we demonstrate that it outperforms the original QMIX implementation, as well as a modified QMIX algorithm using local curiosity. We show that active exploration is a key component for multi-agent learning in environments with sparse rewards. Moreover, joint exploration enables the discovery of optimal coordinated behaviours that would be hard to find otherwise as they necessitate a high level of coordination between agents.

This shows the importance of using joint observations in the process of computing intrinsic rewards for a multi-agent system. In fact, the joint observation is the best estimate of the global state of the environment available for the agents. Using it allows more efficient learning of multi-agent joint behaviours and is computationally less expensive than having to compute local intrinsic rewards for each agent. These results should encourage research on how joint observations can be used in other kinds of intrinsic rewards to shape the agents' behaviour further.

Chapter 5

Language-augmented multi-agent learning

Learning to communicate with a pre-defined, discrete language

5.1 Introduction

Multi-agent deep reinforcement learning (MADRL) aims at learning how to behave in social settings where multiple intelligent agents interact. This social component adds complexity to the learning process, requiring agents to master additional skills to engage in these environments properly. One essential skill in such settings is communication, playing a crucial role in facilitating social interactions. It is observed in various forms across many social animal species (Searcy & Nowicki, 2010; Smith & Harper, 2003), and of course, humans, where it enables a great range of social abilities from sharing information about local observations to negotiation, knowledge transmission, and teaching (Greene & Burleson, 2003). Taking inspiration from these natural systems, groups of artificial agents can certainly benefit from being able to communicate with partners for sharing information gathered locally and expressing their intentions.

As described in Section 3.4.5, learning to communicate accounts to learning (i) *what information to transmit* and (ii) *how to transmit it*. A solution provided by MADRL is **differentiable emergent communication**, where agents develop communication protocols from scratch driven mainly by the goal of maximising returns. Using differentiable neural networks to produce messages allows learning communication as a sub-module of the policy through gradient back-propagation of the RL objective (Zhu et al., 2024). In other words, the problem of learning (i) and (ii) is entirely entrusted to the deep RL algorithm. Whether this approach could lead to the emergence of a language is itself debatable (Galke et al., 2022; Lazaridou & Baroni, 2020). Some approaches try to improve emergent communication in this regard, as we will see in Section 5.2.1. However, in MADRL research, this problem is usually disregarded to focus on how communication improves performance.

However, this approach suffers from significant drawbacks. One important issue is the **lack of interpretability** of the emerging communication systems. Since multi-agent systems are targeted toward applications in human-populated environments, interpretability becomes an important matter (Mikolov et al., 2018). If provided, it would enable transparent human-agent interactions and offer tools for evaluating the reasoning of learnt artificial agents. Researchers have proposed techniques to analyse the resulting communication processes by clustering messages based on state-message

correlation (Karten, Tucker, Li, et al., 2023; Li, Zhou, et al., 2022; Lin et al., 2021; Tucker et al., 2021), identifying patterns in emergent languages (Havrylov & Titov, 2017), or mapping emergent languages to known entities (Kottur et al., 2017). These approaches require tedious analysis of the emergent languages, offer no guarantees of success, and need to be applied separately to each independently trained group of agents, as each one develops its own unique communication protocol. Another approach is to bias the emerging communication protocols to carry meaning related to external modalities (e.g., visual inputs, actions, real-world concepts). These grounding approaches, reviewed in Section 5.2.2, guide emergent communication by informing potential useful meanings to convey in the communicated messages. However, this does not solve the interpretability issue, as emergent languages still require some post hoc analysis to be, still imperfectly, interpreted (Lin et al., 2021; Tucker et al., 2021).

A suitable solution to the interpretation issue would be to **have agents converse using a language** understood by humans. Rather than trusting the agents to develop a functioning communication system on their own, teaching them a pre-defined (as opposed to "emergent") language would have many benefits. Natural languages have evolved into very efficient communication systems, enabling the vast range of social interactions we take part in. Language allows describing parts of our environments to share local information with others. It allows expressing needs to explain intentions, negotiate, or ask for help. As a shared tool for expressing meanings, it supports the development of culture and the transmission of knowledge. In addition, language has been shown to play an important role in the intellectual development of children (Tomasello, 2009; Vygotsky, 1934). This idea has motivated research on language-augmented learning, reviewed in Section 5.2.4, to study how language can be used to guide the learning of artificial agents. Overall, providing a language to agents is a way to guide training by introducing a bias on how the environment should be understood and how communication should be conducted. If done successfully, the resulting language-augmented agents would be easier to evaluate, interact and bond with (Crandall et al., 2018; Liu et al., 2022; Mikolov et al., 2018).

While some previous works have experimented with language-based communication, to the best of our knowledge, no study applied it to embodied multi-agent environments. Indeed, learning to communicate with a pre-defined language to solve an embodied task presents its own challenges. It requires a reliable source of language examples and an algorithm capable of learning a language without interfering with policy learning. In this chapter, we propose a solution to these issues by training MADRL agents to produce language utterances that describe their observations. The learning algorithm, detailed in Section 5.3, trains agents to solve the multi-agent task while grounding their observations in language examples. This serves a double purpose: (i) **enabling agents to learn language-based communication**, while (ii) **guiding representation learning** by providing an efficient way of extracting valuable information from the world. To allow this, we provide agents with language examples that show how the pre-defined language should be used to describe observations. This data is used to train supervised language objectives concurrently to training on the multi-agent RL objective.

Through a series of experiments, we show that the cost we pay to train language-augmented agents is easily justified by the advantages brought by language. We demonstrate that language-based communication is easier to learn and more efficient than emergent communication baselines, as the pre-defined language is made to compress the informational content of observations to fit the needs of the task at hand. Having agents communicate with language also enables interesting capacities, namely:

- **better generalisation** of experience to changes in the environment,
- **better adaptation** to new partners,
- and easy **interactions** between humans and the learnt agents.

Crucially, language guidance helps agents structure their understanding of the world with linguistic concepts, resulting in faster learning.

5.2 Language and Communication in Artificial Intelligence: Related Works

5.2.1 Language and Emergent Communication

A language can be defined as a shared set of conventions for expressing meaning (Pinker & Bloom, 1990; Steels, 1995). In natural languages – i.e., languages that emerged naturally in human cultures –, large vocabularies and complex grammar systems form a rich sets of shared conventions allowing humans to communicate about a large variety of different matters. This is permitted by two key properties of these languages. First, natural languages are combinatorial. They use basic building blocks (e.g., phonemes, syllables) and combine them to construct rich semantic constructs that can hold complex meanings (Zuidema & de Boer, 2018). Second, they are compositional: the meaning of a combination of language blocks is a function of the meaning of the combined blocks (Smith, Brighton, & Kirby, 2003). These properties allow natural languages to express a seemingly infinite number of different meanings from a finite set of different symbols. Importantly, they allow to compress information to ensure less costly communication acts and simpler language transmission (Gibson et al., 2019; Sigurd et al., 2004). As a result, natural languages have been shown to have a nearly optimal expressiveness to compression ratio (Gibson et al., 2019; Kirby et al., 2015), ensuring high efficiency of communication acts and making these languages easier to learn (Carr et al., 2016).

Experiments involving emergent communication were initially aimed at studying how such languages emerge and evolve in populations of independent agents (Brighton et al., 2005; Kirby et al., 2015; Nowak & Krakauer, 1999; Steels, 1995; Vogt, 2005). These experiments showed that languages are cultural adaptive processes that evolve through their numerous independent uses (Christiansen & Chater, 2008; Kirby et al., 2015; Steels, 1995), and are shaped by environmental (Perfors & Navarro, 2014) and physiological (Christiansen & Chater, 2008) constraints. Iterated learning experiments studied the transmission of language between successive generations of language users (Kirby et al., 2015; Smith, Kirby, & Brighton, 2003; Vogt, 2005). They revealed that learning from a limited set of language acts results in a transmission bottleneck ultimately favouring combinatorial and compositional structures in languages. In this context, differential emergent communication has also been used to extend the study of language evolution to the multi-agent reinforcement learning setting (Chaabouni et al., 2019; Havrylov & Titov, 2017; Kottur et al., 2017; Lazaridou et al., 2017; Rita, Strub, et al., 2022). To this end, it has an interesting potential for studying the impact of embodied learning on the evolution of language (Van Eecke et al., 2022).

However, in the MADRL context, emergent communication, in its differentiable form, has been re-purposed as a tool for optimising multi-agent policies (Zhu et al., 2024). In this new task-oriented setting, the emergence of language properties is disregarded to focus on performance on a given multi-agent task. Differentiable communication fits well in the neural paradigm of deep RL, allowing end-to-end learning

of communication mechanisms without supervision (Das et al., 2019; Foerster et al., 2016b; Han et al., 2023; Jaques et al., 2019; Jiang & Lu, 2018; Kim et al., 2019; Mordatch & Abbeel, 2018; Singh et al., 2019; Sukhbaatar et al., 2016; Zhang et al., 2019). But, while emergent communication is almost always shown to be effective (i.e., improving performance), multiple works have highlighted its important limitations for this new purpose (Kottur et al., 2017; Lazaridou & Baroni, 2020). Emergent languages trained only from return maximisation lack the advantageous qualities of natural languages, namely compositionality (Galke et al., 2022), compression (Chaabouni et al., 2019), and consistency of used signals (Kottur et al., 2017). Bouchacourt and Baroni (2018) also showed that agents trained with differentiable communication are able to successfully communicate about noisy inputs. This shows that they have not learnt to communicate about underlying concepts present in their input data, but rather land on an arbitrary consensus on how to represent their observations.

These issues demonstrate that learning emergent communication only from communication success does not result in structured languages. However, by refining the training process, emergent languages can recover some qualities of natural languages. For example, compositionality can be improved by constraining the size of the vocabulary (Mordatch & Abbeel, 2018) or increasing the size of the population (Chaabouni et al., 2022; Rita, Strub, et al., 2022). Consistency of used signals can also be improved by maximising the mutual information between the input data and the generated messages (Eccles et al., 2019).

These numerous works on emergent communication help us identify the important qualities of natural language and how they enable efficient communication. They show that emergent languages can be driven to emulate these qualities, provided the right experimental designs and algorithmic choices. Crucially, they show that communication performance does not necessarily imply communication efficiency and, thus, that learning emergent communication from return maximisation alone produces poor communication mechanisms.

5.2.2 Improving Emergent Communication with Grounding

Previously cited issues of differentiable emergent communication may be all connected to a more fundamental problem: a lack of conceptual content in communicated signals. This can be related to the **symbol grounding problem**, defined by Harnad (1990) as the problem of associating meaning from the environment to a priori meaningless symbols. In natural languages, words and sentences carry meaning that is independent of the language itself: e.g., the symbol "apple" encodes a meaning that does not depend on the English language to exist. This consensus on how to express universally recognised meanings allows effective communication about these meanings. In the language evolution domain, the problem of grounding is crucial to understand how humans learn and use language (Brent & Siskind, 2001; Siskind, 1992), and how languages can emerge from grounded linguistic interactions (Botoko Ekila, 2024; Nevens et al., 2020; Steels & Kaplan, 2000; Steels, 2015; Vogt, 2002).

In Section 3.4.5, we defined differentiable emergent communication as learning a differentiable sub-step of the agent's policy to maximise the obtained returns. This training process does not have any requirement on how communication actually occurs: if it carries any information, if messages have any impact on the receivers, or if it has any kind of structure. While we can expect agents to learn to communicate about their observations, learning only from RL feedback gives no guarantee of this (Lowe et al., 2019). Indeed, we expect agents to learn to extract meaningful concepts from

their input data, with only task performance as guidance. This is probably too much to expect. As shown by Bouchacourt and Baroni (2018), a more probable outcome is that agents will find an arbitrary consensus on how to communicate about observations, without actually using any concepts that appear meaningful to us. This may be one source of explanation for the issues cited in the previous section: *how can symbols be combined and composed if they do not carry meaning?* This is also a fundamental problem for interpretation, as trying to understand potentially meaningless utterances would be pointless.

Therefore, instead of relying on return maximisation to produce meaningful communication, agents could benefit from explicitly grounding their messages into meaningful concepts. To do so, we can bias the training process by forcing communication to carry meaning related to external modalities. This can be achieved by adding auxiliary learning tasks on which agents are either pre-trained or trained on concurrently to RL training. By external modalities, we mean any source of meaning that is independent of the agents and the task at hand. The environment, and how the agents perceive it, is a source of meaning that can be leveraged to infuse meaning in the information transmitted between agents. By learning to reconstruct the input data from the communicated messages, agents can learn to maximise the informational content about their observations in their messages (Karten, Kailas, & Sycara, 2023; Karten, Tucker, Kailas, & Sycara, 2023; Lin et al., 2021; Lowe et al., 2020). Another external source of meaning is natural language. When used to describe the environment, it associates important environmental configurations with words and sentences. It also purposefully discards worthless or superfluous information in the input signals by simply not putting words on these. Learning to make these connections, by associating observations with corresponding language descriptions, allows agents to identify key concepts of their input space to communicate about (Das et al., 2017; Havrylov & Titov, 2017; Lazaridou et al., 2017, 2020; Tucker et al., 2021). These grounding approaches are a way to help agents learn meaningful communication with the support of the additional supervised learning objectives. It helps fill the gap of task-oriented RL, by providing guidance for learning to identify concepts.

5.2.3 Learning Natural Language

Taking this to the extreme allows learning to use natural language for communication, by learning to replicate human-generated sentences and generating natural language messages (Agarwal et al., 2019; Das et al., 2017; Gupta et al., 2021; Lazaridou et al., 2020; Lee, Cho, & Kiela, 2019; Lewis et al., 2017; Wang, Liang, & Manning, 2016). Using natural language instead of emergent communication does not completely solve the problem of learning to communicate. Agents still need to understand what symbol relates to what meaning and how compositional structures describe different environmental configurations. But, it solves the problem of interpretability, by making agents converse with the same tools as human experimenters.

However, learning to use a pre-existing language requires access to demonstrations of this particular language. Thus, to teach natural language to agents, previous works have been restricted to training environments that feature a large amount of language examples. Such environments are usually centred around language, with language being either the agents' observation or action domain. For example, the *Lewis signalling games* (Lewis, 1969), or *language games* (Steels, 1995; Steels & Kaplan, 2000), usually involve interactions between two agents: one speaker and one listener; where the goal is for the speaker to transmit some information about its observation

and for the listener to correctly interpret the incoming message. This framework can be used for describing natural language processing tasks, such as translation (Lee, Cho, & Kiela, 2019; Lee et al., 2018; Lu et al., 2020) or image captioning (Gupta et al., 2021; Lazaridou et al., 2020; Lee, Cho, & Kiela, 2019), allowing to exploit large datasets of language examples. Another example is *visual dialogue* (Agarwal et al., 2019; Das et al., 2017), which can benefit from datasets of conversation about images to learn communication with natural language.

While these settings show successful instances of learnt language-based communication, they fail to emulate the complete role of language in human communication: as a tool for supporting multi-agent interactions. In language-centred environments, agents lack of embodiment: they are defined only by their language abilities and have no physical interactions independent of the language task. We argue that this limits both:

- the complexity of the learning problem: in embodied, multi-agent settings, communication is not necessarily central to the task, agents need to learn when and how to use it best for the task at hand;
- the actual language abilities of these agents: physical interactions play an important role in learning language, thus, learning only from language data, even with visual grounding, can result in incomplete knowledge of the meaning behind natural language symbols.

In Section 5.3, we propose an approach for alleviating these issues, making a step towards learning language in multi-agent robotic settings.

Lastly, when it comes to using natural language almost as proficiently as human beings, *large language models* (LLMs) are now widely recognised as the most evident choice. Thanks to their extremely large architectures, extensive pre-training on large amounts of human-generated text, and use of RL to better fit human preferences (Christiano et al., 2017; Ouyang et al., 2022), they excel in generating human-like text. With some fine-tuning, they can easily be adapted to work with multi-modal inputs, allowing to handle visual and even behavioural modalities (Driess et al., 2023). With specific prompting techniques, LLMs can be driven to exhibit particular behaviours and even adopt personas (Li, Hammoud, et al., 2023; Park et al., 2023; Perez et al., 2024). All these features allow to define *agent-based LLMs* (Li, Chong, et al., 2023; Liu et al., 2024; Zhang et al., 2024) for studying multi-agent settings with LLMs mimicking human reasoning. There is no doubt that these capabilities have an interesting potential for handling more complex multi-agent tasks and interacting with artificial agents. However, there are still some issues and limitations preventing the wide adoption of LLMs. Their extremely high pre-training and fine-tuning costs prevent their adaptation to specific settings. While using LLMs "out-of-the-box" as a human-like language module can enable general language capacities, this inevitably means that the language-learning phase is lost. One goal of this work is to demonstrate that the act of learning the language can guide the agents in learning how to behave in their environment. In the next section and later in the experiments (see Section 5.5), we provide arguments and experimental results to back this claim.

5.2.4 Language-Augmented Learning

An important idea at the core of this work is the fact that language supports learning and intellectual development. In developmental psychology, Vygotsky (1934) has shown the central role of language and language-based social interactions in the intellectual development of children. Language serves structured reasoning as a way to

describe the learning environment, set goals based on previously experienced or observed interactions, imagine new goals based on the composition of known, language-expressed goals, and internalise social interactions for supporting later goal-based learning (Lupyan, 2012; Piaget, 1952; Tomasello, 2009; Vygotsky, 1934).

Following these observations, researchers have explored different ways by which language can guide the training of RL agents. Language can be used to ground the agents' understanding of their environment, by learning to describe observed situations (Hanjie et al., 2021; Hill et al., 2021; Ruis et al., 2020). This allows efficient representation learning and better generalisation to unseen environmental configurations and even different domains with similar entities (Narasimhan et al., 2018). Describing the world with language can also enable the prediction of future outcomes (Huang, Xia, Xiao, et al., 2023; Lin et al., 2023; Nottingham et al., 2023), allowing language-grounded model-based RL. Like with human beings, language can be used to express goals (Lynch & Sermanet, 2021). By expressing goals with language, agents can learn to associate the experienced trajectories with the linguistic concepts in the observed goals. In such goal-directed learning settings, language models can be used to generate rewards to guide training (Carta et al., 2022). This can make the learning of new goals easier (Co-Reyes et al., 2019; Li, Puig, et al., 2022; Shridhar et al., 2021) and even enable agents to imagine new goals by themselves using the learnt structures of language (Akakzia et al., 2021; Colas et al., 2020, 2022). This can be further implemented into hierarchical RL, with a high-level instruction policy generating language goals and a low-level policy selecting actions to complete these goals (Hu et al., 2019; Jiang et al., 2019; Weir et al., 2023). In this domain, the great language abilities of LLMs are also welcome to describe the agents observations (Huang, Xia, Xiao, et al., 2023; Zhu & Simmons, 2024), give them instructions (Ahn et al., 2022; Carta et al., 2023; Huang, Xia, Shah, et al., 2023; Huang, Xia, Xiao, et al., 2023), provide them with rewards (Baumli et al., 2023), or even serve as policies conditioned on both visual input and language (Brohan et al., 2023; Driess et al., 2023; Reed et al., 2022).

These numerous applications show the immense potential of language to guide the training of artificial agents. In this work, we aim to show that this is also true in multi-agent settings. In addition to communication, language can help agents acquire a better understanding of their environment and better generalisation abilities for handling dynamic multi-agent settings.

5.3 Language-Augmented Multi-Agent Communication: Our Method

5.3.1 Problem Statement

In this section, we describe our approach for learning multi-agent communication with a pre-defined language used for grounding and communication. Drawing inspiration from the different bodies of literature reviewed in the previous section, we design an architecture that allows embodied agents to learn a language for describing what they observe. Our objective is to show that this approach can help improve learning and communication in classical multi-agent settings. In such settings, communication is not central to the learning problem. Agents have to perform physical actions to interact with the environment and ultimately complete the task. Communication is a tool for influencing the behaviour of other agents. Agents have to learn how and when to use it to maximise their returns. With our language-based approach, we aim to demonstrate that language helps both the policy learning problem, by grounding

environmental elements and actions in the given language, and the communication learning problem, by providing a known-efficient way of communicating about the world.

The objective is to emulate a setting where a group of robots learns to perform a multi-agent task with an oracle, human or machine, that teaches them how to use a given language for the purpose of the task. Thus, during training, we assume that we have access to language descriptions of local observations. In Section 5.4.1, we define the language used in our setting and describe how the language descriptions are generated. Because we aim to train agents for rather simple robotic tasks, we do not need to teach them a complete natural language made for handling a wide variety of human situations. Instead, a simple language is defined with only the required vocabulary and grammar to fit the needs of the task at hand. Importantly, the provided descriptions give no additional information about the environment. They only provide a particular way of representing the meaningful content that is inside the agents' observations. By teaching this specific structured representation system to our agents, we aim to guide them in learning how to understand their observations and how to communicate efficiently about them.

5.3.2 Agent Architecture

To enable learning language-based communication in a multi-agent setting, we augment MADRL agents to equip them with language capabilities. We define a learning algorithm that fits in the CTDE paradigm:

- decentralised agents can generate messages and actions from their local observations only;
- these agents are trained in a centralised manner to improve the training efficiency of local policies and language modules.

Concretely, we build upon the MAPPO algorithm (see Section 3.4.3; Yu et al., 2021b): each agent has a local policy for selecting its actions and a centralised value used during training, as shown in Figure 5.1. In addition, the agent architecture is modified to include a communication policy that selects the information to communicate and language modules for generating and understanding language utterances.

Decentralised Action-Selection

At each time step t, agents start by receiving their local observation o^i_t, which is fed to the *observation encoder* E^i_{obs}. It produces an observation embedding: $\mathrm{E}^i_{obs,t}(h^i_{o,t}) = c^i_{obs,t} \in \mathbb{R}^H$, with E the embedding dimension and $h^i_{o,t} = (o^i_0, ..., o^i_{t-1})$ the observation history of agent i. Here, agents have access to their observation history because the encoder is defined as a recurrent neural network, allowing them to memorise some information from previous steps (see Appendix B.2 for a detailed definition of the neural networks in each module). The observation embedding then goes to the *communication policy* π^i_{comm} to produce the communication context: $\pi^i_{comm}(c^i_{obs,t}) = c^i_{comm,t} \in \mathbb{R}^C$, with C the context dimension. This communication vector $c^i_{comm,t}$ is a latent representation of what the agent wants to communicate to its partners. It goes through the *language decoder* D^i to generate a message, as described later in Section 5.3.3: $\mathrm{D}^i(c^i_{comm,t}) = m^i_t \in L$, with L the set of all possible sentences in our language. The generated message is then sent to other agents via the communication channel described below. After exchanging messages, the incoming message $\mathbf{m}_t$ goes through the *language encoder* L^i to encode the message into a

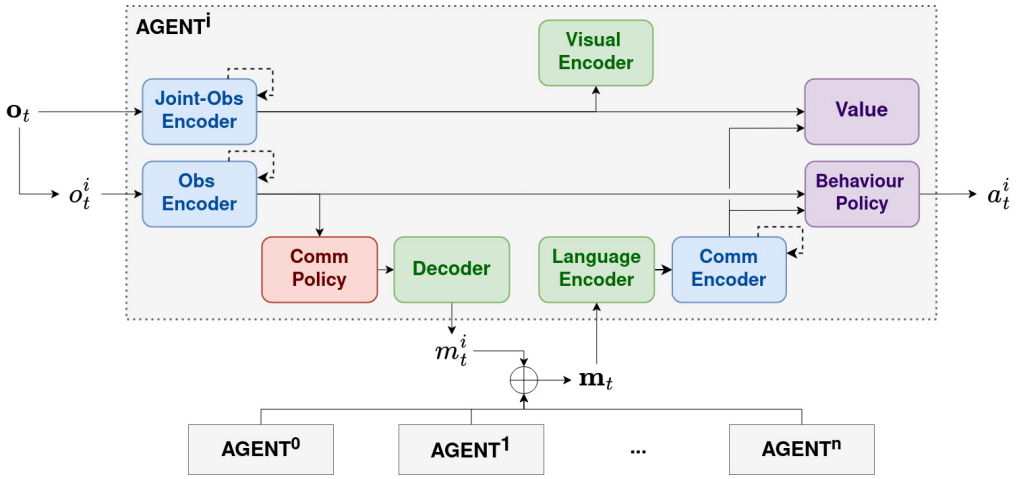


Figure 5.1: Illustration of the architecture of language-augmented agents. Each module represents a small neural network with a different purpose in the architecture. **Encoder modules** receive incoming information (observations or communication) and embed it in latent representation used in further modules, with dashed arrows indicating that they contain a recurrent neural network to enable having memory of previous steps. The **communication policy** selects what information in the observation should be communicated. **Language modules** provide language capabilities: the decoder generates messages, the language encoder transforms incoming messages in a compact numerical representation, and the visual encoder is there for training the language encoder (as described in Section 5.3.3). Finally, **RL modules** take information from both observations and communication to select actions.

vector representation, as described in Section 5.3.3, which itself is fed to the *communication encoder* E^i_{comm} to produce the social context: $\mathrm{E}^i_{comm}(\mathrm{L}^i(\mathbf{m}_t)) = c^i_{soc,t} \in \mathbb{R}^C$. Note that the communication encoder is also defined as a recurrent neural network to allow keeping some memory of previous messages. Finally, the social context is concatenated to the observation embedding to construct the input of the behaviour policy π^i, which generates the agent's action: $\pi^i(c^i_{obs,t}, c^i_{soc,t}) = a^i_t$.

Communication Channel

For exchanging messages, we choose to stick with a rather simple solution where messages are sent to all agents of the system. Thus, after all agents have generated their message, the individual messages are gathered and concatenated to form the broadcast message $\mathbf{m}_t = \{m^0_t, ..., m^n_t\}$ that all agents receive. This broadcasting approach fits well the kind of settings we are interested in: cooperative tasks with a relatively small number of agents. In such settings, there is no benefit to keeping some information from other agents and we can assume that all agents can be connected by a common communication channel. However, this could easily be adapted to environments with restricted communication channels. For example, we could define a communication range that limits the distance by which agents can send messages to other agents.

Centralised Actor-Critic Learning

In MAPPO, agents learn a centralised value function to guide the training of their decentralised policy. The value function is said to be "centralised" because it takes the joint-observation as input. Using this centralised information during training helps the value function making better value estimation. In our case, the value module takes

as input the output of the *joint-observation encoder* $\mathrm{E}^i_{joint,t}(\mathbf{h}_{o,t}) = c^i_{joint,t} \in \mathbb{R}^H$, with $\mathbf{h}_{o,t}$ the joint-observation history; and the social context $c^i_{soc,t}$ coming from the communication step. These two vectors are concatenated and through a neural network to generate the value estimate for the current state $V^i(c^i_{joint,t}, c^i_{obs,t}) \in \mathbb{R}$. This value is used during training to compute the loss of PPO described in Section 2.5.2.

5.3.3 Language Learning

To enable agents to use a pre-defined language for sharing information, agents must be able to understand the given language and generate language utterances. To learn these two skills, techniques from the domain of natural language processing can be applied. Here, we define the two supervised learning objectives used for training the **language encoder** and the **decoder**.

Grounding Language with Contrastive Learning

To understand a given language, one solution is to learn to associate language examples with their meaning as presented in a different data space. Say we have a set of observations O and a set of language descriptions L, with one language description l^i for each observation o^i. Language descriptions provide a semantic proposal for describing the informational content of observations. To learn how this language works for extracting and arranging meaning, we can learn to associate observation-description pairs together and dissociate the non-paired elements. This is the idea behind **contrastive learning**, that learns relations between different data spaces by learning to associate paired data points (Oord et al., 2018; Tian et al., 2020). We follow the method **CLIP** (contrastive language-image pre-training; Radford et al., 2021) that trains two encoders, one for language descriptions and one for observations, both generating compact vector representations of the input. To ground language learning, the encoders are trained to generate representations of observations and descriptions that maximise the mutual information between correct pairs (o^i, l^i) and minimise the mutual information of incorrect pairs (o^i, l^j). Concretely, the *visual encoder* $\mathrm{V} : O \to \mathbb{R}^C$ and *language encoder* $\mathrm{L} : L \to \mathbb{R}^C$ are trained to maximise the cosine similarity of correct pairs:

$$\max_{\theta_\mathrm{V}, \theta_\mathrm{L}} \left[cosim(\mathrm{V}(o^i), \mathrm{L}(l^i)) \right], \tag{5.1}$$

and minimise it for incorrect ones:

$$\min_{\theta_\mathrm{V}, \theta_\mathrm{L}} \left[cosim(\mathrm{V}(o^i), \mathrm{L}(l^j)) \right], \tag{5.2}$$

with any $i, j \setminus i \neq j$, θ_V and θ_L the parameters of the visual and language encoders respectively, and the cosine similarity of two vectors: $cosim(a, b) := \frac{a \cdot b}{\|a\| \|b\|}$. Overall, the objective of CLIP can be formulated as minimising the following loss:

$$L^{CLIP}(\theta_\mathrm{V}, \theta_\mathrm{L}) = \sum_{i,j} CLIP(i, j), \tag{5.3}$$

$$CLIP(i, j) = \begin{cases} -cosim(\mathrm{V}(o^i), \mathrm{L}(l^i)) & \text{if } i = j, \\ cosim(\mathrm{V}(o^i), \mathrm{L}(l^j)) & \text{if } i \neq j. \end{cases} \tag{5.4}$$

To model the two encoders, we use different kinds of neural network architectures. The visual encoder V can be modelled as a simple MLP (see Section 2.4.3) that

transforms an observation vector $o^i \in \mathbb{R}^N$, with N the dimension of the observation space, into the observation embedding $\mathrm{V}(o^i) \in \mathbb{R}^C$. For the language encoder L, we need a more complex architecture as a language description l^i is made of a sequence of tokens: $l^i = (t_0, ..., t_K)$, with each token $t_k \in \mathbb{R}^{D^V}$ being drawn from a vocabulary V of size D^V, and represented as a one-hot vector. In natural language processing, tokens are the basic blocks of language sequences: e.g., characters, syllables, words, or punctuation marks. To encode a sequence of tokens, L must be defined as a recurrent neural network, in our case a GRU (see Section 2.4.2). Each token in l^i is successively passed through the GRU: $f_{GRU}(h_k, t_k) = h_{k+1}$, with h the hidden state of the GRU (h_0 being initialised with zeros). After all tokens have been handled, the final hidden state h_K is passed through a single neural network layer to output the description embedding $\mathrm{L}(l^i) \in \mathbb{R}^C$.

Generating Language by Learning Captioning

To learn how to generate language, we use a second supervised learning task from natural language processing. We adapt the **image captioning** task that trains a model to generate the language descriptions of images, by feeding it a large number of image-description pairs. In our case, we want the agents to learn to describe their observations. Thus, agents are equipped with a *decoder* $\mathrm{D} : O \to L$. Given a set of training examples (o^i, l^i), the decoder takes o^i as input and is tasked to generate the corresponding language description $l^i \in L$.

Similar to the language encoder, the decoder needs a recurrent architecture to generate language utterances. Thus, the decoder will use a GRU that takes as input a hidden state h_k and a token t_k, and outputs the token t_{k+1} that follows in the sequence. To generate a single utterance $l^i = (t_0, ..., t_K)$, the observation o^i is passed through a neural network to produce a compact representation that can be passed to the decoder as its initial hidden state: $f_{in}(o^i) = h_0$, with $f_{in} : O \to \mathbb{R}^H$ an MLP and H the dimension of the decoder's hidden state. The GRU then takes as input h_0 and a special "start-of-sequence" token $t_0 = <\mathrm{SOS}>$, and outputs the new hidden state and following token: $f_{GRU}(h_0, t_0) = (h_1, t_1)$. The process continues until the GRU produces the special "end-of-sequence" token $t_K = <\mathrm{EOS}>$. To learn the captioning task, for each training pairs (o^i, l^i), the decoder generates a candidate sequence $\hat{l}^i = (\hat{t}_0, ..., \hat{t}_K)$ and is trained to minimise the cross-entropy:

$$L^{capt}(l^i, \hat{l}^i, \theta_{\mathrm{D}}) = -\sum_{k=1}^{K} p(t_k) \log \hat{p}(t_k), \tag{5.5}$$

with p the target probability density and $\hat{p}$ the probability density generated by f_{GRU} from which $\hat{t}_k$ is sampled.

Integrating Language Modules

With these two natural language approaches, we can enable the agents to generate messages that describe their observations and encode incoming messages into embeddings grounded in their observation space. To integrate these capacities in the agent architecture, we include three required language modules (depicted in green in Figure 5.1). They are placed in the agent architecture to allow agents to learn the objectives of CLIP and the captioning task.

The decoder is placed after the communication policy to generate messages. The input network f_{in} described previously is replaced by the observation encoder and

communication policy. Thus, the observation encoder, the communication policy, and the decoder are trained jointly on the captioning task to generate accurate descriptions of the observation.

Because the language encoder is tasked to encode broadcast messages – i.e., descriptions of all local observations combined – it seems adequate to place the visual encoder after the joint-observation encoder. This way, the contrastive learning objective is learnt with centralised information as input of both encoders.

The data required for training on these objectives is gathered by the agents as they interact with the environment: at each time step, an observation-description pair is gathered by each agent. When the agents' policies are trained, we compute the losses for PPO, CLIP, and the captioning task, and optimise the three together by minimising:

$$L^{TOT} = \beta_{PPO} L^{PPO} + \beta_{CLIP} L^{CLIP} + \beta_{capt} L^{capt}, \tag{5.6}$$

with L^{PPO} defined in Equation 2.52, L^{CLIP} and L^{capt} defined above, and parameters β for weighting each loss (see Section 5.4.4 for details on how they are defined).

This training procedure allows training the agents' policy and language skills completely end-to-end. Note that with the proposed architecture, both the observation and joint-observation encoders contribute to the language losses. Therefore, both the policy and value sides of the agents will be grounded in language, with the input encoders learning to extract language-based concepts from the observations.

5.4 Implementation

5.4.1 Language and Oracle Definition

To enable agents to communicate about their observations, we define a simple language that allows describing elements from the environments that are important for the task at hand. In this work, we focus on the Predator-Prey setting, where agents need to find preys and coordinate to catch them (see Section 5.5.1 for details on the scenario). As agents have a limited field of view, sharing the positions of observed preys to the group should improve the catching speed. Therefore, we design a language that allows this by stating the cardinal position (i.e., "North", "South", "East", "West", and "Center") of the observed preys, as shown in Figure 5.2. The **oracle** is a rule-based program that takes local observations as input and generates the corresponding language descriptions. The resulting language has the following limited vocabulary:

$$V = \{\text{Prey, North, South, East, West, Center}\}$$

By describing the positions of two preys at most, it can produce descriptions of six tokens maximum (e.g., "Prey North East Prey South West").

Despite its apparent simplicity, this language provides the tools for efficiently signalling the region of the grid that other agents should head towards. It is compositional, with the "Prey" token providing structure in sentences – i.e., after "Prey" always comes a localisation –, and the cardinal tokens that can be composed to express different regions. Importantly, it offers a structured, condensed representation of the agent's observation. This provides the language guidance we sought for understanding the environment. By learning the observation captioning task, the agents will learn to recognise important features in the observations and represent them efficiently.

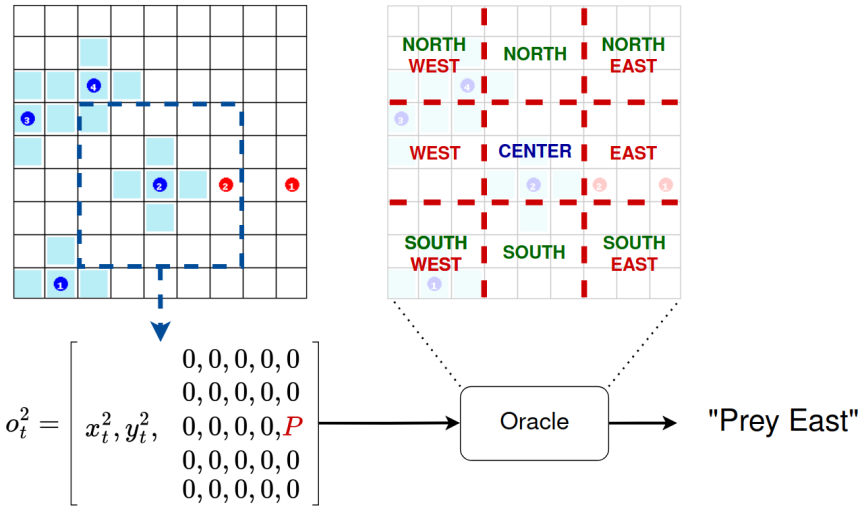


Figure 5.2: Illustration of the oracle's process for describing observations. In the top-left part is a screenshot of the *ma-gym* environment in the Predator-Prey task (agents in blue and preys in red). Agents observe their absolute position in the grid and the objects in their surrounding 5×5 observation range (shown in dark blue). The oracle generates a language description that describes the cardinal position of each observed prey.

5.4.2 Baselines Definition

To try to measure the benefits of learning to communicate with a pre-defined language, we compare the language-augmented agents with four other communication strategies. Here, we define the five compared agent versions:

- `Language` agents use our proposed method for learning to communicate with the pre-defined language. They learn to describe their observations and use the generated descriptions as messages during the episodes.
- `Oracle` agents use the descriptions given by the oracle as messages. They learn the language tasks as well but do not use the generated messages and instead directly send the oracle's description. They provide a baseline for measuring the impact of RL training on language generation in the `Language` agents.
- `Emergent-Discrete` agents learn differentiable emergent communication, with discrete symbols. We provide these agents with the same theoretical language capabilities as the language-augmented agents: they use the same architecture and can generate sentences of the same length with a vocabulary of the same dimension. Thus, they differ in the fact that they do not learn from the language tasks. Instead, they learn to use their discrete symbols through back-propagation of the RL objective, using the Gumbel-Softmax trick to allow gradients to flow through the sampling operation (Jang et al., 2017).
- `Emergent-Continuous` agents use differentiable emergent communication, with continuous signals. They use the architecture illustrated in Figure 5.1 but with no language modules. The messages are directly generated by the communication policy: $m_t^i = c_{comm,t}^i \in \mathbb{R}^C$, with $C = 2$ in this case. Inspired by previous work (Das et al., 2019), we choose this rather small size for continuous messages that still theoretically allows expressing an infinite number of different meanings.
- `No Communication` agents do not communicate at all. They are here as a baseline to measure the impact of the different communication approaches.

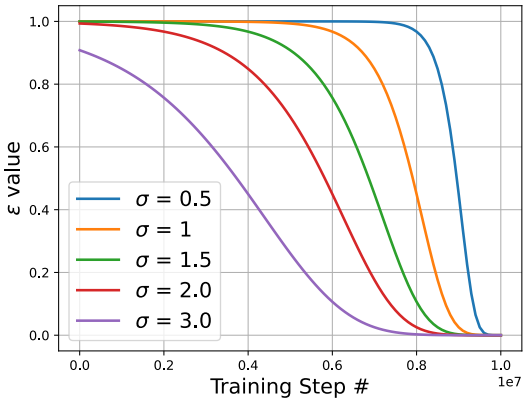


Figure 5.3: Decay curves of the ε parameter used in the ε-oracle strategy for learning to communicate with language. The parameter σ allows controlling the slope of the decay.

5.4.3 Communication ε-Oracle Strategy

When starting to learn language, agents will inevitably generate bad messages. To prevent this from hindering communication learning, we allow agents to use the oracle's description as messages at first and slowly transition to the generated messages. Thus, communication is ensured to serve its purpose from the beginning of training, allowing the policy to learn to use it accordingly.

To enable this transition from oracle to generated messages, we devise a "*ε-oracle*" communication strategy (inspired from ε-greedy in Q-learning). At each time-step, agents randomly choose whether to send the oracle's description or the generated message, with probabilities of $p(\text{oracle}) = \varepsilon$ and $p(\text{generated}) = 1-\varepsilon$. The ε parameter is decayed from 1 to 0 throughout training, following a sigmoid-like decay curve, as illustrated in Figure 5.3. In Appendix B.3, we define formally the function for decaying ε. The parameter σ allows controlling the slope of the decay. In our experiments, we stick with $\sigma = 2$ to have a slow transition, but it could be interesting to study the effects of different values of σ on communication learning.

5.4.4 Loss Weighting

In Equation 5.6, we define the total loss optimised by our algorithm, combining losses from different objectives by summing them together. A problem with this method is that different losses might take values of different orders of magnitude, which can prevent all losses from being optimised at the same rate. This is a well-known problem in the multi-task learning literature, that unfortunately does not have a widely recognised preeminent solution (Vandenhende et al., 2021). Intuitively, we consider that all losses should be of the same magnitude to ensure that they contribute equally to the total loss. Knowing that all losses evolve at very different rates, and inspired from previous work (Liu, Johns, & Davison, 2019), we dynamically update the weighting parameters β_{PPO}, β_{CLIP}, and β_{capt} so they all have values close to 1. To do so, during training iteration k, we have:

$$\beta_k = \frac{1}{L_{k-1}}, \tag{5.7}$$

for each different loss[1]. In other words, each loss is normalised by the value of the loss at the last iteration of training. We found that it was a simple solution for weighting many very different losses, that yielded the best results in our case.

5.4.5 Implementation Details

In Appendix B.2, we provide a detailed definition of the agent architecture with the neural networks in each module and the set of hyperparameters used in our experiments. All compared versions of the architecture use the same set of hyperparameters, apart from the context dimension C that is set to 2 for `Emergent-Continuous` as previously mentioned, and to 16 for other variants. Our code for the architecture and training is available online[2].

One important information to point out is the use of embedding layers in the language modules, to learn a specific vector representation of each token in the vocabulary (Mikolov et al., 2013). This is a common technique in natural language processing, to help learn the role of different tokens in the given language and how they can be composed in structured sentences. The embedding layers are shared between the decoder and language encoder and learnt during training with the rest of the architecture.

5.5 Experiments

To demonstrate how language can help in learning multi-agent behaviour and communication, we devise a set of experiments to showcase different interesting advantages. We first show that our architecture enables efficiently solving a classical multi-agent problem: the Predator-Prey task; and doing so comparatively better than emergent communication baselines. Then, we highlight several advantages of language-based communication for generalising better, teaming with unknown partners, and enabling human-agent interaction.

5.5.1 Learning to Communicate

Task

In this work, we focus on the Predator-Prey task to showcase the advantages of using language. We use the *ma-gym* environment (Koul, 2019), illustrated in Figure 5.2, that offers a two-dimensional grid playground for cooperative multi-agent tasks. Most of our experiments are set in a 9×9 map, with four agents as predators, and two preys with random behaviour. In our setting, agents observe:

- their absolute (x, y) position in the grid,
- the objects present in their 5×5 observation range: a vector of 25 values with 0 for nothing, 1 for a prey, and 2 for another agent.

The agents' objective is to capture the two preys, which triggers the end of the episode. To capture a prey, a minimum of two agents have to be orthogonally adjacent to the prey. If a prey is caught, it disappears from the map and agents collectively receive a large positive reward. If an agent tries to capture a prey alone, which we refer to

[1] For simplicity, we noted only one parameter β_{PPO} for the PPO objective. But, L^{PPO} is itself made of multiple losses, as defined in Section 2.5.2. In our implementation, we actually have different β parameters for the policy and value parts of PPO's loss

[2] https://github.com/MToquebiau/Language-Augmented-MADRL

as a "missed capture", a penalty is received by all agents. The overall reward at each time step is:

$$r_t = p_{step} + N_t^{missed} p_{missed} + N_t^{capture} r_{capture}, \tag{5.8}$$

$$p_{step} = -0.01, \tag{5.9}$$

$$p_{missed} = -0.5, \tag{5.10}$$

$$r_{capture} = 5.0, \tag{5.11}$$

with $N_t^{capture}$ and N_t^{missed} the numbers of captures and missed captures, respectively, during time step t. Agents have a maximum of 100 steps to catch the prey, otherwise the episode stops.

Results

In Figure 5.4, we present the results of training the five agent architectures in the Predator-Prey environment for 10 million environment steps. Each curve corresponds to 15 independent runs with different initial random seeds, with the median and 95% confidence interval displayed. The first clear observation to make is that the two language-based versions, `Language` and `Oracle`, both achieve faster learning and better final performance. `Language` agents successively learn to use the given language, with 92% of generated messages that are equal to the oracle's descriptions. We observed that longer messages (i.e., when two preys are observed) are harder to generate perfectly. In Section 5.6.2, we discuss potential ways for improving language learning further. However, this is still a largely acceptable level of effective language generation, enabling interpretability of generated messages.

Agents with `No Communication` achieve quite good performance, indicating that this setting does not require communication to be solved, even though it can help to be faster. Interestingly, the worst agents here are the ones with `Emergent-Continuous` communication. It seems that it would take them more time to reach the level of performance of other variants. This can be attributed to the fact they are learning to use continuous vectors for communication, and thus have far more options for the consensus they could achieve. Importantly, we see that the `Language` agents achieve similar results as the `Oracle` agents. This shows that there is no loss to using the generated messages. Overall, learning to use a language clearly helps agents solve the task more efficiently.

Ablation Study

To evaluate our language-augmented agents more thoroughly, we conduct an ablation study to investigate the benefits brought by language-based communication and language grounding separately. We compare the `Language` agents with three variants of our architecture:

- `No Comm+Language` agents do not communicate but they still learn the language the language tasks. This means that their understanding of the environment is still grounded in language. Thus, these agents will suffer from the lack of communication, but by grounding their representation learning in language they should learn more efficiently.
- `Oracle+No Language` agents communicate the language descriptions given by the oracle but do not learn the language tasks. The language encoder is still required for encoding incoming messages, but it is now learnt only from the RL

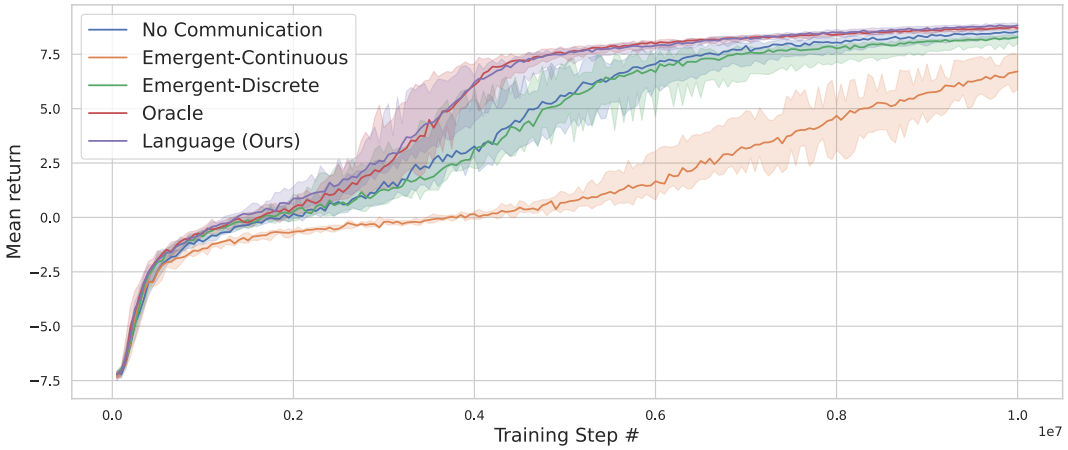


Figure 5.4: Training performance in the Predator-Prey setting (15 runs each, with median and 95% confidence interval). `Language` agents are compared to multiple baselines, showing that language-based communication obtains the best results, both being faster to converge and obtaining better final performance than the emergent communication and no communication baselines. `Language` obtains similar performance as the `Oracle` baseline, showing that our algorithm successfully learns to use language.

objective. Thus, these agents communicate efficiently with the pre-defined language, but their policy is not grounded in language and language understanding is not grounded in observations.

- `Observations` agents use their local observation as message: $m_t^i = o_t^i$. Thus, they communicate with a pre-defined structured system (i.e., not emergent) and their messages contain all the information they have gathered from the environment at each step. But, there is no compression at all, meaning that irrelevant information will be communicated.

These ablations and their communication and grounding properties are summarised in Table 5.1.

The results of training these ablated versions are shown in Figure 5.5. They reveal multiple interesting insights. First, both the `No Comm+Language` and `Oracle+No Language` variants are worse than `Language`, showing that combining language-based communication and language grounding enables the improved performance of our `Language` agents. `No Comm+Language` achieves the same final performance as `No Communication` but is slightly faster to increase, indicating that language grounding may help agents to learn faster. Experiments in an environment with more variety in the type of observed objects, and thus more potential for language grounding, may help validate this point. `Oracle+No Language` achieves a similar final performance as the `Language` agents but is significantly slower. This validates the importance of learning the language tasks to ground representation learning in language concepts. Lastly, the `Observations` agents are clearly inferior. They learn slower and achieve lower performance in the given time frame. With this communication strategy, the dimension of the incoming messages is significantly larger. This may explain the slower learning, as agents struggle to understand how to exploit this high-dimensional input space. This strongly highlights the importance of compression of information in communication systems. Communicating efficiently means sending the right information *and* expressing it in a way that will be understood easily. With our pre-defined language, we manually introduce these two qualities and thus allow efficient communication.

Agent version	Communication Strategy	Language Grounding
Language	Learnt language	Yes
No Comm+Language	No Communication	Yes
Oracle+No Language	Oracle	No
Observations	Observations	No

Table 5.1: Summary of the properties of each ablated version. `No Comm+Language` does not communicate but still learns to use language, thus it conserves the language grounding advantage. `Oracle+No Language` communicates perfectly formed language messages, but does not receive gradients from language learning, thus it has the advantage of language communication but lacks language grounding. Finally, `Observations` communicates the local observations directly, thus messages are structured and contain all information available to the agent, but they do not compress or ground this information in any way.

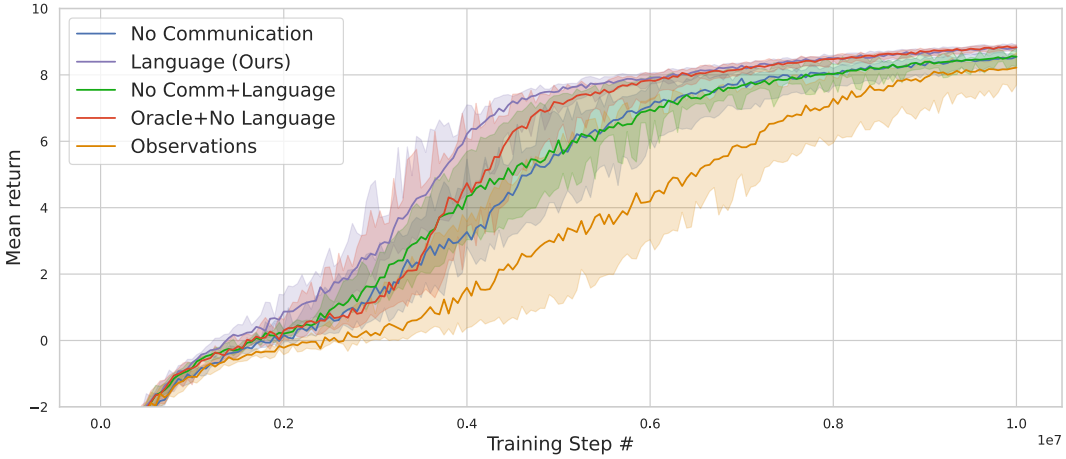


Figure 5.5: Training performance of the ablated versions of the architecture, with `Language` `No Communication` for reference (15 runs each, with median and 95% confidence interval). `Language` agents are clearly superior, showing the importance of language-based communication, combined with language-grounded policy learning.

5.5.2 Generalisation

In Section 3.5.3, we presented the problem of generalisation in (multi-agent) RL. When training agents on a given task, we would like them to be able to generalise their acquired knowledge to be robust to changes in the environment. By structuring meaning from the environment using task-agnostic concepts, language is a tool that enables the generalisation of previously acquired knowledge. Thus, learning to ground their knowledge in language concepts should help artificial agents to better adapt to similar environments (Narasimhan et al., 2018). To test this, we transfer the different agents to a slightly different version of their training environment and look at how fast they adapt.

Protocol

To investigate the generalisation ability of the previously trained agents, we take the best-performing run of each agent version in the 9×9 Predator-Prey setting and transfer them to a larger, 15×15 version of the environment. This affects the observation space by changing the values taken by the (x, y) coordinates observed by the agents. This also affects the effectiveness of the learnt policies, as traversing the environment now requires more steps. Note that this larger environment is too difficult to solve for all agent versions when trained from scratch (see Appendix B.1 for the training graph).

In the initial training phase of the `Language` agents, we had to pay a cost to enable language learning, by providing language descriptions for each observation. In this new setting, we want to show that this cost can be significantly reduced by receiving language examples during a limited number of steps. After a given number K of environment steps, we stop providing language descriptions and training the language modules on contrastive learning and captioning. Consequently, the ε-oracle strategy is also modified to take place during the K first steps of this fine-tuning phase. We will call the resulting variant `Language (Frozen-K)`.

Results

Figure 5.6 presents the results of this transfer experiment. For each version, we display a horizontal dotted line indicating the final performance of the run used for fine-tuning in the 9×9 setting. This should not be seen as a metric for measuring performance. Because the environment has changed, the old return measures do not preserve their meaning. However, because the change in the environment does not affect the task itself, we use these return values as a proxy for measuring the adaptation speed of each variant. They show how fast each model retrieves the performance they had in the smaller environment. We can see that the language-augmented agents are the fastest to adapt, `Language` reaching its old performance in 1.2 million steps, confirming that language can help generalise more easily. The `No Communication` agents never reach their old performance, showing that, in this larger environment, communication is more valuable for achieving better results. The `Emergent-Discrete` agents follow a similar curve as the non-communicating agents. This may indicate that agents are not using the communication channel accordingly. Finally, the `Emergent-Continuous` agents show better adaptability, by surpassing their previous performance. This confirms the speculation that agents need more training to reach an effective continuous communication system.

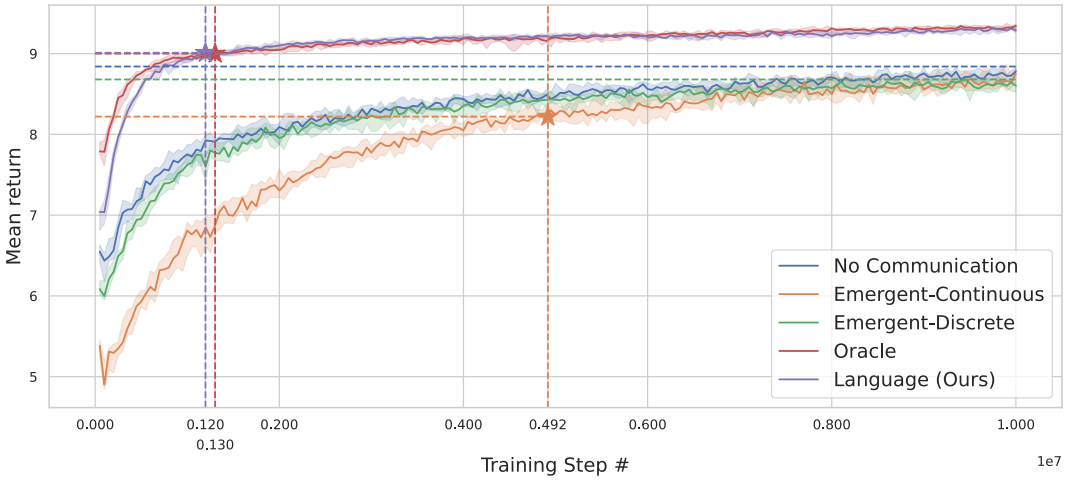


Figure 5.6: Fine-tuning performance in the 15×15 version of the environment (9 runs each, with median and 95% confidence interval). The horizontal dotted lines represent the final performance of the fine-tuned model in the 9×9 setting. Stars indicate the step when models surpass their old performance. Both language-based versions adapt faster to the new environment.

Results of the fine-tuning experiments on the `Language (Frozen-K)` agents are displayed in Figure 5.7. The bottom graph shows that when language training is stopped, the ratio of generated messages that match the oracle's description starts dropping. Across all variants, we can see that this ratio drops by approximately 5% for each 1 million steps. This may cause interpretability issues. Thus, the frozen approach is not perfect for limiting the cost of language supervision. In Section 5.6.2, we discuss this and propose an alternative method that could solve this issue. However, looking at the task performance in the top graph shows that stopping language training has only a slight impact task performance. The longer-trained `Language (Frozen-5M)` agents reach a similar level of performance as the non-frozen agents, while others still achieve better results than the emergent communication strategy in Figure 5.6. This is promising as it shows that agents are robust to slight alteration of the communicated messages.

5.5.3 Zero-Shot Teaming

One other aspect of generalisation that is specific to the multi-agent setting is the problem of adapting to new, unknown teammates. Agents trained with a fixed team of partners may overfit their learnt policy to the policies of other agents. This can be a problem if agents are expected to work with new partners during their lifetime, as could be the case in robotic settings. We expect the overfitting problem to be even more problematic when agents are trained with emergent communication, as the emergent language will likely differ between separate teams of agents. On the other hand, language is, by definition, a tool for enabling communication with unknown partners. Thus, language can be helpful to improve zero-shot teams.

Protocol

To evaluate communication strategies in zero-shot teaming, we take the four best teams of each variant after being trained in the 9×9 environment. We then build new teams with agents randomly picked from these four original teams. Each new team is

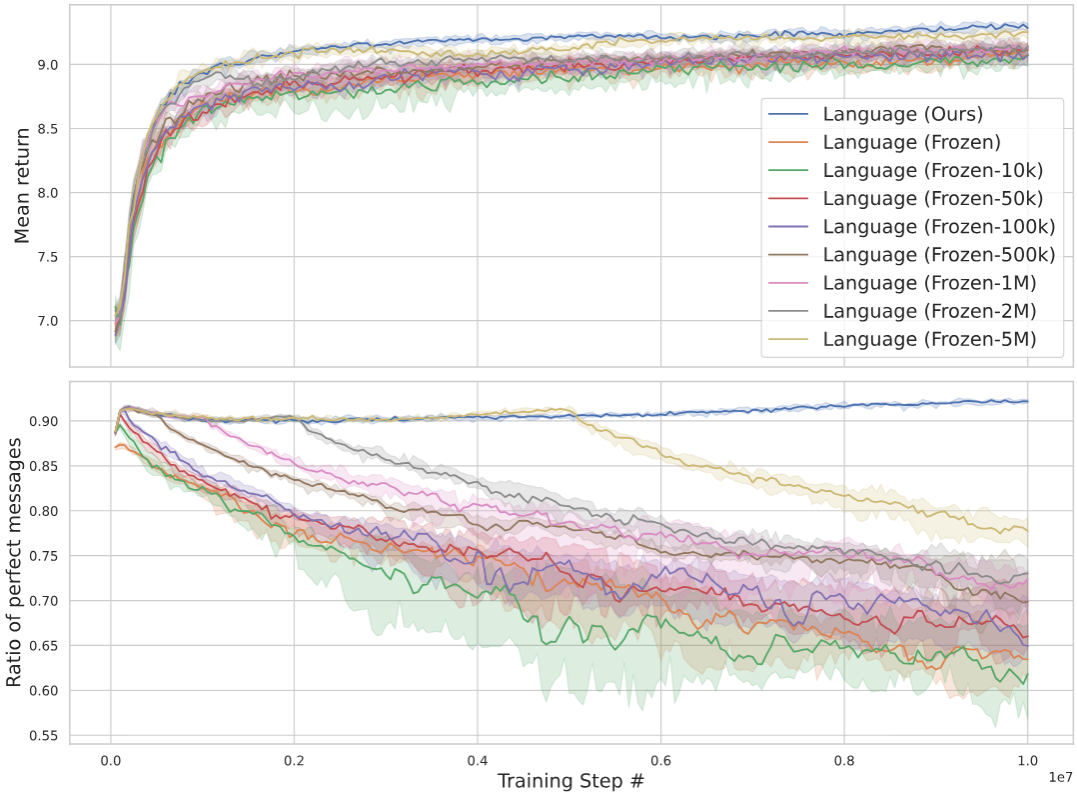


Figure 5.7: Effects of stopping language guidance in the fine-tuning experiment (9 runs each, with median and 95% confidence interval). The top graph shows the task performance across fine-tuning. The bottom graph displays the ratio of generated messages that match the "perfect" description given by the oracle.

	Team Composition				
	AAAA	AAAB	AABB	AABC	ABCD
Em-Continuous	7.98 ± 0.30	4.63 ± 1.78	3.43 ± 1.69	2.97 ± 1.59	2.49 ± 1.42
Em-Discrete	8.63 ± 0.17	6.81 ± 1.49	6.15 ± 1.7	5.67 ± 1.72	5.16 ± 1.79
Oracle	8.93 ± 0.19	7.19 ± 1.41	6.48 ± 1.36	6.35 ± 1.38	5.91 ± 1.6
Language	9.06 ± 0.1	7.38 ± 1.19	6.43 ± 1.49	6.06 ± 1.67	5.47 ± 1.73

Table 5.2: Results of the zero-shot teaming experiments. New teams are composed from the best four runs of each variant. The scores shown are the mean and standard deviation of one thousand evaluation runs, with one run corresponding to (i) picking agents in pre-trained teams randomly according to the team composition, (ii) running 250 episodes in the 9×9 map, and (iii) record the mean return on these episodes as evaluation score. Increasingly heterogeneous teams are composed to measure the impact of having unknown partners on performance. Letters indicate the team composition: "AAAA" is an unmodified team, "AAAB" has three agents from one run and one agent from another, etc.

evaluated on 250 episodes of Predator-Prey. The team's evaluation score is the mean of all returns obtained in these episodes. For each agent version, we evaluate one thousand different randomly picked teams and then record the mean and standard deviation of evaluation scores. Table 5.2 shows these results. To investigate the effect of increasing heterogeneity in teams, we evaluate different team compositions, as shown in the table. The team compositions go from an unmodified team ("AAAA" in the table), used as a baseline, to a fully mixed team with one agent from each pre-trained run ("ABCD").

Results

Results shown in Table 5.2 confirm our hypothesis and demonstrate the potential of language for improving zero-shot teaming. As expected, newly formed teams are less efficient in all variants, but language-based ones suffer less from this. `Emergent-Continuous` agents suffer the most from having to communicate with unknown agents, with performance dropping significantly with only one external agent in the team. Interestingly, the `Emergent-Discrete` agents suffer less. This may indicate that discrete symbols can be more easily interpreted by unknown agents, or that these agents are less concerned by communication, which is beneficial in this case. Again, a deeper analysis of the emergent language could help clarify the situation. Another interesting observation is that `Language` agents lose more performance than `Oracle` ones, showing that the small language imperfections add up in more heterogeneous teams which hinders performance. However, this confirms that communicating with a shared language allows better understanding with unknown partners.

5.5.4 Interaction

A desired feature for communicating agents is the possibility of interacting with them. If the agents' communication system is interpretable, then we can try using this system for interaction. Making agents learn a language we designed solves the issue of interpretability and should make interaction straightforward. To evaluate this quantitatively, we investigate the impact of human messages on the agents' behaviour.

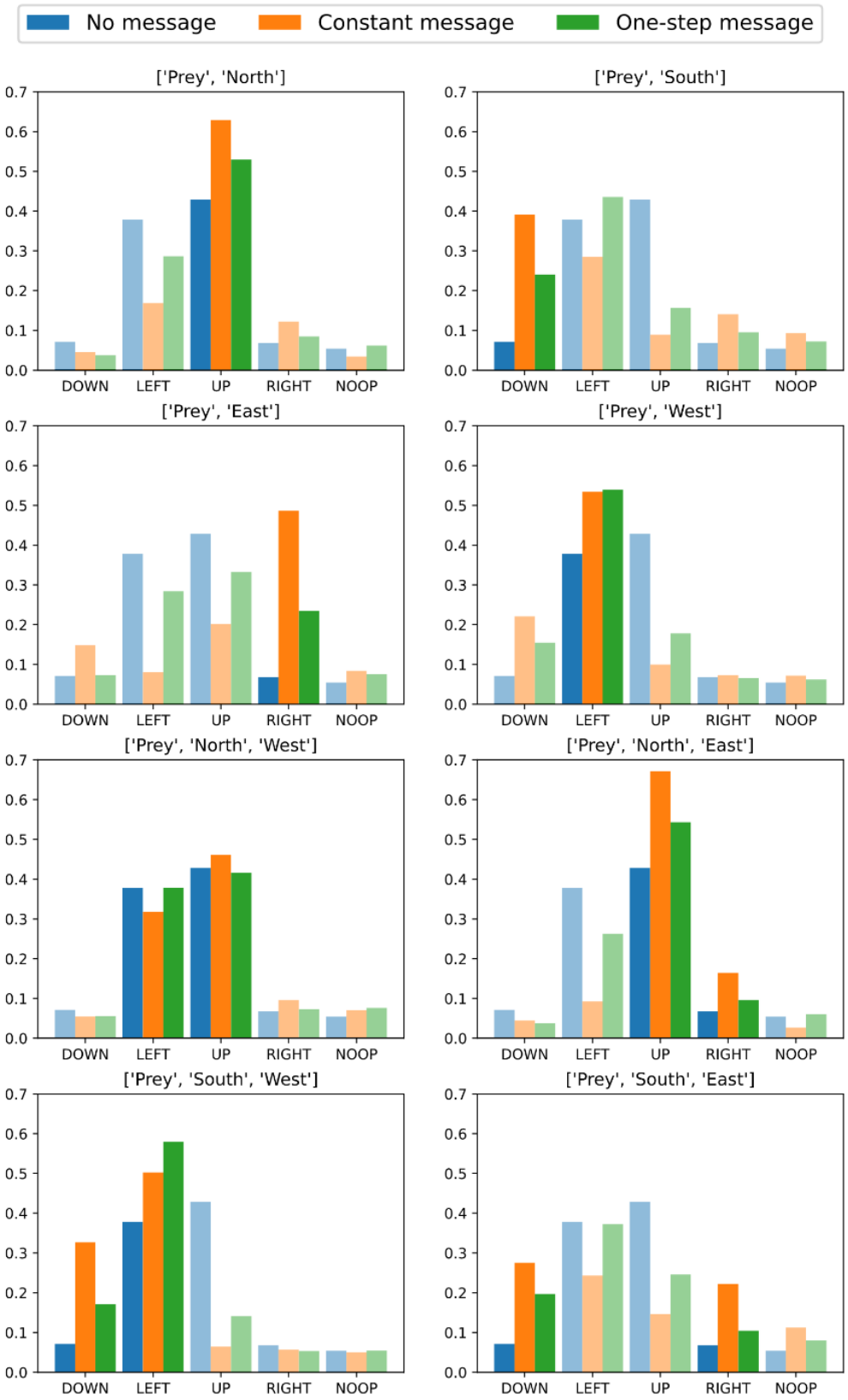


Figure 5.8: Results of the interaction experiments. Each plot shows the impact of the corresponding messages on action probabilities. "No message" is a control group that receives no message (same for all graphs). "Constant message" receives the message at every step. "One-step message" receives it only on the first step. The highlighted bars correspond to the direction that should be favoured given the input message.

Protocol

To measure this, we put a pre-trained team of `Language` agents in an empty environment, send a message into the communication channel, and record the actions performed in the following 5 steps. Agents are spawned in central positions and provided a message indicating that a prey is located in one region of the environment. We execute these short episodes a large number of times and record the performed actions. In Figure 5.8, we display the observed probabilities of each action following the given messages. We compare the action probabilities for agents receiving no message ("No messages"), agents constantly receiving the messages during 5 steps ("Constant message"), and agents that receive the message on the first step and no message during the four remaining steps ("One-step message"). The latter can be indicative of the good use of memory by the agents.

We found that four-agent teams are difficult to evaluate this way because they tend to have different agents focusing on the four corners of the map. Thus, regardless of the input message, action probabilities will balance themselves. Therefore, we train a team of two `Language` agents in the 15×15 map and evaluate them. Because these two agents need to catch preys together, their average action probabilities should be impacted by the input messages. The larger map size will help increase the contribution of communication over exploration.

Results

Looking at the graph of Figure 5.8, we find that messages almost always impact the agents' behaviour accordingly. Only the "Prey North West" message has no clear impact, which may be linked to the fact that these agents seem to be biased toward the "LEFT" and "UP" actions, as seen in the control group. For all other messages, the corresponding action probabilities are significantly increased. Even the "One-step message" impacts the agents' actions, showing that agents have learnt to memorise previously received messages. Overall, it is clear that we can impact the behaviour of `Language` agents by communicating with them.

5.6 Discussions

5.6.1 Analysing and Discussing Results

With this set of experiments, we have shown that:

- **Agents successively learn to use a language** that describes their observations, concurrently to learning an embodied multi-agent task.
- **Language improves communication**. Agents using language learn faster than the ones learning emergent communication. This seems natural, as agents are given an efficient communication system and are trained to not deviate from it. They do not experience the struggle of finding a consensus on how to communicate, while also learning a policy. Thus, language-based communication makes training more stable.
- **Learning language allows faster learning** by guiding the learning of efficient representation. By learning language, agents (i) ground their representations of the observation space in concepts expressed in language and (ii) ground their representations of language messages in their knowledge of the observation space.

- **Using a pre-defined language grants multiple desired features** to our agents: better adaptation to new environments or partners, interpretability, and interaction.

Of course, all these advantages brought by language come at the cost of providing the supervision required for learning this language. In our case, the cost is two-fold. First, we need the oracle to generate the language descriptions used for learning language. Relying on such rule-based systems restricts us to simple environments with relatively poor language structures. This is not necessarily an issue: simple robotic tasks might not benefit from using more complex languages. However, if more complex tasks are envisioned, more capable techniques may be required. In the robotic setting, for instance, many recent works have provided datasets (Nair et al., 2022) and models (Zeng et al., 2022) for captioning images related to robot behaviour. Second, we pay the cost of training and using language-based tools. The sequenced structure of language makes its processing quite slow. In our case, learning to communicate with language has doubled the training time compared to the continuous emergent communication variant. We still consider that the gains of language-based agents, in their understanding of the world and communication skills, can justify these costs. However, it would still be beneficial to try reducing them. In the next section, we provide some avenues of improvement for better language learning.

About the emergent communication baseline that we define, it is important to note that they could be improved in many ways, as shown in the extensive related literature (see Section 3.4.5 and 5.2.2). Our point is not to say that emergent communication does not work, nor that it has no use. Rather, we want to show that, for robotic-like settings, it fails to fit the requirements on multiple points: interpretability, generalisation, and interaction. Improving emergent communication on these points is possible but will have an important cost with no guaranteed perfect solution. This cost might as well be paid for learning a language that improves communication, grounds representation learning, and provides many desired features to our agents.

Lastly, to improve our comparison with emergent communication and back our claims further, we could benefit from extending our experiments and analysis of the different communication systems. Experimenting in a more complex setting with more diverse entities and objects could highlight the advantages of learning a pre-defined language further. Additionally, improving our analysis of the learnt communication mechanisms, by measuring the informational content of messages and their impact on other agents (Jaques et al., 2019; Lowe et al., 2019), would be helpful to better showcase the shortcomings of emergent communication systems and the qualities of pre-defined languages.

5.6.2 Perspectives for Improving Language Learning

Our implementation of language learning has one flaw that increases its costs substantially: it does not exploit the acquired language examples sufficiently. The data used for training on the language objectives is the same as the one used for training PPO, i.e., we use only the data gathered between each training phase to train the language modules. A more sensible approach would be to build a dataset of language examples throughout training, without discarding data after the training phase. This could improve language learning by training on more diverse data. Importantly, this would allow limiting the cost of annotation by limiting the number of calls to the oracle. As in the generalisation experiment (see Section 5.5.2), we could stop the generation of language examples after a given number of training steps, but continue training on

the language objectives indefinitely using the buffered language examples. This would prevent the loss of language quality observed in the "frozen" agents in Figure 5.7.

Another way of improving language learning would be to use more advanced techniques for the language modules. In our architecture, we use one-layer GRUs and small MLPs for these modules. In other words, we use the most basic neural networks possible for implementing the language modelling skills. This is largely sufficient for learning a simple language like the one we have. But, more complex techniques like deeper neural networks or Transformers would improve language learning and allow learning more complex languages.

5.6.3 Conclusion and Future Works

To conclude, in this chapter, we have proposed a method for enabling embodied agents to learn a pre-defined language while training on a cooperative task with MADRL. By learning this language, agents gain the ability to communicate efficiently about their local observations, as the language has been purposefully designed to express meanings that serve the task at hand. Additionally, agents can exploit the structures in the given language to ground their understanding of the world, allowing more efficient representation learning. In addition to these learning improvements, training language-augmented agents also presents multiple practical advantages. Because communication is carried out in a known language, interpreting the communication acts is straightforward. This works in the other way, with agents being able to receive information from human experimenters. Language-augmented agents are also able to better generalise their knowledge, enabling faster adaptation to changes in the environment. Lastly, language-based communication enables easier cooperation with unknown partners, by providing a shared communication system to support social interactions.

With this work, we showcase the great potential of language-augmented agents for the multi-agent setting. This should encourage further works on this subject to explore this potential in more depth. In future works, we will study ways to expand the language capabilities of our agents. As human beings, artificial agents would benefit from sharing their intentions with their partners. Taking inspiration from works in language-augmented, goal-driven learning, we could extend our architecture to allow agents to communicate about intents. Another area for improvement is the diversity of communicating behaviours. Fine-tuning language modules with RL could allow more diverse communication strategies. Finally, the generalisation abilities of language-grounded agents are particularly interesting. As a single language can describe similar situations in completely different environments, it can support the transfer of a learnt policy to new domains, by grounding different state spaces in common linguistic concepts. This could help to drastically reduce the cost of learning multi-agent strategies in realistic robotic domains.

Chapter 6

Conclusion

Summary

Throughout this thesis, we have studied techniques to tackle multi-agent robotic problems with deep reinforcement learning. Being anchored in robotics motivates a specific perspective on these learning problems, asking: *how should we design multi-agent deep reinforcement learning algorithms for enabling their applications to robotic settings?* Robotic agents live in complex, dynamic environments, alongside humans and other robots. They need to handle large observation and action spaces, incomplete information about their environment, sparse reward signals, and dynamic settings. Therefore, they require strong representation learning capacities to make sense of the information they receive and the actions they can perform. They need efficient algorithms to learn what behaviour can lead to what outcomes, and how to behave to fulfill their objectives efficiently. They need to generalise their knowledge and experience, to handle changes in their environments and new partners. Finally, for engaging in human-populated environments, they require the right tools for observing, understanding, and interacting with humans.

Learning such agents with reinforcement learning tools is a challenge. Deep neural networks have allowed expanding the potential of reinforcement learning for training better policies in more complex settings. But, there are still many challenges for reinforcement learners to obtain more general and robust learning abilities. The multi-agent problem is one of them. Having multiple learning agents raises several issues at once, amplifying the limitations of reinforcement learning techniques. To mitigate these issues, research on multi-agent deep reinforcement learning slowly builds a set of tools for improving the training of multi-agent policies and the design of more capable agents.

To progress in this direction, we first presented an analysis of the field of multi-agent deep reinforcement learning, emphasising its challenges and reviewing its main lines of research. Importantly, we have highlighted the gap between state-of-the-art algorithms and their practical applications in robotics. Because of the great challenges faced when learning multi-agent policies, works in the field often neglect real-world complexities. While this is inevitable for building our way to more capable algorithms, we may also benefit from rethinking some of our approaches and try embracing some of the issues we face. We have identified four crucial matters: benchmarking, exploration, generalisation, and interaction; that all should be addressed for improving the research domain, the algorithms, and the agents that result from them.

Following this, we have contributed two new methods for improving multi-agent learning. First, by tackling the problem of multi-agent exploration. Similar to previous works, we showed that state-of-the-art multi-agent deep reinforcement learning algorithms suffer from the problem of relative overgeneralisation. Because of poor

exploration strategies, they are unable to efficiently find optimal joint policies that require strong coordination of agents to be revealed. To tackle this issue, we introduced the Joint Intrinsic Motivation method to incite agents to explore the space of joint observations. This method demonstrated an important idea: local exploration is inefficient in multi-agent settings because outcomes depend on multiple agents. However, as we have shown, actively exploring the joint-observation space helps for finding new coordinated behaviours. This should motivate further study of joint exploration strategies.

Finally, we tackled the problem of multi-agent communication. Communication is a crucial skill for agents to face the challenges of multi-agent settings. For human beings, it enables the sharing of local information, the expression of intents, and the transmission of knowledge. We proposed to take inspiration from humans by learning to communicate with a pre-defined language, allowing efficient information transmission and communicating with unknown agents and humans. We intentionally defined agents that learn language during their training, as opposed to using pre-trained language modules, allowing language training to also serve as guidance for learning better representations of meaning from the world. With these language-augmented agents, we gain important qualities for (i) improving learning: by learning faster how the world works, how to communicate efficiently, and how to adapt to changes in the environment; and (ii) improving communication: by allowing communication with new partners and human-agent interactions.

Discussions

Working and experimenting with deep reinforcement learning methods consistently shows that human-like intelligence, as in reasoning and behaving like a human being would, does not naturally emerge from reinforcement learning training, at least with the tools we have at our disposal. Often, we undoubtedly expect agents to converge towards some logical (to us) behaviour, and remain confused by the results of training. *Why would they not go directly to the goal, when it provides them with a great reward?* We can find two sound examples of this in the present work:

- In environments with sparse rewards, we expect agents to quickly recognise the states associated with positive rewards, and we often struggle to predict how hard a given task will be. This is the case with the problem of relative overgeneralisation, where the multi-agent settings exacerbates the sparsity of positive reward signals by requiring coordinated exploration to discover them.
- When training agents with differentiable emergent communication, we can expect them to learn to communicate about elements important for solving the task, to discard redundant or irrelevant information, and to listen to incoming information for improving their policy; experiments show that learning to maximise returns is insufficient to guarantee such features.

Once the unexpected outcomes have been observed, logical explanations often arise by themselves. For sparse environments, one explanation is that reinforcement learning is inefficient. It needs to observe a certain outcome many times to learn efficiently from it. Intrinsic motivation can help by guiding the agents to develop a more active exploratory strategy. But, this is limited as it does not teach agents the concept of exploration – what it is, when it should be used –, it only induces some behaviour change by essentially modifying the task. To go beyond this, techniques like goal-directed RL and hierarchical learning could help build agents that can switch their strategy momentarily to explore more when required.

For investigating the failures of differentiable emergent communication, looking at language evolution provides some explanations, as explained in Section 5.2.1. Note that, while we argued that learning language was a preferable solution for multi-agent communication, it should not be seen as a magical solution for teaching communication to reinforcement learning agents. In the algorithm we proposed in Chapter 5, learning of how information coming from other agents should be used for action-selection is still entrusted to the reinforcement learning algorithm. Thus, we do not control fully how messages will be used. For example, agents may learn to ignore the incoming information – which was not the case in our setting, but has been observed previously (Jaques et al., 2019). To gain further control over the training of reinforcement learners, we may expand our use of language grounding by adding other auxiliary learning objectives.

Future Works

Studying the fields of multi-agent learning, deep reinforcement learning, robotics, and language at the same time feels both daunting and exciting. These domains are rich of concepts, techniques, and experiments, that should all be understood to be used, merged, and reproduced for advancing the state of research. But, at the same time, at the intersection of these fields appear many avenues for improvements and new approaches to shape the capabilities of agents learning in groups. Looking into each problem sparks the desire to dig deeper and contribute to the collective progress. In future works, there are three main directions I would like to pursue.

First, because communication is such an important skill in multi-agent interactions, and language plays such a big part in human learning, we should continue working on language-augmented agents in social settings. As described in Section 5.2.4, language can play a central role in human learning, as a way to understand our world and internalise observed behaviour in social interactions. To fully enable this kind of learning in artificial agents, many extensions to our work may be envisioned. Agents should be able to communicate about their actions. They should use their language abilities to learn from their interactions with the world and other agents. Fulfilling these requirements will require thorough studies and experiments on how language can be used to augment learning agents.

Second, a problem that we have not tackled during this thesis is the learning of hierarchical policies for allowing to reason on different levels of abstraction. Hierarchical agent architectures allow dissociating the learning of high-level strategy – *what goal should I pursue at the moment?* – and low-level action-selection – *given my current objective, how should I command my actuators?*. This has been widely studied in the single-agent case (Pateria et al., 2021), but is also very relevant to multi-agent settings for handling complex social interactions. This can also be related to language learning. Language is used to describe high-level concepts (e.g., objects) that arise from low-level signals (e.g., sensor data). Investigating the mechanisms of these two levels of abstraction separately and their different roles in learning and behaving can help us design more capable social agents.

Lastly, I believe that model-based multi-agent reinforcement learning should be investigate further. We humans constantly use models of the world and other entities to decide what we want to do – *what actions should I perform?* –, to understand the situations we are in –*what led to the present state?* –, our role or possibilities in these situations – *what goal can I, and should I pursue?* –, and to learn from our experience – *what should I have done differently?*. Of course, all these questions are central

in our learning and conducting of social interactions. Thus, artificial agents would benefit from having similar capacities to handle multi-agent learning. The reason why they are mostly not equipped with models is technical: it is extremely difficult to model complex, dynamic systems like multi-agent environments. However, despite this difficulty, the intuitive and observed benefits of models should still motivate us to develop model-based multi-agent approaches. Agent modelling techniques, presented in Section 3.4.6, are a step towards this objective. The great progress of single-agent model-based RL, provided by deep learning techniques (Hafner et al., 2023), should also be a source of inspiration for experimenting with models in social environments.

As always, these three avenues of research, and their potential intersections, pave the way for compelling problems to solve, challenges to overcome, and, hopefully, some interesting results along the way. So, let us get to work!

Appendix A

Joint Intrinsic Motivation

A.1 Hyperparameters

Table A.1: Hyperparameters used in the climbing game.

Hyperparameter	Algorithm		
	JIM	JIM 4 agents	LIM
Intrinsic reward weight β		1	
Encoding dim $D_{\phi/\psi}$	64	64	32
Hidden dim D_{hidden}	128	256	64
Scaling factor α		**0.5**, 0.6	
Intrinsic reward learning rate α_{int}	0.0001	0.0001, **0.0002**	0.0001

Table A.2: Hyperparameters used in the cooperative box pushing scenario.

Hyperparameter	Algorithm	
	JIM	LIM
Intrinsic reward weight β	1	1
Encoding dim $D_{\phi/\psi}$	64	32
Hidden dim D_{hidden}	128	64
Scaling factor α	0.5	
Intrinsic reward learning rate α_{int}	0.0001	

Table A.3: Hyperparameters used in the coordinated placement scenario.

Hyperparameter	Algorithm			
	JIM	LIM	JIM-LLEC	JIM-EEC
Intrinsic reward weight β	1, **2**, 4	1, **4**, 8	1, **3**	**0.1**, 1
Encoding dim $D_{\phi/\psi}$	64	36	64	64
Hidden dim D_{hidden}	512	256	512	512
Scaling factor α	0.5			
Intrinsic reward learning rate α_{int}	0.0001			

Tables 1 to 3 present the values chosen for hyperparameters. We only list hyperparameters specific to the intrinsic reward module, as for QMIX we use the default hyperparameters described in the original paper (Rashid et al., 2018). When we performed some search over a specific hyperparameter, we put the list of all the values we tried and put the best one in bold. Here, we detail the different parameters presented:

- Intrinsic reward weight β: weight of the intrinsic reward r_i against the extrinsic reward r_e in the reward given to agents: $r_t = r_t^e + \beta r_t^{int}$.

- Encoding dimension $D_{\phi/\psi}$: dimension of the output of the embedding networks ϕ and ψ used in N_{LLEC} and N_{EEC} respectively (see Section 3).
- Hidden dimension D_{hidden}: dimension of the hidden layers in the embedding network ϕ and ψ used in N_{LLEC} and N_{EEC} respectively.
- Scaling factor α: parameter used in the definition of N_{LLEC} (see Eq. (6)). It controls how much novelty gain we want agents to find between each step.
- Intrinsic reward learning rate α_{int}: learning rate used for training the intrinsic reward module.

Appendix B

Language-augmented multi-agent learning

B.1 Additional training results on Predator-Prey

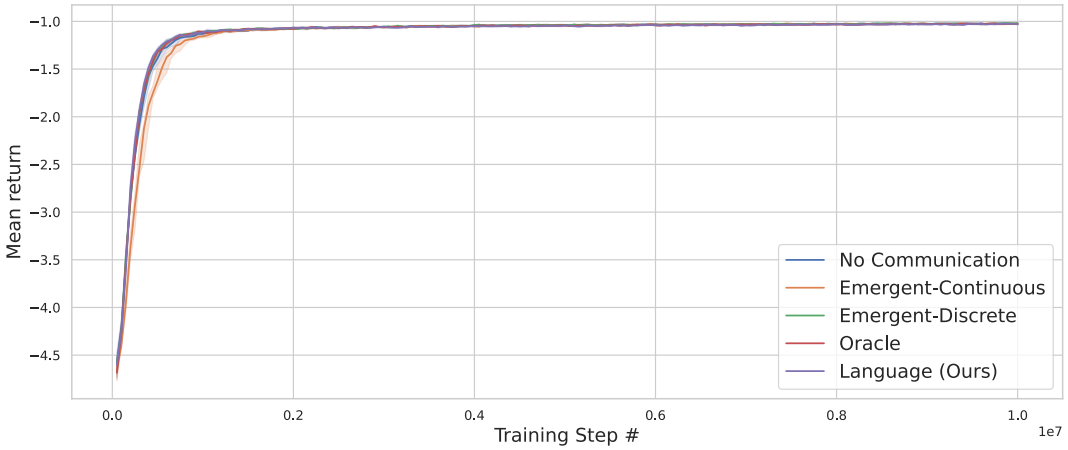


Figure B.1: Training performance in the 15x15 version of Predator-Prey (15 runs each, with median and confidence interval 95% shown). No variant is able to find a successful strategy in this larger environment when trained from scratch. They all converge towards the suboptimal strategy of avoiding preys to prevent penalties from missed captures. This results in a final return of -1: $T \times p_{step} = 100 \times -0.01$, with T the number of steps in the episode and p_{step} the penalty for each step.

B.2 Implementation details

B.2.1 Detailed agent architecture

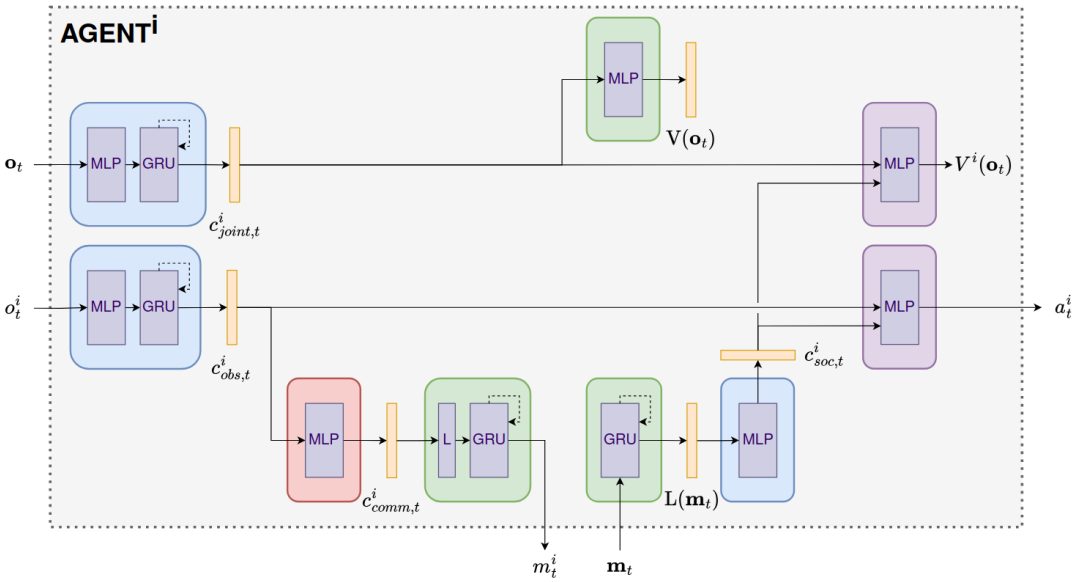


Figure B.2: Detailed architecture of the language-augmented architecture presented in Section 5.1. Neural networks are shown in purple, with MLP for multi-layer perceptron, GRU for gated recurrent unit, and L for a single neural network layer. All MLPs and GRUs have two hidden layers.

B.2.2 Hyperparameters

Table B.1: Hyperparameters used for training the different agent versions.

Hyperparameter	Algorithm				
	No Comm	Em-Cont	Em-Disc	Oracle	Lang
ε-oracle strategy					
σ	-	-	-	-	2
Language training					
Lang. batch size	-	-	-	1024	
Embedding dimension	-	-	-	4	
Learning rate	-	-	-	0.007	
Agent architecture					
Context dimension C	-	2	16	16	16
Hidden dimension H	64				
PPO training					
Nb. of parallel rollouts	250				
Learning rate	0.0005				
Nb. of PPO epochs	15				
Nb. of mini-batches	1				
Nb. of warm-up steps	50000				

In Table B.1, we present the main hyperparameters used in the experiments of Chapter 5. For training with PPO, many other hyperparameters are used, but we show only the ones that differ from the original implementation. The σ parameter used for the ε-oracle strategy (as described in the next section) is fixed to 2 in all experiments. Specific hyperparameters are defined for training the language modules: the sizes of the batches, the dimension of the embedding layers in the decoder and language encoder, and the learning rate applied on these modules. In the agent architecture, the context dimension C used for c^i_{comm} and c^i_{soc}, defined in Section 5.3.2, is set to 16 in all language-based agents and the `Emergent-Discrete` agents, as they use the same architecture as the language-based agents. For the `Emergent-Continuous` agents, it is set to 2, as c^i_{comm} serves as a continuous message. Larger values were also tried (4, 6, 8, and 16), but they performed worse. For the training of agents with the objective of PPO, we use 250 parallel rollouts: i.e., 250 parallel episodes run between each training phase. Each training phase has 15 consecutive training updates with 1 mini-batch (i.e., all the data collected during rollouts is used in one single batch). Finally, we begin training with a "warm-up" phase of 50000 steps, during which the learning rate is lowered by a factor of 10^{-2}.

B.3 ε-oracle parameter decay

In the ε-oracle communication strategy (described in Section 5.4.3), the parameter ε is decayed throughout training with a sigmoid-like function defined by:

$$\varepsilon(t) = \frac{1}{\left[1 + e^{-\frac{16}{\sigma}\left(1-\frac{t}{T}\right)}\right]^{20}}, \tag{B.1}$$

with t the current step of training, T the total number of training steps planned, and σ the parameter for controlling the rate of the decay (shown in Figure 5.4.3), fixed to 2 in our experiments.

Bibliography

Abbeel, P., Coates, A., Quigley, M., & Ng, A. (2006). *An application of reinforcement learning to aerobatic helicopter flight*. In B. Schölkopf, J. Platt, & T. Hoffman (Eds.), *Advances in neural information processing systems* (Vol. 19). MIT Press.

Ackermann, J., Gabler, V., Osa, T., & Sugiyama, M. (2019). *Reducing overestimation bias in multi-agent domains using double centralized critics* [arxiv:1910.01465].

Agarwal, A., Gurumurthy, S., Sharma, V., Lewis, M., & Sycara, K. (2019). *Community regularization of visually-grounded dialog*. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 1042–1050.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., ... Zeng, A. (2022). *Do as i can, not as i say: Grounding language in robotic affordances* [arxiv:2204.01691].

Akakzia, A., Colas, C., Oudeyer, P.-Y., Chetouani, M., & Sigaud, O. (2021). *Grounding language to autonomously-acquired skills via goal generation*. *ICLR 2021 - Ninth International Conference on Learning Representation*.

Albrecht, S. V., & Ramamoorthy, S. (2013). *A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems*. *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, 1155–1156.

Albrecht, S. V., & Stone, P. (2018). *Autonomous agents modelling other agents: A comprehensive survey and open problems*. *Artificial Intelligence*, *258*, 66–95.

Amato, C., Konidaris, G., Kaelbling, L. P., & How, J. P. (2019). *Modeling and planning with macro-actions in decentralized pomdps*. *Journal of Artificial Intelligence Research*, *64*, 817–859.

Andres, A., Villar-Rodriguez, E., & Ser, J. D. (2022). *An evaluation study of intrinsic motivation techniques applied to reinforcement learning over hard exploration environments* [arxiv:2205.11184].

Andrychowicz, O. M., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., & Zaremba, W. (2020). *Learning dexterous in-hand manipulation*. *The International Journal of Robotics Research*, *39*(1), 3–20.

Apperly, I. (2011). *Mindreaders: The cognitive basis of "theory of mind"*. Psychology Press.

Aru, J., Labash, A., Corcoll, O., & Vicente, R. (2023). *Mind the gap: Challenges of deep learning approaches to theory of mind*. *Artificial Intelligence Review*, *56*(9), 9141–9156.

Austin, J. L. (1975). *How to do things with words* (J. O. Urmson & M. Sbisà, Eds.). Harvard university press.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., & Blundell, C. (2020). *Agent57: Outperforming the Atari human benchmark*. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 507–517, Vol. 119). PMLR.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., & Blundell, C. (2020). *Never give up: Learning directed exploration strategies*. *8th International Conference on Learning Representations*.

Bagnell, J., & Schneider, J. (2001). *Autonomous helicopter control using reinforcement learning policy search methods*. *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, *2*, 1615–1620 vol.2.

Baird, L. (1995). *Residual algorithms: Reinforcement learning with function approximation*. *Machine Learning Proceedings 1995*, 30–37.

Bard, N., Foerster, J. N., Chandar, S., Burch, N., Lanctot, M., Song, H. F., Parisotto, E., Dumoulin, V., Moitra, S., Hughes, E., Dunning, I., Mourad, S., Larochelle, H., Bellemare, M. G., & Bowling, M. (2020). *The hanabi challenge: A new frontier for ai research*. *Artificial Intelligence*, *280*, 103216.

Barrett, S., Rosenfeld, A., Kraus, S., & Stone, P. (2017). *Making friends on the fly: Cooperating with new teammates*. *Artificial Intelligence*, *242*, 132–171.

Barsalou, L. W., Niedenthal, P. M., Barbey, A. K., & Ruppert, J. A. (2003). *Social embodiment*. In *Psychology of learning and motivation volume 43* (pp. 43–92). Elsevier.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). *Neuronlike adaptive elements that can solve difficult learning control problems*. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-13*(5), 834–846.

Baumli, K., Baveja, S., Behbahani, F., Chan, H., Comanici, G., Flennerhag, S., Gazeau, M., Holsheimer, K., Horgan, D., Laskin, M., Lyle, C., Masoom, H., McKinney, K., Mnih, V., Neitz, A., Pardo, F., Parker-Holder, J., Quan, J., Rocktäschel, T., ... Zhang, L. (2023). *Vision-language models as a source of rewards* [arxiv:2312.09187].

Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). *The arcade learning environment: An evaluation platform for general agents*. *Journal of Artificial Intelligence Research*, *47*, 253–279.

Bellemare, M. G., Dabney, W., & Munos, R. (2017). *A distributional perspective on reinforcement learning*. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 449–458, Vol. 70). PMLR.

Bellman, R. (1957). *A markovian decision process*. *Journal of Mathematics and Mechanics*, *6*(5), 679–684.

Bellman, R. (1966). *Dynamic programming*. *Science*, *153*(3731), 34–37.

Ben-Ari, M. (2006). *Principles of concurrent and distributed programming (2nd edition)*. Addison-Wesley Longman Publishing Co., Inc.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). *Curriculum learning*. *Proceedings of the 26th Annual International Conference on Machine Learning*.

Bettini, M., Prorok, A., & Moens, V. (2023). *Benchmarl: Benchmarking multi-agent reinforcement learning* [arxiv:2312.01472].

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer New York, NY.

Böhmer, W., Kurin, V., & Whiteson, S. (2020). *Deep coordination graphs*. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 980–991, Vol. 119). PMLR.

Botoko Ekila, J. (2024). *Emergence of linguistic conventions in multi-agent systems through situated communicative interactions*. *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, 2725–2727.

Bouchacourt, D., & Baroni, M. (2018). *How agents see things: On visual representations in an emergent language game*. *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 981–985.

Bousquet, F., & Le Page, C. (2004). *Multi-agent simulations and ecosystem management: A review*. *Ecological Modelling*, *176*(3–4), 313–332.

Bowling, M., & Veloso, M. (2001). *Rational and convergent learning in stochastic games*. *International joint conference on artificial intelligence*, *17*(1), 1021–1026.

Bowling, M., & Veloso, M. (2002). *Multiagent learning using a variable learning rate*. *Artificial Intelligence*, *136*(2), 215–250.

Brembs, B. (2010). *Towards a scientific concept of free will as a biological trait: Spontaneous actions and decision-making in invertebrates*. *Proceedings of the Royal Society B: Biological Sciences*, *278*(1707), 930–939.

Brent, M. R., & Siskind, J. M. (2001). *The role of exposure to isolated words in early vocabulary development*. *Cognition*, *81*(2), B33–B44.

Brighton, H., Smith, K., & Kirby, S. (2005). *Language as an evolutionary system*. *Physics of Life Reviews*, *2*(3), 177–226.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Chen, X., Choromanski, K., Ding, T., Driess, D., Dubey, A., Finn, C., Florence, P., Fu, C., Arenas, M. G., Gopalakrishnan, K., Han, K., Hausman, K., Herzog, A., Hsu, J., Ichter, B., … Zitkovich, B. (2023). *Rt-2: Vision-language-action models transfer web knowledge to robotic control*. In J. Tan, M. Toussaint, & K. Darvish (Eds.), *Proceedings of the 7th conference on robot learning* (pp. 2165–2183, Vol. 229). PMLR.

Brown, G. W. (1951). *Iterative solution of games by fictitious play*. *Proceedingsof the Conference on Activity Analysis of Production and Allocation, Cowles CommissionMonograph 13*, 374–376.

Brunke, L., Greeff, M., Hall, A. W., Yuan, Z., Zhou, S., Panerati, J., & Schoellig, A. P. (2022). *Safe learning in robotics: From learning-based control to safe reinforcement learning*. *Annual Review of Control, Robotics, and Autonomous Systems*, *5*(1), 411–444.

Burda, Y., Edwards, H., Storkey, A., & Klimov, O. (2019). *Exploration by random network distillation* [arxiv:1810.12894]. *7th International Conference on Learning Representations*.

Busoniu, L., Babuska, R., De Schutter, B., & Ernst, D. (2017). *Reinforcement learning and dynamic programming using function approximators*. CRC press.

Cao, K., Lazaridou, A., Lanctot, M., Leibo, J. Z., Tuyls, K., & Clark, S. (2018). *Emergent communication through negotiation*. *6th International Conference on Learning Representations*.

Carr, J. W., Smith, K., Cornish, H., & Kirby, S. (2016). *The cultural evolution of structured languages in an open-ended, continuous world*. *Cognitive Science*, *41*(4), 892–923.

Carroll, M., Shah, R., Ho, M. K., Griffiths, T., Seshia, S., Abbeel, P., & Dragan, A. (2019). *On the utility of learning about humans for human-ai coordination*. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R.

Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Carta, T., Oudeyer, P.-Y., Sigaud, O., & Lamprier, S. (2022). *Eager: Asking and answering questions for automatic reward shaping in language-guided rl* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh, Eds.). *35*, 12478–12490.

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., & Oudeyer, P.-Y. (2023). *Grounding Large Language Models in Interactive Environments with Online Reinforcement Learning*. *Proceedings of Machine Learning Research*, *202*(3676-3713).

Chaabouni, R., Kharitonov, E., Dupoux, E., & Baroni, M. (2019). *Anti-efficient encoding in emergent communication*. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Chaabouni, R., Strub, F., Altché, F., Tarassov, E., Tallec, C., Davoodi, E., Mathewson, K. W., Tieleman, O., Lazaridou, A., & Piot, B. (2022). *Emergent communication at scale*. *International Conference on Learning Representations*.

Chebotar, Y., Handa, A., Makoviychuk, V., Macklin, M., Issac, J., Ratliff, N., & Fox, D. (2019). *Closing the sim-to-real loop: Adapting simulation randomization with real world experience*. *2019 International Conference on Robotics and Automation (ICRA)*.

Chen, L., Wu, P., Chitta, K., Jaeger, B., Geiger, A., & Li, H. (2023). *End-to-end autonomous driving: Challenges and frontiers* [arxiv:2306.16927].

Chen, Y. F., Liu, M., Everett, M., & How, J. P. (2017). *Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning*. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 285–292.

Chiappa, A. S., Marin Vargas, A., Huang, A., & Mathis, A. (2023). *Latent exploration for reinforcement learning*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 56508–56530, Vol. 36). Curran Associates, Inc.

Christiano, P. F., Leike, J., Brown, T., Martic, M., Legg, S., & Amodei, D. (2017). *Deep reinforcement learning from human preferences*. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Christianos, F., Schäfer, L., & Albrecht, S. (2020). *Shared experience actor-critic for multi-agent reinforcement learning*. *Advances in Neural Information Processing Systems 33 (NeurIPS 2020)*, 10707–10717.

Christiansen, M. H., & Chater, N. (2008). *Language as shaped by the brain*. *Behavioral and Brain Sciences*, *31*(5), 489–509.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). *Empirical evaluation of gated recurrent neural networks on sequence modeling* [arxiv:1412.3555].

Claus, C., & Boutilier, C. (1998). *The dynamics of reinforcement learning in cooperative multiagent systems*. *AAAI/IAAI*, *1998*(746-752), 2.

Cobbe, K., Hesse, C., Hilton, J., & Schulman, J. (2020). *Leveraging procedural generation to benchmark reinforcement learning*. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 2048–2056, Vol. 119). PMLR.

Colas, C., Karch, T., Lair, N., Dussoux, J.-M., Moulin-Frier, C., Dominey, P., & Oudeyer, P.-Y. (2020). *Language as a cognitive tool to imagine goals in curiosity driven exploration*. In H. Larochelle, M. Ranzato, R. Hadsell, M. F.

Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 3761–3774, Vol. 33). Curran Associates, Inc.

Colas, C., Karch, T., Moulin-Frier, C., & Oudeyer, P.-Y. (2022). *Language and culture internalization for human-like autotelic ai*. *Nature Machine Intelligence*, *4*(12), 1068–1076.

Conitzer, V., & Sandholm, T. (2007). *Awesome: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents*. *Machine Learning*, *67*(1-2), 23–43.

Co-Reyes, J. D., Gupta, A., Sanjeev, S., Altieri, N., DeNero, J., Abbeel, P., & Levine, S. (2019). *Guiding policies with language via meta-learning*. *International Conference on Learning Representations*.

Coulom, R. (2006). *Efficient selectivity and backup operators in monte-carlo tree search*. In H. J. van den Herik, P. Ciancarini, & H. H. L. M. (Donkers (Eds.), *Computers and games* (pp. 72–83). Springer Berlin Heidelberg.

Crandall, J. W., Oudah, M., Tennom, Ishowo-Oloko, F., Abdallah, S., Bonnefon, J.-F., Cebrian, M., Shariff, A., Goodrich, M. A., & Rahwan, I. (2018). *Cooperating with machines*. *Nature Communications*, *9*(1).

D'Ambrosio, D. B., Abeyruwan, S., Graesser, L., Iscen, A., Amor, H. B., Bewley, A., Reed, B. J., Reymann, K., Takayama, L., Tassa, Y., Choromanski, K., Coumans, E., Jain, D., Jaitly, N., Jaques, N., Kataoka, S., Kuang, Y., Lazic, N., Mahjourian, R., ... Sanketi, P. R. (2024). *Achieving human level competitive robot table tennis* [arxiv:2408.03906].

Das, A., Gervet, T., Romoff, J., Batra, D., Parikh, D., Rabbat, M., & Pineau, J. (2019). *TarMAC: Targeted multi-agent communication*. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 1538–1546, Vol. 97). PMLR.

Das, A., Kottur, S., Moura, J. M. F., Lee, S., & Batra, D. (2017). *Learning cooperative visual dialog agents with deep reinforcement learning*. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Ding, D., Wei, X., Yang, Z., Wang, Z., & Jovanovic, M. (2021). *Provably efficient safe exploration via primal-dual policy optimization*. In A. Banerjee & K. Fukumizu (Eds.), *Proceedings of the 24th international conference on artificial intelligence and statistics* (pp. 3304–3312, Vol. 130). PMLR.

Doran, J., & Palmer, M. (1995). *The eos project: Integrating two models of palaeolithic social change*. *Artificial Societies: The ComputerSimulation of Social Life*, 103–125.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). *An image is worth 16x16 words: Transformers for image recognition at scale*. *International Conference on Learning Representations*.

Driess, D., Xia, F., Sajjadi, M. S. M., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., Huang, W., Chebotar, Y., Sermanet, P., Duckworth, D., Levine, S., Vanhoucke, V., Hausman, K., Toussaint, M., Greff, K., ... Florence, P. (2023). *Palm-e: An embodied multimodal language model*. In PMLR (Ed.), *Proceedings of the 40th international conference on machine learning* (Vol. 202).

Du, Y., Han, L., Fang, M., Dai, T., Liu, J., & Tao, D. (2019). *LIIR: Learning individual intrinsic reward in multi-agent reinforcement learning*. *Advances in Neural Information Processing Systems*, *32*.

Eccles, T., Bachrach, Y., Lever, G., Lazaridou, A., & Graepel, T. (2019). *Biases for emergent communication in multi-agent reinforcement learning*. *Advances in Neural Information Processing Systems*.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., & Clune, J. (2021). *First return then explore*. *Nature*, (590), 580–586.

Eiben, A. E., & Smith, J. E. (2003). *Introduction to evolutionary computing*. Springer Berlin Heidelberg.

Ellis, B., Cook, J., Moalla, S., Samvelyan, M., Sun, M., Mahajan, A., Foerster, J., & Whiteson, S. (2023). *Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 37567–37593, Vol. 36). Curran Associates, Inc.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., & Madry, A. (2020). *Implementation matters in deep policy gradients: A case study on ppo and trpo*. *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Eysenbach, B., Gupta, A., Ibarz, J., & Levine, S. (2019). *Diversity is all you need: Learning skills without a reward function*. *International Conference on Learning Representations*.

Farrell, J., & Rabin, M. (1996). *Cheap talk*. *Journal of Economic Perspectives*, *10*(3), 103–118.

Fatima, S. S., Wooldridge, M., & Jennings, N. R. (2008). *A linear approximation method for the shapley value*. *Artificial Intelligence*, *172*(14), 1673–1699.

Feine, J., Gnewuch, U., Morana, S., & Maedche, A. (2019). *A taxonomy of social cues for conversational agents*. *International Journal of Human-Computer Studies*, *132*, 138–161.

Finnsson, H., & Björnsson, Y. (2008). *Simulation-based approach to general game playing*. *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, 259–264.

Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., & Geist, M. (2021). *Adversarially guided actor-critic*. *9th International Conference on Learning Representations*.

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). *Counterfactual multi-agent policy gradients*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., & Whiteson, S. (2017). *Stabilising experience replay for deep multi-agent reinforcement learning*. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 1146–1155, Vol. 70). PMLR.

Foerster, J., Song, F., Hughes, E., Burch, N., Dunning, I., Whiteson, S., Botvinick, M., & Bowling, M. (2019). *Bayesian action decoder for deep multi-agent reinforcement learning*. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 1942–1951, Vol. 97). PMLR.

Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016a). *Learning to communicate to solve riddles with deep distributed recurrent q-networks* [arxiv:1602.02672].

Foerster, J. N., Assael, Y. M., de Freitas, N., & Whiteson, S. (2016b). *Learning to communicate with deep multi-agent reinforcement learning*. *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2145–2153.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., & Legg, S. (2018). *Noisy networks for exploration*. *International Conference on Learning Representations*.

Friston, K. (2010). *The free-energy principle: A unified brain theory?* *Nature Reviews Neuroscience*, *11*(2), 127–138.

Fudenberg, D., & Levine, D. K. (1995). *Consistency and cautious fictitious play*. *Journal of Economic Dynamics and Control*, *19*(5–7), 1065–1089.

Fujimoto, S., van Hoof, H., & Meger, D. (2018). *Addressing function approximation error in actor-critic methods*. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 1587–1596, Vol. 80). PMLR.

Galke, L., Ram, Y., & Raviv, L. (2022). *Emergent communication for understanding human language evolution: What's missing?* *Emergent Communication Workshop at ICLR 2022*.

García, J., Fern, & o Fernández. (2015). *A comprehensive survey on safe reinforcement learning*. *Journal of Machine Learning Research*, *16*(42), 1437–1480.

Gardner, R. A., & Gardner, B. T. (1984). *A vocabulary test for chimpanzees (pan troglodytes)*. *Journal of Comparative Psychology*, *98*(4), 381–404.

Gehring, J., Synnaeve, G., Krause, A., & Usunier, N. (2021). *Hierarchical skills for efficient exploration*. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 11553–11564, Vol. 34). Curran Associates, Inc.

Gibson, E., Futrell, R., Piantadosi, S. P., Dautriche, I., Mahowald, K., Bergen, L., & Levy, R. (2019). *How efficiency shapes human language*. *Trends in Cognitive Sciences*, *23*(5), 389–407.

Glorot, X., Bordes, A., & Bengio, Y. (2011). *Deep sparse rectifier neural networks*. In G. Gordon, D. Dunson, & M. Dudík (Eds.), *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323, Vol. 15). PMLR.

Gombolay, M., Bair, A., Huang, C., & Shah, J. (2017). *Computational design of mixed-initiative human–robot teaming that considers human factors: Situational awareness, workload, and workflow preferences*. *The International Journal of Robotics Research*, *36*(5–7), 597–617.

Gorsane, R., Mahjoub, O., de Kock, R. J., Dubb, R., Singh, S., & Pretorius, A. (2022). *Towards a standardised performance evaluation protocol for cooperative marl*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 5510–5521, Vol. 35). Curran Associates, Inc.

Greene, J., & Burleson, B. (2003). *Handbook of communication and social interaction skills*. Taylor & Francis.

Guestrin, C., Lagoudakis, M., & Parr, R. (2002). *Coordinated reinforcement learning*. *Proceedings of 19th International Conference on Machine Learning*, *2*, 227–234.

Gupta, A., Lanctot, M., & Lazaridou, A. (2021). *Dynamic population-based meta-learning for multi-agent communication with natural language*. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems*.

Gupta, A., Devin, C., Liu, Y., Abbeel, P., & Levine, S. (2017). *Learning invariant feature spaces to transfer skills with reinforcement learning*. *International Conference on Learning Representations*.

Gupta, J. K., Egorov, M., & Kochenderfer, M. (2017). *Cooperative multi-agent control using deep reinforcement learning*. In *Autonomous agents and multiagent systems* (pp. 66–83). Springer International Publishing.

Guss, W. H., Codel, C., Hofmann, K., Houghton, B., Kuno, N., Milani, S., Mohanty, S., Liebana, D. P., Salakhutdinov, R., Topin, N., Veloso, M., & Wang, P. (2019). *The minerl 2019 competition on sample efficient reinforcement learning using human priors* [arxiv:1904.10079].

Ha, D., Dai, A., & Le, Q. V. (2017). *Hypernetworks*. *5th International Conference on Learning Representations (ICLR)*.

Haarnoja, T., Moran, B., Lever, G., Huang, S. H., Tirumala, D., Humplik, J., Wulfmeier, M., Tunyasuvunakool, S., Siegel, N. Y., Hafner, R., Bloesch, M., Hartikainen, K., Byravan, A., Hasenclever, L., Tassa, Y., Sadeghi, F., Batchelor, N., Casarini, F., Saliceti, S., … Heess, N. (2024). *Learning agile soccer skills for a bipedal robot with deep reinforcement learning*. *Science Robotics*, *9*(89), eadi8022.

Haarnoja, T., Tang, H., Abbeel, P., & Levine, S. (2017). *Reinforcement learning with deep energy-based policies*. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 1352–1361, Vol. 70). PMLR.

Haarnoja, T., Zhou, A., Abbeel, P., & Levine, S. (2018). *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*. In J. Dy & A. Krause (Eds.), *Proceedings of the 35th international conference on machine learning* (pp. 1861–1870, Vol. 80).

Hafner, D., Lee, K.-H., Fischer, I., & Abbeel, P. (2022). *Deep hierarchical planning from pixels* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh, Eds.). *35*, 26091–26104.

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). *Mastering diverse domains through world models* [arxiv:2301.04104].

Hamann, H. (2018). *Swarm robotics: A formal approach*. Springer International Publishing.

Hamill, L., & Gilbert, N. (2015). *Agent-based modelling in economics*. John Wiley & Sons.

Han, S., Dastani, M., & Wang, S. (2023). *Model-based sparse communication in multi-agent reinforcement learning*. *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 439–447.

Hanjie, A. W., Zhong, V. Y., & Narasimhan, K. (2021). *Grounding language to entities and dynamics for generalization in reinforcement learning*. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 4051–4062, Vol. 139). PMLR.

Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., & Wang, Z. (2024). *Exploration in deep reinforcement learning: From single-agent to multiagent domain*. *IEEE Transactions on Neural Networks and Learning Systems*, *35*(7), 8762–8782.

Harnad, S. (1990). *The symbol grounding problem*. *Physica D: Nonlinear Phenomena*, *42*(1), 335–346.

Harsanyi, J. C., & Selten, R. (1988). *A General Theory of Equilibrium Selection in Games* (Vol. 1). The MIT Press.

Hausknecht, M., & Stone, P. (2015). *Deep recurrent q-learning for partially observable mdps*. *Sequential Decision Making for Intelligent Agents Papers from the AAAI 2015 Fall Symposium*.

Havrylov, S., & Titov, I. (2017). *Emergence of language with multi-agent games: Learning to communicate with sequences of symbols*. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2146–2156.

Heinrich, J., & Silver, D. (2016). *Deep reinforcement learning from self-play in imperfect-information games* [arxiv:1603.01121].

Henaff, M., Raileanu, R., Jiang, M., & Rocktäschel, T. (2022). *Exploration via elliptical episodic bonuses*. In A. H. Oh, A. Agarwal, D. Belgrave, & K. Cho (Eds.), *Advances in neural information processing systems* (pp. 37631–37646, Vol. 35).

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., & Meger, D. (2018). *Deep reinforcement learning that matters*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., & Silver, D. (2018). *Rainbow: Combining improvements in deep reinforcement learning*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Hester, T., & Stone, P. (2012). *Texplore: Real-time sample-efficient reinforcement learning for robots*. *Machine Learning*, *90*(3), 385–429.

Hester, T., Vecerik, M., Pietquin, O., Lanctot, M., Schaul, T., Piot, B., Horgan, D., Quan, J., Sendonaris, A., Osband, I., Dulac-Arnold, G., Agapiou, J., Leibo, J., & Gruslys, A. (2018). *Deep q-learning from demonstrations*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*(1).

Heyes, C. M., & Frith, C. D. (2014). *The cultural evolution of mind reading*. *Science*, *344*(6190).

Hill, F., Tieleman, O., von Glehn, T., Wong, N., Merzic, H., & Clark, S. (2021). *Grounded language learning fast and slow*. *International Conference on Learning Representations*.

Hochreiter, S., & Schmidhuber, J. (1997). *Long short-term memory*. *Neural Computation*, *9*(8), 1735–1780.

Hofbauer, J., & Sandholm, W. H. (2002). *On the global convergence of stochastic fictitious play*. *Econometrica*, *70*(6), 2265–2294.

Hong, Y., Jin, Y., & Tang, Y. (2022). *Rethinking individual global max in cooperative multi-agent reinforcement learning*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 32438–32449, Vol. 35). Curran Associates, Inc.

Hoshen, Y. (2017). *Vain: Attentional multi-agent predictive modeling*. *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2698–2708.

Hu, E. S., Chang, R., Rybkin, O., & Jayaraman, D. (2023). *Planning goals for exploration*. *The Eleventh International Conference on Learning Representations*.

Hu, H., & Foerster, J. N. (2020). *Simplified action decoder for deep multi-agent reinforcement learning*. *International Conference on Learning Representations*.

Hu, H., Yarats, D., Gong, Q., Tian, Y., & Lewis, M. (2019). *Hierarchical decision making by generating and following natural language instructions*. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Huang, W., Xia, F., Shah, D., Driess, D., Zeng, A., Lu, Y., Florence, P., Mordatch, I., Levine, S., Hausman, K., & Ichter, B. (2023). *Grounded decoding: Guiding text generation with grounded models for embodied agents* [arxiv:2303.00855].

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L.,

Levine, S., Hausman, K., & brian ichter. (2023). *Inner monologue: Embodied reasoning through planning with language models*. In K. Liu, D. Kulic, & J. Ichnowski (Eds.), *Proceedings of the 6th conference on robot learning* (pp. 1769–1782, Vol. 205). PMLR.

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., & Levine, S. (2021). *How to train your robot with deep reinforcement learning: Lessons we have learned*. *The International Journal of Robotics Research*, *40*(4-5).

Iqbal, S., De Witt, C. A. S., Peng, B., Boehmer, W., Whiteson, S., & Sha, F. (2021). *Randomized entity-wise factorization for multi-agent reinforcement learning*. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 4596–4606, Vol. 139). PMLR.

Iqbal, S., & Sha, F. (2019a). *Coordinated exploration via intrinsic rewards for multi-agent reinforcement learning* [arxiv:1905.12127].

Iqbal, S., & Sha, F. (2019b, September). *Actor-attention-critic for multi-agent reinforcement learning*. In K. Chaudhuri & R. Salakhutdinov (Eds.), *Proceedings of the 36th international conference on machine learning* (pp. 2961–2970, Vol. 97). PMLR.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J. Z., Silver, D., Hassabis, D., Kavukcuoglu, K., & Graepel, T. (2019). *Human-level performance in first-person multiplayer games with population-based deep reinforcement learning*. *Science*, *364*(6443), 859–865.

Jakobi, N., Husbands, P., & Harvey, I. (1995). *Noise and the reality gap: The use of simulation in evolutionary robotics*. In *Advances in artificial life* (pp. 704–720). Springer Berlin Heidelberg.

James, S., Wohlhart, P., Kalakrishnan, M., Kalashnikov, D., Irpan, A., Ibarz, J., Levine, S., Hadsell, R., & Bousmalis, K. (2019). *Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks*. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jang, E., Gu, S., & Poole, B. (2017). *Categorical reparameterization with gumbel-softmax*. *International Conference on Learning Representations*.

Jaques, N., Lazaridou, A., Hughes, E., Gulcehre, C., Ortega, P. A., Strouse, D., Leibo, J. Z., & de Freitas, N. (2019). *Social influence as intrinsic motivation for multi-agent deep reinforcement learning*. *Proceedings of the 36th International Conference on Machine Learning*, *97*, 3040–3049.

Jiang, J., & Lu, Z. (2018). *Learning attentional communication for multi-agent cooperation*. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc.

Jiang, J., & Lu, Z. (2022). *I2q: A fully decentralized q-learning algorithm*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 20469–20481, Vol. 35). Curran Associates, Inc.

Jiang, J., Su, K., & Lu, Z. (2024). *Fully decentralized cooperative multi-agent reinforcement learning: A survey* [arxiv:2401.04934].

Jiang, Y., Gu, S. (, Murphy, K. P., & Finn, C. (2019). *Language as an abstraction for hierarchical deep reinforcement learning*. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Jing, Y., Li, K., Liu, B., Zang, Y., Fu, H., FU, Q., Xing, J., & Cheng, J. (2024). *Towards offline opponent modeling with in-context learning*. *The Twelfth International Conference on Learning Representations*.

Jordan, J. (1991). *Bayesian learning in normal form games*. *Games and Economic Behavior*, *3*(1), 60–81.

Ju, H., Juan, R., Gomez, R., Nakamura, K., & Li, G. (2022). *Transferring policy of deep reinforcement learning from simulation to reality for robotics*. *Nature Machine Intelligence*, *4*(12), 1077–1087.

Jung, M. F., Difranzo, D., Shen, S., Stoll, B., Claure, H., & Lawrence, A. (2020). *Robot-assisted tower construction—a method to study the impact of a robot's allocation behavior on interpersonal dynamics and collaboration in groups*. *ACM Transactions on Human-Robot Interaction*, *10*(1), 1–23.

Kalai, E., & Lehrer, E. (1993). *Rational learning leads to nash equilibrium*. *Econometrica*, *61*(5), 1019.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., & Amodei, D. (2020). *Scaling laws for neural language models* [arxiv:2001.08361].

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., & Dabney, W. (2019). *Recurrent experience replay in distributed reinforcement learning*. *7th International Conference on Learning Representations*.

Karol Gregor, D. W., Danilo Jimenez Rezende. (2017). *Variational intrinsic control*. *5th International Conference on Learning Representations (ICLR)*.

Karten, S., Kailas, S., & Sycara, K. (2023). *Emergent compositional concept communication through mutual information in multi-agent teams*. *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2391–2393.

Karten, S., Tucker, M., Kailas, S., & Sycara, K. (2023). *Towards true lossless sparse communication in multi-agent systems*. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 7191–7197.

Karten, S., Tucker, M., Li, H., Kailas, S., Lewis, M., & Sycara, K. (2023). *Interpretable learned emergent communication for human-agent teams*. *IEEE Transactions on Cognitive and Developmental Systems*.

Kim, D., Moon, S., Hostallero, D., Kang, W. J., Lee, T., Son, K., & Yi, Y. (2019). *Learning to schedule communication in multi-agent reinforcement learning*. *International Conference on Learning Representations*.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S., & Perez, P. (2022). *Deep reinforcement learning for autonomous driving: A survey*. *IEEE Transactions on Intelligent Transportation Systems*, *23*(6), 4909–4926.

Kirby, S., Tamariz, M., Cornish, H., & Smith, K. (2015). *Compression and communication in the cultural evolution of linguistic structure*. *Cognition*, *141*, 87–102.

Kirk, R., Zhang, A., Grefenstette, E., & Rocktäschel, T. (2023). *A survey of zero-shot generalisation in deep reinforcement learning*. *Journal of Artificial Intelligence Research*, *76*, 201–264.

Kiverstein, J. (2012). *The meaning of embodiment*. *Topics in Cognitive Science*, *4*(4), 740–758.

Koller, T., Berkenkamp, F., Turchetta, M., & Krause, A. (2018). *Learning-based model predictive control for safe exploration*. *2018 IEEE Conference on Decision and Control (CDC)*.

Kottur, S., Moura, J., Lee, S., & Batra, D. (2017). *Natural language does not emerge 'naturally' in multi-agent dialog*. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Koul, A. (2019). *Ma-gym: Collection of multi-agent environments based on openai gym.*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks*. In F. Pereira, C. Burges, L. Bottou, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc.

Kurach, K., Raichuk, A., Stańczyk, P., Zając, M., Bachem, O., Espeholt, L., Riquelme, C., Vincent, D., Michalski, M., Bousquet, O., & Gelly, S. (2020). *Google research football: A novel reinforcement learning environment*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(04), 4501–4510.

Lakoff, G., & Johnson, M. (1999). *Philosophy in the flesh: The embodied mind and its challenge to western thought*. New York: Basic Books.

Lanctot, M., Schultz, J., Burch, N., Smith, M. O., Hennes, D., Anthony, T., & Perolat, J. (2023). *Population-based evaluation in repeated rock-paper-scissors as a benchmark for multiagent reinforcement learning*. *Transactions on Machine Learning Research*.

Laud, A. D. (2004). *Theory and application of reward shaping in reinforcement learning* [Doctoral dissertation, University of Illinois at Urbana-Champaign].

Lazaridou, A., & Baroni, M. (2020). *Emergent multi-agent communication in the deep learning era* [arxiv:2006.02419].

Lazaridou, A., Hermann, K. M., Tuyls, K., & Clark, S. (2018). *Emergence of linguistic communication from referential games with symbolic and pixel input*. *International Conference on Learning Representations (ICLR)*.

Lazaridou, A., Peysakhovich, A., & Baroni, M. (2017). *Multi-agent cooperation and the emergence of (natural) language*. *International Conference on Learning Representations*.

Lazaridou, A., Potapenko, A., & Tieleman, O. (2020). *Multi-agent communication meets natural language: Synergies between functional and structural language learning*. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7663–7674.

Lechner, M., yin lianhao, l., Seyde, T., Wang, T.-H. J., Xiao, W., Hasani, R., Rountree, J., & Rus, D. (2023). *Gigastep - one billion steps per second multi-agent reinforcement learning*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 155–170, Vol. 36). Curran Associates, Inc.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., & Jackel, L. (1989). *Handwritten digit recognition with a back-propagation network*. In D. Touretzky (Ed.), *Advances in neural information processing systems* (Vol. 2). Morgan-Kaufmann.

Lee, A. X., Nagabandi, A., Abbeel, P., & Levine, S. (2019). *Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model*. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 741–752, Vol. 33). Curran Associates, Inc.

Lee, J., Cho, K., & Kiela, D. (2019). *Countering language drift via visual grounding*. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 4385–4395.

Lee, J., Cho, K., Weston, J., & Kiela, D. (2018). *Emergent translation in multi-agent communication*. *International Conference on Learning Representations*.

Lehman, J., & Stanley, K. O. (2011). *Abandoning objectives: Evolution through the search for novelty alone*. *Evolutionary Computation*, *19*, 189–223.

Leike, J., Krueger, D., Everitt, T., Martic, M., Maini, V., & Legg, S. (2018). *Scalable agent alignment via reward modeling: A research direction* [arxiv:1811.07871].

Leroy, P., Morato, P. G., Pisane, J., Kolios, A., & Ernst, D. (2023). *Imp-marl: A suite of environments for large-scale infrastructure management planning via marl*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 53522–53551, Vol. 36). Curran Associates, Inc.

Levine, S., & Abbeel, P. (2014). *Learning neural network policies with guided policy search under unknown dynamics*. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, & K. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 27). Curran Associates, Inc.

Levine, S., & Koltun, V. (2013). *Guided policy search*. In S. Dasgupta & D. McAllester (Eds.), *Proceedings of the 30th international conference on machine learning* (pp. 1–9, Vol. 28). PMLR.

Lewis, D. (1969). *Convention: A philosophical study*. Harvard University Press.

Lewis, M., Yarats, D., Dauphin, Y., Parikh, D., & Batra, D. (2017). *Deal or no deal? end-to-end learning of negotiation dialogues*. *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*.

Leyton-Brown, K., & Shoham, Y. (2008). *Essentials of game theory: A concise, multidisciplinary introduction*. Springer International Publishing.

Li, G., Hammoud, H., Itani, H., Khizbullin, D., & Ghanem, B. (2023). *Camel: Communicative agents for "mind" exploration of large language model society*. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 51991–52008, Vol. 36). Curran Associates, Inc.

Li, H., Chong, Y., Stepputtis, S., Campbell, J., Hughes, D., Lewis, C., & Sycara, K. (2023). *Theory of mind for multi-agent collaboration via large language models*. *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Li, J., Kuang, K., Wang, B., Li, X., Wu, F., Xiao, J., & Chen, L. (2023). *Two heads are better than one: A simple exploration framework for efficient multi-agent reinforcement learning*. In A. Oh, T. Neumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 20038–20053, Vol. 36). Curran Associates, Inc.

Li, S., Gupta, J. K., Morales, P., Allen, R., & Kochenderfer, M. J. (2021). *Deep implicit coordination graphs for multi-agent reinforcement learning*. *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 764–772.

Li, S., Zhou, Y., Allen, R., & Kochenderfer, M. J. (2022). *Learning emergent discrete message communication for cooperative reinforcement learning*. *2022 International Conference on Robotics and Automation (ICRA)*.

Li, S., Puig, X., Paxton, C., Du, Y., Wang, C., Fan, L., Chen, T., Huang, D.-A., Akyürek, E., Anandkumar, A., Andreas, J., Mordatch, I., Torralba, A., & Zhu, Y. (2022). *Pre-trained language models for interactive decision-making* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh, Eds.). *35*, 31199–31212.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). *Continuous control with deep reinforcement learning* [arxiv:1509.02971].

Lin, J., Du, Y., Watkins, O., Hafner, D., Abbeel, P., Klein, D., & Dragan, A. (2023). *Learning to model the world with language* [arxiv:2308.01399].

Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks* [Doctoral dissertation]. Carnegie Mellon University.

Lin, T., Huh, M., Stauffer, C., Lim, S.-N., & Isola, P. (2021). *Learning to ground multi-agent communication with autoencoders*. *Advances in Neural Information Processing Systems*.

Liu, C.-C., Liao, M.-G., Chang, C.-H., & Lin, H.-M. (2022). *An analysis of children' interaction with an ai chatbot and its impact on their interest in reading*. *Computers & Education*, *189*, 104576.

Liu, I.-J., Jain, U., Yeh, R. A., & Schwing, A. (2021). *Cooperative exploration for multi-agent deep reinforcement learning*. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 6826–6836, Vol. 139). PMLR.

Liu, J., Yu, C., Gao, J., Xie, Y., Liao, Q., Wu, Y., & Wang, Y. (2024). *Llm-powered hierarchical language agent for real-time human-ai coordination*. *International Conference on Autonomous Agents and Multiagent Systems*.

Liu, S., Johns, E., & Davison, A. J. (2019). *End-to-end multi-task learning with attention*. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Liu, S., Lever, G., Merel, J., Tunyasuvunakool, S., Heess, N., & Graepel, T. (2019). *Emergent coordination through competition*. *International Conference on Learning Representations*.

Lowe, R., Foerster, J., Boureau, Y.-L., Pineau, J., & Dauphin, Y. (2019). *On the pitfalls of measuring emergent communication*. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 693–701.

Lowe, R., Gupta, A., Foerster, J., Kiela, D., & Pineau, J. (2020). *On the interaction between supervision and self-play in emergent communication*. *International Conference on Learning Representations*.

Lowe, R., Wu, Y., Tamar, A., Harb, J., Abbeel, P., & Mordatch, I. (2017). *Multi-agent actor-critic for mixed cooperative-competitive environments*. *Advances in Neural Information Processing Systems*, *30*, 1–12.

Lu, Y., Singhal, S., Strub, F., Courville, A., & Pietquin, O. (2020). *Countering language drift with seeded iterated learning*. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 6437–6447, Vol. 119). PMLR.

Lupu, A., Cui, B., Hu, H., & Foerster, J. (2021). *Trajectory diversity for zero-shot coordination*. *Proceedings of the 38th International Conference on Machine Learning*, *139*, 7204–7213.

Lupyan, G. (2012). *What do words do? toward a theory of language-augmented thought*. In B. H. Ross (Ed.), *The psychology of learning and motivation* (pp. 255–297, Vol. 57). Academic Press.

Lynch, C., & Sermanet, P. (2021). *Language conditioned imitation learning over unstructured data*. *Robotics: Science and Systems*.

Lyu, X., Baisero, A., Xiao, Y., Daley, B., & Amato, C. (2023). *On centralized critics in multi-agent reinforcement learning*. *Journal of Artificial Intelligence Research*, *77*, 295–354.

Lyu, X., Xiao, Y., Daley, B., & Amato, C. (2021). *Contrasting centralized and decentralized critics in multi-agent reinforcement learning*. *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, 844–852.

Ma, Z., Wang, R. E., Fei-Fei, L., Bernstein, M. S., & Krishna, R. (2022). *ELIGN: Expectation alignment as a multi-agent intrinsic reward*. *Advances in Neural Information Processing Systems*, *35*, 8304–8317.

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). *Robot operating system 2: Design, architecture, and uses in the wild*. *Science Robotics*, *7*(66), eabm6074.

Mahajan, A., Rashid, T., Samvelyan, M., & Whiteson, S. (2019). *MAVEN: Multi-agent variational exploration*. *Advances in Neural Information Processing Systems*, *32*.

Michalak, T. P., Aadithya, K. V., Szczepanski, P. L., Ravindran, B., & Jennings, N. R. (2014). *Efficient computation of the shapley value for game-theoretic network centrality*. *Journal Of Artificial Intelligence Research, Volume 46, pages 607-650, 2013*.

Michalski, A., Christianos, F., & Albrecht, S. V. (2023). *Smaclite: A lightweight environment for multi-agent reinforcement learning* [arxiv:2305.05566].

Michie, D., & Chambers, R. A. (1968). *Boxes: An experiment in adaptive control*. *Machine intelligence*, *2*(2), 137–152.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space* [arxiv:1301.3781].

Mikolov, T., Joulin, A., & Baroni, M. (2018). *A roadmap towards machine intelligence*. In *Lecture notes in computer science* (pp. 29–61). Springer International Publishing.

Minsky, M. (1961). *Steps toward artificial intelligence*. *Proceedings of the IRE*, *49*(1), 8–30.

Mitchell, M. (2021). *Why ai is harder than we think*. *Proceedings of the Genetic and Evolutionary Computation Conference*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). *Playing atari with deep reinforcement learning* [arxiv:1312.5602].

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). *Human-level control through deep reinforcement learning*. *Nature 518*, *518*(7540), 529–533.

Moerland, T. M., Broekens, J., Plaat, A., & Jonker, C. M. (2023). *Model-based reinforcement learning: A survey*. *Foundations and Trends® in Machine Learning*, *16*(1), 1–118.

Montague, P. R., Dolan, R. J., Friston, K. J., & Dayan, P. (2012). *Computational psychiatry*. *Trends in Cognitive Sciences*, *16*(1), 72–80.

Mordatch, I., & Abbeel, P. (2018). *Emergence of grounded compositional language in multi-agent populations*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *32*, 1495–1502.

Nachum, O., Ahn, M., Ponte, H., Gu, S. (, & Kumar, V. (2020). *Multi-agent manipulation via locomotion using hierarchical sim2real*. In L. P. Kaelbling, D. Kragic, & K. Sugiura (Eds.), *Proceedings of the conference on robot learning* (pp. 110–121, Vol. 100). PMLR.

Nair, S., Mitchell, E., Chen, K., Ichter, B., Savarese, S., & Finn, C. (2022). *Learning language-conditioned robot behavior from offline data and crowd-sourced annotation*. *Proceedings of the 5th Conference on Robot Learning*.

Narasimhan, K., Barzilay, R., & Jaakkola, T. (2018). *Grounding language for transfer in deep reinforcement learning*. *Journal of Artificial Intelligence Research*, *63*, 849–874.

Narendra, K., & Parthasarathy, K. (1990). *Identification and control of dynamical systems using neural networks*. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

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

Nevens, J., Van Eecke, P., & Beuls, K. (2020). *From continuous observations to symbolic concepts: A discrimination-based strategy for grounded concept learning*. *Frontiers in Robotics and AI*, *7*.

Ng, A. Y., Harada, D., & Russell, S. J. (1999). *Policy invariance under reward transformations: Theory and application to reward shaping*. *Proceedings of the Sixteenth International Conference on Machine Learning*, 278–287.

Nottingham, K., Ammanabrolu, P., Suhr, A., Choi, Y., Hajishirzi, H., Singh, S., & Fox, R. (2023). *Do embodied agents dream of pixelated sheep: Embodied decision making using language guided world modelling*. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, & J. Scarlett (Eds.), *Proceedings of the 40th international conference on machine learning* (pp. 26311–26325, Vol. 202). PMLR.

Nowak, M. A., & Krakauer, D. C. (1999). *The evolution of language*. *Proceedings of the National Academy of Sciences*, *96*(14), 8028–8033.

Nowé, A., Vrancx, P., & De Hauwere, Y.-M. (2012). *Game theory and multi-agent reinforcement learning*. In M. Wiering & M. van Otterlo (Eds.), *Reinforcement learning: State-of-the-art* (pp. 441–470). Springer Berlin Heidelberg.

Oh, J., Chockalingam, V., Singh, S., & Lee, H. (2016, 20–22 Jun). *Control of memory, active perception, and action in minecraft*. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning* (pp. 2790–2799, Vol. 48). PMLR.

Oh, J., Guo, X., Lee, H., Lewis, R. L., & Singh, S. (2015). *Action-conditional video prediction using deep networks in atari games*. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 28). Curran Associates, Inc.

Oliehoek, F. A., & Amato, C. (2016). *A concise introduction to decentralized pomdps*. Springer.

Omidshafiei, S., Pazis, J., Amato, C., How, J. P., & Vian, J. (2017). *Deep decentralized multi-task multi-agent reinforcement learning under partial observability*. In D. Precup & Y. W. Teh (Eds.), *Proceedings of the 34th international conference on machine learning* (pp. 2681–2690, Vol. 70). PMLR.

Oord, A. v. d., Li, Y., & Vinyals, O. (2018). *Representation learning with contrastive predictive coding* [arxiv:1807.03748].

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., d. O. Pinto, H. P., Raiman, J., Salimans, T., ... Zhang, S. (2019). *Dota 2 with large scale deep reinforcement learning* [arXiv:1912.06680].

Orr, J., & Dutta, A. (2023). *Multi-agent deep reinforcement learning for multi-robot applications: A survey*. *Sensors*, *23*(7), 3625.

Osband, I., Aslanides, J., & Cassirer, A. (2018). *Randomized prior functions for deep reinforcement learning*. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 31). Curran Associates, Inc.

Osband, I., Blundell, C., Pritzel, A., & Van Roy, B. (2016). *Deep exploration via bootstrapped dqn*. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 29). Curran Associates, Inc.

Ostrovski, G., Bellemare, M. G., van den Oord, A., & Munos, R. (2017). *Count-based exploration with neural density models*. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 2721–2730.

Oudeyer, P.-Y., & Kaplan, F. (2007). *What is intrinsic motivation? a typology of computational approaches*. *Frontiers in neurorobotics*, *1*, 6.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P. F., Leike, J., & Lowe, R. (2022). *Training language models to follow instructions with human feedback*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 27730–27744, Vol. 35). Curran Associates, Inc.

Owen, G. (2013). *Game theory*. Emerald Group Publishing.

Papadimitriou, C. H., & Tsitsiklis, J. N. (1987). *The complexity of markov decision processes*. *Mathematics of Operations Research*, *12*(3), 441–450.

Papoudakis, G., Christianos, F., & Albrecht, S. (2021). *Agent modelling under partial observability for deep reinforcement learning*. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 19210–19222, Vol. 34). Curran Associates, Inc.

Papoudakis, G., Christianos, F., Schäfer, L., & Albrecht, S. V. (2021). *Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks*. *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*.

Park, J. S., O'Brien, J., Cai, C. J., Morris, M. R., Liang, P., & Bernstein, M. S. (2023). *Generative agents: Interactive simulacra of human behavior*. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

Parker, L. E., Rus, D., & Sukhatme, G. S. (2016). *Multiple mobile robot systems*. In B. Siciliano & O. Khatib (Eds.), *Springer handbook of robotics* (pp. 1335–1384). Springer International Publishing.

Pateria, S., Subagdja, B., Tan, A.-h., & Quek, C. (2021). *Hierarchical reinforcement learning: A comprehensive survey*. *ACM Computing Surveys*, *54*(5), 1–35.

Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). *Curiosity-driven exploration by self-supervised prediction*. *Proceedings of the 34th International Conference on Machine Learning*, *70*, 2778–2787.

Peng, B., Rashid, T., Schroeder de Witt, C., Kamienny, P.-A., Torr, P., Boehmer, W., & Whiteson, S. (2021). *Facmac: Factored multi-agent centralised policy gradients*. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 12208–12221, Vol. 34). Curran Associates, Inc.

Perez, J., Léger, C., Ovando-Tellez, M., Foulon, C., Dussauld, J., Oudeyer, P.-Y., & Moulin-Frier, C. (2024). *Cultural evolution in populations of large language models* [arxiv:2403.08882].

Perfors, A., & Navarro, D. J. (2014). *Language evolution can be shaped by the structure of the world*. *Cognitive Science*, *38*(4), 775–793.

Petersen, K., Nagpal, R., & Werfel, J. (2012). *Termes: An autonomous robotic system for three-dimensional collective construction*. In *Robotics* (pp. 257–264). The MIT Press.

Pfeifer, R., & Bongard, J. (2006). *How the body shapes the way we think: A new view of intelligence*. MIT Press.

Piaget, J. (1952). *The origins of intelligence in children.* (M. Cook, Ed.). W W Norton & Co.

Pierson, H. A., & Gashler, M. S. (2017). *Deep learning in robotics: A review of recent research*. *Advanced Robotics*, *31*(16), 821–835.

Pinker, S., & Bloom, P. (1990). *Natural language and natural selection*. *Behavioral and Brain Sciences*, *13*(4), 707–727.

Pinto, L., & Gupta, A. (2016). *Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours*. *2016 IEEE International Conference on Robotics and Automation (ICRA)*.

Pislar, M., Szepesvari, D., Ostrovski, G., Borsa, D. L., & Schaul, T. (2022). *When should agents explore?* *International Conference on Learning Representations*.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., & Andrychowicz, M. (2018). *Parameter space noise for exploration*. *International Conference on Learning Representations*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., & Sutskever, I. (2021). *Learning transferable visual models from natural language supervision*. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th international conference on machine learning* (pp. 8748–8763, Vol. 139). PMLR.

Rahman, A., Carlucho, I., Höpner, N., & Albrecht, S. V. (2023). *A general learning framework for open ad hoc teamwork using graph-based policy learning*. *Journal of Machine Learning Research*, *24*(298), 1–74.

Raileanu, R., & Rocktäschel, T. (2020). *RIDE: Rewarding impact-driven exploration for procedurally-generated environments*. *9th International Conference on Learning Representations*.

Rashid, T., Farquhar, G., Peng, B., & Whiteson, S. (2020). *Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning*. *Advances in Neural Information Processing Systems 33*, 10199–10210.

Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., & Whiteson, S. (2018). *QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning*. *Proceedings of the 35th International Conference on Machine Learning*, *80*, 4295–4304.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-maron, G., Giménez, M., Sulsky, Y., Kay, J., Springenberg, J. T., Eccles, T., Bruce, J., Razavi, A., Edwards, A., Heess, N., Chen, Y., Hadsell, R., Vinyals, O., Bordbar, M., & de Freitas, N. (2022). *A generalist agent*. *Transactions on Machine Learning Research*.

Rescorla, R., & Wagner, A. (1972, January). *A theory of pavlovian conditioning: Variations in the effectiveness of reinforcement and nonreinforcement*.

Rita, M., Strub, F., Grill, J.-B., Pietquin, O., & Dupoux, E. (2022). *On the role of population heterogeneity in emergent communication*. *International Conference on Learning Representations*.

Rita, M., Tallec, C., Michel, P., Grill, J.-B., Pietquin, O., Dupoux, E., & Strub, F. (2022). *Emergent communication: Generalization and overfitting in lewis games*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 1389–1404, Vol. 35). Curran Associates, Inc.

Rizk, Y., Awad, M., & Tunstel, E. W. (2019). *Cooperative heterogeneous multi-robot systems: A survey*. *ACM Computing Surveys (CSUR)*, *52*(2), 1–31.

Robinson, J. (1951). *An iterative method of solving a game*. *The Annals of Mathematics*, *54*(2), 296.

Roesler, E., Vollmann, M., Manzey, D., & Onnasch, L. (2024). *The dynamics of human–robot trust attitude and behavior — exploring the effects of anthropomorphism and type of failure*. *Computers in Human Behavior*, *150*, 108008.

Rosenblatt, F. (1958). *The perceptron: A probabilistic model for information storage and organization in the brain.* *Psychological Review*, *65*(6), 386–408.

Rosenschein, J. S., & Zlotkin, G. (1994). *Rules of encounter*. The MIT Press.

Rudin, C. (2019). *Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead*. *Nature Machine Intelligence*, *1*(5), 206–215.

Ruis, L., Andreas, J., Baroni, M., Bouchacourt, D., & Lake, B. M. (2020). *A benchmark for systematic generalization in grounded language understanding*. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 19861–19872, Vol. 33). Curran Associates, Inc.

Rummery, G. A., & Niranjan, M. (1994). *On-line q-learning using connectionnist systems* (tech. rep.). Engineering Department, Cambridge University.

Rusu, A. A., Večerík, M., Rothörl, T., Heess, N., Pascanu, R., & Hadsell, R. (2017). *Sim-to-real robot learning from pixels with progressive nets*. In S. Levine, V. Vanhoucke, & K. Goldberg (Eds.), *Proceedings of the 1st annual conference on robot learning* (pp. 262–270, Vol. 78). PMLR.

Samek, W., Montavon, G., Lapuschkin, S., Anders, C. J., & Muller, K.-R. (2021). *Explaining deep neural networks and beyond: A review of methods and applications*. *Proceedings of the IEEE*, *109*(3), 247–278.

Samuel, A. L. (1959). *Some studies in machine learning using the game of checkers*. *IBM Journal of Research and Development*, *3*(3), 210–229.

Samvelyan, M., Rashid, T., Schroeder de Witt, C., Farquhar, G., Nardelli, N., Rudner, T. G. J., Hung, C.-M., Torr, P. H. S., Foerster, J., & Whiteson, S. (2019). *The starcraft multi-agent challenge*. *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2186–2188.

Sartoretti, G., Paivine, W., Shi, Y., Wu, Y., & Choset, H. (2019). *Distributed learning of decentralized control policies for articulated mobile robots*. *IEEE Transactions on Robotics*, *35*(5), 1109–1122.

Scassellati, B., Boccanfuso, L., Huang, C.-M., Mademtzi, M., Qin, M., Salomons, N., Ventola, P., & Shic, F. (2018). *Improving social skills in children with asd using a long-term, in-home social robot*. *Science Robotics*, *3*(21).

Schäfer, L., Slumbers, O., McAleer, S., Du, Y., Albrecht, S. V., & Mguni, D. (2023). *Ensemble value functions for efficient exploration in multi-agent reinforcement learning* [arxiv:2302.03439].

Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). *Prioritized experience replay*. *4th International Conference on Learning Representations*.

Schmidhuber, J. (1991). *A possibility for implementing curiosity and boredom in model-building neural controllers*. *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, 222–227.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., & Silver, D. (2019). *Mastering atari, go, chess and shogi by planning with a learned model* [arxiv:1911.08265].

Schroeder de Witt, C., Gupta, T., Makoviichuk, D., Makoviychuk, V., Torr, P. H. S., Sun, M., & Whiteson, S. (2020). *Is independent learning all you need in the starcraft multi-agent challenge?* [arxiv:2011.09533].

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015). *Trust region policy optimization*. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd international conference on machine learning* (pp. 1889–1897, Vol. 37). PMLR.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2016). *High-dimensional continuous control using generalized advantage estimation* (Y. Bengio & Y. LeCun, Eds.).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms* [arxiv:1707.06347].

Schultz, W., Dayan, P., & Montague, P. R. (1997). *A neural substrate of prediction and reward*. *Science*, *275*(5306), 1593–1599.

Searcy, W. A., & Nowicki, S. (2010). *The evolution of animal communication: Reliability and deception in signaling systems*. Princeton University Press.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., & Pathak, D. (2020). *Planning to explore via self-supervised world models*. *Proceedings of the 37th International Conference on Machine Learning*, *PMLR 119*, 8583–8592.

Shapley, L. S. (1953). *A value for n-person games*. *Contributions to the Theory of Games*, *2*(28), 307–317.

Shih, A., Sawhney, A., Kondic, J., Ermon, S., & Sadigh, D. (2021). *On the critical role of conventions in adaptive human-ai collaboration*. *International Conference on Learning Representations*.

Shoham, Y., Powers, R., & Grenager, T. (2007). *If multi-agent learning is the answer, what is the question?* [Foundations of Multi-Agent Learning]. *Artificial Intelligence*, *171*(7), 365–377.

Shridhar, M., Manuelli, L., & Fox, D. (2021). *Cliport: What and where pathways for robotic manipulation*. In A. Faust, D. Hsu, & G. Neumann (Eds.), *Proceedings of the 5th conference on robot learning* (pp. 894–906, Vol. 164). PMLR.

Shu, T., Xiong, C., & Socher, R. (2018). *Hierarchical and interpretable skill acquisition in multi-task reinforcement learning*. *International Conference on Learning Representations*.

Shyam, P., Jaśkowski, W., & Gomez, F. (2019). *Model-based active exploration*. *Proceedings of the 36th International Conference on Machine Learning*, *PMLR 97*, 5779–5788.

Sigurd, B., Eeg-Olofsson, M., & Van Weijer, J. (2004). *Word length, sentence length and frequency – zipf revisited*. *Studia Linguistica*, *58*(1), 37–52.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T. P., Leach, M., Kavukcuoglu, K., Graepel, T., & Hassabis, D. (2016). *Mastering the game of go with deep neural networks and tree search*. *Nature 529*, *529*(7587), 484–489.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., & Riedmiller, M. (2014). *Deterministic policy gradient algorithms*. In E. P. Xing & T. Jebara (Eds.), *Proceedings of the 31st international conference on machine learning* (pp. 387–395, Vol. 32). PMLR.

Silver, D., Singh, S., Precup, D., & Sutton, R. S. (2021). *Reward is enough*. *Artificial Intelligence*, *299*, 103535.

Simonyan, K., & Zisserman, A. (2015). *Very deep convolutional networks for large-scale image recognition*. *3rd International Conference on Learning Representations (ICLR 2015)*, 1–14.

Singh, A., Jain, T., & Sukhbaatar, S. (2019). *Learning when to communicate at scale in multiagent cooperative and competitive tasks*. *International Conference on Learning Representations*.

Singh, S., Mahjoub, O., de Kock, R., Khlifi, W., Vall, A., Tessera, K.-a., & Pretorius, A. (2023). *How much can change in a year? revisiting evaluation in multi-agent reinforcement learning* [arxiv:2312.08463].

Siskind, J. M. (1992). *Naive physics, event perception, lexical semantics, and language acquisition* [Doctoral dissertation, Massachusetts Institute of Technology].

Smith, J., & Harper, D. (2003). *Animal signals*. OUP Oxford.

Smith, K., Brighton, H., & Kirby, S. (2003). *Complex systems in language evolution: The cultural emergence of compositional structure*. *Advances in Complex Systems*, *06*(04), 537–558.

Smith, K., Kirby, S., & Brighton, H. (2003). *Iterated learning: A framework for the emergence of language*. *Artificial Life*, *9*(4), 371–386.

Son, K., Kim, D., Kang, W. J., Hostallero, D. E., & Yi, Y. (2019). *Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning*. *Proceedings of the 36th International Conference on Machine Learning*, *PMLR 97*, 5887–5896.

Steels, L. (1995). *A Self-Organizing Spatial Vocabulary*. *Artificial Life*, *2*(3), 319–332.

Steels, L., & Kaplan, F. (2000). *Aibo's first words: The social learning of language and meaning*. *Evolution of Communication*, *4*(1), 3–32.

Steels, L. L. (2015). *The Talking Heads experiment: Origins of words and meanings*. Language Science Press.

Stiennon, N., Ouyang, L., Wu, J., Ziegler, D., Lowe, R., Voss, C., Radford, A., Amodei, D., & Christiano, P. F. (2020). *Learning to summarize with human feedback*. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 3008–3021, Vol. 33). Curran Associates, Inc.

Stone, P., Kaminka, G., Kraus, S., & Rosenschein, J. (2010). *Ad hoc autonomous agent teams: Collaboration without pre-coordination*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *24*(1), 1504–1509.

Stone, P., & Veloso, M. (2000). *Multiagent systems: A survey from a machine learning perspective*. *Autonomous Robots*, *8*(3), 345–383.

Strouse, D., McKee, K., Botvinick, M., Hughes, E., & Everett, R. (2021). *Collaborating with humans without human data*. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 14502–14515, Vol. 34). Curran Associates, Inc.

Sukhbaatar, S., Szlam, A., & Fergus, R. (2016). *Learning multiagent communication with backpropagation*. *Advances in Neural Information Processing Systems*, 2244–2252.

Sun, W.-F., Lee, C.-K., See, S., & Lee, C.-Y. (2023). *A unified framework for factorizing distributional value functions for multi-agent reinforcement learning*. *Journal of Machine Learning Research*, *24*(220), 1–32.

Sünderhauf, N., Brock, O., Scheirer, W., Hadsell, R., Fox, D., Leitner, J., Upcroft, B., Abbeel, P., Burgard, W., Milford, M., & Corke, P. (2018). *The limits and potentials of deep learning for robotics*. *The International Journal of Robotics Research*, *37*(4-5), 405–420.

Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., & Graepel, T. (2018). *Value-decomposition networks for cooperative multi-agent learning based on team reward*. *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, 2085–2087.

Sutton, R. S. (1988). *Learning to predict by the methods of temporal differences*. *Machine Learning*, *3*(1), 9–44.

Sutton, R. S. (1991). *Dyna, an integrated architecture for learning, planning, and reacting*. *ACM SIGART Bulletin*, *2*(4), 160–163.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction, second edition*. The MIT Press.

Sutton, R. S., Szepesvári, C., Geramifard, A., & Bowling, M. (2008). *Dyna-style planning with linear function approximation and prioritized sweeping*. *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, 528–536.

Sutton, R. S. (1984). *Temporal credit assignment in reinforcement learning* [Doctoral dissertation]. University of Massachusetts Amherst.

Tampuu, A., Matiisen, T., Kodelja, D., Kuzovkin, I., Korjus, K., Aru, J., Aru, J., & Vicente, R. (2017). *Multiagent cooperation and competition with deep reinforcement learning*. *PLOS ONE*, *12*(4), 1–15.

Tan, M. (1993). *Multi-agent reinforcement learning: Independent versus cooperative agents*. *Proceedings of the Tenth International Conference on Machine Learning*, 330–337.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., Lillicrap, T., & Riedmiller, M. (2018). *Deepmind control suite* [arxiv:1801.00690].

Tesauro, G. (1994). *Td-gammon, a self-teaching backgammon program, achieves master-level play*. *Neural Computation*, *6*(2), 215–219.

Tesauro, G., & Galperin, G. (1996). *On-line policy improvement using monte-carlo search*. In M. Mozer, M. Jordan, & T. Petsche (Eds.), *Advances in neural information processing systems* (Vol. 9). MIT Press.

Thorndike, E. L. (1911). *Animal intelligence; experimental studies*. The Macmillan Company.

Tian, Y., Krishnan, D., & Isola, P. (2020). *Contrastive multiview coding*. In A. Vedaldi, H. Bischof, T. Brox, & J.-M. Frahm (Eds.), *Computer vision – eccv 2020* (pp. 776–794). Springer International Publishing.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). *Domain randomization for transferring deep neural networks from simulation to the real world*. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 23–30.

Todorov, E., Erez, T., & Tassa, Y. (2012). *Mujoco: A physics engine for model-based control*. *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033.

Tomasello, M. (2009). *The cultural origins of human cognition*. Harvard university press.

Tseng, S.-H., Chao, Y., Lin, C., & Fu, L.-C. (2016). *Service robots: System design for tracking people through data fusion and initiating interaction with the human group by inferring social situations*. *Robotics and Autonomous Systems*, *83*, 188–202.

Tucker, M., Li, H., Agrawal, S., Hughes, D., Sycara, K. P., Lewis, M., & Shah, J. (2021). *Emergent discrete communication in semantic spaces*. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems*.

Uchendu, I., Xiao, T., Lu, Y., Zhu, B., Yan, M., Simon, J., Bennice, M., Fu, C., Ma, C., Jiao, J., Levine, S., & Hausman, K. (2023, 23–29 Jul). *Jump-start reinforcement learning*. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, & J. Scarlett (Eds.), *Proceedings of the 40th international conference on machine learning* (pp. 34556–34583, Vol. 202). PMLR.

Van Eecke, P., Beuls, K., Botoko Ekila, J., & Rădulescu, R. (2022). *Language games meet multi-agent reinforcement learning: A case study for the naming game*. *Journal of Language Evolution*, *7*(2), 213–223.

Vandenhende, S., Georgoulis, S., Van Gansbeke, W., Proesmans, M., Dai, D., & Van Gool, L. (2021). *Multi-task learning for dense prediction tasks: A survey*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

van Hasselt, H. (2010). *Double q-learning*. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems* (Vol. 23). Curran Associates, Inc.

van Hasselt, H., Guez, A., & Silver, D. (2016). *Deep reinforcement learning with double q-learning*. In D. Schuurmans & M. P. Wellman (Eds.), *Proceedings of the thirtieth AAAI conference on artificial intelligence, february 12-17, 2016, phoenix, arizona, USA* (pp. 2094–2100). AAAI Press.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). *Attention is all you need*. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 30). Curran Associates, Inc.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T., & Riedmiller, M. (2017). *Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards* [arxiv:1707.08817].

Villalobos, P., Sevilla, J., Besiroglu, T., Heim, L., Ho, A., & Hobbhahn, M. (2022). *Machine learning model sizes and the parameter gap* [arxiv:2207.02852].

Vinitsky, E., Lichtlé, N., Yang, X., Amos, B., & Foerster, J. (2022). *Nocturne: A scalable driving benchmark for bringing multi-agent learning one step closer to the real world*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 3962–3974, Vol. 35). Curran Associates, Inc.

Vogt, P. (2002). *The physical symbol grounding problem*. *Cognitive Systems Research*, *3*, 429–457.

Vogt, P. (2005). *The emergence of compositional structures in perceptually grounded language games*. *Artificial Intelligence*, *167*(1–2), 206–242.

Vygotsky, L. S. (1934). *Thought and language*. MIT Press.

Walter, W. G. (1950). *An imitation of life*. *Scientific American*, *182*(5), 42–45.

Wang, J., Ren, Z., Han, B., Ye, J., & Zhang, C. (2021). *Towards understanding cooperative multi-agent q-learning with value factorization*. In M. Ranzato, A.

Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems* (pp. 29142–29155, Vol. 34). Curran Associates, Inc.

Wang, J., Ren, Z., Liu, T., Yu, Y., & Zhang, C. (2021). *Qplex: Duplex dueling multi-agent q-learning*. *9th International Conference on Learning Representations*.

Wang, J., Zhang, Y., Gu, Y., & Kim, T.-K. (2022). *Shaq: Incorporating shapley value theory into multi-agent q-learning*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 5941–5954, Vol. 35). Curran Associates, Inc.

Wang, J., Zhang, Y., Kim, T.-K., & Gu, Y. (2020). *Shapley q-value: A local reward approach to solve global reward games*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(05), 7285–7292.

Wang, R. E., Everett, M., & How, J. P. (2020). *R-maddpg for partially observable environments and limited communication*. *Reinforcement Learning for Real Life (RL4RealLife) Workshop in the 36th International Conference on Machine Learning, Long Beach, California, USA, 2019*.

Wang, R., He, X., Yu, R., Qiu, W., An, B., & Rabinovich, Z. (2020). *Learning efficient multi-agent communication: An information bottleneck approach*. In H. D. III & A. Singh (Eds.), *Proceedings of the 37th international conference on machine learning* (pp. 9908–9918, Vol. 119). PMLR.

Wang, S. I., Liang, P., & Manning, C. D. (2016). *Learning language games through interaction*. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Wang, T., Wang, J., Wu, Y., & Zhang, C. (2020). *Influence-based multi-agent exploration*. *8th International Conference on Learning Representations*.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). *Dueling network architectures for deep reinforcement learning*. In M. Balcan & K. Q. Weinberger (Eds.), *Proceedings of the 33rd international conference on machine learning, ICML 2016, new york city, ny, usa, june 19-24, 2016* (pp. 1995–2003, Vol. 48).

Watkins, C. J. C. H. (1989). *Learning from delayed rewards* [Doctoral dissertation].

Wei, E., & Luke, S. (2016). *Lenient learning in independent-learner stochastic cooperative games*. *The Journal of Machine Learning Research*, *17*, 2914–2955.

Wei, E., Wicke, D., Freelan, D., & Luke, S. (2018). *Multiagent soft q-learning* [arXiv:1804.09817].

Weir, N., Yuan, X., Côté, M.-A., Hausknecht, M., Laroche, R., Momennejad, I., Van Seijen, H., & Van Durme, B. (2023). *One-shot learning from a demonstration with hierarchical latent language*. *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2388–2390.

Wiegand, R. P. (2003). *An analysis of cooperative coevolutionary algorithms* [Doctoral dissertation, George Mason University].

Wierstra, D., Foerster, A., Peters, J., & Schmidhuber, J. (2007). *Solving deep memory pomdps with recurrent policy gradients*. In J. M. de Sá, L. A. Alexandre, W. Duch, & D. Mandic (Eds.), *Artificial neural networks – icann 2007* (pp. 697–706). Springer Berlin Heidelberg.

Williams, R. J. (1992). *Simple statistical gradient-following algorithms for connectionist reinforcement learning*. *Machine Learning*, *8*(3–4), 229–256.

Wolpert, D. H., & Tumer, K. (1999). *An introduction to collective intelligence* (tech. rep.) (NASA-ARC-IC-99-63). NASA.

Wolpert, D. H., & Tumer, K. (2002). *Optimal payoff functions for members of collectives*. In F. Schweitzer (Ed.), *Modeling complexity in economic and social systems* (pp. 355–369). World Scientific Publishing Co. Pte. Ltd.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John wiley & sons.

Wooldridge, M., & Jennings, N. R. (1995). *Intelligent agents: Theory and practice*. *The Knowledge Engineering Review*, *10*(2), 115–152.

Xie, A., Losey, D. P., Tolsma, R., Finn, C., & Sadigh, D. (2021). *Learning latent representations to influence multi-agent interaction*. In J. Kober, F. Ramos, & C. Tomlin (Eds.), *Proceedings of the 2020 conference on robot learning* (pp. 575–588, Vol. 155). PMLR.

Xu, Z., Zhang, B., li dapeng, d., Zhou, G., Zhang, Z., & Fan, G. (2023). *Dual self-awareness value decomposition framework without individual global max for cooperative marl*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 73898–73918, Vol. 36). Curran Associates, Inc.

Yan, X., Guo, J., Lou, X., Wang, J., Zhang, H., & Du, Y. (2023). *An efficient end-to-end training approach for zero-shot human-ai coordination*. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, & S. Levine (Eds.), *Advances in neural information processing systems* (pp. 2636–2658, Vol. 36). Curran Associates, Inc.

Yang, Y., Hao, J., Liao, B., Shao, K., Chen, G., Liu, W., & Tang, H. (2020). *Qatten: A general framework for cooperative multiagent reinforcement learning* [arxiv:2002.03939].

Yu, C., Gao, J., Liu, W., Xu, B., Tang, H., Yang, J., Wang, Y., & Wu, Y. (2023). *Learning zero-shot cooperation with humans, assuming humans are biased*. *The 11th International Conference on Learning Representations*.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021a). *Benchmarking multi-agent deep reinforcement learning algorithms*.

Yu, C., Velu, A., Vinitsky, E., Wang, Y., Bayen, A., & Wu, Y. (2021b). *The surprising effectiveness of ppo in cooperative, multi-agent games* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh, Eds.). *35*, 24611–24624.

Zeng, A., Attarian, M., Ichter, B., Choromanski, K., Wong, A., Welker, S., Tombari, F., Purohit, A., Ryoo, M., Sindhwani, V., Lee, J., Vanhoucke, V., & Florence, P. (2022). *Socratic models: Composing zero-shot multimodal reasoning with language*. *11th International Conference on Learning Representations*.

Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., & Gan, C. (2024). *Building cooperative embodied agents modularly with large language models*. *International Conference on Learning Representations*.

Zhang, S. Q., Zhang, Q., & Lin, J. (2019). *Efficient communication in multi-agent reinforcement learning via variance based control*. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information processing systems* (Vol. 32). Curran Associates, Inc.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., & Tian, Y. (2021). *NovelD: A simple yet effective exploration criterion*. *Advances in Neural Information Processing Systems*, *34*, 25217–25230.

Zhao, R., Song, J., Yuan, Y., Hu, H., Gao, Y., Wu, Y., Sun, Z., & Yang, W. (2023). *Maximum entropy population-based training for zero-shot human-ai coordination*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *37*(5), 6145–6153.

Zheng, L., Chen, J., Wang, J., He, J., Hu, Y., Chen, Y., Fan, C., Gao, Y., & Zhang, C. (2021). *Episodic multi-agent reinforcement learning with curiosity-driven exploration*. In A. Beygelzimer, Y. Dauphin, P. Liang, & J. W. Vaughan (Eds.), *Advances in neural information processing systems*.

Zhou, H., Lan, T., & Aggarwal, V. (2022). *Pac: Assisted value factorization with counterfactual predictions in multi-agent reinforcement learning*. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, & A. Oh (Eds.), *Advances in neural information processing systems* (pp. 15757–15769, Vol. 35). Curran Associates, Inc.

Zhou, M., Liu, Z., Sui, P., Li, Y., & Chung, Y. Y. (2020). *Learning implicit credit assignment for cooperative multi-agent reinforcement learning*. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems* (pp. 11853–11864, Vol. 33). Curran Associates, Inc.

Zhu, C., Dastani, M., & Wang, S. (2024). *A survey of multi-agent deep reinforcement learning with communication*. *Autonomous Agents and Multi-Agent Systems*, *38*(1).

Zhu, F., & Simmons, R. (2024). *Bootstrapping cognitive agents with a large language model*. *Proceedings of the AAAI Conference on Artificial Intelligence*, *38*(1), 655–663.

Ziemke, T. (2003). *What's that thing called embodiment? Proceedings of the 25th Annual Cognitive Science Society.*

Złotowski, J., Proudfoot, D., Yogeeswaran, K., & Bartneck, C. (2014). *Anthropomorphism: Opportunities and challenges in human–robot interaction*. *International Journal of Social Robotics*, *7*(3), 347–360.

Zuidema, W., & de Boer, B. (2018). *The evolution of combinatorial structure in language*. *Current Opinion in Behavioral Sciences*, *21*, 138–144.